University of Southern Queensland

Faculty of Engineering & Surveying

Echo Cancellation in VoIP using Digital Adaptive Filters

A dissertation submitted by

S. Kmita

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Electrical and Electronic)

Submitted: October, 2011

Abstract

VoIP calls frequently suffer from echoes which degrade the quality of voice transmissions. Traditional methods of using adaptive filters to cancel line echoes over the public switched telephone network are not as effective when applied to VoIP channels. This is because VoIP echo paths are generally longer due to longer network delays and non-stationary due to dynamic de-jitter buffering. Also, non-linearities introduced by dropped packets and lossy signal compression algorithms can reduce the performance of the adaptive filters. This project researched the theory of digital adaptive filter algorithms and their application to the echo cancellation problem in VoIP networks. A VoIP, adaptive echo cancellation (AEC) simulation was designed and then implemented in MATLAB. The simulation modeled an echo path which incorporated VoIP channel characteristics and room acoustic effects. The simulation was used to test the echo cancelling effectiveness of three different adaptive algorithm schemes: a normalised least mean squares (NLMS) filter, a NLMS filter in Dual-H configuration and a recursive least squares (RLS) filter. Echo cancellation performance was primarily determined by measuring the loudness of echoes before and after they enter the system (represented by an ERLE value). The Telecommunication Standardization Sector of the International Telecommunications Union (ITU-T) G.131 echo objection rate gives the recommended echo attenuation levels required by an echo canceller. The Dual-H NLMS AEC system designed in this project achieved an ERLE of approximately 35 dB during simulations in the VoIP environment which is above the ITU-T recommended level. This showed that satisfactory echo cancellation performance in a VoIP environment could be achieved by the use of this relatively simple AEC system.

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

S. Kmita

0050093012

Signature

Date

Acknowledgments

I would like to thank Dr John Leis for supervising this project and his willingness to share his vast technical knowledge.

Thanks also to my wife Chantelle and kids Hayden, Sophia and Mollie for their endless patience during the course of my studies.

S. Kmita

University of Southern Queensland

October 2011

Contents

List of Figures

List of Tables

Acronyms

ADC	Analogue to Digital Converter
AEC	Adaptive Echo Cancellation
CODEC	COder/DECoder
DAC	Digital to Analogue Converter
ERL	Echo Return Loss
ERLE	Echo Return Loss Enhancement
IP	Internet Protocol
ITU-T	Telecommunication Standardization Sector of the International Telecommunications Union
LMS	Least Mean Squares
NLMS	Normalised Least Mean Squares
PCM	Pulse Code Modulation
PSTN	Public Switched Telephone Network
RLS	Recursive Least Squares
SIP	Session Initiation Protocol
SNR	Signal to Noise Ratio
VoIP	Voice over Internet Protocol

Chapter 1

Introduction

Voice over Internet Protocol (VoIP) applications enable voice communications to be conducted over Internet Protocol (IP) networks such as the internet. They provide several advantages over traditional public switched telephone network (PSTN) systems such as the ability to consolidate separate data and voice networks and avoiding call fees from telecommunication companies. They are becoming a popular solution for large organisations and individual users alike to take advantage of the increasing speed and capacity of new data networks such as the National Broadband Network (NBN).

However, the voice quality of VoIP calls is generally worse than PSTN calls due to IP networks inherently suffering from variable network delay and packet loss. These can be mitigated by buffering but this leads to increased signal delays (CISCO 2006). If there is coupling between the microphone and speaker of the telephone at either end (due to using a 'hands-free' phone for example) then these large delays can result in annoying feedback echoes being heard on the line.

Reducing these acoustic echoes is generally attempted using echo cancelling systems based on adaptive filters, a method that has successfully been applied to the removal of electrical echoes in the PSTN (Haykin 1995, p. 56). The simplest adaptive filter algorithm used for this purpose is the ubiquitous least mean squares (LMS) method invented by Widrow and Hoff in 1959 (Widrow & Hoff 1960). However, certain inherent characteristics of VoIP channels, such as high latencies and non-linearities, make LMS a less suitable choice for cancelling acoustic echoes in VoIP. Other, more complicated methods such as the recursive least square (RLS) and affine projection (AP) algorithms have been shown to perform better at the expense of increased computational cost and implementation complexity (Huang & Goubran 2000).

Besides changing adaptive filter algorithms completely, there are a number of refinements (such as the normalised version of LMS (NLMS)) that could be made to the algorithm in an attempt to boost its echo cancellation performance. Also, extra functional blocks may be added to the system (e.g. Double-Talk Detector (DTD), Dual-H Filter, Non-Linear Processor). However, with each added refinement comes a corresponding increase in system complexity and computational cost. Therefore the designer of the system must weigh up any performance increase with these higher costs.

With this in mind, the initial adaptive echo cancellation (AEC) system developed in this project was based on a basic adaptive filter algorithm (NLMS) and a simple system design. This was implemented in MATLAB and its echo cancellation performance was measured by simulation. The simulations used an echo path model which incorporated both the VoIP channel and he acoustic effects of a typical small room. The performance of this system was used as the baseline to compare with another, more complicated RLS-based system, to assess whether or not the extra computational effort required by this new system was worth any performance increase.

Since the performance of a particular AEC system is greatly influenced by its specific operating conditions, various testing scenarios were used in the simulations to measure the effects of variables such as ambient noise levels, acoustic environment, network delay and packet loss. The real-world suitability of each AEC system was then able to be determined by seeing whether or not it could attenuate echoes below the maximum acceptable loudness level recommended by The Telecommunication Standardization Sector of the International Telecommunications Union (ITU-T) (ITU-T 2003).

1.1 Project Aim

To design and implement a digital adaptive filter system for the purpose of cancelling echoes in VoIP transmissions.

1.2 Project Objectives

1. Research the background information on echo cancellation using digital adaptive filters and the characteristics of VoIP channels.
2. Simulate an IP channel with variable parameters such as bandwidth, latency, jitter and packet-loss.
3. Design and implement experiments to compare the effectiveness of various filter algorithms in cancelling echoes in a VoIP setting.
4. Analyse the results from these experiments to determine the most effective algorithms.

1.3 Overview of the Dissertation

This dissertation is organized as follows:

Chapter 2 presents a background of the nature of telephony echoes and introduces a basic echo cancelling system. A brief description of the important digital signal processing concepts used in the derivation of digital adaptive filters is then presented.

Chapter 3 introduces a linear optimum filter called the Wiener Filter which is then used as a basis to derive the Adaptive Filters used in this project.

Chapter 4 describes the general characteristics of VoIP channels and their likely implications to echo cancellation.

Chapter 5 outlines the methodology used to design and implement the computer simulations used to test each AEC system n a VoIP environment.

Chapter 6 presents and discusses the results of the echo cancellation computer simulations.

Chapter 7 presents the project conclusion and suggests areas of possible future research.

Chapter 2

Echo Cancellation Theory

2.1 Chapter Overview

Understanding echo cancellation requires a knowledge of the source of echoes. This chapter begins by explaining the nature of telephony echoes and then introduces the AEC system used in this project. The relationship between echo loudness and network delay on echo perception is then explained with reference to the ITU-T echo objection rate curve, which is used to determine the level of echo attenuation required by an AEC system. Lastly, the background digital signal processing concepts required to derive adaptive filters are presented.

2.2 Echoes in Telephony Systems

The two most prevalent sources of echoes in telephony networks are:

- Electrical echoes
- Acoustic echoes

2.2.1 Electrical Echoes

Electrical echoes are caused by signal reflections due to circuit imperfections and impedance mismatches (Sondhi & Berkley 1980). A common source of impedance mismatch in the PSTN is at the connection of the 2-wire local loop (used to connect the subscriber to the local exchange) and the 4-wire circuit (used for long haul transmission). This connection is made by an electrical device called a hybrid transformer. Impedance mismatch between the two circuits causes some of the signal energy to be reflected back down the 2-wire circuit to the local subscriber. These unwanted reflections are called hybrid echoes.

2.2.2 Acoustic Echoes

Suppose a far-end caller is speaking to a near-end receiver. The far-end voice transmission is played though the near-end telephone speaker and some of the sound emitted is picked up by the near-end microphone and retransmitted to the far-end. The far-end signal may directly pass from speaker to microphone (if they are poorly isolated) or sound waves may be reflected a number of times around the room first. Since each echo path has variable length and since every reflection attenuates the sound wave, the retransmitted signal will contain decreasing, time-delayed images of the original far-end signal. This unwanted retransmitted signal is termed *acoustic echo*.

Figure 2.1 shows the block diagram of a simplified speaker-phone telephone call. Note that to aid clarity, only the discrete-time sampled speech signals are shown[1]. A far-end input voice signal, $x(n)$ is generated by the far-end talker and picked up by the far-end microphone. This signal travels to the near-end via an IP network where it is emitted through the speaker. $\mathbf{H}$ represents the unknown acoustic impulse response of the near-end room and its output, $d(n)$, is the acoustic echo of the far-end signal picked up by the near-end microphone. $s(n)$ represents the near-end speech picked up by the microphone. It can be seen that the signal that travels back to the far-end is

[1]In reality, the analogue signal picked up by the microphones would be digitised using an ADC. Also, the digital signals would be converted back to analogue by a DAC before being emitted from the speakers. See Chapter 4 below for more detail of how this happens in a VoIP application.

a combination of the near-end speech as well as the unwanted acoustic echoes. The elimination of these echoes is the subject of the next section.

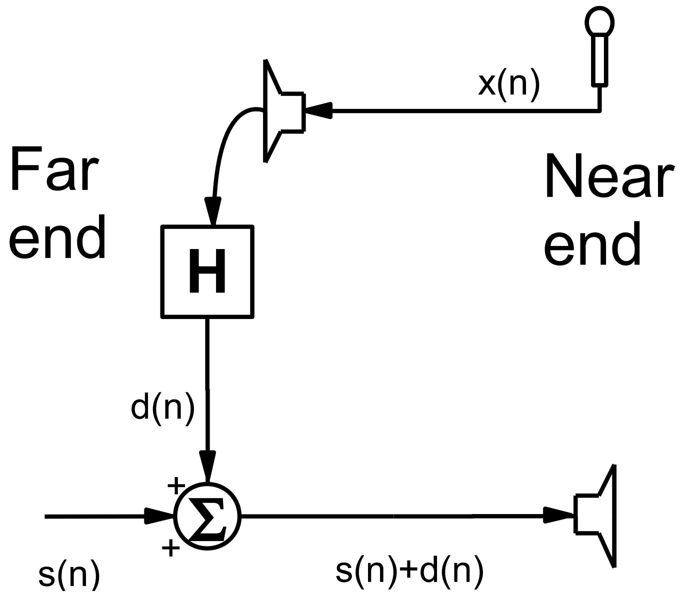

Figure 2.1: Acoustic echo diagram.

2.3 Adaptive Echo Cancellation (AEC)

Sondhi (1967) was the first to publish the idea of applying adaptive filters to echo cancellation, although in his paper the author acknowledges J. L. Kelly Jr of Bell Laboratories as having the original idea. Figure 2.2 shows the simple AEC system used for echo cancellation in this project. **W** is a special type of filter called an Adaptive Filter which is able to adjust its transfer function to mimic **H** (see Chapter 3 below for details of its operation). The output of **W** is $y(n)$ which is then subtracted from $s(n) + d(n)$ to give $e(n)$. It can be seen that if $y(n)$ is a good approximation of $d(n)$ then $e(n)$ will be a good approximation of $s(n)$ - the near-end speech less any acoustic echoes as required.

Although the above explanation of AEC refers only to acoustic echoes it will be shown in Chapter 4 that the system is also capable of cancelling any electrical echoes that occur to the left of the adaptive filter input in Figure 2.2. This is a consideration when deciding on the placement of the AEC system within the circuit.

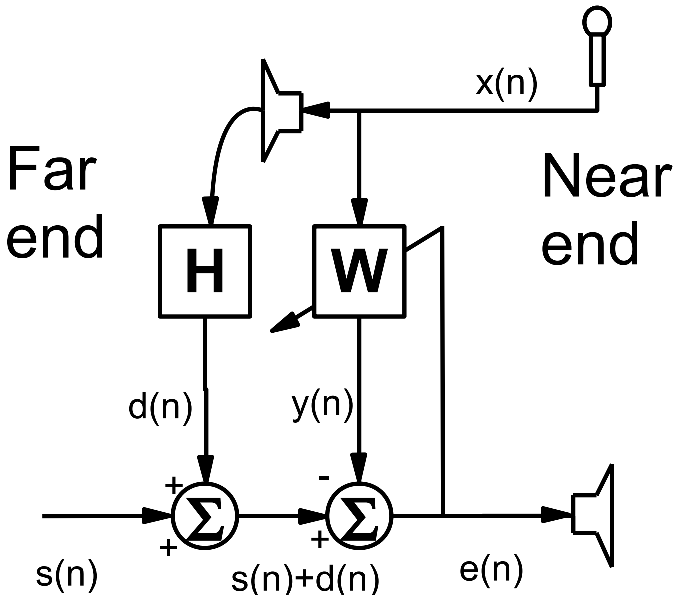

Figure 2.2: A simple AEC system.

2.4 Echo Measurement

Echo Return Loss (ERL) is a quantitative measure of echo loudness given by:

$$ERL = \frac{Power\ Level\ of\ Original\ Signal}{Power\ Level\ of\ Echo\ Signal} \tag{2.1}$$

Echo Return Loss Enhancement (ERLE) is the measurement of the echo attenuation by an adaptive filter given by:

$$ERLE = \frac{Power\ Level\ of\ Echo\ Signal}{Power\ Level\ of\ Residual\ Echo\ Signal} \tag{2.2}$$

For example, the ERL of the AEC shown in Figure 2.2 would be:

$$ERL(dB) = 10log\frac{\boldsymbol{x}^2(n)}{\boldsymbol{d}^2(n)} \tag{2.3}$$

and its ERLE would be:

$$ERLE(dB) = 10log\frac{\boldsymbol{d}^2(n)}{\boldsymbol{e}^2(n)} \tag{2.4}$$

Note that the ERL and ERLE formulae require that the signals, $x(n)$, $e(n)$ and $d(n)$ are able to be measured. It can be seen that $x(n)$ and $e(n)$ can be measured by taking the near-end received and sent signals respectively. However, since $d(n)$ represents the echo signal before it enters the microphone it is not observable. The signal being picked up by the microphone *is* observable and is made up of the echo and any near-end speech $(d(n)+s(n))$. When there is no near-end speech $s(n) = 0$ and so this signal can be used to calculate ERL and ERLE but these measurements will only be accurate under these conditions. This was an important consideration when implementing these calculations in the AEC computer simulations.

2.5 Echo Perception

The ITU-T gives the relationship between echo delay, loudness and listener annoyance in terms of Talker Echo Loudness Rating (TELR)[2]. The TELR can be converted into the ERL measurement (Ditech Networks 2011) and the resulting plot is shown in Figure 2.3 below. It shows that echoes become more noticeable (and annoying) as delay and loudness increase. This plot shows the minimum level of echo attenuation that an echo cancelling system will need to be able to achieve to be considered to be adequate.

2.6 Digital Signal Processing (DSP)

2.6.1 Discrete-Time Signals

In digital voice communications, continuous analogue signals are digitised before they are transmitted. Suppose a continuous-time signal, $x(t)$ is sampled every T_s time

[2]TELR is a measure of the echo loss perceived by a listener (ITU-T 2003).

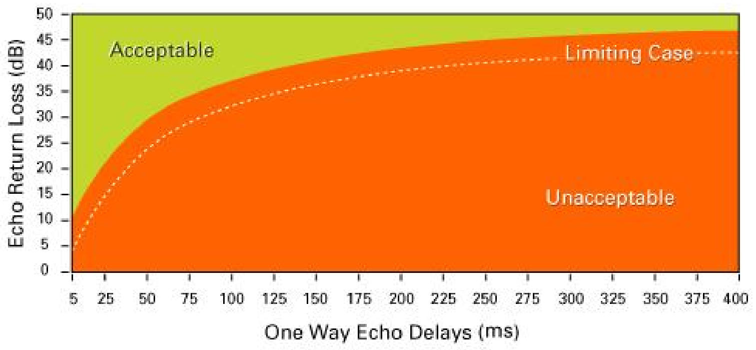

Figure 2.3: ITU-T G.131 echo objection rate (reproduced from `http://www.ditechnetworks.com/learningCenter/echoBasics.html`).

periods (the so called *sample period*) then the digitised signal may be represented by the sequence $\{x(0),\ x(T_s),\ x(2T_s),\ \ldots,\ x(nT_s),\ \ldots\}$.

To facilitate computer processing, the section of the signal we are currently processing inside a filter may be represented by the vector of length N:

$$\boldsymbol{x}(n) = [x(n)\ x(n-1)\ \ldots\ x(n-N+1)]^T \tag{2.5}$$

Throughout the rest of this document the boldfaced $\boldsymbol{x}(n)$ shall refer to such a vector and the plain typed $x(n)$ shall refer to the nth sample.

2.6.2 Expectation

The expectation value of a random variable $x(n)$ at time n is equal to the mean value of an ensemble of realisations of the variable (Poularikas & Ramadan 2006, p. 22) so that:

$$\hat{E}\{x(n)\} = \lim_{N\to\infty} \left\{ \frac{1}{N} \sum_{n=1}^{N} x(n) \right\} \tag{2.6}$$

where $\hat{E}\{.\}$ is called the expectation operator. Since $x(n)$ is a real-time speech signal and we only have access to one realisation, it is assumed that the process is ergodic[3].

[3]A stochastic process is said to be *ergodic* if its statistical properties can be calculated from one sufficiently long realisation.

Under this assumption an estimate of the expectation, $E\{.\}$ can be calculated from the sample mean of the last N samples according to:

$$E\{x(n)\} = \frac{1}{N}\sum_{n=0}^{N-1} x(n) \tag{2.7}$$

and this estimate shall be used for the remainder of this thesis.

2.6.3 Correlation

The cross correlation of two sampled signals $x(n)$ and $y(n)$ is given by:

$$\hat{R}_{xy}(k) = \lim_{N\to\infty}\left\{\frac{1}{N}\sum_{i=1}^{N} x_i(n)y_i(n-k)\right\} \tag{2.8}$$

where k is either a positive or negative lag value (Leis 2002, p. 124). Maintaining our assumption that all signals are ergodic yields:

$$R_{xy}(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)y(n-k) \tag{2.9}$$

The autocorrelation is simply the cross correlation of a signal with a lagged version of itself:

$$R_{xx}(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)x(n-k) \tag{2.10}$$

The relationship between the expectation operator and correlation function can be seen if we take the expectation of the product of $x(n)$ and $y(n-k)$:

$$\begin{aligned} E\{x(n)y(n-k)\} &= \frac{1}{N}\sum_{n=0}^{N-1} x(n)y(n-k) \\ &= R_{xy}(k) \end{aligned} \tag{2.11}$$

We now define the correlation matrix of $x(n)$, $\boldsymbol{R}_x$ as:

$$\begin{aligned}\boldsymbol{R}_x &= E\{x(n)x(n)^T\} \\ &= \begin{bmatrix} R_{xx}(0) & R_{xx}(1) & \cdots & R_{xx}(N-1) \\ R_{xx}(-1) & R_{xx}(0) & \cdots & R_{xx}(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ R_{xx}(-N+1) & R_{xx}(-N+2) & \cdots & R_{xx}(0) \end{bmatrix}\end{aligned} \tag{2.12}$$

Finally, the cross correlation vector, $\boldsymbol{p}_{xy}$ of $x(n)$ and $y(n)$ is defined as:

$$\boldsymbol{p}_{xy} = [R_{xy}(0)\ R_{xy}(1)\ \ldots\ R_{xy}(N-1)]^T \tag{2.13}$$

2.6.4 Stationary Signals

A stochastic process is one that creates signals which are indeterministic i.e. at any instance in time it is not possible to exactly predict how the signal will evolve in the future. In the derivation of the adaptive filters in this thesis, it is assumed that all signals have the property of being wide-sense stationary (WSS). A stochastic process is WSS if it meets the following criteria (Farhang-Boroujeny 1999, p. 37-38):

1. The expectations (mean values) of the signal are time invariant:

$$E\{x(n)\} = E\{x(n+k)\} \tag{2.14}$$

where k is an arbitrary lag value.

2. The autocorrelation function of the signal is time invariant:

$$R_{xx}(n) = R_{xx}(n+k) \tag{2.15}$$

where again k is an arbitrary lag value.

2.6.5 Speech Signals

Figure 2.4 below shows a plot of Amplitude vs Time for a typical digitised speech signal. It can clearly be seen that this signal is not WSS since its mean values vary over the total time shown. However, it is common practise to assume that speech signals show short term stationary behavior over intervals of about 30-40ms (Oppenheim & Schafer 1989, p. 724) and that this is an adequate amount of time for the adaptive filter to operate correctly.

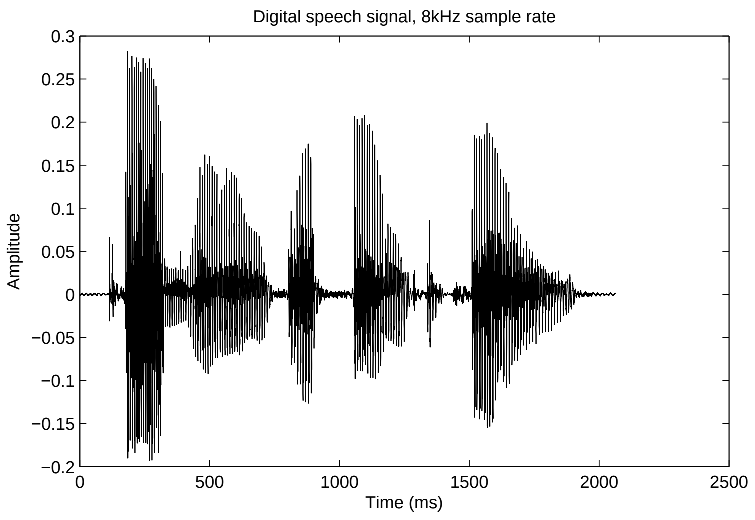

Figure 2.4: A typical digital speech signal.

2.6.6 Discrete-Time Systems

A discrete, linear, time-invariant (LTI) system is characterised by its response, $h(n)$ to the unit impulse function, $\delta(n)$ defined by:

$$\delta(n) = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases} \tag{2.16}$$

The output of the system, $y(n)$ can then be calculated from the *convolution* of a given input, $x(n)$ with $h(n)$ (Poularikas & Ramadan 2006, p. 13):

$$\begin{aligned} y(n) &= x(n) * h(n) \\ &= \sum_{k=-\infty}^{\infty} h(k)x(n-k) \end{aligned} \tag{2.17}$$

To implement this with a computer with finite memory, it is necessary to reduce $h(n)$ to finite order rather than infinite length. Considering only causal systems, (systems whose output depend only on past and present input values) the system output is now:

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k) \tag{2.18}$$

the so called Finite Impulse Response (FIR) filter[4]. The FIR filter structure is shown in Figure 2.5 below. This is an example of a *transversal filter* or *tapped delay-line filter* (Haykin 1995, p. 5). The unit delay operator $z^{-1}\{.\}$ has the effect of lagging the input signal by one sample i.e. $z^{-1}\{x(n)\} = x(n-1)$ so that each element of the filter input vector $\boldsymbol{x}(n)$ is multiplied by the corresponding filter coefficient which is consistent with Equation (2.18).

Figure 2.5 shows that the output of an FIR filter is determined using forward paths only (this is expected since we have the requirement that the system is causal). It can be shown that such filter structures lead to impulse responses that are finite in length and so are inherently stable (Leis 2002, p. 117). It is also possible to use a filter structure that includes both feed-forward and feed-back paths and in doing so creating an impulse response that is infinitely long, a so called *infinite impulse response (IIR) filter*. IIR filters generally require less taps than FIR filters and so use less computations to achieve a similar result. However the the feed-back paths now introduce the possibility that the output of an IIR filter can become unstable oscillations if care is not taken in the choice of feed-back tap-weights. FIR filters are therefore simpler to design and will be used as the basis for the adaptive filters in this project.

[4]An LTI system that is designed for a specific purpose, in our case modelling the acoustic echoes in a room, is called a *filter* (Isen 2008*a*, p. 46).

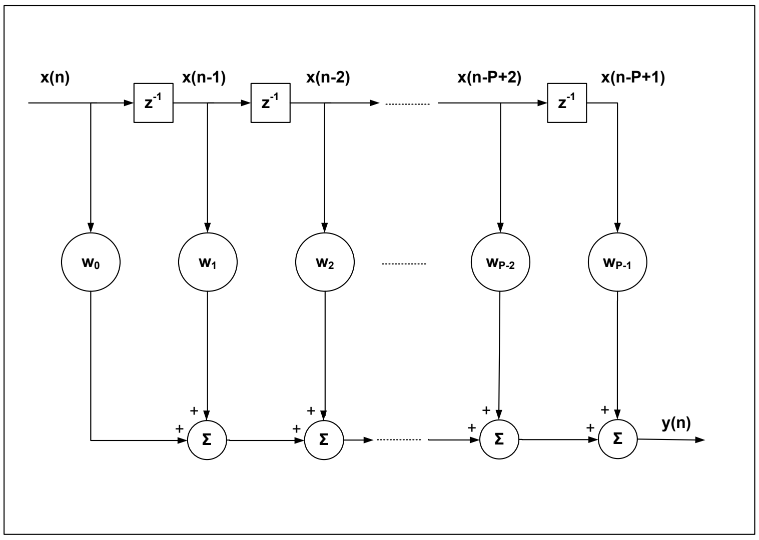

Figure 2.5: An FIR filter of order P-1.

2.7 Chapter Summary

This chapter introduced the background information regarding the source of telephony echoes and also the factors effecting echo perception to the listener. The digital filter implemented in the AEC simulations, the FIR filter, was described as well as some important digital signal processing formulae. It was also shown that discrete-time speech signals, whilst not strictly stationary, were stationary in the wide sense. This important assumption is maintained during the derivation of adaptive filters, the topic of the next chapter.

Chapter 3

Adaptive Filters

3.1 Chapter Overview

The AEC system proposed in the previous chapter relies on the use of an adaptive filter to mimic the unknown impulse response of a room. This chapter builds upon the basic DSP theory of the previous chapter to mathematically derive the adaptive filters used in the AEC simulation. These derivations provide insight into the operation of adaptive filters which is essential in understanding the implementation and output of the AEC simulations.

3.2 Wiener Filters

3.2.1 Mean Squared Error (MSE) Function

Wiener (1949) was the first to propose the filter system to recover a desired signal, $d(n)$ from an additive noise corrupted input, $x(n) = d(n) + v(n)$ as shown in Figure 3.1 below. The goal is to find a filter with an input signal $x(n)$ and output that is an estimate of the desired signal, $\hat{d}(n)$ such that the MSE function:

$$\begin{aligned} J &= E\{[d(n) - \hat{d}(n)]^2\} \\ &= E\{e^2(n))\} \end{aligned} \tag{3.1}$$

is minimised.

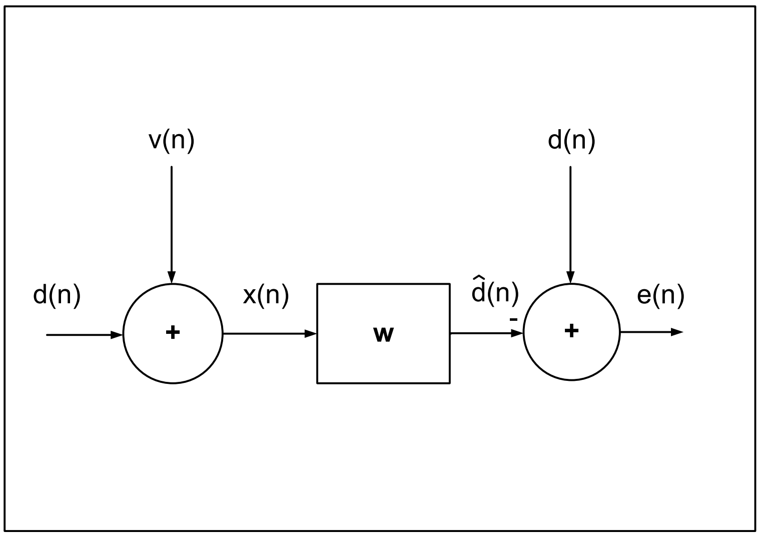

Figure 3.1: Wiener filter block diagram.

Let $\boldsymbol{w}(n) = [w(0)\ w(1)\ \ldots\ w(N-1)]^T$ be the vector of coefficients for a N-1 order transversal FIR filter and $\boldsymbol{x}(n) = [x(n)\ x(n-1)\ \ldots\ x(n-N+1)]^T$ be the input. The output, $\hat{d}(n)$ is then given by the convolution of $\boldsymbol{w}(n)$ with $\boldsymbol{x}(n)$ as in Equation (2.18) above:

$$\begin{aligned} \hat{d}(n) &= \sum_{k=0}^{N-1} w(k)x(n-k) \\ &= \boldsymbol{w}^T(n)\boldsymbol{x}(n) \end{aligned} \tag{3.2}$$

Substituting this result into Equation (3.1) gives:

$$\begin{aligned}
J(\boldsymbol{w}) &= E\{[d(n) - \hat{d}(n)]^2\} \\
&= E\{[d(n) - \boldsymbol{w}^T\boldsymbol{x}(n)]^2\} \\
&= E\{[d(n) - \boldsymbol{w}^T\boldsymbol{x}(n)][d(n) - \boldsymbol{w}^T\boldsymbol{x}(n)]\} \\
&= E\{d^2(n) - \boldsymbol{w}^T\boldsymbol{x}(n)d(n) - d(n)\boldsymbol{w}^T\boldsymbol{x}(n) + \boldsymbol{w}^T\boldsymbol{x}(n)\boldsymbol{x}^T(n)\boldsymbol{w}\} \\
&= E\{d^2(n)\} - 2\boldsymbol{w}^T E\{d(n))\boldsymbol{x}(n)\} + \boldsymbol{w}^T E\{\boldsymbol{x}(n)\boldsymbol{x}^T(n)\}\boldsymbol{w} \\
&= \sigma_d^2 - 2\boldsymbol{w}^T\boldsymbol{p}_{dx} + \boldsymbol{w}^T\boldsymbol{R}_x\boldsymbol{w}
\end{aligned} \tag{3.3}$$

where σ_d^2 is the variance of $d(n)$.

