Cancellation of Acoustic Reverberation Using Adaptive Filters

by

Joseph T. Khalife

Center for Communications and Signal Processing

Department of Electrical and Computer Engineering
North Carolina State University

December 1985

CCSP-TR-85/18

ABSTRACT

KHALIFE, JOSEPH T. Cancellation of Acoustic Reverberation Using Adaptive Filters. (Under the direction of Dr. S. T. Alexander)

The hollow sound of speech is one of the main problems limiting the consumer acceptance of the hands-free telephone (speakerphone). The hollow sound is the result of the microphone picking up not only the speech coming directly from the talker but reflections from walls, floor and furniture as well. The speech signal $s(n)$ picked up by the microphone is the result of many reflections each of which contributes a waveform similar in shape to the original speech signal $x(n)$. However, since all the waveforms are overlapping it is extremely difficult to recover the original speech signal.

An adaptive filter based on the method of predictive deconvolution is derived and used to remove the reverberation from reverberant speech signals. In order to improve the performance of the dereverberation filter, the incorporation of a relatively simple pitch tracker is investigated.

The two algorithms used to update the coefficients of the adaptive filter are the Least Mean Squares algorithm (LMS) and the Fast Transversal Filter algorithm (FTF). The LMS is one of the simplest algorithms; however, one disadvantage of the use of LMS is its slow convergence time. The FTF algorithm overcomes the convergence problem but requires more arithmetic computations.

As a metric of the degree of cancellation the signal to noise ratio of the processed speech is computed and compared to that of the reverberant speech signal. Results of computer simulations verify the ability of the proposed dereverberation filter to reduce reverberation. The incorporation of a relatively simple pitch tracker slightly improved the performance of the filter. Results also show that in stationary conditions more reverberations were removed using the FTF filter. However, in nonstationary cases the FTF filter is shown to be more subject to divergence than the LMS filter.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1. General

In general, whenever a conversation takes place some kind of echo is generated. Speech waves are reflected by floors, walls, furniture and other neighboring objects. Also, any electric impedance mismatch in the transmission channel causes the speech signal to be reflected. If the energy of the reflected signal can return to the transmitting point, it will be added to, and interfere with any other signal being received there. For a speech signal being reflected back to the talker, the effects depend on the strength of the reflected signal and on the time delay. For a very short time delay the echo is not noticeable and the effect is the same as enhancement of the speech by the instantaneous echo (sidetone); a somewhat longer delay produces a hollow sound of the sidetone, and a long delay yields to distinct echoes which could be very annoying to the talker and could under certain conditions completely disrupt the conversation.

1.2. Problem Description

The use of a simple hands-free telephone set (speakerphone) introduces some degradations not encountered with the handset usage. In the case of a handset telephone the microphone and the speaker are so close to the head of the talker that performance is largely independent of the surrounding conditions. However, with the speakerphone room conditions can be a controling factor. When the microphone is at some distance from the talker, the talker's speech reaches the microphone by several paths. One path is the direct line from the talker to the microphone. In addition sound spreads out in the room and is

reflected back to the microphone from walls, floor, furniture and other solid objects. The listener at the other end is very conscious of the presence of reflections since they produce a hollow effect commonly called reverberation, which is very much like talking in a barrel. This hollow sound of speech is one of the main problems limiting the consumer acceptance of the hands-free telephone. The speech signal picked up by the microphone is the result of many speech reflections, each of which is very similar in waveform shape to the original (nonreverberant) speech signal. However, since all the waveforms are overlapping, it is extremely difficult to directly recover the original speech signal.

Several methods of reducing the effects of reverberation have previously been introduced. The amplitude of the reflections can be reduced by treating the room with sound absorbing material. However, this is a very expensive and not very practical approach if hands-free telephones are to be used in varying locations. A completely different approach is to remove the effects of reverberation by signal processing methods. Center clipping, a process that removes signals of small amplitudes while leaving signals of large amplitudes unaffected, has been used to remove echoes with long delays [1]. One disadvantage of the center clipping method is that it results in harmonic distortions evident in the discontinuities of the clipped waveform. Early arriving echoes have been reduced or cancelled in some cases by the use of two or more microphones [2]. Since the microphones are at different positions, nulls resulting from an early echo are at different frequencies. The outputs of the microphones are combined in such a way as to reduce the effects of the frequency nulls. A different method of solving the reverberation problem is to to extract the essential features of speech from the reverberant signal and from it reconstruct a nonreverberant signal [3]. Speech with no reverberation can be reconstructed using this approach, although the speech is not yet of telephone quality.

The object of the thesis is to present a new method of solving the reverberation problem. The method of predictive deconvolution has been applied with great success for the deconvolution of seismograms. An adaptive filter, based on the method of predictive deconvolution, is derived and applied to reverberant speech signals in order to remove reverberation. Two different adaptive algorithms are used to update the coefficients of the adaptive filter. These are the least mean square (LMS) and the fast transversal filter (FTF).

1.3. Thesis Organization

In the next chapter, a general explanation of the electric and acoustic echo generation processes is presented. Several echo control methods are summarized.

Chapter 3 consists of the derivation of a mathematical model to simulate the acoustic reverberation in a small room. The coupling between the talker and the microphone is discussed and modeled.

The derivation of an adaptive filter to cancel reverberation is presented in chapter 4. The derivation is based on the method of predictive deconvolution. Also in this chapter the incorporation of a pitch tracker is investigated.

Chapter 5 introduces the adaptive filtering techniques and summarizes the least mean square and the fast transversal filter algorithms. It also explains the advantages and disadvantages of the LMS algorithm over the FTF algorithm.

In chapter 6, computer simulations of the adaptive dereverberation filter are provided. The LMS and the FTF are the two adaptive algorithms used to update the filter coefficients.

In chapter 7 a comparison of the performance of the LMS and FTF algorithms in cancelling reverberation is presented. Summary, conclusions and recommendations are provided.

Chapter 2

ECHO GENERATION AND CONTROL

This chapter describes the electric and the acoustic echo generation process. It also presents several methods of controlling electric echoes as well as acoustic echoes caused by the microphone/loudspeaker coupling.

2.1. Electrically Produced Echoes

We will refer to electric echoes as those echoes generated in telephone circuits due to electrical impedance mismatches. Electric echoes with delays sufficiently long to disturb a conversation are observed only on long-distance telephone connections. In a typical long-distance telephone circuit, every telephone set in a given geographical area is connected to a central office by a two-wire line. This two-wire line, known as the customer's loop, is used for communication in both directions, allowing considerable savings of wires and of local switching equipment. For circuits longer than about 35 miles, a separate two-wire path is necessary for each direction [4]. On all calls where such a separation of path is necessary, a device, called the hybrid, is used to connect the two-wire local circuit to the four-wire long-distance circuit. Ideally, a hybrid should pass all the signal on the incoming four-wire channel to the two-wire circuit, with no leakage into the outgoing four-wire channel. However, isolating the two-wire receive signal from the four-wire transmit path requires that the input impedance of the two-wire circuit be accurately matched by a balancing impedance. Unfortunately, the two-wire input impedance cannot be known with accuracy [5]. Typically, the hybrid is on the four-wire side of a switching office and can be connected at different times to any of the customer lines served by that office. These lines differ in lengths causing the input impedance to be highly variable. In most cases, there-

fore, due to impedance mismatch, the "in" side is coupled to the "out" side of the hybrid, thus giving rise to an echo.

2.2. Acoustic Echoes

Acoustic coupling and echoes give rise to a very annoying problem in teleconferencing and in " hands-free " telephony. A hands-free telephone set (speakerphone) contains a microphone and an output transducer (loudspeaker). Speech coupled from the talker to the microphone often produces a hollow and blurred sound commonly called reverberation, which is as though the talker was "speaking in a barrel" [5]. This distortion results from the microphone picking up not only the speech coming directly from the talker but reflections from walls, floor and furniture as well. Room reverberation consists of a series of echoes of the the original speech, delayed from a few milliseconds to several seconds. Earliest echoes are single reflections while later echoes are multiple reflections of speech. In a typical office, later reflections are attenuated by 60 dB after about one-third of a second [6]. This time is known as reverberation time. A more general discussion of the reverberation in a room will be presented in a following section. In addition to the reverberation problem, far end speech coming out of the loudspeaker is picked up by the microphone. Speech travels from the loudspeaker to the microphone by a direct path and by way of reflections and returns to the far end talker causing audible echoes to arise.

2.3. Echo Control

Several methods have previously been introduced and used for controlling echoes [7-10]. On short telephone circuits the " via net loss " method [4] produces very satisfactory performance. This method consists of introducing a loss, say L dB, in each direction resulting in the improvement of the signal to echo ratio by L dB. For long circuits, this

procedure cannot be used because it results in unacceptably low signal levels at the receiver.

2.3.1. Echo Suppression

For long circuits the echo suppression method has originally been used for echo control. An echo suppressor strongly reduces if not eliminates the echo by physically separating the transmission and reception channels. Echo suppressors are based on the dynamic selection of the transmission or reception mode, by means of voice-switched attenuators, according to the levels of incoming and outgoing speech signals; the unused channel is blocked while one party is talking. However, if both customers speak simultaneously, there is an obvious problem: Opening the switch to block echo will also interrupt the desired speech. Another problem with echo suppressors is the clipping they impart to speech, especially weak speech. Clipping is hard to avoid, because suppressors must recognize the onset of the desired speech in the presence of echo [7].

2.3.2. Echo Cancellation

For eliminating the inconveniences associated with the previous methods, alternative approaches have been based on the simulation of the echo path by means of an adaptive digital linear filter and on the subtraction of the signal produced by it from the returned signal [8-10]. This method is known by "adaptive echo cancellation " and is illustrated in figure 2.1. Adaptive echo cancellation is probably one of the most efficient approaches currently used to remove electric echoes generated at the hybrid. In addition, several experiments have confirmed that echo cancellers can successfully reduce the acoustic echo coupling between microphones and loudspeakers [11-12].

The simplest version of this method is to use an adaptive linear predictor to construct the echo replica. The adaptive linear predictor is frequently implemented in a transversal filter, and uses a delay line to hold the P most recent samples of the incoming speech signal. Each sample is weighted by a corresponding weighting coefficient. The weighted samples are summed to form an estimate of the echo. An error function, usually formed by subtracting this estimate from the outgoing echo return signal, is used to update the filter coefficients.

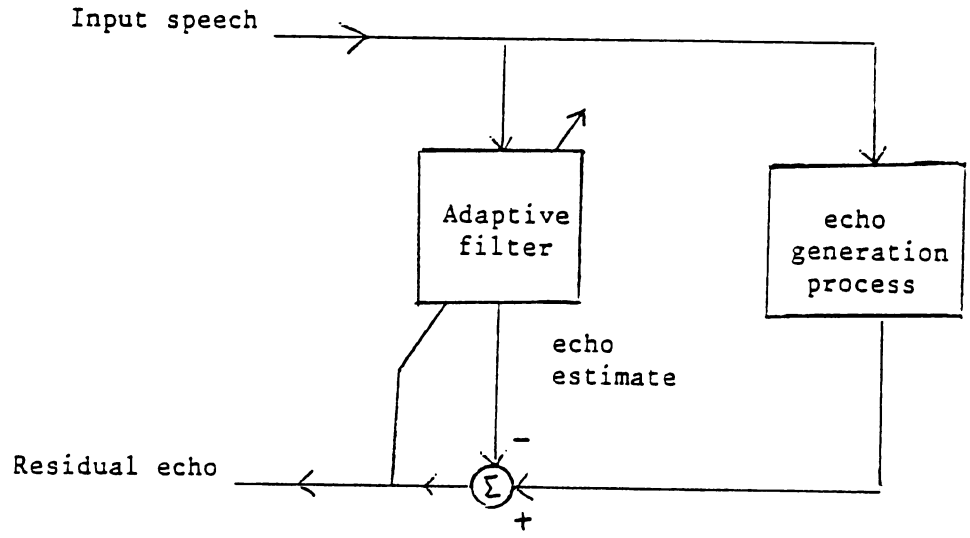


Fig. 2.1 : Echo cancellation

Chapter 3

ACOUSTIC ECHO PATH MODEL

3.1. Introduction

In this section, a mathematical model for acoustic reverberation for speakerphone environments will be derived. This will be of substantial benefit in simulation and analytical work to follow. As shown in figure 3.1, speech from the talker travels to the microphone by a direct path and by way of reflection from walls, floor and furniture.

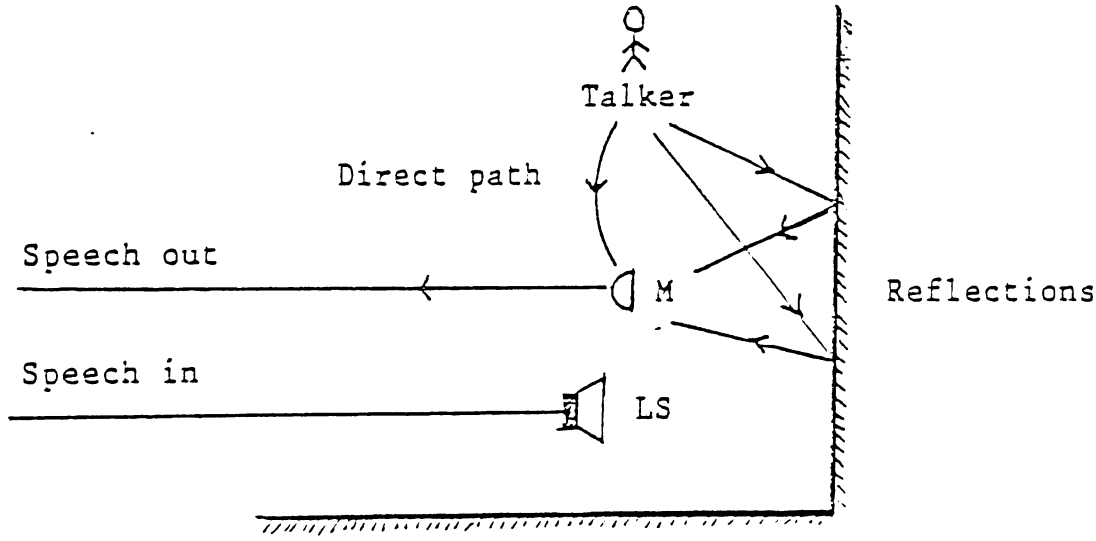


Fig. 3.1: Echo path in a typical office.

The speech picked up by the microphone has a hollow sound similar to the "speaking in a barrel" effect. An overall model of an office should represent the direct coupling as well as the indirect coupling (reverberation) of the speech signal.

3.2. Direct Model

The speed of sound at 20'C (68 deg. F) is known to be 344 m/sec (1127 ft/sec) which is a little over 1 ft/msec. Therefore, within a few msecs after the speech signal is produced by the talker, it is coupled to the microphone by means of the direct path. The direct coupling introduces some attenuation of the signal level represented by a factor $b<1$. Figure 3.2 represents the impulse response of the direct path assuming that the direct coupling occurs at time $t=0$.

Fig. 3.2 : Direct path impulse response.

3.3. Reverberation Model

The reverberation in a typical office is caused by the room's multiple reflective acoustical properties and is the resulting tendency for the sound level to persist in the

room after direct sound ceases. One measure of reverberation is the time T for the sound level to decay 60 dB from its steady state level after the sound has been turned off [13]. The commonly used Sabine reverberation equation is:

$$T = 0.049\, V/a \tag{3.1}$$

where V= room volume, and a = total sound absorption in sabins [13]. The total sound absorption is calculated from :

$$a = \sum \alpha_i S_i = \alpha S \tag{3.2}$$

where α_i = absorption coefficient (percentage of incident sound absorbed) for each surface S_i, S = total interior surface and α = average absorption coefficient. A reverberant sound field decays approximately in a logarithmic manner [13] and consequently, the sound level in decibels falls approximately linearly when the time scale is linear. In a typical office, the reverberation time is about 1/3 of a second [5], resulting in 180 dB/sec decay. To construct a preliminary model of reverberation in a room, it is assumed that only one reflection every 20 msec is reflected to the microphone. The number 20 msec is not a strict assignment but represents a path of about 10 feet to the main reflection surface and the 10 feet return path. Therefore, the distance traveled by the strongest echo before it reenters the communication link is about 20 feet. The level of the speech echoes in a typical office having a reverberation time of 1/3 of a second (decay of 180 db/sec) is shown in table 2.1. The parameter b is related to the attenuation coefficient of the reflecting surfaces. An example of calculating b is shown in the next section. An impulse response of room reverberation is shown in figure 3.3.

TIME (msecs)	samples (@ 8kHz)	DECAY dB	ECHO LEVEL
20	160	3.6	.6607 b
40	320	7.2	.4365 b
60	480	10.8	.2884 b
80	640	14.4	.1905 b
100	800	18.0	.1259 b
120	960	21.6	.0832 b
140	1120	25.2	.0549 b
160	1280	28.8	.0400 b
180	1440	32.4	.0240 b
200	1600	36.0	.0158 b
220	1760	39.6	.0147 b
240	1920	43.2	.0070 b
260	2080	46.8	.0046 b
280	2240	50.4	.0030 b
300	2400	54.0	.0020 b
320	2560	57.6	.0013 b

Table 2.1 : Level of Reverberations

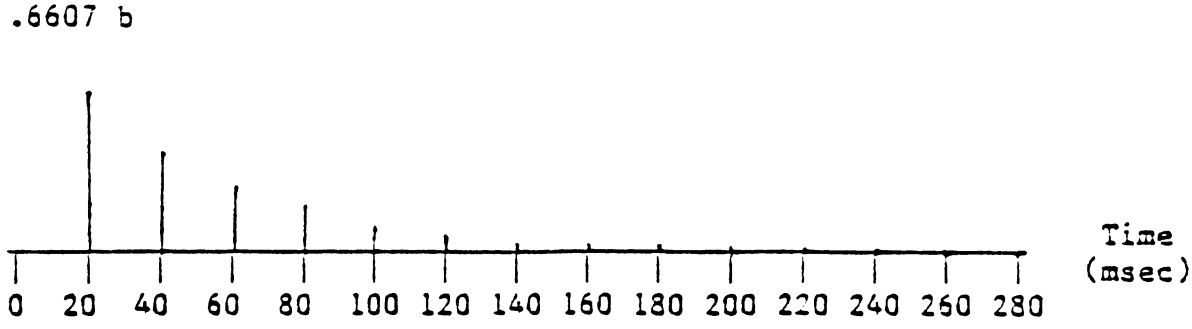


Fig. 3.3 : Reverberation impulse response.

3.4. Room Model

As an example of a typical office let us consider a rectangular room 20 ft long, 13 ft wide and 8 ft high (V = 2080 , S = 1048). From Sabine equation, the total sound

absorption coefficient that yields to a reverberation time of 1/3 s is :

$$a = .049\,V/T = .049(2080/.3333) = 306\ sabins = .29(1048) \tag{3.3}$$

The average absorption coefficient is 0.29, meaning that 29% of the sound is absorbed by the room. Therefore, the constant b was found by assuming a power gain of 0.71 for the room transfer function. The sum of the magnitude of all the pulses of the room impulse response is equal to 0.71; this led to b = 0.2417. Figure 3.4 represents an overall impulse response to simulate the acoustic proprieties of a typical office.

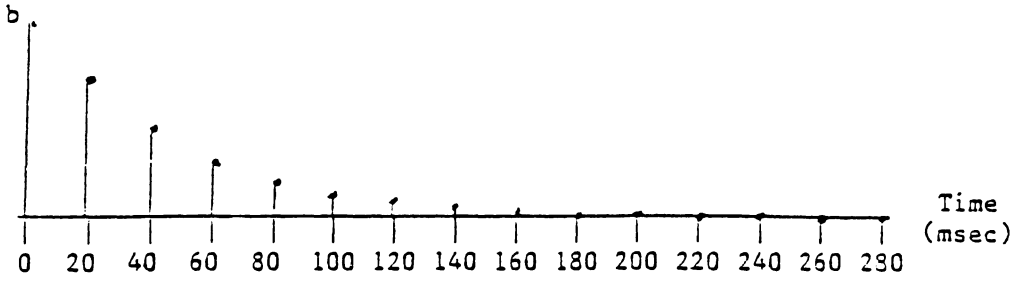


Fig. 3.4 : Overall impulse response.

A more accurate model to represent the sound behavior in a room is quite involved and beyond the scope of the present investigation. The image method is an advanced method commonly used to predict the sound behavior in a room [14]. A computer program for simulating small room acoustics using the image method is presented in [14], and the impulse response of a small room [14] based on the image method is shown in figure 3.5.

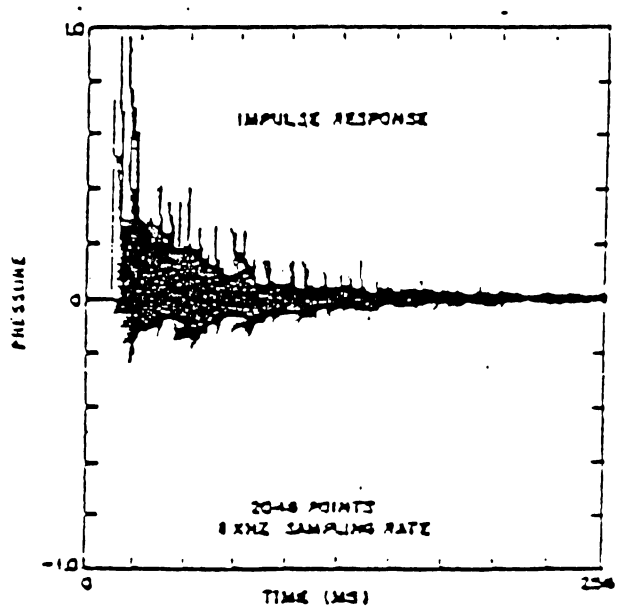


Fig. 3.5 : Room impulse response based on the image method [14].

