# A Psychoacoustically Motivated Speech Enhancement System

by

Siddhartan Govindasamy

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

February 1, 2000

Author________________________________________________________

Department of Electrical Engineering and Computer Science
February 1, 2000

Certified by____________________________________________________

Dr. Thomas F. Quatieri
Thesis Supervisor

Accepted by____________________________________________________

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

A Psychoacoustically Motivated Speech Enhancement Algorithm
by
Siddhartan Govindasamy

Submitted to the
Department of Electrical Engineering and Computer Science

February 1, 2000

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

## ABSTRACT

A new method is introduced to perform enhancement of speech degraded by acoustic noise using the psychoacoustic property of masking. The goal of this algorithm is to preserve the natural quality of the noise while keeping the speech perceptually intact. Distortion masking principles based on prior work of Gustafsson are used to derive a hybrid gain function comprising a function minimizing speech distortion and another minimizing noise distortion. The system is implemented in floating-point software and was tested against several existing algorithms. In a forced choice listening test, the new system was preferred over the Enhanced Variable Rate Codec (EVRC) noise suppression algorithm in 88% of the cases. Informal listening tests showed preferable speech quality than Gustafsson's algorithm. As a front end to a vocoder, the new system was preferred over the other two by all the test subjects. Ideas on future work in speech enhancement are also explored.

Thesis Supervisor: Thomas F. Quatieri
Title: Senior Staff, MIT Lincoln Laboratory

# Acknowledgements

The research for this project was carried out at Qualcomm Inc. San Diego California as part of the MIT VI-A internship program. I would like to thank my supervisor at Qualcomm Inc. Mr. Samir Gupta for all his guidance, support and help. I would also like to express my gratitude to Dr. Thomas Quatieri of MIT Lincoln Laboratory for supervising my thesis, for all the guidance, feedback and wonderful discussions. Thanks are also due to Dr. Saed Younis at Qualcomm Inc. for his guidance and advice and Dr. Robert Dunn at Lincoln Laboratory for his help in designing the perceptual tests and feedback on the data.

I would also like to thank all my friends for their support and for helping out in the perceptual tests. Finally, I would like to thank my family for their guidance, support, understanding and all that they have done for me.

# Contents

# Table of Figures

# Chapter 1: Introduction

Noise suppression in speech systems is important for a variety of reasons. In today's highly competitive mobile telephone market for instance, it is important and often necessary to have a good noise suppression algorithm. The goal of these algorithms is to reduce the amount of background noise in a noisy speech signal while minimizing distortion of the speech, and keeping the residual noise sounding natural.

For mobile telephony, noise suppression is important as a front end to voice coders (vocoders). Vocoders use redundancy in the information content of speech signals to perform signal compression. Most voice coding algorithms use models of speech production in order to achieve their goal. Hence, if a vocoder is presented with a speech signal that has high noise content, one can expect poor performance because of the fact that the noise in the signal is modeled poorly by the speech production models used in the vocoders. Preliminary experiments in this research using the TIA standard Enhanced Variable Rate Codec (EVRC) [1] without noise suppression have shown this clearly.

The increased popularity of smaller and smaller mobile telephones has also further given rise to the need for good noise suppression. Smaller telephones result in increased distance between the mouth of the user and the microphone. Consequently, more sensitive and more omni-directional microphones are needed. This in turn causes more background noise to be picked up by the microphones. In hands-free carkit situations, even more noise is picked up because of the use of highly sensitive omni-directional microphones and the increased noise levels due to car engines, air-conditioning and wind.

Current noise suppression algorithms leave much room for improvement. The traditional methods of performing noise suppression fall short in a number of different ways. The spectral subtraction algorithms result in musical residual noise [2, 3]. Musical noise takes form of short-term tones that have a "gurgling" effect. Wiener filtering requires an estimate of the speech power spectrum as well as the noise power spectrum and also results in musical noise, albeit not as severe as in spectral subtraction. Also, Wiener filtering is a minimum mean square error (MMSE) algorithm which is not the minimum perceived error. Other algorithms such as signal to noise ratio (SNR) based spectral attenuation algorithms [1] cause unacceptable levels of speech distortion for high degrees of noise suppression. The background research section of this thesis elaborates more on these algorithms and their characteristics.

The objective of this thesis is to develop a noise suppression system that results in low levels of both noise and speech distortion while achieving a high degree of noise suppression. The psychoacoustic property of masking [4] will be used to achieve this target. This property has been exploited successfully for several years in vocoders [5, 6] as well as audio compression schemes such as the Motion Picture Experts Group (MPEG) [7] standard. It has recently been successfully used in several speech enhancement algorithms as well [2,3,15].

The principle of distortion masking which has been used in audio coders [6, 7] and the speech enhancement algorithm due to Gustafsson et. al [3] will be used in this thesis. This thesis expands on work done by Gustafsson et. al. by introducing a hybrid gain function that uses Gustafsson's results as well as some new ideas. The goal of the hybrid function is to minimize noise distortion for highly noisy portions of the speech and minimize speech distortions for portions that are high in speech content. Equations to achieve this goal are derived and then implemented in a floating point C-simulation. Speech data with added car noise was used to test the system. A binary forced choice test was then performed to evaluate the system from a subjective point. The system introduced here was preferred over that of the TIA EVRC noise suppressor [1] in the subjective tests. 10 subjects picked the new noise suppressor over the EVRC noise suppressor 89 % of the time. Informal testing also suggests that the hybrid function is preferrable to the Gustafsson algorithm in terms of speech quality with noise quality that is comparable.

This document is organized as follows. After the introduction chapter, a chapter on the background research done prior to undertaking the major work on this thesis is presented. This includes information on psychoacoustics as well as state of the art speech enhancement algorithms. This is followed by a chapter on the theory behind the hybrid algorithm. The derivation of the hybrid algorithm as well as some discussion on its expected performance are given in this chapter. A chapter on implementation details follows. The results obtained using this algorithm are presented and discussed in the next chapter. Lastly, the results are discussed and ideas on future work are explored.

# Chapter 2: Background Research

## 2.1 Introduction

This chapter describes the background research that was done before the major research work was undertaken. It is designed to provide the reader with information on psychoacoustics as well as state of the art speech enhancement algorithms. In the first section, the psychoacoustic properties used in this and other systems are described. The second section describes existing speech enhancement algorithms including classical methods as well as recent, psychoacoustically motivated algorithms.

## 2.2 Psychoacoustic Properties

This section describes the two psychacoustic properties that are most widely used in perceptually based audio signal processing algorithms. Namely, they are the critical band theory and the masking property. The advantage of using these psychoacoustic models in performing audio signal processing is that algorithms can be designed to match the human auditory system better.

### 2.2.1 Critical Band Theory

In the process of hearing, the vibration of air molecules gets propagated from outside the human body by various mechanical means until they reach the cochlea. The cochlea contains a membrane called the basilar membrane (BM). The vibrations of the BM are converted into electrical impulses and are transmitted to the brain [10].

The BM can be modeled as a bank of about 10,000 overlapping bandpass filters [4, 10] which perform a conversion of sound pressure level in time into the frequency domain. The bandwidths of these filters are known as the critical bands of hearing. It has been found that the perceived loudness over the range of a critical band is dependent on the intensity of sound in that band [11]. Experiments have also shown that the human ear can distinguish frequencies with poorer resolution for higher frequencies [4, 10], i.e. the widths of the critical bands increase with frequency. Thus, the nonlinear Bark scale for the critical bands of hearing was conceived. The Bark scale denotes frequencies matched to the widths of the critical bands. For instance, a Bark frequency of 7 means that there are 7 critical bands between that frequency and zero. Therefore it is a frequency scale that is better suited to describing hearing related frequencies. In the low end of the scale, the bandwidths of the band pass filters are found to be about 100 Hz and in higher frequencies the band widths reach up to about 3000 Hz [4, 10]. Thus, the distance in Hz between Bark frequency 0 and Bark frequency 1 is approximately 100 Hz since the critical bandwidths for low frequencies is approximately 100 Hz . The following expression due to Zwicker [4] describes the mapping from Hz frequencies to Bark frequencies.

$$b = 13\arctan\left(0.76\times 10^{-13} f\right) + 3.5\arctan\left[\left(\frac{f}{7500}\right)^2\right]$$

where $b$ is the bark frequency corresponding to the Hz frequency $f$ .

Most perceptually motivated algorithms, however, do not use a continuous mapping from Hz frequencies into bark frequencies. The commonly used method is to use a fixed-band approach. The range of hearing is divided into a number of bands depending on the maximum frequency that is handled by the system. For speech processing systems, typically, 16 bands are used [1, 5, 6]. As a result, the frequency values are quantized along the bark scale and are commonly referred to as Band Numbers or Bark Numbers [14]. The following table shows an example of critical band boundaries that are used in this system to convert Hz frequencies into Bark frequencies. This critical band structure is also used in [5].

| Band Number (Quantized Bark Frequency) | Upper Bound Frequency (Hz) |
|---|---|
| 1 | 109.375 |
| 2 | 359.375 |
| 3 | 484.375 |
| 4 | 609.375 |
| 5 | 734.375 |
| 6 | 859.375 |
| 7 | 1046.875 |
| 8 | 1078.125 |
| 9 | 1421.875 |
| 10 | 1671.875 |
| 11 | 1921.875 |
| 12 | 2234.375 |
| 13 | 2609.375 |
| 14 | 3046.875 |
| 15 | 3484.375 |
| 16 | 3984.375 |

**Table 2.1: Critical Band Boundaries**

The fixed band approach is not entirely accurate because it is known that the bands in our ear are overlapping and are far greater in number. However, it was found to be sufficient for the purposes of the system in this project as was the case in [1, 5, 6, 14]. It requires significantly fewer computations compared to a more accurate model of 10,000 overlapping filters.

### 2.2.2 Masking Properties of the Human Auditory System

The phenomenon of auditory masking is the effect of one sound on our ability to perceive others [10]. In a sense it is a measure of the ability of one sound to drown another. Research in psychoacoustics has shown that we cannot perceive weak signals that are in the time and frequency vicinity of stronger signals. These two properties are called temporal and frequency masking respectively. In this system, temporal masking is not considered. Only masking in the frequency domain is used. This is because the entire system is based on frequency domain processing of the input signals. Furthermore, temporal masking is a property that has been less used in the field of audio signal processing due to difficulty in properly quantifying this property. In contrast, frequency masking has been used in perceptual audio coders [5, 6, 7] as well as other psychoacoustically motivated speech enhancement algorithms [2, 3, 15]. The following describes the property of frequency masking.

If we have a tone at a certain frequency (masker), there exists a threshold (masking threshold) of sound pressure level (SPL) in adjacent frequencies below which other signals are imperceptible. The general shape of the masking curve for a single tone at frequency v is shown in Figure 2.1. Signals that have a sound pressure level (SPL) below the broken lines are imperceptible in the presence of the tone at v.

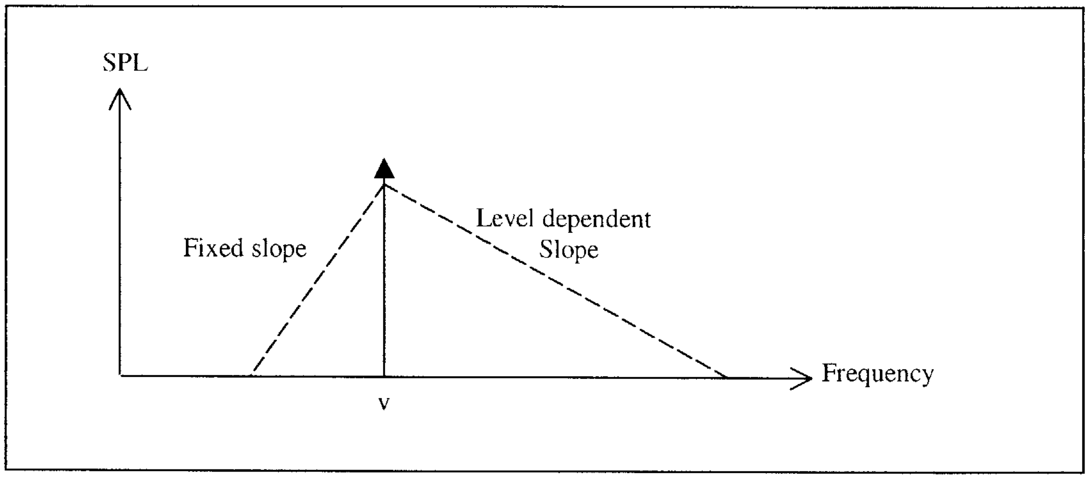

**Figure 2.1: General Shape of Masking Curve Due to Tone**

It has been found that the ability of a tone to mask other signals is greater in frequencies higher than that of the tone [4]. Hence, in Figure 2.1 the masking threshold for frequencies higher than v has a greater slope. The slope in the higher frequencies is dependent on the SPL of the tone at frequency v. For frequencies lower than v, it can be modeled as having a fixed slope [12].

The masking phenomenon is not only limited to tones as maskers. Noise is also able to mask other signals. In fact, noise maskers have been found to be more effective than tonal maskers [4, 10]. Noise maskers are able to mask signals that are about 4-6 dB greater in SPL than corresponding tonal maskers [14]. Adjustments to the computed masking curve have to be made to account for this difference as is done in [2, 6]. The computation of the masking curve is detailed in Chapter 4.

The effects of individual maskers are additive [10]. This means that the masking due to each frequency component can be added up to come up with a "global" masking threshold. This threshold tells us what is or is not perceptible across the spectrum.

The phenomenon of noise masking has also led to the concept of perceptual entropy [6, 22]. Perceptual entropy is the measure of how much error can be added to a signal so that the resultant signal is perceptually the same as the original. The basic idea is that if the error that is added to a signal falls below the masking threshold, then that error cannot be perceived. This concept is exploited in perceptual audio coders like [5, 6, 7, 22] whereby bit allocation is done in a manner such that the quantization noise falls below the masking curve and cannot be perceived by the listener.

It has to be noted that the masking curves for various frequencies and SPLs have been measured experimentally by researchers in psychoacoustics. The actual computation of the masking threshold is just an estimation of the actual masking that occurs. Thus, corrections to the computed masking threshold have to be performed. Typically, the masking threshold is lowered to obtain a more conservative estimate. This will result in more room for error. The details of the correction to the masking curve that is done in this system are described in Chapter 4.

## 2.3 Current Noise Suppression Algorithms

This section provides a description of existing noise suppression algorithms. It begins by giving a general overview of noise suppression systems. This is followed by a subsection describing the classical methods of performing noise suppression (i.e. without considering psychoacoustic effects). The next subsection describes more recent psychoacoustically motivated algorithms.

### 2.3.1 General Noise Suppression Systems

Most noise suppression systems use frequency domain filtering methods. The following describes a generalized noise suppression system. It has to be noted though that other methods not involving frequency domain filtering do exist as for example, the pitch filtering method described later in this chapter.

Assuming an additive noise model, consider a signal $x(n) = s(n) + b(n)$ where $x(n)$ is the sum of the speech signal $s(n)$ and the background noise $b(n)$. Because of the short time stationarity of speech, the conversion into the frequency domain has to be performed on a frame by frame basis with overlapping windows. So, in the frequency domain, we have $X(f,m) = S(f,m) + B(f,m)$. Where $f$ is the frequency and $m$ is the frame number.

In general, noise suppression algorithms apply a gain function $H(f,m)$ to the signal giving the output $Y(f,m) = H(f,m)X(f,m)$. This gain function (also called the spectral weighting function) has the goal of reducing the magnitude of the noise components while keeping the speech components intact. The different weighting functions correspond to different suppression algorithms.

Most noise suppression algorithms make use of noise estimates in the computation of $H(f,m)$. In general, long term stationarity of noise is assumed. Voice activity detection (VAD) is used to detect pauses in speech during which only noise is present. The noise power spectrum in these frames is averaged over time to obtain an estimate of the noise power spectrum density (PSD). This estimate is then used in the computation of $H(f,m)$. Figure 2.2 illustrates a general noise supression system:

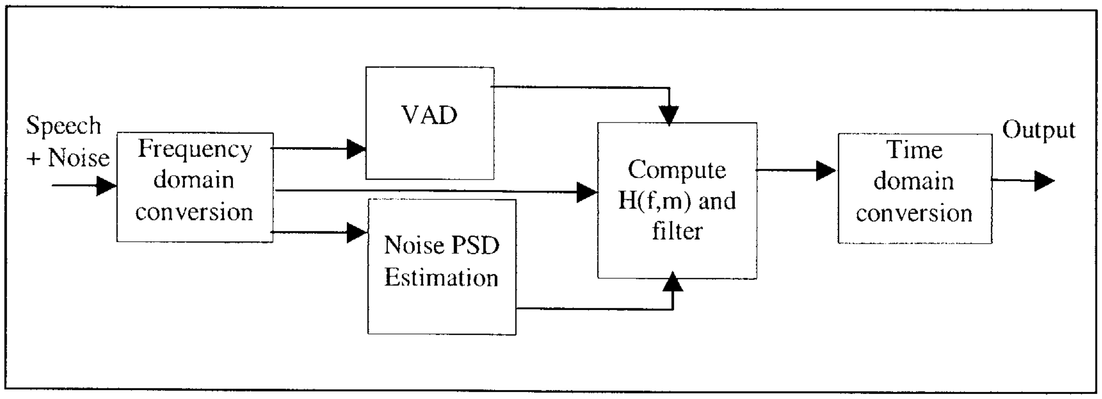

**Figure 2.2: Overview of General Noise Suppression Systems**

### 2.3.2 Spectral Subtraction

One of the classical methods of performing noise suppression is by spectral subtraction [8]. This algorithm assumes long term stationarity of the noise and short term stationarity of the speech signal. It also assumes that the noise and the speech are uncorrelated. When the VAD detects speech, a fraction of the magnitude of the estimated noise is subtracted out of the magnitude of the input signal. Hence, the term spectral subtraction. The resulting magnitude is combined with the phase of the noisy input signal to produce the output. Thus, we have:

$$\begin{aligned} |Y(f,m)|^{\gamma} &= |X(f,m)|^{\gamma} - \alpha^{\gamma}|\tilde{B}(f,m)|^{\gamma} \quad && \text{when } |X(f,m)|^{\gamma} > \alpha^{\gamma}|\tilde{B}(f,m)|^{\gamma} \\ &= 0 && \text{otherwise} \end{aligned}$$

so that:

$$H(f,m)=\left(1-\frac{\alpha^{\gamma}\left|\tilde{B}(f,m)\right|^{\gamma}}{\left|X(f,m)\right|^{\gamma}}\right)^{\frac{1}{\gamma}} \qquad (2.1)$$

where $\tilde{B}(f,m)$ is the noise estimate for frame m. $\alpha$ is the amount of suppression and $\gamma$ is an exponent that controls the abruptness of the transition between full attenuation and no attenuation. Typically used values for $\gamma$ are $\gamma=1$ for magnitude spectral subtraction where the magnitude of the noise is subtracted from the magnitude of the input signal and $\gamma=2$ for power spectral subtraction where the power of the noise is subtracted from the power of the input signal. The magnitude of the output signal is then given by:

$$\left|Y(f,m)\right|=H(f,m)\left|X(f,m)\right|.$$

$\left|Y(f,m)\right|$ is then combined with the phase of $X(f,m)$ to produce the output as follows:

$$Y(f,m)=\left|Y(f,m)\right|e^{\angle[X(f,m)]j}.$$

The following is an analysis of the mean value of the output using power spectral subtraction. From the additive noise model,

$$X(f,m)=S(f,m)+B(f,m).$$

Thus,

$$\left|Y(f,m)\right|^{2}=\left|S(f,m)\right|^{2}+\left[\left|B(f,m)\right|^{2}-\alpha^{2}\left|\tilde{B}(f,m)\right|^{2}\right]+S^{*}(f,m)B(f,m)+S(f,m)N^{*}(f,m)$$

If $S(f,m)$ and $B(f,m)$ are uncorrelated and if we subtract the noise out fully, i.e. $\alpha = 1$,

$$E\left\{ \left|Y(f,m)\right|^2 \right\} = E\left\{ \left|X(f,m)\right|^2 \right\}$$

Thus, the expected power of the estimate equals the expected power of the original speech. A similar argument can be made for magnitude spectral subtraction (for which the magnitude of the noise is subtracted from the magnitude of the input signal) where the expected magnitude of the output equals the expected magnitude of the speech.

This algorithm relies heavily on having an excellent estimate of the noise power spectrum. Since by definition, noise is random, it is impossible to get a perfectly accurate estimate of the noise power spectrum. As a result, some frequency lines get subtracted out wrongly resulting in short term tonal components being left behind in the output signal. This unnatural sounding noise is called musical noise. This is the biggest problem posed by spectral subtraction.