Equation (3.3) is therefore an N-dimensional quadratic surface with respect to the coefficients of $\boldsymbol{w}$ - the so called *mean squared error surface.*

3.2.2 The Wiener Solution

The first derivative of $J(\boldsymbol{w})$ with respect to the coefficients of $\boldsymbol{w}$ is:

$$\begin{aligned}
\nabla J(\boldsymbol{w}) &= \left[\frac{\partial}{\partial w_0}\ \frac{\partial}{\partial w_1}\ \cdots\ \frac{\partial}{\partial w_{N-1}}\right]^T \left(\sigma_d^2 - 2\boldsymbol{w}^T\boldsymbol{p}_{dx} + \boldsymbol{w}^T\boldsymbol{R}_x\boldsymbol{w}\right) \\
&= 2\boldsymbol{R}_x\boldsymbol{w} - 2\boldsymbol{p}_{dx}
\end{aligned} \tag{3.4}$$

From Equation (3.1) it can be seen that the *mean squared error surface* will always concave upwards and so its minimum point can be found by setting $\nabla J(\boldsymbol{w})$ to zero. This optimal solution is given the symbol $\boldsymbol{w}^o$:

$$\begin{aligned}
2\boldsymbol{R}_x\boldsymbol{w}^o - 2\boldsymbol{p}_{dx} &= \boldsymbol{0} \\
\boldsymbol{w}^o &= \boldsymbol{R}_x^{-1}\boldsymbol{p}_{dx}
\end{aligned} \tag{3.5}$$

3.2.3 Steepest Descent Algorithm

The optimal solution can also be found iteratively utilising the fact that $-\nabla J(\boldsymbol{w})$ points in the direction of steepest descent of the MSE surface. Starting with an estimate of $\boldsymbol{w}(n)$, the next estimate equals this value plus a small step in the direction of $-\nabla J(\boldsymbol{w}(n))$:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \hat{\mu}\nabla J(\boldsymbol{w}(n)) \tag{3.6}$$

where $\hat{\mu}$ is a constant that must be kept small so that the algorithm does not become unstable.

The main limitation in using a Wiener Filter for AEC is that calculating $\nabla J(\boldsymbol{w}(n))$ accurately requires prior knowledge of both $\boldsymbol{R}_x$ and $\boldsymbol{p}_{dx}$ and so $x(n)$ and $d(n)$ must be stationary signals. For a real-time AEC system, it is therefore necessary to find approximations for $\boldsymbol{R}_x$ and $\boldsymbol{p}_{dx}$ that do not require that $x(n)$ and $d(n)$ are stationary. This is achieved in the Least Mean Squares algorithm described in the next section.

3.3 Least Mean Square (LMS) Adaptive Filter

The LMS algorithm is a widely used adaptive filter based on the Wiener Filter. Developed in 1959 by Widrow and Hoff, (Widrow & Hoff 1960) the LMS algorithm is able to operate without the requirement of stationary signals (as in the Wiener Filter) though this leads to a small error between the final solution and the optimal Wiener solution as seen below.

Following on from the Wiener filter derivation, we introduce the *instantaneous estimates* of $\boldsymbol{R}_x$ and $\boldsymbol{p}_{dx}$:

$$\begin{aligned} \boldsymbol{R}_x &\approx \boldsymbol{x}(n)\boldsymbol{x}^T(n) \\ \boldsymbol{p}_{dx} &\approx d(n)\boldsymbol{x}(n) \end{aligned} \tag{3.7}$$

Substituting these into Equation (3.4) and Equation (3.6) gives the filter weight update recursion;

$$\begin{aligned}\boldsymbol{w}(n+1) &= \boldsymbol{w}(n) - 2\hat{\mu}\boldsymbol{x}(n)(\boldsymbol{x}^T(n)\boldsymbol{w}(n) - d(n)) \\ &= \boldsymbol{w}(n) + 2\hat{\mu}\boldsymbol{x}(n)(d(n) - \boldsymbol{w}^T(n)\boldsymbol{x}(n)) \\ &= \boldsymbol{w}(n) + \mu e(n)\boldsymbol{x}(n)\end{aligned} \tag{3.8}$$

where $\mu = 2\hat{\mu}$ is called the *step-size* parameter usually chosen to so that $0 < \mu < 1$.

Equation (3.8) shows that the next filter weights depend on the current filter weights, the current filter inputs and the current error signal value, $e(n)$. The error signal is fed-back into the filter which adjusts its tap weights to minimise this error. Such a filter is called an adaptive filter and its block diagram is shown in Figure 3.2.

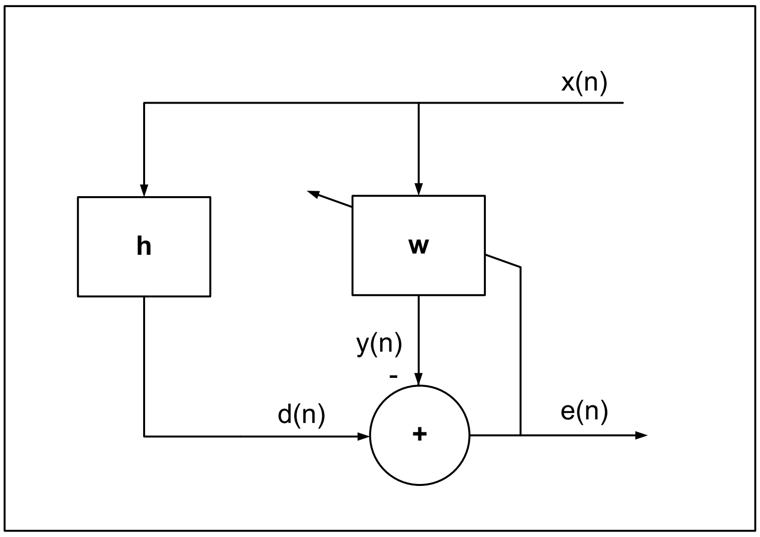

Figure 3.2: Adaptive filter block diagram.

The fact that the calculated gradient is now a function of stochastic signals ($x(n)$ and $e(n)$) rather than the deterministic gradient (as in the Wiener steepest descent method) leads the LMS algorithm to be categorised as a *stochastic gradient algorithm.* Whereas the Wiener steepest descent method converges to the Wiener Solution, $\boldsymbol{w}^o$, the LMS algorithm is convergent to an area around $\boldsymbol{w}^o$ that can be made as small as we like by making μ smaller. This type of convergence is called *convergent in the mean square* and leads to there being a slight difference in the final solution that the LMS converges to and the Wiener solution. The measure of this error is called the *misadjustment* (Haykin 1995, p. 365-367).

3.3.1 LMS summary

The LMS algorithm can be expressed as a set of steps that can be programmed into a computer for simulation as follows:

Initialise the filter weights:

$$\boldsymbol{w}(0) = \mathbf{0} \tag{3.9}$$

Then at each iteration calculate:

1. The adaptive filter output:

$$y(n) = \boldsymbol{w}^T(n)\boldsymbol{x}(n) \tag{3.10}$$

2. The estimation error:

$$e(n) = d(n) - y(n) \tag{3.11}$$

3. The updated filter weights:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu e(n)\boldsymbol{x}(n) \tag{3.12}$$

It can be seen from the above summary that the LMS algorithm is simple to implement and that, for an adaptive filter length of L, uses only $L+1$ floating point addition/subtraction operations and $2L+1$ multiplication/division operations per iteration. Therefore it has a linear time complexity ($O(L)$) which makes LMS a suitable choice for implementations with large adaptive filter orders.

3.3.2 Choice of Step-Size Parameter μ

Although making μ smaller leads to a more accurate final solution it also increases the number of iterations needed to reach it. Correspondingly, the *convergence time* (the time taken to reach the final solution) is increased. It is desireable for an adaptive filter to have a small convergence time so that the AEC system can begin working effectively as quickly as possible at the start of a conversation. A small convergence time is also necessary for the adaptive filter to be able to track any changes in the room impulse impulse that might occur during the course of the conversation e.g. people moving within the room will alter the room impulse response.

The LMS algorithm derivation is based on the idea that the output from the adaptive filter $y(n)$ mimics the output from the room impulse filter $d(n)$ which is solely made up from the convolution of the input signal with the room impulse response, $\boldsymbol{x}(n) * \boldsymbol{h}$ as in Figure 3.2. But what happens if the near-end speaker is talking at the same time (the so called *double talk* condition) to create an added near-end signal $s(n)$ as in Figure 2.2? Now the adaptive filter tries to converge on a solution that gives an output of $d(n) + s(n)$ which now contains a random component since $s(n)$ is uncorrelated to $x(n)$. The result is that the adaptive filter coefficients diverge from the optimal solution and echo cancellation performance rapidly deteriorates. This happens at a faster rate for a larger step-size and so μ is usually chosen to be small enough that divergence is minimised during double talk. Therefore choice of μ is critical for LMS based AEC systems and is chosen to balance convergence time and divergence during double talk.

3.4 Normalised Least Mean Square (NLMS) Adaptive Filter

The standard LMS algorithm updates filter weights by adding $\mu e(n)\boldsymbol{x}(n)$ to the previous weights. This adjustment is proportional to the magnitude of $\boldsymbol{x}(n)$ and so will vary as the values in $\boldsymbol{x}(n)$ vary resulting in undesirable *gradient noise amplification*. The NLMS algorithm is a modification of LMS algorithm that attempts to *normalise* these adjustments (Albert & Gardner 1967). The NLMS recursion is:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{\mu}{\varepsilon + \boldsymbol{x}^T(n)\boldsymbol{x}(n)} e(n)\boldsymbol{x}(n) \tag{3.13}$$

where $\boldsymbol{x}^T(n)\boldsymbol{x}(n)$ represents the power of $\boldsymbol{x}(n)$ in the filter. ε is a small constant that avoids division by a very small number when $\boldsymbol{x}^T(n)\boldsymbol{x}(n)$ is small.

3.4.1 NLMS summary

The NLMS algorithm summary is very similar to the LMS summary but uses the normalised step size parameter:

Initialise the filter weights:

$$\boldsymbol{w}(0) = \boldsymbol{0} \tag{3.14}$$

Then at each iteration calculate:

1. The adaptive filter output:

$$y(n) = \boldsymbol{w}^T(n)\boldsymbol{x}(n) \tag{3.15}$$

2. The estimation error:

$$e(n) = d(n) - y(n) \tag{3.16}$$

3. The updated filter weights:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{\mu}{\varepsilon + \boldsymbol{x}^T(n)\boldsymbol{x}(n)} e(n)\boldsymbol{x}(n) \tag{3.17}$$

The above summary shows that the NLMS algorithm, for an adaptive filter length of L, uses only $L+1$ floating point addition/subtraction operations and $3L+2$ multiplication/division operations per iteration. This is L more multiplications than the LMS algorithm but does not increase the time complexity order (it remains $O(L)$). The increased stability of the NLMS algorithm is worth the extra computational cost (Nagumo & Noda 1967) and so this algorithm was implemented in the AEC simulation.

3.5 Recursive Least Squares (RLS) Adaptive Filter

This section introduces the other adaptive algorithm implemented in the AEC simulation, the RLS algorithm, a special case of the Kalman filter. The mathematics behind RLS is significantly more complicated than LMS and so only a summary will be given here. For more detail the accounts presented in (Haykin 1995, p. 562-569) or (Sayed 2008, p. 492-495) are recommended.

Similar to the LMS algorithm which adjusts its filter coefficients in order to minimise a cost function (the mean squared error function in Equation (3.1)), the RLS algorithm recursively adjusts its filter coefficients in order to minimise a cost function called the *exponentially weighted linear least squares function*:

$$C(n) = \sum_{i=1}^{n} \lambda^{n-i} e^2(i) \tag{3.18}$$

where λ is the *forgetting factor* parameter chosen so that $0 < \lambda < 1$ (usually close to 1). It can be seen that the forgetting factor weights older samples exponentially less than newer samples and that smaller values of λ give less weight to older samples than larger values.

3.5.1 RLS Summary

The RLS algorithm summary is as follows:

Initialisations:

1. The filter weights:

$$\boldsymbol{w}(0) = \boldsymbol{0} \tag{3.19}$$

2. The inverse input correlation matrix:

$$\boldsymbol{P}(0) = \delta^{-1}\boldsymbol{I} \tag{3.20}$$

where δ is a small positive constant.

Then at each iteration calculate:

1. The gain vector:

$$\boldsymbol{k}(n) = \frac{\boldsymbol{P}(n-1)\boldsymbol{x}(n)}{\lambda + \boldsymbol{x}^T(n)\boldsymbol{P}(n-1)\boldsymbol{x}(n)} \tag{3.21}$$

2. The estimation error:

$$e(n) = d(n) - \boldsymbol{w}^T(n-1)\boldsymbol{x}(n) \tag{3.22}$$

3. The updated filter weights:

$$\boldsymbol{w}(n) = \boldsymbol{w}(n-1) + \boldsymbol{k}(n)e(n) \tag{3.23}$$

4. The updated inverse input correlation matrix:

$$\boldsymbol{P}(n) = \lambda^{-1}\{\boldsymbol{P}(n-1) - \boldsymbol{k}(n)[\boldsymbol{x}^T(n)\boldsymbol{P}(n-1)]\} \tag{3.24}$$

Due to the use of matrix as opposed to vector operations, the RLS algorithm uses considerably more computations than LMS or NLMS. The above summary shows that the RLS algorithm, for an adaptive filter length of L, uses $3L^2 + L$ floating point addition/subtraction operations and $4L^2 + 5L$ multiplication/division operations per iteration. Therefore it has a time complexity order of $O(L^2)$ and so runs in polynomial time. This makes RLS a less suitable choice of algorithm as the adaptive filter order is increased.

The RLS algorithm was chosen for the AEC simulations in this project because it has been shown to be capable of achieving lower levels of misadjustment and lower convergence times than NLMS (typically 10× the convergence rate of NLMS (Radecki, Zilic & Radecka 2002)). Comparing the performance of RLS to NLMS will enable the importance of both of these metrics to echo cancellation to be examined.

3.6 Dual-H Filter Configuration

The concept of double-talk was described and shown to be a potentially fatal problem during AEC (see 3.3.2) and so some way of negating its effects must be found. The idea of setting a smaller step size is one way to slow down adaption and therefore divergence during double-talk, however the major drawback is that adaption will also be slowed during the rest of the time as well. A better approach is to detect when double-talk is occurring and then halt adaption only during these periods. A functional block that is capable of this is called a Double-Talk Detector (DTD) (Isen 2008*b*). There are numerous DTDs operate by comparing the near-end signal level to some threshold (e.g. Geigel detectors) but these methods are not effective when the loudness of near-end or echo signals is varied (Gansler, Gay, Sondhi & Benesty 2000).

A more robust method of dealing with the double-talk (and the method implemented in the AEC simulations) is the Dual-H filter configuration (Eriksson & Karlsen 2001). This adds a second filter called the On-line filter ($\boldsymbol{W'}$) to the AEC system block diagram as in Figure 3.3. The adaptive filter $\boldsymbol{W}$ is now called the Off-line filter and its purpose is to converge towards the optimal solution as usual. The figure shows that echo cancellation is performed by the On-line filter and that the error signal from the Off-line filter is now only used to update its tap weights. The On-line filter is not adaptive but updates its tap weights from the Off-line adaptive filter only if the Off-line ERLE is greater than the On-line ERLE. That is, the On-line filter is only updated if the Off-line filter is performing better at cancelling echoes. Since this performance will drop sharply during periods of double-talk, the On-line tap weights are frozen during this time. The overall effect is that the On-line filter behaves similar to an adaptive filter in normal operation but switches to a static filter during double-talk.

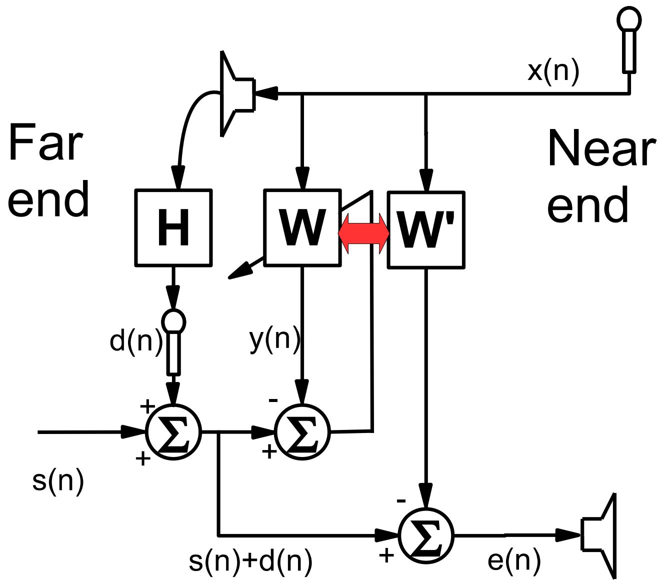

Figure 3.3: The Dual-H AEC system.

3.7 Chapter Summary

This chapter introduced an important class of linear optimal filters, the Wiener Filter, which was used as the mathematical basis for the adaptive filters used in this project. A derivation of a basic adaptive filter algorithm, LMS, was given as well as the details of the popular refined version, NLMS. A more sophisticated algorithm that promises increased performance at the expense of increased computations, RLS, was then described. A summary of the recursions for each algorithm was then given which were used as the basis for the AEC computer simulations described in Chapter 5. These summaries show that although the mathematical theory behind the adaptive filters is fairly complex, the algorithms themselves are very simple to implement.

Chapter 4

Adaptive Echo Cancellation in VoIP

4.1 Chapter Overview

The adaptive filters described in the previous chapter have a wide range of applications. For example they have successfully been used for active noise cancellation, periodic signal extraction, linear prediction and more. Implementing each of these applications requires a knowledge the specific operating environment of the adaptive filter. In our case the environment is a VoIP channel and so this chapter describes the characteristics of a VoIP network and the consequences that they might have to AEC.

4.2 VoIP Networks

The term ‘VoIP’ refers to the technology that enables voice communications to be conducted over IP networks such as the internet. Whilst the size and complexity of a VoIP implementation can range from large, fully featured corporate networks to single home phones, all will consist of the same basic signal flow shown in Figure 4.1. The analogue signal from the telephone is first digitised into a pulse code modulation (PCM) signal by a voice coder-decoder (codec). The PCM signal is then compressed and packetised before transmission over the IP network where it is decompressed and converted back to an analogue signal (CISCO 2006).

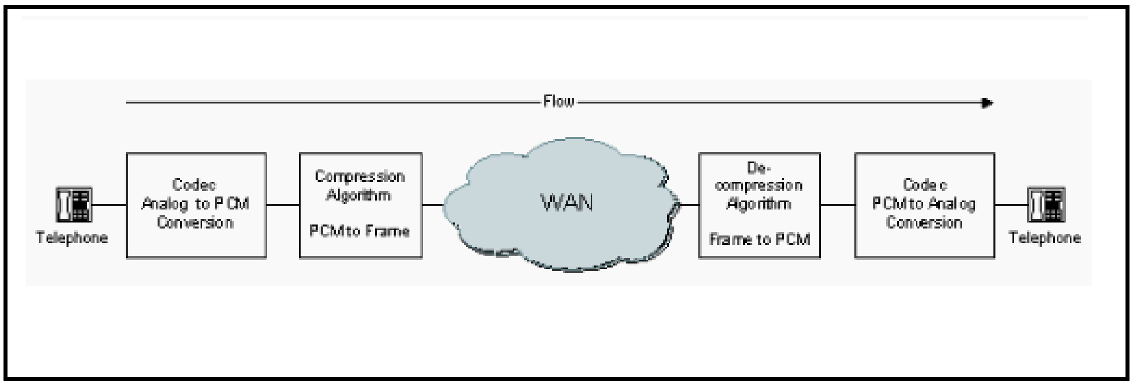

Figure 4.1: End-to-end signal flow in a simplified VoIP transmission (reproduced from `http://www.cisco.com/en/US/tech/tk652/tk698/technologies_white_paper09186a00800a8993.shtml`).

There are a variety of signaling protocols that might be used to control a VoIP session such as H.323, IAX and Session Initiation Protocol (SIP). However, the choice of protocols is largely irrelevant to our AEC system since any DSP must take place on the PCM signals which are common to any choice of protocol.

Figure 4.2 shows the signal flow for a far-end transmission with more detail. This is a more general VoIP circuit in which the signal may travel over the PSTN before it is is enters the packet network at a VoIP gateway. It shows that the speech signal generated by the far-end analogue telephone travels to the PSTN on a 2-wire analogue circuit before it is digitised (usually at a digital hybrid transformer at the local exchange) and then packetised at the VoIP gateway. The packets travel over the network before a reverse of the above process converts the packets back to a PCM signal then analogue before it travels to the near-end telephone to be played over the speaker.

4.3 Difficulties of AEC over VoIP

There is much published research reporting the difficulties of AEC over VoIP (see (Periakarruppan, Low, Azhar & Rashid 2006), (Benetti, Damiani & Houngue 2008), (Ding, El-Hennawey & Goubran 2006) for example). The main focus of the research is

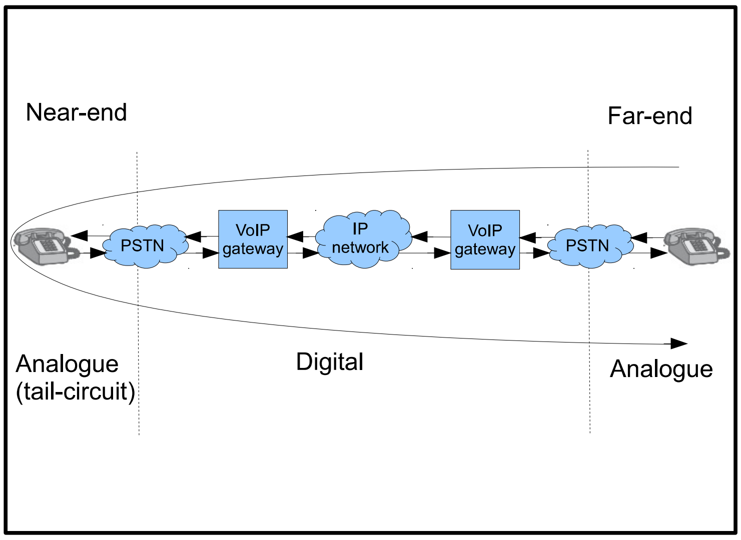

Figure 4.2: VoIP channel analogue/digital signals.

the fact that VoIP channels have certain characteristics that have a negative effect on adaptive filter operation and make AEC difficult. These are:

- Large delays in packet networks.
- Variable delays due to dynamic de-jitter buffers.
- Dropped packets.
- Lossy compression/decompression algorithms introducing non-linearities.

4.3.1 Delay in Packet Networks

Delay in VoIP networks can be either be fixed (e.g. coder delay, packetisation delay) or variable (e.g. queuing delay, network delay). Both types of delay are undesirable because longer round trip delays increase echo perception and can lead to conversation overlap. A generally accepted upper limit for a good quality connection is 200ms one way delay (CISCO 2006).

Network delays (network jitter) are the largest source of variable delays and cause packets to arrive out of order at the receiving end. A de-jitter buffer is used to hold incoming packets for an amount of time before passing them on in correct order. De-jitter buffers therefore decrease the variable delay in the VoIP channel whilst increasing the fixed delay.

It is important that the de-jitter buffer is of the correct length. If it is too long then the delay increases too much. If it is too short then it will fail to eliminate jitter resulting in dropped packets. Some de-jitter buffers are able to optimise their length by dynamically adapting to changing levels of jitter on the network. This will have the effect of constantly changing the length of the echo path and so the adaptive filter will need to be able to track these changes quickly enough.

4.3.2 Variable Delay

It is important that the de-jitter buffer is of the correct length. If it is too long then the delay increases too much. If it is too short then it will fail to eliminate jitter resulting in dropped packets. Some de-jitter buffers are able to optimise their length by dynamically adapting to changing levels of jitter on the network. This has the effect of constantly changing the length of the echo path and so the adaptive filter will need to be able to track these changes quickly enough.

4.3.3 Dropped Packets

Depending on the codec used, dropped packets could be substituted with some approximation of the transmitted signal. This approximation will not be perfect and the sporadic nature of dropped packets means that the adaptive filter will find it difficult to track these changes.

4.3.4 Codec Non-Linearities

Lossy compression/decompression algorithms introduce non-linearities to the echo path which may not be able to be cancelled fully using a linear FIR adaptive filter.

4.4 AEC System Placement in the VoIP Network

The AEC difficulties described in 4.3 are all based on phenomena which occur in the packet network and so will only effect echo paths that incorporate the packet network. Figure 4.3 shows that the obvious location to place an AEC system to cancel far-end echo is at the far-end gateway. But this envelops the packet network and so leads to the above difficulties. It is also a long path - the packet network has a typical one-way delay of >80 ms as opposed to the PSTN which has typical one-way delay times of <10 ms for local calls (usually the VoIP gateway is chosen to be within the local call radius to save on call costs).

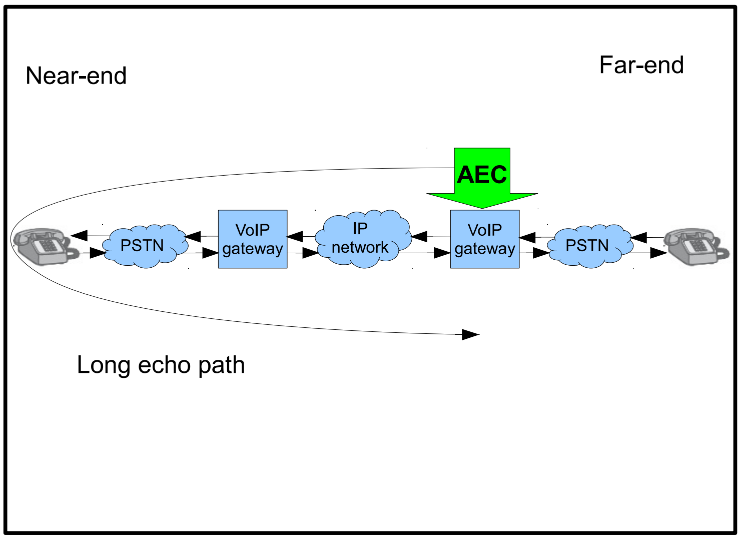

Figure 4.3: The natural AEC system placement to cancel far-end echo.

A better placement for the AEC system is at the near-end VoIP gateway as shown in Figure 4.4. Echoes can only be created on the analogue sections of the circuit where send and receive signals are transmitted as superimposed voltage waves on the 2-wire loop. The analogue sections of the circuit are also where hybrid and acoustic echoes can be generated. The digital portion of the circuit uses dedicated send and receive lines so any electrical reflections generated there are not transmitted through the channel.

Far-end echoes that are generated at the far-end analogue section will not have a long enough delay to be perceived by the listener as being echo - the signal will be masked by the side-tone signal generated in his telephone. Therefore the only section of the VoIP channel that can generated unwanted echoes is the near-end analogue circuit (tail-circuit) and so the AEC system should be placed at the near-end gateway. As well as avoiding the undesirable effects of the packet network on the echo path, this placement also drastically reduces the length of the echo path.

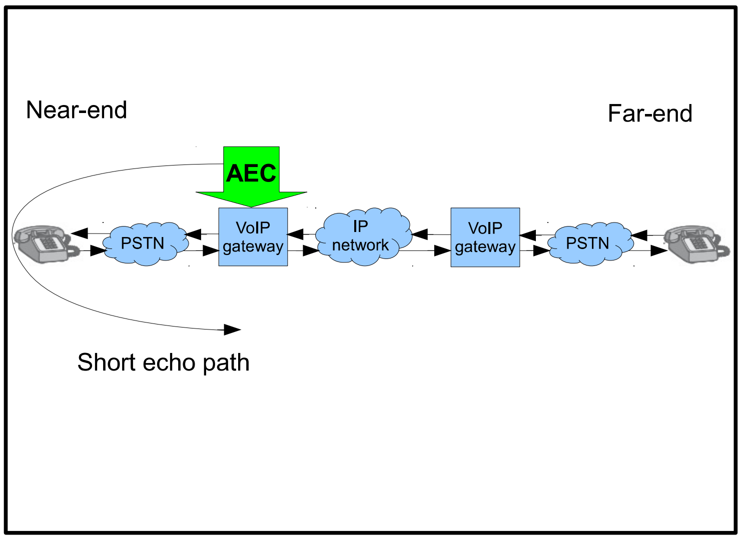

Figure 4.4: A superior AEC system placement to cancel far-end echo.