The impulse response of Fig.5 shows some aspects of periodicity in the occurrence of strong responses. This is because the strongest echo reenters the communication link in a periodic manner.

3.5. Simulation of the Echo Generation Process

A computer program was written to simulate the echo generation process in a small room based on the model of figure 3.4. In the simulation only the first 7 impulses of figure 3.4 were considered. An actual speech signal d(n), shown in figure 3.6, was convolved with the first 7 pulses of the impulse response model (Fig. 3.4 ; b=1) to form the reverberant speech signal of figure 3.7.

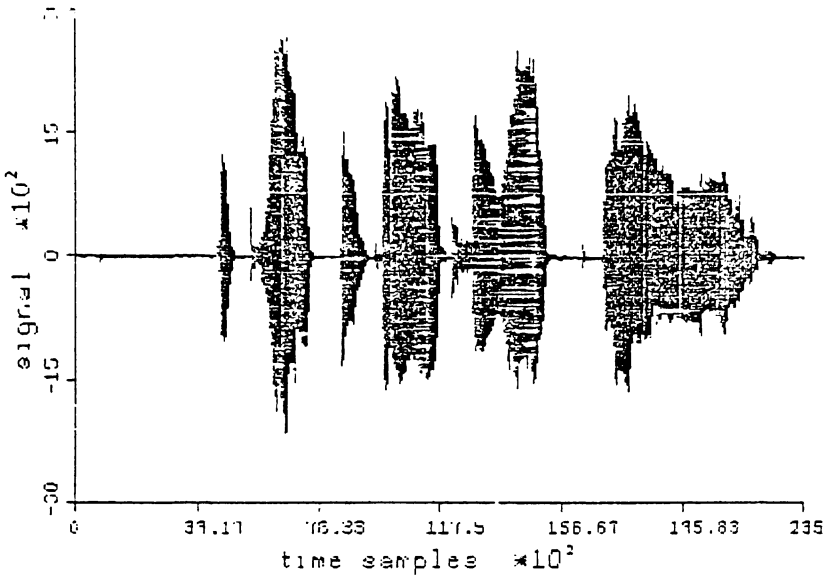


Fig. 3.6 : Clean speech signal

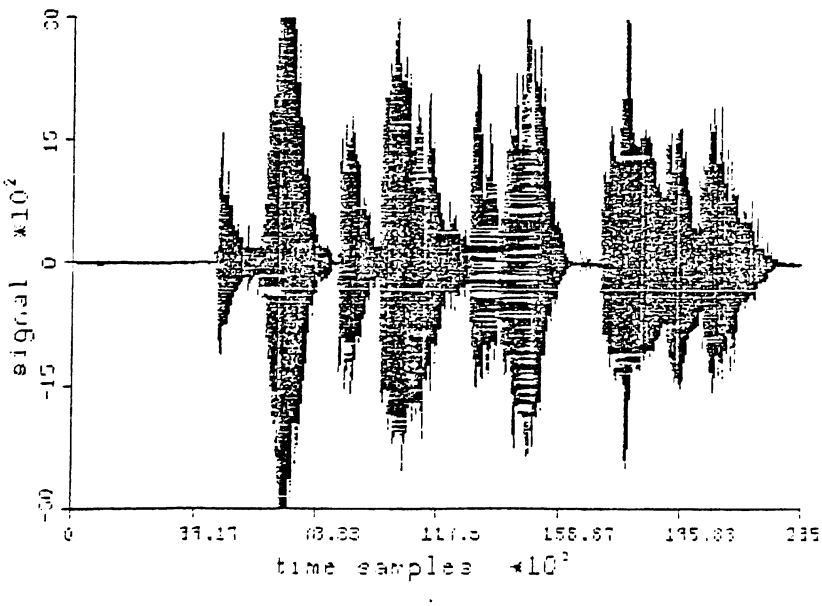


Fig. 3.7 : Reverberant speech signal

Chapter 4

Adaptive Dereverberation Filter

In this chapter an adaptive dereverberation filter is introduced. The purpose of the filter is to cancel reverberation from speech signal. Also, the incorporation of a pitch estimator within the dereverberation filter is investigated.

4.1. Filter structure

The hollow and blurred sound of speech is one of the main problems limiting the consumer acceptance of hands-free telephones. As was discussed earlier the hollow effect is the result of the microphone picking up not only the speech coming directly from the talker but the reflections from the walls, floor and furniture as well. The purpose of this chapter is to derive an adaptive filter able to cancel the echoes caused by reverberations.

The method of predictive deconvolution is a very effective method when used for the suppression of multiple reflection patterns. This method has found widespread use in the deconvolution of seismograms. One of its most gratifying successes has been its ability to ability to attenuate ghost reflections and reverberations [15].

4.1.1. Speech Reverberations

In the case of speakerphones or teleconferences, the speech signal $d(n)$ picked up by the microphone is the result of many reflections, each of which contributes a waveform of the shape of the original nonreverberant speech signal $s(n)$. However, since all the waveforms are overlapping it is extremely difficult if not impossible to directly recover the original clean speech signal. The available speech signal $d(n)$ is the result of convolving the clean speech waveform with a spike series that represents the room impulse response

similar to that shown in figure 3.4 . The timing of a spike represents the direct time of arrival of an echo and the amplitude represents the strength of that echo. A method similar to the method of predictive deconvolution can be applied to the reverberant speech signal in order to reduce reverberations. The ideal goal of the adaptive dereverberation filter is to remove the echoes caused by reflections but leave the original signal $s(n)$ intact.

4.1.2. Adaptive Filtering

In general, an adaptive filter involves the four signals illustrated in figure 4.1 : (1) the input signal $x(n)$ (used to compute the prediction), (2) the desired or reference signal $d(n)$, (3) the estimate $\hat{d}(n)$, and (4) the error signal $e(n)$.

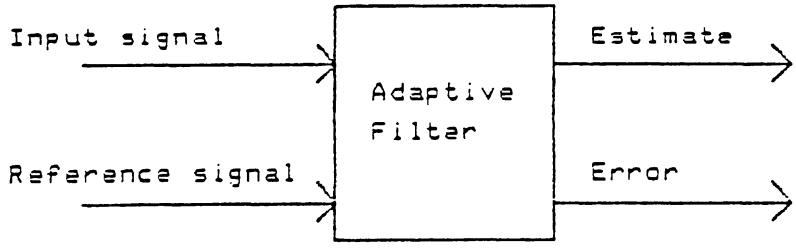


Figure 4.1 : Adaptive filter

The error signal $e(n)$ is used in order to adjust the filter parameters and seek the optimal performance according to a specific criterion. The linear structure most often used in adaptive filtering is the transversal filter shown in figure 4.2 . A delay line holds the N

most recent samples of the input signal $x(n)$. Each sample is first multiplied by a corresponding weighting coefficient, $w_i(n)$, and then all weighted samples are summed to form the estimate or prediction signal $\hat{d}(n)$.

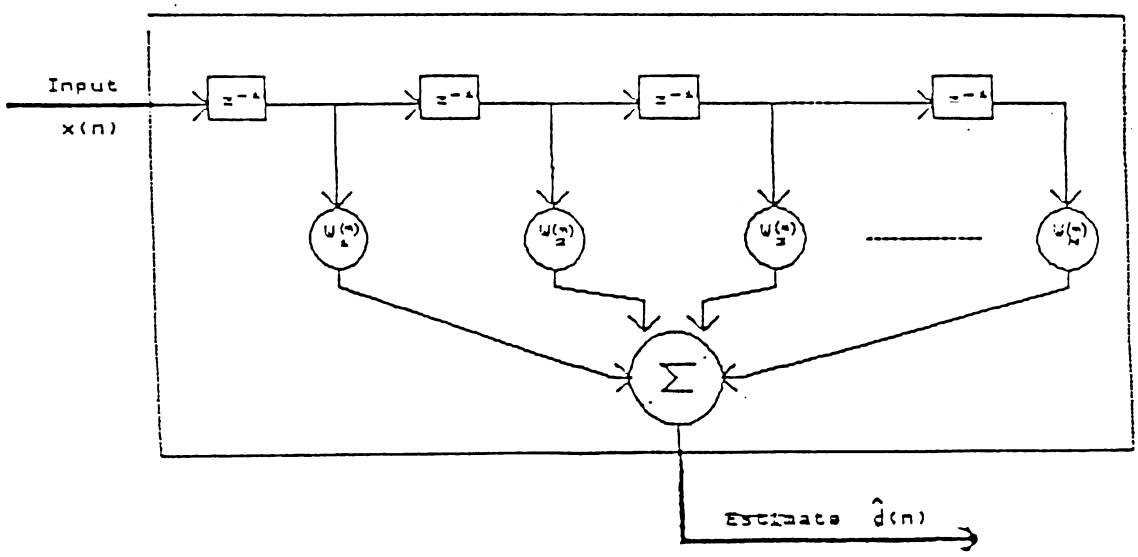


Figure 4.2 Transversal filter structure

The prediction signal $\hat{d}(n)$ is expressed in terms of the input signal $x(n)$ as follows:

$$\hat{d}(n) = \sum_{i=1}^{N} w_i(n)\, x(n-i) \tag{4.1}$$

Considerable research has been performed on a criterion for optimizing the filter coefficients $w_i(n)$. The two algorithms of interest in this thesis are the least mean square (LMS) and the fast transversal filter (FTF). In the following chapter a summary of these algorithms is presented.

4.1.3. Dereverberation of speech

The adaptive dereverberation filter is an adaptive filter used to reduce reverberations from a speech signal. In the case of adaptive dereverberation the desired signal $d(n)$ is the reverberant signal itself. This same signal is delayed by Δ samples in order to form the input signal $x(n)$. figure 4.3 represents an illustration of the dereverberation filter.

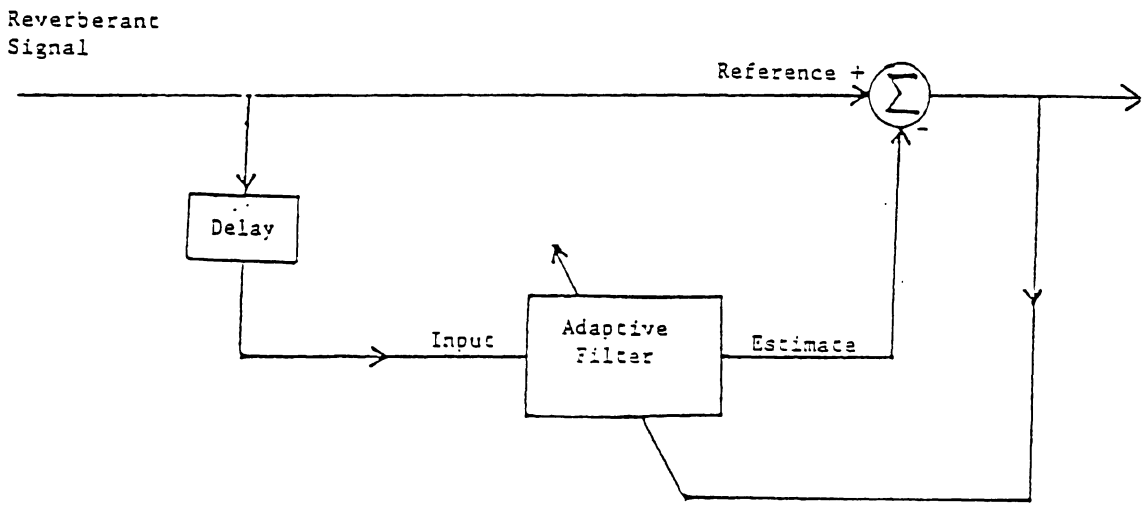


Figure 4.3 : Adaptive dereverberation filter

The basic principle of the dereverberation filter is that it predicts only the reverberant part of the speech signal. The error signal $e(n)$ is obtained by subtracting the prediction $\hat{d}(n)$ from the reference $d(n)$. Ideally, the estimate $\hat{d}(n)$ cancels all the reverberant part of the speech signal $d(n)$; therefore, the error signal $e(n)$ represents the nonreverberant part of $d(n)$. For the filter to be able to predict only the reverberant part of $d(n)$ and then cancel it, the following two hypotheses must hold:

First, the time of arrival of the strong echoes should be within the range of the filter, that is between $\Delta+1$ and $\Delta+N$. As it was mentioned earlier, the impulse response of a room was represented by a spike series. The timing of a spike represents the direct time of arrival of an echo. We denote by T_e the timing of the first spike which is the time of arrival of the first echo. For the filter to be able to predict and cancel the first reverberation, T_e should be between $\Delta+1$ and $\Delta+N$. When the time of arrival of the strong echoes is periodic, the filter is as well able to use the uncancelled first echo in order to predict the second echo component of the impulse response model. The second echo lies between $T_e+\Delta+1$ and $T_e+\Delta+N$. The same principle applies for predicting a third echo using the uncancelled second echo, and so on.

Second, the optimal performance of the filter occurs when input signal $x(n)$ is uncorrelated with the nonreverberant part of the reference signal $d(n)$. The least correlation possible between $x(n)$ and the nonreverberant part of $d(n)$ is desired. Most of the reverberation is cancelled when the correlation between the input signal and the reference signal is due mainly to the presence of an echo and not to the nature of speech. Otherwise, the filter tends to predict nonreverberant components of speech and cancel them as well.

A speech signal is by nature a very correlated signal. In the case of voiced speech most of the correlation occurs at the pitch or at multiples of the pitch period. The pitch period varies with time. As a result the autocorrelation function for higher lags is relatively small. The delay Δ is a very important factor used in order to decorrelate voiced speech and to reduce the correlation caused by the nature of speech. When a delay Δ is used, most of the correlation between the input signal and the reference signal is due to the existence of echoes. The estimate consists mainly of the reverberations and the error represents the nonreverberant signal.

As it was discussed earlier, the autocorrelation function of a speech signal is relatively small for higher lags. However, it might still be of a significant value at or around a multiple of the pitch period. In the case of adaptive dereverberation, this makes the prediction and then cancellation of nonreverberant components very possible. In the next section we will discuss how a pitch tracker can be used in order to deal with this problem and improve the performance of the dereverberation filter.

4.2. Incorporation of a Pitch Tracker

4.2.1. Pitch Estimation

Pitch detection or the measurements of long term periodicity is by itself a very important function and has a variety of applications. Several pitch tracking algorithms have been developed. The autocorrelation function (ACF) pitch detection algorithm is one method proven to be reliable and is easily implemented in hardware. As described in [16], the pitch can be estimated by finding the maximum in a recursively computed autocorrelation function estimate. The choice of the minimum lag and the maximum lag determines the range of pitch estimate of the detector. The *mth* autocorrelation lag at time sample n is recursively computed according to the following equation:

$$r_m(n) = \gamma\, r_{n-1}(n) + y(n)\, y(n-m) \tag{4.2}$$

The choice of γ determines the duration of the exponential window used for the ACF computations. The effective length of the window is approximately $1/1\text{-}\gamma$ samples. The ACF is next windowed in such a way that attenuates larger lags. This is done in order to help prevent " pitch doubling " . Once the weighted ACF values are determined the pitch is estimated to be equal to the value m_o corresponding to the largest weighted autocorre-

lation lag.

4.2.2. Constrained Dereverberation Filter

Speech sounds can be classified into different classes according to to their mode of excitation. Voiced sound is produced when quasi-periodic pulses of air excite the vocal track. A voiced speech signal is by its nature a highly correlated signal. Most of the correlation occurs at the pitch or at multiples of the pitch period. Figure 4.4 shows the autocorrelation function (ACF) of a segment of voiced speech.

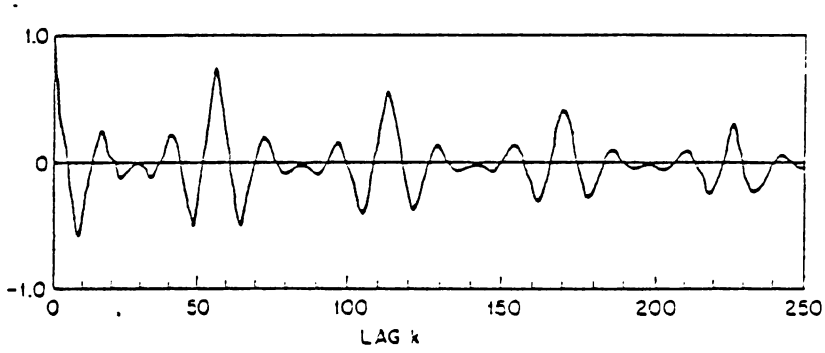


Figure 4.4 : ACF of a segment of voiced speech

We note that the ACF has many peaks; the peaks at the pitch period have the largest amplitudes. An adaptive filter can produce a very good prediction of a reference signal when this same signal is delayed by a pitch period or by a multiple of pitch period and used as input to the filter. In the case of adaptive dereverberation, the nonreverberant part of the speech signal might also be predicted very well, especially when the filter in question has a large processing window. To cope with this problem we propose a constrained adaptive filter. In general, an estimate consists of a weighted sum of the N most

recent samples of the input signal $x(n)$. The N most recent samples of $x(n)$ correspond to the samples of the reverberant signal that lie between $n-\Delta-N$ and $n-\Delta$. If any of these samples is near a pitch period or a multiple of pitch period, the sample will be disregarded and will not be used in the computation of the reverberation estimate. A pitch period estimate T_p is computed at every time sample n. The filter is constrained when the pitch period or a multiple of the pitch period is in the processing window; that is between Δ and $\Delta-N$. The input samples that might cause the prediction of the nonreverberant part are constrained to zero. The filter coefficients corresponding to these samples are saved and updated. When the pitch period estimate changes different samples of the input signal might be constrained to zero and disregarded in the computation of the estimate. A flowchart of the constrained adaptive filter is presented in Figure 4.5 .

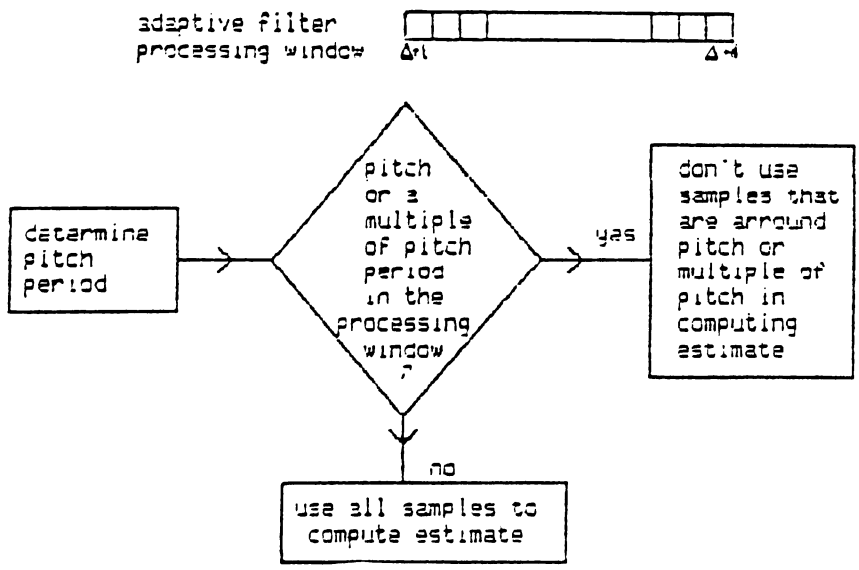


Figure 4.5 : Incorporation of a pitch tracker

To write a mathematical model for the constrained filter, the pitch period estimate is denoted by $T_p(n)$ and the number of samples that are constrained at each side of the pitch or multiple of pitch period is denoted by c . The estimate signal $\hat{d}(n)$ is then expressed in terms of the input signal $x(n)$ as follows:

(i) if $k\ T_p(n) \leq \Delta$ and $k\ T_p(n) > \Delta + N$ (k=1,2,3...)

$$\hat{d}(n) = \sum_{i=1}^{N} w_i(n)\ x(n-i) \tag{4.3a}$$

(ii) if $\Delta < k\ T_p(n) \leq \Delta + N$ (k=1,2,3...)

$$\hat{d}(n) = \sum_{i=1}^{N} w_i(n)\ x(n-i) \ - \ \sum_{i=a}^{b} w_i(n)\ x(n-i) \tag{4.3b}$$

Where $a = \Delta - k\ T_p(n) - c$ and $b = \Delta - k\ T_p(n) + c$.

The use of a constrained algorithm might improve, in most cases, the performance of the filter. The improvement in performance is explained by the fact that in computing the dereverberation estimate, the filter is disregarding the input samples which cause the non-reverberant part of the signal to be predicted. However, constraining the adaptive filter requires a larger number of arithmetic computations. First, the the ACF must be computed in order to estimate the pitch. In addition, two decision processes must take place. The first decision process takes place in order to determine the largest autocorrelation lag. The second is to determine whether or not the pitch or a multiple of the pitch period lie in the processing window of the filter.