### 2.3.3 Wiener Filtering

The Wiener filter is a minimum mean square error (MMSE) algorithm that minimizes the expected error between the estimated speech and the actual speech signal. It assumes the uncorrelatedness of the speech and noise. The following is the derivation of the Wiener filter gain function. From the orthogonality principle, for the minimum mean square error, the error signal has to be orthogonal to the input signal [26]. The error is given by the following expression.

$$\varepsilon(n) = y(n) - s(n)$$

For the error to be orthogonal to the input signal,

$$E\{[y(n)-s(n)]\ x(l)\}=0 \qquad \text{for all } l$$

$$E\{[s(n)-y(n)]\ x(l)\}=0$$

but $y(n)$ is $x(n)$ filtered by $h(n)$ i.e.

$$y(n)=\sum_{k=-\infty}^{\infty}h(k)x(n-k)$$

so, we have

$$E\left\{\left[s(n)-\sum_{k=-\infty}^{\infty}h(k)x(n-k)\right]x(l)\right\}=0 \qquad \text{for all } l$$

After some algebraic manipulation,

$$\sum_{k=-\infty}^{\infty}h(k)R_{XX}(n-k-l)=R_{SX}(n-l) \qquad \text{for all } l$$

where $R_{XX}(n)$ is the autocorrelation sequence of $x(n)$ and $R_{SX}(n)$ is the cross correlation between $s(n)$ and $x(n)$.

It can be noted that the summation above can be written in terms of a convolution as follows:

$$h(n)*R_{XX}(n-l)=R_{SX}(n-l)$$

where $*$ denotes the convolution operator. Taking the Fourier transform on both sides,

$$H(f)P_X(f)=S_{SX}(f)$$

where $S_{SX}(f)$ is the Fourier transform of $R_{SX}(n)$.

Assuming uncorrelatedness of the speech and noise,

$$H(f)[P_S(f)+P_B(f)]=P_S(f)$$

which leads to the expression for the Wiener filter

$$H(f) = \frac{P_S(f)}{P_S(f) + P_B(f)}. \tag{2.2}$$

### 2.3.4 SNR Based Weighting Rules

Yet another method of performing noise supression is the use of the instantaneous estimated SNR to perform attenuation of noise [1]. A measure of the instantaneous SNR is used to compute the gain function as follows:

$$H(f) = \min\left\{G\left(\frac{X(f,m)}{\tilde{B}(f,m)}\right), 1\right\}$$

A possible function $G(\bullet)$ is the following limited linear function that is used in [1]

$$G(k) = \max[\mu(k - k_o) - \xi, \xi] \tag{2.3}$$

where $\mu$ is an appropriately chosen slope, $\xi$ is the minimum gain and $k_o$ is the largest value of $k$ for this minimum gain.This gain function attenuates regions of low SNR and leaves regions of high SNR intact. Regions of SNR $k_o$ or less are attenuated by the full amount, $\xi$. The slope $\mu$ controls how much difference in attenuation should be applied for a given difference in SNR. Figure 2.3 shows how the gain varies with the SNR for (2.3).

This weighting rule preserves noise characteristics fairly well (provided that the noise is stationary). This is because the frequency components that have high noise content have low SNR and get attenuated by a fixed amount. However, much of unvoiced speech is lost as it tends to be low in SNR and as a result causes $G(k)$ to be small. In addition, obtaining a measure of instantaneous SNR is itself difficult to do accurately due to the difficulty in obtaining good noise and speech PSD estimates.

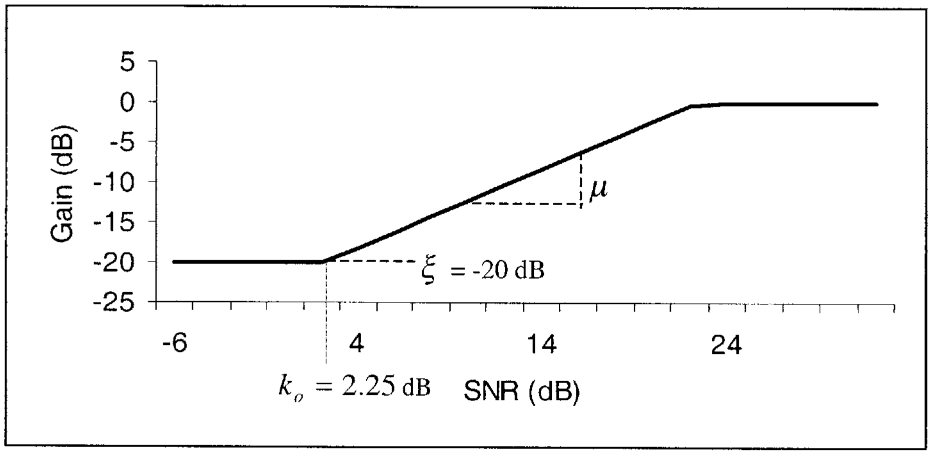

**Figure 2.3: Gain versus SNR for SNR Based Algorithm**

### 2.3.5 Weighting Rule of Ephraim and Malah

Ephraim and Malah presented a set of rules that drastically reduce the amount of musical residual noise while having a high degree of noise suppression [19, 20, 21]. This method uses a complicated gain function which involves measures of *a priori* and *a posteriori* SNRs. The following are the equations used in the computation of this gain function for the *m*th input frame.

$$H(m,f)=\frac{\sqrt{\pi}}{2}\sqrt{\left(\frac{1}{1+R_{post}(m,f)}\right)\left(\frac{R_{prio}(m,f)}{1+R_{prio}(m,f)}\right)}$$
$$\times M\left[\frac{R_{prio}(m,f)+R_{post}(m,f)R_{prio}(m,f)}{1+R_{prio}(m,f)}\right]$$

where

$$M[\theta] = e^{-\frac{\theta}{2}}\left[(1+\theta)\ I_0\left(\frac{\theta}{2}\right) + \theta\ I_1\left(\frac{\theta}{2}\right)\right]$$

$R_{prio}(m,f)$ is the *a priori* SNR for frame $m$. The term *a priori* is used because it is an estimate of what the SNR in the current frame is based on the previous frame. $R_{post}(m,f)$ is the *a posteriori* SNR for frame m. The term *a posteriori* is used because it is a direct estimate of what the SNR in the current frame is based on the information from the current frame only. $I_0(f)$ and $I_1(f)$ are the modified Bessel functions of the zeroth and first order respectively [20, 21]. $R_{prio}(m,f)$ and $R_{post}(m,f)$ can be obtained using the following equations presented in [21] which are slightly simpler than the ones used in the original work [20]:

$$R_{post}(m,f) = \frac{|\ X(m,f)\ |^2}{|\ \tilde{B}(m,f)\ |^2} - 1$$

$$R_{prio}(m,f) = (1-\alpha)P[R_{post}(m,f)] + \alpha\frac{|\ H(m-1,f)X(m-1,f)\ |^2}{|\ \tilde{B}(m,f)\ |^2}$$

where $P(x) = x$ for $x \geq 0$ and $P(x) = 0$ for $x < 0$ which ensures that $R_{prio}(m,f)$ will always have a positive value. This is required for the square-root operation in the gain function computation to produce a real value. $\alpha$ is a weighting factor satisfying $|\alpha| < 1$. $\alpha$ is set to a value close to 1.

Thus, the *a priori* SNR is a combination of the current *a posteriori* SNR and an estimate of the previous SNR. In the expression for $H(m,f)$, the *a priori* SNR is the dominant factor [21]. It can be shown numerically that $R_{prio}(m,f)$ is a heavily smoothed version of $R_{post}(m,f)$ when $R_{post}(m,f)$ is small and that $R_{prio}(m,f)$ is just a delayed version of $R_{post}(m,f)$ when $R_{post}(m,f)$ is very large [21].

Thus, for low SNR cases, $R_{prio}(m,f)$ is heavily smoothed which results in smoother values in the low regions of the spectrum. Since musical noise occurs in the low SNR regions of the spectrum, the smoother gain function translates to reduced musical noise. In the higher SNR cases, $R_{prio}(m,f)$ tracks $R_{post}(m,f)$ which is a direct estimate of the SNR. This results in an SNR based gain function much like that described in the previous section. Thus, the Ephraim and Malah algorithm in a nutshell can be described as one that uses a fast tracking SNR estimate for high SNR frequency components and a highly smoothed SNR estimate for the low SNR components.

This algorithm drastically reduces the amount of musical noise compared to the spectral subtraction and Wiener filtering methods [21]. However, some musical residual noise can be noticed if the smoothing of the *a posteriori* SNR estimate is not gradual enough. At the same time, if the smoothing is too slow, the beginnings and ends of words that are low in SNR will cause distortions in the speech due to the fact that the *a posteriori* SNR estimate is too slow in catching up with the transient speech [21].

### 2.3.6 Pitch Filtering

One of the earliest methods of speech enhancement was by pitch filtering [13, 16]. This method does not require frequency domain filtering although it can also be done in the frequency domain. The basic premise of this system is to exploit the periodicity of speech signals. To understand this method, a rough model of speech production is needed.

Human speech can be roughly divided into unvoiced and voiced speech. Voiced speech is produced by forcing air through the glotis so that the vocal cords vibrate quasi-periodically [24]. These periodic air pulses get filtered through the vocal tract and produce a periodic output. This corresponds to vowel like sounds. One can demonstrate this to oneself by feeling the vibration of the throat while pronouncing vowel like sounds. Unvoiced speech on the other hand is produced by turbulent, high velocity air being forced through various portions of the vocal tract. This corresponds to sounds like "ss" and "ff".

Voiced speech can thus be modeled as a periodic pulse train filtered through a time-varying filter. This results in a harmonic signal. Comb or pitch filtering performs the enhancement by enhancing the periodic components of the voiced speech. In order to do this, an estimate of the pitch period is needed. Various methods of doing this have been proposed [1, 24]. Comb filtering is then performed on the signal to enhance the harmonics. This can be done in the time domain via an infinite impulse response (IIR) filter of the form:

$$H(z) = \frac{1}{1 - bz^{-L}}$$

where $L$ is the estimated pitch period and $b$ is a factor called the long term prediction gain that controls the widths of the "teeth" of the comb filter. The closer the value of $b$ to unity, the sharper the peaks of the comb filter are. Figure 2.4 illustrates a pitch filtering system for speech enhancement.

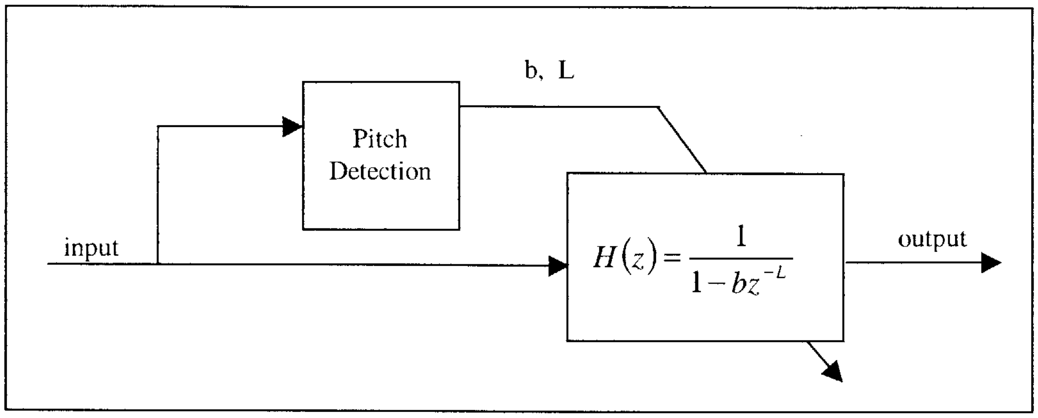

**Figure 2.4: Comb Filtering Speech Enhancement System**

Other methods include frequency domain filtering as well as time domain comb filters of other forms as with

$$H(z)=\frac{1}{\sum_{k=1}^{N}(-1)^{2k}z^{-2k}}.$$

The most obvious problem with comb filtering is that it only enhances the harmonic components of the speech. Unvoiced speech is not enhanced. Neither do the non-periodic components of voiced speech. This results in an unnatural sounding output and loss of intelligibility. For instance, using a pitch filtering algorithm, words like steam and team can easily be confused due to the loss of the unvoiced "s" sound. In addition, pitch period estimation is an extremely difficult problem, even when dealing with clean speech [24]. Poor pitch period detection results in a chorus-like sounding output due to new harmonics created by enhancing the wrong frequency components.

### 2.3.7 Psychoacoustically Motivated Rules

Recently, several psychoacoustically motivated speech enhancement algorithms have surfaced [2, 3, 15]. All of them involve frequency domain filtering with various gain functions. The three major approaches are as follows:

- To attenuate less where noise is heavily masked [2]
- To ensure that distortions are masked [3]
- To increase the amount of masking in order to cause more noise to be imperceptible [15]

#### *2.3.7.1 Virag Algorithm*

Virag [2] uses a subtractive-type algorithm in order to perform noise reduction. The basic gain function that is used is a generalized form of spectral subtraction due to Berouti et. al [25]. Berouti's gain function is as follows:

$$H(f)=\left[1-\alpha\left(\frac{|\tilde{B}(f)|}{|X(f)|}\right)^{\gamma_1}\right]^{\gamma_2} \qquad \text{for} \quad \left(\frac{|\tilde{B}(f)|}{|X(f)|}\right)^{\gamma_1}<\frac{1}{\alpha+\beta}$$

$$=\left[\beta\left(\frac{|\tilde{B}(f)|}{|X(f)|}\right)^{\gamma_1}\right]^{\gamma_2} \qquad \text{otherwise}$$

where $\alpha$ is the oversubtraction factor which controls how much noise should be subtracted out of the speech. $\beta$ is the spectral flooring factor which determines the minimum value taken by the gain function. $\gamma_1 = \frac{1}{\gamma_2}$ is an exponent that determines the abruptness of the transition from full suppression to no suppression.

This function has the flexibility of variable subtraction parameters. Less attenuation can be achieved by lowering $\alpha$ and $\beta$. Virag sets $\gamma_1 = 2$ and $\gamma_2 = 0.5$ which results in a power spectral subtraction algorithm. The values of $\alpha$ and $\beta$ are varied according to the level of masking at the frequency of concern. This is done on a frame by frame basis as follows:

$$\alpha_m(f) = F_\alpha[\alpha_{\min}, \alpha_{\max}, T(f)]$$

$$\beta_m(f) = F_\beta[\beta_{\min}, \beta_{\max}, T(f)]$$

where m is the frame number, $\alpha_{\min}$, $\alpha_{\max}$, $\beta_{\min}$ and $\beta_{\max}$ are the maximum and minimum values of $\alpha$ and $\beta$ respectively. $T(f)$ is the level of the masking threshold. $F_\alpha$ and $F_\beta$ are functions that cause $\alpha$ and $\beta$ to be minimized when the masking threshold is high and vice-versa. This results in a low amount of suppression in cases where the masking threshold is high. The advantage of this lies in the fact that when the masking threshold is high, more noise is masked. Thus, less noise needs to be attenuated. If less attenuation is performed, then less speech distortion would result because more of the speech signal is left intact.

### *2.3.7.2 Gustafsson's Algorithm*

Gustafsson et. al. [3] , proposed an algorithm to keep the noise perceptually equivalent to an attenuated version of the input noise. This algorithm ensures that any distortions in the noise fall below the masking curve and are thus imperceptible. The main drawback to this system is that it does not consider the effects of the algorithm on the speech signal. While the noise is kept sounding natural, the speech is distorted. Since the Gustafsson algorithm plays a major role in this system, a thorough description and discussion of this method is given in Chapter 3.

### *2.3.7.3 Czyzewski's Algorithm*

Czyzewski et. al. proposed a method of increasing the amount of masking present in a speech frame in order to make the noise imperceptible [15]. This is done by classifying the frequency components in each frame as useful and useless components. The useless components are the noise components and the useful components are considered to be the speech.

The masking threshold is raised by applying a gain of $\alpha$ to the useful components of the input signal such that all the useless frequency components are masked. The value of $\alpha$ is found by numerically solving a complicated implicit function. This function is obtained by computing the masking threshold due to the useless frequency components, $T_1(f)$ and an expression for the masking threshold due to the useful frequency components with gain $\alpha$ , $T_2^{\alpha}(f)$. The fact that the masking due to different frequency components is additive [4, 10, 14] is used to find an expression for the global masking threshold as follows:

$$T(f) = T_2^{\alpha}(f) + T_1(f) \qquad (2.2)$$

$\alpha$ is found by solving

$$T(f) = \max\{D^p\}$$

where $D^p$ is the set of power values of the useless frequency components. Thus, this value of $\alpha$ ensures that the power of all useless frequency components is less than or equal to the masking threshold which results in the useless components being masked.

There are several drawbacks to this system. Firstly, the frequency components have to be classified as useful or useless. Errors in this classification can cause either speech components to be masked or noise components to be amplified. Furthermore, in order to completely mask all audible noise in a frame, the value of $\alpha$ has to be very high. Experiments that were conducted using a similar method of raising the masking threshold showed that gains of around 60 dB are not uncommon. Such large gains can cause errors in the classification of useful and useless components to be enhanced significantly. For instance, a noise component that was mistakenly classified as useful will be amplified by 60 dB. Conversely, a speech component wrongly classified as useless will be 60 dB lower than the speech components that were properly classified. Thus, distortions in the speech and noise are a big problem when the classification of frequency components is not done accurately . Furthermore, the computation of $\alpha$ has to be done numerically leading to increased overhead as well as accuracy problems.

# Chapter 3: Theory of Hybrid Algorithm

## 3.1 Introduction

As is mentioned in Chapter 2, psychoacoustically motivated algorithms can be divided into three basic approaches. They are, less attenuation of noise when noise is already masked, the raising of the masking threshold to mask more noise and the masking of distortions in the speech or noise. The algorithm that is used in this thesis is one that masks distortions in the speech and in the noise. This approach was chosen because it offers an explicit way in which to ensure natural sounding speech and noise. The last section of this chapter compares this algorithm to the approach used by Virag, Czyzewski and Gustafsson [2, 3, 15].

The goal of this algorithm is to enhance the speech in the input signal while keeping the noise level the same, where the word enhance is used to mean amplify. It is implemented via a frequency domain filtering system. Figure 3.1 gives a brief description of such a system. The gain function used in this algorithm can be divided into two parts. The first part is a gain function that is due to Gustafsson et. al. [1] which keeps the noise distortions imperceptible without explicitly considering the distortions to the speech. The second is a gain function based on Gustafsson's work that ensures speech distortions are imperceptible without an explicit consideration of the noise characteristics. The gain functions are then combined via a weighting factor. Section 3.2 details the derivation of Gustafsson's gain function. The next section is devoted to deriving the new weighting rule. This is followed by a section describing the combination of the two gain functions.

Then, the derivation of a suitable weighting factor to perform the combination is described. The last section in this chapter is an analysis of the performance of this gain function and a comparison with existing psychoacoustically motivated algorithms.

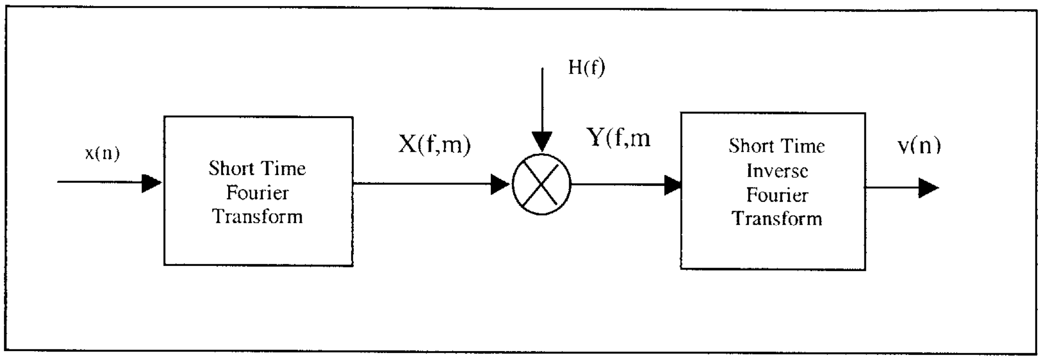
H(f)
x(n)
Short Time Fourier Transform
X(f,m)
Y(f,m
Short Time Inverse Fourier Transform
v(n)

**Figure 3.1: General Frequency Domain Filtering System**
where m is the frame number of the windowed input signal

## 3.2 Gustafsson's Gain Function, $H_G$

The goal of the $H_G$ function is to suppress the noise in the input signal in a manner such that the noise is undistorted. The distortion of the speech is not explicitly considered. The following is the derivation of the $H_G$ function which is based on Gustafsson's method for keeping noise distortions masked [3].