4.5 Chapter Summary

This chapter presented the important characteristics of VoIP networks and their possible adverse effects to AEC. These were used as the basis of the testing scenarios used in the AEC simulations. It was also shown that the placement of the AEC system in the VoIP channel has the potential to greatly effect its echo cancellation performance and that the best placement of the system for far-end echo cancellation was at the near-end gateway.

Chapter 5

VoIP Adaptive Echo Cancellation Simulation

5.1 Chapter Overview

The previous chapter described the important characteristics of VoIP networks and the consequences this may have to AEC. These were used as the basis of the testing scenarios used in the AEC simulations. This chapter explains how these scenarios were implemented in the simulation and also explains the way in which the echo path was modeled.

5.2 MATLAB Simulation Implementation

A VoIP channel AEC simulation was implemented in MATLAB as the script file `aec_sim.m` (see Appendix B for the full program listing). The simulation tests the performance of three different adaptive algorithm schemes:

1. A NLMS adaptive filter.

2. A NLMS adaptive filter in a dual-H configuration.

3. A RLS adaptive filter.

The simulated echo path is composed of two parts:

1. The acoustic effects of the near-end room (modeled by $\mathbf{H}$ in Figure 2.2).
2. The VoIP channel.

When the script is run the user is prompted to choose from a selection of experiments that run the simulation under various adverse conditions (see 5.5) and enable the performance of each adaptive algorithm scheme to be evaluated. In this way each experiment manipulates an independent variable (the adverse condition) and measures its effect on a dependent variable (one or more of the various performance metrics explained in 5.6 below).

5.3 Room Acoustic Model

The near-end room acoustic echo is created by convolving the far-end digital speech signal with the rooms impulse response according to Equation (2.18). The room impulse response vector was created using the MATLAB function `rir.m`[1]. Figure 5.1 shows a room impulse response vector of the default configuration used in the simulations - a $3m \times 3m \times 3m$ room with the microphone separated from the speaker by $1m$. This models a hands free telephone or speaker-phone setup in a small room or office.

The figure shows the typical form of a room impulse response: a flat section at the start before a maximum peak and then a rapidly decaying series of peaks. The flat initial section and first spike represents the time taken for the sound waves to travel the 1m from the speaker directly into the microphone. The next peaks are single reflections of sound waves off the walls, floor and ceiling, attenuated as some of the sound wave energy is absorbed by the solid surface. Finally there is a diminishing tail section, caused by multiple reflection pathways from the speaker to the microphone.

[1]`rir.m` by Stephen G. McGovern uses the method of images to create the impulse response. It can be freely downloaded along with supporting documentation from `http://www.2pi.us/`

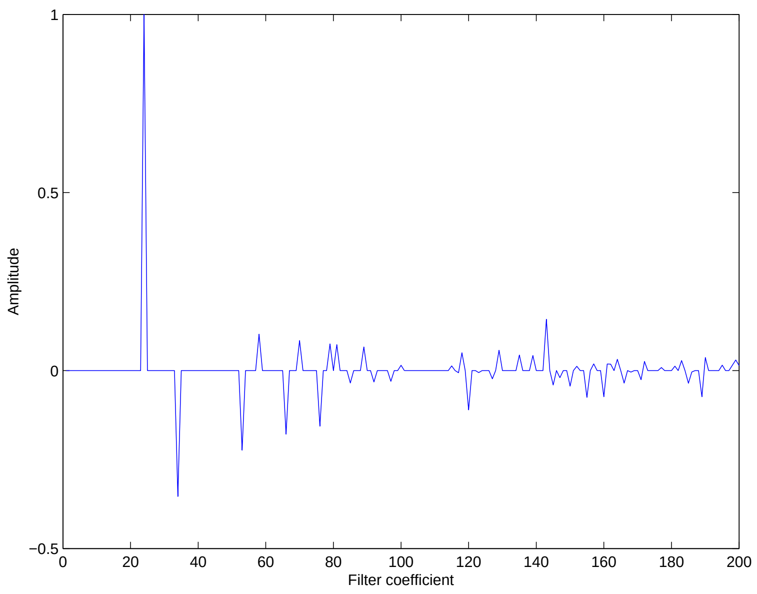

Figure 5.1: Room impulse response of a $3m \times 3m \times 3m$ room with the microphone separated from the speaker by $1m$.

5.4 VoIP Channel Model

Various VoIP channel parameters were incorporated so that the simulation could measure their effect on AEC performance. These are:

Network Delay

The simulation implements fixed delays by lagging the signals at each end of the network.

Background Noise

White noise was added to the near-end signal to see the effects of noise.

Voice Compression

By compressing and decompressing the far-end input signal with a variety of different CODECs and bitrates the effects of compression can be measured.

Dropped Packets

By removing packet-sized sections of the input signal the effects of dropped packets on AEC was able to be measured.

5.5 Test Signals

There are three different speech conditions that the AEC system must be able to operate in:

Far-end speech only

Here the adaptive filter is continuously adapting to the echo path as described in Chapter 3 above.

Near-end speech only

Here the adaptive filter ceases adapting because $\boldsymbol{x}(n) = \mathbf{0}$ and so $\boldsymbol{w}(n+1) = \boldsymbol{w}(n)$ and the adaptive filter tap-weights do not change (see Equations (3.13) and (3.23)).

Double-talk (simultaneous far-end and near-end speech)

Here the adaptive filter tries to adapt to the echo path with the added near-end signal. As mentioned in 3.3.2 the result is that the adaptive filter sees a rapidly changing stochastic echo path which causes the filter coefficients to diverge from the optimal solution. An AEC system must have some way of recognising when double-talk is occurring and halt adaptation when it does. This was accomplished by the DTD inherent in the Dual-H NLMS implementation.

Therefore the input signals to the simulation were predominately short ($\approx 10\ s$) digitised speech signals which were chosen to included instances of each of the above when required by the experiment. Speech signals were not appropriate for certain experiments such as during convergence time measurements. In this case white Gaussian noise was used in place of the far-end signal to ensure that filter adaptation was taking place during the entire duration being measured (if speech signals were used then adaptation will stop during periods of silence and give inaccurate measurements).

The digitised speech signals were created by taking a high bit-rate `wav` file and resampling it at 8 kHz sample rate and 8 bits/sample using `ffmpeg`[2]. This was done so

[2]`ffmpeg` is a freeware audio/video file transcoder with a large codec library. It can be freely downloaded from `http://ffmpeg.org/`

that the signals resembled the PCM signals from the popular ITU-T voice codec G.711 which are also encoded at 8 kHz and 8 bits/sample. These signals were used in all experiments apart from the experiment in 6.7 which used a variety signals encoded at different bit-rates to examine the effects of codec compression.

5.6 AEC System Performance Metrics

5.6.1 ERLE

The performance of each AEC system was measured by calculating its Echo Return Loss Enhancement (ERLE) value (see 2.4 above for the definition of ERLE). As well as measuring the level of echo cancellation, the ERLE measurement implicitly incorporates a number of other performance metrics (i.e. if the system is deficient in any of these then it will be reflected in a poor ERLE score) and so is a good indicator of overall performance. These are:

Rate of convergence

The speed in which the adaptive filter converges to the optimal solution in a stationary environment. A fast rate is important so that echoes are not noticeable during periods of convergence.

Misadjustment

The difference between the final solution that the adaptive filter converges to and the Wiener solution (see 3.3 above).

Tracking

The ability of the adaptive filter to converge in a non-stationary environment.

As stated in 2.4 the ERLE calculation is inaccurate during double talk. This makes calculation of ERLE highly dependent on the particular speech signals used and the amount of double-talk that occurs. For this reason the Maximum ERLE value is used as the performance metric and the test signals are made to be of a long enough duration for the adaptive filter to converge. During testing the longest convergence time for any adaptive filter was found to be about 2s and so the 10s test signals had more than adequate length.

As well as during the period when double talk is occurring, ERLE calculations are inaccurate for a time just after double talk has finished. Since the ERLE formula Equation (2.2) calculates the signal power levels in a buffer (chosen to be the same length as the adaptive filter in the MATLAB implementation) the buffer will still contain some of the near-end signal power until this time is reached, which gives an inaccurate result. This has to be considered when analysing the output ERLE data from the simulation.

5.6.2 Convergence Time

The convergence time is the time taken for an adaptive filters tap weights to reach the final solution from the initial state ($\boldsymbol{w}(0) = \mathbf{0}$). When this happens, ERLE will be maximal (or nearly maximal). Therefore in this project the convergence time was defined as the time taken from the start of the simulation to when the ERLE reaches 90% of the maximum ERLE.

5.6.3 Residual Error

The residual error signal $e(n)$ is a composition of $s(n)$, $d(n)$ and $y(n)$ according to:

$$e(n) = d(n) + s(n) - y(n) \tag{5.1}$$

Removing $s(n)$ gives the residual echo after cancellation less the near-end input:

$$e(n) - s(n) = d(n) - y(n) \tag{5.2}$$

The smaller this error is the better the level of echo cancellation (for perfect echo cancellation this quantity equals zero) and so it is useful to see how this quantity varies over time during the simulation. In the simulation the absolute value of this error is shown on the output plots labeled as '|Error |'.

5.7 Testing Scenarios

5.7.1 Experiment 1: Adaptive Filter Length

Function File: `exp1_filtlen.m`

Aim: To investigate the effect of increasing the adaptive filter order on echo cancellation performance and convergence rate.

Description: Using a far-end signal of Gaussian white noise, adaptive echo cancellation was carried out using the adaptive filter lengths of 50, 100, 200, 500 and 1000.

Far-end Signal: 10 s of Gaussian white noise (80000 samples sampled at 8 kHz sample rate)

Near-end Signal: No signal (80000 samples with zero amplitude)

Output: A plot of Maximum ERLE vs Filter Length and a plot Convergence Time vs Filter Length.

5.7.2 Experiment 2: Double-Talk

Function File: `exp2_dt.m`

Aim: To investigate the effect of double-talk on echo cancellation performance.

Description: Using overlapping far-end and near-end speech signals to create double-talk, adaptive echo cancellation was carried out using the adaptive filter lengths of 100, 200 and 250.

Far-end Signal: 10 s of speech (sampled at 8kHz sample rate and 8 bits/sample).

Near-end Signal: 10 s silence with 3 small sections of speech (sampled at 8 kHz sample rate and 8 bits/sample).

Output: Plots of absolute residual error (residual echo) vs Sample Number and plots of ERLE vs Sample Number for increasing adaptive filter lengths.

5.7.3 Experiment 3: Tail-Circuit Delay

Function File: `exp3_delay.m`

Aim: To investigate the effect of increasing the tail-circuit delay on echo cancellation performance.

Description: A delay was added to the far-end signal by appending additional zero valued tap weights to the start of the RIR filter. Adaptive echo cancellation was then carried out using the adaptive filter lengths of 100, 200 and 250.

Far-end Signal: 10 s digitised of speech (sampled at 8 kHz sample rate and 8 bits/sample)

Near-end Signal: 10 s of silence (zero amplitude signal)

Output: Plots of absolute residual error (residual echo) vs Sample Number and plots of ERLE vs Sample Number for increasing adaptive filter lengths.

5.7.4 Experiment 4: RIR Filter

Function File: `exp4_longrir.m`

Aim: To investigate the effect of making the room impulse response filter length to be longer than the adaptive filter length and increasing the reverberation time of the room.

Description: The simulation was run for using room impulse response filter with 1000 coefficients using the adaptive filter lengths of 100, 200 and 250. The reverberation time of of the room was increased by using increasing values of reflection coefficient ($R = 0.2,\ 0.4,\ 0.6,\ 0.8$).

Far-end Signal: 10 s digitised of speech (sampled at 8 kHz sample rate and 8 bits/sample)

Near-end Signal: 10 s of silence (zero amplitude signal)

Output: Plots of absolute residual error (residual echo) vs Sample Number and plots of Maximum ERLE vs Sample Number for increasing adaptive filter lengths and values of *R*.

5.7.5 Experiment 5: Background Noise

Function File: `exp5_noise.m`

Aim: To investigate the effect of adding background noise to the near-end signal on echo cancellation performance.

Description: The simulation was run using speech as the far-end signal and varying levels of white Gaussian noise as the near-end signal. The adaptive filter lengths of 100, 200 and 250 were tested for the following noise levels: 0.3%, 1%, 3%, 10% and 30% (expressed as a percentage of the maximum signal level amplitude).

Far-end Signal: 10 s of Gaussian white noise (80000 samples sampled at 8 kHz sample rate)

Far-end Signal: 10 s of Gaussian white noise of varying maximum amplitude (80000 samples sampled at 8 kHz sample rate)

Output: Plots of absolute residual error (residual echo) vs Sample Number for a variety of SNR values and plots of Maximum ERLE vs SNR for a variety of adaptive filter lengths.

5.7.6 Experiment 6: Codec Compression

Function File: `exp6_comp.m`

Aim: To investigate the effects of varying the level of compression of the far-end signal on echo cancellation performance.

Description: The simulation was run using far-end speech that had been compressed using the mp2 codec at various bit-rates before being decompressed to the standard 64 kbps PCM signal used in the other experiments. The adaptive filter lengths of 100, 200 and 250 were tested for the following compression bit-rates: 8, 16, 32 and 64 kbps.

Far-end Signal: 10 s digitised of speech (encoded at varying levels of compression (bit-rates) then decoded to a 8 kHz sample rate and 8 bits/sample)

Near-end Signal: 10 s of silence (zero amplitude signal)

Output: Plots of absolute residual error (residual echo) vs Sample Number for a variety of bit-rate values and plots of Maximum ERLE vs bit-rate for a variety of adaptive filter lengths.

5.7.7 Experiment 7: Dropped Packets

Function File: `exp7_drop_pack.m`

Aim: To investigate the effect of bursts of dropped packets on echo cancellation performance.

Description: The far-end signal of was made up of Gaussian white noise with sections of the signal replaced by discrete lengths zeros to simulate dropped packets. The packet sizes were calculated according to typical parameters used by the G.711 codec and dropped packet bursts occurred once every second. The simulation was run for the adaptive filter lengths of 100, 200 and 250 and dropped packet burst lengths of 5, 15, 25, 35 and 45.

Far-end Signal: 10 s of Gaussian white noise (80000 samples sampled at 8kHz sample rate) with dropped packets replaced by zeros.

Near-end Signal: 10 s of silence (zero amplitude signal).

Output: Plots of absolute residual error (residual echo) vs Sample number for a variety of Packet Loss Burst Size values, plots of ERLE vs Sample Number and plots of Maximum ERLE vs Packet Loss Burst Size for a variety of adaptive filter lengths.

5.7.8 Experiment 8: Complete Simulation

Function File: `exp8_combo.m`

Aim: To measure the echo cancellation performance of the Dual-H NLMS filter using the full range of VoIP conditions.

Description: The simulation was run using a NLMS Dual-H filter with a length of 500 using a combined range of adverse conditions including double-talk, tail-circuit delay, dropped packet bursts, compressed input signal and long RIR filter.

Far-end Signal: 10 s digitised of speech (encoded then decoded using the G.711 codec to a 8 kHz sample rate and 8 bits/sample)

Near-end Signal: 10 s silence with three small sections of speech (sampled at 8 kHz sample rate and 8 bits/sample). White Gaussian noise was then added.

Output: Plots of absolute residual error (residual echo) vs Sample number and a plot of ERLE vs Sample Number.

5.7.9 Simulation User Instructions

- The program code for the simulation was written in MATLAB version R2007a and so it is recommended that either this or a later version is used when running the simulation to ensure compatibility.
- Ensure that the main script file `aec_sim.m` and also the following experiment function files are in the current directory: `exp1_filtlen.m`, `exp2_dt.m`, `exp3_delay.m`, `exp4_longrir.m`, `exp5_noise.m`, `exp6_comp.m`, `exp7_drop_pack.m` and `exp8_combo.m`.
- Ensure that the following helper function files are in the current directory: `rir.m`, `fconv.m` and `mtit.m.m`.
- Ensure that the following sound files are in the current directory: `karl10s_mp2_8_dec.wav`, `karl10s_mp2_16_dec.wav`, `karl10s_mp2_32_dec.wav`, `karl10s_mp2_64_dec.wav`, `karl10s_8kHz_8bit.wav`, `karl10s_8kHz_8bit_mulaw.wav` and `ricky10s_8kHz_8bit.wav`.
- Start MATLAB and navigate to the current directory.

- At the command line type '`aec_sim`' and then press '`ENTER`' to start the simulation.

- Follow the instructions to run the desired experiments.

5.8 Chapter Summary

This chapter gave a detailed account of the methodology behind the AEC simulations. It also described the various performance metrics measured during the simulations and how these should be interpreted to assess the echo cancelling effectiveness of each each of the algorithms tested.

Chapter 6

Results and Discussion

6.1 Chapter Overview

The results and analysis of the AEC experiments are presented in this chapter. The first seven experiments investigate the effect of altering a single independent variable (such as dropped packets or noise levels) on the echo cancellation performance. Measurements of various adaptive filter performance metrics were taken during each experiment and then plotted for analysis. The final experiment simulates a real-world VoIP AEC scenario in which all of the adverse conditions of the previous experiments are applied simultaneously.

6.2 Experiment 1: Adaptive Filter Length

Figure 6.1 shows that for a filter length of L, the maximum ERLE rose proportionally to $log_{10}L$ (note the base 10 logarithmic scale on the x-axis). This demonstrates that increasing the number of tap-weights in the adaptive filter leads to increased echo cancellation performance but that the performance gain drops off the higher you go. It also shows that the rate of increase is practically the same for both NLMS and RLS. Considering that NLMS is $O(L)$ and RLS is $O(L^2)$ this shows that NLMS is able to achieve a similar level of performance as RLS at a reduced level of computational effort, particulary as the filter length increases. This makes NLMS a better choice of algorithm for large adaptive filters.

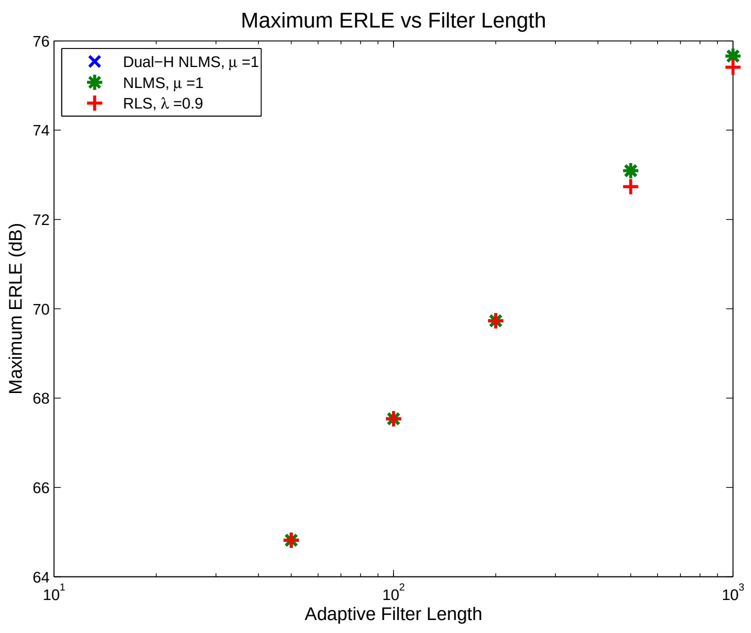

Figure 6.1: Experiment 1: Maximum ERLE vs Adaptive filter length.

Figure 6.2 shows that convergence time rose linearly with the length of the adaptive filter. The NLMS algorithm increased at a rate of approximately 1.2 ms per additional filter weight and the RLS algorithm increased at a rate of approximately 0.13 ms per additional filter weight. This shows that RLS is able to converge roughly 10 times faster than NLMS which would make it the better choice of algorithm in non-stationary environments such as in a room where people were moving around.

6.3 Experiment 2: Double-Talk

Figure 6.3 shows error plots for each algorithm during 10 s of far-End speech overlapped with three short sections of near-End speech causing double-talk. The topmost plot shows when the double-talk occurred during the simulation. It can clearly be seen that the Dual-H filter design is very effective during double-talk and that echo cancellation performance does not drop.

The NLMS and RLS algorithms are not able function correctly as shown by the large

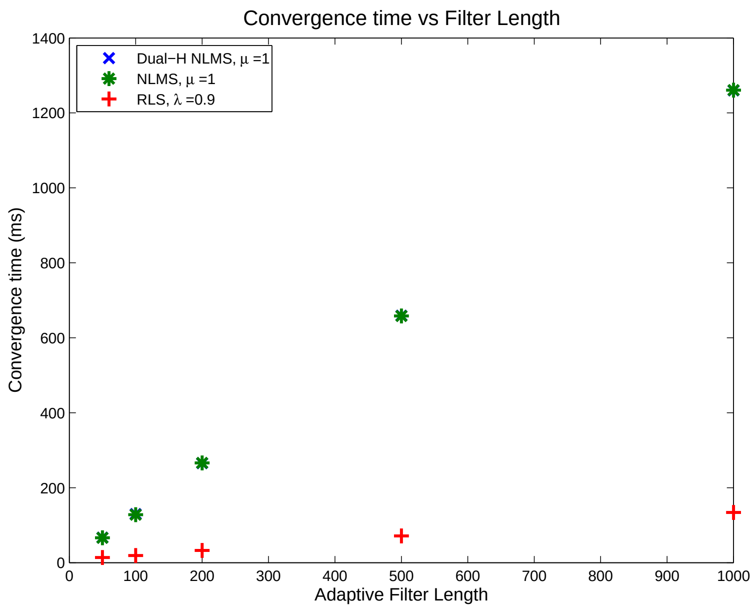

Figure 6.2: Experiment 1: Convergence Time vs Adaptive filter length.

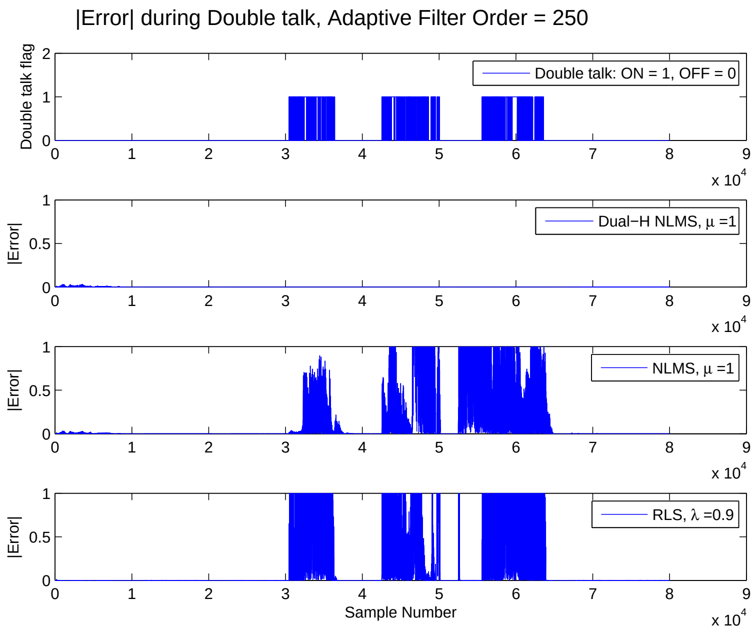

Figure 6.3: Experiment 2: Double-talk error plots.

residual errors during the double-talk periods. This is due to the adaptive filter diverging away from the correct solution as it tries to adapt to the near-End input which acts as noise to the system (see 3.3.2). This effect is shown in Figure 6.4 where the top plot shows the adaptive filter coefficients the off-line filter of the Dual-H system (a regular NLMS adaptive filter). At the beginning of the simulation the adaptive filter is operating as required and the coefficients quickly converge from initial values of zero to the correct solution. However the coefficients diverge markedly during the three sections of double talk. The bottom plot shows that the on-line Dual-H filter is able to maintain the correct weights throughout the double-talk.

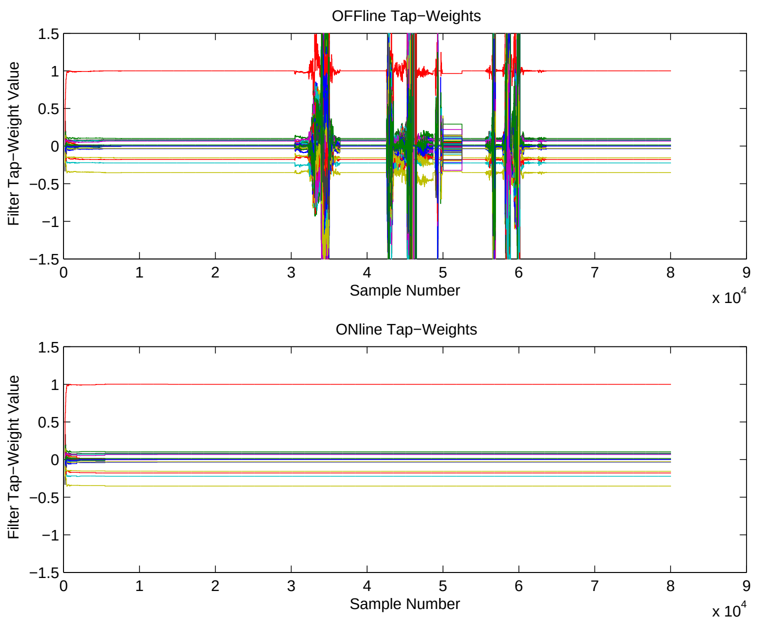

Figure 6.4: Diverging adaptive filter weights during double-talk.

6.4 Experiment 3: Tail-Circuit Delay

Figure 6.5 is a plot of maximum ERLE vs tail-circuit delay. It shows that there is a linearly decreasing relationship between maximum ERLE and tail-circuit delay and that the rate that maximum ERLE drops is approximately 0.9 dB/ms. The apparently poor performance of each algorithm can be explained by the fact that the RIR tail length was left longer than the adaptive filter length (rather than truncating the RIR to match the adaptive filter length) for this experiment and this degrades performance (see Experiment 4 below). This was done because the delay was implemented by inserting a vector of zeros to the start of the RIR filter. If the RIR was truncated then this would remove the tail section, leading to less non-zero elements in the RIR filter. The result would be that longer delays would reduce the number of non-zero elements in the optimal solution leading to an undesirable artificially increased adaptive filter performance.

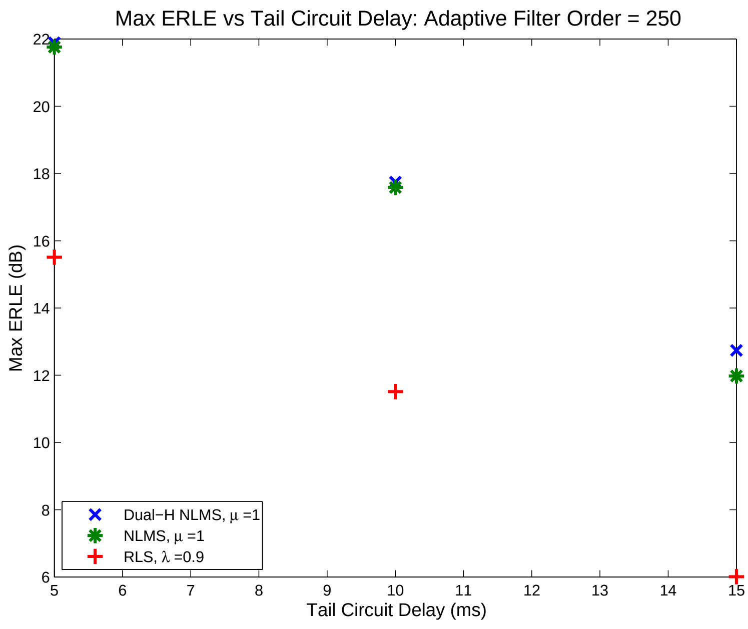

Figure 6.5: Experiment 3: Maximum ERLE vs Tail-circuit delay when the adaptive filter order is 250.

Figure 6.6 (adaptive filter length = 100) and Figure 6.7 (adaptive filter length = 250) compare the error plots when the tail-circuit delay is set to 5 ms. They show that increasing the adaptive filter length increases performance if a tail-circuit delay is present.

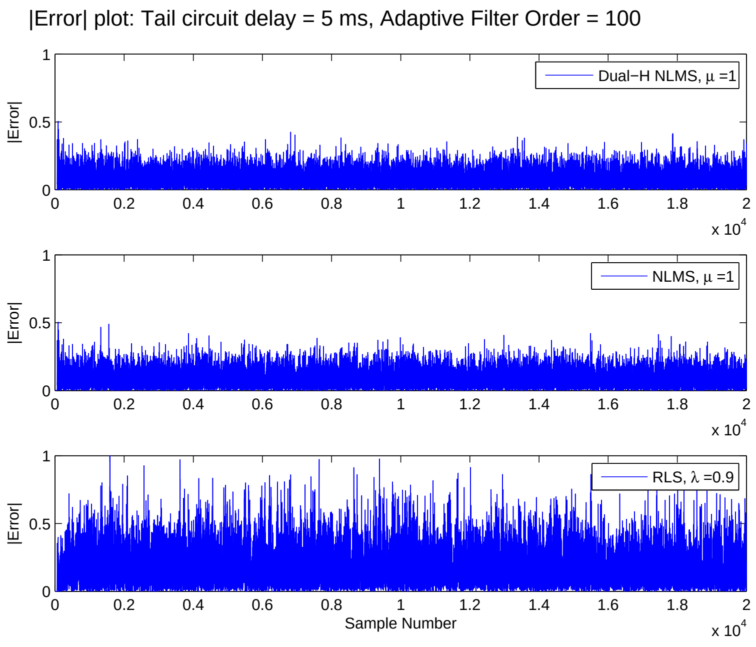

Figure 6.6: Experiment 3: Error plot when the tail-circuit delay is 5ms and the adaptive filter order is 100.