Chapter 5

ADAPTIVE ALGORITHMS

5.1. Introduction to adaptive filters

Prediction filter design was originated by the pioneering work of Wiener [17] and is still currently a very strong and active area of research. When the impulse response of a filter does not change with time, fixed filters are designed and used. Adaptive filters have the ability to adjust their own parameters in order to obtain, according to a specific criterion, the optimal output. The least mean square (LMS) [18-19] algorithm is one of the simplest and most easily analyzed of the adaptive algorithms. In LMS, the mean square error is minimized, and the filter coefficients are adjusted recursively according to an approximation to the method of steepest descent. This algorithm requires only $2N$ multiplications per sample, N being the order of the filter. However, its convergence rate is relatively slow and depends on the eigenvalues of the input signal autocorrelation matrix.

Another approach is to consider minimizing a function of the actual acquired data without using any expectations or statistical information. In some cases, a weighted sum of the squared prediction errors is minimized leading to the technique of recursive least squares filters. Recursive least squares filters converge much faster than the LMS but require many more arithmetic computations. In order to reduce the computation complexity, alternate approaches to recursive least squares have been developed. Using vector space relations and projection techniques, Cioffi and Kailath derived an exact least squares adaptive filter known as the fast transversal filter (FTF) [20]. The FTF algorithm requires only $7N$ multiplications per time sample and produces an exact solution at every iteration. The LMS and the FTF are the two algorithms of interest in this thesis

and are summarized in the next sections.

5.2. The Least Mean Square Algorithm

Considerable research has been performed on the criterion for finding the optimal adaptive filter coefficients $w_i(n)$. The mean square error is one criterion that has widespread use. Under this criterion the adaptive filter coefficients are adjusted to minimize the mean square value of the prediction error. In this section, the derivation of the LMS algorithm, similar to that presented in [18], is summarized. It is desired to predict a reference signal $d(n)$ using an input signal $x(n)$. The prediction is denoted by $\hat{d}(n)$ and the error by $e(n)$. The prediction $\hat{d}(n)$ is a weighted sum of the N most recent input samples:

$$\hat{d}(n) = \sum_{i=1}^{N} w_i(n)\, x(n-i) \tag{5-1}$$

Note that in the case of of adaptive dereverberation, the input signal $x(n)$ is a delayed version of the reference signal $d(n)$:

$$x(n) = d(n-\Delta) \tag{5-2}$$

The vector form of equation (5-1) is :

$$\hat{d}(n) = \mathbf{w_N}^T(n)\, \mathbf{x_N}(n) \tag{5-3}$$

where

$$\mathbf{x_N}(n) = [x(n-1), x(n-2), \ldots, x(n-N)]^T$$

and

$$\mathbf{w_N}(n) = [w_1(n), w_2(n), \ldots, w_N(n)]^T$$

The prediction error is given by :

$$e(n) = d(n) - \mathbf{w_N}^T(n)\,\mathbf{x_N}(n) \tag{5-4}$$

The mean square error ϵ obtained by taking the expected value of the squared prediction error squared is :

$$\epsilon = E[e(n)^2] = E[x(n)^2] - 2\mathbf{w_N}(n)^T\mathbf{p_N} + \mathbf{w_N}(n)^T\mathbf{R_{NN}}\,\mathbf{w_N}(n) \tag{5-5}$$

where the vector $\mathbf{p_N}$ is the cross correlation between the desired response $d(n)$ and the data vector $\mathbf{x_N}(n)$, and $\mathbf{R_{NN}}$ is the input autocorrelation matrix.

$$\mathbf{p_N} = E[d(n)\,\mathbf{x_N}(n)]$$

$$\mathbf{R_{NN}} = E[\mathbf{x_N}(n)\,\mathbf{x_N}^T(n)]$$

It is clearly seen that the mean square error is a function of the filter coefficients vector $\mathbf{w_N}(n)$. This function can be pictured as a concave hyperparaboloidal surface that never goes negative. Referring to the mean square error as "bowl-shaped" it is desired to get the optimal filter coefficient vector $\mathbf{w_N}^*$ that yields the bottom of the bowl. A very common method used for this purpose is the gradient algorithm, which consists of taking the gradient of ϵ with respect to $\mathbf{w_N}(n)$, setting it to zero and solving for $\mathbf{w_N}^*$ [18]. The optimal filter coefficients obtained by setting the gradient of the mean square error to zero are:

$$\mathbf{w_N}^* = \mathbf{R_{NN}^{-1}P_N} \tag{5-6}$$

This equation is an exact solution of the generally called normal equation (Wiener-Hopf equation). It requires a priori knowledge of $\mathbf{p_N}$ and $\mathbf{R_{NN}}$ as well as an inversion of the matrix $\mathbf{R_{NN}}$. In the case of stationary data, the autocorrelation matrix $\mathbf{R_{NN}}$ and the crosscorrelation vector $\mathbf{p_N}$ can be estimated. However, many signals of interest (such as speech) do not have stationary statistics; hence extensive computation is required if the weights are to be continuously updated to their optimal value. In the very common case of unknown statistics, a more practical procedure uses an iterative method to find an

approximation to the optimal filter coefficients, and it does not require any correlation measurements or matrix inversion. According to this method, the next weight vector $\mathbf{w}_N(n+1)$ is equal to the present weight vector $\mathbf{w}_N(n)$ plus a change proportional to the negative of the mean square error gradient [18]:

$$\mathbf{w}_N(n+1) = \mathbf{w}_N(n) - \mu\nabla_w(\epsilon) \tag{5-7}$$

μ being the factor that controls the stability and the rate of convergence. The LMS algorithm simply approximates the mean square error by the square of the instantaneous error. An efficient estimate of the gradient can then be written as follows:

$$\nabla_W(\epsilon) = -2e(n)\mathbf{x}_N(n) \tag{5-8}$$

The final recursive formula for the LMS algorithm for computing the filter coefficients would then be:

$$\mathbf{w}_N(n+1) = \mathbf{w}_N(n) + 2\mu e(n)\mathbf{x}_N(n) \tag{5-9}$$

This algorithm is easily implemented in real time and it only requires $2N$ multiplications per time update. It has been shown that the mean coefficients provided by the LMS algorithm converge to the optimal solution and remains stable as long as μ is greater than zero and less than the reciprocal of the largest eigenvalue of the correlation matrix $\mathbf{R}_{NN}$ [19] :

$$0 \leq \mu \leq 1/\lambda_{max} \tag{5-10}$$

5.3. The Fast Transversal Filter Algorithm

In this section we present a summary of the fast transversal filter algorithm. The FTF is an exact recursive least squares filter based on vector space methods and projection techniques. It only requires $7N$ multiplication per iteration which, is a reduction of 30% compared to the fast Kalman algorithm of Falconer and Ljung [21]. In this summary

of the FTF, it is again desired to predict a reference (desired) signal $d(n)$ using an input signal $x(n)$. The estimate or prediction signal is denoted by $\hat{d}(n)$ and the error obtained at time n using a predictor computed at time i is denoted by $e(n|i)$. Four data vectors are defined as follows:

$$\mathbf{x}(n)=\begin{bmatrix} x(1) \\ x(2) \\ \cdot \\ \cdot \\ \cdot \\ x(n) \end{bmatrix}, \quad \mathbf{d}(n)=\begin{bmatrix} d(1) \\ d(2) \\ \cdot \\ \cdot \\ \cdot \\ d(n) \end{bmatrix}, \quad \hat{\mathbf{d}}(n)=\begin{bmatrix} \hat{d}(1) \\ \hat{d}(2) \\ \cdot \\ \cdot \\ \cdot \\ \hat{d}(n) \end{bmatrix}, \text{ and } \quad \mathbf{e}(n|n)=\begin{bmatrix} e(1|n) \\ e(2|n) \\ \cdot \\ \cdot \\ \cdot \\ e(n|n) \end{bmatrix} \tag{5-11}$$

Then

$$\mathbf{e}(n|n) = \mathbf{d}(n) - \hat{\mathbf{d}}(n) \tag{5-12}$$

and

$$\hat{\mathbf{d}}(n) = \mathbf{X}_N(n)\, \mathbf{w}_N(n) \tag{5-13}$$

where

$$\mathbf{X}_N(n) = \begin{bmatrix} x(1) & 0 & \cdots & 0 \\ x(2) & x(1) & \cdots & 0 \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ x(n) & x(n-1) & \cdots & x(n-N-1) \end{bmatrix} \tag{5-14}$$

The objective of the FTF algorithm is to obtain efficiently the set of filter coefficients $\mathbf{w}_N(n)$ that corresponds to the minimal cumulative squared error. The cumulative squared error $E(n)$ is defined as follows:

$$E(n) = \mathbf{e}^T(n|n)\, \mathbf{e}(n|n) \tag{5-15}$$

The exact least squares solution obtained by minimizing $E(n)$ with respect to $\mathbf{w}_N(n)$ [20] is:

$$\mathbf{w}_N{}^*(n) = [\mathbf{X}_N{}^T(n)\, \mathbf{X}_N(n)]^{-1}\, \mathbf{X}_N{}^T(n)\, \mathbf{d}(n) \;=\; \mathbf{K}_X(n)\, \mathbf{d}(n) \tag{5-16}$$

The matrix $\mathbf{K}_X(n)$ is called the transversal filter operator and is defined as follows:

$$\mathbf{K}_X(n) = [\mathbf{X}_N{}^T(n)\, \mathbf{X}_N(n)]^{-1}\, \mathbf{X}_N{}^T(n) \tag{5-17}$$

The estimate $\hat{\mathbf{d}}(n)$ can be written as:

$$\hat{\mathbf{d}}(n) = \mathbf{P}_{\mathbf{X}}(n)\, \mathbf{d}(n) \tag{5-18}$$

where

$$\mathbf{P}_{\mathbf{X}}(n) = \mathbf{X}_{\mathbf{N}}(n)[\mathbf{X}_{\mathbf{N}}{}^{T}(n)\, \mathbf{X}_{\mathbf{N}}(n)]^{-1}\, \mathbf{X}_{\mathbf{N}}{}^{T}(n) \tag{5-19}$$

The matrix $\mathbf{P}_{\mathbf{X}}(n)$ is commonly called the projection matrix since it generates the projection of the reference vector $\mathbf{d}(n)$ into the subspace spanned by the column vectors of $\mathbf{X}_{\mathbf{N}}(n)$. From the projection matrix and the identity matrix I the orthogonal projection matrix $\mathbf{P}_{\mathbf{X}}^{\perp}(n)$ is computed:

$$\mathbf{P}_{\mathbf{X}}^{\perp}(n) = I - \mathbf{P}_{\mathbf{X}}(n) \tag{5-20}$$

The orthogonal projection matrix generates the component of the reference vector $\mathbf{d}(n)$ orthogonal to the subspace spanned by the column vectors of $\mathbf{X}_{\mathbf{N}}(n)$. It can be shown [20] that :

$$\mathbf{e}(n|n) = \mathbf{P}_{\mathbf{X}}^{\perp}(n)\, \mathbf{d}(n) \tag{5-21}$$

The set of optimal filter coefficients corresponding to the optimal least squares filter changes with time. However, the matrix $\mathbf{X}_{\mathbf{N}}(n)$ grows systematically in a sense that the new filter can be computed using a knowledge of the old filter. The basis behind the FTF algorithm is to develop a recursive update for the filter operator and for the projection matrix. The development is better understood by the consideration of four separate transversal filters: the least squares projection filter $\mathbf{w}_{\mathbf{N}}(n)$, the forward prediction filter $\mathbf{f}_{\mathbf{N}}(n)$, the backward prediction filter $\mathbf{b}_{\mathbf{N}}(n)$ and the normalized gain filter $\mathbf{g}_{\mathbf{N}}(n)$. These quantities are represented as projections and then updated recursively in a simple manner. The following table represents a summary of the FTF algorithm [20]. The equations are listed in an order by which they can be efficiently implemented.

TABLE 5.1 : SUMMARY OF FTF ALGORITHM

i) Initialization

$$\mathbf{f}_N(0) = \mathbf{b}_N(0) = \mathbf{w}_N(0) = \mathbf{g}_N(0) = 0$$

$$\epsilon^f(0) = \epsilon^b(0) = [d(0)]^2$$

$$\gamma_N(0) = 1.$$

ii) Recursion

1. Compute least square error $e(n|n-1)$

$$e(n|n-1) = d(n) - \sum_{i=1}^{N} w_i(n-1)\, x(n-i) \tag{5-22}$$

2. Compute the forward prediction error $e^f(n|n-1)$

$$e^f(n|n-1) = x(n) - \sum_{i=1}^{N} f_i(n-1)\, x(n-i) \tag{5-23}$$

3. Update the forward prediction error $e^f(n|n)$

$$e^f(n|n) = \gamma_N(n-1)\, e(n|n-1) \tag{5-24}$$

4. Update the forward residual $\epsilon^f(n)$

$$\epsilon^f(n) = \epsilon^f(n-1) + e^f(n|n-1)\, e^f(n|n) \tag{5-25}$$

5. Update the forward filter coefficients $\mathbf{f}_N(n)$

$$\mathbf{f}_N(n) = \mathbf{f}_N(n-1) + e^f(n|n)\, \mathbf{g}_N(n-1) \tag{5-26}$$

6. Update the normalized gain filter coefficients $\mathbf{g}_N(n)$

$$\mathbf{g}_{N+1}(n) = \begin{bmatrix} 0 \\ \hline \mathbf{g}_N(n-1) \end{bmatrix} + \frac{e^f(n|n)}{\epsilon^f(n-1)} \begin{bmatrix} 1 \\ \hline -\mathbf{f}_N(n-1) \end{bmatrix} \tag{5-27}$$

$$e^b(n|n-1) = g_{N-1}(n)\, \epsilon^b(n-1) \tag{5-28}$$

$$\begin{bmatrix} \mathbf{g}_N(n) \\ \hline 0 \end{bmatrix} = \mathbf{g}_{N+1}(n) + g_{N+1}(n) \begin{bmatrix} \mathbf{b}_N(n-1) \\ \hline -1 \end{bmatrix} \tag{5-29}$$

7. Update the " angle update " $\gamma_N(n)$

$$\gamma_{N+1}(n) = \gamma_N(n-1) \frac{\epsilon^f(n-1)}{\epsilon^f(n)} \tag{5-30}$$

$$\gamma_N(n) = \left[1 - \gamma_{N+1}(n)\, g_{N+1}(n)\, e^b(n|n-1) \right]^{-1} \gamma_{N+1}(n) \tag{5-31}$$

8. Update backward prediction error $e^b(n|n)$

$$e^b(n|n) = \gamma_N(n)\, e^b(n|n-1) \tag{5-32}$$

9. Update the backward prediction residual $\epsilon^b(n)$

$$\epsilon^b(n) = \frac{\gamma_N(n)}{\gamma_{N+1}(n)} \epsilon^b(n-1) \tag{5-33}$$

10. Update the backward prediction filter coefficients $\mathbf{b}_N(n)$

$$\mathbf{b}_N(n) = \mathbf{b}_N(n-1) + e^b(n|n)\, \mathbf{g}_N(n) \tag{5-34}$$

11. Update the least squares error $e(n|n)$

$$e(n|n) = \gamma_N(n)\, e(n|n-1) \tag{5-35}$$

12. Update the least square filter coefficients $\mathbf{w}_N(n)$

$$\mathbf{w}_N(n) = \mathbf{w}_N(n-1) + e(n|n)\, \mathbf{g}_N(n) \tag{5-36}$$

5.4. LMS vs FTF

The least mean square algorithm is one of the easiest and most often used adaptive algorithms. The simplicity of the LMS is attributed to the fact that only $2N$ multiplications per iteration are required, N being the order of the filter. However the convergence rate of the LMS is relatively slow and is dependent on the strength of the eigenvalues of the input signal autocorrelation matrix. The convergence rate is especially slow when the

eigenvalues exhibit a wide spread such as in highly sinusoidal signal environments such as voiced speech. The dependence of the convergence rate on the statistics of the input signal have prompted researchers to develop alternative algorithms.

Substantial improvement in the rate of the convergence is obtained when using the fast transversal algorithm [20]. The FTF algorithm minimizes the sum of the squared errors and yields the exact optimal solution at every iteration. The improvement in convergence rate is obtained at a modest increase in computation. The FTF algorithm requires $7N$ multiplications per iteration . But, with the increasingly ingenious and inexpensive integrated hardware, an efficient and relatively cheap implementation of the FTF algorithm may well be possible. Thus, the FTF may soon become a compatible alternative of the LMS algorithm.

5.5. Real System Implementation

A relatively simple hardware and software implementation of the LMS adaptive filter was presented in [22]. The design is based on a ring organization of several TMS320 processors. All the processors execute the same machine instruction in synchrony (Single Instruction Multiple Data architecture). The proposed multiprocessor configuration behaves as a pipelined structure for updating the input vector, and then behaves as a parallel structure to apply the adaptation algorithm [22]. A design based on four TMS320 processors showed that at 8000 Hz sampling rate, 58 taps per processor can be accommodated [22].

For the FTF algorithm, a very rough estimation suggests that about $44N$ machine instructions per time iteration are required in order to update the filter coefficients. The

TMS320 instruction cycle is 200 nsec; thus, at 8000 Hz sampling rate 625 instructions per iteration can be handled. This suggests that an estimate of about 14 taps can be accommodated by every processor.

Chapter 6

SIMULATIONS

In this chapter, simulations of the adaptive dereverberation filter are presented. All experiments are conducted using the "C" language, on a Vax 11/780, under the Unix operating system. As a performance metric, the degree of cancellation represented by the signal to noise ratio (snr) is used. In all the simulations a clean nonreverberant signal $s(n)$ is available. The quantity $s(n)$ is not available in actual applications; however, in simulations, it is very useful for computing a measurement of the degree of cancellation. The clean signal $s(n)$ is convolved with an impulse response similar to the one developed in chapter 3. The result of the convolution is a reverberant signal denoted by $d(n)$. The signal $d(n)$ forms the input of the dereverberation filter. The output of the filter $e(n)$ is compared with the nonreverberant signal $s(n)$. The signal to noise ratio of the reverberant signal is computed as follows:

$$snr = 10 \log \frac{s(n)^2}{[d(n)-s(n)]^2} \tag{6-1}$$

To measure the amount of cancellation, the snr of the filter output is also computed and according to the following equation:

$$snr = 10 \log \frac{s(n)^2}{[e(n)-s(n)]^2} \tag{6-2}$$

Synthetic data as well as real speech are used for the simulations. The filter coefficients are updated by both the LMS algorithm and the FTF algorithm.

6.1. Unconstrained Filter

6.1.1. Simulation using synthetic data

Synthetic speech signals are generated by passing an impulse train through a second order IIR filter with poles $p_1 = 0.9e^{j\pi/3}$ and $p_2 = 0.9e^{-j\pi/3}$. The separation between the impulses corresponds to the pitch period. The sampling frequency is equal to 8000 Hz. Figure 6.1 represents a 1000 sample signal having a pitch period of 200. Only one echo at a delay of 60 samples is considered in order to form the reverberant signal of figure 6.2 . Even though the above case does not constitute a relevant representation of actual speakerphone situations, it does however provide us with a good illustration of the operation of the dereverberation filter. The reverberant signal of figure 6.2 is then passed through a one coefficient dereverberation filter ($\Delta = 59$). The output of the dereverberation filter is plotted for the LMS algorithm in figure 6.3 . It can be seen from the plots that most of the echo is cancelled. As it was explained in chapter 4, the prediction signal consists of only the reverberations. The output of the filter e(n) corresponds to the nonreverberant part of the signal. In fig 6.4, 200 points of the input, reference, estimate, and output signals are shown.