Let the input signal, $x(n) = s(n) + b(n)$, where $s(n)$ is the speech signal and $b(n)$ is the noise signal. Let the output signal be $y(n)$. The desired signal, noise suppressed version of $x(n)$, is $d(n) = s(n) + ab(n)$ where $a$ is the noise suppression factor. For notational simplicity, let the windowed frequency domain version of the signal be denoted by the capitalized letter of the time signal. Also, for convenience, we shall drop the notation of the frame number as this variable is not used in the derivation of this algorithm. With reference to Figure 3.1, assuming deterministic inputs, in the frequency domain, we have:

$$X(f) = S(f) + B(f)$$

$$Y(f) = H(f)X(f) = H(f)S(f) + H(f)B(f)$$

$$D(f) = S(f) + aB(f) \tag{3.1}$$

The noise error in the frequency domain, defined as the difference between the desired and the output noise signal is given as:

$$E_B(f) = H(f)B(f) - aB(f)$$

$$E_B(f) = [H(f) - a]B(f)$$

Similarly, the speech error is given as:

$$E_s(f) = [H(f) - 1]S(f)$$

For stochastic signals, we take the power spectral densities of the signals (PSDs) assuming uncorrelatedness of the speech and the noise. Thus, we have:

$$P_X(f) = P_S(f) + P_B(f)$$

$$P_Y(f) = H^2(f)P_S(f) + H^2(f)P_B(f) \tag{3.2}$$

$$P_D(f) = P_S(f) + a^2 P_B(f) \tag{3.3}$$

$$P_{E_S}(f) = [H(f) - a]^2 P_S(f) \tag{3.4}$$

$$P_{E_B}(f) = [H(f) - 1]^2 P_B(f) \tag{3.5}$$

As is done in perceptually based audio coders [5, 6, 7], in order to keep the noise error masked (hence, imperceptible) the PSD of the noise error has to be below the masking curve. Thus,

$$P_{E_B}(f) < P_T(f)$$

where $P_T(f)$ is an estimate of the power value of the masking threshold of the output signal where the power value refers to the square of the linear value of the masking threshold.

from (3.4)

$$[H(f)-a]^2 P_B(f) < P_T(f)$$

which leads to

$$a - \sqrt{\frac{P_T(f)}{P_B(f)}} < H(f) < a + \sqrt{\frac{P_T(f)}{P_B(f)}}.$$

We now have a range of values of $H(f)$ for which the output noise will sound like the desired noise. In order to attenuate the speech as little as possible, the largest value of $H(f)$ satisfying this range is chosen. This is because the less attenuation is performed the more intact the speech signal will be. Thus,

$$H(f) = a + \sqrt{\frac{P_T(f)}{P_B(f)}}.$$

In frequencies that are low in SNR in a *noise segment*, $\sqrt{\frac{P_T(f)}{P_B(f)}}$ has a low value. This is because the noise power in that frequency is high whereas the masking threshold is low.

In frequencies with high speech content, the noise power will be low and the masking threshold will be high due to the masking from the speech component. Thus $\sqrt{\frac{P_T(f)}{P_B(f)}}$ will have a high value. Thus, the gain function behaves as expected. For frequencies that are high in speech content, the gain is high. For frequencies with high noise content, the resulting gain is low. In order to keep $H(f)$ from possibly taking values greater than 1, it has to be limited as follows:

$$H(f) = \min\left( a + \sqrt{\frac{P_T(f)}{P_B(f)}}, 1 \right)$$

This algorithm performs well so far as the noise is concerned. As expected, the noise in the output is perceptually equivalent to the noise in the input. However, the speech gets distorted. This is not surprising considering that this algorithm does not explicitly take the speech distortions into account. Gustafsson himself reports of speech attenuation of around 7 dB for low values of SNR [3]. The experiments in this thesis have confirmed this.

## 3.3 The $H_S$ Algorithm

The goal of the $H_S$ algorithm is to keep the speech in the output sounding the same as an enhanced version of the original speech. This is accomplished by ensuring that the speech distortions remain masked. The effects of this algorithm on the noise isn't explicitly considered. The following is the derivation of the $H_S$ function which is based on ideas from Gustafsson's work in [3].

Let the desired signal be a speech enhanced version of the input signal as follows:

$$D(f) = AS(f) + B(f)$$

where $A$ is the speech enhancement factor[1].

Algebraic manipulation similar to that done in deriving the Gustafsson function leads to the following expression for the PSD of the speech error:

$$P_{E_S}(f) = [H(f) - A]^2 P_S . \tag{3.6}$$

To keep the speech error imperceptible, the PSD of the speech error has to be less than the estimated masking threshold of the output.

Thus,

$$P_{E_S}(f) < P_T(f)$$

substituting (3.6)

$$[H(f) - A]^2 P_S < P_T(f)$$

which leads to

$$A - \sqrt{\frac{P_T(f)}{P_S(f)}} < H(f) < A + \sqrt{\frac{P_T(f)}{P_S(f)}}$$

Thus, we now have a range of values for which the gain function produces an output that is perceptually equivalent to the desired signal. In order to enhance the noise

[1] This desired output is needed because the use of the same desired output as in $H_G$ results in a gain function with no parameter to vary the level of noise suppression or speech enhancement.

as little as possible while satisfying this condition, the minimum value of $H(f)$ in this range is selected.

Thus,

$$H(f) = A - \sqrt{\frac{P_T(f)}{P_S(f)}}$$

The term $\sqrt{\frac{P_T(f)}{P_S(f)}}$ is small for the frequencies in which there is high speech content. This is because $P_S(f)$ will be large in these frequencies. Thus, in high speech content frequencies, $H(f) \approx A$. For frequencies with low speech content (low SNR) but in a *speech segment*, $\sqrt{\frac{P_T(f)}{P_S(f)}}$ will be large. This is because $P_S(f)$ will be small for these frequencies and $P_T(f)$ will be quite large because of the masking due to other frequencies that do contain speech. Therefore, for frequencies with greater noise content, $H(f)$ will be small.

This function was tested and as expected was found to keep the speech perceptually equivalent to the desired speech. However, the noise characteristics are not guaranteed. It was observed that the level of noise that was present in the output speech was not predictable and was often too high and unnatural sounding. This is due to the fact that for frequencies that have high noise content but with a low masking threshold, both $P_S(f)$ and $P_T(f)$ are low in value which results in $H(f)$ having a value close to $A-1$. Thus, the noise in these frequencies doesn't get attenuated by much which is undesirable.

Therefore, for frequencies that are more noise-like, this gain function performs poorly. The following section describes a hybrid function that is used to overcome this problem.

## 3.4 The Hybrid Algorithm

In the previous two sections, two gain functions were described. The first was Gustafsson's algorithm that keeps the noise error imperceptible while applying the minimal amount of attenuation to the speech. The second was a gain function based on Gustafsson's work that keeps the speech error imperceptible while applying the minimal amount of gain to the noise. Since the Gustafsson gain function performs poorly for speech components and the $H_S$ function performs poorly for noise components, a hybrid of the two functions is used whereby $H_G$ is weighted more for more noiselike signals and $H_S$ is weighted more for more speech like signals. The following is the description of the combined gain function.

Let the desired signal in be a speech enhanced version of the input signal as follows:

$$D(f) = AX(f) + B(f) \tag{3.7}$$

To achieve this desired signal in a perceptual sense, the noise signal has to sound as if it were not attenuated. In order to do this, a modification to the Gustafsson gain function is necessary. This is because the $H_G$ function is derived using a noise suppressed version of the input signal as the desired output as in section 3.2. In order to use a hybrid function, both $H_G$ and $H_S$ have to give the same desired output. Otherwise, using a hybrid of both the functions will be difficult because both the functions have a different desired result.

The following shows that the use of $H_G$ with a particular desired noise suppression level will result in the same desired signal as that used in the $H_S$ function.

Consider the following signal,

$$X'(f) = S'(f) + B'(f)$$

$$= AX(f) \qquad (3.7)$$

$$= AS(f) + AB(f) \qquad (3.8)$$

$$\text{where } S'(f) = AS(f) \text{ and } B'(f) = AB(f) \qquad (3.9)$$

If we use the $H_G$ rule on $X'(f)$ with the noise attenuation factor $a = \frac{1}{A}$, from (3.7), the desired output would be

$$D'(f) = S'(f) + aB'(f)$$

from (3.8) $$= AS(f) + B(f)$$

from (3.6) $$= D(f)$$

Therefore, if we have $X'(f)$, to obtain the desired signal equal to $D(f)$ (the desired signal for the $H_S$ algorithm) we need the Gustafsson gain function to use an attenuation factor, $a = \frac{1}{A}$. That is, by using this value of $a$ in the Gustafsson algorithm the signal

$X'(f)$ will have a desired output equivalent to that of the $H_S$ algorithm with the input $X(f)$. The use of this attenuation factor results in the following gain function.

$$H'(f) = \min\left(\frac{1}{A} + \sqrt{\frac{P_T(f)}{P_{B'}(f)}}, 1\right)$$

where $P_{B'}(f)$ is the PSD of $B'(f)$ and $P_T(f)$ is the estimated masking threshold due to the output signal. The spectrum of the output signal is then given by:

$$Y(f) = H'(f)X'(f)$$

$$= \min\left\{\left(\frac{1}{A} + \sqrt{\frac{P_T(f)}{P_{B'}(f)}}\right)X'(f), X'(f)\right\}$$

Substituting (3.7)

$$= \min\left\{\left(\frac{1}{A} + \sqrt{\frac{P_T(f)}{P_{B'}(f)}}\right)AX(f), AX(f)\right\}$$

$$= \min\left\{\left(1 + A\sqrt{\frac{P_T(f)}{P_{B'}(f)}}\right), A\right\}X(f)$$

from (3.9) $B'(f) = AB(f)$, which means that the PSDs of the two signals are related by:

$$P_{B'}(f) = A^2 P_B(f).$$

Thus, in order to obtain the desired signal in (3.6) via Gustafsson's algorithm with the input $X(f)$, the gain function has to equal

$$H(f) = \min\left\{\left(1 + \sqrt{\frac{P_T(f)}{P_B(f)}}\right), A\right\}.$$

It should be noted that throughout the derivation of this gain function, the masking threshold that is used in the computation is the estimate of the masking threshold due to the final output signal.

Let $H_B(f)$ denote the gain computed from this method and $H_S(f)$ denote the gain function derived in section 3.3. $H_B(f)$ keeps the background noise perceptually intact and $H_S(f)$ keeps the speech perceptually equivalent to the desired enhanced speech.

If we have a measure of how noiselike or speechlike a particular frequency component is, we will be able to get a composite gain that is a combination of $H_B(f)$ and $H_S(f)$. Denoting this measure as $\alpha$ (described in Section 3.5) which ranges from 0 for frequency components that are purely noise and 1 for frequency components that are purely speech, we can compute a composite gain $H_{S,B}(f)$ as follows:

$$H_{S,B}(f) = \alpha(f)H_S(f) + [1 - \alpha(f)]H_B(f)$$

Thus, $H_{S,B}(f)$ is a composite gain function consisting of $H_B(f)$ which is a modified version of Gustaffson's rule (as derived earlier in this section) and $H_S(f)$, which is the function derived in Section 3.5. The contribution of these rules to the composite function is dependent on how noiselike or speech like a particular frequency component is. For frequency components in which the speech content is high, $H_{S,B}(f)$ takes on a value close to $H_S(f)$ which ensures that the hybrid function at that frequency component keeps the speech perceptually intact. Conversely, for frequency components that are high in noise content, $H_{S,B}(f)$ takes on a value close to $H_B(f)$ which results in intact noise. Thus we are able to keep speech components sounding the same as the desired speech and the noise components the same as the desired noise.

## 3.5 Speech Content Measure

The speech content measure $\alpha(f)$ has to reflect how speechlike or noiselike frequency component $f$ is. In a sense, all gain functions that perform frequency domain filtering are indeed such measures. This is due to the fact that the purpose of these gain functions is to attenuate frequency components that are noise-like and to let speech components through unmolested. As a result, the gain for noise components is low and the gain for speech components is close to 1. The speech content measure that was selected to use in this system is based on the SNR weighting rule described in Chapter 2. There are several reasons for this.

For simplicity, a measure based on the speech enhancement algorithms described in Chapter 2 were considered. Spectral subtraction was discarded because the speech PSD estimation uses the results of a simple spectral subtraction subsystem. See Chapter 4 for details. Thus, some musical noise artifacts are expected to be present in the output through the $H_S$ algorithm which uses the speech PSD estimate. For the frequency components in which there is musical noise, the speech content measure should be low so that the $H_G$ function (which keeps the noise consistent) contributes more to the hybrid gain function. Using a spectral subtraction based speech content measure will result in false high values for $\alpha$ at the very frequencies for which $H_s$ produces musical noise. This will result in $H_s$ contributing more to the hybrid gain. Thus, musical noise will be heard in the output. Experiments that verified this were conducted.

A Wiener filtering based approach was not selected because it does not give an explicit way to control the manner in which the transition of $\alpha$ from one to zero occurs. Equation (2.2) does not contain a variable that can control this transition. Therefore, the Wiener filter based speech content measure will not be able to explicitly control the manner in which the contributions of $H_S$ and $H_B$ to the hybrid gain function vary. The SNR based measure provides this via a slope factor.

The speech content measure used for this system is a limited linear function of the log of the SNR as in (2.3). The following are the equations that are used to compute it.

$$\alpha(f) = 10^{\min\left\{G\left[\log_{10}\left(\frac{X(f)}{\tilde{B}(f)}\right)\right],1\right\}}$$

where $$G(k) = \max[\mu(k - k_o) + \xi, \xi]$$

$\xi$ is the minimum value of $\log\alpha$, $k_o$ is the value of the SNR when this first occurs. $\mu$ is the parameter that controls how the value of $\alpha$ varies as it moves from its minimum value to 1.

This results in the following curve for the $\log\alpha(f)$ versus the log of the SNR for a particular frequency. It shows how $\alpha$ changes with SNR on a log scale.

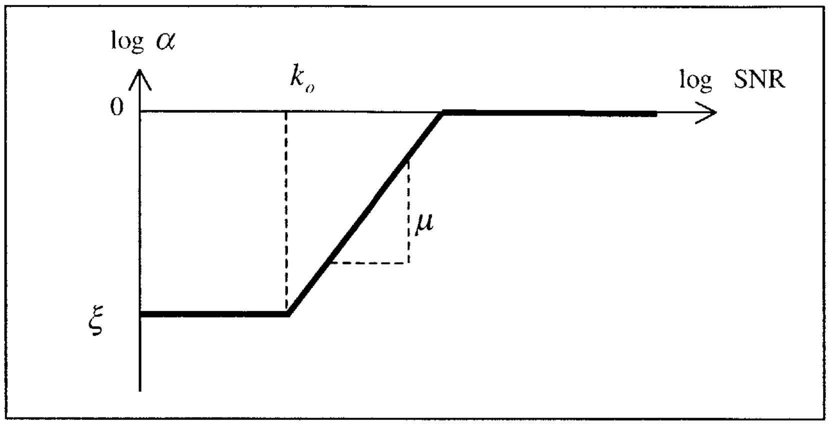

**Figure 3.2: Graph of Speech Content Measure vs SNR**

Figure 3.2 can be interpreted as a graph of what the fractional contribution of $H_S$ and $H_G$ are to the composite function. When $\alpha(f)$ is greater, $H_S$ contributes more and vice-versa. The maximum fraction of the hybrid function that is from $H_B$ is $10^{\xi}$ and the maximum fractional contribution from $H_S$ is 1 (i.e. when $\log\alpha = 0$). The parameter $\mu$ controls how rapidly the transition between $H_B$ and $H_S$ is when the SNR increases.

The parameters $\xi$, $\mu$ and $k_o$ were all obtained experimentally. The first set of parameters were taken from the gain function used in the TIA speech enhancement system [1]. These parameters were then experimentally improved on. This was done by changing the values of the parameters and listening to the output signal and examining the SNR values during the processing. If the noise distortion was great, the value of $\xi$ was lowered. The values of the other parameters were changed accordingly. The final set of parameters that were chosen are $\mu = 0.8$, $\xi = -34$ dB and $k_o = 2.25$.

The following graph illustrates how the contribution of the $H_S$ and $H_G$ algorithms on the hybrid gain changes with SNR for a given frequency $f$. The vertical lines marked $H_S$ and $H_G$ denote the points at which the hybrid gain function completely comprises $H_S(f)$ and $H_G(f)$ respectively.

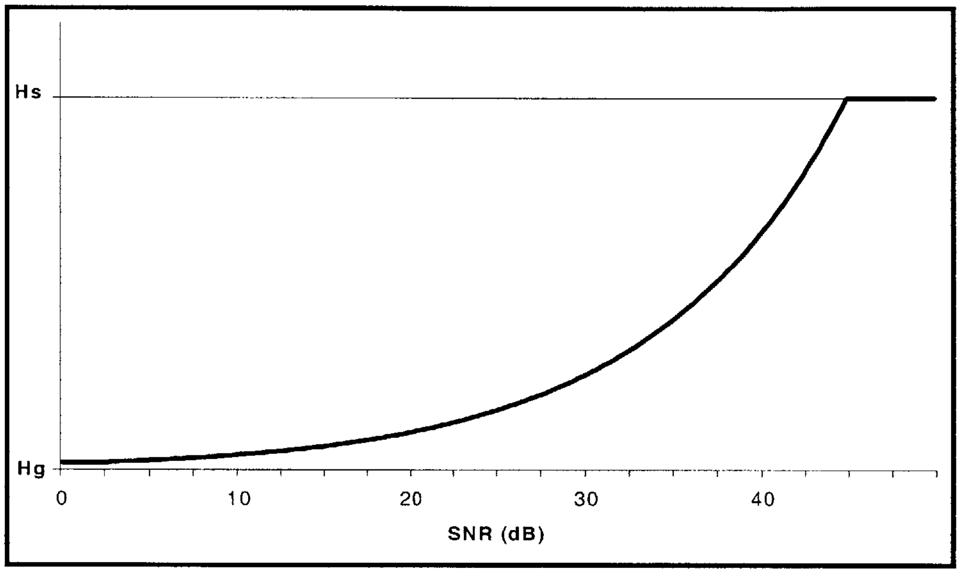

**Figure 3.3: Contribution of $H_S$ and $H_G$ to hybrid function vs SNR**

## 3.6 Discussion of Hybrid Function

### 3.6.1 Comparison with TIA Noise Suppressor

The Hybrid gain function can be written as follows:

$$H_{S,B}(f) = \alpha(f) \times \max\left[\left(A - \underbrace{\sqrt{\frac{P_T(f)}{P_S(f)}}}_{MSR}\right), 1\right] + [1-\alpha(f)] \times \min\left[\left(1 + \underbrace{\sqrt{\frac{P_T(f)}{P_B(f)}}}_{MNR}\right), A\right]$$

It is thus a function of 3 variables, namely the SNR (used to determine $\alpha(f)$), a masking to speech ratio (MSR) and a masking to noise ratio (MNR). The desired characteristic of this function is that it have a high value for high SNR (high speech/low noise), low MSR (high speech) and high MNR (low noise) and vice versa. This can be verified by inspecting the above expression. The traditional method of SNR vs gain suppression curves will not illustrate the operation of this gain function due to the fact that $H_{S,G}$ is strongly dependent on the MSR and MNR, quantities which are meaningless in algorithms that do not use the masking property. Furthermore, the three quantities are not independent. Thus, in order to illustrate the operation of this gain function, a suppression curve using real data is used. The value of the gain for different SNR's is averaged to obtain a value for the gain given a particular SNR. Figure 3.4 illustrates the output gain as a function of SNR for both the TIA noise suppresion algorithm and the hybrid function. The SNR varies from –12 dB to 19 dB.

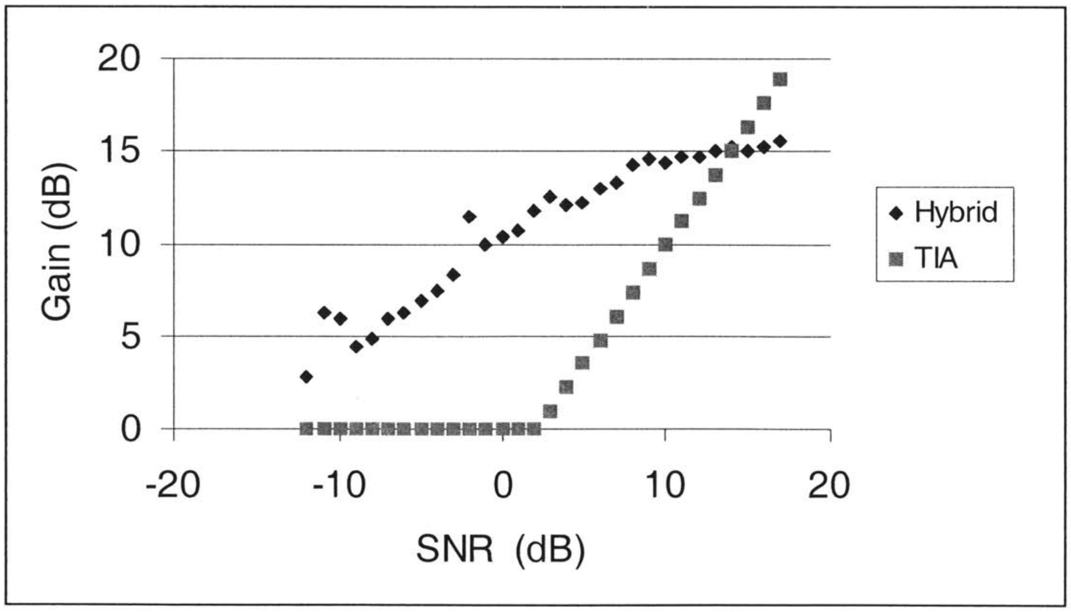

**Figure 3.4: Gain Curve for TIA Noise Suppressor and Hybrid Function**

From the graph, we can see that the hybrid function does what is required of a noise suppression algorithm in that it has a low value for the gain in low SNR cases and a high value of the gain for high SNR cases. However, the range of values over which the hybrid function varies is smaller than the range of values over which the TIA gain function varies. From the psychoacoustic model, we know that signals of lower strengths are perceptually equivalent to larger signals provided the difference in levels is masked. Thus, in order to make a signal component sound like it is enhanced by some factor A, would require some gain that is equal to (when there is no masking present) or less than A. This explains the smaller range of values that the gain function takes. If the gain function varies over a smaller range of values, the difference in the gain applied to the lower SNR components versus the higher SNR components is smaller. Thus, the less distortion in the output signal can be expected. The results obtained and discussed in Chapter 5 clearly indicate this.