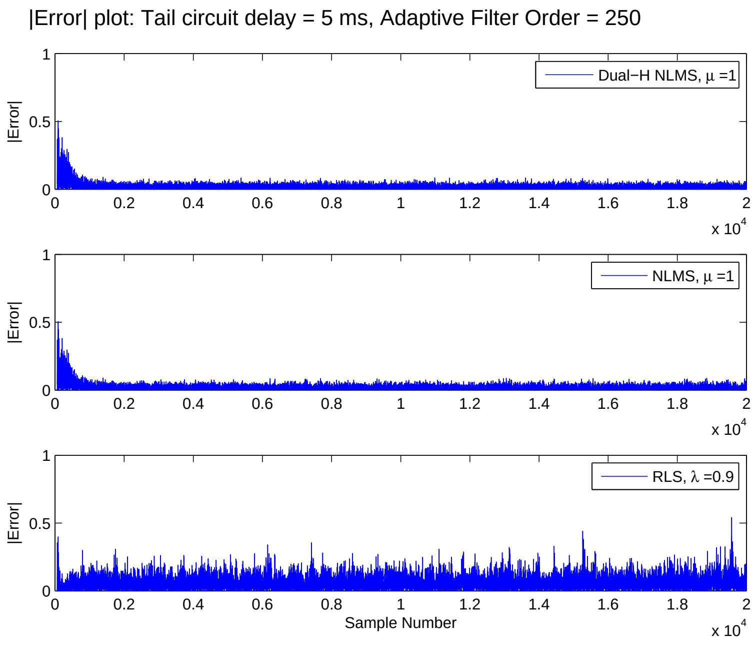

Figure 6.7: Experiment 3: Error plot when the tail-circuit delay is 5ms and the adaptive filter order is 250.

6.5 Experiment 4: RIR Filter

This experiment examined how echo cancellation performance was effected by setting the RIR filter to be longer than the adaptive filter length and then increasing the reverberation in the room, making it more prone to echoes. This was done by using a RIR with length of 1000 and running simulations for adaptive filter lengths up to 250 for a range of room reflection coefficient values (R).

Figure 6.8 shows two RIR filters created using different values of room reflection coefficients with the top plot using R = 0.4 and the bottom plot R = 0.8. The dispersion time of the bottom plot is clearly longer due in this more echoic environment and increases the loudness of the echoes outside of the range of the adaptive filter thereby reducing its effectiveness.

Figure 6.9 is a plot of maximum ERLE vs R for an adaptive filter length of 50. It shows that there is a linearly decreasing relationship between maximum ERLE and R and that the rate that maximum ERLE drops is approximately 5 dB per 0.1 increase in R value.

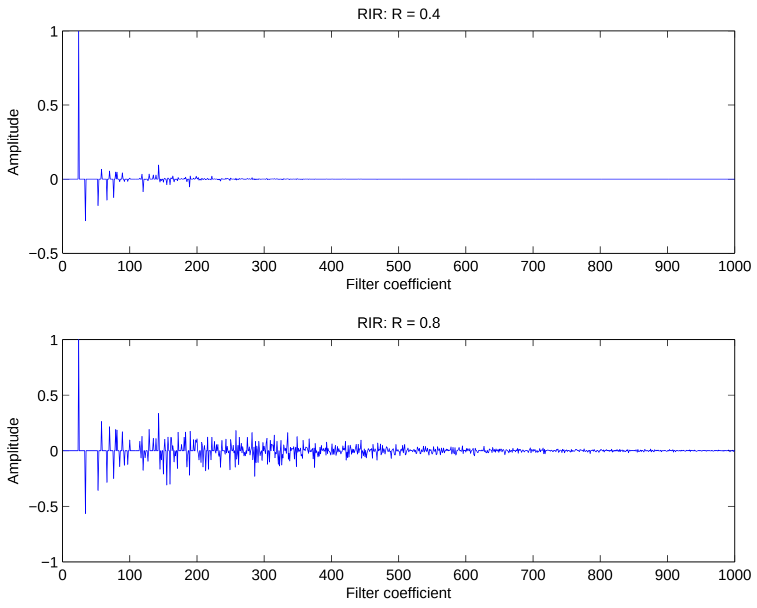

Figure 6.8: Increasing reverberation by increasing the room reflection coefficient, R.

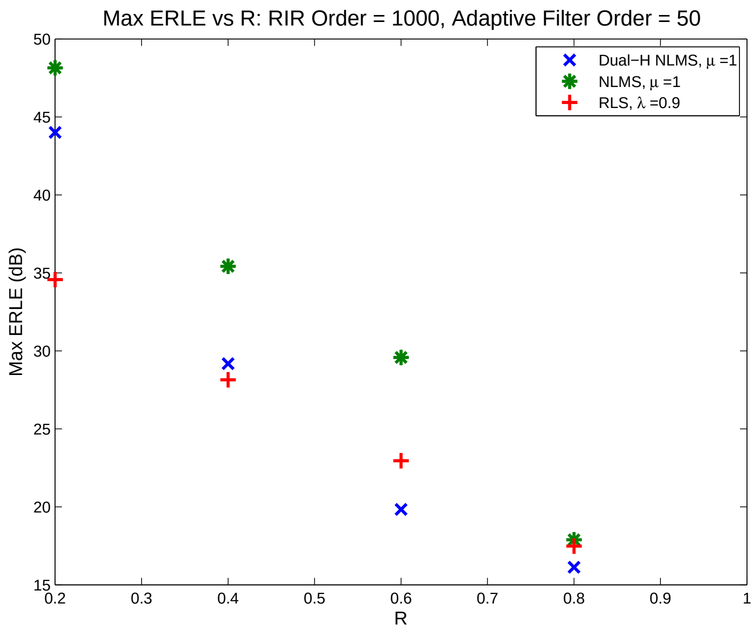

Figure 6.9: Experiment 4: Maximum ERLE vs R for an adaptive filter order of 50.

In Figure 6.10 the adaptive filter length is increased to 250. It shows the same linearly decreasing relationship between maximum ERLE and R and that the rate that maximum ERLE drops is approximately 6 dB per 0.1 increase in R value. It also shows that the maximum ERLE values have increased which demonstrates that increasing the adaptive filter length gives better performance in echoic rooms.

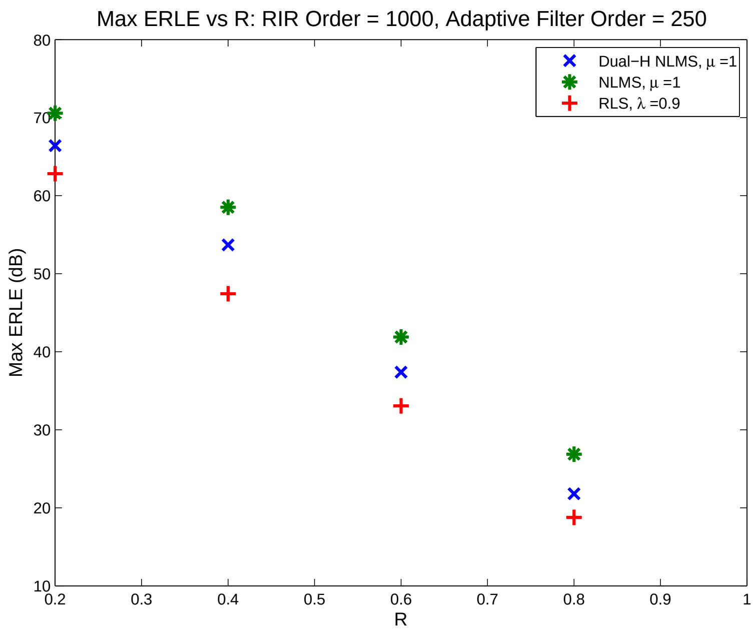

Figure 6.10: Experiment 4: Maximum ERLE vs R for an adaptive filter order of 250.

6.6 Experiment 5: Background Noise

This experiment examined the effects of using increasing levels of white Gaussian as the near-End input. Figure 6.11 (SNR = 40.7) and Figure 6.12 (SNR = 11.1) show error plots during a simulation when the adaptive filter length was 250. They show that for low levels of noise all of the algorithms were capable of removing the far-End echo but they performed poorly as the noise levels increased.

Figure 6.13 and Figure 6.14 show that at SNRs above approximately 20 dB, echo cancellation performance is relatively constant but below this level performance rapidly declines. The figures also show that increasing the adaptive filter length has negligible effect on the performance in this noisy environment.

6.7 Experiment 6: Codec Compression

This experiment looked at the effects of codec compression on echo cancellation performance. Figure 6.15 (adaptive filter length = 100) and Figure 6.16 (adaptive filter length = 250) show the performance of each adaptive algorithm when the far-End signal

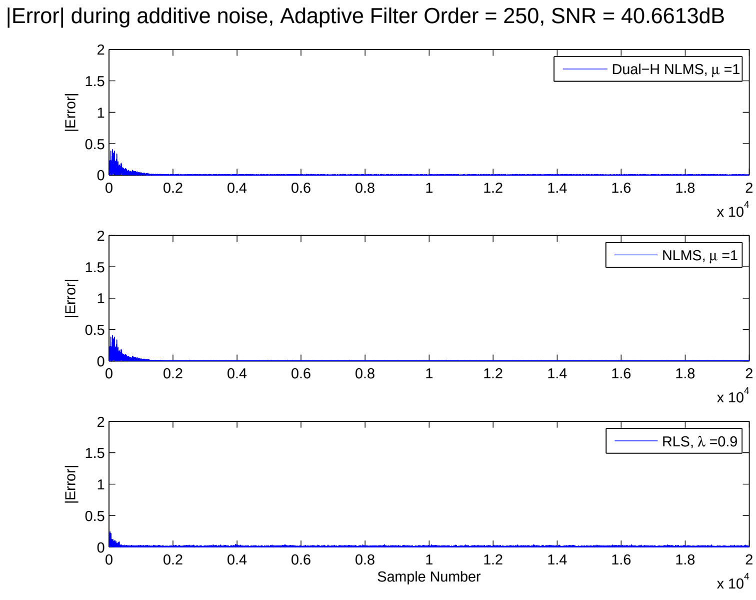

Figure 6.11: Experiment 5: Error when SNR = 40.7 dB.

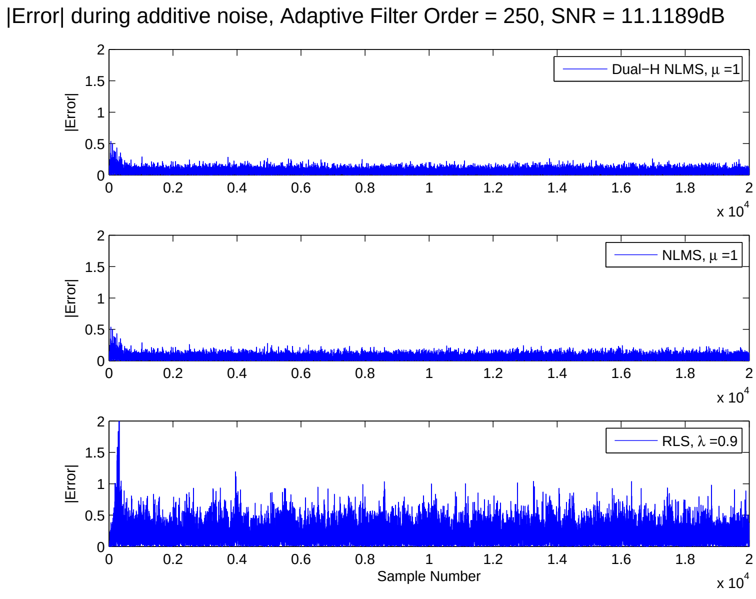

Figure 6.12: Experiment 5: Error when SNR = 11.1 dB.

was compressed and then decompressed at various bit-rates, a process which results in information being lost from the original signal. The figures show that there is negligible a performance difference between the various compression rates and this is due to the placement of the AEC system in the near-end VoIP gateway as opposed to the far-end gateway (see 4.4).

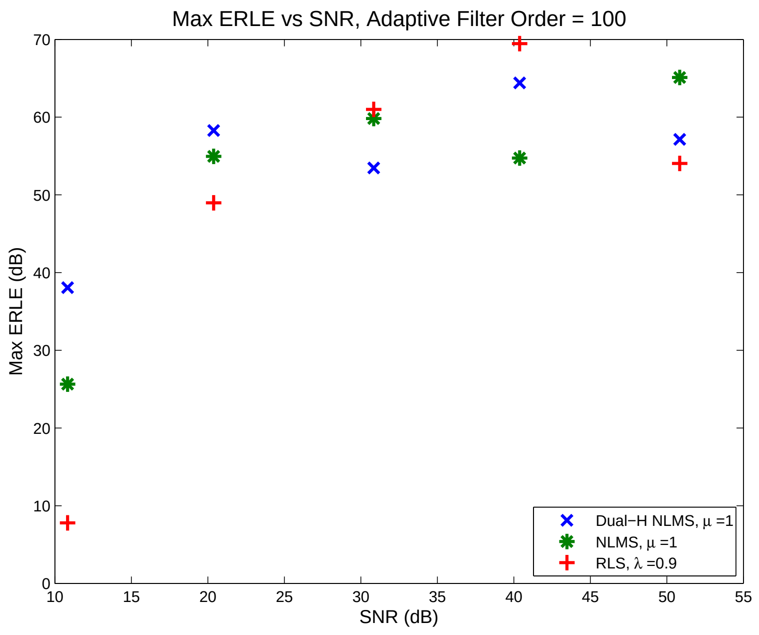

Figure 6.13: Experiment 5: Maximum ERLE vs SNR for an adaptive filter order of 100

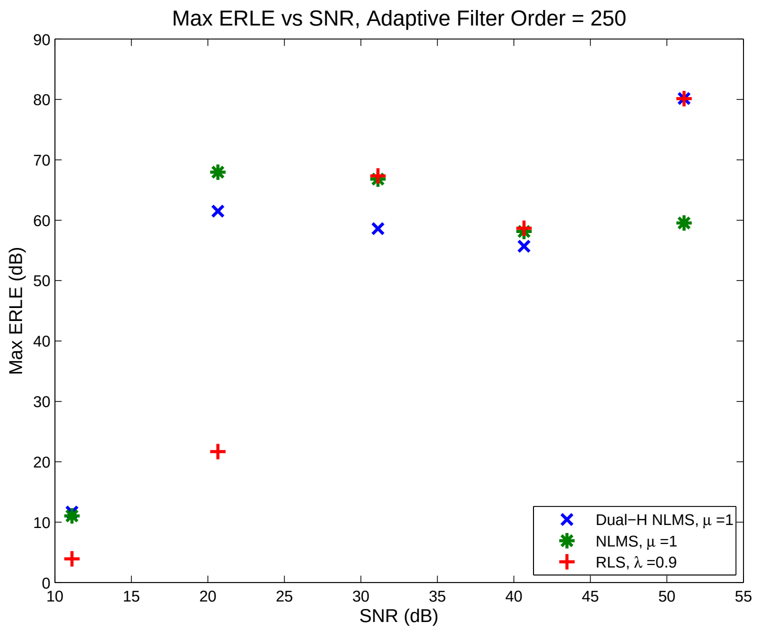

Figure 6.14: Experiment 5: Maximum ERLE vs SNR for an adaptive filter order of 250

6.8 Experiment 7: Dropped Packets

This experiment examined the effects of dropped packet bursts on echo cancellation performance. In this experiment multiple consecutive packet-sized segments of the far-End signal were replaced with silence (zero amplitude signal) to simulate a burst of

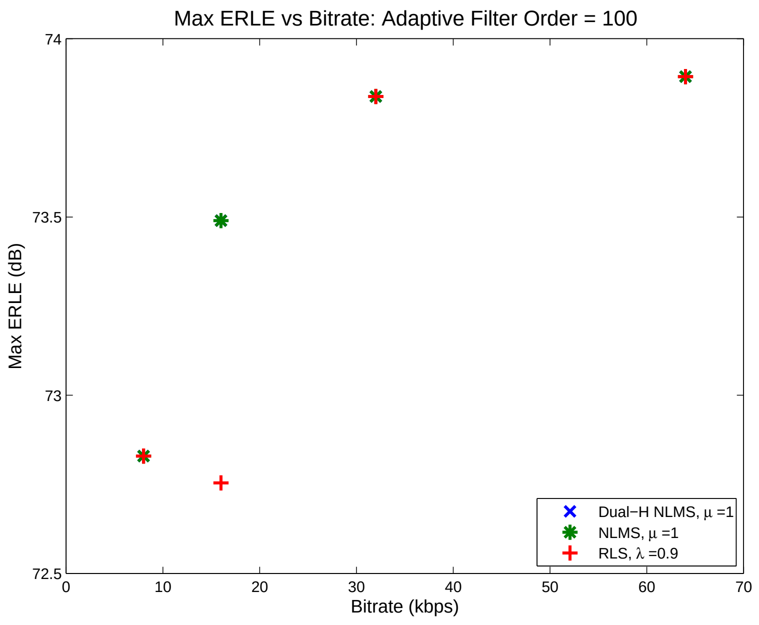

Figure 6.15: Experiment 6: Maximum ERLE vs Bit-rate for an adaptive filter order of 100

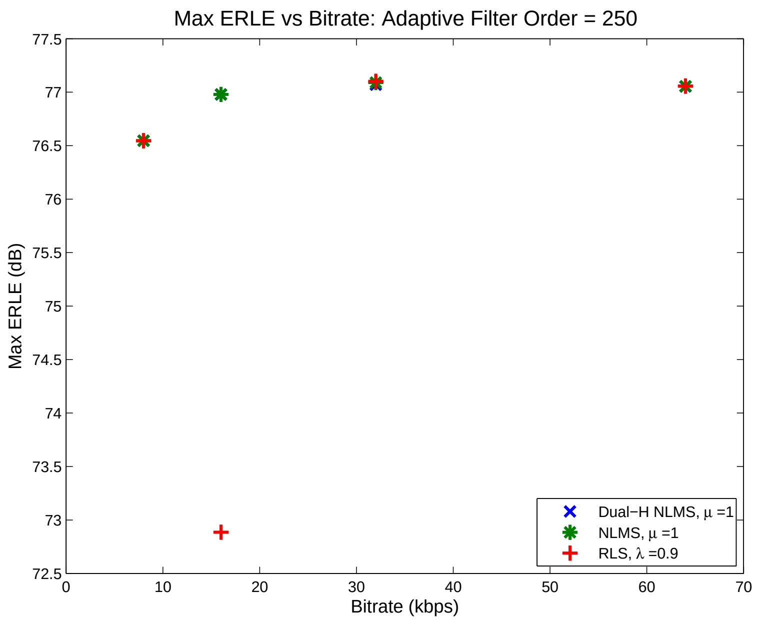

Figure 6.16: Experiment 6: Maximum ERLE vs Bit-rate for an adaptive filter order of 250

dropped packets which was repeated once every second. Figure 6.17 shows that the maximum ERLE of each adaptive algorithm is not effected by the dropped packets although the overall convergence time is increased. This is explained by Figure 6.18 which is a plot of ERLE vs sample number for an extreme case in which roughly 90% of the signal consists of dropped packet silence. It shows that the adaptive filters stop converging during the silence periods because $\boldsymbol{w}(n+1) = \boldsymbol{w}$ when $\boldsymbol{x}(n) = \mathbf{0}$ (see the adaptive filter update formulas in 3.4.1 and 3.5.1). During the periods in between the silence they are able to converge as normal and will eventually reach the optimal solution given enough time. In reality the slowing convergence times probably would not be observed since it requires such an extreme rate of drop packets to notice the effect. In this situation the degradation of the speech signal by the missing packets would make it indiscernible.

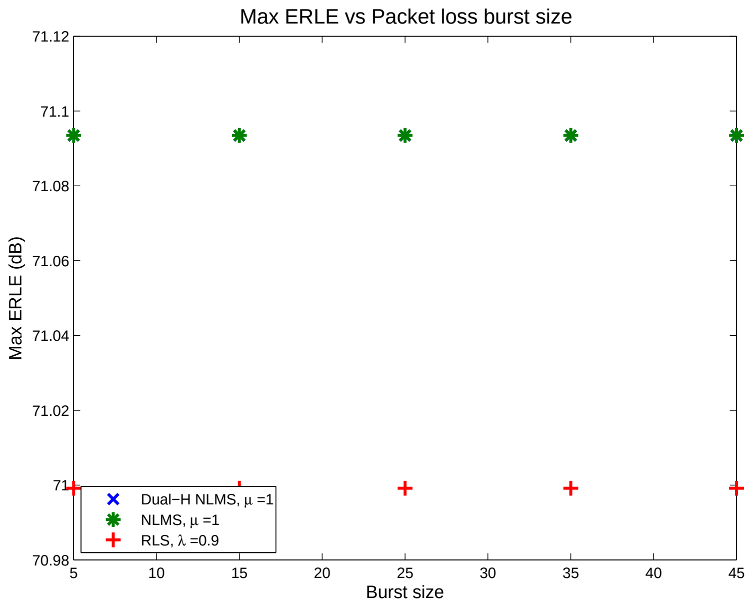

Figure 6.17: Experiment 7: Maximum ERLE vs Packet loss burst size for an adaptive filter order of 250

6.9 Experiment 8: Complete Simulation

During this experiment the simulation was run using a NLMS Dual-H filter under the following conditions:

- An adaptive filter order of 500
- A tail-circuit delay of 10 ms
- A dropped packet burst of 5 packets every second

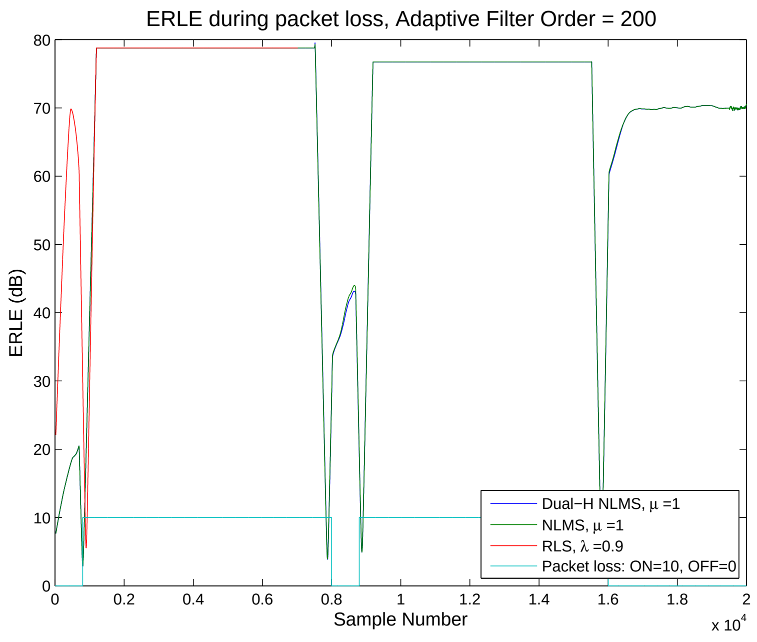

Figure 6.18: Experiment 7: ERLE plot for a large packet loss burst size and an adaptive filter order of 250

- The far-end speech signal was compressed/decompressed using the G.711 codec (64kbps)
- A near-end speech signal to produce double-talk
- A RIR filter of order 3050
- Gaussian white noise added to the near-end signal with a SNR of 35 dB

The error plot of the simulation is shown in Figure 6.19. It shows that whilst most of the echo was able to be removed, a moderate amount of residual echo still remained. This was confirmed during subjective listening tests of the residual error signal where the some background echo was still noticeable though the near-end speech was clearly distinguishable.

Figure 6.20 shows the ERLE plot during the simulation, note the erroneous peaks coinciding with the dropped packets. It shows that the NLMS Dual-H filter was able to achieve approximately 35 dB of ERLE during the simulation which is an adequate figure for an echo-canceller. For example if the echo on a phone line had an an ERL of 10 dB then after echo-cancellation the ERL becomes 10+35=45 dB. Referring to

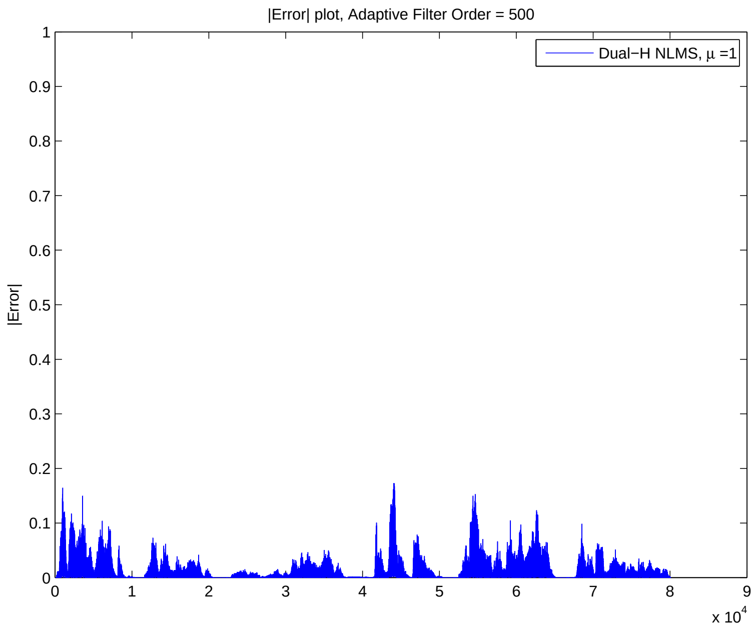

Figure 6.19: Experiment 8: Error plot for complete VoIP simulation.

Figure 2.3 this would be able to attenuate echoes to an acceptable level for one-way delays of up to 300 ms which is a more than adequate level of performance since above approximately 200 ms talker-overlap becomes a more annoying problem.

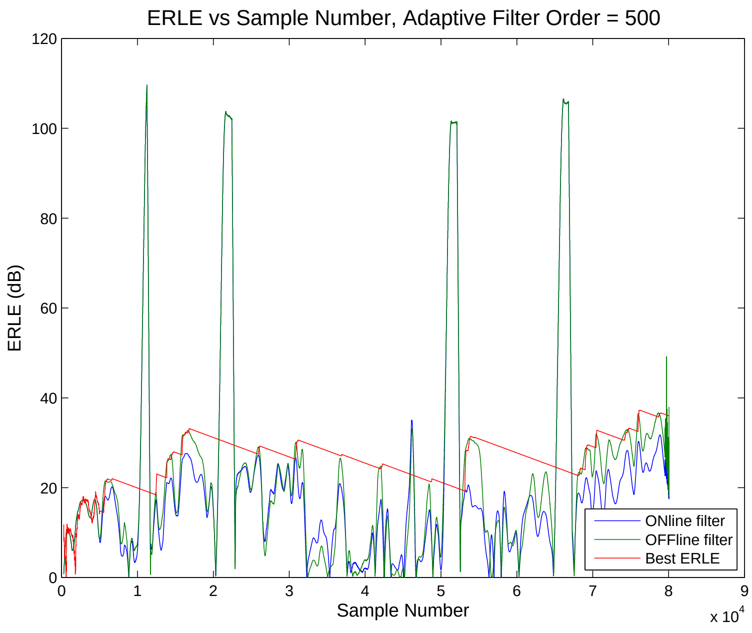

Figure 6.20: Experiment 8: ERLE plot for complete VoIP simulation.

6.10 Chapter Summary

The results of the VoIP AEC simulation experiments were presented as output plots which were then analysed and the effects of the various VoIP channel characteristics on echo cancellation performance was assessed. The results of the final experiment, which incorporated a combination of all of these characteristics, showed that a reasonable level of echo cancelling performance could be achieved using a Dual-H NLMS filter configuration. Table 6.1 shows a summary of the findings from the experiments:

Table 6.1: Summary of experiment results.

Experiment	Summary
1	Both NLMS and RLS showed increased performance and convergence times as the adaptive filter length was increased. RLS had markedly lower convergence times than NLMS.
2	Without a DTD double-talk is extremely detrimental to echo cancellation performance. The Dual-H filter configuration is very effective at AEC during double-talk conditions.
3	Performance decreases as the tail-circuit delay increases. Increasing the adaptive filter length increases performance when a delay was present.
4	Performance decreases as the room is made more echoic. Increasing the adaptive filter length is an effective way to increase performance in echoic conditions.
5	Both NLMS and RLS showed decreased performance when background noise was added. Increasing the adaptive filter length did not improve performance in these conditions.
6	Codec compression had negligible effect on performance due to the placement of the AEC system at the near-end gateway of the VoIP channel.
7	Increasing the rate of dropped packets had negligible effect on performance but increased convergence times. Although this effect is minimal using realistic packet loss rates.
8	The Dual-H NLMS filter design was able to cancel echoes effectively in the simulated VoIP environment.

Chapter 7

Conclusions and Further Work

7.1 Achievement of Project Objectives

The following objectives have been addressed:

Research the background AEC theory

The results of this research are shown in Chapter 2 which explains the nature of telephony echoes and describes an AEC system. The background literature on adaptive filter theory was also researched and used to give the mathematical derivations of the NLMS and RLS algorithms which are presented in Chapter 3.

Research the characteristics of VoIP channels

Evidence of this research is shown in Chapter 4 where the important characteristics of VoIP networks is presented as well as the likely impacts that these will have on AEC.

Design and implement AEC VoIP experiments

An AEC VoIP computer simulation was able to be successfully designed and implemented during this project. Chapter 5 describes the methodology used to design the simulation experiments and explains how the VoIP channel and room acoustics were modeled. It lists the various performance metrics used and explains how these were interpreted to assess effectiveness of each AEC system. The

MATLAB implementation is described in this chapter as well as in the program listing in Appendix B.