A closer representation of the actual situation is achieved when a signal with a pitch period of 57 is generated (figure 6.5). To form a reverberant signal we convolve the signal of figure 6.5 with the impulse response of figure 6.6 . The reverberant signal of figure 6.7 is then passed through a 5 coefficient dereverberation filter ($\Delta = 57$). For each block of 200 samples the signal to noise ratio is computed. In table 6.1, we present the snr of the reverberant signal and the snr of the output of the filter for both the LMS and the FTF algorithms. A plot of table 6.1 is shown in figure 6.8 . The optimal step size for the LMS

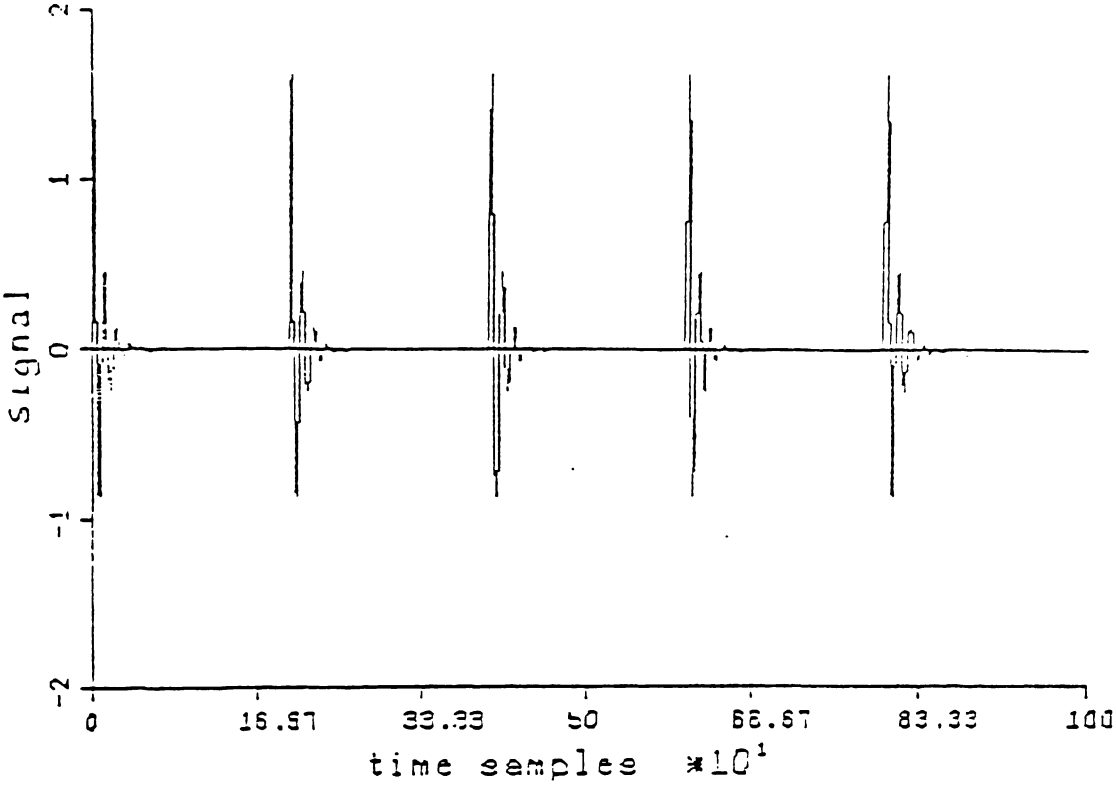


Fig. 1 : Clean Synthetic Signal (pitch period = 200)

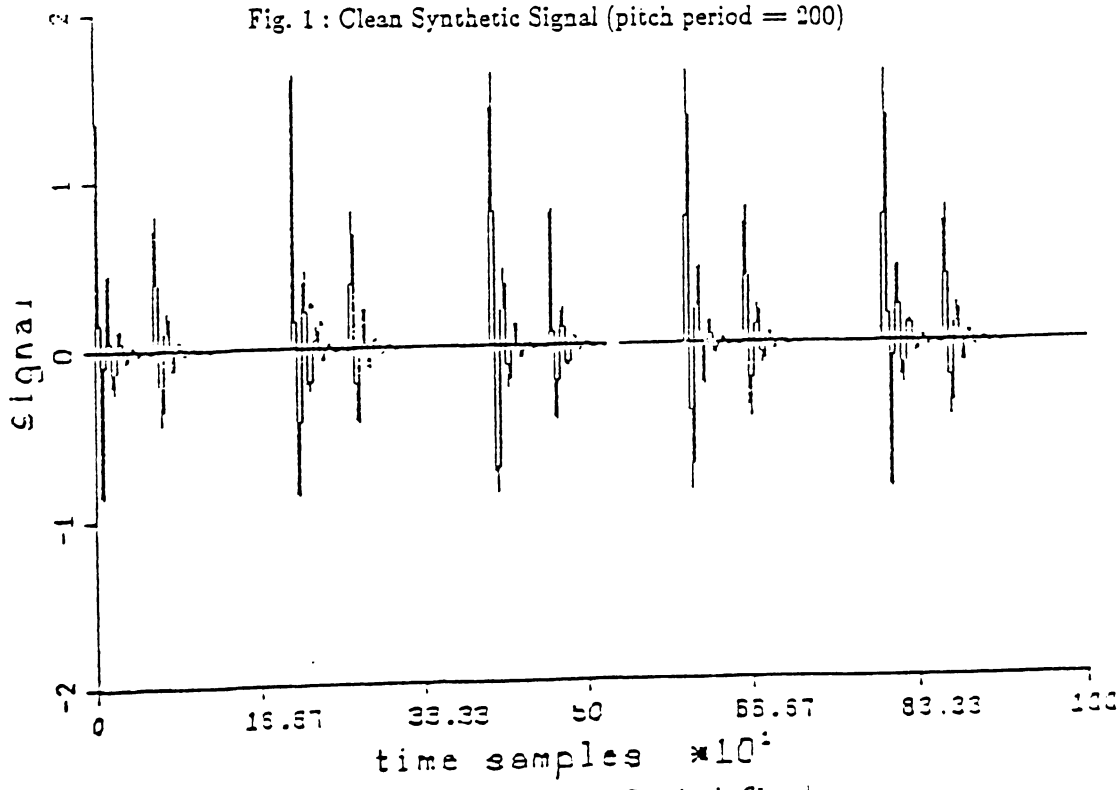


Fig. 2 : Reverberant Synthetic Signal

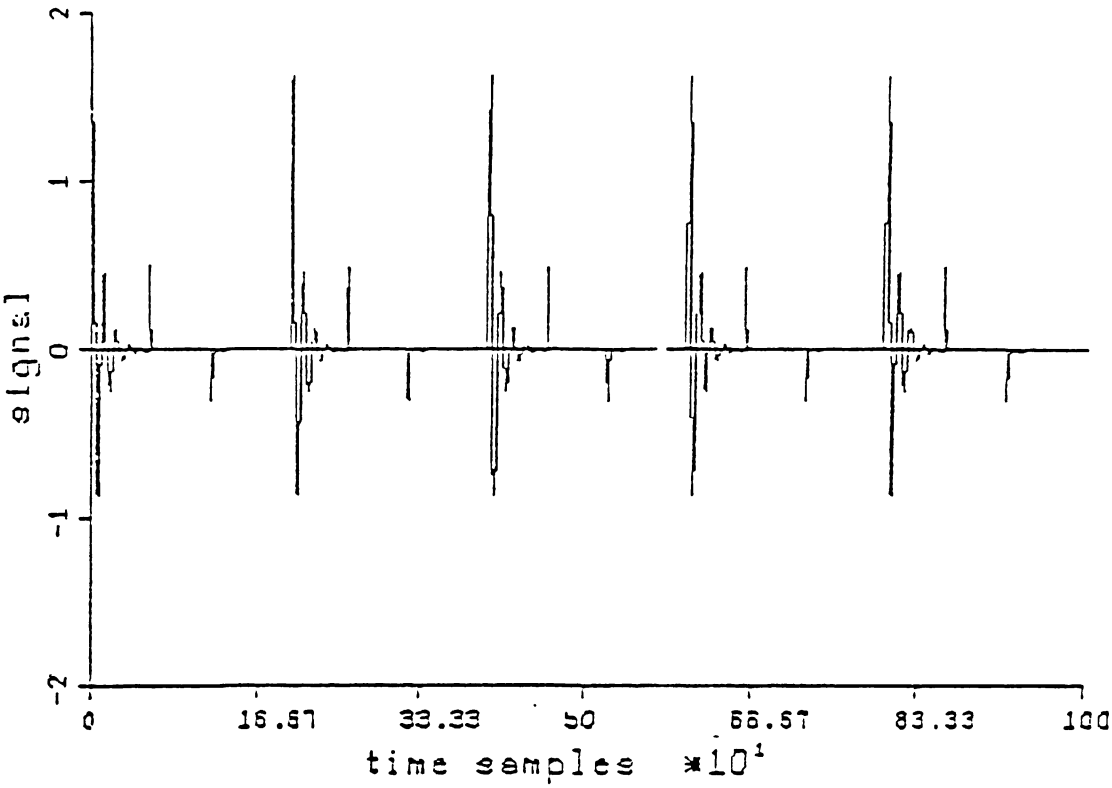


Fig. 3 : Processed Signal (LMS)

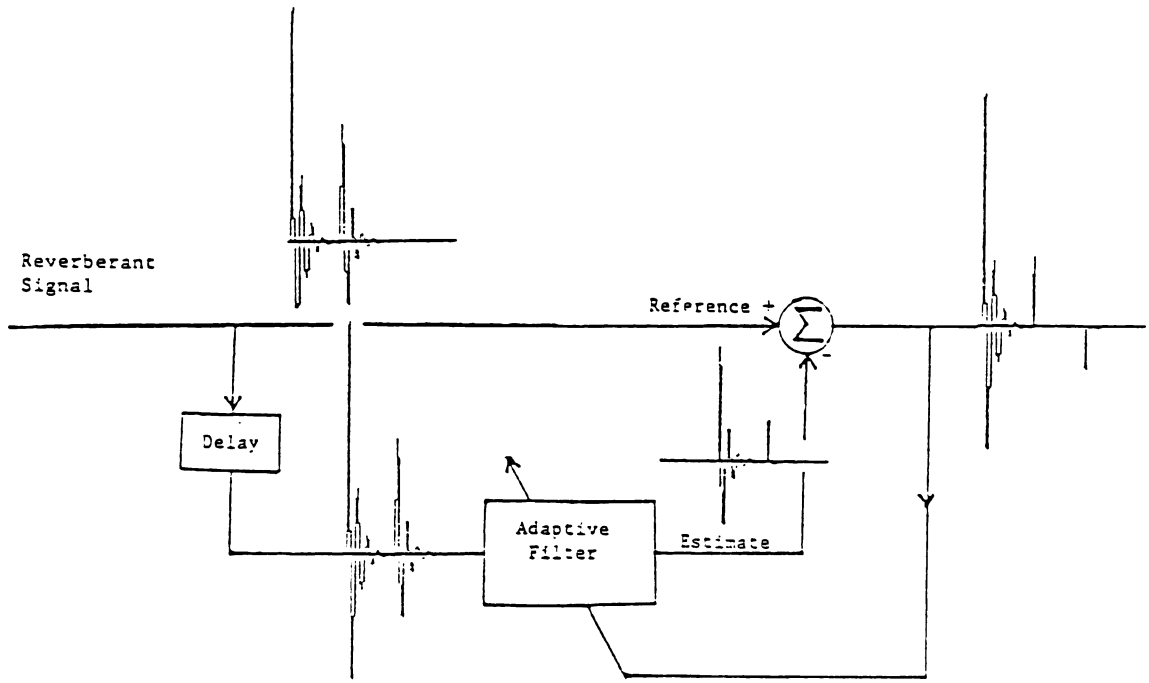


Fig. 4 : Illustration of Adaptive Dereverberation

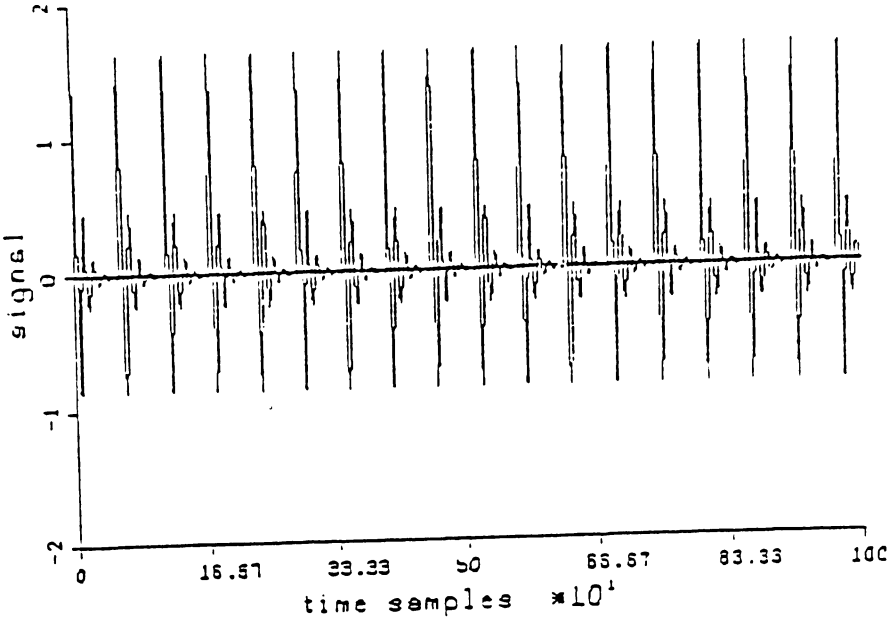


Fig. 5 : Clean Synthetic Signal (pitch period = 57)

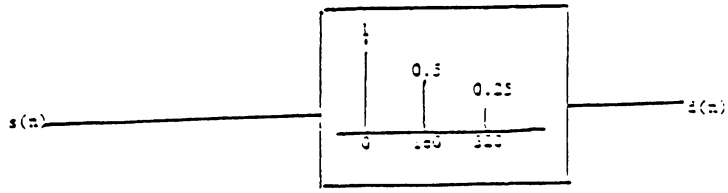


Fig. 6 : Impulse Response

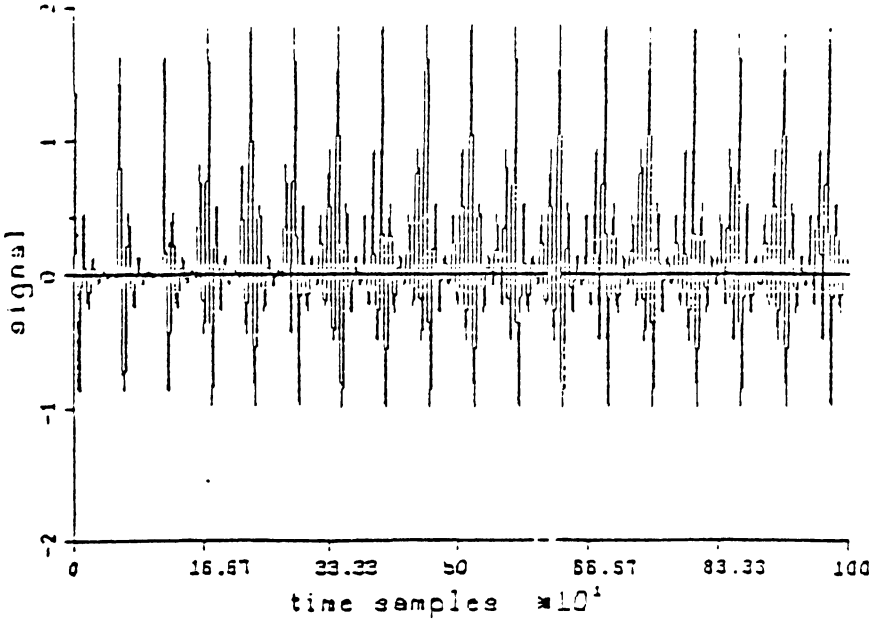


Fig. 7 : Reverberant Synthetic Signal

Block #	Signal to noise ratio in dB		
	Reverberant signal	processed with LMS	processed with FTF
1	11.94	12.78	19.85
2	4.31	11.22	11.13
3	5.09	11.86	10.65
4	3.79	10.04	9.87
5	4.82	10.37	10.57

Table 6.1 : Synthetic speech (order=5)

Fig. 8 : Synthetic Data snr (filter order = 5)

was equal to 0.001 . As shown in figure 6.8, the FTF converges much faster then the LMS. In fact for the case of the LMS almost no improvement in the snr was obtained for the first 200 points block. But after that, both the LMS and FTF converged to about the same solution giving almost identical processed signal snr.

The results of using a filter with 30 coefficients are presented in table 6.2 . A plot of table 6.2 is presented in figure 6.9 . Compared with the 5 coefficients filter of table 6.1, lower signal to noise ratios are obtained. For the LMS case some of the processed signal contains more degradation than the reverberant signal. The step size was 0.001 . If a larger step size is used, the snr of the processed signal becomes even smaller. The degradation for the LMS case can be attributed to the fact that the excess mean square error is proportional to the filter order: each filter coefficient fluctuates around the optimal mean and causes some misadjustment noise [18]. In a large order filter the noise adds up limiting the performance of the filter. The degradation can also be caused by the fact that the synthetic signal is a very correlated signal. This leads to the prediction and cancellation of some parts of the nonreverberant signal. In the case of real speech signals, the periodicity is less dominant, and as it will be shown in the next section, larger order filters can be used and still be able to obtain good performance.

6.1.2. Simulations using real speech

The waveform representation $s(n)$ of the speech segment " the pipe began to rust while new " is shown in figure 6.10 . In order to form a reverberant signal, the signal $s(n)$ is convolved with the impulse response of figure 6.11. The result of the convolution is denoted by $d(n)$ and shown in figure 6.12 . The reverberant signal $d(n)$ was played over the DSC digital acquisition and playback system. Audible reverberation was achieved,

Block #	Signal to noise ratio in dB		
	Reverberant signal	processed with LMS	processed with FTF
1	11.94	12.23	12.07
2	4.31	7.02	5.76
3	5.09	7.82	8.79
4	3.79	5.47	8.83
5	4.82	3.83	9.93

Table 6.2 : Synthetic speech (order=30)

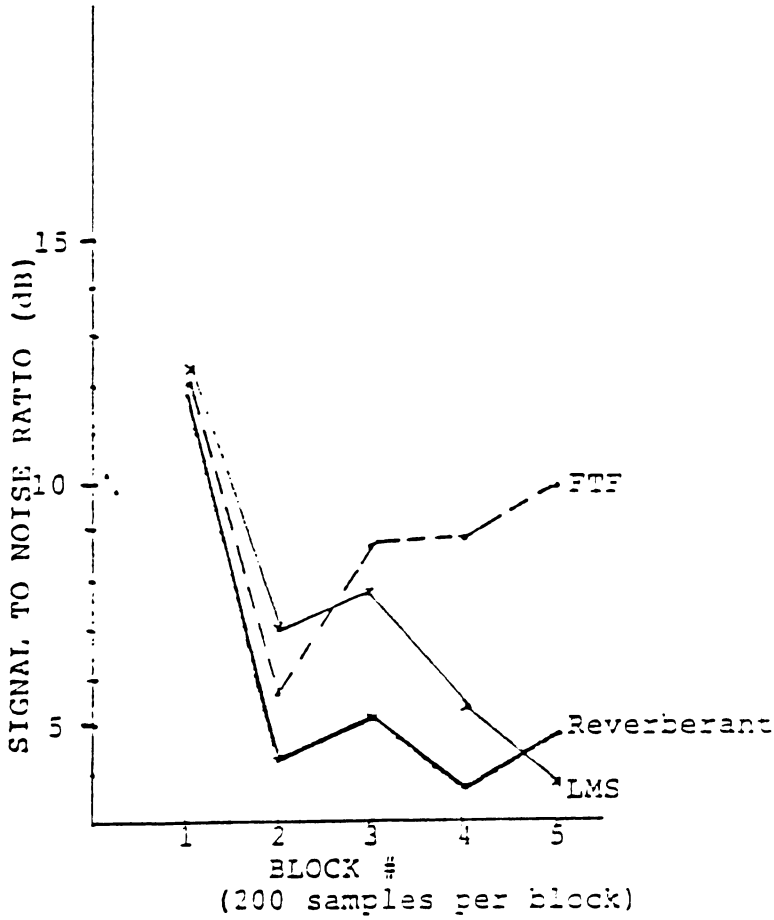


Fig. 9 : Synthetic Data snr (filter order = 30)