### 3.6.2 Comparison with Psychoacoustic Noise Suppressors

This method can be expected to perform at least as good as Gustafsson's algorithm [3]. This is due to the fact that the hybrid gain function includes Gustafsson's algorithm which works best for high noise content portions of the speech segment. In high speech content frequencies, the second gain function which is more suitable for high speech content frequencies contributes more to the hybrid function. Thus, if the speech content measure $\alpha$ is chosen well, this algorithm will perform at least as good as Gustafsson's algorithm.

The Virag algorithm does not take into account the distortions in the speech and noise as explicitly as is done in the algorithm in this thesis. As is described in Chapter 2, this algorithm changes the suppression parameters based on the level of masking present. Where there are high levels of masking, the amount of suppression is low. This is done in a heuristic manner compared with that done in the algorithm in this thesis which explicitly ensures that distortions are masked.

Czyzewski's algorithm involves the raising of the masking threshold by amplifying the speech in order to mask noise as is described in Chapter 2. Czyzewskitries to ensure that the noise is inaudible which is a stricter requirement than merely reducing the level of the noise as is done in the system in this thesis. Experiments conducted in this research have shown that in order to completely mask noise components, gains in excess of 50 dB have to be applied to some of the speech portions of the spectrum. Such high gains will cause a lot of distortion in the speech. In any case, Czyzewski's algorithm has the goal of masking the noise completely which as expected will require extremely high levels of attenuation.

# Chapter 4: Implementation

## 4.1 Introduction

The system in this thesis was implemented in a modular form in a floating point simulation. The following is a block diagram of the system in this thesis.

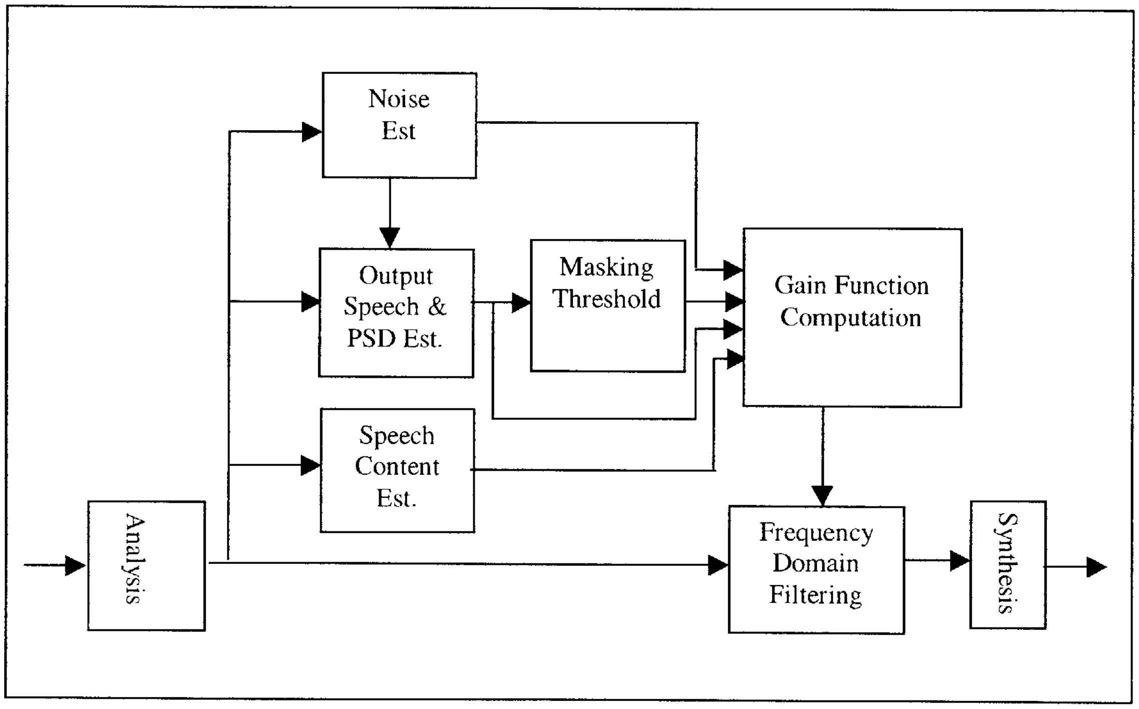

**Figure 4.1: Block Diagram of System**

The following sections describe the implementation of each of the blocks in detail. Appendix A lists the full code for the implementation.

## 4.2 Analysis – Synthesis

### 4.2.1 Overview

The analysis-synthesis subsystem performs the conversion from the time to the frequency domain and vice versa. Due to the short-time stationarity of speech signals [24], the input needs to processed on a frame by frame basis using some amount of overlap. Thus, the conversion to the frequency domain is performed on windowed-overlapped segments of the input data. Due to this, the analysis subsystem can be divided into a windowing system followed by a Discrete Fourier Transform (DFT) computation. Likewise, the synthesis subsystem can be divided up into an Inverse DFT (IDFT) computation followed by overlap addition to obtain the correct output samples.

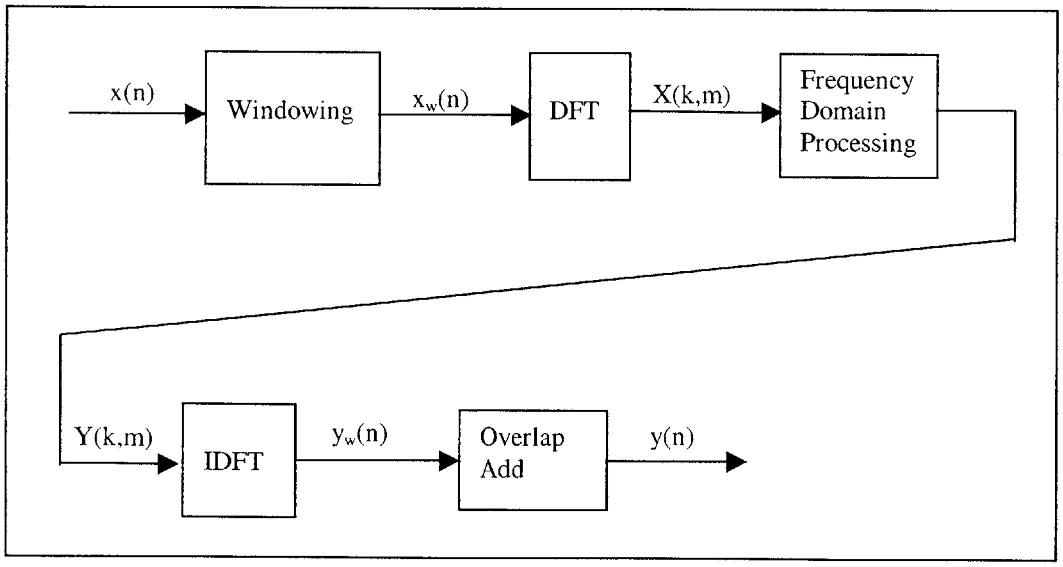

**Figure 4.2: Analysis-Synthesis Subsystem**

### 4.2.2 Windowing

The requirements of the windowing system are as follows:

- The window length should provide sufficient frequency resolution. The longer the window in timer, the narrower it is in frequency. The window length has to be long enough so that the input spectrum will not be smeared across many spectral lines.

- The window shape should have side lobes that are sufficiently lower than the main lobe to ensure minimal leakage of sidelobe structure into adjacent frequencies.

- The length of the window should not be too long such that noticeable pre-echo occurs. This and the next requirement are time resolution issues which are in contention with frequency resolution

- The overlap and window length should be such that non-stationary events are handled properly. This is to ensure that sudden changes in the input signal, e.g. the beginning of words do not get processed poorly due to the fact that most of the samples in the window capture the silence between words.

- Framing delay should be kept minimal. This is to enable real-time operation of the system.

The window used in this system is a standard Hamming window of length 320 with 75% overlap and an 80 sample frame size. This corresponds to a 40 ms window with a 10 ms frame size. The 75% overlap provides for nearly perfect signal reconstruction [24] using overlap add techniques. Figure 4.3 shows how this overlap provides nearly perfect signal reconstruction.

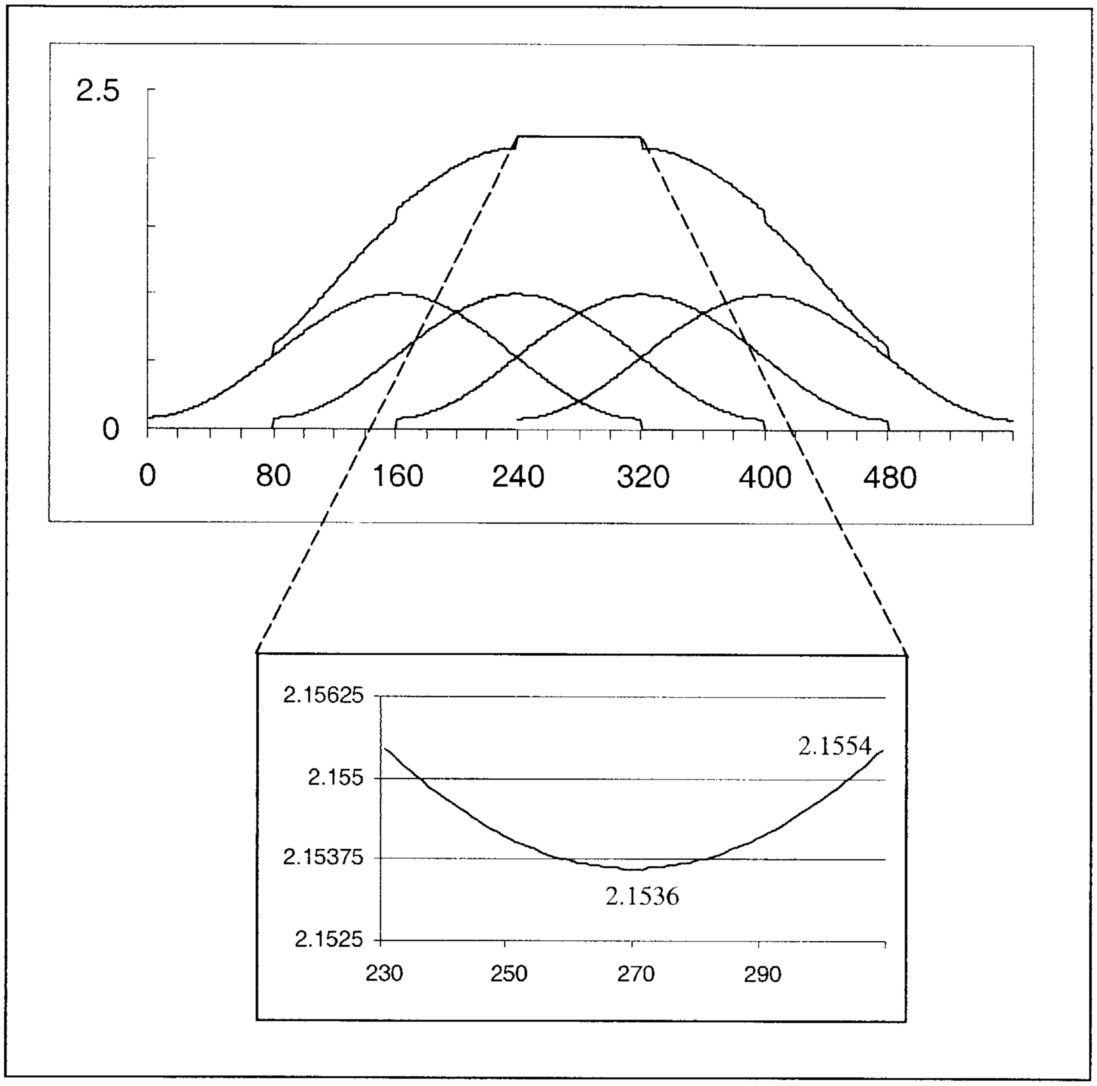

**Figure 4.3: Illustration of 75% Overlap-Add Hamming Window**
With the 80 sample frame used in this system, sample nos. 240 to 319 which are in the approximately flat region above, are used in the output. The windowing error in using this method is less than 0.1%. That is, the "flat" region that is zoomed in above is flat to up to 0.1%.

The sampling rate used in this system is 8 kHz. Thus the frame size is 10 ms and the window is 40 ms wide. This window offers good frequency resolution as shown by Figure 4.4 which shows that this window gives about 50 Hz of resolution with sidelobes 43 dB below the main lobe. Such high resolution is sufficient for the masking threshold estimation because the smallest critical band is approximately 100 Hz. The fine structure of the spectrum needs to be preserved in order for the masking threshold computation to be accurate for the low frequency components where the critical bandwidths are small.

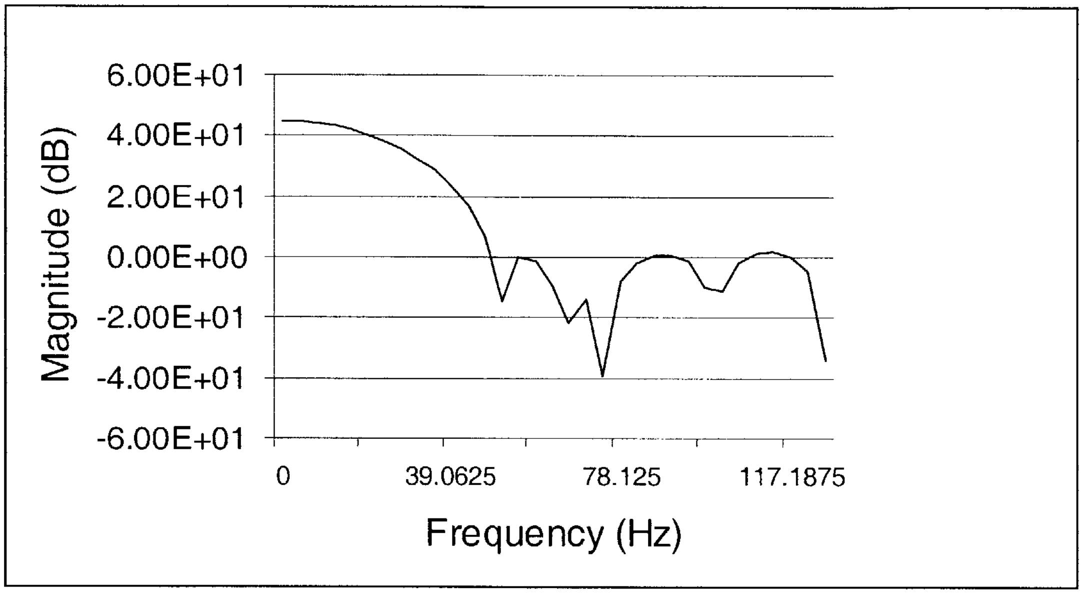

**Figure 4.4: Fourier Transform of Window, Zoomed on Positive Mainlobe**

The 320 point window translates to a 320/8000 = 40 ms window. Pre-echo and poor handling of non-stationary events were examined to determine if this window can be used. It was found to be satisfactory in that the amount of reverberation noticed was small. Some pre-echo was noticed but the time resolution cannot be improved because of the requirements on frequency resolution.

### 4.2.3 Discrete Fourier Transform (DFT)

The time to frequency domain conversion of the signal is performed using a 512 point real to complex Fast Fourier Transform (FFT) as per [1]. The equation for this is given by the following:

$$X(k,m) = \frac{2}{512}\sum_{n=0}^{511} x_w(n)e^{-j2\pi nk/512}$$

where $X(k,m)$ is the DFT sample of the frequency component at $k \times \frac{8000}{512}$ Hz in windowed input frame $m$. $x_w(n)$ is the value of the $n$-th windowed input sample in frame $m$.

This equation causes the negative frequency components to be folded onto the positive components resulting in 256 complex points spanning from 0 Hz to 3,9999 Hz. The frequency resolution of this system is 8000/512 = 15.625 Hz per spectral line.

The conversion back to the time domain is performed using the following equation which performs a complex to real inverse FFT (IFFT).

$$y_w(n) = \frac{1}{2}\sum_{n=0}^{511} Y(k,m)e^{j2\pi nk/512} \qquad 0 \le n < 512$$

where $y_w(n)$ is the nth output sample, $Y(k,m)$ is the kth DFT sample frequency in time frame $m$.

## 4.3 Noise PSD Estimation

### 4.3.1 Overview

The noise PSD estimation is performed by smoothing the power spectrum of the input signal in periods between speech. The speech/non-speech frame discrimination is performed using the VAD from [1]. This VAD uses an instantaneous SNR estimate and a long term spectral deviation measures to determine whether an input frame contains speech.

### 4.3.2 Voice Activity Detector

The voice activity detector used in this system is borrowed from [1]. The system uses a quantity called the voice metric sum which is a measure of the SNR in each frame. An instantaneous SNR estimate is obtained on a critical band basis using the bands in Table 2.1. The SNR is estimated as the square root of the ratio of the input signal power to the estimated noise PSD.

The SNR estimate for each channel is then quantized into integer values and limited between 0 and 89. The quantized SNR is then used as an index into a voice metric table shown in Table 4.1 in Appendix B which indicates how voicelike a particular channel is based on the quantized SNR value. As shown in the table, the voice metric is greater for greater SNR. Also, for low SNR values, the voice metric is at is minimal value of 2. Thus the voice metric is a measure of the SNR for which low SNR values all map to the same voice metric.

The voice metric for each critical band is added to obtain the voice metric sum, $\zeta$ which is used in determining whether a frame was speech or noise. The principle behind this is that frames with high total SNR, are most likely speech frames.

The long term spectral deviation is another parameter used in determining whether or not a frame is voice or noise. This is done by taking the difference between the input signal energy and a long term average energy in each critical band. The following equation describes the computation of the long term spectral deviation:

$$\text{spectral deviation in frame } m,\ \Delta(m) = \sum_{0}^{15} V\left[SNR_q(i,m)\right]$$

where V is the voice metric vector in Table 4.1, $SNR_q(i,m)$ is the quantized SNR for critical band i in frame m. Note that there are 16 critical bands in Table 2.1. The principle behind the use of the long term spectral deviation is that noise frames will have an input spectrum that is similar to the long term average spectrum. Speech frames are more likely to have a spectrum that is significantly different from the long term average spectrum.

The algorithm for the VAD incorporates the VM_SUM as well as the long term spectral deviation parameters as is given in pseudo-code in Appendix C. The explanation for each block is in C style comments. The update flag is set when the noise should be updated.

### 4.3.3 Noise Power Spectrum Estimation

The noise power is estimated in a over each critical band as per Table 2.1 to estimate the noise power spectrum. The noise power is smoothed when the VAD indicates a noise frame using the following IIR filter:

$$E_n(m+1,i) = \max\{E_{\min}, 0.9E_n(m,i) + 0.1E_{ch}(m+1,i)\}$$

where $E_n(m,i)$ is the noise energy for the $m$-th frame in critical band $i$, $E_{\min}$ is the minimum allowed channel energy and $E_{ch}(m,i)$ is the energy in critical band $i$ in the $m$-th frame. The critical band energy is found by averaging the energy of all the spectral lines in that particular band.

## 4.4 Estimation of Speech Signal and Speech Power Spectrum

### 4.4.1 Output Speech Estimate

An estimate of the output speech signal as well as an estimate of the speech PSD are needed in this algorithm. The output signal estimate is needed to estimate the masking threshold due to the output signal as is done in [2, 3]. The speech PSD estimate is used directly in the computation of the gain function. The first part of this section details the computation of the output speech estimate and the second part details the computation of the speech PSD estimate.

The clean speech signal is estimated using a modified version of simple magnitude spectral subtraction using equation (2.1). This is a similar estimate to the one that was used by Gustafsson and Virag to compute the masking threshold [2, 3] of the output signal. This was chosen because spectral subtraction has the characteristic that the speech signal is preserved well. The noise, however, is distorted and is musical in nature. The tonal components which cause the musical noise are of relatively low magnitude compared with the speech components. Thus, their contribution to the masking curve is not significant in comparison with the contribution of the speech components. Therefore, the effect of the tonal components are not audible in the output.