Analyse the results from these experiments

Chapter 6 gives a detailed discussion of the results of the simulations. They show that the simulation is capable of modeling the important characteristics of a VoIP channel as well as room acoustic effects. A summary of the findings from the simulations is given in Table 6.1. An important result of the project was that one of the AEC systems that was tested by simulation, the Dual-H NLMS system, was shown to be capable of adequate echo cancelling performance in a VoIP environment.

7.2 Further Work

Although all of the major objectives of the project were achieved successfully, time did not permit the completion of the optional objectives which were:

- Design and implement a real-time VoIP simulation environment.
- Investigate the use of more than one microphone for increasing echo cancellation performance.

Both of these would be good candidates for future research. In particular, a real-time simulation environment would be very useful and could be used to verify whether or not the computational and memory requirements of the AEC systems in this project are realistic.

Perhaps even more useful would be a real-world implementation rather than a simulation environment. This could be done either in software running on a PC or on a dedicated DSP board. This would be useful to verify the legitimacy of the VoIP channel and room acoustic models used in this project.

One concern I had during the project was that each of the various VoIP channel parameters that was implemented in the simulation had the effect of degrading the adaptive filter performance to some extent. Did the simulation account for all the important parameters? A real-world simulation incorporating a real VoIP channel and room acoustic effects would be the most thorough way to test each AEC system in this respect.

Lastly, more adaptive algorithms could be tested. This project only considered two different algorithms but there are a myriad of alternatives to choose from. Implementing a new algorithm could be achieved with a reduced effort by modifying the pre-existing MATLAB functions created for this project.

References

Albert, A. E. & Gardner, L. S. (1967), *Stochastic Approximation and Nonlinear Regression*, MIT Press.

Benetti, D., Damiani, E. & Houngue, P. (2008), Voip echo suppression in critical environments, *in* 'Digital Ecosystems and Technologies, 2008. DEST 2008. 2nd IEEE International Conference on', pp. 558 –562.

CISCO (2006), 'Understanding Delay in Packet Voice Networks', `http://www.cisco.com/en/US/tech/tk652/tk698/technologies_white_paper09186a00800a8993.shtml`. [Document ID: 5125].

Ding, L., El-Hennawey, M. & Goubran, R. (2006), 'Nonintrusive measurement of echo-path parameters in voip environments', *Instrumentation and Measurement, IEEE Transactions on* **55**(6), 2062 –2071.

Ditech Networks (2011), 'Echo Basics Tutorial', `http://www.ditechnetworks.com/learningCenter/echoBasics.html`. [Online; accessed May-2011].

Eriksson, A. & Karlsen, J. (2001), 'Patent No. US6,219,418 B1 Adaptive Dual Filter Echo Cancellation Method', *United States Patent* .

Farhang-Boroujeny, B. (1999), *Adaptive Filters, Theory and Applications*, Wiley.

Gansler, T., Gay, S., Sondhi, M. & Benesty, J. (2000), 'Double-talk robust fast converging algorithms for network echo cancellation', *Speech and Audio Processing, IEEE Transactions on* **8**(6), 656 –663.

Haykin, S. (1995), *Adaptive Filter Theory*, 3rd edn, Prentice Hall.

Huang, Y. & Goubran, R. (2000), Effects of vocoder distortion on network echo cancellation, *in* 'Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on', Vol. 1, pp. 437 –439 vol.1.

Isen, F. W. (2008*a*), *DSP for MATLAB and LabVIEW Volume I: Fundamentals of Discrete Signal Processing*, Synthesis Lectures on Signals Processing, Morgan and Claypool. Lecture 4.

Isen, F. W. (2008*b*), *DSP for MATLAB and LabVIEW Volume IV: LMS Adaptive Filtering*, Synthesis Lectures on Signals Processing, Morgan and Claypool. Lecture 7.

ITU-T (2003), 'ITU-T G.131: Control of talker echo', `http://www.itu.int/rec/T-REC-G/e`.

Leis, J. (2002), *Digital Signal Processing: A MATLAB-Based Tutorial Approach*, Research Studies Press LTD.

Nagumo, J. & Noda, A. (1967), 'A learning method for system identification', *IEEE Trans. Autom. Control* **AC-12**(4), 282–287.

Oppenheim, A. V. & Schafer, R. W. (1989), *Discrete-Time Signal Processing*, Prentice-Hall.

Periakarruppan, G., Low, A., Azhar, H. & Rashid, A. (2006), Packet based echo cancellation for voice over internet protocol simulated with variable amount of network delay time, *in* 'TENCON 2006. 2006 IEEE Region 10 Conference', pp. 1 –4.

Poularikas, A. D. & Ramadan, Z. M. (2006), *Adaptive Filtering Primer with MATLAB*, CRC Press.

Radecki, J., Zilic, Z. & Radecka, K. (2002), 'Echo Cancellation in IP Networks', *Proc. The 2002 45th Midwest Symposium on Circuits and Systems MWSCAS-2002* **2**, 219–222.

Sayed, A. (2008), *Adaptive Filters*, 1st edn, Wiley.

Sondhi, M. M. (1967), 'An adaptive echo canceller', *Bell Syst. Tech. J.* **46**(3).

Sondhi, M. M. & Berkley, D. A. (1980), 'Silencing echoes in the telephone network', *Proc. IEEE* **68**, 948–963.

Widrow, B. & Hoff, M. E. (1960), 'Adaptive switching circuits', *IRE WESCON Conv. Rec.* (4), 96–104.

Wiener, N. (1949), *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*, Wiley.

Appendix A

Project Specification

ENG 4111/2 Research Project

Project Specification

For:	**Shane Kmita**
Topic:	Echo cancellation in VoIP
Supervisors:	J. Leis
Sponsorship:	Faculty of Engineering
Project Aim:	To design and implement a digital adaptive filter system for the purpose of reducing echoes in VoIP transmissions.

Program:

1. Research the background information on echo cancellation using digital adaptive filters and the characteristics of voice over packet channels.
2. Simulate an IP channel with variable parameters such as bandwidth, latency, jitter and packet-loss.
3. Design and implement experiments to compare the effectiveness of various filter algorithms in cancelling echoes in a VoIP setting.
4. Analyse the results from these experiments to determine the most effective algorithms.

As time and resources permit:

1. Design and implement a real-time VoIP simulation environment for testing the echo cancelling system.
2. Investigate the use of more than one microphone for increasing echo cancellation performance.

Agreed:

Student Name:	Shane Kmita
Date:	23/03/2011
Supervisor Name:	John Leis
Date:	23/03/2011
Examiner/Co-Examiner:	
Date:	

Appendix B

AEC Simulation Source Code

B.1 The `aec_sim.m` MATLAB Script

`aec_sim.m` is the main script file for the AEC simulation. After running the script the user is prompted to select from one of the eight different AEC experiments to run. See 5.7.9 for user instructions and lists of input sound files and function files necessary for correct operation.

Listing B.1: The main script file for the AEC simulation.

```
% aec_sim.m
%
% * Adaptive Echo Cancellation over VoIP simulation script
%
% * Calls various AEC experiments based on the user input from command
%   prompt
%
% * Compares performance of the following adaptive algorithms:
%   NLMS
%   NLMS in dual-H configuration
%   RLS
%
% * Experiment designed as part of the final year engineering project 'Ec
%   Cancellation in VoIP' for ENG4111/2 University of Southern Queensland
%
% * LMS implementation adapted from adechosp.m by J.Leis
%
% * Room impulse response filter created using rir.m and fconv.m
%   (Copyright   2003 Stephen G. McGovern)
%
% * Requires the following function files in the current directory:
%   exp1_filtlen.m, exp2_dt.m, exp3_delay.m, exp4_longrir.m, exp5_noise.m
%   exp6_comp.m, exp7_drop_pack.m and exp8_combo.m
%
% * Requires the following input sound files in the working directory:
%   karl10s_mp2_8_dec.wav    karl10s_mp2_16_dec.wav,
%   karl10s_mp2_32_dec.wav   karl10s_mp2_64_dec.wav,
%   karl10s_8kHz_8bit.wav    ricky10s_8kHz_8bit.wav
%   karl10s_8kHz_8bit_mulaw.wav
%
% * Requires the following helper functions in the working directory:
%   rir.m, fconv.m and mtit.m
%
% Shane Kmita, Oct 2011

go = 1;

while go==1

    close all
    clc

    %-----------
    disp('##################################')
    disp('# AEC Simulator by Shane Kmita #')
    disp('################################')
    disp(' ')
    disp('Experiment #1 = Variable adaptive filter length')
    disp('Experiment #2 = Double talk condition')
    disp('Experiment #3 = Tail-circuit delay')
    disp('Experiment #4 = RIR filter')
    disp('Experiment #5 = Background noise')
    disp('Experiment #6 = Codec compression')
    disp('Experiment #7 = Dropped packet bursts')
    disp('Experiment #8 = Combined effect adverse conditions')
    disp('Experiment #0 = No experiment - quit simulator')
    disp(' ')
    experiment = input('Enter the experiment number you wish to run and p

    switch experiment
        case '1'
            clc
            disp('Experiment #1 = Variable adaptive filter length')
            disp('Please wait a minute while simulator is running...')
            exp1_filtlen()
        case '2'
            clc
            disp('Experiment #2 = Double talk condition')
            disp('Please wait a minute while simulator is running...')
            exp2_dt()
        case '3'
            clc
            disp('Experiment #3 = Effects of adding a tail-cicuit delay')
            disp('Please wait a minute while simulator is running...')
            exp3_delay()
        case '4'
            clc
            disp('Experiment #4 = Effects of changing the RIR')
            disp('A great choice, this is a good one!')
            disp('Please wait a minute while simulator is running...')
            exp4_longrir()
        case '5'
            clc
            disp('Experiment #5 = Effects of adding background noise')
            disp('Please wait a minute while simulator is running...')
            exp5_noise()
        case '6'
            clc
            disp('Experiment #6 = Effects of codec compression')
            disp('Please wait a minute while simulator is running...')
            exp6_comp()
        case '7'
            clc
            disp('Experiment #7 = Effects of dropped packet bursts')
            disp('Please wait a minute while simulator is running...')
            exp7_drop_pack()
        case '8'
            clc
            disp('Experiment #8 = Combined effect adverse conditions')
            disp('Please wait a minute while simulator is running...')
            exp8_combo()
        case '0'
```

```
            clc
            go = -1;
            disp('Quitting simulator - see you later!')
        otherwise
            fprintf( 1, 'Invalid input!\nPress a key to try again...\n');
            pause
    end
end
```

B.2 The exp1_filtlen.m MATLAB function

The function exp1_filtlen.m runs Experiment 1 (see 5.7.1) and is called by the main script file for the AEC simulation, aec_sim.m.

Listing B.2: The AEC Experiment 1 function file.

```
function [] = exp1_filtlen()
% exp1_filtlen.m
%
% * Experiment #1 = Variable adaptive filter length
%
% * Function file called by AEC simulator (aec_sim.m)
%
% * Creates plots of ERLE and Convergence time vs adaptive filter length
%
% * Compares performance of the following adaptive algorithms:
%   NLMS
%   NLMS in dual-H configuration
%   RLS
%
% * Experiment designed for of the final year engineering project 'Echo
%   Cancellation in VoIP' for ENG4111/2 University of Southern Queensland
%
% * Adapted from adechosp.m by J.Leis
%
% * Room impulse response filter created using rir.m and fconv.m
%   (Copyright   2003 Stephen G. McGovern)
%
% * The main script aec.sim.m requires the following input sound files in
%   the working directory:
%   karl10s_mp2_8_dec.wav    karl10s_mp2_16_dec.wav,
%   karl10s_mp2_32_dec.wav   karl10s_mp2_64_dec.wav,
%   karl10s_8kHz_8bit.wav    ricky10s_8kHz_8bit.wav
%   karl10s_8kHz_8bit_mulaw.wav
%
% * Requires the following helper functions in the working directory:
%   rir.m, fconv.m and mtit.m
%
% Shane Kmita, Oct 2011

%-------------------------------------------------------------------------
% Input sound vectors
%-------------------------------------------------------------------------
% Far-End (fe) signal
N = 80000; Fs = 8000; x = randn(N,1);      % 10s of Gaussian white noise

% Near-End (ne) signal
s = zeros(length(x),1);                    % no ne speech

% Level shift to 80% maximum
x = 0.8*x/(max(abs(x)));

N = min(length(x), length(s));
x = x(1:N);
```

```
s = s(1:N);

% vector of adaptive filter lengths
test_length = [50, 100, 200];

for next = 1:length(test_length)

    L = test_length(next);
    %-------------------------------------------------------------------------
    % Create ne room impulse response
    %-------------------------------------------------------------------------

    rm=[3 3 3];           % room dimensions [L W H] in metres
    mic=[2.5 4 0.9];      % mic position
    src=[2.5 4 1.9];      % source position
    r=-0.5;                % reflection coefficient (-1<r<1)
    n=24;
    b=rir(Fs, mic, n, r, rm, src);
    b = b(1:L);          % truncate impulse response to adfilter length

    %-------------------------------------------------------------------------
    % Echo delay ne signal
    %-------------------------------------------------------------------------

    d = zeros(N,1);       % echo delayed fe signal, (*not* observable)
    for k = 1:N
       for i = 0:length(b)-1
           if k-i > 0
               d(k) = d(k) + b(i+1)*x(k-i);
           end
       end
    end

    r = s + d;                           % ne signal + fe echo (observable)

    %-------------------------------------------------------------------------
    % Initialise variables / set parameters
    %-------------------------------------------------------------------------
    % Dual-H NLMS
    mu = 1;                    % NLMS step size
    wON = zeros(L, 1);         % ONline adaptive filter weights
    wOFF = zeros(L, 1);        % OFFline adaptive filter weights
    yON = zeros(N,1);          % ONline adaptive filter output
    yOFF = zeros(N,1);         % OFFline adaptive filter otput
    eON  = zeros(1, N);        % ONline residual error
    eOFF  = zeros(1, N);       % OFFline residual error
    delta = 0.000001;          % NLMS/ERLE constant to avoid division by 0

    % NLMS
    mu2 = 1;                   % NLMS step size
    wNLMS = zeros(L, 1);       % NLMS adaptive filter weights
    yNLMS = zeros(N,1);        % NLMS adaptive filter otput (estimate of d)
    eNLMS  = zeros(1, N);      % NLMS residual error

    % RLS
    lambda = 0.9;              % RLS forgetting factor
    % RLS variables
    wRLS = zeros(L, 1);        % RLS filter weights
    x_filter = zeros(L, 1);    % input signal in filter
    P = eye(L);                % inverse input correlation matrix
    int= zeros(L, 1);          % intermediate calculation step = P(n-1)*x(n)
```

```
gain = zeros(L, 1);       % gain vector
yRLS = zeros(N,1);        % RLS adaptive filter output
eRLS  = zeros(1, N);      % RLS residual error signal

% ERLE variables
ERLE_L = L;               % order of ERLE calculation vectors
ERLE_ONdh = 0;            % current ONline filter (ERLE estimate)
ERLE_OFFdh = 0;           % current OFFline filter (ERLE estimate)
dp = 0;                   % power of d in ERLE vector
xp = 0;                   % power of x in ERLE vector
epON = 0;                 % power of eON in ERLE vector
epOFF = 0;                % power of eOFF in ERLE vector
epNLMS = 0;               % power of eNLMS in ERLE vector
epRLS = 0;                % power of eRLS in ERLE vector
ERLEdh_best = 0;          % best ERLE (dual-H estimate) found so far

% Plotting variables - otherwise not necessary for simulation
WON = zeros(L, N);        % saves the ONline adaptive weights to plot
WOFF = zeros(L, N);       % saves the OFFline adaptive weights to plot
ERLE_ON = zeros(N,1);     % saves the ONline ERLE for plotting
ERLE_OFF = zeros(N,1);    % saves the OFFline ERLE for plotting
ERLE_NLMS = zeros(N,1);   % saves the NLMS ERLE for plotting
ERLE_RLS = zeros(N,1);    % saves the RLS ERLE for plotting
ERLEdh_best_plot = zeros(N,1);   % saves the current best ERLE to plot
dt_plot = zeros(N,1);     % double-talk flag (0 if fe only, 1 if dt)
%-----------------------------------------------------------------

%-----------------------------------------------------------------
% AEC Simulation
%-----------------------------------------------------------------

for k = 1:N
    %
    % RLS
    %
    x_filter(1)=x(k);
    int = P*x_filter;

    % 1. calculate gain vector
    gain = (1/(lambda+dot(x_filter, int)))*int;

    % 2. calculate estimation error
    yRLS(k) = 0; % adaptive filter output
    for n = 0:L-1
        if( (k-n) > 0)
            yRLS(k) = yRLS(k) + (wRLS(n+1) * x(k-n));
        end
    end
    eRLS(k) = r(k)-yRLS(k);

    % 3. update filter weights
    wRLS = wRLS + gain*eRLS(k);

    % 4. update inverse matrix
    P = (1/lambda)*(P - gain*((x_filter')*P));

    % get next input vector
    for j=L:-1:2
        x_filter(j)=x_filter(j-1);
    end
```

```
%
% NLMS Dual-H, NLMS
%

% calculate the ONline and OFFline filter output:
yON(k) = 0;
yOFF(k) = 0;
yNLMS(k) = 0;
for n = 0:L-1
    if( (k-n) > 0)
        yOFF(k) = yOFF(k) + (wOFF(n+1) * x(k-n));
        yNLMS(k) = yNLMS(k) + (wNLMS(n+1) * x(k-n));
        yON(k) = yON(k) + (wON(n+1) * x(k-n));
    end
end

% calculate the ONline and OFFline error:
% error sig = r(sig+echo) - y(est echo)
% = est of sig
eON(k) = r(k) - yON(k);
eOFF(k) = r(k) - yOFF(k);
eNLMS(k) = r(k) - yNLMS(k);

% find power of d and e in ERLE buffer(for ERLE calculation below
% adds next value and subtracts last value rather than recomputing
% whole buffer each time
if k > ERLE_L
    xp = xp + x(k)*x(k) - x(k-ERLE_L)*x(k-ERLE_L);
    dp = dp + d(k)*d(k) - d(k-ERLE_L)*d(k-ERLE_L);
    epON = epON + eON(k)*eON(k) - eON(k-ERLE_L)*eON(k-ERLE_L);
    epOFF = epOFF + eOFF(k)*eOFF(k) - eOFF(k-ERLE_L)*eOFF(k-ERLE_
    epNLMS = epNLMS + eNLMS(k)*eNLMS(k) - eNLMS(k-ERLE_L)*eNLMS(k-
    epRLS = epRLS + eRLS(k)*eRLS(k) - eRLS(k-ERLE_L)*eRLS(k-ERLE_
else
    xp = xp + x(k)*x(k);
    dp = dp + d(k)*d(k);
    epON = epON + eON(k)*eON(k);
    epOFF = epOFF + eOFF(k)*eOFF(k);
    epNLMS = epNLMS + eNLMS(k)*eNLMS(k);
    epRLS = epRLS + eRLS(k)*eRLS(k);
end

% calculate true ERLE (dB) for plotting
ERLE_ON(k) = 10*log10(dp / (epON - s(k) + delta));
ERLE_OFF(k) = 10*log10(dp / (epOFF - s(k) + delta));
ERLE_NLMS(k) = 10*log10(dp / (epNLMS - s(k) + delta));
ERLE_RLS(k) = 10*log10(dp / (epRLS - s(k) + delta));

% calculate estimated ERLE (dB) for dual-H operation
ERLE_ONdh = 10*log10(dp / (epON + delta));
ERLE_OFFdh = 10*log10(dp / (epOFF + delta));

% update the OFFline adaptive filter coeff (Dual-H NLMS, NLMS)
for n = 0:L-1
    if( (k-n) > 0)
        wOFF(n+1) = wOFF(n+1) + mu / (xp + delta)*eOFF(k)*x(k-n);
        wNLMS(n+1) = wNLMS(n+1) + mu2 / (xp + delta)*eNLMS(k)*x(k-
    end
end
```

```
        % update the ONline filter coeff with the OFFline coeff if the
        % OFFline ERLE is larger than the best ERLE found so far
        if (k < 10*L)
            wON = wOFF;
            ERLEdh_best = ERLE_OFFdh;
        elseif (ERLE_OFFdh >= ERLEdh_best)
            wON = wOFF;
            ERLEdh_best = ERLE_OFFdh;
        end

        % update best ERLE if ONline ERLE is larger
        if (ERLE_ONdh >= ERLEdh_best) && (k>ERLE_L)
            ERLEdh_best = ERLE_ONdh;
        end

        % update the OFFline filter coeff with the ONline coeff if the
        % ONline ERLE is larger than the OFFline ERLE
        if (ERLE_ONdh > (ERLE_OFFdh+3)) && (k>ERLE_L)
            wOFF = wON;
        end

        % update best ERLE plotting variable
        ERLEdh_best_plot(k) = ERLEdh_best;

        % reduce ERLEdh_best at a rate of 5dB/s
        ERLEdh_best = ERLEdh_best - 5/Fs;

        % update plotting variables
        WON(:,k) = wON;
        WOFF(:,k) = wOFF;

        if (abs(d(k)) > delta) && (abs(s(k)) > delta)  % double talk
            dt_plot(k) = 20;
        else              % not double talk
            dt_plot(k) = 0;
        end
    end

    %
    % calculate plotting data
    %

    % Dual-H NLMS output
    Len(next) = L;                              % current filter length
    plotmaxDH(next) = max(ERLE_ON); % find dual-H NLMS max ERLE
    convDH = find(ERLE_ON(L:length(ERLE_ON)) > 0.9*plotmaxDH(next)); % fi
                                                        % ERLE > 90% max ERLE
    conv_timeDH(next) = convDH(1)/Fs*1000;              % finds the time to 9


    % NLMS output
    plotmaxNLMS(next) = max(ERLE_NLMS); % find NLMS max ERLE
    convNLMS = find(ERLE_NLMS(L:length(ERLE_NLMS)) > 0.9*plotmaxNLMS(next
                                                        % ERLE > 90% max ERLE
    conv_timeNLMS(next) = convNLMS(1)/Fs*1000;              % finds the time

    % RLS output
    plotmaxRLS(next) = max(ERLE_RLS); % find RLS max ERLE
    convRLS = find(ERLE_RLS(L:length(ERLE_RLS)) > 0.9*plotmaxRLS(next)); 
                                                        % ERLE > 90% max ERLE
    conv_timeRLS(next) = convRLS(1)/Fs*1000;             % finds the time to
```

```
end

%
% output plots
%
figure(1)
semilogx(Len,plotmaxDH,'x',Len,plotmaxNLMS,'*',Len,plotmaxRLS,'+','LineWi
                    'MarkerSize',10);
title('Maximum␣ERLE␣vs␣Filter␣Length','fontsize', 14);
xlabel('Adaptive␣Filter␣Length','fontsize', 12);
ylabel('Maximum␣ERLE␣(dB)','fontsize', 12);
legend(['Dual−H␣NLMS,␣\mu␣=',num2str(mu)],['NLMS,␣\mu␣=',num2str(mu2)], ..
       ['RLS,␣\lambda␣=',num2str(lambda)],2);

figure(2)
plot(Len,conv_timeDH,'x',Len,conv_timeNLMS,'*',Len,conv_timeRLS,'+','LineV
                    'MarkerSize',10);
title('Convergence␣time␣vs␣Filter␣Length','fontsize', 14);
xlabel('Adaptive␣Filter␣Length','fontsize', 12);
ylabel('Convergence␣time␣(ms)','fontsize', 12);
legend(['Dual−H␣NLMS,␣\mu␣=',num2str(mu)],['NLMS,␣\mu␣=',num2str(mu2)], ..
       ['RLS,␣\lambda␣=',num2str(lambda)],2);

disp('##########################################################################
disp('Experiment␣finished!')
fprintf( 1, 'Save␣figures␣if␣necessary␣then␣press␣ENTER␣to␣run␣a␣new␣exper
pause
```

B.3 The exp2_dt.m MATLAB function

The function `exp2_dt.m` runs Experiment 2 (see 5.7.2) and is called by the main script file for the AEC simulation, `aec_sim.m`.