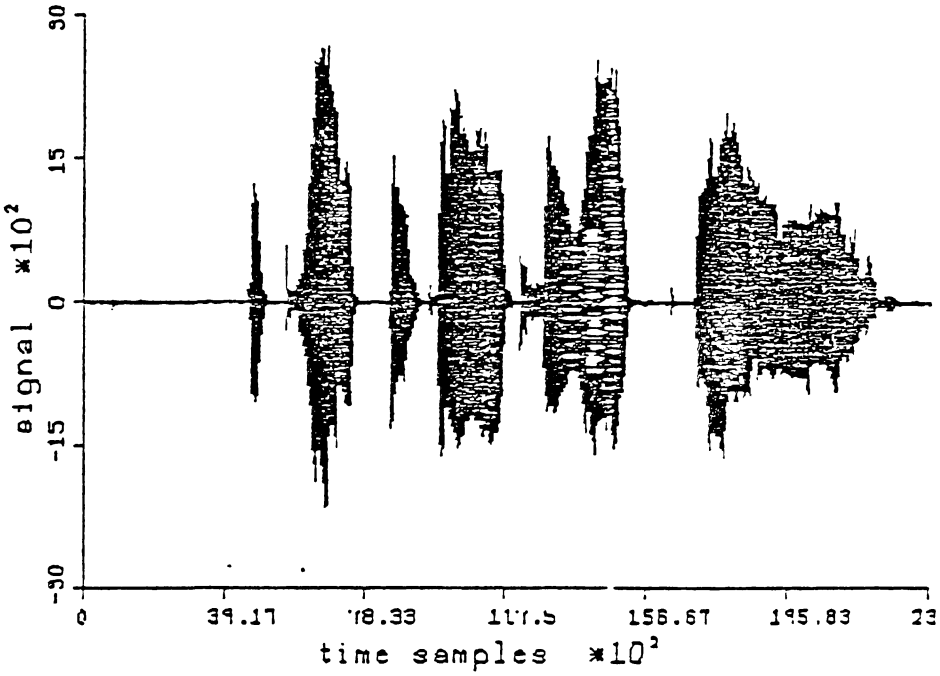


Fig. 10 : Real Speech Signal

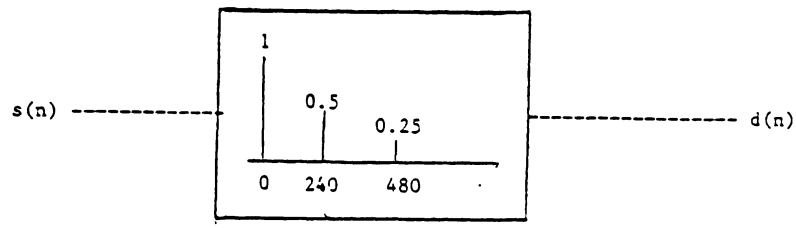


Fig. 11 : Impulse response

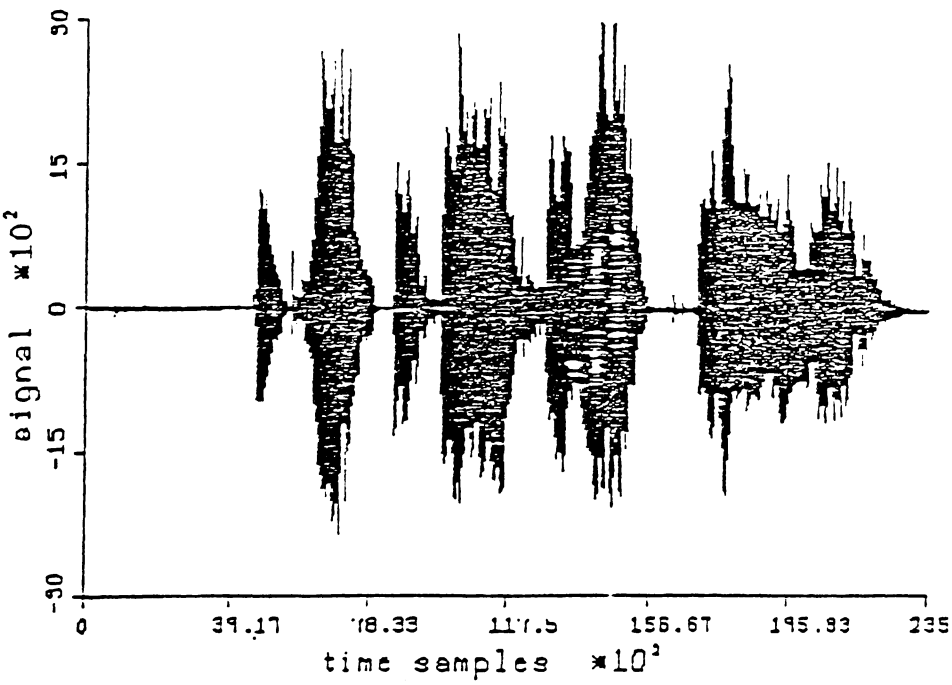


Fig. 12 : Reverberant Speech Signal

and the speech had a hollow sound as if the talker was speaking in a barrel. The reverberant signal $d(n)$ is then passed through a one coefficient dereverberation filter ($\Delta = 240$). The (optimal) step size used for the LMS is 10^{-9}. In figure 6.13 we show a 1000 points block of the nonreverberant waveform $d(n)$. The corresponding reverberant block is presented in figure 6.14. Processed blocks for both the LMS algorithm and the FTF algorithm are plotted in figures 6.15 and 6.16 respectively. Most of the reverberations are cancelled and the output blocks for both the LMS and the FTF cases are very similar to the clean nonreverberant block. The signal to noise ratio is computed for each block of 2000 samples. In table 6.3 the snr of the reverberant signal, of the processed with LMS signal and of the processed with FTF signal are presented and compared. The same results are plotted in figure 6.17 . Significant improvement in the snr is obtained. The LMS algorithm converges much slower than the FTF algorithm. The improvement in the quality of speech was demonstrated by playing the processed speech files over the digital acquisition and playback system. Audible improvement for both the LMS and the FTF was achieved. For the first second or so the speech processed with FTF sounded better (less reverberant) than that processed with LMS.

In real time situations, the time of arrival of strong echoes varies with time and is unknown. Filters with large processing windows are needed in order to obtain good cancellation. The reverberant signal of figure 6.12 is passed through an adaptive dereverberation filter of order 40 . The results for both the LMS and FTF algorithms are presented in table 6.4 and are plotted in figure 6.18 . For both the LMS and the FTF cases substantial improvement in the snr is achieved. The snr is in general slightly better for the FTF case. However, the LMS technique gave a better performance during a very short period of time. Next, the reverberant signal of figure 3.7 is processed using an adaptive

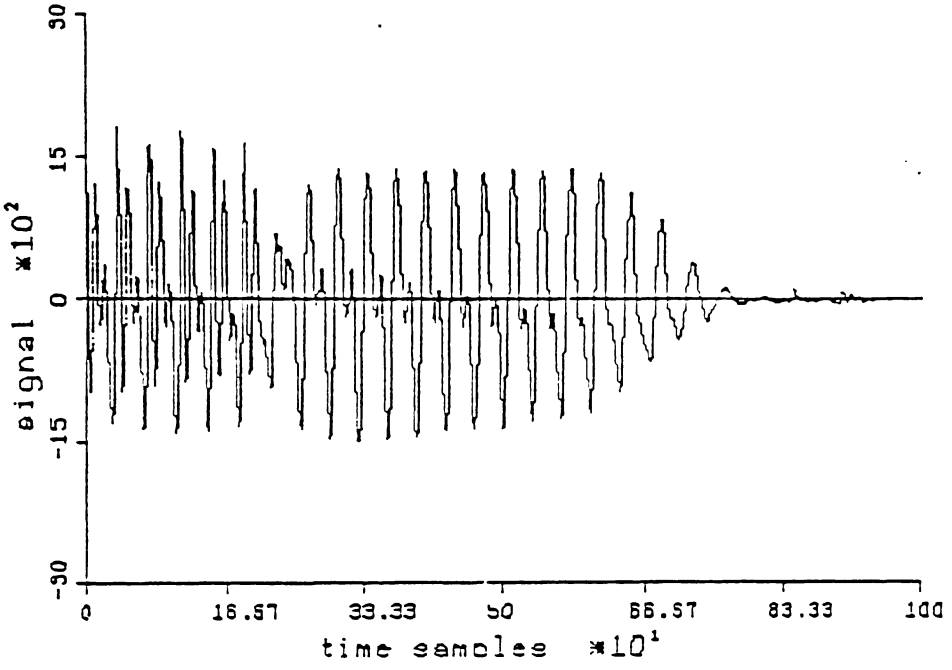


Fig. 13 : Nonreverberant Block of Speech

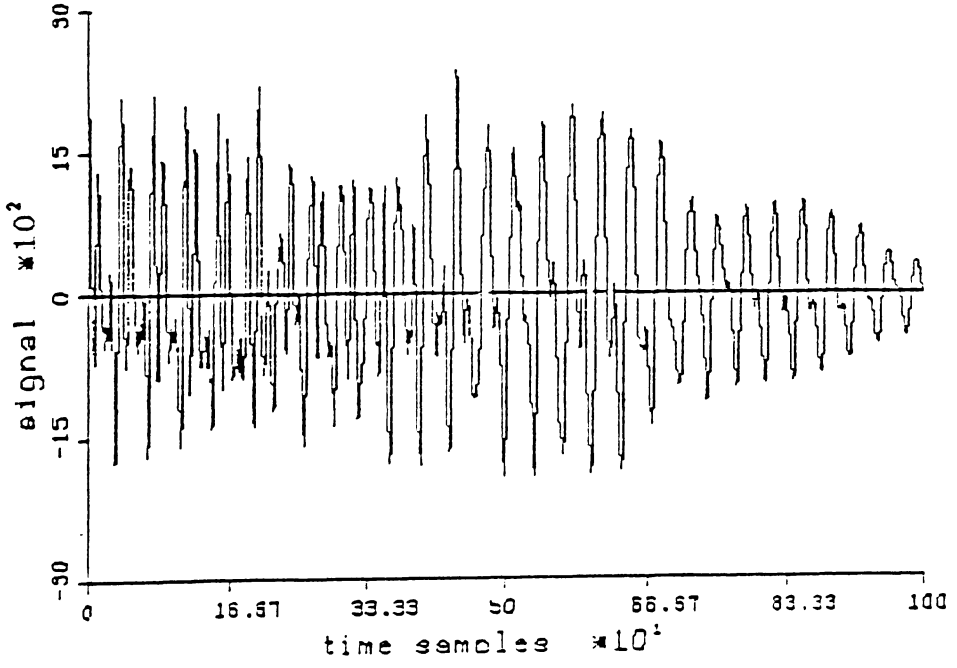


Fig. 14 : Reverberant Block of Speech

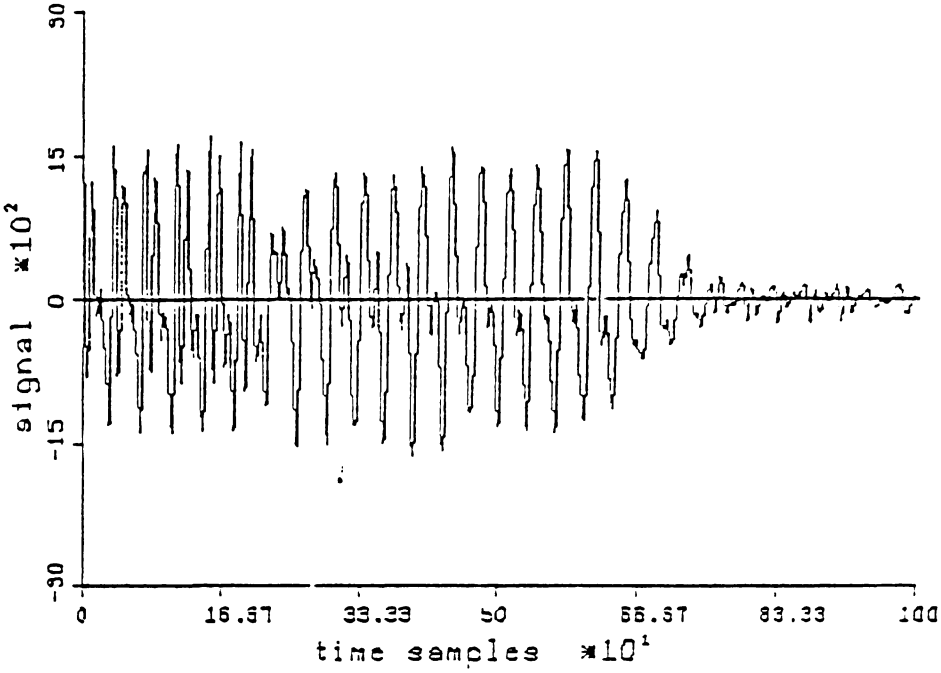


Fig. 15 : Processed Block (LMS)

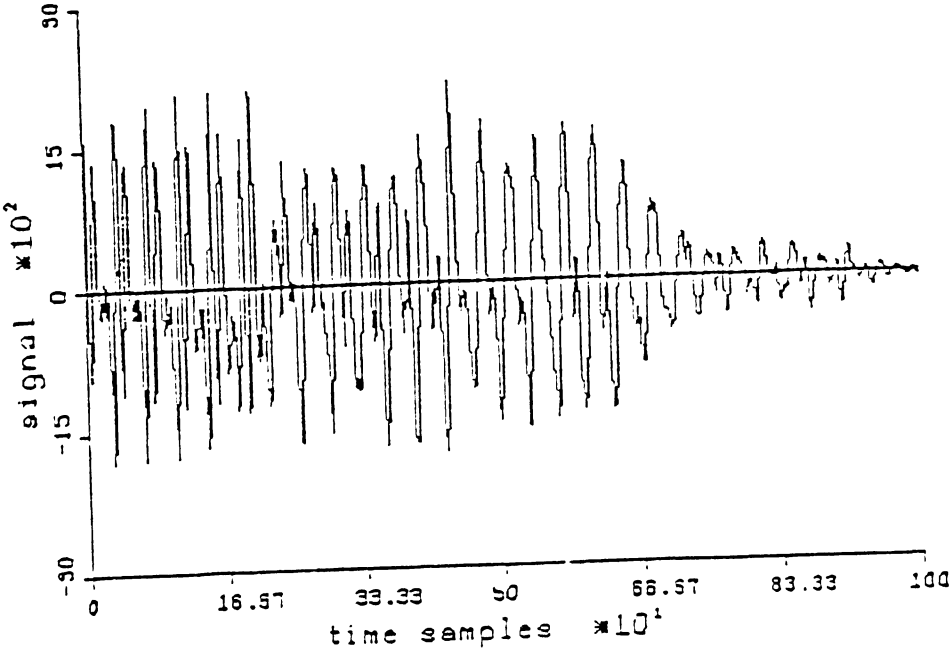


Fig. 16 : Processed Block (FTF)

Block #	Reverberant signal snr in dB	processed with LMS snr in dB	Processed with FTF snr in dB
1	5.11	5.11	12.70
2	4.30	4.30	14.65
3	5.11	5.65	10.43
4	6.15	7.73	10.01
5	6.78	11.86	12.74
6	4.54	14.25	13.33
7	5.82	17.98	17.50
8	4.14	15.93	16.99
9	7.31	24.78	23.48
10	4.89	16.32	16.53
11	4.04	15.11	15.88

(each block contains 2000 speech samples)

Table 6.3 : Real speech (order = 1)

Block #	Reverberant signal snr in dB	processed with LMS snr in dB	Processed with FTF snr in dB
1	5.11	5.11	5.99
2	4.30	4.30	10.08
3	5.11	5.48	2.30
4	6.15	6.08	2.39
5	6.78	9.97	10.89
6	4.54	10.15	11.30
7	5.82	15.70	17.08
8	4.14	11.23	12.15
9	7.31	16.08	19.21
10	4.89	12.17	13.74
11	4.04	12.81	14.72

(each block contains 2000 speech samples)

Table 6.4 : Real speech (order = 40)

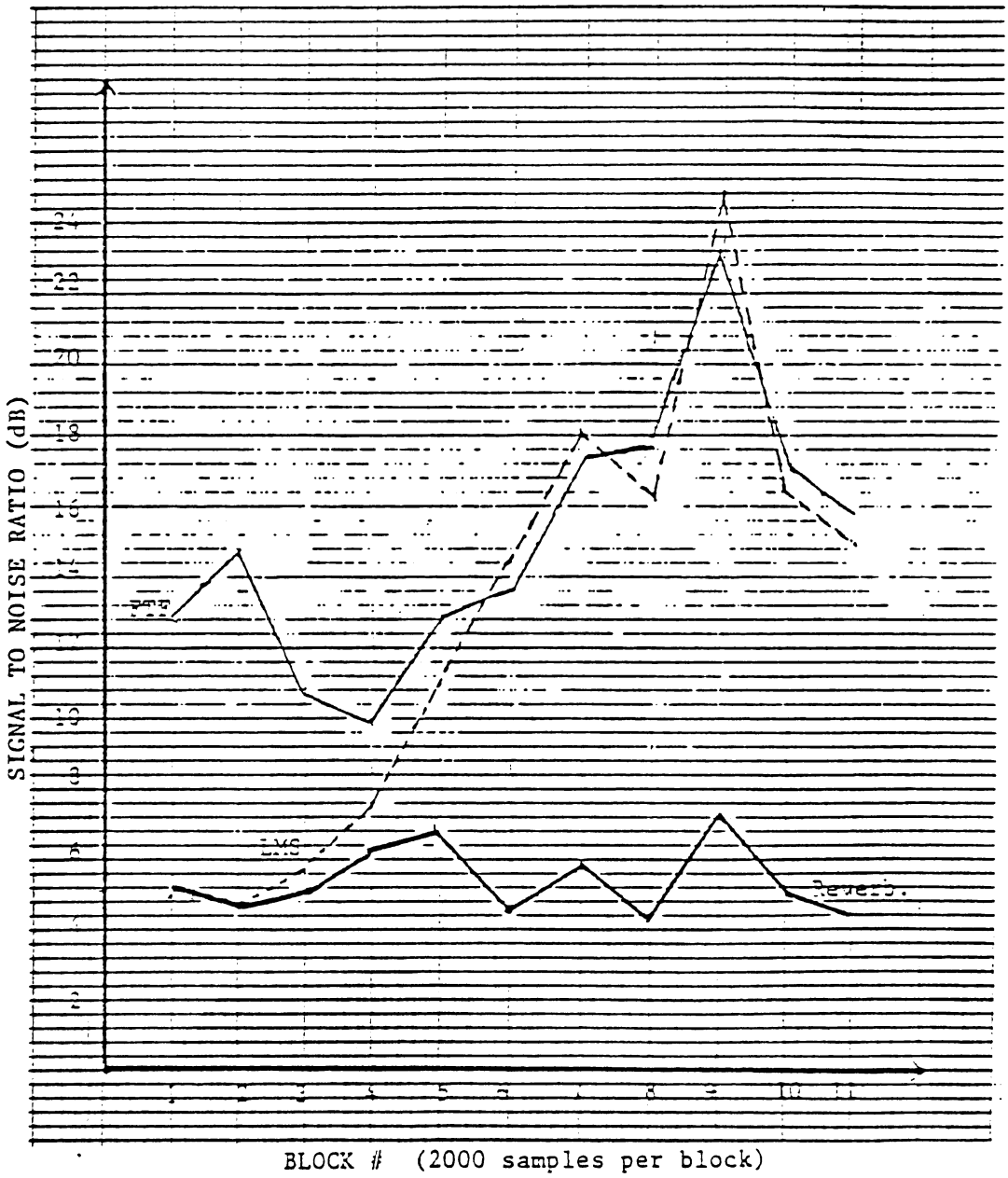


Fig. 17 : Real Speech snr (filter order = 1)

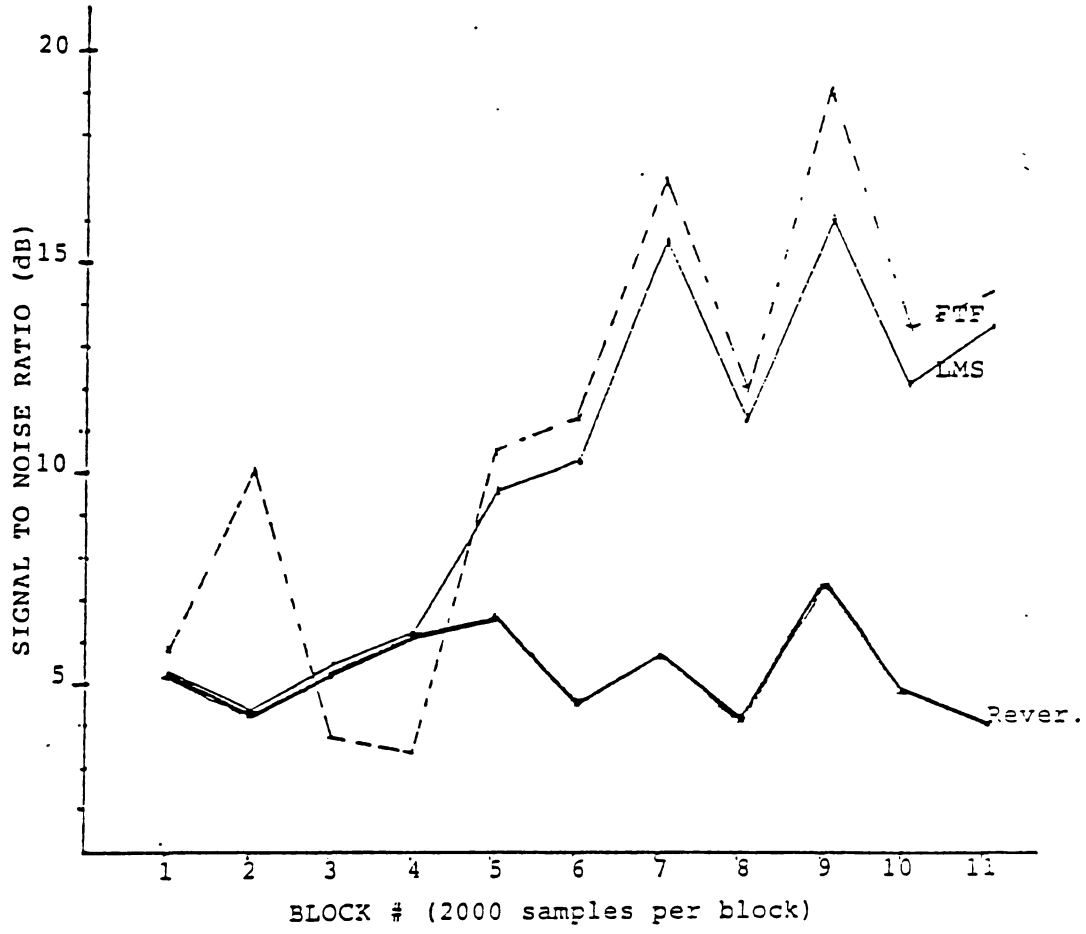


Fig. 18 : Real Speech snr (filter order = 40)

dereverberation filter of order 40. The signal to noise ratio is computed for each block of 2000 samples. In table 6.5 we compare the snr of the reverberant signal, the snr of the signal processed with LMS, and the snr of the signal processed with FTF. A plot of table 6.5 is shown in figure 6.19 . For both the LMS and FTF cases, substantial improvement in the snr is obtained. In general, the FTF algorithm is slightly better.