The method used is a modified version of spectral subtraction which preserves noise characteristics between words and sentences. During speech frames, the standard spectral subtraction algorithm is used. In noise frames, a uniform attenuation is applied to all frequencies. Thus, no musical noise occurs in noise frames.

The voice metric sum computed by the VAD is used to detect pauses between words. This is done by comparing the voice metric sum $\zeta$ (see section 4.3.2) against a fixed threshold, $\zeta_T$. During these pauses, the gain for all frequencies are uniformly floored to the value of the desired attenuation. Thus, the output speech estimate is done as follows.

$$\begin{array}{ll} \left| \tilde{S}(f) \right| = \left| X(f) \right| + \alpha \left| \tilde{B}(f) \right| & \text{when } \left| X(f) \right| \geq \alpha \left| \tilde{X}(f) \right| \text{ and } \zeta > \zeta_T \\ \left| \tilde{S}(f) \right| = 0 & \text{when } \left| X(f) \right| < \alpha \left| \tilde{X}(f) \right| \text{ and } \zeta > \zeta_T \\ \left| \tilde{S}(f) \right| = \alpha \left| X(f) \right| & \text{otherwise.} \end{array}$$

The phase of the noisy speech is used as the phase of the speech estimate for all frames as follows

$$\angle\tilde{S}(f) = \angle X(f)$$

where $\tilde{S}(f)$ is the estimate of the output speech.

The value for $\zeta_T$ was computed experimentally to be 35. This was determined by plotting the values of the voice metric sum on top of the input speech signal and finding (by inspection) the largest value of the voice metric sum that occurs during noise. This was done for several different input files and the lowest value of the threshold from those tests was taken as the final value.

### 4.4.2 Speech PSD Estimate

The speech PSD estimate is done using an average of the power of the speech estimate. This is approach is reasonable. Although the speech estimate from the spectral subtraction contains musical noise, the algorithm that is used is robust to noise in the speech PSD estimate. This is due to the fact that musical noise occurs in the low SNR frequencies for which the Gustafsson algorithm is weighted strongly in the hybrid function (see Chapter 3). The Gustafsson algorithm does not use the speech PSD estimate and is derived based on keeping the noise sounding natural. Therefore, it is possible to use an estimate of the PSD based on spectral subtraction and still obtain good results. The PSD is estimate on a frame by frame basis when the VAD detects speech instead of noise as follows:

$$P_S(m,f) = (\alpha_{PSD} - 1)P_S(m-1,f) + \alpha_{PSD}\tilde{S}^2(m-1,f).$$

## 4.5 Estimation of the Masking Threshold

### 4.5.1 Basic Computation

Several different methods have been proposed for the estimation of the masking threshold [6, 5, 7, 12, 14, 22]. The method used in this project is based on Terhardt and Sen's methods of computing the masking threshold combined with ideas from the MPEG standard and Johnston. This method is used because it provides a straightforward measure of how much masking is present at each frequency point.

In Terhardt [12] and Sen's [5] methods, the masking threshold has a slope dependent on the masker level for frequencies higher than the maskers and a fixed slope (fixed over the bark scale) for frequencies below the masker [5, 12, 14]. Johnston uses a fixed slope for both sides of the maskee [6, 24]. Therhardt proposed the following equations to compute the masking of a frequency component u (maskee) by frequency component v (masker) as described in Chapter 2 [12, 5].

$$\text{slope, } s_v = -24 - \frac{230}{f_v} + 0.2L_v \text{ dB / Bark} \qquad \text{for} \quad u > v$$

$$s_v = -27 \text{ dB/ Bark} \qquad \text{otherwise}$$

where $f_v$ is the frequency of the vth frequency component and $L_v$ is the sound pressure level of the vth frequency component.

The level of the masking at frequency $u$ due to the signal component at frequency $v$ is given by the following linear (in dB) relationship:

$$Th(f_u, f_v) = L_v - s_v(z_v - z_u)$$

where $z_v$ and $z_u$ are the quantized bark values from Table 2.1 of the $u$-th and $v$-th frequency components respectively.

The overall masking at frequency component $u$ due to all the other frequency components is given by the sum of the masking due to the indivudual frequency components [5, 10, 12]. The overall masking threshold is then given by:

$$Th(f_u) = 10 \log \sum_{v \neq u} 10^{\frac{(L_v - s_v(z_v - z_u))}{20}}$$

## 4.5.2 Asymmetry of Tonal and Noise Masking

The masking of tones by noise and noise by tones was found by Hellman to be different [23]. Figure 4.4 (a) and Figure 4.5 (b) illustrates noise masking tones and tones masking noise. The ability of noise signals to mask tonal components was found to be much greater than the ability of tonal signals to mask noiselike signals [23]. Johnston proposed that the masking threshold for the masking of tones by noise and vice versa be offset by the following values to account for the difference in the ability of tones and noise as maskers. For tones masking noise, a fixed offset is used.

$$O[i] = 5.5 \qquad (4.1)$$

For noise masking tones:

$$O[i] = (14 - i)\,\mathrm{dB} \qquad (4.2)$$

where $O[i]$ is the offset in dB that has to be applied to the masking threshold at Bark band $i$.

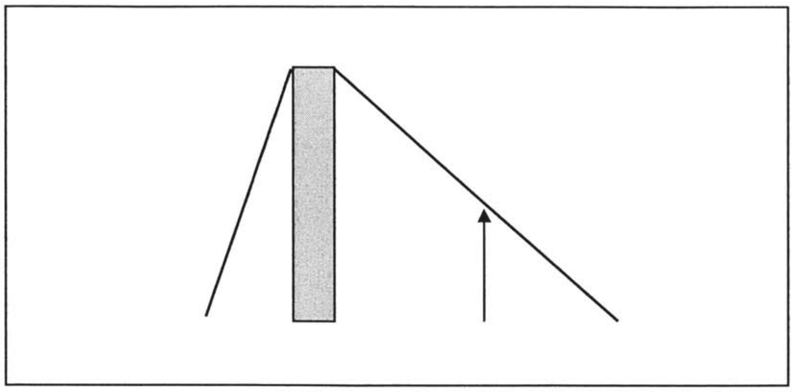

**Figure 4.5 (a): Narrow-band noise masking tone**

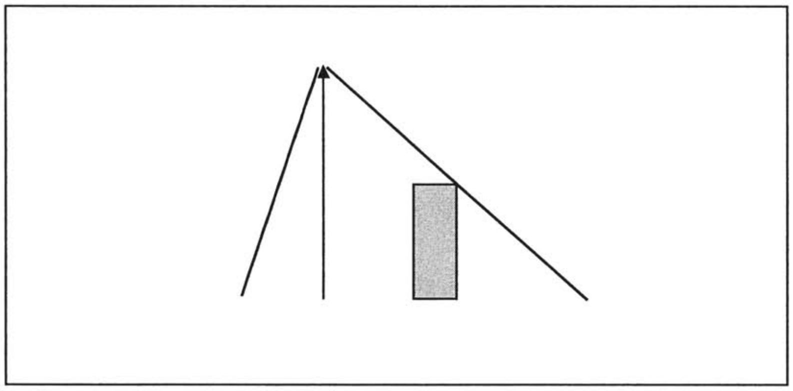

**Figure 4.5 (b): Tone masking noise**

The system distinguishes between these two kinds of masking by performing a spectral flatness measure (SFM) as done by Johnston [6, 24]. The spectral flatness measure, as its name implies is a measure of how flat the spectrum of a particular frame is. It is obtained by taking the ratio of the geometric mean[2] to the arithmetic mean. Figure 4.6 (a) and Figure 4.6 (b) illustrate how this ratio is able to distinguish between tonal components and more noise-like, spectrally flat components. The signals in both

[2] To avoid zero samples from causing the geometric mean to be zero, zero samples have to be checked

diagrams have the same arithmetic mean of 1. However, the geometric mean of the signal in Figure 4.6 (a) is 1 whereas the signal in Figure 4.6 (b) has a geometric mean of 0.866.

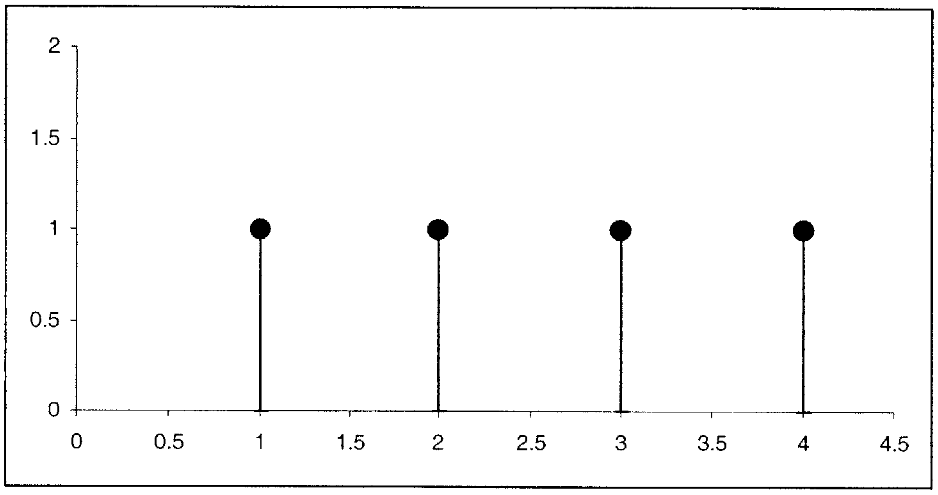

**Figure 4.6 (a): Illustration of Arithmetic Mean vs Gometric Mean**
Arithmetic mean = 1, geometric mean = 1

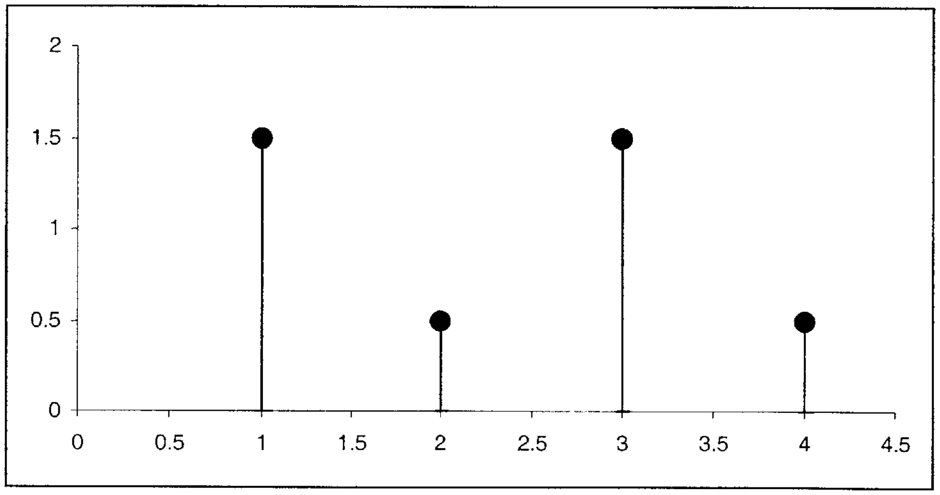

**Figure 4.6 (b): Illustration of Arithmetic Mean vs Gometric Mean**
Arithmetic mean = 1, geometric mean = 0.866

Thus, for a particular frame, we are able to distinguish how noiselike or tonelike the spectral components are due to the fact that noise spectrums are flat and tonal spectrums have peaks. This gives us a very good indication of the tonelike or noiselike nature of the maskers.

Johnston in [6] and [22], assumes that the nature of the maskee is opposite of that of the maskers. This is reflected by the fact that he offsets his computation of the masking threshold by a linear combination of (4.1) and (4.2) with each part's contribution controlled by the SFM. The following equations describe the computation of the offset as per Johnston:

$$\alpha_{SFM} = \min\left(\frac{SFM}{SFM_{\max}}, 1\right)$$

$$SFM_{\max} = -60\,\text{dB}$$

The offset is then given by

$$O[i] = \alpha_{SFM}(14+i) + (1-\alpha_{SFM})\ 5.5\,\text{dB}$$

Thus, Johnston doesn't take into account the nature of the maskee; he only considers the whole speech frame, i.e. the maskers. If the whole spectrum was completely noiselike, the offset would be 5.5 dB regardless of whether the maskee was tonelike.

In this system however, the nature of the maskee is also accounted for by using a chaos measure to compute a tonality index as presented by Schroeder [10]. It is a measure of predictability in the magnitude of the frequency component from frame to frame. The chaos measure uses the fact that a more tonal component is more predictable than a more noiselike component which is expected to be randomly changing. A linear prediction of what the current magnitude and phase of each spectral line is made based on 2 previous values of the magnitude and phase. The error between this predicted value and the known value is computed. For completely tonelike signals, the prediction would be accurate. For noiselike signals, the prediction will be poor. The algorithm to compute the tonal measure due to Schroeder is as follows:

$r(t,f)$ = magnitude of spectral line of frequency $f$ at time $t$.

$\Phi(t,f)$ = phase of spectral line of frequency $f$ at time $t$.

$\tilde{r}(t,f)$ = predicted magnitude of spectral line of frequency $f$ at time $t$.

$\tilde{\Phi}(t,f)$ = predicted phase of spectral line of frequency $f$ at time $t$.

The predicted values of phase and magnitude are computed as the sum of the previous value and the difference between the previous value and the one before that as follows:

$$\tilde{r}(t,f) = \tilde{r}(t-1,f) + \left(\tilde{r}(t-1,f) - \tilde{r}(t-2,f)\right)$$

$$\tilde{\Phi}(t,f) = \tilde{\Phi}(t-1,f) + \left(\tilde{\Phi}(t-1,f) - \tilde{\Phi}(t-2,f)\right)$$

The chaos measure $c(t,f)$ is then computed by finding the error between the predicted values of the magnitude and phase versus the known values. This error is normalized by the magnitudes of the predicted and actual magnitude as follows.

$$c(t,f) = \frac{D_E\left\{ \left[\tilde{r}(t,f)\ ,\ \tilde{\Phi}(t,f)\right],\ \left[r(t,f)\ ,\ \Phi(t,f)\right] \right\}}{r(t,f) + \left|\ \tilde{r}(t,f)\ \right|}$$

where $D_E$ is defined as the Euclidian distance between the two points given as

$$D_E\{(x,y),(a,b)\} = \sqrt{(x-a)^2 + (y-b)^2}\ .$$

The computation of the offset is then done with a combination of the SFM as well as the tonality index in bark band $i$, $T[i]$ as follows:

$$\text{offset, } O[i] = \left\| T[i] - \sigma \right| - 1\right| \times \left[\sigma(14.5 + i) + (1 - \sigma)5.5\right] \text{ dB}$$

where $\sigma$ denotes the spectral flatness measure. This ensures that the value of the offset is also based upon the tonality of the maskee. If the maskee and the maskers are both completely noiselike, then $T[i] = 0$ (denoting completely non-tonal) and $\sigma = 1$ resulting in zero offset. For completely tonal maskers and maskees, $T[i] = 1$ and $\sigma = 0$, also resulting in zero offset. For noise masking tones and tones masking noise, $\left\| T[i] - \sigma \right| - 1\right| = 1$ and $\sigma$ will distinguish if it is a tone masking noise case or vice-versa.

## 4.6 Gain Function Computation and Frequency Domain Filtering

The gain function is computed based on the results in Chapter 3. This is done using the noise PSD estimate as described in section 4.3, the speech PSD estimate as in section 4.4, the masking threshold estimate as in the last section and a speech content measure based on the SNR. Section 3.5 describes an SNR based speech content measure. The SNR is estimated using the following equation:

$$SNR(f) = \sqrt{\frac{E_S(f)}{\tilde{N}^2(f)}}$$

where $\tilde{N}(f)$ is the noise estimate and $E_S(f)$ is the input signal energy.

# Chapter 5: Results

## 5.1 Informal Tests and Author Evaluation

Informal testing of the algorithm on several different input files shows an improvement over the TIA's Enhanced Variable Rate Codec (EVRC) noise suppression algorithm [1] as well as the Gustafsson algorithm. The tests were conducted on 8 input files. 6 of the files were generated by adding car noise to clean speech samples. The SNR's were divided into low, medium and high. 15 dB of SNR was used for the high SNR data , 8 dB of SNR for the medium SNR and 4 dB for the low SNR. The two additional files are from actual recordings of speech in road noise. The target noise suppression is 20 dB. The evaluation was done by the author and another expert listener. The following is based on the comments from the listeners.

The output from the hybrid algorithm contains no audible musical noise. The speech was of high quality except in very low SNR segments (below 3 dB), where there is some noticable attenuation of the speech. This is due to the fact that the Gustafsson part of the hybrid function gets weighted highly in the low SNR regions. Since the Gustafsson algorithm attenuates the speech significantly for low SNR's [3], the hybrid function does the same as well. The EVRC algorithm attenuates low SNR speech portions (below 3 dB) by more than 19 dB resulting in choppy speech and the loss of begginings and ends of words. As for the Gustafsson algorithm, speech attenuation was noticable even in high SNR speech. The EVRC algorithm also has a "choruslike" effect on some low SNR speech. This effect is due to noise being let through in bands that contain speech for which the speech does not completely mask the noise. As a result the noise that is let through is correlated with the frequency components of the speech and sounds “choruslike”. This effect is not present in the hybrid function or the Gustafsson algorithm.

The system in this thesis causes some reverbaration in the speech. This is due to the large time window used in this system. The requirements on frequency resolution for the masking threshold estimation and the low sampling rate of 8 kHz used for telephony, force the use of such a large window. The EVRC noise suppressor uses a window of 13.5 ms compared to the 40 ms used in the system in this thesis. Therefore the EVRC algorithm performs better in terms of reverbration. The Gustafsson algorithm has the same reverbration effect since the implementation of the Gustafsson algorithm in this research project uses the same windowing scheme.

## 5.2 Quantitative Performance Measures

### 5.2.1 Objective Measures

In Chapter 3, the notions of Masking-to-Noise-Ratio (MNR) and Masking-to-Speech-Ratio (MSR) as measures of performance for algorithms using the masking model were introduced. The resultant gain from the hybrid algorithm as well as the Gustafsson algorithm are plotted versus the MNR in Figure 5.1, versus the MSR in Figure 5.2 and versus the SNR in Figure 5.3. All data were obtained by several runs of the algorithms on various inputs with a target noise suppression of 20 dB. It is important to note that a 20 dB speech enhancement (i.e. amplification of speech) and a 20 dB noise suppression are equivalent up to a normalizing factor.

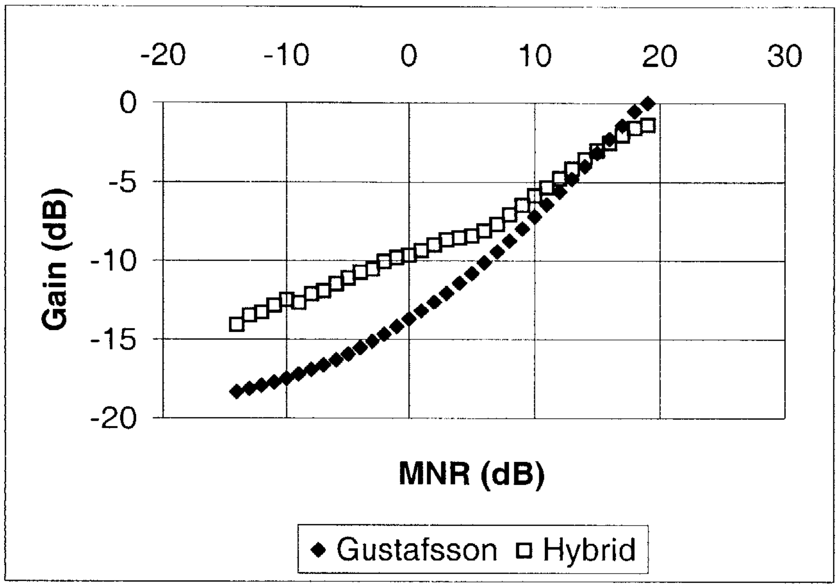

**Figure 5.1: Gain versus MNR for Gustafsson and Hybrid Algorithms**

The data in Figure 5.1 show that both the algorithms behave as expected for different MNR's. It is expected that for high MNRs, the noise level is low and the signal level is high (which causes a high degree of masking). Thus, the gain is expected to be close to 0 dB. Conversely, when the MNR is low, there is low masking and/or high noise. High noise will have to be attenuated more and low masking (low signal power) will also require higher attenuation. Thus, low MNR requires higher attenuation.

The Gustafsson algorithm gives the minimum amount of attenuation needed to make the noise perceptually equivalent to the desired noise level. From the graph above, the hybrid function attenuates the signal less than the required amount of attenuation as given by the Gustafsson gain function for most values of the SNR. The maximum difference between the required attenuation and the actual attenuation provided by the hybrid algorithm is about 4 dB. This means that the use of the hybrid function will result in the noise sounding up to approximately 4 dB louder than desired. However, this has not been noticable in the test data. This is most probably caused by underestimation of the masking threshold which results in more noise being masked. The fact that more noise is let through by the hybrid function represents a tradeoff between keeping the noise perceptually equivalent to the desired noise and the speech perceptually equivalent to the desired speech.