Listing B.3: The AEC Experiment 2 function file..

```
function [] = exp2_dt()
% exp2_dt.m
%
% * Experiment #2 = Double talk condition
%
% * Function file called by AEC simulator (aec_sim.m)
%
% * Creates plots of ERLE and |Error| for a range of adaptive filter
%   lengths
%
% * Compares performance of the following adaptive algorithms:
%   NLMS
%   NLMS in dual−H configuration
%   RLS
%
% * Experiment designed as part of the final year engineering project 'Ec
%   Cancellation in VoIP' for ENG4111/2 University of Southern Queensland
%
% * Adapted from adechosp.m by J.Leis
%
% * Room impulse response filter created using rir.m and fconv.m
%   (Copyright  2003 Stephen G. McGovern)
%
% * The main script aec_sim.m requires the following input sound files in
%   the working directory:
%   karl10s_mp2_8_dec.wav   karl10s_mp2_16_dec.wav,
%   karl10s_mp2_32_dec.wav  karl10s_mp2_64_dec.wav,
%   karl10s_8kHz_8bit.wav   ricky10s_8kHz_8bit.wav
%
% * Requires the following helper functions in the working directory:
%   rir.m, fconv.m and mtit.m
%
% Shane Kmita, Oct 2011

%--------------------------------------------------------------------------
% Input sound vectors
%--------------------------------------------------------------------------
% Far-End (fe) signal
[x Fs] = wavread('Test_signals\karl10s_8kHz_8bit.wav');

% Near-End (ne) signal
[s Fs] = wavread('Test_signals\ricky10s_8kHz_8bit.wav');

% Level shift to 80% maximum
x = 0.8*x/(max(abs(x)));
s = 0.8*s/(max(abs(s)));

N = min(length(x), length(s));
x = x(1:N);
s = s(1:N);

% vector of adaptive filter lengths
test_length = [100, 200, 250];   % adaptive filter order

for next = 1:length(test_length)

    L = test_length(next);
    %----------------------------------------------------------------------
    % Create ne room impulse response
    %----------------------------------------------------------------------

    rm=[3 3 3];           % room dimensions [L W H] in metres
    mic=[2.5 4 0.9];      % mic position
    src=[2.5 4 1.9];      % source position
    r=-0.5;                % reflection coefficient (-1<r<1)
    n=24;
    b=rir(Fs, mic, n, r, rm, src);
    b = b(1:L);          % truncate impulse response to adfilter length

    %----------------------------------------------------------------------
    % Echo delay ne signal
    %----------------------------------------------------------------------

    d = zeros(N,1);       % echo delayed fe signal, (*not* observable)
    for k = 1:N
       for i = 0:length(b)-1
           if k-i > 0
```

```
            d(k) = d(k) + b(i+1)*x(k-i);
        end
    end
end

r = s + d;                          % ne signal + fe echo (observable)

%--------------------------------------------------------------------------
% Initialise variables / set parameters
%--------------------------------------------------------------------------
% Dual-H NLMS
mu = 1;                     % NLMS step size
wON = zeros(L, 1);          % ONline adaptive filter weights
wOFF = zeros(L, 1);         % OFFline adaptive filter weights
yON = zeros(N,1);           % ONline adaptive filter output
yOFF = zeros(N,1);          % OFFline adaptive filter otput
eON  = zeros(1, N);         % ONline residual error
eOFF  = zeros(1, N);        % OFFline residual error
delta = 0.000001;           % NLMS/ERLE constant to avoid division by 0

% NLMS
mu2 = 1;                    % NLMS step size
wNLMS = zeros(L, 1);        % NLMS adaptive filter weights
yNLMS = zeros(N,1);         % NLMS adaptive filter otput (estimate of d)
eNLMS  = zeros(1, N);       % NLMS residual error

% RLS
lambda = 0.9;               % RLS forgetting factor
% RLS variables
wRLS = zeros(L, 1);         % RLS filter weights
x_filter = zeros(L, 1);     % input signal in filter
P = eye(L);                 % inverse input correlation matrix
int= zeros(L, 1);           % intermediate calculation step = P(n-1)*x(n)
gain = zeros(L, 1);         % gain vector
yRLS = zeros(N,1);          % RLS adaptive filter output
eRLS  = zeros(1, N);        % RLS residual error signal

% ERLE variables
ERLE_L = L;                 % order of ERLE calculation vectors
ERLE_ONdh = 0;              % current ONline filter (ERLE estimate)
ERLE_OFFdh = 0;             % current OFFline filter (ERLE estimate)
dp = 0;                     % power of d in ERLE vector
xp = 0;                     % power of x in ERLE vector
epON = 0;                   % power of eON in ERLE vector
epOFF = 0;                  % power of eOFF in ERLE vector
epNLMS = 0;                 % power of eNLMS in ERLE vector
epRLS = 0;                  % power of eRLS in ERLE vector
ERLEdh_best = 0;            % best ERLE (dual-H estimate) found so far

% Plotting variables - otherwise not necessary for simulation
WON = zeros(L, N);          % saves the ONline adaptive weights for plotti
WOFF = zeros(L, N);         % saves the OFFline adaptive weights for ploti
ERLE_ON = zeros(N,1);       % saves the ONline ERLE for plotting
ERLE_OFF = zeros(N,1);      % saves the OFFline ERLE for plotting
ERLE_NLMS = zeros(N,1);     % saves the NLMS ERLE for plotting
ERLE_RLS = zeros(N,1);      % saves the RLS ERLE for plotting
ERLEdh_best_plot = zeros(N,1);  % saves the current best ERLE for plo
dt_plot = zeros(N,1);       % double-talk flag (0 if fe only, 1 if dt)
%--------------------------------------------------------------------------
```

```
%------------------------------------------------------------------------
% AEC Simulation
%------------------------------------------------------------------------

for k = 1:N
    %
    % RLS
    %
    x_filter(1)=x(k);
    int = P*x_filter;

    % 1. calculate gain vector
    gain = (1/(lambda+dot(x_filter, int)))*int;

    % 2. calculate estimation error
    yRLS(k) = 0; % adaptive filter output
    for n = 0:L-1
        if( (k-n) > 0)
            yRLS(k) = yRLS(k) + (wRLS(n+1) * x(k-n));
        end
    end
    eRLS(k) = r(k)-yRLS(k);

    % 3. update filter weights
    wRLS = wRLS + gain*eRLS(k);

    % 4. update inverse matrix
    P = (1/lambda)*(P - gain*((x_filter')*P));

    % get next input vector
    for j=L:-1:2
        x_filter(j)=x_filter(j-1);
    end

    %
    % NLMS Dual-H, NLMS
    %

    % calculate the ONline and OFFline filter output:
    yON(k) = 0;
    yOFF(k) = 0;
    yNLMS(k) = 0;
    for n = 0:L-1
        if( (k-n) > 0)
            yOFF(k) = yOFF(k) + (wOFF(n+1) * x(k-n));
            yNLMS(k) = yNLMS(k) + (wNLMS(n+1) * x(k-n));
            yON(k) = yON(k) + (wON(n+1) * x(k-n));
        end
    end

    % calculate the ONline and OFFline error:
    % error sig = r(sig+echo) - y(est echo)
    % = est of sig
    eON(k) = r(k) - yON(k);
    eOFF(k) = r(k) - yOFF(k);
    eNLMS(k) = r(k) - yNLMS(k);

    % find power of d and e in ERLE buffer(for ERLE calculation below
    % adds next value and subtracts last value rather than recomputing
    % whole buffer each time
    if k > ERLE_L
```

```
        xp = xp + x(k)*x(k) - x(k-ERLE_L)*x(k-ERLE_L);
        dp = dp + d(k)*d(k) - d(k-ERLE_L)*d(k-ERLE_L);
        epON = epON + eON(k)*eON(k) - eON(k-ERLE_L)*eON(k-ERLE_L);
        epOFF = epOFF + eOFF(k)*eOFF(k) - eOFF(k-ERLE_L)*eOFF(k-ERLE_I
        epNLMS = epNLMS + eNLMS(k)*eNLMS(k) - eNLMS(k-ERLE_L)*eNLMS(k-
        epRLS = epRLS + eRLS(k)*eRLS(k) - eRLS(k-ERLE_L)*eRLS(k-ERLE_I
    else
        xp = xp + x(k)*x(k);
        dp = dp + d(k)*d(k);
        epON = epON + eON(k)*eON(k);
        epOFF = epOFF + eOFF(k)*eOFF(k);
        epNLMS = epNLMS + eNLMS(k)*eNLMS(k);
        epRLS = epRLS + eRLS(k)*eRLS(k);
    end

    % calculate true ERLE (dB) for plotting
    ERLE_ON(k) = 10*log10(dp / (epON - s(k) + delta));
    ERLE_OFF(k) = 10*log10(dp / (epOFF - s(k) + delta));
    ERLE_NLMS(k) = 10*log10(dp / (epNLMS - s(k) + delta));
    ERLE_RLS(k) = 10*log10(dp / (epRLS - s(k) + delta));

    % calculate estimated ERLE (dB) for dual-H operation
    ERLE_ONdh = 10*log10(dp / (epON + delta));
    ERLE_OFFdh = 10*log10(dp / (epOFF + delta));

    % update the OFFline adaptive filter coeff (Dual-H NLMS, NLMS)
    for n = 0:L-1
        if( (k-n) > 0)
            wOFF(n+1) = wOFF(n+1) + mu / (xp + delta)*eOFF(k)*x(k-n);
            wNLMS(n+1) = wNLMS(n+1) + mu2 / (xp + delta)*eNLMS(k)*x(k-
        end
    end

    % update the ONline filter coeff with the OFFline coeff if the
    % OFFline ERLE is larger than the best ERLE found so far
    if (k < 10*L)
        wON = wOFF;
        ERLEdh_best = ERLE_OFFdh;
    elseif (ERLE_OFFdh >= ERLEdh_best)
        wON = wOFF;
        ERLEdh_best = ERLE_OFFdh;
    end

    % update best ERLE if ONline ERLE is larger
    if (ERLE_ONdh >= ERLEdh_best) && (k>ERLE_L)
        ERLEdh_best = ERLE_ONdh;
    end

    % update the OFFline filter coeff with the ONline coeff if the
    % ONline ERLE is larger than the OFFline ERLE
    if (ERLE_ONdh > (ERLE_OFFdh+3)) && (k>ERLE_L)
        wOFF = wON;
    end

    % update best ERLE plotting variable
    ERLEdh_best_plot(k) = ERLEdh_best;

    % reduce ERLEdh_best at a rate of 5dB/s
    ERLEdh_best = ERLEdh_best - 5/Fs;

    % update plotting variables
```

```
        WON(:,k) = wON;
        WOFF(:,k) = wOFF;

        if (abs(d(k)) > delta) && (abs(s(k)) > delta)  % double talk
            dt_plot(k) = 20;
        else            % not double talk
            dt_plot(k) = 0;
        end
    end
    %-------------------------------------------------------------------------------

    % smooth output using moving average
    for i=1:N-500
        ERLE_ON(i) = mean(ERLE_ON(i:i+500));
        ERLE_NLMS(i) = mean(ERLE_NLMS(i:i+500));
        ERLE_RLS(i) = mean(ERLE_RLS(i:i+500));
    end

    %-------------------------------------------------------------------------------
    figure(next)
    subplot(4,1,1);
    plot(dt_plot/20);
    ylim([0 2])
    ylabel('Double␣talk␣flag');
    legend('Double␣talk:␣ON␣=␣1,␣OFF␣=␣0',1);
    %-------------------------------------------------------------------------------
    %-------------------------------------------------------------------------------
    subplot(4,1,2);
    plot(abs(eON - s'));
    ylim([0 1])
    ylabel('|Error|');
    legend(['Dual-H␣NLMS,␣\mu␣=',num2str(mu)],1);
    %-------------------------------------------------------------------------------
    subplot(4,1,3);
    plot(abs(eNLMS - s'));
    ylim([0 1])
    ylabel('|Error|');
    legend(['NLMS,␣\mu␣=',num2str(mu2)],1);
    %-------------------------------------------------------------------------------
    %-------------------------------------------------------------------------------
    subplot(4,1,4);
    plot(abs(eRLS - s'));
    ylim([0 1])
    ylabel('|Error|');
    xlabel('Sample␣Number');
    legend(['RLS,␣\lambda␣=',num2str(lambda)],1);

    % Put a main title above the subplot titles
        main_title = ['|Error|␣during␣Double␣talk,␣Adaptive␣Filter␣Order␣=
        mtit(main_title, 'xoff',-.1,'yoff',.025, 'fontsize', 14);

    figure(next+length(test_length))
    plot(1:N, [abs(ERLE_ON), abs(ERLE_NLMS), abs(ERLE_RLS), dt_plot]);
    title(['ERLE␣during␣Double␣talk,␣Adaptive␣Filter␣Order␣=␣',num2str(L)]
    xlabel('Sample␣Number','fontsize', 12);
    ylabel('ERLE␣(dB)','fontsize', 12);
    legend(['Dual-H␣NLMS,␣\mu␣=',num2str(mu)],['NLMS,␣\mu␣=',num2str(mu2)]
      ['RLS,␣\lambda␣=',num2str(lambda)],'Double␣talk:␣ON=20,␣OFF=0',2);
```

```
end
disp('################################################################################
disp('Experiment finished!')
fprintf( 1, 'Save figures if necessary then press ENTER to run a new exper
pause
```

B.4 The exp3_delay.m MATLAB function

The function exp3_delay.m runs Experiment 3 (see 5.7.3) and is called by the main script file for the AEC simulation, aec_sim.m.

Listing B.4: The AEC Experiment 3 function file..

```
function [] = exp3_delay()
% exp3_delay.m
%
% * Experiment #3 = Effects of round-trip delay
%
% * Function file called by AEC simulator (aec_sim.m)
%
% * Creates plots of |Error| for various values one-way network delay
%   and adaptive filter length. Also plots Max ERLE vs delay.
%
% * Compares performance of the following adaptive algorithms:
%   NLMS
%   NLMS in dual-H configuration
%   RLS
%
% * Experiment designed as part of the final year engineering project 'Ec
%   Cancellation in VoIP' for ENG4111/2 University of Southern Queensland
%
% * Adapted from adechosp.m by J.Leis
%
% * Room impulse response filter created using rir.m and fconv.m
%   (Copyright  2003 Stephen G. McGovern)
%
% * The main script aec.sim.m requires the following input sound files in
%   the working directory:
%   karl10s_mp2_8_dec.wav    karl10s_mp2_16_dec.wav,
%   karl10s_mp2_32_dec.wav   karl10s_mp2_64_dec.wav,
%   karl10s_8kHz_8bit.wav    ricky10s_8kHz_8bit.wav
%   karl10s_8kHz_8bit_mulaw.wav
%
% * Requires the following helper functions in the working directory:
%   rir.m, fconv.m and mtit.m
%
% Shane Kmita, Oct 2011

%------------------------------------------------------------------------
% Input sound vectors
```

```
%------------------------------------------------------------------------
% Far-End (fe) signal
N = 20000; Fs = 8000; x = randn(N,1);     % N Gaussian white noise samples

% Near-End (ne) signal
s = zeros(N,1);                           % no ne speech

% Level shift fe to 80% maximum
x = 0.8*x/(max(abs(x)));

%------------------------------------------------------------------------
% Create ne room impulse response
%------------------------------------------------------------------------

rm=[3 3 3];            % room dimensions [L W H] in metres
mic=[2.5 4 0.9];       % mic position
src=[2.5 4 1.9];       % source position
r=-0.5;                 % reflection coefficient (-1<r<1)
n=24;
b1=rir(Fs, mic, n, r, rm, src);

% vector of adaptive filter lengths
test_length = [100, 200, 250];   % adaptive filter order

% vector of one-way delays
tdelay_range = [5, 10, 15]; % one-way tail circuit delay (ms)

%------------------------------------------------------------------------
% AEC Simulation
%------------------------------------------------------------------------

for next = 1:length(test_length)

    L = test_length(next);


    for tdelay_next = 1:length(tdelay_range)

        tdelay = tdelay_range(tdelay_next);        % delay in ms
        tdelay_samp = floor(tdelay*Fs/1000);       % delay in samples
        b = [zeros(tdelay_samp,1); b1];              % add delay to rir
        %b = b(1:L);          % truncate impulse response to adfilter lengt

        %--------------------------------------------------------------------
        % Echo delay ne signal
        %--------------------------------------------------------------------

        d = zeros(N,1);       % echo delayed fe signal, (*not* observable)
        for k = 1:N
           for i = 0:length(b)-1
                if k-i > 0
                    d(k) = d(k) + b(i+1)*x(k-i);
                end
            end
        end

        r = s + d;                          % ne signal + fe echo (observable

        %--------------------------------------------------------------------
        % Initialise variables / set parameters
        %--------------------------------------------------------------------
        % Dual-H NLMS
        mu = 1;                        % NLMS step size
```

```
wON = zeros(L, 1);          % ONline adaptive filter weights
wOFF = zeros(L, 1);         % OFFline adaptive filter weights
yON = zeros(N,1);           % ONline adaptive filter output
yOFF = zeros(N,1);          % OFFline adaptive filter otput
eON  = zeros(1, N);         % ONline residual error
eOFF  = zeros(1, N);        % OFFline residual error
delta = 0.000001;           % NLMS/ERLE constant to avoid division by

% NLMS
mu2 = 1;                    % NLMS step size
wNLMS = zeros(L, 1);        % NLMS adaptive filter weights
yNLMS = zeros(N,1);         % NLMS adaptive filter otput (estimate of
eNLMS  = zeros(1, N);       % NLMS residual error

% RLS
lambda = 0.9;               % RLS forgetting factor
% RLS variables
wRLS = zeros(L, 1);         % RLS filter weights
x_filter = zeros(L, 1);     % input signal in filter
P = eye(L);                 % inverse input correlation matrix
int= zeros(L, 1);           % intermediate calculation step = P(n-1)*
gain = zeros(L, 1);         % gain vector
yRLS = zeros(N,1);          % RLS adaptive filter output
eRLS  = zeros(1, N);        % RLS residual error signal

% ERLE variables
ERLE_L = L;                 % order of ERLE calculation vectors
ERLE_ONdh = 0;              % current ONline filter (ERLE estimate)
ERLE_OFFdh = 0;             % current OFFline filter (ERLE estimate)
dp = 0;                     % power of d in ERLE vector
xp = 0;                     % power of x in ERLE vector
epON = 0;                   % power of eON in ERLE vector
epOFF = 0;                  % power of eOFF in ERLE vector
epNLMS = 0;                 % power of eNLMS in ERLE vector
epRLS = 0;                  % power of eRLS in ERLE vector
ERLEdh_best = 0;            % best ERLE (dual-H estimate) found so fa

% Plotting variables - otherwise not necessary for simulation
WON = zeros(L, N);          % saves the ONline adaptive weights for p
WOFF = zeros(L, N);         % saves the OFFline adaptive weights for
ERLE_ON = zeros(N,1);       % saves the ONline ERLE for plotting
ERLE_OFF = zeros(N,1);      % saves the OFFline ERLE for plotting
ERLE_NLMS = zeros(N,1);     % saves the NLMS ERLE for plotting
ERLE_RLS = zeros(N,1);      % saves the RLS ERLE for plotting
ERLEdh_best_plot = zeros(N,1);  % saves the current best ERLE for
dt_plot = zeros(N,1);       % double-talk flag (0 if fe only, 1 if dt
%------------------------------------------------------------------------

for k = 1:N
    %
    % RLS
    %
    x_filter(1)=x(k);
    int = P*x_filter;

    % 1. calculate gain vector
    gain = (1/(lambda+dot(x_filter, int)))*int;

    % 2. calculate estimation error
```

```
yRLS(k) = 0; % adaptive filter output
for n = 0:L-1
    if( (k-n) > 0)
        yRLS(k) = yRLS(k) + (wRLS(n+1) * x(k-n));
    end
end
eRLS(k) = r(k)-yRLS(k);

% 3. update filter weights
wRLS = wRLS + gain*eRLS(k);

% 4. update inverse matrix
P = (1/lambda)*(P - gain*((x_filter')*P));

% get next input vector
for j=L:-1:2
    x_filter(j)=x_filter(j-1);
end

%
% NLMS Dual-H, NLMS
%

% calculate the ONline and OFFline filter output:
yON(k) = 0;
yOFF(k) = 0;
yNLMS(k) = 0;
for n = 0:L-1
    if( (k-n) > 0)
        yOFF(k) = yOFF(k) + (wOFF(n+1) * x(k-n));
        yNLMS(k) = yNLMS(k) + (wNLMS(n+1) * x(k-n));
        yON(k) = yON(k) + (wON(n+1) * x(k-n));
    end
end

% calculate the ONline and OFFline error:
% error sig = r(sig+echo) - y(est echo)
% = est of sig
eON(k) = r(k) - yON(k);
eOFF(k) = r(k) - yOFF(k);
eNLMS(k) = r(k) - yNLMS(k);

% find power of d and e in ERLE buffer(for ERLE calculation b
% adds next value and subtracts last value rather than recomp
% whole buffer each time
if k > ERLE_L
    xp = xp + x(k)*x(k) - x(k-ERLE_L)*x(k-ERLE_L);
    dp = dp + d(k)*d(k) - d(k-ERLE_L)*d(k-ERLE_L);
    epON = epON + eON(k)*eON(k) - eON(k-ERLE_L)*eON(k-ERLE_L)
    epOFF = epOFF + eOFF(k)*eOFF(k) - eOFF(k-ERLE_L)*eOFF(k-E
    epNLMS = epNLMS + eNLMS(k)*eNLMS(k) - eNLMS(k-ERLE_L)*eNL
    epRLS = epRLS + eRLS(k)*eRLS(k) - eRLS(k-ERLE_L)*eRLS(k-E
else
    xp = xp + x(k)*x(k);
    dp = dp + d(k)*d(k);
    epON = epON + eON(k)*eON(k);
    epOFF = epOFF + eOFF(k)*eOFF(k);
    epNLMS = epNLMS + eNLMS(k)*eNLMS(k);
    epRLS = epRLS + eRLS(k)*eRLS(k);
end
```

```
        % calculate true ERLE (dB) for plotting
        ERLE_ON(k) = 10*log10(dp / (epON - s(k) + delta));
        ERLE_OFF(k) = 10*log10(dp / (epOFF - s(k) + delta));
        ERLE_NLMS(k) = 10*log10(dp / (epNLMS - s(k) + delta));
        ERLE_RLS(k) = 10*log10(dp / (epRLS - s(k) + delta));

        % calculate estimated ERLE (dB) for dual-H operation
        ERLE_ONdh = 10*log10(dp / (epON + delta));
        ERLE_OFFdh = 10*log10(dp / (epOFF + delta));

        % update the OFFline adaptive filter coeff (Dual-H NLMS, NLMS
        for n = 0:L-1
            if( (k-n) > 0)
                wOFF(n+1) = wOFF(n+1) + mu / (xp + delta)*eOFF(k)*x(k-
                wNLMS(n+1) = wNLMS(n+1) + mu2 / (xp + delta)*eNLMS(k)*
            end
        end

        % update the ONline filter coeff with the OFFline coeff if the
        % OFFline ERLE is larger than the best ERLE found so far
        if (k < 10*L)
            wON = wOFF;
            ERLEdh_best = ERLE_OFFdh;
        elseif (ERLE_OFFdh >= ERLEdh_best)
            wON = wOFF;
            ERLEdh_best = ERLE_OFFdh;
        end

        % update best ERLE if ONline ERLE is larger
        if (ERLE_ONdh >= ERLEdh_best) && (k>ERLE_L)
            ERLEdh_best = ERLE_ONdh;
        end

        % update the OFFline filter coeff with the ONline coeff if the
        % ONline ERLE is larger than the OFFline ERLE
        if (ERLE_ONdh > (ERLE_OFFdh+3)) && (k>ERLE_L)
            wOFF = wON;
        end

        % update best ERLE plotting variable
        ERLEdh_best_plot(k) = ERLEdh_best;

        % reduce ERLEdh_best at a rate of 5dB/s
        ERLEdh_best = ERLEdh_best - 5/Fs;

        % update plotting variables
        WON(:,k) = wON;
        WOFF(:,k) = wOFF;

        if (abs(d(k)) > delta) && (abs(s(k)) > delta)  % double talk
            dt_plot(k) = 20;
        else             % not double talk
            dt_plot(k) = 0;
        end
    end
    %---------------------------------------------------------------
    %
    % calculate plotting data
    %
    tdelay_plot_DH(tdelay_next,:) = [tdelay_range(tdelay_next), max(r
        tdelay_plot_NLMS(tdelay_next,:) = [tdelay_range(tdelay_next), max
        tdelay_plot_RLS(tdelay_next,:) = [tdelay_range(tdelay_next), max(

        % smooth output using moving average
        for i=1:N-500
            ERLE_ON(i) = mean(ERLE_ON(i:i+500));
            ERLE_NLMS(i) = mean(ERLE_NLMS(i:i+500));
            ERLE_RLS(i) = mean(ERLE_RLS(i:i+500));
        end
        %-----------------------------------------------------------------------
        figure(tdelay_next+10*next)
        subplot(3,1,1);
        plot(abs(eON - s'));
        ylim([0 1])
        ylabel('|Error|');
        legend(['Dual-H NLMS, \mu = ',num2str(mu)],1);
        %-----------------------------------------------------------------------
        subplot(3,1,2);
        plot(abs(eNLMS - s'));
        ylim([0 1])
        ylabel('|Error|');
        legend(['NLMS, \mu = ',num2str(mu2)],1);
        %-----------------------------------------------------------------------
        subplot(3,1,3);
        plot(abs(eRLS - s'));
        ylim([0 1])
        ylabel('|Error|');
        xlabel('Sample Number');
        legend(['RLS, \lambda = ',num2str(lambda)],1);

        % Put a main title above the subplot titles
        main_title = ['|Error| plot: Tail circuit delay = ',...
            num2str(tdelay_range(tdelay_next)), ...
            ' ms, Adaptive Filter Order = ',num2str(L)];
        mtit(main_title, 'xoff',-.1,'yoff',.025, 'fontsize', 14);
    end
    figure(tdelay_next+1+10*next)
    plot(tdelay_plot_DH(:,1), tdelay_plot_DH(:,2),'x', tdelay_plot_NLMS(:,
        '*',tdelay_plot_RLS(:,1), tdelay_plot_RLS(:,2),'+','LineWidth',2,
            'MarkerSize',10);
    title(['Max ERLE vs Tail Circuit Delay: Adaptive Filter Order = ',num
        'fontsize', 14);
    xlabel('Tail Circuit Delay (ms)','fontsize', 12);
    ylabel('Max ERLE (dB)','fontsize', 12);
    legend(['Dual-H NLMS, \mu = ',num2str(mu)],['NLMS, \mu = ',num2str(mu2)]
      ['RLS, \lambda = ',num2str(lambda)],3);
end

disp('##########################################################################
disp('Experiment finished!')
fprintf( 1, 'Save figures if necessary then press ENTER to run a new exper
pause
```

B.5 The `exp4_longrir.m` MATLAB function

The function `exp4_longrir.m` runs Experiment 4 (see 5.7.4) and is called by the main script file for the AEC simulation, `aec_sim.m`.

Listing B.5: The AEC Experiment 4 function file..

```
function [] = exp4_longrir()
% exp4_longrir.m
%
% * Experiment #4 = Effects of a RIR filter longer than the adaptive filte
%   and changing the room reflection coefficient, R.
%
% * Function file called by AEC simulator (aec_sim.m)
%
% * Creates plots of |Error| for various values of R (room reflection coe
%   and RIR filter length. Also plots Max ERLE vs R.
%
% * Compares performance of the following adaptive algorithms:
%   NLMS
%   NLMS in dual-H configuration
%   RLS
%
% * Experiment designed as part of the final year engineering project 'Ec
%   Cancellation in VoIP' for ENG4111/2 University of Southern Queensland
%
% * Adapted from adechosp.m by J.Leis
%
% * Room impulse response filter created using rir.m and fconv.m
%   (Copyright   2003 Stephen G. McGovern)
%
% * The main script aec.sim.m requires the following input sound files in
%   the working directory:
%   karl10s_mp2_8_dec.wav    karl10s_mp2_16_dec.wav,
%   karl10s_mp2_32_dec.wav   karl10s_mp2_64_dec.wav,
%   karl10s_8kHz_8bit.wav    ricky10s_8kHz_8bit.wav
%   karl10s_8kHz_8bit_mulaw.wav
%
% * Requires the following helper functions in the working directory:
%   rir.m, fconv.m and mtit.m
%
% Shane Kmita, Oct 2011

%------------------------------------------------------------------------
% Input sound vectors
%------------------------------------------------------------------------
% Far-End (fe) signal
[x Fs] = wavread('karl10s_8kHz_8bit.wav');
N = length(x);

% Near-End (ne) signal
s = zeros(N,1);                        % no ne speech

% Level shift fe to 80% maximum
x = 0.8*x/(max(abs(x)));
```

```
N = min(length(x), length(s));
x = x(1:N);
s = s(1:N);

% vector of reflection coefficients
R_range = [0.2,0.4,0.6,0.8];

% vector of adaptive filter lengths
test_length = [100, 200, 250];

RIR_order = 1000;

%------------------------------------------------------------------------
% AEC Simulation
%------------------------------------------------------------------------

for next = 1:length(test_length)

    L = test_length(next);

    for R_next = 1:length(R_range)

        %------------------------------------------------------------------------
        % Create ne room impulse response
        %------------------------------------------------------------------------

        rm=[3 3 3];            % room dimensions [L W H] in metres
        mic=[2.5 4 0.9];       % mic position
        src=[2.5 4 1.9];       % source position
        R = R_range(R_next);            % reflection coefficient (-1<r<1)
        n=24;
        b=rir(Fs, mic, n, R, rm, src);

        %------------------------------------------------------------------------
        % Echo delay ne signal
        %------------------------------------------------------------------------

        d = zeros(N,1);        % echo delayed fe signal, (*not* observable)
        for k = 1:N
            for i = 0:length(b)-1
                if k-i > 0
                    d(k) = d(k) + b(i+1)*x(k-i);
                end
            end
        end

        r = s + d;                          % ne signal + fe echo (observable

        %------------------------------------------------------------------------
        % Initialise variables / set parameters
        %------------------------------------------------------------------------
        % Dual-H NLMS
        mu = 1;                   % NLMS step size
        wON = zeros(L, 1);        % ONline adaptive filter weights
        wOFF = zeros(L, 1);       % OFFline adaptive filter weights
        yON = zeros(N,1);         % ONline adaptive filter output
        yOFF = zeros(N,1);        % OFFline adaptive filter otput
        eON  = zeros(1, N);       % ONline residual error
        eOFF  = zeros(1, N);      % OFFline residual error
        delta = 0.000001;         % NLMS/ERLE constant to avoid division by
```

```
% NLMS
mu2 = 1;                     % NLMS step size
wNLMS = zeros(L, 1);         % NLMS adaptive filter weights
yNLMS = zeros(N,1);          % NLMS adaptive filter otput (estimate of
eNLMS  = zeros(1, N);        % NLMS residual error

% RLS
lambda = 0.9;                % RLS forgetting factor
% RLS variables
wRLS = zeros(L, 1);          % RLS filter weights
x_filter = zeros(L, 1);      % input signal in filter
P = eye(L);                  % inverse input correlation matrix
int= zeros(L, 1);            % intermediate calculation step = P(n-1)*
gain = zeros(L, 1);          % gain vector
yRLS = zeros(N,1);           % RLS adaptive filter output
eRLS  = zeros(1, N);         % RLS residual error signal

% ERLE variables
ERLE_L = L;                  % order of ERLE calculation vectors
ERLE_ONdh = 0;               % current ONline filter (ERLE estimate)
ERLE_OFFdh = 0;              % current OFFline filter (ERLE estimate)
dp = 0;                      % power of d in ERLE vector
xp = 0;                      % power of x in ERLE vector
epON = 0;                    % power of eON in ERLE vector
epOFF = 0;                   % power of eOFF in ERLE vector
epNLMS = 0;                  % power of eNLMS in ERLE vector
epRLS = 0;                   % power of eRLS in ERLE vector
ERLEdh_best = 0;             % best ERLE (dual-H estimate) found so fa

% Plotting variables - otherwise not necessary for simulation
WON = zeros(L, N);           % saves the ONline adaptive weights for p
WOFF = zeros(L, N);          % saves the OFFline adaptive weights for
ERLE_ON = zeros(N,1);        % saves the ONline ERLE for plotting
ERLE_OFF = zeros(N,1);       % saves the OFFline ERLE for plotting
ERLE_NLMS = zeros(N,1);      % saves the NLMS ERLE for plotting
ERLE_RLS = zeros(N,1);       % saves the RLS ERLE for plotting
ERLEdh_best_plot = zeros(N,1);  % saves the current best ERLE for
dt_plot = zeros(N,1);        % double-talk flag (0 if fe only, 1 if dt
%---------------------------------------------------------------------

for k = 1:N
    %
    % RLS
    %
    x_filter(1)=x(k);
    int = P*x_filter;

    % 1. calculate gain vector
    gain = (1/(lambda+dot(x_filter, int)))*int;

    % 2. calculate estimation error
    yRLS(k) = 0; % adaptive filter output
    for n = 0:L-1
        if( (k-n) > 0)
            yRLS(k) = yRLS(k) + (wRLS(n+1) * x(k-n));
        end
    end
    eRLS(k) = r(k)-yRLS(k);

    % 3. update filter weights
```

```
wRLS = wRLS + gain*eRLS(k);

% 4. update inverse matrix
P = (1/lambda)*(P - gain*((x_filter')*P));

% get next input vector
for j=L:-1:2
    x_filter(j)=x_filter(j-1);
end

%
% NLMS Dual-H, NLMS
%

% calculate the ONline and OFFline filter output:
yON(k) = 0;
yOFF(k) = 0;
yNLMS(k) = 0;
for n = 0:L-1
    if( (k-n) > 0)
        yOFF(k) = yOFF(k) + (wOFF(n+1) * x(k-n));
        yNLMS(k) = yNLMS(k) + (wNLMS(n+1) * x(k-n));
        yON(k) = yON(k) + (wON(n+1) * x(k-n));
    end
end

% calculate the ONline and OFFline error:
% error sig = r(sig+echo) - y(est echo)
% = est of sig
eON(k) = r(k) - yON(k);
eOFF(k) = r(k) - yOFF(k);
eNLMS(k) = r(k) - yNLMS(k);

% find power of d and e in ERLE buffer(for ERLE calculation b
% adds next value and subtracts last value rather than recomp
% whole buffer each time
if k > ERLE_L
    xp = xp + x(k)*x(k) - x(k-ERLE_L)*x(k-ERLE_L);
    dp = dp + d(k)*d(k) - d(k-ERLE_L)*d(k-ERLE_L);
    epON = epON + eON(k)*eON(k) - eON(k-ERLE_L)*eON(k-ERLE_L)
    epOFF = epOFF + eOFF(k)*eOFF(k) - eOFF(k-ERLE_L)*eOFF(k-E
    epNLMS = epNLMS + eNLMS(k)*eNLMS(k) - eNLMS(k-ERLE_L)*eNL
    epRLS = epRLS + eRLS(k)*eRLS(k) - eRLS(k-ERLE_L)*eRLS(k-E
else
    xp = xp + x(k)*x(k);
    dp = dp + d(k)*d(k);
    epON = epON + eON(k)*eON(k);
    epOFF = epOFF + eOFF(k)*eOFF(k);
    epNLMS = epNLMS + eNLMS(k)*eNLMS(k);
    epRLS = epRLS + eRLS(k)*eRLS(k);
end

% calculate true ERLE (dB) for plotting
ERLE_ON(k) = 10*log10(dp / (epON - s(k) + delta));
ERLE_OFF(k) = 10*log10(dp / (epOFF - s(k) + delta));
ERLE_NLMS(k) = 10*log10(dp / (epNLMS - s(k) + delta));
ERLE_RLS(k) = 10*log10(dp / (epRLS - s(k) + delta));

% calculate estimated ERLE (dB) for dual-H operation
ERLE_ONdh = 10*log10(dp / (epON + delta));
```

```
          ERLE_OFFdh = 10*log10(dp / (epOFF + delta));

          % update the OFFline adaptive filter coeff (Dual-H NLMS, NLMS)
          for n = 0:L-1
               if( (k-n) > 0)
                    wOFF(n+1) = wOFF(n+1) + mu / (xp + delta)*eOFF(k)*x(k-
                    wNLMS(n+1) = wNLMS(n+1) + mu2 / (xp + delta)*eNLMS(k)*
               end
          end

          % update the ONline filter coeff with the OFFline coeff if the
          % OFFline ERLE is larger than the best ERLE found so far
          if (k < 10*L)
               wON = wOFF;
               ERLEdh_best = ERLE_OFFdh;
          elseif (ERLE_OFFdh >= ERLEdh_best)
               wON = wOFF;
               ERLEdh_best = ERLE_OFFdh;
          end

          % update best ERLE if ONline ERLE is larger
          if (ERLE_ONdh >= ERLEdh_best) && (k>ERLE_L)
               ERLEdh_best = ERLE_ONdh;
          end

          % update the OFFline filter coeff with the ONline coeff if the
          % ONline ERLE is larger than the OFFline ERLE
          if (ERLE_ONdh > (ERLE_OFFdh+3)) && (k>ERLE_L)
               wOFF = wON;
          end

          % update best ERLE plotting variable
          ERLEdh_best_plot(k) = ERLEdh_best;

          % reduce ERLEdh_best at a rate of 5dB/s
          ERLEdh_best = ERLEdh_best - 5/Fs;

          % update plotting variables
          WON(:,k) = wON;
          WOFF(:,k) = wOFF;

          if (abs(d(k)) > delta) && (abs(s(k)) > delta)  % double talk
               dt_plot(k) = 20;
          else             % not double talk
               dt_plot(k) = 0;
          end
     end
     %------------------------------------------------------------------------
     %
     % calculate plotting data
     %
     R_plot_DH(R_next,:) = [R, max(real(ERLE_ON))];
     R_plot_NLMS(R_next,:) = [R, max(real(ERLE_NLMS))];
     R_plot_RLS(R_next,:) = [R, max(real(ERLE_RLS))];

     % smooth output using moving average
     for i=1:N-500
          ERLE_ON(i) = mean(ERLE_ON(i:i+500));
          ERLE_NLMS(i) = mean(ERLE_NLMS(i:i+500));
          ERLE_RLS(i) = mean(ERLE_RLS(i:i+500));
     end
```

```
        %----------------------------------------------------------------------
        figure(R_next+10*next)
        subplot(3,1,1);
        plot(abs(eON - s'));
        ylim([0 2])
        ylabel('|Error|');
        legend(['Dual-H NLMS, \mu = ',num2str(mu)],1);
        %----------------------------------------------------------------------
        subplot(3,1,2);
        plot(abs(eNLMS - s'));
        ylim([0 2])
        ylabel('|Error|');
        legend(['NLMS, \mu = ',num2str(mu2)],1);
        %----------------------------------------------------------------------
        %----------------------------------------------------------------------
        subplot(3,1,3);
        plot(abs(eRLS - s'));
        ylim([0 2])
        ylabel('|Error|');
        xlabel('Sample Number');
        legend(['RLS, \lambda = ',num2str(lambda)],1);

        % Put a main title above the subplot titles
        main_title = ['|Error| plot: RIR Order = ',num2str(RIR_order), ...
        ', Adaptive Filter Order = ',num2str(L),...
        ', R = ',num2str(R)];
        mtit(main_title, 'xoff',-.1,'yoff',.025, 'fontsize', 15);
    end
    figure(R_next+1+10*next)
    plot(R_plot_DH(:,1), R_plot_DH(:,2),'x', R_plot_NLMS(:,1), R_plot_NLMS
        '*',R_plot_RLS(:,1), R_plot_RLS(:,2),'+','LineWidth',2,...
                'MarkerSize',10);
    title(['Max ERLE vs R: RIR Order = ',num2str(RIR_order), ...
        ', Adaptive Filter Order = ',num2str(L)],'fontsize', 14);
    xlabel('R','fontsize', 12);
    ylabel('Max ERLE (dB)','fontsize', 12);
    legend(['Dual-H NLMS, \mu = ',num2str(mu)],['NLMS, \mu = ',num2str(mu2)]
      ['RLS, \lambda = ',num2str(lambda)],1);
end

disp('###########################################################################
disp('Experiment finished!')
fprintf( 1, 'Save figures if necessary then press ENTER to run a new exper
pause
```