Results of simulations show that the proposed adaptive dereverberation filter was able to reduce reverberations produced by stationary, widely spaced echoes. In most cases, significant improvement in the snr was achieved (figures 6.8, 6.9, 6.17, 6.18, and 6.19). Audible improvement in the quality of speech was obtained and was demonstrated by playing reverberant and processed speech files over the digital sound system. Results also show that in most cases the FTF filter is slightly better than the LMS filter (figures 6.8, 6.9, 6.17, 6.18, and 6.19). This is caused by the fact that the FTF algorithm has a faster rate of convergence. The results obtained demonstrates the ability of the proposed dereverberation filter to reduce reverberations. In the case of a fixed room impulse response, the use of the FTF algorithm did not produce major improvement compared to the LMS algorithm. The advantage of the FTF filter is its ability to quickly track changes in a system. This advantage is not evident when one is using a fixed impulse response to model the echo generation process.

In most of the simulations, improvement in the signal to noise ratio is achieved. Yet, as it was discussed, the degree of cancellation is limited by the fact that some of the non-reverberant part of the signal are predicted and cancelled. The incorporation of a pitch tracker is a method that could reduce or eliminate the cancellation of the nonreverberant part of a signal. In the next section we present the results of incorporating a pitch tracker into the dereverberation algorithm.

Block #	Reverberant signal snr in dB	processed with LMS snr in dB	Processed with FTF snr in dB
1	1.16	1.16	5.23
2	-0.18	-0.18	9.58
3	2.04	2.31	-0.02
4	-0.45	5.45	0.39
5	1.71	8.95	7.02
6	1.54	9.77	9.12
7	2.41	12.58	16.73
8	1.19	10.35	13.28
9	2.84	12.38	18.53
10	0.17	9.95	14.51
11	-0.02	11.32	15.96

(each block contains 2000 speech samples)

Table 6.5 : Real speech (order = 40)

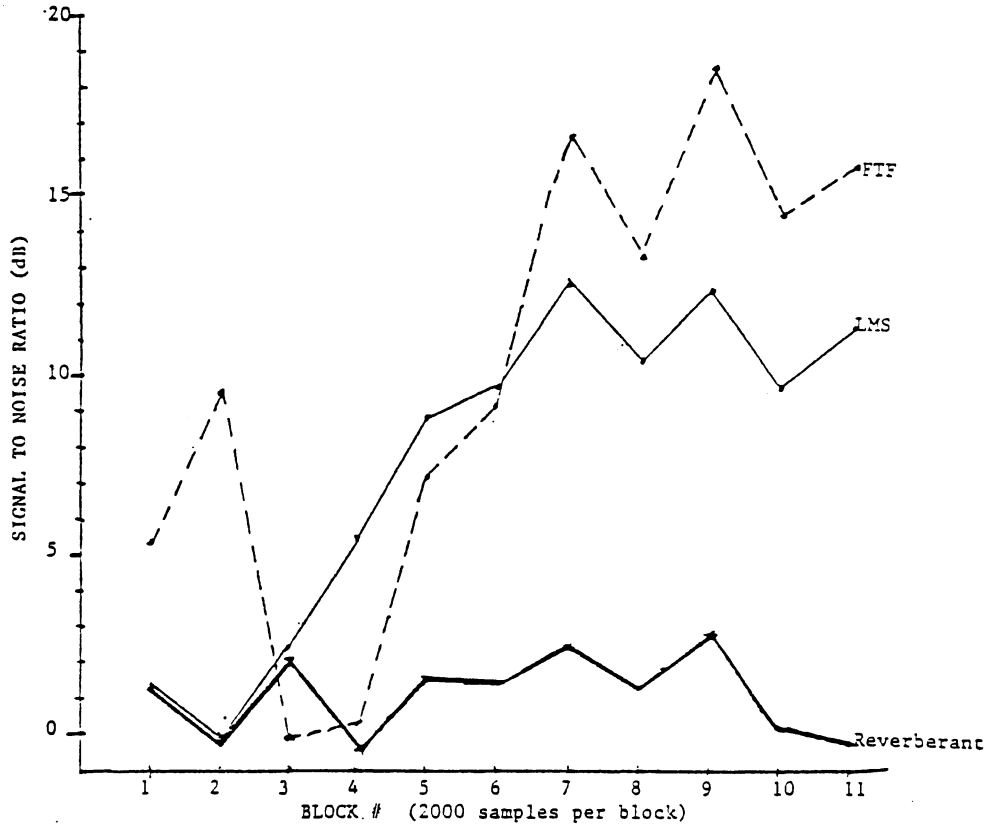


Fig. 19 : Real Speech snr (filter order = 40)

6.2. Incorporation of a Pitch Tracker

The performance of the constrained filter is very dependent on how good one is able to estimate the pitch period. To test the pitch estimation algorithm of section 5.2, a synthetic signal of 2000 samples is generated and shown in figure 6.20 . The true pitch period of the first and last blocks of 500 samples is 57. For the second block of 500 samples the pitch period is 65. The pitch period of the third 500 samples block is 75. The estimate of the pitch is presented in figure 6.21 . The transition time, that is, the time needed in order for the estimator to adapt to a new pitch period, is about 75 samples. At 8000 Hz sampling frequency, this corresponds to about 9.4 msec.

The constrained adaptive dereverberation filter for the LMS case is tested using synthetic data as well as real speech signals. The reverberant signal of figure 6.7 is passed through a constrained filter of order 30. The step size is equal to 0.001 and the delay Δ is equal to 160. The snr of the output is computed and compared to that of the unconstrained filter. The results are shown in table 6.6 and plotted in figure 6.22 . Significant improvement in the snr is achieved when using the constrained filter. The performance of the LMS algorithm is very dependent on the step size. As discussed in chapter 5, the LMS algorithm converges as long as the step size is less than the reciprocal of the maximum eigenevalue of the input autocorrelation matrix (equation 5-10). A different step size (step size $= 0.01$) is used and results are given in table 6.7 . A plot of table 6.7 is presented in figure 6.23 . The snr of the unconstrained filter output is in some case lower than that of the reverberant signal. However, improvement of a few dB is obtained when using the constrained filter. The constrained filter gave significant improvement in the case of synthetic data. Next, real speech signals are used to study the performance of the constrained filter.

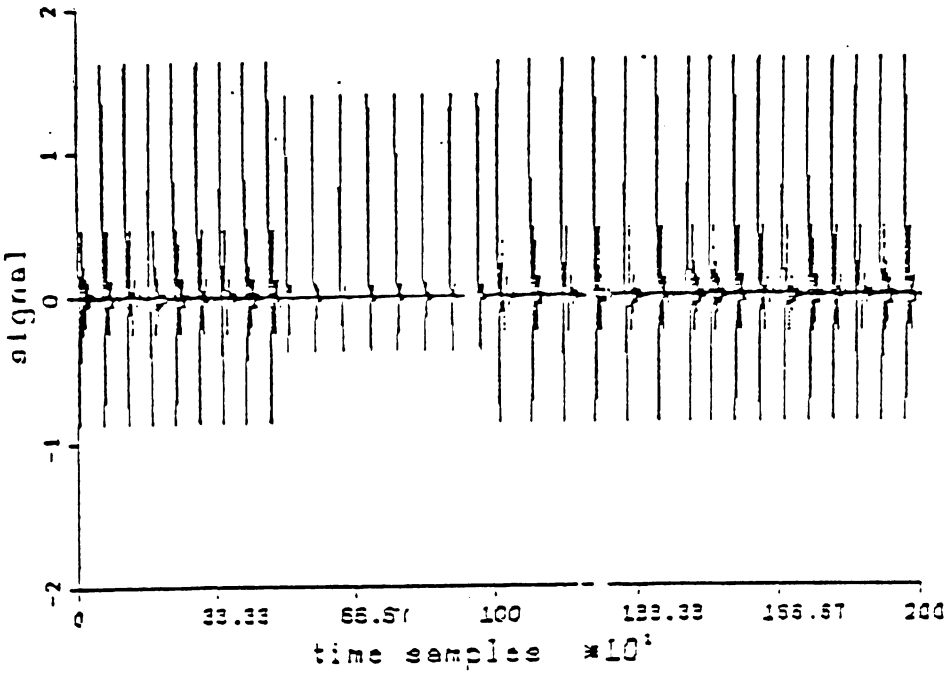


Fig. 20 : Synthetic Data Signal

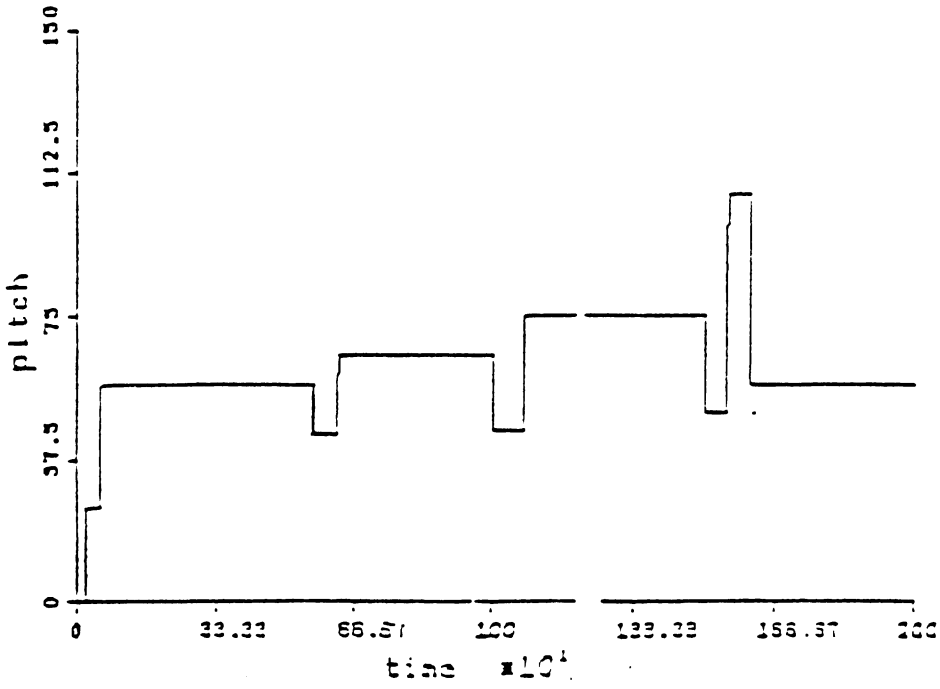


Fig. 21 : Pitch Period Estimate

Block #	Signal to noise ratio in dB		
	Reverberant signal	unconstrained filter	constrained filter
1	11.94	12.23	12.14
2	4.31	7.02	6.63
3	5.09	7.82	10.33
4	3.79	5.47	9.59
5	4.82	3.83	9.52

Table 6.6 : Synthetic speech (order=30)
LMS algorithm, step size = 0.001

Block #	Signal to noise ratio in dB		
	Reverberant signal	unconstrained filter	constrained filter
1	11.94	12.66	13.61
2	4.31	2.50	7.24
3	5.09	1.11	6.07
4	3.79	0.78	4.70
5	4.82	0.36	4.73

Table 6.7 : Synthetic speech (order=30)
LMS algorithm, step size = 0.01

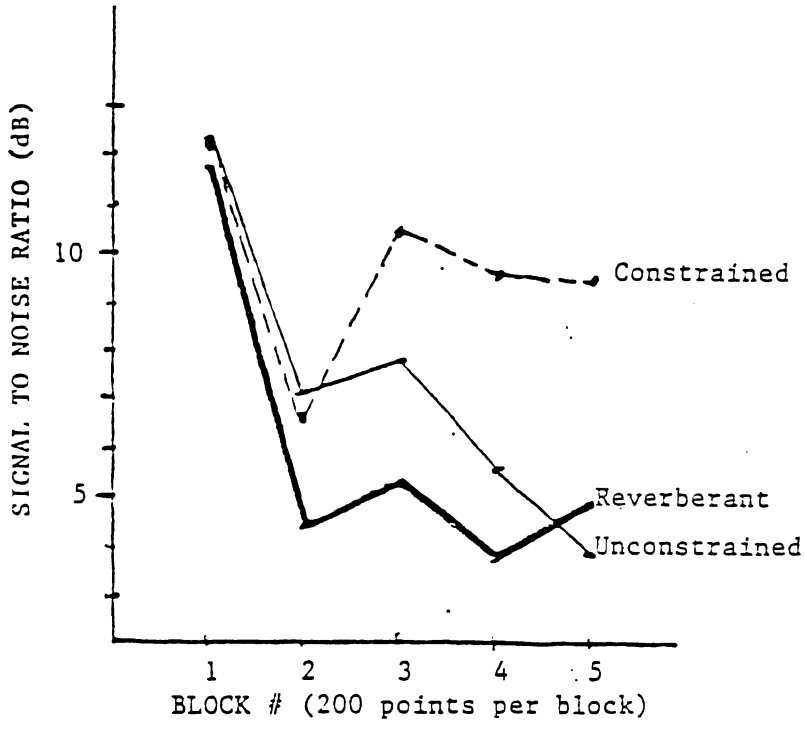


Fig. 22 : Synthetic Data snr (step size = 0.001)

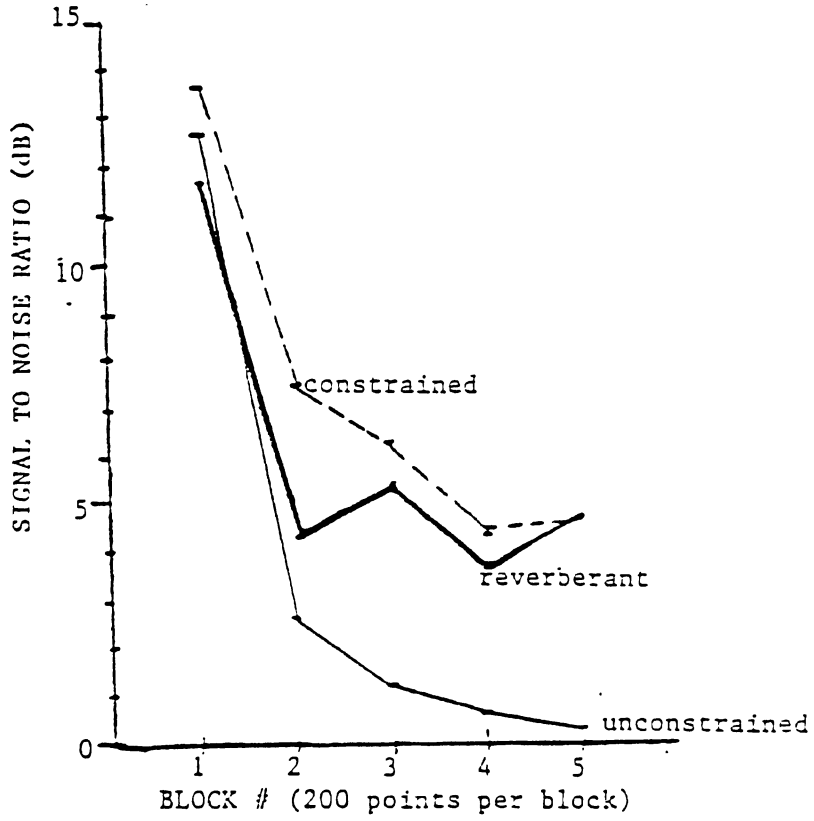


Fig. 23 : Synthetic Data snr (step size = 0.01)

The reverberant signal of figure 3.7 is used as input to a constrained filter of order 40. The LMS algorithm is used to update the filter coefficients. In table 6.8, the snr of the output produced by the constrained filter is compared to that produced by an unconstrained LMS filter of the same order. Figure 6.24 represents a plot of table 6.8 . The snr is slightly higher for the constrained filter.

Results show that in the case of synthetic data signals, the incorporation of a pitch tracker improved the cancellation capability of the adaptive dereverberation filter. As shown in figures 6.22 and 6.23 significant improvement in the signal to noise ratio was achieved with the use of a constrained filter. However, in real speech signals simulations only a very slight improvement was obtained with the incorporation of a pitch tracker (figure 6.24). In the case of real speech signals the pitch period varies with time, causing many transition periods and thus, preventing us from obtaining a good pitch period estimate. The performance of the constrained adaptive filter is very dependent on the ability of obtaining an accurate pitch period estimate. Thus, in the case of real speech, the improvement obtained with the incorporation of a pitch tracker was not of great significance.

6.3. Simulations for a Time Varying Echo Model

In all the previous simulations, a constant impulse response was used to model the echo generation process in a small room. Yet, in real time situations the response of a room varies with every movement of the talker (talker walking, moving his head, ...). The purpose of this section is to investigate the ability of the dereverberation filter to track changes in the room impulse response. A reverberant signal is generated using a time varying echo model and then passed through an adaptive dereverberation filter. The

Block #	Signal to noise ratio in dB		
	Reverberant signal	unconstrained filter	constrained filter
1	1.16	1.16	1.16
2	-0.18	-0.18	-0.18
3	2.04	2.31	2.36
4	-0.45	5.45	6.41
5	1.71	8.95	8.98
6	1.54	9.77	10.32
7	2.41	12.58	12.67
8	1.19	10.35	11.68
9	2.84	12.38	12.89
10	0.17	9.95	11.44
11	-0.02	11.32	11.63
(each block contains 2000 speech samples)			

Table 6.8 : Real speech (order = 40)

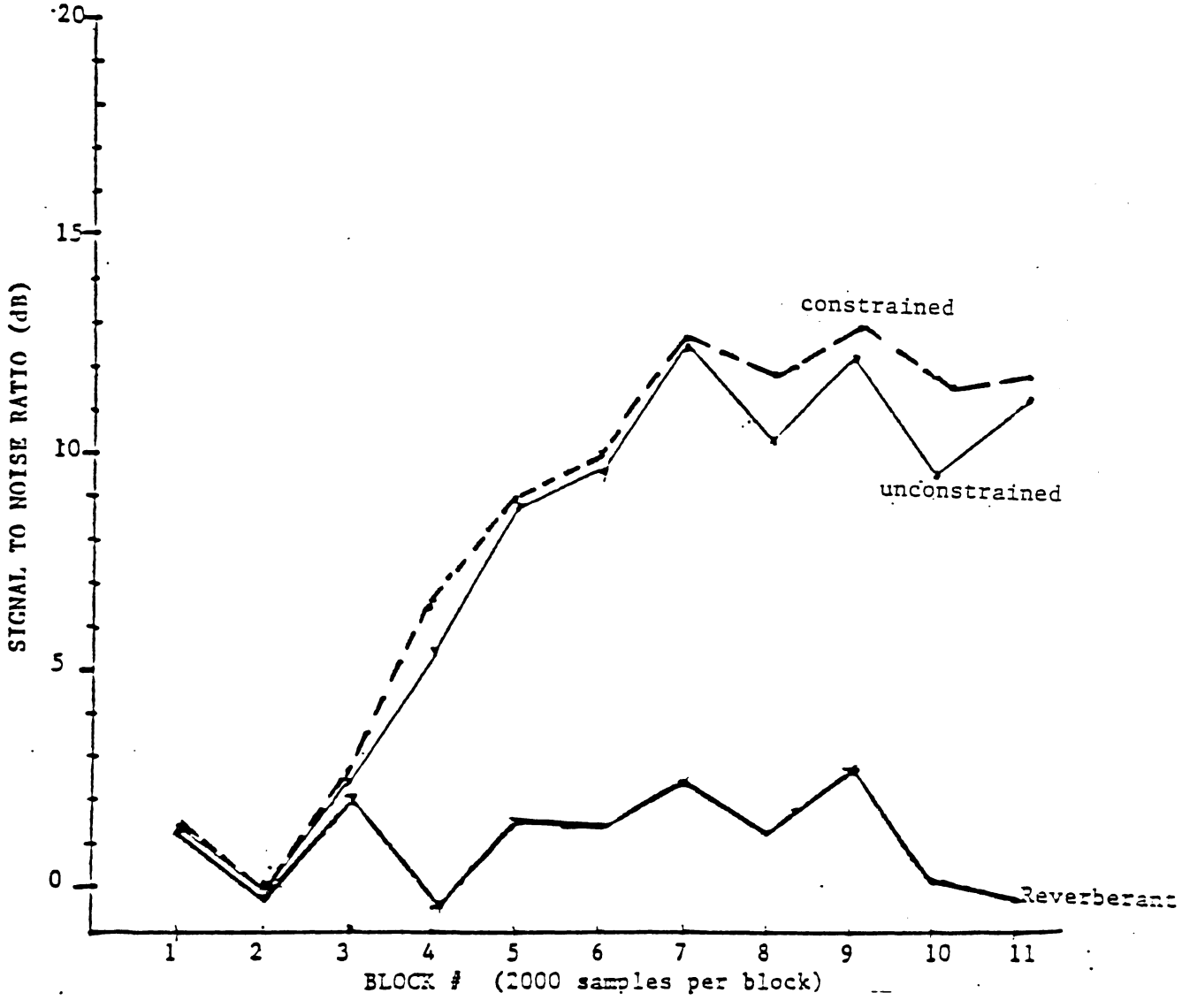


Fig. 24 : Real Speech snr (filter order = 40)

snr of the processed signal (output of the filter) is computed and compared to that of the reverberant signal. For the filter to cancel reverberation it must be able to track the changes in the time varying echo model.