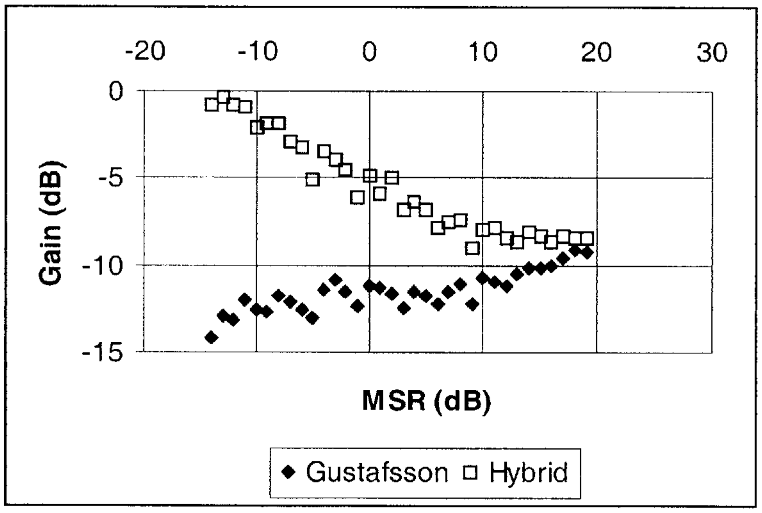

**Figure 5.2 Gain versus MSR for Gustafsson and Hybrid Functions**

From figure 5.2 the hybrid function behaves as desired. For high MSR regions, the speech power is much lower than the masking. This means that the speech power at that point is much lower than the speech power in adjacent points (which contribute to the masking threshold). As a result the SNR in those points is low and would require higher attenuation. Conversely, when the MSR is low, the masking thershold is much lower than the speech power and naturally, the gain function should let the signal component pass with little or no attenuation. The Gustafsson function does not exhibit this behavior. This is because the Gustafsson algorithm does not take the MSR into account. The hybridization with the $H_S$ function is what allows this behavior with the tradeoff described in the previous paragraph and illustrated in Figure 5.1. In the worst case, the Gustafsson algorithm differs from the hybrid function by 14 dB compared to 5 dB in the previous case.

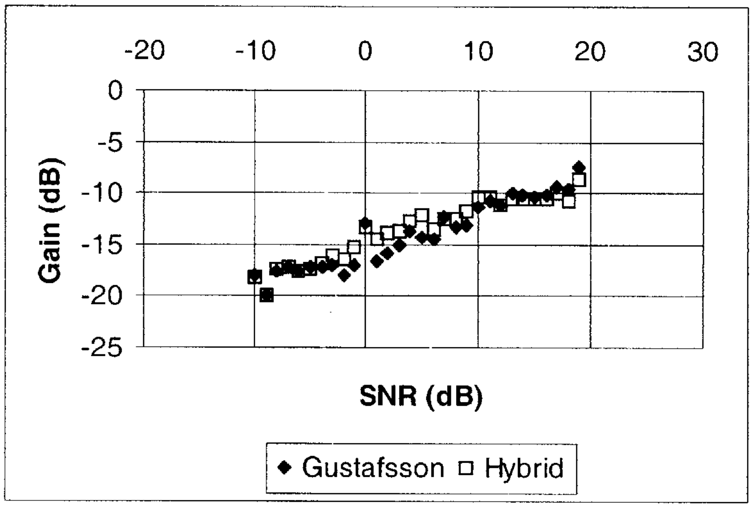

**Figure 5.3: Gain versus SNR for Gustafsson and Hybrid Algorithms**

As expected, both the Gustafsson function and the hybrid function have higher attenuation levels for lower SNRs and vice versa which is the expected behaviour of any noise suppression system. Both the functions have similar characteristics in this plot. The range of attenuations that are spanned by both the algorithms is approximately 13 dB. Thus, the amount of distortion can be expected to be significantly lower than other algorithms that span the entire range between –20 dB and 0 dB [1, 8]. The psychoacoustic model allows this because the amount of attenuation necessary to perceptually achieve a desired level of noise is less than the amount of attenuation required to achieve it mathematically. Also, for the enhancement of speech, a smaller gain is needed to make the speech sound perceptually intact. Figure 5.3 also shows a shortcoming of the Gustafsson gain function. For high levels of noise suppression like the 20 dB that is used to generate the data in Figures 5. 3, the Gustafsson algorithm attenuates speech components significantly. In Figure 5.3, the maximal gain that is applied by the Gustafsson rule is approximately –7 dB. Thus, the speech signal is always attenuated. As for the hybrid rule, the maximum gain that is applied is around –7 dB as well. However, the hybrid function takes into account the maximum amount of attenuation that can be applied while keeping the speech perceptually intact. This is done via the $H_S$ function in the hybrid rule. Thus, the gain from the hybrid rule will result in the output perceptually equivalent to the desired signal.

### 5.2.2 Perceptual Binary Forced Choice Test

A binary forced choice test was performed for the data from the hybrid function and the EVRC Noise Suppressor with non-expert listeners. The same test was not conducted against the Gustafsson algorithm due to the fact that the data processed with the Gustafsson algorithm resulted in the speech being significantly lower in power than the speech signals in both the EVRC noise suppressor and the hyrid system introduced in this thesis. As a result the overall SNR in the output data was approximately 6 dB lower than that from the EVRC and the hybrid function. It was thus decided that a fair comparison could not be done. This is due to the fact that the Gustafsson algorithm has the goal of keeping the noise consistent without explicitly considering the speech. The following describes the procedure of the binary forced choice test. Eight noisy speech segments were used. They were generated by adding car noise to clean speech signals. The test data comprised the following.

- 2 Female speaker segments with high SNR (12 dB)
- 2 Male speaker segments with high SNR (12 dB)
- 2 Female speaker segments with low SNR (4 dB)
- 2 Male speakers segments with low SNR (4 dB)

Each segment contained two sentences approximately 6 seconds in length. They were processed using both the EVRC and the system in this thesis with a noise attenuation factor of 20 dB.

10 subjects were asked to listen to an input segment. Then, they were asked to listen to the input segment processed by the two systems being tested. The order in which the outputs from the two systems were presented was random. The subjects were asked to choose which of the two output segments was preferable. The criteria for the decision is that the speech has to sound as close as possible to the speech in the input data and the noise has to sound like an attenuated version of the noise in the input data. When the subjects did not have a preference, they were asked to pick one randomly.

The following table lists the results obtained from the binary forced choice test.

| | Female | | Male | |
|---|---|---|---|---|
| | High SNR (12 dB) | Low SNR (4 dB) | High SNR (12 dB) | Low SNR (4 dB) |
| EVRC | 3 | 2 | 3 | 1 |
| Hybrid | 17 | 18 | 17 | 19 |

**Table 5.1: Results of Binary Forced Choice Test**

As is evident from the results, the algorithm introduced in this thesis is prefered over that of the EVRC noise suppressor, especially for the low SNR data. In 88.75 % of the cases, the hybrid function was prefered. Discussions with the subjects after the tests were conducted revealed that for the low SNR cases, the speech in the data processed by the hybrid system was strongly preferred especially in the edges of sentences. This is due to the $H_S$ function's contribution to the hybrid function which ensures that the speech error is masked. They also reported that the noise in the data processed by the hybrid function sounded closer in terms of frequency response to the original. The EVRC noise suppressor was found to have a highpass response to the noise. Three of the subjects had did not have a preference for one of the high SNR files with the male speaker and were asked to chose randomly. For the low SNR cases, there was no such problem.

### 5.2.3 Performance with Vocoders

The EVRC noise suppressor, the Gustafsson algorithm and the hybrid system were all tested with the TIA standard EVRC vocoder [1]. The output of the noise suppressors was coded and decoded by the vocoder and the performance was subjectively evaluated.

The data comprised speech segments from male and female speakers corrupted by car noise. Two levels of SNR, 15 dB and 4 dB were used. 6 input files were tested: One with just a female voice, one with a male voice, and one with both male and female voices one after the other. These files were tested with both SNR levels. The segments were approximately 30 seconds in length. For each data file the following was presented to the listener:

- The input file was.
- The file processed by the the EVRC noise suppressor.
- The file processed by the noise suppressor and the vocoder was played
- The last two steps were repeated for the Gustafsson algorithm and the hybrid system.

The tests were conducted with 5 expert listeners who gave their preferred system and comments on why they preferred that system. The results described here are the opinion of the listeners of the data processed by the vocoders (after the noise suppressors).

All 5 of the listeners preferred the hybrid system to the other two. All of them mentioned that the speech quality using the hybrid system was preferable and that the low SNR portions of the speech were less attenuated than that of the other two systems. The frequency response of the hybrid system was also preferred by two of the listeners. They noted that the results of the hybrid system did not sound as "tinny" as the EVRC noise suppressor or as "muffled" as that of the Gustafsson algorithm.

However, all of the listners were of the opinion that the voice coder caused more distortion in the noise of both the Gustafsson algorithm and the hybrid function compared to the EVRC noise suppressor. This can be attributed to the fact that the Gustafsson and hybrid algorithms both keep the distortions in speech and noise just below the masking threshold, i.e. the distortions are just masked. Since the voice coder codes noise poorly, the coding error in the noise is high. When added to the noise suppression error which is just below the masking threshold, the combined error ends up being higher than the masking threshold and thus, is no longer masked. The listeners also noted that both the Gustafsson algorithm and the hybrid function had some reverberation in the speech.

The results of this test show that the use of the hybrid function is feasible in terms of audio quality (as opposed to computation load) as a front-end to voice coders. In order to improve the tandem performance further, a combined voice coder-noise suppressor could be designed as  is described in Chapter 6.

## 5.3 Processing Overhead

### 5.3.1 Windowing Delay

The windowing delay for this algorithm is high. This is due to the required frequency resolution for the masking threshold estimation and the low sampling rate of 8 kHz that is used for telephony. With a 75% overlap and 320 sample window, the output samples are delayed by 240 samples which corresponds to a delay of 30 ms. Figure 4.3 shows that the output sampes are the samples numbered 240 to 319. Since the 80 newest samples are introduced at sample number 480 to 559, the output is delayed by 240 samples. In order to make this system suitable for real time operation, this windowing delay has to be reduced. A higher sampling frequency will reduce the delay. Cleverer windowing schemes could also possibly be used. A masking threshold estimation method that requires less frequency resolution would also decrease the required window length and reduce framing delay.

### 5.3.2 Computational Overhead

The most computation intensive part of this algorithm is the estimation of the masking threshold. Computing the masking threshold for 512 frequency points requires a nested loop of 512 operations in both levels. Equation 4.1 requires a divide operation as well as a power operation. Also, 2 add operations and one multiply operation are required per loop cycle. Thus, this algorithm requires a powerful processor in order to run near real time.

# Chapter 6: Discussion

## 6.1 Conclusion

The hybrid function introduced in this thesis was found to perform well both in terms of speech quality as well as noise quality with no audible musical noise. In the informal tests, it was noted that the speech quality for the hybrid function is better than that of the Gustafsson algorithm due to distrortion caused in the speech by the Gustafsson algorithm. The Gustafsson algorithm caused a high of attenuation in low SNR speech as well as attenuations of up to 10 dB for high SNR cases (above 12 dB). The hybrid algorithm was overwhelmingly preferred both in terms of speech distortions as well as noise distortions compared to the EVRC noise suppressor [1] in subjective tests. As a front end to vocoders, this algorithm is also preferred. The distortion masking principle applied selectively to speech errors and noise distortions is the key to the performance of this algorithm.

A small amount of reverberation in the speech is an artifact of the system in this thesis in its present form. This is caused by the large input window that is used (40 ms) which is necessary in order to achieve the frequency resolution required to estimate the masking threshold. Overall, the system introduced in this thesis provides a viable alternative to the various speech enhancement algorithms that exist today.

## 6.2 Future Work

In the course of this research several ideas and opportunities have arisen for further exploration. Virags's use of the masking property in a subtractive-type algorithm [2] opens the door to the use of the masking property in other existing algorithms; for instance, a perceptual Wiener filter which minimizes the perceptual error. Another idea is the use of speech production models as well as auditory models in a speech enhancement algorithm. One possibility is a pitch filtering algorithm that uses the masking model to ensure noise signals are masked. In the course of this research, some preliminary work on this idea was performed; it is feasible if adequate pitch detection and filtering can be performed. Such a system that uses both speech production and hearing models can be expected to perform better than systems that use just the auditory models in the sense that it can be tailored to match characteristics of speech signals as well as the hearing process.

The performance of both the Gustafsson algorithm and the hybrid algorithm in tandem with voice coders was found to have more perceivable noise distortions than expected. Research into the tandem operation of noise suppression systems with vocoders is an exciting possibility. One possibility is to integrate a perceptual speech enhancement and perceptual voice coding system whereby the bit allocation and noise suppression would be performed such that the combination of the coding error and the noise suppression error is masked.

Overall, the field of speech processing using psychoacoustic models is very exciting and offers numerous opportunity for interesting work. Although the human hearing process is far from being fully understood, the last 50 years of research in psychoacoustics has led to an enormous wealth of knowledge that can be exploited further in digital speech processing.

# References

[1] Interim Standard TIA/EIA/IS-127-1. Enhanced Variable Rate Codec, Speech Service Option 3 for Wideband Spread Spectrum Systems – Addendum 1 August 1998

[2] N. Virag. Single Channel Speech Enhancement Based on Masking Properties of the Human Auditory System. *IEEE Transactions on Speech and Audio Processing,* vol. 7, no. 2, March 1999.

[3] S. Gustafsson, P. Jax, P. Vary. A Novel Psychoacoustically Motivated Audio Enhancement Algorithm Preserving Background Noise Characteristics. *Proc. ICASSP,* 1998, Seattle.

[4] E. Zwicker, H. Fastl. *Psychoacoustics: Facts and Models,* Springer, New York, 1990.

[5] D. Sen, D.H. Irving, W.H. Holmes. Use of an Auditory Model to Improve Speech Coders, *Proc. ICASSP*, vol. 2 pp. 411-414, 1993.

[6] J. Johnston. Transform Coding of Audio Signals Using Perceptual Noise Criteria. *IEEE Journal on Selected Areas in Communications*, vol. 6. no. 2. February 1988.

[7] Draft Standard ISO 11172-3 MPEG Audio, London, November 1992.

[8] S. Boll. Suppression of Acoustic Noise in Speech Using Spectral Subtraction". *IEEE Transactions on Speech and Audio Processing*, vol. 27, no.2 pp. 113-120, 1979.

[9] S.V. Vaseghi, *Advanced Signal Processing and Digital Noise Reduction,* John Wiley and Teubner, 1996.

[10] M.R. Schroeder. *Computer Speech: Recognition, Compression, Synthesis.* Springer Series in Information Sciences. Springer-Verlag, Berlin Heiderberg 1999.

[11] E.B. Goldstein, *Sensation and Perception.* Brooks/Cole Publishing Company, Pacific Grove 1996.

[12] E. Terhardt. Calculating Virtual Pitch, *Hearing Research*, 1979, pp. 155 – 199.

[13] J.S. Lim, A.V. Oppenheim, L. D. Braida, Evaluation of an Adaptive Comb Filtering Method of Enhancing Speech Degraded by White Noise Addition, *IEEE Transactions on Acoustics, Speech and Signal Processing,* vol. ASSP-26, no. 4, August 1978.

[14] N. Jayant, J. Johnston, R, Safranek. Signal Compression Based on Models of Human Perception. *Proc. of the IEEE,* vol. 81, no. 10, October 1993.

[15] A. Czyzewski, R. Krolikowski, Noise Reduction in Audio Signals Based on the Perceptual Coding Approach. *Proc. 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New York, Oct 1999.

[16] R.H. Frazier, *An Adaptive Filtering Approach Toward Speech Enhancement*, S.M. thesis, MIT, Cambridge 1970.

[18] R.H. Frazier et. al. Enhancement of Speech by Adaptive Filtering, Proc. *IEEE ICASSP*, April 1976, pp. 251-253.

## References (Cont.)

[19] Y. Ephraim and D. Malah. Speech Enhancement Using a Optimal Non-linear Spectral Amplitude Estimation. *Proc. IEEE ICASSP*, 1983, Boston.

[20] Y. Ephraim and D. Malah. Speech Enhancement Using a Minimum Mean-square Error Log-spectral Amplitude Estimation. *Proc. IEEE Trans. Acoustics Speech and Signal Processing*, vol. ASSP-32, no. 6, pp. 1109-1121, 1984.

[21] O. Cappe. Elimination of the Musical Noise Phenomenon with the Ephraim and Malah Noise Suppressor, *IEEE Transactions on Speech and Audio Processing*, vol. 2, no. 2, April 1994.

[22] J. D. Johnston and K. Brandenburg. Wideband Coding — Perceptual Considerations for Speech and Music. *Advances in Speech and Signal Processing*. S. Furui and M. M. Sondhi Eds. New York. 1992

[23] R. P. Hellman, Asymmetry of Masking Between Noise and Tone. *Perception and Psychophysics*, vol. 11, pp. 241-246. 1972

[24] L.R. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall. New Jersey. 1978.

[25] M. Berouti, R. Schwartz and J. Makhoul. Enhancement of Speech Corrupted by Acoustic Noise. *Proc. IEEE ICASSP*, Washington DC. Apr. 1979. Pp. 208 – 211.

[26] M. A. Tuffy and D. I. Laurenson. Estimating Clean Speech Thresholds for Perceptual Based Speech Enhancement. *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. New York, Oct 17-20, 1999.