B.6 The exp5_noise.m MATLAB function

The function exp5_noise.m runs Experiment 5 (see 5.7.5) and is called by the main script file for the AEC simulation, aec_sim.m.

Listing B.6: The AEC Experiment 5 function file..

```
function [] = exp5_noise()
% exp5_noise.m
%
% * Experiment #5 = Effects of background noise
%
% * Function file called by AEC simulator (aec_sim.m)
%
% * Creates plots of |Error| for various values of SNR and adaptive filte
%   length. Also plots Max ERLE vs SNR.
%
% * Compares performance of the following adaptive algorithms:
%   NLMS
%   NLMS in dual-H configuration
%   RLS
%
% * Experiment designed as part of the final year engineering project 'Ec
%   Cancellation in VoIP' for ENG4111/2 University of Southern Queensland
%
% * Adapted from adechosp.m by J.Leis
%
% * Room impulse response filter created using rir.m and fconv.m
%   (Copyright   2003 Stephen G. McGovern)
%
% * The main script aec.sim.m requires the following input sound files in
%   the working directory:
%   karl10s_mp2_8_dec.wav    karl10s_mp2_16_dec.wav,
%   karl10s_mp2_32_dec.wav   karl10s_mp2_64_dec.wav,
%   karl10s_8kHz_8bit.wav    ricky10s_8kHz_8bit.wav
%   karl10s_8kHz_8bit_mulaw.wav
%
% * Requires the following helper functions in the working directory:
%   rir.m, fconv.m and mtit.m
%
% Shane Kmita, Oct 2011

%---------------------------------------------------------------------------
% Input sound vectors
%---------------------------------------------------------------------------
% Far-End (fe) signal
N = 20000; Fs = 8000; x = randn(N,1);     % N Gaussian white noise samples

% Near-End (ne) signal
s = randn(N,1);     % N Gaussian white noise samples

% Level shift fe to 80% maximum
x = 0.8*x/(max(abs(x)));

% Level shift ne to 100% maximum
s = s/(max(abs(s)));

N = min(length(x), length(s));
x = x(1:N);
s = s(1:N);

% vector of noise multipliers - sets maximum noise amplitude
noise_range = [0.003, 0.01, 0.03, 0.1, 0.3];

% vector of adaptive filter lengths
test_length = [100, 200, 250];   % adaptive filter order
```

```
%-----------------------------------------------------------------------
% AEC Simulation
%-----------------------------------------------------------------------

for next = 1:length(test_length)

    L = test_length(next);
    %-----------------------------------------------------------------------
    % Create ne room impulse response
    %-----------------------------------------------------------------------

    rm=[3 3 3];             % room dimensions [L W H] in metres
    mic=[2.5 4 0.9];        % mic position
    src=[2.5 4 1.9];        % source position
    r=-0.5;                  % reflection coefficient (-1<r<1)
    n=24;
    b=rir(Fs, mic, n, r, rm, src);
    b = b(1:L);           % truncate impulse response to adfilter length

    %-----------------------------------------------------------------------
    % Echo delay ne signal
    %-----------------------------------------------------------------------
    %-----------------------------------------------------------------------
    % Echo delay ne signal
    %-----------------------------------------------------------------------

    d = zeros(N,1);         % echo delayed fe signal, (*not* observable)
    for k = 1:N
       for i = 0:length(b)-1
           if k-i > 0
               d(k) = d(k) + b(i+1)*x(k-i);
           end
       end
    end

    for nr = 1:length(noise_range)

        % Level shift by noise_range multiplier
        s_new = noise_range(nr)*s;

        r = s_new + d;                          % ne signal + fe echo (observ

        SNR = 10*log10( sum(d.*d) / (sum(s_new.*s_new)+0.0000001) );
        %-----------------------------------------------------------------------
        % Initialise variables / set parameters
        %-----------------------------------------------------------------------
        % Dual-H NLMS
        mu = 1;                     % NLMS step size
        wON = zeros(L, 1);          % ONline adaptive filter weights
        wOFF = zeros(L, 1);         % OFFline adaptive filter weights
        yON = zeros(N,1);           % ONline adaptive filter output
        yOFF = zeros(N,1);          % OFFline adaptive filter otput
        eON  = zeros(1, N);         % ONline residual error
        eOFF  = zeros(1, N);        % OFFline residual error
        delta = 0.000001;           % NLMS/ERLE constant to avoid division by

        % NLMS
        mu2 = 1;                    % NLMS step size
        wNLMS = zeros(L, 1);        % NLMS adaptive filter weights
        yNLMS = zeros(N,1);         % NLMS adaptive filter otput (estimate of
        eNLMS   = zeros(1, N);     % NLMS residual error

        % RLS
        lambda = 0.9;              % RLS forgetting factor
        % RLS variables
        wRLS = zeros(L, 1);        % RLS filter weights
        x_filter = zeros(L, 1);    % input signal in filter
        P = eye(L);                % inverse input correlation matrix
        int= zeros(L, 1);          % intermediate calculation step = P(n-1)*
        gain = zeros(L, 1);        % gain vector
        yRLS = zeros(N,1);         % RLS adaptive filter output
        eRLS  = zeros(1, N);       % RLS residual error signal

        % ERLE variables
        ERLE_L = L;                % order of ERLE calculation vectors
        ERLE_ONdh = 0;             % current ONline filter (ERLE estimate)
        ERLE_OFFdh = 0;            % current OFFline filter (ERLE estimate)
        dp = 0;                    % power of d in ERLE vector
        xp = 0;                    % power of x in ERLE vector
        epON = 0;                  % power of eON in ERLE vector
        epOFF = 0;                 % power of eOFF in ERLE vector
        epNLMS = 0;                % power of eNLMS in ERLE vector
        epRLS = 0;                 % power of eRLS in ERLE vector
        ERLEdh_best = 0;           % best ERLE (dual-H estimate) found so fa

        % Plotting variables - otherwise not necessary for simulation
        WON = zeros(L, N);         % saves the ONline adaptive weights for p
        WOFF = zeros(L, N);        % saves the OFFline adaptive weights for
        ERLE_ON = zeros(N,1);      % saves the ONline ERLE for plotting
        ERLE_OFF = zeros(N,1);     % saves the OFFline ERLE for plotting
        ERLE_NLMS = zeros(N,1);    % saves the NLMS ERLE for plotting
        ERLE_RLS = zeros(N,1);     % saves the RLS ERLE for plotting
        ERLEdh_best_plot = zeros(N,1);  % saves the current best ERLE for
        dt_plot = zeros(N,1);      % double-talk flag (0 if fe only, 1 if dt
        %----------------------------------------------------------------------

        for k = 1:N
            %
            % RLS
            %
            x_filter(1)=x(k);
            int = P*x_filter;

            % 1. calculate gain vector
            gain = (1/(lambda+dot(x_filter, int)))*int;

            % 2. calculate estimation error
            yRLS(k) = 0; % adaptive filter output
            for n = 0:L-1
                if( (k-n) > 0)
                    yRLS(k) = yRLS(k) + (wRLS(n+1) * x(k-n));
                end
            end
            eRLS(k) = r(k)-yRLS(k);

            % 3. update filter weights
            wRLS = wRLS + gain*eRLS(k);

            % 4. update inverse matrix
            P = (1/lambda)*(P - gain*((x_filter')*P));
```

```
% get next input vector
for j=L:-1:2
    x_filter(j)=x_filter(j-1);
end

%
% NLMS Dual-H, NLMS
%

% calculate the ONline and OFFline filter output:
yON(k) = 0;
yOFF(k) = 0;
yNLMS(k) = 0;
for n = 0:L-1
    if( (k-n) > 0)
        yOFF(k) = yOFF(k) + (wOFF(n+1) * x(k-n));
        yNLMS(k) = yNLMS(k) + (wNLMS(n+1) * x(k-n));
        yON(k) = yON(k) + (wON(n+1) * x(k-n));
    end
end

% calculate the ONline and OFFline error:
% error sig = r(sig+echo) - y(est echo)
% = est of sig
eON(k) = r(k) - yON(k);
eOFF(k) = r(k) - yOFF(k);
eNLMS(k) = r(k) - yNLMS(k);

% find power of d and e in ERLE buffer(for ERLE calculation b
% adds next value and subtracts last value rather than recomp
% whole buffer each time
if k > ERLE_L
    xp = xp + x(k)*x(k) - x(k-ERLE_L)*x(k-ERLE_L);
    dp = dp + d(k)*d(k) - d(k-ERLE_L)*d(k-ERLE_L);
    epON = epON + eON(k)*eON(k) - eON(k-ERLE_L)*eON(k-ERLE_L);
    epOFF = epOFF + eOFF(k)*eOFF(k) - eOFF(k-ERLE_L)*eOFF(k-E
    epNLMS = epNLMS + eNLMS(k)*eNLMS(k) - eNLMS(k-ERLE_L)*eNL
    epRLS = epRLS + eRLS(k)*eRLS(k) - eRLS(k-ERLE_L)*eRLS(k-E
else
    xp = xp + x(k)*x(k);
    dp = dp + d(k)*d(k);
    epON = epON + eON(k)*eON(k);
    epOFF = epOFF + eOFF(k)*eOFF(k);
    epNLMS = epNLMS + eNLMS(k)*eNLMS(k);
    epRLS = epRLS + eRLS(k)*eRLS(k);
end

% calculate true ERLE (dB) for plotting
ERLE_ON(k) = 10*log10(dp / (epON - s(k) + delta));
ERLE_OFF(k) = 10*log10(dp / (epOFF - s(k) + delta));
ERLE_NLMS(k) = 10*log10(dp / (epNLMS - s(k) + delta));
ERLE_RLS(k) = 10*log10(dp / (epRLS - s(k) + delta));

% calculate estimated ERLE (dB) for dual-H operation
ERLE_ONdh = 10*log10(dp / (epON + delta));
ERLE_OFFdh = 10*log10(dp / (epOFF + delta));

% update the OFFline adaptive filter coeff (Dual-H NLMS, NLMS
for n = 0:L-1
```

```
            if( (k-n) > 0)
                wOFF(n+1) = wOFF(n+1) + mu / (xp + delta)*eOFF(k)*x(k-
                wNLMS(n+1) = wNLMS(n+1) + mu2 / (xp + delta)*eNLMS(k)*
            end
        end

        % update the ONline filter coeff with the OFFline coeff if the
        % OFFline ERLE is larger than the best ERLE found so far
        if (k < 10*L)
            wON = wOFF;
            ERLEdh_best = ERLE_OFFdh;
        elseif (ERLE_OFFdh >= ERLEdh_best)
            wON = wOFF;
            ERLEdh_best = ERLE_OFFdh;
        end

        % update best ERLE if ONline ERLE is larger
        if (ERLE_ONdh >= ERLEdh_best) && (k>ERLE_L)
            ERLEdh_best = ERLE_ONdh;
        end

        % update the OFFline filter coeff with the ONline coeff if the
        % ONline ERLE is larger than the OFFline ERLE
        if (ERLE_ONdh > (ERLE_OFFdh+3)) && (k>ERLE_L)
            wOFF = wON;
        end

        % update best ERLE plotting variable
        ERLEdh_best_plot(k) = ERLEdh_best;

        % reduce ERLEdh_best at a rate of 5dB/s
        ERLEdh_best = ERLEdh_best - 5/Fs;

        % update plotting variables
        WON(:,k) = wON;
        WOFF(:,k) = wOFF;

        if (abs(d(k)) > delta) && (abs(s(k)) > delta)  % double talk
            dt_plot(k) = 20;
        else              % not double talk
            dt_plot(k) = 0;
        end
    end
    %------------------------------------------------------------------------
    %
    % calculate plotting data
    %
    SNR_plot_DH(nr,:) = [SNR, max(real(ERLE_ON))];
    SNR_plot_NLMS(nr,:) = [SNR, max(real(ERLE_NLMS))];
    SNR_plot_RLS(nr,:) = [SNR, max(real(ERLE_RLS))];

    % smooth output using moving average
    for i=1:N-500
        ERLE_ON(i) = mean(ERLE_ON(i:i+500));
        ERLE_NLMS(i) = mean(ERLE_NLMS(i:i+500));
        ERLE_RLS(i) = mean(ERLE_RLS(i:i+500));
    end
    %------------------------------------------------------------------------
    figure(nr+10*next)
    subplot(3,1,1);
    plot(abs(eON - s_new'));
```

```
          ylim([0  2])
          ylabel('|Error|');
          legend(['Dual-H␣NLMS,␣\mu␣=',num2str(mu)],1);
          %------------------------------------------------------------------
          subplot(3,1,2);
          plot(abs(eNLMS - s_new'));
          ylim([0  2])
          ylabel('|Error|');
          legend(['NLMS,␣\mu␣=',num2str(mu2)],1);
          %------------------------------------------------------------------
          %------------------------------------------------------------------
          subplot(3,1,3);
          plot(abs(eRLS - s_new'));
          ylim([0  2])
          ylabel('|Error|');
          xlabel('Sample␣Number');
          legend(['RLS,␣\lambda␣=',num2str(lambda)],1);

          % Put a main title above the subplot titles
          main_title = ['|Error|␣during␣additive␣noise,␣Adaptive␣Filter␣Ord
              ',␣SNR␣=␣',num2str(SNR),'dB'];
          mtit(main_title, 'xoff',-.1,'yoff',.025, 'fontsize', 15);
     end
     figure(nr+1+10*next)
     plot(SNR_plot_DH(:,1), SNR_plot_DH(:,2),'x', SNR_plot_NLMS(:,1), SNR_
          '*',SNR_plot_RLS(:,1), SNR_plot_RLS(:,2),'+','LineWidth',2,...
                    'MarkerSize',10);
     title(['Max␣ERLE␣vs␣SNR,␣Adaptive␣Filter␣Order␣=␣',num2str(L)],'fontsi
     xlabel('SNR␣(dB)','fontsize', 12);
     ylabel('Max␣ERLE␣(dB)','fontsize', 12);
     legend(['Dual-H␣NLMS,␣\mu␣=',num2str(mu)],['NLMS,␣\mu␣=',num2str(mu2)]
        ['RLS,␣\lambda␣=',num2str(lambda)],4);
end

disp('##########################################################################
disp('Experiment␣finished!')
fprintf( 1, 'Save␣figures␣if␣necessary␣then␣press␣ENTER␣to␣run␣a␣new␣exper
pause
```

B.7 The exp6_comp.m MATLAB function

The function `exp6_comp.m` runs Experiment 6 (see 6.7) and is called by the main script file for the AEC simulation, `aec_sim.m`.

Listing B.7: The AEC Experiment 6 function file..

```
function [] = exp6_comp()
% exp6_comp.m
%
% * Experiment #6 = Effects of codec compression
```

```
%
% * Function file called by AEC simulator (aec_sim.m)
%
% * Creates plots of |Error| for various values of input signal bitrate a
%   adaptive filter length. Also plots Max ERLE vs bitrate.
%
% * Compares performance of the following adaptive algorithms:
%   NLMS
%   NLMS in dual-H configuration
%   RLS
%
% * Experiment designed as part of the final year engineering project 'Ec
%   Cancellation in VoIP' for ENG4111/2 University of Southern Queensland
%
% * Adapted from adechosp.m by J.Leis
%
% * Room impulse response filter created using rir.m and fconv.m
%   (Copyright   2003 Stephen G. McGovern)
%
% * The main script aec.sim.m requires the following input sound files in
%   the working directory:
%   karl10s_mp2_8_dec.wav    karl10s_mp2_16_dec.wav,
%   karl10s_mp2_32_dec.wav   karl10s_mp2_64_dec.wav,
%   karl10s_8kHz_8bit.wav    ricky10s_8kHz_8bit.wav
%   karl10s_8kHz_8bit_mulaw.wav
%
% * Requires the following helper functions in the working directory:
%   rir.m, fconv.m and mtit.m
%
% Shane Kmita, Oct 2011

%------------------------------------------------------------------------
% Input sound vectors
%------------------------------------------------------------------------
% Far-End (fe) signal
% fe inputs compressed/decompressed by mp2 codec at different bit-rates
[x1 Fs] = wavread('karl10s_mp2_8_dec.wav'); % 8kbps
[x2 Fs] = wavread('karl10s_mp2_16_dec.wav');% 16kbps
[x3 Fs] = wavread('karl10s_mp2_32_dec.wav');% 32kbps
[x4 Fs] = wavread('karl10s_mp2_64_dec.wav');% 64kbps
x_array = [x1, x2, x3, x4];

N = length(x_array);

% Near-End (ne) signal
s = zeros(N,1);                      % no ne speech

%------------------------------------------------------------------------
% Create ne room impulse response
%------------------------------------------------------------------------

rm=[3 3 3];           % room dimensions [L W H] in metres
mic=[2.5 4 0.9];      % mic position
src=[2.5 4 1.9];      % source position
r=-0.5;                % reflection coefficient (-1<r<1)
n=24;
b1=rir(Fs, mic, n, r, rm, src);

% vector of bitrates
br_range = [8,16,32,64];
```

```
% vector of adaptive filter lengths
test_length = [100, 200, 250];   % adaptive filter order

%------------------------------------------------------------------------
% AEC Simulation
%------------------------------------------------------------------------

for next = 1:length(test_length)

    L = test_length(next);
    b = b1(1:L);          % truncate impulse response to adfilter length

    for br_next = 1:length(br_range)

        br = br_range(br_next);
        x = x_array(:,br_next);
        % Level shift fe to 80% maximum
        x = 0.8*x/(max(abs(x)));

        %------------------------------------------------------------------------
        % Echo delay ne signal
        %------------------------------------------------------------------------

        d = zeros(N,1);       % echo delayed fe signal, (*not* observable)
        for k = 1:N
           for i = 0:length(b)-1
               if k-i > 0
                   d(k) = d(k) + b(i+1)*x(k-i);
               end
            end
        end

        r = s + d;                        % ne signal + fe echo (observable

        %------------------------------------------------------------------------
        % Initialise variables / set parameters
        %------------------------------------------------------------------------
        % Dual-H NLMS
        mu = 1;                  % NLMS step size
        wON = zeros(L, 1);       % ONline adaptive filter weights
        wOFF = zeros(L, 1);      % OFFline adaptive filter weights
        yON = zeros(N,1);        % ONline adaptive filter output
        yOFF = zeros(N,1);       % OFFline adaptive filter otput
        eON  = zeros(1, N);      % ONline residual error
        eOFF  = zeros(1, N);     % OFFline residual error
        delta = 0.000001;        % NLMS/ERLE constant to avoid division by

        % NLMS
        mu2 = 1;                 % NLMS step size
        wNLMS = zeros(L, 1);     % NLMS adaptive filter weights
        yNLMS = zeros(N,1);      % NLMS adaptive filter otput (estimate of
        eNLMS  = zeros(1, N);    % NLMS residual error

        % RLS
        lambda = 0.9;            % RLS forgetting factor
        % RLS variables
        wRLS = zeros(L, 1);      % RLS filter weights
        x_filter = zeros(L, 1);  % input signal in filter
        P = eye(L);              % inverse input correlation matrix
        int= zeros(L, 1);        % intermediate calculation step = P(n-1)*
```

```
gain = zeros(L, 1);        % gain vector
yRLS = zeros(N,1);         % RLS adaptive filter output
eRLS  = zeros(1, N);       % RLS residual error signal

% ERLE variables
ERLE_L = L;                % order of ERLE calculation vectors
ERLE_ONdh = 0;             % current ONline filter (ERLE estimate)
ERLE_OFFdh = 0;            % current OFFline filter (ERLE estimate)
dp = 0;                    % power of d in ERLE vector
xp = 0;                    % power of x in ERLE vector
epON = 0;                  % power of eON in ERLE vector
epOFF = 0;                 % power of eOFF in ERLE vector
epNLMS = 0;                % power of eNLMS in ERLE vector
epRLS = 0;                 % power of eRLS in ERLE vector
ERLEdh_best = 0;           % best ERLE (dual-H estimate) found so fa

% Plotting variables - otherwise not necessary for simulation
WON = zeros(L, N);         % saves the ONline adaptive weights for p
WOFF = zeros(L, N);        % saves the OFFline adaptive weights for
ERLE_ON = zeros(N,1);      % saves the ONline ERLE for plotting
ERLE_OFF = zeros(N,1);     % saves the OFFline ERLE for plotting
ERLE_NLMS = zeros(N,1);    % saves the NLMS ERLE for plotting
ERLE_RLS = zeros(N,1);     % saves the RLS ERLE for plotting
ERLEdh_best_plot = zeros(N,1);  % saves the current best ERLE for
dt_plot = zeros(N,1);      % double-talk flag (0 if fe only, 1 if dt
%--------------------------------------------------------------------

for k = 1:N
    %
    % RLS
    %
    x_filter(1)=x(k);
    int = P*x_filter;

    % 1. calculate gain vector
    gain = (1/(lambda+dot(x_filter, int)))*int;

    % 2. calculate estimation error
    yRLS(k) = 0; % adaptive filter output
    for n = 0:L-1
        if( (k-n) > 0)
            yRLS(k) = yRLS(k) + (wRLS(n+1) * x(k-n));
        end
    end
    eRLS(k) = r(k)-yRLS(k);

    % 3. update filter weights
    wRLS = wRLS + gain*eRLS(k);

    % 4. update inverse matrix
    P = (1/lambda)*(P - gain*((x_filter')*P));

    % get next input vector
    for j=L:-1:2
        x_filter(j)=x_filter(j-1);
    end

    %
    % NLMS Dual-H, NLMS
    %
```

```
% calculate the ONline and OFFline filter output:
yON(k) = 0;
yOFF(k) = 0;
yNLMS(k) = 0;
for n = 0:L-1
    if( (k-n) > 0)
        yOFF(k) = yOFF(k) + (wOFF(n+1) * x(k-n));
        yNLMS(k) = yNLMS(k) + (wNLMS(n+1) * x(k-n));
        yON(k) = yON(k) + (wON(n+1) * x(k-n));
    end
end

% calculate the ONline and OFFline error:
% error sig = r(sig+echo) - y(est echo)
% = est of sig
eON(k) = r(k) - yON(k);
eOFF(k) = r(k) - yOFF(k);
eNLMS(k) = r(k) - yNLMS(k);

% find power of d and e in ERLE buffer(for ERLE calculation b
% adds next value and subtracts last value rather than recomp
% whole buffer each time
if k > ERLE_L
    xp = xp + x(k)*x(k) - x(k-ERLE_L)*x(k-ERLE_L);
    dp = dp + d(k)*d(k) - d(k-ERLE_L)*d(k-ERLE_L);
    epON = epON + eON(k)*eON(k) - eON(k-ERLE_L)*eON(k-ERLE_L)
    epOFF = epOFF + eOFF(k)*eOFF(k) - eOFF(k-ERLE_L)*eOFF(k-E
    epNLMS = epNLMS + eNLMS(k)*eNLMS(k) - eNLMS(k-ERLE_L)*eNL
    epRLS = epRLS + eRLS(k)*eRLS(k) - eRLS(k-ERLE_L)*eRLS(k-E
else
    xp = xp + x(k)*x(k);
    dp = dp + d(k)*d(k);
    epON = epON + eON(k)*eON(k);
    epOFF = epOFF + eOFF(k)*eOFF(k);
    epNLMS = epNLMS + eNLMS(k)*eNLMS(k);
    epRLS = epRLS + eRLS(k)*eRLS(k);
end

% calculate true ERLE (dB) for plotting
ERLE_ON(k) = 10*log10(dp / (epON - s(k) + delta));
ERLE_OFF(k) = 10*log10(dp / (epOFF - s(k) + delta));
ERLE_NLMS(k) = 10*log10(dp / (epNLMS - s(k) + delta));
ERLE_RLS(k) = 10*log10(dp / (epRLS - s(k) + delta));

% calculate estimated ERLE (dB) for dual-H operation
ERLE_ONdh = 10*log10(dp / (epON + delta));
ERLE_OFFdh = 10*log10(dp / (epOFF + delta));

% update the OFFline adaptive filter coeff (Dual-H NLMS, NLMS
for n = 0:L-1
    if( (k-n) > 0)
        wOFF(n+1) = wOFF(n+1) + mu / (xp + delta)*eOFF(k)*x(k
        wNLMS(n+1) = wNLMS(n+1) + mu2 / (xp + delta)*eNLMS(k)
    end
end

% update the ONline filter coeff with the OFFline coeff if th
% OFFline ERLE is larger than the best ERLE found so far
if (k < 10*L)
```

```
            wON = wOFF;
            ERLEdh_best = ERLE_OFFdh;
        elseif (ERLE_OFFdh >= ERLEdh_best)
            wON = wOFF;
            ERLEdh_best = ERLE_OFFdh;
        end

        % update best ERLE if ONline ERLE is larger
        if (ERLE_ONdh >= ERLEdh_best) && (k>ERLE_L)
            ERLEdh_best = ERLE_ONdh;
        end

        % update the OFFline filter coeff with the ONline coeff if the
        % ONline ERLE is larger than the OFFline ERLE
        if (ERLE_ONdh > (ERLE_OFFdh+3)) && (k>ERLE_L)
            wOFF = wON;
        end

        % update best ERLE plotting variable
        ERLEdh_best_plot(k) = ERLEdh_best;

        % reduce ERLEdh_best at a rate of 5dB/s
        ERLEdh_best = ERLEdh_best - 5/Fs;

        % update plotting variables
        WON(:,k) = wON;
        WOFF(:,k) = wOFF;

        if (abs(d(k)) > delta) && (abs(s(k)) > delta)  % double talk
            dt_plot(k) = 20;
        else            % not double talk
            dt_plot(k) = 0;
        end
    end
    %---------------------------------------------------------------------------
    %
    % calculate plotting data
    %
    br_plot_DH(br_next,:) = [br, max(real(ERLE_ON))];
    br_plot_NLMS(br_next,:) = [br, max(real(ERLE_NLMS))];
    br_plot_RLS(br_next,:) = [br, max(real(ERLE_RLS))];

    % smooth output using moving average
    for i=1:N-500
        ERLE_ON(i) = mean(ERLE_ON(i:i+500));
        ERLE_NLMS(i) = mean(ERLE_NLMS(i:i+500));
        ERLE_RLS(i) = mean(ERLE_RLS(i:i+500));
    end
    %---------------------------------------------------------------------------
    figure(br_next+10*next)
    subplot(3,1,1);
    plot(abs(eON - s'));
    ylim([0 2])
    ylabel('|Error|');
    legend(['Dual-H NLMS, \mu = ',num2str(mu)],1);
    %---------------------------------------------------------------------------
    subplot(3,1,2);
    plot(abs(eNLMS - s'));
    ylim([0 2])
    ylabel('|Error|');
```

```
        legend(['NLMS, \mu = ',num2str(mu2)],1);
        %--------------------------------------------------------------------------
        %--------------------------------------------------------------------------
        subplot(3,1,3);
        plot(abs(eRLS - s'));
        ylim([0 2])
        ylabel('|Error|');
        xlabel('Sample Number');
        legend(['RLS, \lambda = ',num2str(lambda)],1);

        % Put a main title above the subplot titles
        main_title = ['|Error| plot: Bitrate = ',num2str(br), ...
        'kbps, Adaptive Filter Order = ',num2str(L)];
        mtit(main_title, 'xoff',-.1,'yoff',.025, 'fontsize', 15);
    end
    figure(br_next+1+10*next)
    plot(br_plot_DH(:,1), br_plot_DH(:,2),'x', br_plot_NLMS(:,1), br_plot_N
        '*',br_plot_RLS(:,1), br_plot_RLS(:,2),'+','LineWidth',2,...
                'MarkerSize',10);
    title(['Max ERLE vs Bitrate: Adaptive Filter Order = ',num2str(L)],...
        'fontsize', 14);
    xlabel('Bitrate (kbps)','fontsize', 12);
    ylabel('Max ERLE (dB)','fontsize', 12);
    legend(['Dual-H NLMS, \mu = ',num2str(mu)],['NLMS, \mu = ',num2str(mu2)]
      ['RLS, \lambda = ',num2str(lambda)],4);
end

disp('############################################################################
disp('Experiment finished!')
fprintf( 1, 'Save figures if necessary then press ENTER to run a new exper
pause
```