A time varying echo model is generated by assuming that the time of arrival of each echo changes with time. A delay in the time of arrival corresponds to the fact that the echo travel a longer distance before it reaches the microphone. To form a reverberant signal we will use the impulse response shown in figure 6.25 . The time of arrival T_e of the first echo of the impulse response varies with time as shown in figure 6.26 . T_e is first delayed by 3 msec and then advanced by 3 msec. At a sampling rate of 8000 Hz, this implies delaying or advancing T_e by one sample every about 334 samples of speech.

The reverberant signal generated using the above varying room impulse response is passed through an adaptive dereverberation filter of order 40 ($\Delta = 160$). Then the snr of the processed signal is computed for both the LMS and the FTF case and compared to that of the reverberant signal. Blocks of 2000 data samples are used in the computation of the snr. Results are plotted in figure 6.27 . For the LMS, the optimal step size is determined by trial and error. The snr of the entire processed signal is computed for five different step sizes (figure 6.28) . The optimal step size, that is the one that produced the highest snr is equal to 2.10^{-10} . For the results presented and plotted in figure 6.28, the FTF data weighting factor is equal to 1.0 .

As shown in figure 6.28, improvement in the snr was achieved with the LMS algorithm. However, in the case of FTF some parts of the processed signal were at a lower snr level than that of the reverberant signal. This is due to the fact that a varying room impulse response is used to model the echo generation process. For the FTF to be able to

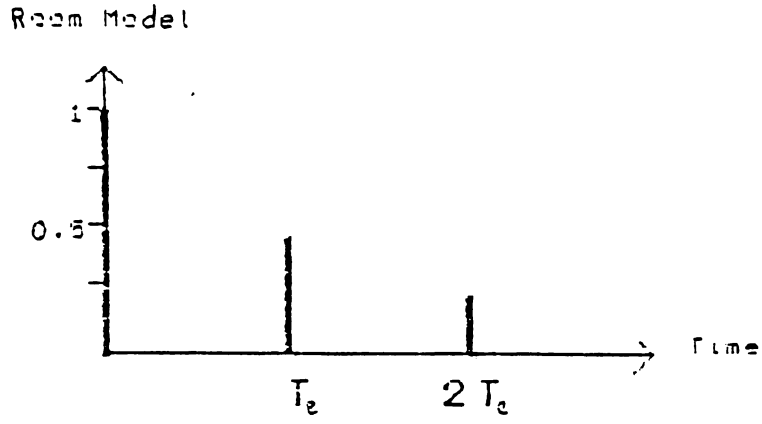


Fig. 25 : Room Impulse Response

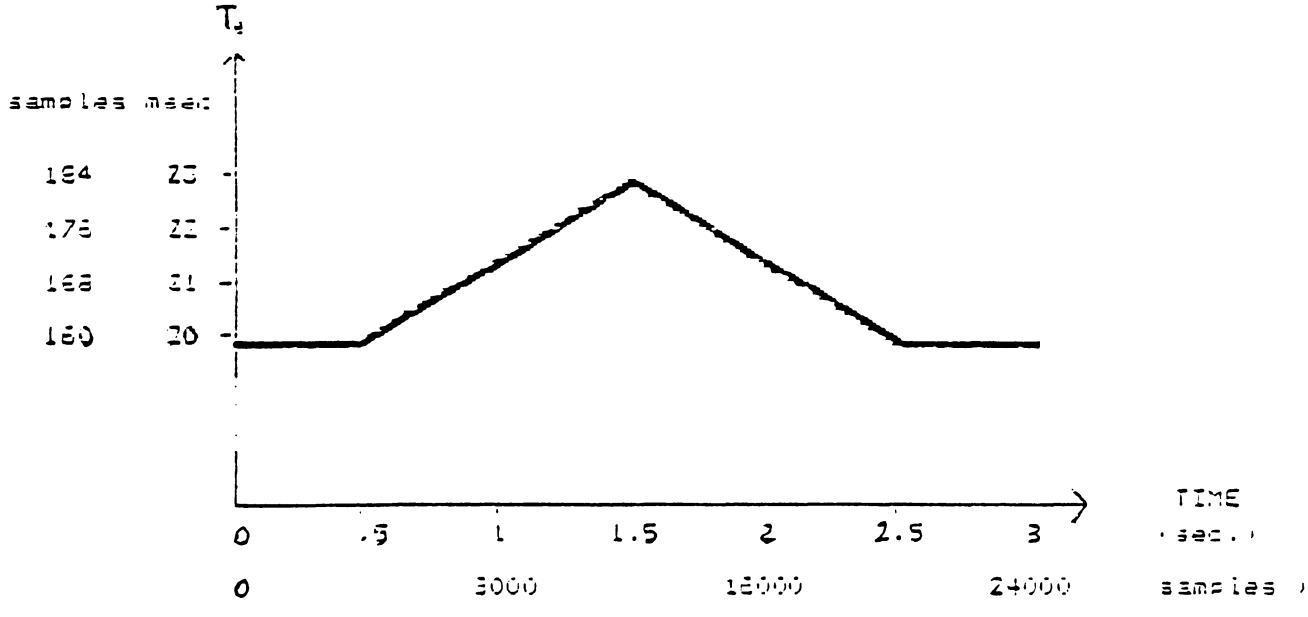


Fig. 26 : Time of Arrival of First Echo

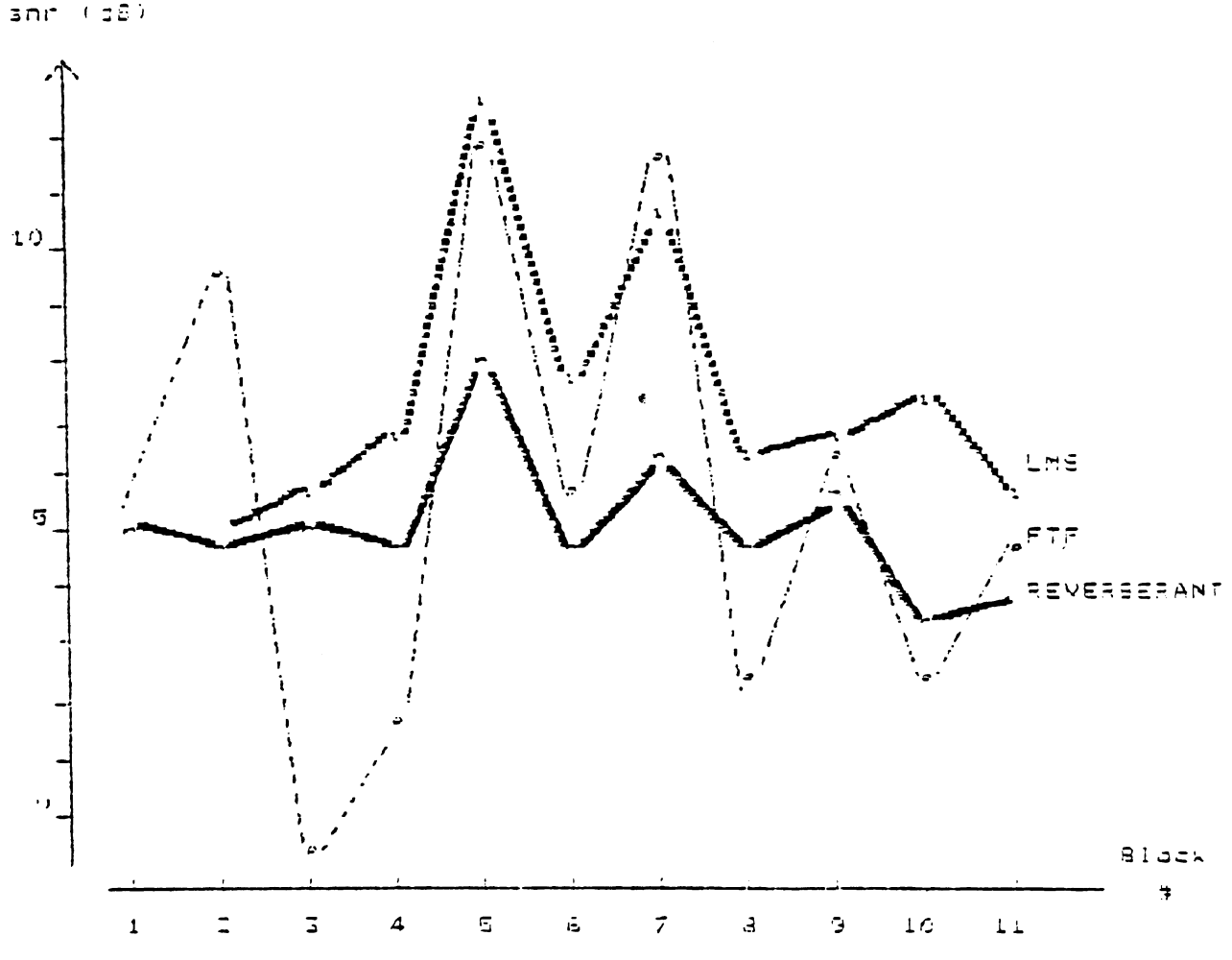


Fig. 27 : Real Speech snr (2000 samples per block)
filter order = 40

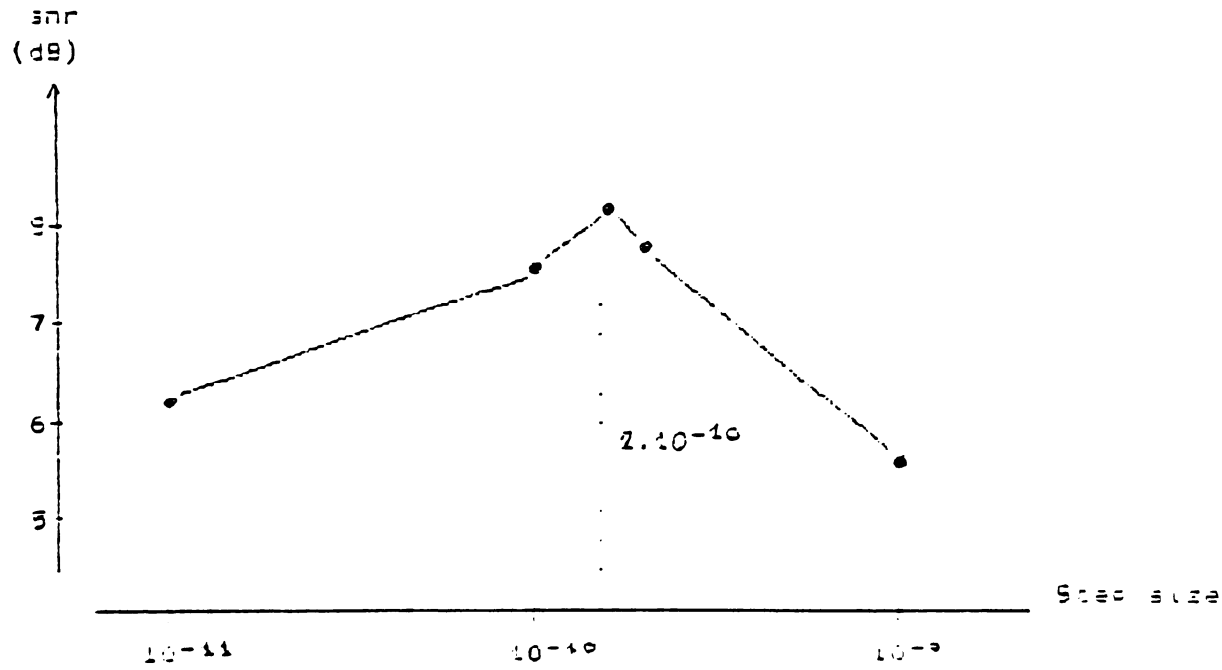


Fig. 28 : Determination of the LMS Optimal Step Size
Based the snr of the Entire Data File

track changes in the room impulse response a data weighting factor λ of a value less than 1.0 must be used [20]. Yet, results show that the FTF filter diverges every time λ is less than 0.99999 . Simulations were performed for values of λ equal to 0.98, 0.999, and 0.9999 . For all these 3 cases the FTF filter became unstable. To cope with the problem of divergence it was suggested in [20] the use of soft-constrained rescue; every time a so called rescue variable becomes negative an initialization procedure is used to continue the algorithm. In the simulations ($\lambda = 0.98$ and $\lambda = 0.999$), the rescue variable stays positive and yet the FTF filter diverges.

Results of simulations show that the LMS filter was able to slowly track changes in the room impulse response (figure 6.28). However, the FTF filter was found to be subject to divergence every time the so called forgetting factor was less than 1. Therefore, further investigation is needed in order to understand and solve the stability problem associated with the FTF algorithm.

6.4. Dereverberation of Actual Reverberant Data

In this section, we present the results of applying the adaptive dereverberation filter on actual reverberant data. A 3-second speech signal was recorded in a very reverberant environment (racketball court) and then transferred to the Vax. At 8 kHz sampling rate, the signal consisted of 24000 samples. The signal was used as input to the adaptive dereverberation filter. The output of the filter was then played back and compared to the reverberant signal. In the case of actual reverberant data, the signal to noise ratio cannot be used to measure the degree of cancellation. As a metric of performance, the processed signal is played over the digital sound system, and its quality is compared to that of the reverberant signal. Several experiments were run for various values of the filter length, the

bulk delay, and the LMS gain factor. Table 6.9 lists the values of the parameters associated with each experiment. In most of the experiments, the processed signal sounded slightly different than the reverberant signal. The difference was more evident when using larger order filters. Signal power of the reverberant and processed signals were compared and that of the processed was less. This suggests that parts of the reverberant signal have been removed by the dereverberation filter. The rationale for this suggestion is the following: The bulk delay that is used in order to form the input to the adaptive filter decorolates the speech. The correlation that exists between the reference and input signal is due to the presence of an echo rather than to the nature of speech itself. Any power removed is therefore due to the cancellation of some of the echoes.

Results of the experiments suggest that some of the reverberations have been cancelled. Yet, the improvement in the quality of the speech was not obvious. The time of duration of the reverberations in a typical reverberant room is sometimes more than 200 millisecond. To obtain a significant improvement in the quality of speech a filter consisting of more 1000 coefficients is needed. However, In simulations the order of the filter we can use is very limited by the compiler speed.

Experiment #	Algorithm	LMS gain	Bulk delay (samples)	Filter order (samples)
1	LMS	2.10E-10	240	40
2	LMS	2.10E-10	200	40
3	FTF	-	240	40
4	LMS	10E-10	200	80
5	LMS	10E-10	280	80
6	LMS	10E-10	240	120
7	LMS	10E-10	320	120
8	LMS	10E-10	440	120
9	LMS	10E-11	240	120
10	FTF	-	240	120

Table 6.9 : Parameters descriptions

Chapter 7

CONCLUSION

7.1. Adaptive Dereverberation

An adaptive dereverberation filter was derived and applied to reverberant speech signals, in order to remove reverberations. The snr of the processed signal was computed and compared to that of the reverberant signal. Results show that in most cases improvement in the snr to noise ratio was achieved. The improvement was limited (1-17 dB) by the fact that small parts of the nonreverberant signal can predicted and cancelled. Yet. an improvement in the quality of speech was also obtained and was demonstrated by playing the processed speech files over the digital acquisition and playback system. The proposed dereverberation filter offered significant cancellation of reverberation in simple situations where only few prominent reflections exist. The effects in actual rooms are still in need of more investigation. It is evident from the results of this thesis that adaptive filters can be used to remove reverberations and therefore warrants further investigation.

7.2. Incorporation of a Pitch Tracker

As it was discussed in chapter 4, the prediction and cancellation of some nonreverberant components of the speech is very possible. The incorporation of a pitch tracker was suggested in order to improve the performance. At each iteration the pitch period is estimated and the adaptive filter is constrained in order to prevent cancellation of nonreverberant parts of speech. Results show that in the case of synthetic data signals the incorporation of a pitch tracker increases the cancellation capability of the dereverberation filter. However, in simulations using real speech signals, only a slight improvement

in performance was achieved. The performance of the constrained filter is very dependent on how good one is able to estimate the pitch period. In the case of real speech, the pitch period varies with time causing many transition periods and thus, preventing us from obtaining good pitch period estimates. From the experimental results, we conclude that the slight improvement in performance is not worth the number of additional arithmetic computations involved with the incorporation of a relatively simple pitch tracker.

7.3. LMS and FTF adaptive algorithms

In this thesis the two adaptive algorithms of interest are the LMS and the FTF. As it was mentioned earlier, the LMS algorithm requires 2N multiplications per time iteration compared with 7N multiplications per iteration for the FTF. A relatively simple hardware and software implementation of the LMS was presented in [22]. Design based on four TMS320 microprocessors was completed. It was shown that at 8000 Hz sampling rate, 58 taps per processor can be handled [22]. A very rough and preliminary study suggests that in order to implement the FTF algorithm only about 14 taps per processor (at 8000 Hz) can be accommodated. Compared with the LMS, it takes about four times more hardware and software in order to implement a same length FTF filter. This increase in implementation requirement does not constitute a significant drawback by the standards of today's advanced technology.

In the case of a fixed room impulse response, simulations results show substantial improvement in the snr for both the LMS and FTF. In general, the FTF is slightly better. In real life situations, the room impulse response varies with time and a dereverberation filter must track the variations. The LMS dereverberation filter was able to slowly track variations and thus it produced an improvement in the snr of a few dB. For

the FTF to be able to track changes the filter must be windowed. An exponential window was suggested in [20] by using a forgetting factor λ of a value less than 1 . However, in the simulations the FTF filter diverged every time the value of λ was chosen to be less than 0.99999 . At this point, and in my opinion, the LMS is still the adaptive algorithm to use in real life applications; for the FTF to become the compatible alternative of the LMS better understanding of its stability must be achieved.

REFERENCES

[1] O. M. M. Mitchell and D. A. Berkley, " Reduction of Long-Time Reverberation by a Center-Clipping Process," *Journal of the Acoustic Society of America,* vol. 47 (1970).

[2] J. L. Flanagan and R. C. Lummis, " Signal Processing to Reduce Multipath Distortion in Small Rooms," *Journal of the Acoustic Society of America,* vol. 47 (1970).

[3] J. B. Allen, " Synthesis of Pure Speech from a Reverberant Signal," U.S. Patent No. 3786188, January 1974.

[4] M. M. Sondhi and D. A. Berkley, "Silencing Echo on the Telephone Network," *Proceedings IEEE,* August 1980.

[5] D. L. Duttweiler, "Bell's Echo Killer Chip," *IEEE Spectrum,* October 1980.

[6] D. A. Berkley and O. M. M. Mitchell, "Seeking the Ideal in 'Hands-Free' Telephony," *Bell Labs. Record,* Nov. 1974.

[7] A. B. Clark and R. C. Mathes, "Echo Suppressors for Long Telephone Circuits," Proceedings IEEE, April 1925.

[8] S. J. Campanella, H. G. Suyderhoud and M. Onufry, "Analysis of an Adaptive Impulse Response Echo Canceller," *COMSAT Technical Review,* Vol. 2, Spring 1972.

[9] O. A. Horna, "Logarithmic Echo Canceller," U.S. Patent No. 4064379, December 20, 1977.

[10] O. A. Horna, "Echo Canceller With Adaptive Transversal Filters," *Comsat Technical Review,* Vol. 8, Fall 1978.

[11] O. A. Horna,"Echo Control in Teleconferencing", *Proceedings,* 1983 Globecom, San Diego, CA; December 1983.

[12] R. Ceruty and F. Pira, "Application of Echo-Cancelling Techniques to Audioconference", *Proceedings,* ICASSP, 1982.

[13] C. M. Harris, *Handbook of Noise Control,* McGraw-Hill, New York, 1957.

[14] J. B. Allen and D. A. Berkley, "Image Method for Efficently Simulating Small Room Acoustics", *J. Acoust. Soc. Am.,* 65(4), April 1979.

[15] E. A. Robinson and S. Treitel, *Geophysical Signal Analysis,* Chapters 10-12, Prentice-Hall Inc., Inglewood, NJ, 1980.

[16] R. V. Cox , R. E. Crochiere and J. D. Johnston, " Real-time Implementation of Time-Domaine Harmonic Scaling of Speech for Rate Modification and Scaling," *IEEE Trans.* ASSP, February 1983.

[17] J. McCool and B. Widrow, "Principles and Applications of Adaptive Filters: A Tutorial Review," Naval Undersea Center, San Diego, CA, Tech. Publ. 530, March 1977.

[18] B. Widrow, et. al., " Stationary and Nonstationary Learning Characteristics of the LMS Adaptive Filter," *Proc. IEEE,* vol. 64 No.8, August 1976.

[19] N. Levinson, " The Wiener RMS Error Criteria in Filter Design and Prediction," *J. Math. Phys.*, Vol. 25, 1974.