[27] D. F. Mix. *Random Signal Processing*. Prentice Hall. New Jersey. 1995

# Appendix A: C-Code For Hybrid System

```
/* hybrid.c */


/***************************************************************/
/*                                                        */
/*                                                        */
/*          COPYRIGHT 1999 Qualcomm Inc.                  */
/*                                                        */
/*                                                        */
/***************************************************************/


/***************************************************************
*
* Hybrid Function Noise Suppression*
* Input:  The input to the function is a float pointer to the
*         array of data to be noise suppressed.
*
* Output: There is no return value.  The input array is replaced
*         with the noise suppressed values.
*
*
* Written by:                     Siddhartan Govindasamy
* Date:                           December 5, 1999
*
*
***************************************************************/


/* Includes */


#include    <math.h>
```

```
#include    <stdio.h>

#include    <stdlib.h>

#include    "siddhartan.h"

#include    "window.h"

#include    "bark512.h"

#include    "nstables.h"




//#define GRAPH

/* Defines */


/* testing */


//#define NO_FILTER

//#define NO_FFT

//#define TEST_WINDOW

//#define WIENER

#define GUSTAF



/* defines for algorithm modifications */

//#define BND_CLS


//#define USE_ENRG


//#define TRAP_WIN


#define SPEC_SUB

//#define CUT_LO_SNR

#ifndef WIENER

#define DO_MASK
```

```
#endif
#define TONAL_MEASURE
//#define OLD_NOISE_CHANNELS
//#define DECIMATE_ADJACENT_TONES
//#define USE_ASSEMBLY
//#define ADPT_SS
#define TBL_BARK
//#define MASK_TEST
//#define CNT_BANDS
//#define MPEG_MASK


/* General defines */

#define                 TRUE            1
#define                 FALSE           0
#define                 PI  (4.0*atan(1.0))

#define                 MAG             0
#define                 PHZ             1


#define                 FLAT_GAIN       1.0  // input gain



/* Pre-processing */

#define                 PRE_EMP_FAC     (-0.0)
#define                 DE_EMP_FAC      0.0
```

```
/* Channels */
#ifndef LARGE_CHANS

#define                 NUM_CHAN          64
#define                 LO_CHAN           0
#define                 MID_CHAN          21
#define                 HI_CHAN           63

#else

#define                 NUM_CHAN          16
#define                 LO_CHAN           0
#define                 MID_CHAN          5
#define                 HI_CHAN           15



#endif



#define                 NUM_NOISE_CHAN    16
#define                 LO_NOISE_CHAN     0
#define                 HI_NOISE_CHAN     15
#define                 MID_NOISE_CHAN    5



/* Noise  And SNR Estimate */

#define                 UPDATE_THLD       35
#define                 METRIC_THLD       45
#define                 INDEX_THLD        12
#define                 SETBACK_THLD      12
```

```
#define                SNR_THLD          ((int)(2.25/0.375))   /*
equals 6 */



#ifndef LARGE_CHANS

#define                INDEX_CNT_THLD    20

#else

#define                INDEX_CNT_THLD    5

#endif



#define                UPDATE_CNT_THLD   ((int)50)//*1.6)

#define                HYSTER_CNT_THLD   ((int)3)    //* 1.6)


#define                NORM_ENRG         (1.0) /* use (32768.0 *
32768.0) for */

                                                             /*
fractional */

#define                NOISE_FLOOR       (1.0 / NORM_ENRG)

#define                MIN_CHAN_ENRG     (0.0625 / NORM_ENRG)

#define                INE               (16.0 / NORM_ENRG)




#define                HIGH_TCE_DB       (200.)      /* 50 -
10log10(NORM_ENRG) */

#define                LOW_TCE_DB        (120.)      /* 30 -
10log10(NORM_ENRG) */

#define                TCE_RANGE         (HIGH_TCE_DB - LOW_TCE_DB)

#define                HIGH_ALPHA        0.99

#define                LOW_ALPHA         0.50

#define                ALPHA_RANGE       (HIGH_ALPHA - LOW_ALPHA)

#define                SAMPLE_RATE       8000
```

```
#ifndef LARGE_CHANS

#define                 DEV_THLD          (28.0 * 4.0)

#else

#define                 DEV_THLD           28.0

#endif




/* smoothing factors for noise and energy */


#define                 CNE_SM_FAC        0.1
#define                 CEE_SM_FAC        0.7212


/* Gain slopes for SNR based gain coeffs */


#define                 MIN_GAIN          (-20.0)

#define                 GAIN_SLOPE        0.50





/* Spectral subtraction factor */


#define                 SPEC_SUB_FAC       (2.0)


/* Attenuation factor */


#define                 TONAL_THLD         (0.2)

#define                 SPEECH_THLD        (32)
```

```
/* Adjustment (in dB) to the masking curve */

#define                 MASK_ADJ        (0.0)


/* Macros */

/* general */

#define                 min(a,b)          ((a)<(b)?(a):(b))
#define                 max(a,b)          ((a)>(b)?(a):(b))
#define                 square(a)         ((a)*(a))




float tmp = 0.0;

int tmp_int = 1;


/*  Function Prototypes */

/* Graphing functions */

#ifdef GRAPH

void master_load(int graph, float* buf);
void master_graph();
void master_suspend();
void master_var_display(int var, float value, char* label);
void master_setup(int scroll_size, int bar_size, int scatter_size);
```

```
#endif




/* variables for display functions */


#ifdef GRAPH

extern int ch_snr_disp_draw, ch_gain_disp_draw, FTMags_draw,
Threshold_draw, Threshold2_draw, SNR_draw, FiltDat_draw, FiltDat2_draw,
UnFiltDat_draw, Noise_draw;

extern int ch_snr_disp_num, ch_gain_disp_num, FTMags_num, Threshold_num,
Threshold2_num, SNR_num, FiltDat_num, FiltDat2_num, UnFiltDat_num,
Noise_num;

extern int spec_sub_mags_draw, Threshold3_draw, FTMags2_draw,
gain_x_draw, gain_n_draw;

extern int spec_sub_mags_num, Threshold3_num, FTMags2_num, gain_n_num,
gain_x_num;


#endif





/* Window function */

void window( float window[], int len, int kind);


/* mask threshold calculation */


void get_pmask(float *, float *);


void tonal_measure(float[], float[]);

float get_bark(int i);

void get_MPEG_mask(float *L, float *th, float tonality[]);
```

```
/* The noise supression function */

void noise_suprs (float *farray_ptr, int stop_display, short
UpdateBuf[], short VmBuf[], float TmpBuf[])


{





/* Housekeeping */

    static int    first = TRUE;

    static unsigned long frame_cnt;

    int           i, j, j1, j2;

    static int loop = 0;



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

/* Vars and vectors for windowing and preprocessing */

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


/* vars for pre-emphasis */

static float      pre_emp_mem = 0.0, de_emp_mem = 0.0;



/* Vectors for windowing */

static float      window [BUF_LEN], window_overlap [OVL_LEN],
overlap_add[2*BUF_LEN];




/* Vector for channel gains */

static float      ch_gain [FFT_LEN/2];


/* Vars for Processing and gain computation*/
```

```
float data_buffer [FFT_LEN], data_bufferTmp[FFT_LEN],
Pthresh[FFT_LEN/2], FTMagsTmp[FFT_LEN/2];

float   spec_sub_mags[FFT_LEN/2];

static  float spec_sub_fac;

float spec_sub_enrg;

float output_enrg;

static float spec_sub_flat[FFT_LEN], spec_sub_flat_mags[FFT_LEN/2],
pitch_inp[128];


int pitch_number;


float   offset[FFT_LEN/2], tonality[FFT_LEN/2];

float   gain;




/* channel energy and noise vars */

static float      ch_enrg [NUM_CHAN], ch_noise [NUM_CHAN],
ch_noise_tmp[NUM_NOISE_CHAN];

static float ch_noise_tmp_long[FFT_LEN/2], ch_enrg_tmp[NUM_NOISE_CHAN];

float   enrg, snr;

float tne, tce;

int         ch_snr [NUM_CHAN];

static float ch_enrg_long_db [NUM_CHAN];

float phz = 0.0, mag = 0.0, noise_tmp[FFT_LEN];

float noise_r, noise_i, noise_phz;

float       ch_enrg_dev;            /* for forced update... */

float       ch_enrg_db [NUM_CHAN];

float       alpha;


float   gain_alpha, gain_x, gain_n, pre_gain;


/* This is vector stores the snrs as if there were the same number of
```

```
   channels as the original to use the same voice metric table */

int ch_snr_tmp [NUM_CHAN], ch_snr_cnt = 0;




/* counters for update decisions */


static int  update_cnt = 0;

static int  hyster_cnt;       /* forced update statics... */

static int  last_update_cnt;

int         vm_sum;

int         update_flag, modify_flag, index_cnt;


int noise_flag;



/* Variables for Display */

/* These are used to display the graphs on screen */



float             ch_snr_disp[FFT_LEN/2];

float           ch_gain_disp[FFT_LEN/2];

float           FTMags[FFT_LEN/2];

float           PCM[FFT_LEN];


FILE *F_tmp1, *F_tmp2;



/* Variables used for computation of bin based SNR */
```

```
static int bin_snr[FFT_LEN/2], bin_snr_tmp[FFT_LEN/2];

static float bin_enrg[FFT_LEN/2];

static float bin_noise[FFT_LEN/2];

float t, t_i, t_f;


float gainXDisp[FFT_LEN/2], gainNDisp[FFT_LEN/2];

float sc1, sc2, sc3, sc4;



/* Functions */


void         r_fft (float *, int);

void         init_window (float *, int, float);

void         make_window( float window[], int len, int kind);

void get_offset(float offset[], int len, float S[], float tonality[]);



/* Init the window function, channel gains one time */

loop ++;


if (first == TRUE) {


#ifndef TRAP_WIN

    make_window(window, BUF_LEN, HAMMING);

#else


    init_window(window, BUF_LEN, (((float)OVL_LEN)/((float)BUF_LEN)));


#endif

    /* initialize first 4 gain values to 1.0 */


    ch_gain [0] = (ch_gain [1] =  (ch_gain[2] = (ch_gain[3] = 1.0)));
    /* initialize overlap add buffer */

    for(i = 0; i < 2*BUF_LEN; i++)
        overlap_add[i] = 0.0;
/*
#ifdef OLD_NOISE_CHANNELS
    for(i = 0; i < NUM_NOISE_CHAN; i++){
        j1 = noise_channels[i][0];
        j2 = noise_channels[i][1];

        for(j = j1; j <= j2; j++)
            barkval[j] = i;
    }
#else
    for(i = 0; i < NUM_CHAN; i++){
        j1 = ch_tbl[i][0];
        j2 = ch_tbl[i][1];

        for(j = j1; j <= j2; j++)
            barkval[j] = i;
    }

#endif
*/
    spec_sub_fac = SPEC_SUB_FAC;

}


/* Increment frame counter */
frame_cnt++;
```

```
/* copy the overlapping samples from old frame into beginning of new
frame */

#ifndef TRAP_WIN

  for (i = 0; i < BUF_LEN*OVL_RATIO_X_4/4; i++)
    data_buffer [i] = window_overlap [i];

/* preemphasize data */

  data_buffer [BUF_LEN*OVL_RATIO_X_4/4] = FLAT_GAIN**farray_ptr +
PRE_EMP_FAC * pre_emp_mem;

#else

  for (i = 0; i < OVL_LEN; i++)
    data_buffer [i] = window_overlap [i];
    data_buffer [OVL_LEN] = *farray_ptr;

#endif

 for (i = OVL_LEN + 1, j = 1; i < BUF_LEN; i++, j++)
     data_buffer [i] = FLAT_GAIN**(farray_ptr + j) + PRE_EMP_FAC *
                       FLAT_GAIN**(farray_ptr + j - 1);

  pre_emp_mem = FLAT_GAIN**(farray_ptr + j - 1);
```

```
/* zero pad by FFT_LEN - BUF_LEN zero samples */

  for (i = BUF_LEN; i < FFT_LEN; i++)
    data_buffer [i] = 0.0;



/* store the current frame data to be overlapped  in next frame*/


#ifndef TRAP_WIN

  for (i = 0; i < BUF_LEN*OVL_RATIO_X_4/4; i++, j++)
    window_overlap [i] = data_buffer [BUF_LEN - BUF_LEN*OVL_RATIO_X_4/4
+ i];

#else

  for (i = 0; i < OVL_LEN; i++)
    window_overlap [i] = data_buffer [i + (BUF_LEN - OVL_LEN)];


#endif

/* Apply window to frame prior to FFT */

  for (i = 0; i < BUF_LEN; i++)
    data_buffer [i] *= window [i];

#ifndef TEST_WINDOW
```

```
/* If we want to display the windowed PCM instead of PCM
   copy windowed input data to display vector */


for(i = 0; i < FFT_LEN; i++)
    PCM[i] = data_buffer[i];



/* Perform FFT on the data buffer */


#ifndef NO_FFT
  r_fft (data_buffer, +1);
#endif


/*  show the compute the FFT magnitudes*/


output_enrg = 0.0;
for( i = 0; i < FFT_LEN/2; i++)
    output_enrg += (sqrt(square(data_buffer[2*i]) +
square(data_buffer[2*i + 1]))/FFT_LEN);



  for(i = 0, j = 0; i < FFT_LEN; i+=2, j++)
     FTMags[j] =
(float)20*log10(sqrt((double)square(1/FLAT_GAIN*data_buffer[i])
                                +
(double)square(1/FLAT_GAIN*data_buffer[i+1])));




/* Estimate the energy in each channel */


  alpha = (first == TRUE) ? 1.0 : CEE_SM_FAC;
  for (i = LO_CHAN; i <= HI_CHAN; i++) {

    enrg = 0.0;

    j1 = ch_tbl [i][0], j2 = ch_tbl [i][1];
    for (j = j1; j <= j2; j++)
      enrg += square(data_buffer [2*j]) + square(data_buffer [2*j+1]);
    enrg /= (float) (j2 - j1 + 1);
    ch_enrg [i] = (1 - alpha) * ch_enrg [i] + alpha * enrg;
    if (ch_enrg [i] < MIN_CHAN_ENRG) ch_enrg [i] = MIN_CHAN_ENRG;

  }

#ifdef OLD_NOISE_CHANNELS

  for (i = LO_NOISE_CHAN; i <= HI_NOISE_CHAN; i++) {

    enrg = 0.0;

    j1 = noise_channels [i][0], j2 = noise_channels [i][1];
    for (j = j1; j <= j2; j++)
      enrg += square(data_buffer [2*j]) + square(data_buffer [2*j+1]);
    enrg /= (float) (j2 - j1 + 1);
    ch_enrg_tmp [i] = (1 - alpha) * ch_enrg_tmp [i] + alpha * enrg;
    if (ch_enrg_tmp [i] < MIN_CHAN_ENRG) ch_enrg_tmp [i] =
MIN_CHAN_ENRG;

  }

#endif

/* Estimate the energy in each bin */
```

```
  for(i = 0; i < FFT_LEN/2; i++){

    enrg = square(data_buffer [2*i]) + square(data_buffer [2*i+1]);

    bin_enrg [i] = (1 - alpha) * bin_enrg [i] + alpha * enrg;

    if (bin_enrg [i] < MIN_CHAN_ENRG) bin_enrg [i] = MIN_CHAN_ENRG;

  }

/* Initialize channel noise estimate to channel energy for first four
frames */

#ifndef OLD_NOISE_CHANNELS

  if (frame_cnt <= 4)

    for (i = LO_CHAN; i <= HI_CHAN; i++)

      ch_noise [i] = max(ch_enrg [i], INE);

#else

  if (frame_cnt <= 16)

    for (i = LO_CHAN; i <= HI_CHAN; i++)

      ch_noise [i] = max(ch_enrg [i], INE);



#endif


/* Compute the channel SNR indices */


  for (i = LO_CHAN; i <= HI_CHAN; i++) {

    snr = 10.0 * log10 ((double) (ch_enrg [i] / ch_noise [i]));

    if (snr < 0.0) snr = 0.0;
    ch_snr [i] = (snr + 0.1875) / 0.375;

  }

/* store the noise into appropriate vector for bin level SNR computation
*/

  for(i = LO_CHAN; i <= HI_CHAN; i++){

        j1 = ch_tbl [i][0], j2 = ch_tbl [i][1];

        for (j = j1; j <= j2; j++){

           bin_noise[j] = ch_noise[i];
        }
  }



/* compute the snr in each bin */


  for(i = 0; i < FFT_LEN/2; i++){

    snr = 10.0 * log10 ((double) (bin_enrg [i] / bin_noise [i]));


    if (snr < 0.0) snr = 0.0;

    bin_snr [i] = (snr + 0.1875) / 0.375;
  }
```

```
/* Compute temporary snr indices for vm sum computation based on old
   channels */
#ifndef LARGE_CHANS
  for (i = LO_CHAN; i <= 15; i++) {


    /* compute average snr over 4 channels */
    snr = 10.0 * log10 ((double) ((ch_enrg [4*i] + ch_enrg[4*i+1] +
        ch_enrg[4*i+2] + ch_enrg[4*i+3])/ (ch_noise [4*i] + ch_noise[4*i
+ 1]
            + ch_noise[4*i + 2] + ch_noise[4*i + 3])));

    if (snr < 0.0) snr = 0.0;

    ch_snr_tmp [i] = (snr + 0.1875) / 0.375;

  }
  vm_sum = 0;


/* if we use smaller channels, compute the sum using
   the temporary snr vector that is the same as what we would have
   had we used the original number of channels (i.e. 16) */


    for (i = LO_CHAN; i <= 15; i++) {

    j = min(ch_snr_tmp[i],89);
    vm_sum += vm_tbl [j];
    }

#else

    for (i = LO_CHAN; i <= HI_CHAN; i++) {

    j = min(ch_snr[i],89);
    vm_sum += vm_tbl [j];

    }


#endif


/* Compute the total noise estimate (tne) and total channel */
      /* energy estimate (tce) */

  tne = tce = 0.0;

  for (i = LO_CHAN; i <= HI_CHAN; i++) {

    tne += ch_noise [i];
    tce += ch_enrg [i];

  }


/* Calculate log spectral deviation */

  for (i = LO_CHAN; i <= HI_CHAN; i++)
    ch_enrg_db [i] = 10.*log10( ch_enrg [i] );
  if (first == TRUE)
    for (i = LO_CHAN; i <= HI_CHAN; i++)
      ch_enrg_long_db [i] = ch_enrg_db [i];

  ch_enrg_dev = 0.;

  for (i = LO_CHAN; i <= HI_CHAN; i++)
    ch_enrg_dev += fabs( ch_enrg_long_db [i] - ch_enrg_db [i] );


/* Calculate long term integration constant as a function of total
/*  channel energy (tce) (i.e., high tce (-40 dB) -> slow integration
(alpha = 0.99), low tce (-60 dB) -> fast integration (alpha = 0.50) */

  alpha = HIGH_ALPHA - (ALPHA_RANGE / TCE_RANGE) * (HIGH_TCE_DB -
10.*log10(tce));
  if ( alpha > HIGH_ALPHA )
    alpha = HIGH_ALPHA;
  else if ( alpha < LOW_ALPHA )
    alpha = LOW_ALPHA;


/* Calc long term log spectral energy */

  for (i = LO_CHAN; i <= HI_CHAN; i++) {
    ch_enrg_long_db [i] = alpha*ch_enrg_long_db [i] + (1.-
alpha)*ch_enrg_db [i];
  }


/* Set or reset the update flag */
  update_flag = FALSE;
```

```
if (vm_sum <= UPDATE_THLD) {

  update_flag = TRUE;
  update_cnt = 0;

}



else if (tce > NOISE_FLOOR && ch_enrg_dev < DEV_THLD) {
  update_cnt++;
  if (update_cnt >= UPDATE_CNT_THLD)
    update_flag = TRUE;
}


if(vm_sum <= 10)
    noise_flag = TRUE;
else noise_flag = FALSE;



if ( update_cnt == last_update_cnt )
  hyster_cnt++;
else
  hyster_cnt = 0;
last_update_cnt = update_cnt;

if ( hyster_cnt > HYSTER_CNT_THLD )
  update_cnt = 0;
```

```
/* Set or reset modify flag */

  index_cnt = 0;

  for (i = MID_CHAN; i <= HI_CHAN; i++)
    if (ch_snr [i] >= INDEX_THLD)
      index_cnt++;

  modify_flag = (index_cnt < INDEX_CNT_THLD)? TRUE : FALSE;


/* Modify the SNR indices */

  if (modify_flag == TRUE)
    for (i = LO_CHAN; i <= HI_CHAN; i++)


            if ((vm_sum <= METRIC_THLD) || (ch_snr [i] <= SETBACK_THLD))
            ch_snr [i] = 1;




/***************************************************/
/* Spectral Subtraction */
/***************************************************/



/* store the channel noise estimate into a vector for spectral
subtraction */
```

```
noise_tmp[0] = 1.0;
noise_tmp[1] = 1.0;
noise_tmp[2] = 1.0;
noise_tmp[3] = 1.0;

  for(i = LO_CHAN; i <= HI_CHAN; i++){

    j1 = ch_tbl [i][0], j2 = ch_tbl [i][1];

       for (j = j1; j <= j2; j++){

          noise_tmp[j] = sqrt(ch_noise[i]);

          bin_snr_tmp[j] = ch_snr[i];


       }

     }


#ifdef SPEC_SUB

/* Perform spectral subtracton on each FFT bin*/



      for(i = 0; i < FFT_LEN/2; i++){

/* get magnitude of signal */

          mag = sqrt(square(data_buffer[2*i]) + square(data_buffer[2*i +
1]));
```

```
/* get the phase (which is preserved) */
          phz = atan2(data_buffer[2*i + 1], data_buffer[2*i]);


/* subtract the magnitude of the noise (not completely) */
if(!update_flag)
          mag = mag - spec_sub_fac*noise_tmp[i];
else
         mag = mag - spec_sub_fac*mag;


         if(mag < 0.0)
              mag = 0.0;




/* reconstruct the polar value into cartesian */



         data_bufferTmp[2*i] = 1/ATTN*mag*cos(phz);
         data_bufferTmp[2*i + 1] = 1/ATTN*mag*sin(phz);


         mag = sqrt(square(data_buffer[2*i]) + square(data_buffer[2*i +
1]));


          mag = mag - spec_sub_fac*noise_tmp[i];


          if(mag < 0.0)
              mag = 0.0;


          spec_sub_flat[2*i] = mag*cos(phz);
          spec_sub_flat[2*i + 1] = mag*sin(phz);
```

```
/*I        spec_sub_flat[2*i] = data_buffer[2*i];

           spec_sub_flat[2*i + 1] = data_buffer[2*i + 1];

*/

      }




#endif








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

/*  Compute Gains */

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




      for(i = 0; i < FFT_LEN/2; i++){

#ifndef USE_ENRG

          spec_sub_flat_mags[i] = 10 * log10(square(spec_sub_flat[2*i])
+ square(spec_sub_flat[2*i + 1]));

#else

          spec_sub_flat_mags[i] = 10*log10((1- 0.55) * pow(10,
spec_sub_flat_mags[i]/10) + 0.55 * (square(spec_sub_flat[2*i]) +
square(spec_sub_flat[2*i + 1])));


#endif

      }






/* compute magnitudes after spectral subtraction for threshold
computation */
```

```
 for(i = 0; i < FFT_LEN/2; i++){

     if(data_bufferTmp[2*i] == 0.0 && data_bufferTmp[2*i + 1] == 0.0){

         FTMagsTmp[i] = 0.0;

         spec_sub_mags[i] = 0.0;

     }

     else{

         FTMagsTmp[i] = 10 * log10(square(data_bufferTmp[2*i]) +
square(data_bufferTmp[2*i + 1]));

         spec_sub_mags[i] = FTMagsTmp[i];

     }


 }


#ifdef CNT_BANDS


 for(i = 0; i < FFT_LEN/2; i++){


     tmp = get_bark(i);

     j1 = (int) get_i_bark(tmp - 0.5);

     j2 = (int) get_i_bark(tmp + 0.5);

     j1 = max(0,j1);

     j2 = min(FFT_LEN/2 - 1, j2);

     tmp = 0.0;

     for(j = j1; j <= j2; j++){

         tmp += square(data_bufferTmp[2*j]) + square(data_bufferTmp[2*j +
1]);

     }

     tmp /= (j2 - j1 + 1);


     FTMagsTmp[i] = 10*log10(tmp);




 }
```

```
#endif

 /* compute snr based on spectral subtraction results */

/*
      for(i = 0; i < FFT_LEN/2; i++){

             bin_snr_tmp[i] = 20*log10(sqrt(square(data_bufferTmp[2*i]) +
square(data_bufferTmp[2*i + 1]))
                  /noise_tmp[i]);

      }

*/

/* Compute tonality and correction to Masking Curve*/
get_offset(offset, FFT_LEN/2, data_bufferTmp, tonality);

/* Decimate tones that are within 3 lines of each other */

#ifdef DECIMATE_ADJACENT_TONES

for(i = 0; i < FFT_LEN/2; i++){

    if(tonality[i] > TONAL_THLD && tonality[i+1] > TONAL_THLD
        && tonality[i + 2] > TONAL_THLD){

        if(FTMagsTmp[i] > FTMagsTmp[i + 1] && FTMagsTmp[i] > FTMagsTmp[i
+ 2]){
```

```
            tonality[i + 1] = 0.0;
            FTMagsTmp[i + 1] = 0.0;
            tonality[i + 2] = 0.0;
            FTMagsTmp[i + 2] = 0.0;
        }

        else {
            tonality[i] = 0.0;
            FTMagsTmp[i] = 0.0;

            if(FTMagsTmp[i + 1] > FTMagsTmp[i + 2]){
                tonality[i + 2] = 0.0;
                FTMagsTmp[i + 2] = 0.0;
            }
            else{

                tonality[i + 1] = 0.0;
                FTMagsTmp[i + 1] = 0.0;
            }

        }

    }
    else
        if(tonality[i] > TONAL_THLD && tonality[i+1] > TONAL_THLD){

        if(FTMagsTmp[i] > FTMagsTmp[i + 1]){
            tonality[i + 1] = 0.0;
            FTMagsTmp[i + 1] = 0.0;

        }
        else {
            tonality[i] = 0.0;
            FTMagsTmp[i] = 0.0;
            }
        }



}

#endif


/* Compute masking threshold */


for(i = 0; i < FFT_LEN/2; i++){


    if(!(i%3))
        FTMagsTmp[i] = 0.0;
}


#ifdef MASK_TEST


  for(i = 0; i < FFT_LEN/2; i++)
        FTMagsTmp[i] = 10*log10(square(data_buffer[2*i]) +
square(data_buffer[2*i + 1]));


#endif




#ifdef DO_MASK


#ifdef MPEG_MASK
  get_MPEG_mask(FTMagsTmp, Pthresh, tonality);
```

```
#else

  get_pmask(FTMagsTmp, Pthresh);

#endif




#endif






#ifdef MASK_TEST


    r_fft(TmpBuf, +1);


    for(i = 0; i < FFT_LEN/2; i++)

        spec_sub_flat_mags[i] = 10*log10(square(data_buffer[2*i]) +
square(data_buffer[2*i + 1]));



    for(i = 0; i < FFT_LEN/2; i++){

        data_buffer[2*i] += pow(10,(Pthresh[i/2])/20)*TmpBuf[2*i]*2;

        data_buffer[2*i + 1] += pow(10,(Pthresh[i/2])/20)*TmpBuf[2*i +
1]*2;

    }


    for(i = 0; i < FFT_LEN/2; i++)

        FTMags[i] = 10*log10(square(data_buffer[2*i]) +
square(data_buffer[2*i + 1]));




    for(i = 0; i < FFT_LEN/2; i++)

        FTMagsTmp[i] =
10*log10(square(pow(10,(Pthresh[i])/20)*TmpBuf[2*i])) +
square(pow(10,(Pthresh[i])/20)*TmpBuf[2*i+1]);
```

```
   // for(i = 0; i < FFT_LEN/2; i++)

      // FTMags[i] = 10*log10(square(TmpBuf[2*i]) + square(TmpBuf[2*i +
1]));

      //   FTMags[i] = sqrt(square(TmpBuf[2*i]) + square(TmpBuf[2*i +
1]));




    if(loop == 107){


      F_tmp1 = fopen("Mask.dat", "w");


      for(i = 0; i < FFT_LEN/2; i++){


           fprintf(F_tmp1, "%f\t%f\t%f\t%f\t%f\n", Pthresh[i],
FTMagsTmp[i], i*15.625, spec_sub_flat_mags[i], FTMags[i] );


      }

      fclose(F_tmp1);


  }


    r_fft(TmpBuf, -1);


#endif
```

```
#ifdef BND_CLS
  sc1 = 0.0;
  for(i = 0; i < FFT_LEN/8; i++)
    sc1 += bin_snr_tmp[i];



  sc2 = 0.0;
  for(i = FFT_LEN/8; i < 2*FFT_LEN/8; i++)
    sc2 += bin_snr_tmp[i];



  sc3 = 0.0;
  for(i = 2*FFT_LEN/8; i < 3*FFT_LEN/8; i++)
    sc3 += bin_snr_tmp[i];

  sc4 = 0.0;
  for(i = 3*FFT_LEN/8; i < 4*FFT_LEN/8; i++)
    sc4 += bin_snr_tmp[i];

    sc1 /= (FFT_LEN/8);
    sc2 /= (FFT_LEN/8);
    sc3 /= (FFT_LEN/8);
    sc4 /= (FFT_LEN/8);

#endif

  for(i = 0; i < FFT_LEN/2; i++){

      gain_alpha = 0.8* (bin_snr_tmp[i] - SNR_THLD) - 34;
      gain_alpha = pow(10, gain_alpha/20);
      gain_alpha = min(max(gain_alpha, 0.0), 1.0);
```

```
#ifdef BND_CLS
     if(i < FFT_LEN/8)
        gain_alpha = .39* (sc1 - SNR_THLD) - 13;

     else
        if(i < 2*FFT_LEN/8)
           gain_alpha = .39* (sc2 - SNR_THLD) - 13;
        else
            if(i < 3*FFT_LEN/8)
               gain_alpha = .39* (sc3 - SNR_THLD) - 13;
            else
                if(i < 4*FFT_LEN/8)
                    gain_alpha = .39* (sc4 - SNR_THLD) - 13;

        gain_alpha = min(max(gain_alpha, 0.0), 1.0);
#endif


#ifdef MPEG_MASK
      offset[i] = 0.0;
#endif

        gain_x = Pthresh[i]  - offset[i] - spec_sub_flat_mags[i];

        gain_x = 1.0/ATTN - pow(10, gain_x/20.0);

        gain_x = max(1.0, gain_x);

        gainXDisp[i] = 20*log10(gain_x);


#ifndef GUSTAF
```

```
        gain_n = Pthresh[i] - 20.0*log10(noise_tmp[i]);

        gain_n = pow(10,(Pthresh[i] -
offset[i])/10)/square(noise_tmp[i]);

        gain_n = sqrt(gain_n);



        gain_n = 1.0 + gain_n;

        gain_n = min(gain_n, 1/ATTN);

        gainNDisp[i] = 20*log10(gain_n);

        gain = (gain_alpha)*gain_x + (1.0 - gain_alpha) * gain_n;

        //gain = gain_x;



        ch_gain[i] = min(gain, 1/ATTN);

        ch_gain[i] = max(gain, 1.0);

#ifdef WIENER

      ch_gain[i] = 1/(ATTN)*(bin_enrg[i] - bin_noise[i])/bin_enrg[i];


#endif


#else


        gain_n = pow(10,(Pthresh[i] -
offset[i])/10)/square(noise_tmp[i]);

        gain_n = sqrt(gain_n);



        gain_n = ATTN + gain_n;

        gain_n *= 10.;
        ch_gain[i] = min(gain_n, 10);

#endif



  }


  ch_gain[FFT_LEN/2] = 0.0;


for(i = 0; i < FFT_LEN/2; i++){



#ifndef NO_FILTER
    data_buffer[2*i] *= ch_gain[i];
    data_buffer[2*i + 1] *= ch_gain[i];


#endif



}


#ifdef ADPT_SS
if(update_flag){
spec_sub_enrg = 0.0;



    for(i = 0; i < FFT_LEN/2; i++){
        spec_sub_enrg += (sqrt(square(data_bufferTmp[2*i]) +
square(data_bufferTmp[2*i + 1])) / FFT_LEN);
    }
```

```
spec_sub_fac += (spec_sub_enrg - ATTN*output_enrg) /output_enrg;
}

#endif

/* Update the channel noise estimates */

#ifndef OLD_NOISE_CHANNELS

  if (update_flag == TRUE)
    for (i = LO_CHAN; i <= HI_CHAN; i++) {

      ch_noise [i] = (1.0 - CNE_SM_FAC) * ch_noise [i] +
                        CNE_SM_FAC * ch_enrg [i];

      if (ch_noise [i] < MIN_CHAN_ENRG) ch_noise [i] = MIN_CHAN_ENRG;

    }

#else

    if (update_flag == TRUE){

/* Use the old, wider channels for noise estimate */
     for (i = LO_NOISE_CHAN; i <= HI_NOISE_CHAN; i++) {

      ch_noise_tmp [i] = (1.0 - CNE_SM_FAC) * ch_noise_tmp [i] +
                        CNE_SM_FAC * ch_enrg_tmp [i];

      if (ch_noise_tmp [i] < MIN_CHAN_ENRG) ch_noise_tmp [i] =
MIN_CHAN_ENRG;
```

```
    }



/* copy back into ch_noise vector */


     for(i = LO_CHAN; i <= HI_CHAN; i++){


         ch_noise[i] = ch_noise_tmp[i/4];
    }


    }
#endif




#ifdef GRAPH
     for(i = 0; i < FFT_LEN/2; i++){
      FTMagsTmp[i] = 10*log10(square(data_buffer[2*i]) +
square(data_buffer[2*i+1]));
  }


#endif


#ifndef NO_FFT
  r_fft (data_buffer, -1);
#endif




#endif


#ifndef TRAP_WIN
```

```
/* perform overlap add */

  for(i = 0; i < BUF_LEN; i++)
      overlap_add[i + BUF_LEN*OVL_RATIO_X_4/4] += data_buffer[i];

  /* de-emphasis */

  *farray_ptr = 0.4639*overlap_add[BUF_LEN*OVL_RATIO_X_4/4] +
DE_EMP_FAC*de_emp_mem;

  for(i = 1; i < FRM_LEN; i++)
      *(farray_ptr + i) = WINDOW_GAIN_FAC*overlap_add[i +
BUF_LEN*OVL_RATIO_X_4/4] + DE_EMP_FAC * *(farray_ptr + i - 1);

  /* save last sample for de-emphasis in next frame */

  de_emp_mem = *(farray_ptr + FRM_LEN - 1);

/* shift up overlap add buffer for next round */

  for(i = 0; i < (2*BUF_LEN - FRM_LEN); i++)
      overlap_add[i] = overlap_add[i + FRM_LEN];

  for(i = (2*BUF_LEN - FRM_LEN); i < (2*BUF_LEN); i++)
      overlap_add[i] = 0.0;

#else
```

```
  for(i = 0; i < FRM_LEN; i++){

      *(farray_ptr + i) = data_buffer[OVL_LEN/2 + i];
  }

#endif



/* the next call won't be the first frame to be processed  */

  first = FALSE;



/* sum up the snr s for the channels to display */

#ifdef GRAPH

  tmp = 0.0;

for(i = LO_CHAN; i <= HI_CHAN; i++)
      tmp += pow(10,0.375/10*ch_snr[i]);
#endif




/* write the update flag and voice metric sum in this frame into update
buf this is repeated for FRM_LEN times in order to align
 it with the input & output files */



for( i = 0; i < FRM_LEN; i++){
      UpdateBuf[i] = pow(2, 13) * update_flag;
    if(vm_sum > METRIC_THLD)
```

```
        VmBuf[i] = 32764/2;
    else VmBuf[i] = 0.0;
}



/* Graphing */


#ifdef GRAPH



for(i = 0; i < FFT_LEN/2; i++)
  ch_gain_disp[i] = 20*log10(ATTN*ch_gain[i]);


/* display the variables  and graphs*/
#ifdef VAR_GAIN
    master_var_display(2, update_flag, "UPDATE_FLAG");
#endif


    if(ch_snr_disp_draw == 1)
            master_load(ch_snr_disp_num, PCM);

    if(ch_gain_disp_draw == 1)
            master_load(ch_gain_disp_num, ch_gain_disp);

    if(FTMags_draw == 1)
            master_load(FTMags_num, FTMags);
```

```
if(Threshold_draw == 1)
        master_load(Threshold_num, Pthresh);
for(i=0; i < FFT_LEN/2; i++)
    ch_snr_disp[i] = (float)bin_snr_tmp[i];

if(SNR_draw == 1)
        master_load(SNR_num, ch_snr_disp);



if(UnFiltDat_draw == 1)
        master_load(UnFiltDat_num, FTMags); // ftmags


if(FiltDat_draw == 1)
        master_load(FiltDat_num, FTMagsTmp); //ftmagstmp


if(FiltDat2_draw == 1)
        master_load(FiltDat2_num, FTMagsTmp);


if(Threshold2_draw == 1)
        master_load(Threshold2_num, Pthresh);   //pthrest


for(i = 0; i < FFT_LEN/2; i++)
    noise_tmp[i] = 20* log10(noise_tmp[i]);


if(Noise_draw == 1)
    master_load(Noise_num, noise_tmp);


if(FTMags2_draw == 1)
    master_load(FTMags2_num, FTMags);  //ftmags



if(Threshold3_draw == 1)
```

```
        master_load(Threshold3_num, Pthresh); //pthresh

    if(spec_sub_mags_draw == 1)
        master_load(spec_sub_mags_num, spec_sub_mags);


    if(gain_x_draw == 1)
        master_load(gain_x_num, gainXDisp);

    if(gain_n_draw == 1)
        master_load(gain_n_num, gainNDisp);



    master_graph();
    master_suspend();

#endif

}            /* end noise_suprs () */


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

void init_window (float *x, int n, float ovlap)
{
      int i;
      float arg;
      int n1;

      /* use smoothed trapezoidal window */

      n1 = (int) ceil(ovlap * n /2.);
      arg = 2.*atan(1.)/n1;
      for (i=0; i<n1; i++) {
            x[i] = pow( sin( (i+0.5)*arg ), 2. );
      }
      for (i=n1; i<n-n1; i++) {
            x[i] = 1.;
      }
      for (i=n-n1; i<n; i++) {
            x[i] = pow( sin( ((i-n)+0.5)*arg ), 2. );
      }

      return;

}     /* end of init_window() */

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




void get_pmask(float *L, float *th)
{

  float s[FFT_LEN/2];
  int u,v;
  double sum;
  double uth_part,intermed_variable;
  float t, t_i, t_f;

  int SIZE = FFT_LEN/2;
  s[0] = -24 - 230/31.25 + .2*L[0];


  for ( v=1; v<SIZE; v++)
      s[v] = -24.-230.*FFT_LEN/((v)*SAMPLE_RATE)+.2*L[v];


  for ( u=0; u<SIZE; u++){
      sum = 0.;
      for ( v=1; v<u; v++){
          if(L[v] > 0.0 ){
            intermed_variable = L[v]-s[v]*(get_bark(v)-get_bark(u));
              sum += pow(10.0, intermed_variable/20);


         }
      }
      for( v=u+1; v<SIZE;v++){
          if(L[v] > 0.0){
          intermed_variable = L[v]-27.*(get_bark(v)-get_bark(u));
            sum += pow(10.0, intermed_variable/20);
          }
      }
      th[u] = (float)10.*(log10(sum));
  }

} /** end thresh **/








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