B.8 The exp7_drop_pack.m MATLAB function

The function exp7_drop_pack.m runs Experiment 7 (see 5.7.7) and is called by the main script file for the AEC simulation, aec_sim.m.

Listing B.8: The AEC Experiment 7 function file..

```
function [] = exp7_drop_pack()
% exp7_drop_pack.m
%
% * Experiment #7 = Effects of dropped packet bursts
%
% * Function file called by AEC simulator (aec_sim.m)
%
% * Creates plots of |Error| for various sized bursts of dropped packets
%   adaptive filter length. Also plots Max ERLE vs burst size.
%
% * Compares performance of the following adaptive algorithms:
```

```
%   NLMS
%   NLMS in dual-H configuration
%   RLS
%
% * Experiment designed as part of the final year engineering project 'Ec
%   Cancellation in VoIP' for ENG4111/2 University of Southern Queensland
%
% * Adapted from adechosp.m by J.Leis
%
% * Room impulse response filter created using rir.m and fconv.m
%   (Copyright   2003 Stephen G. McGovern)
%
% * The main script aec.sim.m requires the following input sound files in
%   the working directory:
%   karl10s_mp2_8_dec.wav   karl10s_mp2_16_dec.wav,
%   karl10s_mp2_32_dec.wav  karl10s_mp2_64_dec.wav,
%   karl10s_8kHz_8bit.wav   ricky10s_8kHz_8bit.wav
%   karl10s_8kHz_8bit_mulaw.wav
%
% * Requires the following helper functions in the working directory:
%   rir.m, fconv.m and mtit.m
%
% Shane Kmita, Oct 2011

%--------------------------------------------------------------------------
% Input sound vectors
%--------------------------------------------------------------------------
% Far-End (fe) signal
N = 20000; Fs = 8000; x = randn(N,1);     % N Gaussian white noise samples

% Near-End (ne) signal
s = zeros(N,1);                        % no ne speech

% Level shift fe to 80% maximum
x = 0.8*x/(max(abs(x)));

%--------------------------------------------------------------------------
% Create ne room impulse response
%--------------------------------------------------------------------------

rm=[3 3 3];          % room dimensions [L W H] in metres
mic=[2.5 4 0.9];     % mic position
src=[2.5 4 1.9];     % source position
r=-0.5;               % reflection coefficient (-1<r<1)
n=24;
b1=rir(Fs, mic, n, r, rm, src);

% vector of adaptive filter lengths
test_length = [100, 200, 250];   % adaptive filter order

% vector of packet loss burst lengths
pl_range = [5,15,25,35,45]; % packet loss bursts (consecutive packets los

vp = 20;             % voice payload (ms) (160 default for G.711)
br = 64000;          % bitrate (bps) (64000 for G.711)
bd = 8;              % bit depth (8 for G.711)
vpp = vp/1000*br/bd; % voice payload (samples per packet)
pl_range2 = floor(pl_range*vpp); % packet loss bursts (consecutive sample

%--------------------------------------------------------------------------
% AEC Simulation
```

```
%---------------------------------------------------------------------------
for next = 1:length(test_length)

    L = test_length(next);
    b = b1(1:L);          % truncate impulse response to adfilter length

    for pl_next = 1:length(pl_range)

        pl = pl_range2(pl_next);
        pl_plot = zeros(N,1);    % = 10 during packet loss, 0 otherwise

        %
        % remove 1 packet burst from input signal every second
        %
        for k = Fs:Fs:N
            x(k-pl+1:k) = zeros(pl,1);
            pl_plot(k-pl+1:k) = 10;
        end

        %---------------------------------------------------------------------
        % Echo delay ne signal
        %---------------------------------------------------------------------

        d = zeros(N,1);       % echo delayed fe signal, (*not* observable)
        for k = 1:N
           for i = 0:length(b)-1
               if k-i > 0
                   d(k) = d(k) + b(i+1)*x(k-i);
               end
           end
        end

        r = s + d;                        % ne signal + fe echo (observable

        %---------------------------------------------------------------------
        % Initialise variables / set parameters
        %---------------------------------------------------------------------
        % Dual-H NLMS
        mu = 1;                  % NLMS step size
        wON = zeros(L, 1);       % ONline adaptive filter weights
        wOFF = zeros(L, 1);      % OFFline adaptive filter weights
        yON = zeros(N,1);        % ONline adaptive filter output
        yOFF = zeros(N,1);       % OFFline adaptive filter otput
        eON  = zeros(1, N);      % ONline residual error
        eOFF  = zeros(1, N);     % OFFline residual error
        delta = 0.000001;        % NLMS/ERLE constant to avoid division by

        % NLMS
        mu2 = 1;                 % NLMS step size
        wNLMS = zeros(L, 1);     % NLMS adaptive filter weights
        yNLMS = zeros(N,1);      % NLMS adaptive filter otput (estimate of
        eNLMS  = zeros(1, N);    % NLMS residual error

        % RLS
        lambda = 0.9;            % RLS forgetting factor
        % RLS variables
        wRLS = zeros(L, 1);      % RLS filter weights
        x_filter = zeros(L, 1); % input signal in filter
        P = eye(L);              % inverse input correlation matrix
        int= zeros(L, 1);        % intermediate calculation step = P(n-1)*
```

```
gain = zeros(L, 1);       % gain vector
yRLS = zeros(N,1);        % RLS adaptive filter output
eRLS  = zeros(1, N);      % RLS residual error signal

% ERLE variables
ERLE_L = L;               % order of ERLE calculation vectors
ERLE_ONdh = 0;            % current ONline filter (ERLE estimate)
ERLE_OFFdh = 0;           % current OFFline filter (ERLE estimate)
dp = 0;                   % power of d in ERLE vector
xp = 0;                   % power of x in ERLE vector
epON = 0;                 % power of eON in ERLE vector
epOFF = 0;                % power of eOFF in ERLE vector
epNLMS = 0;               % power of eNLMS in ERLE vector
epRLS = 0;                % power of eRLS in ERLE vector
ERLEdh_best = 0;          % best ERLE (dual-H estimate) found so fa

% Plotting variables - otherwise not necessary for simulation
WON = zeros(L, N);        % saves the ONline adaptive weights for p
WOFF = zeros(L, N);       % saves the OFFline adaptive weights for
ERLE_ON = zeros(N,1);     % saves the ONline ERLE for plotting
ERLE_OFF = zeros(N,1);    % saves the OFFline ERLE for plotting
ERLE_NLMS = zeros(N,1);   % saves the NLMS ERLE for plotting
ERLE_RLS = zeros(N,1);    % saves the RLS ERLE for plotting
ERLEdh_best_plot = zeros(N,1);  % saves the current best ERLE for
dt_plot = zeros(N,1);     % double-talk flag (0 if fe only, 1 if dt
%--------------------------------------------------------------------

for k = 1:N
    %
    % RLS
    %
    x_filter(1)=x(k);
    int = P*x_filter;

    % 1. calculate gain vector
    gain = (1/(lambda+dot(x_filter, int)))*int;

    % 2. calculate estimation error
    yRLS(k) = 0; % adaptive filter output
    for n = 0:L-1
        if( (k-n) > 0)
            yRLS(k) = yRLS(k) + (wRLS(n+1) * x(k-n));
        end
    end
    eRLS(k) = r(k)-yRLS(k);

    % 3. update filter weights
    wRLS = wRLS + gain*eRLS(k);

    % 4. update inverse matrix
    P = (1/lambda)*(P - gain*((x_filter')*P));

    % get next input vector
    for j=L:-1:2
        x_filter(j)=x_filter(j-1);
    end

    %
    % NLMS Dual-H, NLMS
    %
```

```
% calculate the ONline and OFFline filter output:
yON(k) = 0;
yOFF(k) = 0;
yNLMS(k) = 0;
for n = 0:L-1
    if( (k-n) > 0)
        yOFF(k) = yOFF(k) + (wOFF(n+1) * x(k-n));
        yNLMS(k) = yNLMS(k) + (wNLMS(n+1) * x(k-n));
        yON(k) = yON(k) + (wON(n+1) * x(k-n));
    end
end

% calculate the ONline and OFFline error:
% error sig = r(sig+echo) - y(est echo)
% = est of sig
eON(k) = r(k) - yON(k);
eOFF(k) = r(k) - yOFF(k);
eNLMS(k) = r(k) - yNLMS(k);

% find power of d and e in ERLE buffer(for ERLE calculation b
% adds next value and subtracts last value rather than recomp
% whole buffer each time
if k > ERLE_L
    xp = xp + x(k)*x(k) - x(k-ERLE_L)*x(k-ERLE_L);
    dp = dp + d(k)*d(k) - d(k-ERLE_L)*d(k-ERLE_L);
    epON = epON + eON(k)*eON(k) - eON(k-ERLE_L)*eON(k-ERLE_L);
    epOFF = epOFF + eOFF(k)*eOFF(k) - eOFF(k-ERLE_L)*eOFF(k-E
    epNLMS = epNLMS + eNLMS(k)*eNLMS(k) - eNLMS(k-ERLE_L)*eNLM
    epRLS = epRLS + eRLS(k)*eRLS(k) - eRLS(k-ERLE_L)*eRLS(k-E
else
    xp = xp + x(k)*x(k);
    dp = dp + d(k)*d(k);
    epON = epON + eON(k)*eON(k);
    epOFF = epOFF + eOFF(k)*eOFF(k);
    epNLMS = epNLMS + eNLMS(k)*eNLMS(k);
    epRLS = epRLS + eRLS(k)*eRLS(k);
end

% calculate true ERLE (dB) for plotting
ERLE_ON(k) = 10*log10(dp / (epON - s(k) + delta));
ERLE_OFF(k) = 10*log10(dp / (epOFF - s(k) + delta));
ERLE_NLMS(k) = 10*log10(dp / (epNLMS - s(k) + delta));
ERLE_RLS(k) = 10*log10(dp / (epRLS - s(k) + delta));

% calculate estimated ERLE (dB) for dual-H operation
ERLE_ONdh = 10*log10(dp / (epON + delta));
ERLE_OFFdh = 10*log10(dp / (epOFF + delta));

% update the OFFline adaptive filter coeff (Dual-H NLMS, NLMS
for n = 0:L-1
    if( (k-n) > 0)
        wOFF(n+1) = wOFF(n+1) + mu / (xp + delta)*eOFF(k)*x(k-
        wNLMS(n+1) = wNLMS(n+1) + mu2 / (xp + delta)*eNLMS(k)*
    end
end

% update the ONline filter coeff with the OFFline coeff if th
% OFFline ERLE is larger than the best ERLE found so far
if (k < 10*L)
```

```
        wON = wOFF;
        ERLEdh_best = ERLE_OFFdh;
    elseif (ERLE_OFFdh >= ERLEdh_best)
        wON = wOFF;
        ERLEdh_best = ERLE_OFFdh;
    end

    % update best ERLE if ONline ERLE is larger
    if (ERLE_ONdh >= ERLEdh_best) && (k>ERLE_L)
        ERLEdh_best = ERLE_ONdh;
    end

    % update the OFFline filter coeff with the ONline coeff if the
    % ONline ERLE is larger than the OFFline ERLE
    if (ERLE_ONdh > (ERLE_OFFdh+3)) && (k>ERLE_L)
        wOFF = wON;
    end

    % update best ERLE plotting variable
    ERLEdh_best_plot(k) = ERLEdh_best;

    % reduce ERLEdh_best at a rate of 5dB/s
    ERLEdh_best = ERLEdh_best - 5/Fs;

    % update plotting variables
    WON(:,k) = wON;
    WOFF(:,k) = wOFF;

    if (abs(d(k)) > delta) && (abs(s(k)) > delta)  % double talk
        dt_plot(k) = 20;
    else              % not double talk
        dt_plot(k) = 0;
    end
end
%------------------------------------------------------------------------
%
% calculate plotting data
%
pl_plot_DH(pl_next,:) = [pl_range(pl_next), max(real(ERLE_ON))];
pl_plot_NLMS(pl_next,:) = [pl_range(pl_next), max(real(ERLE_NLMS)
pl_plot_RLS(pl_next,:) = [pl_range(pl_next), max(real(ERLE_RLS))];

% smooth output using moving average
for i=1:N-500
    ERLE_ON(i) = mean(ERLE_ON(i:i+500));
    ERLE_NLMS(i) = mean(ERLE_NLMS(i:i+500));
    ERLE_RLS(i) = mean(ERLE_RLS(i:i+500));
end
%------------------------------------------------------------------------
figure(pl_next+10*next)
subplot(3,1,1);
plot(abs(eON - s'));
ylim([0 1])
ylabel('|Error|');
legend(['Dual-H_NLMS, \mu =',num2str(mu)],1);
%------------------------------------------------------------------------
subplot(3,1,2);
plot(abs(eNLMS - s'));
ylim([0 1])
ylabel('|Error|');
```

```
        legend(['NLMS, \mu = ',num2str(mu2)],1);
        %--------------------------------------------------------------------------------
        subplot(3,1,3);
        plot(abs(eRLS - s'));
        ylim([0 1])
        ylabel('|Error|');
        xlabel('Sample Number');
        legend(['RLS, \lambda = ',num2str(lambda)],1);

        % Put a main title above the subplot titles
        main_title = ['|Error| plot dropped packets: Burst size = ',...
            num2str(pl_range(pl_next)), ...
            ' packets, Adaptive Filter Order = ',num2str(L)];
        mtit(main_title, 'xoff',-.1,'yoff',.025, 'fontsize', 14);
    end
    figure(pl_next+1+10*next)
    plot(1:N, [abs(ERLE_ON), abs(ERLE_NLMS), abs(ERLE_RLS), pl_plot]);
    title(['ERLE during packet loss, Adaptive Filter Order = ',num2str(L)]
    xlabel('Sample Number','fontsize', 12);
    ylabel('ERLE (dB)','fontsize', 12);
    legend(['Dual-H NLMS, \mu = ',num2str(mu)],['NLMS, \mu = ',num2str(mu2)]
      ['RLS, \lambda = ',num2str(lambda)],'Packet loss: ON=10, OFF=0',4);
end
figure(pl_next+2+10*next)
    plot(pl_plot_DH(:,1), pl_plot_DH(:,2),'x', pl_plot_NLMS(:,1), pl_plot_N
        '*',pl_plot_RLS(:,1), pl_plot_RLS(:,2),'+','LineWidth',2,...
            'MarkerSize',10);
    title(['Max ERLE vs Packet loss burst size'],...
        'fontsize', 14);
    xlabel('Burst size','fontsize', 12);
    ylabel('Max ERLE (dB)','fontsize', 12);
    legend(['Dual-H NLMS, \mu = ',num2str(mu)],['NLMS, \mu = ',num2str(mu2)]
      ['RLS, \lambda = ',num2str(lambda)],3);

disp('#######################################################################
disp('Experiment finished!')
fprintf( 1, 'Save figures if necessary then press ENTER to run a new exper
pause
```

B.9 The exp8_combo.m MATLAB function

The function exp8_combo.m runs Experiment 8 (see 5.7.8) and is called by the main script file for the AEC simulation, aec_sim.m.

Listing B.9: The AEC Experiment 8 function file..

```
function [] = exp8_combo()
% exp8_combo.m
%
% * Experiment #8 = Combined adverse conditions
```

```
%
% * Function file called by AEC simulator (aec_sim.m)
%
% * Measures the echo cancellation performance of the Dual-H NLMS filter
%   using the full range of VoIP conditions.
%
% * Creates plots of |Error| and ERLE vs Sample Number
%
% * Experiment designed as part of the final year engineering project 'Ec
%   Cancellation in VoIP' for ENG4111/2 University of Southern Queensland
%
% * Adapted from adechosp.m by J.Leis
%
% * Room impulse response filter created using rir.m and fconv.m
%   (Copyright  2003 Stephen G. McGovern)
%
% * The main script aec.sim.m requires the following input sound files in
%   the working directory:
%   karl10s_mp2_8_dec.wav    karl10s_mp2_16_dec.wav,
%   karl10s_mp2_32_dec.wav   karl10s_mp2_64_dec.wav,
%   karl10s_8kHz_8bit.wav    ricky10s_8kHz_8bit.wav
%   karl10s_8kHz_8bit_mulaw.wav
%
% * Requires the following helper functions in the working directory:
%   rir.m, fconv.m and mtit.m
%
% Shane Kmita, Oct 2011

%------------------------------------------------------------------
% Input sound vectors
%------------------------------------------------------------------
% Far-End (fe) signal
[x Fs] = wavread('karl10s_8kHz_8bit_mulaw.wav');

% Near-End (ne) signal
[s Fs] = wavread('ricky10s_8kHz_8bit.wav');

N = min(length(x), length(s));
x = x(1:N);
s = s(1:N);

% Level shift fe, ne to 80% maximum
x = 0.8*x/(max(abs(x)));
s = 0.8*s/(max(abs(s)));

%------------------------------------------------------------------
% Create ne room impulse response
%------------------------------------------------------------------

rm=[3 3 3];          % room dimensions [L W H] in metres
mic=[2.5 4 0.9];     % mic position
src=[2.5 4 1.9];     % source position
r=-0.5;              % reflection coefficient (-1<r<1)
n=24;
b1=rir(Fs, mic, n, r, rm, src);

L = 500;   % adaptive filter order

tdelay = 10;         % one-way tail circuit delay (ms)
noise = 0.01;        % noise multiplier (percentage of full scale)
```

```
pl = 5;              % number of packets in dropped packet burst

tdelay_samp = floor(tdelay*Fs/1000);        % delay in samples
b = [zeros(tdelay_samp,1); b1];             % add delay to rir

%
% add ne noise
%
s1 = randn(N,1);              % N Gaussian white noise samples
s1 = s1/(max(abs(s1)));       % level shift ne noise to 100% maximum
s1 = noise*s1;                % reduce noise level according to multiplier
s = s1 + s;                   % add noise to ne signal

%
% dropped packet burst
%
vp = 20;             % voice payload (ms) (160 default for G.711)
br = 64000;          % bitrate (bps) (64000 for G.711)
bd = 8;              % bit depth (8 for G.711)
vpp = vp/1000*br/bd; % voice payload (samples per packet)
pl = floor(pl*vpp); % number of samples in dropped packet burst

%
% remove 1 packet burst from input signal every second
%

for k = Fs:Fs:N
    x(k-pl+1:k) = zeros(pl,1);
    pl_plot(k-pl+1:k) = 10;
end

%-----------------------------------------------------------------------
% Echo delay ne signal
%-----------------------------------------------------------------------

d = zeros(N,1);      % echo delayed fe signal, (*not* observable)
for k = 1:N
   for i = 0:length(b)-1
       if k-i > 0
           d(k) = d(k) + b(i+1)*x(k-i);
       end
    end
end

r = s + d;                          % ne signal + fe echo (observable)
SNR = 10*log10( sum(d.*d) / (sum(s1.*s1)+0.0000001) );
%-----------------------------------------------------------------------
% Initialise variables / set parameters
%-----------------------------------------------------------------------
% Dual-H NLMS
mu = 1;                  % NLMS step size
wON = zeros(L, 1);       % ONline adaptive filter weights
wOFF = zeros(L, 1);      % OFFline adaptive filter weights
yON = zeros(N,1);        % ONline adaptive filter output
yOFF = zeros(N,1);       % OFFline adaptive filter otput
eON  = zeros(1, N);      % ONline residual error
eOFF  = zeros(1, N);     % OFFline residual error
delta = 0.000001;        % NLMS/ERLE constant to avoid division by 0

% ERLE variables
ERLE_L = L;              % order of ERLE calculation vectors
```

```
ERLE_ONdh = 0;           % current ONline filter (ERLE estimate)
ERLE_OFFdh = 0;          % current OFFline filter (ERLE estimate)
dp = 0;                  % power of d in ERLE vector
xp = 0;                  % power of x in ERLE vector
epON = 0;                % power of eON in ERLE vector
epOFF = 0;               % power of eOFF in ERLE vector
ERLEdh_best = 0;         % best ERLE (dual-H estimate) found so far

% Plotting variables - otherwise not necessary for simulation
WON = zeros(L, N);       % saves the ONline adaptive weights for plotting
WOFF = zeros(L, N);      % saves the OFFline adaptive weights for plotting
ERLE_ON = zeros(N,1);    % saves the ONline ERLE for plotting
ERLE_OFF = zeros(N,1);   % saves the OFFline ERLE for plotting
ERLEdh_best_plot = zeros(N,1);  % saves the current best ERLE for plottin
dt_plot = zeros(N,1);    % double-talk flag (0 if fe only, 1 if dt)
%------------------------------------------------------------------------

%------------------------------------------------------------------------
% AEC Simulation
%------------------------------------------------------------------------

for k = 1:N

    %
    % NLMS Dual-H
    %

    % calculate the ONline and OFFline filter output:
    yON(k) = 0;
    yOFF(k) = 0;

    for n = 0:L-1
        if( (k-n) > 0)
            yOFF(k) = yOFF(k) + (wOFF(n+1) * x(k-n));

            yON(k) = yON(k) + (wON(n+1) * x(k-n));
        end
    end

    % calculate the ONline and OFFline error:
    % error sig = r(sig+echo) - y(est echo)
    % = est of sig
    eON(k) = r(k) - yON(k);
    eOFF(k) = r(k) - yOFF(k);

    % find power of d and e in ERLE buffer(for ERLE calculation below)
    % adds next value and subtracts last value rather than recomputing
    % whole buffer each time
    if k > ERLE_L
        xp = xp + x(k)*x(k) - x(k-ERLE_L)*x(k-ERLE_L);
        dp = dp + d(k)*d(k) - d(k-ERLE_L)*d(k-ERLE_L);
        epON = epON + eON(k)*eON(k) - eON(k-ERLE_L)*eON(k-ERLE_L);
        epOFF = epOFF + eOFF(k)*eOFF(k) - eOFF(k-ERLE_L)*eOFF(k-ERLE_L);

    else
        xp = xp + x(k)*x(k);
        dp = dp + d(k)*d(k);
        epON = epON + eON(k)*eON(k);
        epOFF = epOFF + eOFF(k)*eOFF(k);

    end
    % calculate true ERLE (dB) for plotting
    ERLE_ON(k) = 10*log10(dp / (epON - s(k) + delta));
    ERLE_OFF(k) = 10*log10(dp / (epOFF - s(k) + delta));

    % calculate estimated ERLE (dB) for dual-H operation
    ERLE_ONdh = 10*log10(dp / (epON + delta));
    ERLE_OFFdh = 10*log10(dp / (epOFF + delta));

    % update the OFFline adaptive filter coeff (Dual-H NLMS, NLMS)
    for n = 0:L-1
        if( (k-n) > 0)
            wOFF(n+1) = wOFF(n+1) + mu / (xp + delta)*eOFF(k)*x(k-n);
        end
    end

    % update the ONline filter coeff with the OFFline coeff if the
    % OFFline ERLE is larger than the best ERLE found so far
    if (k < 10*L)
        wON = wOFF;
        ERLEdh_best = ERLE_OFFdh;
    elseif (ERLE_OFFdh >= ERLEdh_best)
        wON = wOFF;
        ERLEdh_best = ERLE_OFFdh;
    end

    % update best ERLE if ONline ERLE is larger
    if (ERLE_ONdh >= ERLEdh_best) && (k>ERLE_L)
        ERLEdh_best = ERLE_ONdh;
    end

    % update the OFFline filter coeff with the ONline coeff if the
    % ONline ERLE is larger than the OFFline ERLE
    if (ERLE_ONdh > (ERLE_OFFdh+3)) && (k>ERLE_L)
        wOFF = wON;
    end

    % update best ERLE plotting variable
    ERLEdh_best_plot(k) = ERLEdh_best;

    % reduce ERLEdh_best at a rate of 5dB/s
    ERLEdh_best = ERLEdh_best - 5/Fs;

    % update plotting variables
    WON(:,k) = wON;
    WOFF(:,k) = wOFF;

    if (abs(d(k)) > delta) && (abs(s(k)) > delta)  % double talk
        dt_plot(k) = 20;
    else            % not double talk
        dt_plot(k) = 0;
    end
end
%-----------------------------------------------------------------------
%
% calculate plotting data
%

% smooth output using moving average
for i=1:N-500
    ERLE_ON(i) = mean(ERLE_ON(i:i+500));
    ERLE_OFF(i) = mean(ERLE_OFF(i:i+500));
end
%-------------------------------------------------------------------------
figure(1)
plot(abs(eON - s'));
ylim([0 1])
ylabel('|Error|');
legend(['Dual-H␣NLMS,␣\mu␣=',num2str(mu)],1);
title(['|Error|␣plot,␣Adaptive␣Filter␣Order␣=␣',num2str(L)]);
%-------------------------------------------------------------------------
figure(2)
plot(1:N, [abs(ERLE_ON), abs(ERLE_OFF), abs(ERLEdh_best_plot)]);
title(['ERLE␣vs␣Sample␣Number,␣Adaptive␣Filter␣Order␣=␣',num2str(L)],'fon
xlabel('Sample␣Number','fontsize', 12);
ylabel('ERLE␣(dB)','fontsize', 12);
legend('ONline␣filter','OFFline␣filter','Best␣ERLE',4);
%-------------------------------------------------------------------------

disp('##########################################################################
disp('Experiment␣finished!')
fprintf( 1, 'Save␣figures␣if␣necessary␣then␣press␣ENTER␣to␣run␣a␣new␣expe
pause
```