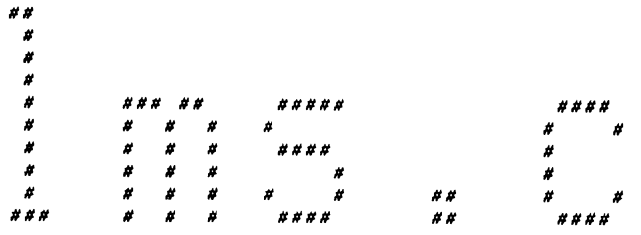
[20] John M. Cioffi and T. Kailath, "Fast, Recursive-Least-Squares Transversal Filters for Adaptive Filtering," *IEEE Trans. on Acoust., Speech, Signal Processing,* vol. ASSP-32, no. 2, April 1984.

[21] D.D. Falconer and L. Ljung, " Application of Fast Kalman Estimation to Adaptive Equalization," *IEEE Trans. on Comm.,* Vol. COM-26, October 1978.

[22] T. K. Miller and S. T. Alexander, "An Implementation of the LMS Adaptive Filter Using an SIMD Multiprocessor Ring Architecture," *ICASSP,* Tampa, Florida, March 1985

APENDIX : Computer Code

jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk



Wed Dec 11 15:34:25 1985

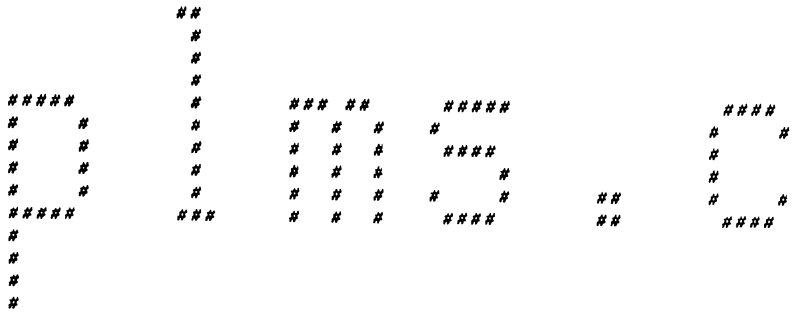
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk

```
.()
/*   ADAPTIVE DEREVERBERATION FILTER (LMS)


      This program is the main program for using the lms algorithm
          in adaptive dereverberation.

      The reference d(n) is the reverberant signal.
      The same signal d(n) is delayed by DELTA samples to form the
          input signal x(n).

      The CCSP   readld( ) and writeld( ) functions are used
          as I/O calls in this program.

*/

/*********************************************************/

float   lms( d, x, w, alpha, dp, e, npts, order, tau )

/*********************************************************/


/*   The lms( ) subroutine implements the Least Mean Squares (LMS)
      adaptive prediction algorithm.  The arguments are defined
      as follows:                                                    */

float  d[ ];      /*  the desired signal input  ... passed from
                      the calling program                          */

float  x[ ];      /*  the signal used as input data to the lms
                      transversal filter ... passed from calling prog.  */

float  w[ ];      /*  the lms filter coefficients ... computed in the
                      subroutine and passed to the calling prog.   */

float  alpha;     /*  the lms feedback gain parameter  ... passed
                      from calling prog.                           */

float  dp[ ];     /*  the prediction of the desired signal ... returned
                      to the calling prog.                         */

float  e[ ];      /*  the lms prediction error ... returned to the
                      calling program                              */

int    tau[ ];    /*  array of delays used in selecting the data samples
                      for the lms prediction                       */

int npts;         /*  the number of data samples in the d,x,dp, and e arrays  */
int order;        /*  the order of the lms transversal filter      */

/*************************************************************************/

/* Declare local variables:                                       */

{

int i,idum,j;     /*  integer dummy variables                     */

float sum;        /*  float  dummy variable                       */

/*************************************************************/

/* Begin Program:                          */

/*  Form lms filter prediction             */
```

```
for (i=1; i<npts; i++) {
    sum = 0.0;
    for ( j=1; j<=order; j++) {
        idum = i-j;
        if (idum >=0)
           sum = sum + w[j]*x[i-tau[j]];
        sum = sum;
        };
    dp[i] = sum;
    e[i]  = d[i] - dp[i];

    /*  Update adaptive filter weights           */

/*        printf(" %d \n",i);       */

    for (j=1; j<=order; j++)
        {
        w[j] = w[j] + ( alpha*e[i]*x[i-tau[j]] );
/*        printf("   %f      ",w[j]);     */
        }
 /*       printf("\n\n\n");    */
    }

/*  lms algorithm complete                    */

}

/*        End the lms subroutine                        */
/*****************************************************************/

#include <math.h>
#include <stdio.h>
#include "/usr/local/dsp/dsp1.h"

#define BARRAY 24100


main( )

{
   float sigin[BARRAY];  /* reverberant signal                */
   float sigout[BARRAY]; /* estimate signal                   */
   float errout[BARRAY]; /* processed signal                  */
   float inrms;          /* input rms power - for output snr calculations    */
   float w[121];         /* the adaptive filter weights                      */
   float alpha;          /* lms feedback gain parameter                      */
   float rdum;           /*   dummy register                                 */
   float err[BARRAY];    /* the lms prediction error                         */

   int siglength;        /* signal length                                    */
   int  idum,i,j,k;      /* dummy counters, etc.                             */
   int order,delay;      /* lms filter order                                 */
   int tau[121];          /* array of delays used in lms transversal filter   */

   FILE *fopen( ),*fpin;

   struct header1d   hdrin,*ptrin;
   struct header1d   hdrerr,*ptrerr;

/* ----------------------------------------------------------------*/

ptrin = &hdrin;
ptrerr = &hdrerr;

/*****************************************************************/
```

```
/*     READ IN SPEECH 1-D DATA FILE
       This file can be any length -- it only must be DSP format
       It is assumed that sigin[ ] is a reverberant signal          */


readld(ptrin,sigin,"name of the reverberant file?\n","all");
inrms     =  ptrin->stdev;
siglength = ptrin->npts;


/*****************************************/
/*  Enter the lms filter parameters       */

/******** Enter LMS gain factor ***********/
fprintf(stderr, " alpha = ? \n");
scanf("%f",&alpha);
fprintf(stderr, " alpha = %.2e \n  , alpha);

/******** Enter LMS filter order *********/
fprintf(stderr, " filter order = ?\n");
scanf("%d",&order);
fprintf(stderr," order = %d \n", order);

/******* Enter the bulk delay ************/
fprintf(stderr," enter delay, then return \n");
scanf("%d", &delay);

for(i=1;i<=order;i++) {

/* set the delay array */

     tau[i] = delay+i;
     fprintf(stderr, "delay [ %d ] =  %d \n", i,tau[i]);

/*   Initialize the lms weights ( to zero in this case )   */

     w[i] = 0.0;
     }

/* Use the lms filter in a pure predictor mode:

(1) The "desired" signal d[i] is sigin[ ];
(2) The "data" signal x[i] is also sigin[i];    */

   lms(sigin, sigin, w, alpha, sigout, err, siglength, order, tau);


/*  Now output the error file in DSP-1 format           */
/*  The error file corresponds to the processed signal  */

ptrerr->npts = siglength;
ptrerr->si = ptrin->si;
writeld(ptrerr,err,"name of the processed signal file?\n","all");

}
.)l
```

jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk



Wed Dec 11 15:34:43 1985

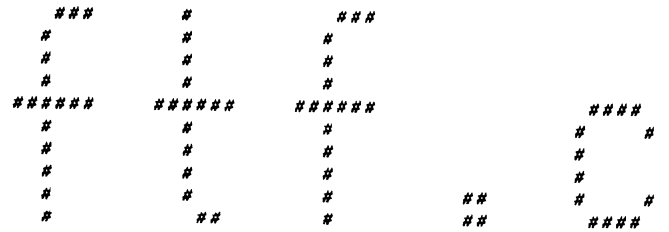
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk

```
/* pit  lms */

/* CONSTRAINED ADAPTIVE DEREVERBERATION FILTER (LMS)
   This program simulates the constrained adaptive derverberation
        filter using the lms algorithm

   The reference d(n) is the reverberant signal.
   the same signal d(n) is delayed by DELTA samples to form the
        input signal d(n).
*/

/************/

#include <math.h>
#define GAM 0.987
#define MGAM 0.013

lms(x, w, alpha, e, npts, order, delay )


float  x[ ];      /*  the signal used as input data to the lms
                      transversal filter ... passed from calling prog.        */

float  w[ ];      /*  the lms filter coefficients ... computed in the
                      subroutine and passed to the calling prog.              */

float  alpha;     /*  the lms feedback gain parameter  ... passed
                      from calling prog.                                      */


float  e[ ];      /*  the lms prediction error ... returned to the
                      calling program                                         */


int npts;         /*  the number of data samples in the d,x,dp, and e arrays  */
int order;        /*  the order of the lms transversal filter                 */
int delay;

/*****************/

{

float acf[256];
int minlag = 25;
int maxlag = 120;
float peak = 0.;
int pitch = 0;

int i,j,k,l,idum,dum;     /*  integer dummy variables                       */
float sum;                /*  float  dummy variable                         */
float window;
int cc[5];

/*************************************************************

Begin Program:                                                                */

for(i=0;i<256;i++)
   acf[i] = 0.0;

for(i=1;i<=order;i++)
   w[i] = 0.0;

/*  Form lms filter prediction                */

for (i=1; i<npts; i++) {
```

```
        /*** Determine the pitch period estimate ***/
        for(j= minlag; j<= maxlag; j++) {
           if((i-j)>=0)


                acf[j] = GAM * acf[j]
                        + MGAM * x[i] * x[i-j];

        if(j<40) window = 1.0;
        else     window = 1. - .0025 * (j-40);

        if(peak < window * acf[j]) {
                peak = window * acf[j];
                pitch = j;
        }
}

/*** Constrain the adaptive filter as discussed in CCSP-TR-18/85 ***/

    sum = 0.0;
    dum = 0;
for(j=1;j<=order;j++){
    for(k=1;k<5;k++){
       cc[k] = (k*pitch) - delay;
       if(cc[k] >= (j+10) || cc[k] <= (j-10))
         dum = dum + 0;
       else
         dum = dum + 1;
       }
        idum = i-(j+delay);
        if (idum >=0 && dum == 0)
                sum = sum + w[j]*x[idum];
    }


    e[i]  = x[i] - sum;

/*  Update adaptive filter weights */

/*  printf(" %d \n",i);  */

    for (j=1; j<=order; j++)
        {
        if(i >= (delay + j))
        w[j] = w[j] + ( alpha*e[i]*x[i-(delay+j)] );

/* printf("   %f     ',w[j]); */

        }
/* printf("\n\n\n"); */

/*  lms algorithm complete                    */
}

}

/*************************************************************/

#include <math.h>
#include <stdio.h>
#include "/usr/local/dsp/dsp1.h"


#define BARRAY 24100
```

```
main( )
{

   float sigin[BARRAY];  /* lms input signal signal                          */
   float w[41];          /* the adaptive filter weights                        */
   float alpha;          /* lms feedback gain parameter                        */
   float rdum;           /*   dummy register                                   */
   float err[BARRAY];    /* the lms prediction error                           */

   int siglength;        /* signal length                                      */
   int order;            /* lms filter order                                   */
   int delay;

   FILE *fopen( ),*fpin;

   struct header1d   hdrin,*ptrin;
   struct header1d   hdrerr,*ptrerr;

/* ------------------------------------------------------------------*/

ptrin = &hdrin;
ptrerr = &hdrerr;


/*    READ IN SPEECH 1-D DATA FILE
      This file can be any length -- it only must be DSP format
      It is assumed that sigin[ ] is a reverberant signal        */


read1d(ptrin,sigin,"name of the spk-plus-interferer file?\n","all");
siglength = ptrin->npts;


/*  Enter the lms filter parameters                                    */

fprintf(stderr, ' alpha = ? \n");
scanf("%f",&alpha);
fprintf(stderr, " alpha = %.2e \n ', alpha);

fprintf(stderr,   filter order = ?\n");
scanf("%d",&order);
fprintf(stderr," order = %d \n", order);

/*   Initialize the lms weights ( to zero in this case )
     and set the table of delay values                                */

    fprintf(stderr,' enter delay, then return \n");
    scanf("%d", &delay);
    fprintf(stderr,"delay = \n",delay);


lms(sigin, w, alpha, err, siglength, order, delay);


/*  Now output the 1-D error file.
    The error file corresponds to the processed signal
                                                                       */
ptrerr->npts = siglength;
ptrerr->si = ptrin->si;
write1d(ptrerr,err,"name of the processed signal file?\n","all");

}
```

jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk



Wed Dec 11 15:35:03 1985

jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk
jk jk jk jk jk jk jk jk jk

```
/* ftf.c */

/* ADAPTIVE DEREVERBERATION FILTER (FTF)
       This program is the main program for using the ftf algorithm
            in adaptive dereverberation.

       The reference d(n) is the reverberant signal.
       The same signal d(n) is delayed by DELTA samples to form the
            input data array y(n).
*/

/*************************************************************************/
#include <stdio.h>
#include <math.h>

#define NMAX 121


ftf(n, N, lambda, d, y, dest, w, fp)

        int n;                  /* number of points in input array, y[] */
        int N;                  /* size of filter (# of taps) */
        float lambda;           /* "forgetting" factor, nonstationary data */
        float d[];              /* input, desired (joint process) signal */
        float y[];              /* input data array */
        float dest[];           /* filter estimate of d[] */
        float w[];              /* filter coefficients (weights) */
        FILE *fp;               /* pointer to diagnostic file */
{
        double dhat;            /* latest estimate of d[] */
        double Y[NMAX+1];       /* vector of "past" values of y[n] */
        double A[NMAX+1];       /* forward prediction filter */
        double B[NMAX+1];       /* backward prediction filter */
        double C[NMAX];         /* "proximity estimation" filter, order N */
        double C1[NMAX+1];      /* "proximity estimation" filter, order N+1 */
        double W[NMAX];         /* transversal filter */
        double Temp[NMAX+1];    /* working vector */
        double e, ep;           /* forward est. error, f. prediction error */
        double r, rp;           /* backward est. error, b. pred. error */
        double oldalpha, alpha; /* old forward residual power, frp */
        double beta;            /* backward residual power */
        double gamma1, gamma;   /* "proximity estimation" errors */
        double eps, epsp;       /* transversal filter est. error, pred. error */
        double rescue;          /* should be > 0 for stability */

        int i, j, k;            /* general indexes */

/*** Initialize ***/
for(i=0; i<N; i++)
        Y[i+1] = A[i+1] = B[i] = C[i] = W[i] = 0.;

A[0] = B[N] = 1.0;
alpha = beta = d[0] * d[0];
gamma = 1.;

/***** Perform FTF Recursion *****/
for(i=0; i<n; i++) {
        /*** Build Y[] vector ***/
        for(j=N; j>0; j--)
                Y[j] = Y[j-1];
        Y[0] = y[i];

        multmat(A, Y, 1, N+1, 1, &ep);
        e = ep * gamma;
        oldalpha = lambda * alpha;
```

```
        alph    oldalpha + ep * e;
        gamma1 = gamma * oldalpha / alpha;

        scalmult(A, N+1, 1, Temp, -ep/oldalpha);
        addmat(C, N, 1, &Temp[1], &C1[1]);
        C1[0] = Temp[0];

        scalmult(C, N, 1, Temp, e);
        addmat(&A[1], N, 1, Temp, &A[1]);       /* A[0] = 1, untouched */
        rp = -lambda * beta * C1[N];
        rescue = 1 + rp * gamma1 * C1[N];

        if ( rescue < 0. ) {    /**** restart ****/
              fprintf(stderr, " ftf rescue at i=%d \n",i);
              for(j=0;j<N;j++)
                  A[j+1] = B[j] = C[j] = 0.;
                  alpha = beta = 1.;
                  gamma = 1.;
        }
        else {
              gamma = gamma1 / rescue;

                r = rp * gamma;
                beta = lambda * beta + rp * r;
                scalmult(B, N, 1, Temp, -C1[N]);
                addmat(C1, N, 1, Temp, C);
                scalmult(C, N, 1, Temp, r);
                addmat(B, N, 1, Temp, B);       /* B[N] = 1, untouched */
        }

        multmat(W, Y, 1, N, 1, &dhat);
        dest[i] = -dhat;
        epsp = d[i] + dhat;
        eps = epsp * gamma;
        scalmult(C, N, 1, Temp, eps);
        addmat(W, N, 1, Temp, W);

        /*** Print Diagnostic Info ***/
        if(fp) {
                fprintf(fp,"i=%3d ap=%.4f be=%.4f gm=%.4f gm1=%.4f e=%.4f ep=%.4f r=%.4f rp=%.4f eps=%.4f epsp=%.4f res=%.4f\n",
                        i,alpha,beta,gamma,gamma1,e,ep,r,rp,eps,epsp,rescue);
                if( !(i%10) || (i<10) ) {
                        fprintf(fp, "Y  ");
                        for(j=0; j<N+1; j++) fprintf(fp, "%6f  , Y[j]);
                        fprintf(fp, "\nA  ");
                        for(j=0; j<N+1; j++) fprintf(fp, "%6f  , A[j]);
                        fprintf(fp, "\nB  ");
                        for(j=0; j<N+1; j++) fprintf(fp, "%6f  , B[j]);
                        fprintf(fp, "\nC  ");
                        for(j=0; j<N; j++) fprintf(fp, "%6f  , C[j]);
                        fprintf(fp, "\nC1 ");
                        for(j=0; j<N+1; j++) fprintf(fp, "%6f  , C1[j]);
                        fprintf(fp, "\nW  ");
                        for(j=0; j<N; j++) fprintf(fp, "%6f  , W[j]);
                        fprintf(fp, "\n");
                }
        }

}       /* ends recursion (i) */

for(i=0; i<N; i++)
        w[i] = W[i];

}       /* ends ftf */
```

```
/* ftest.c */
/*****************************************************************************/
/* This is a main program to simulate the dereverberation filter (ftf).      */
/* It inputs desired (reference) file in dsp1d. The reference file is a       */
/* Reveberant signal.                                                        */
/* Filter coefficients, filter estimates, and filter error may be saved in   */
/* dsp1d files (optional).                                                   */
/* The error file corresponds to the processed signal                        */
/* FTF diagnostics may be saved in ASCII file "ftfile" (optional).           */
/*****************************************************************************/
/*****************************************************************************/

#include <math.h>
#include "/usr/local/dsp/dsp1.h"

main()

{
        struct header1d his, hid, hdro; /* dsp1d file headers */
        FILE *fopen(), *fp;             /* pointers for diagnostic file */

        int i, j, n;                    /* general indexes */
        int npt,delay;                  /* number of points to analyze */
        int response;                   /* user input */
        float signal[24000];            /* input signal data */
        float desired[24000];           /* desired signal */
        int startsig;                   /* first signal point to analyze */
        int startdes;                   /* first desired point to use */
        float estimate[24000];          /* estimate of desired signal */
        float error[24000];             /* desired - estimate */
        float time[24000];              /* time array for plotting */
        int order;                      /* filter order */
        float lambda;                   /* filter forgetting factor */
        float weights[MAXORDER];        /* filter coefficients */

/*** Read in the reference signal from dsp file ***/
read1d(&his, desired, "DSP file to input to filter?\n", "data");
npt = his.npts;

/*** Input the bulk delay ***/
fprintf(stderr,"delay ?\n");
scanf("%d",&delay);

/*** Form the input signal ***/
for(i=0;i<delay;i++)
   signal[i] = 0.0;
for(i=delay;i<npt;i++)
   signal[i] = desired[i-delay];

/*** Input the filter order ***/
fprintf(stderr, "Input filter order (1-42)?\n");
scanf("%d", &order);

/*** Input the FTF forgetting factor ***/
fprintf(stderr, "Input lambda (<=1.)?\n");
scanf("%f", &lambda);

fprintf(stderr,"Do you want to save FTF diagnostic information? (0,1)\n");
scanf("%d", &response);
fp = NULL;
if(response == 1) {
        fp = fopen("ftfile", "w");
        if(fp <= 0) {
                fprintf(stderr, "ftest: can't open ftfile. abort\n");
                exit();
```

```
        }
}
fprintf(stderr, "Working...\n");

/**** Call fast transversal filter routine ***/
ftf(npt, order, lambda, desired, signal, estimate, weights, fp);

for(i=0; i<npt; i++) {
        time[i] = i;
        error[i] = signal[i] - estimate[i];
}

/*** Print Weights to Screen ***/
for(i=0; i<order; i++) {
        fprintf(stderr,"W[%2d] = %6f   ", i, weights[i]);
        if(!((i+1)%4) || (i+1 == order)) fprintf(stderr, "\n");
}

fprintf(stderr,"Do you want to save filter weights? (0,1)\n");
scanf("%d", &response);
if(response == 1) {
        hdro.npts = order;
        hdro.si = 1000.;
        writeld(&hdro, weights, "Enter output file name\n", "d");
}
fprintf(stderr,"Do you want to save filter estimate? (0,1)\n");
scanf("%d", &response);
if(response == 1) {
        hdro.npts = npt;
        hdro.si = 1000.;
        writeld(&hdro, estimate, "Enter output file name\n", "d");
}
fprintf(stderr,"Do you want to save filter error? (0,1)\n");
scanf("%d", &response);
if(response == 1) {
        hdro.npts = npt;
        hdro.si = 1000.;
        writeld(&hdro, error, "Enter output file name\n", "d");
}
}       /* ends ftest.c */
```