```
void tonal_measure(float in[], float out[])
{
    int i, j, mag, phz, loop = 0;

    static float mag_1[FFT_LEN/2], mag_2[FFT_LEN/2];
    static float phz_1[FFT_LEN/2], phz_2[FFT_LEN/2];

    float   mag_est[FFT_LEN/2], phz_est[FFT_LEN/2];
    float   cur_mag[FFT_LEN/2], cur_phz[FFT_LEN/2];

    float   chaos_measure[FFT_LEN/2], tmp;



/* get magnitude and phase of input */

    for(i = 0; i < FFT_LEN/2; i++){

        cur_mag[i] = sqrt(square(in[2*i]) + square(in[2*i + 1]));
        cur_phz[i] = atan2(in[2*i + 1], in[2*i]);
    }



/* compute estimate of magnitude and phase of input */

    for(i = 0; i < FFT_LEN/2; i++){
        mag_est[i] = mag_1[i] + (mag_1[i] - mag_2[i]);
        phz_est[i] = phz_1[i] + (phz_1[i] - phz_2[i]);
    }

/* compute the chaos measure */

    for(i = 0; i < FFT_LEN/2; i++){
        tmp = square(mag_est[i] - cur_mag[i]);
        tmp += square(phz_est[i] - cur_phz[i]);
        tmp = (float)sqrt((double)tmp);

        tmp /= (cur_mag[i] + (float)abs((double)mag_est[i]));

       tmp = max(0.05, min(0.5, tmp));
        chaos_measure[i] = tmp;

    }

/* map chaos measure to tonal measure */

    for(i = 0; i < FFT_LEN/2; i++){
        out[i] = -0.43*log10(chaos_measure[i]) - 0.299;
    }



    for(i = 0; i < FFT_LEN/2; i++){

        mag_2[i] = mag_1[i];
        phz_2[i] = phz_1[i];

        mag_1[i] = cur_mag[i];
        phz_1[i] = cur_phz[i];
    }

} /* end tonal measure */


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

```
void get_offset(float offset[], int len, float S[], float tonality[])
{

    int i, j, j1, j2;
    int first_bark, bark_size;

    float tmp = 0.0, Gm = 0.0, Am = 0.0;


#ifdef TONAL_MEASURE

    tonal_measure(S, tonality);


    for(i = 0; i < FFT_LEN/2; i++){

            tonality[i] = max(tonality[i], -0.17);
            tonality[i] = min(tonality[i], 0.26);

            tonality[i] = tonality[i] + 0.17;
            tonality[i] *= 2.32559;

        //  tmp+= tonality[i];
        //  offset[i] = (1 - tonality[i])*(14.5 + get_bark(i));// +
(1 - tonality[i])*5.5;
    }
    for(i = 0; i < FFT_LEN/2; i++){
        Gm *= (square(S[2*i]) + square(S[2*i + 1]));
        Am += square(S[2*i]) + square(S[2*i + 1]);
    }
    Gm = pow(Gm, 1/(FFT_LEN/2));
    Am /= (FFT_LEN/2);

    tmp = Gm/Am;
    tmp = 10*log10(tmp);

    tmp = min((tmp/(-60)), 1);

    for(i = 0; i < FFT_LEN/2; i++){
        offset[i] = tmp*(14.5 + get_bark(i)) + (1 - tmp)*5.5;

       offset[i] *= fabs(tmp + tonality[i] - 1) - MASK_ADJ;

}

#endif

}  /* end get_offset */

float get_bark(int i)
{
    float tmp;

#ifdef TBL_BARK
    return barkval[i];
#else
    tmp = (float) i;

    tmp *= ((float)SAMPLE_RATE)/((float)FFT_LEN);

    return 26.81f*tmp/(1960. + tmp) - 0.53f;
```

```
#endif

} /* end get_bark */



int get_i_bark(float z)

{


  return (int) ((1960.0f*z + 1038.8f)/(26.68f - z)) / SAMPLE_RATE *
FFT_LEN;

}




void estimate_noise(float in[], float out[])

{

    int i, j, lo_bark_line, hi_bark_line, count;

    static int first = TRUE;

    float z, enrg, alpha;


    alpha = (first == TRUE) ? 1.0 : CEE_SM_FAC;



    for(i = 0; i < FFT_LEN/2; i++){

        z = get_bark(i);

        lo_bark_line = get_i_bark(z - 0.5);

        hi_bark_line = get_i_bark(z + 0.5);

        enrg = 0;

        for(j = lo_bark_line; j <= hi_bark_line; j++);


    }


}
```

```
void get_MPEG_mask(float *L, float *th, float tonality[])
{

  float s[FFT_LEN/2];
  int u,v;
  double sum;
  double uth_part,intermed_variable;
  float t, t_i, t_f, tmp;
  int bark_u, bark_v, i_bark_v;
  int lw_b, up_b;
  float dz;

  int SIZE = FFT_LEN/2;

  for(u = 0; u < SIZE; u++){

      bark_u = get_bark(u);
      sum = 0.0;

      lw_b = get_i_bark(bark_u - 3);
      up_b = get_i_bark(bark_u - 1);
      up_b = min(up_b, SIZE);
      lw_b = max(lw_b, 0);
      for(v = lw_b; v < up_b; v++){

            bark_v = get_bark(v);
            dz = (float) abs( bark_u - bark_v);
            tmp = L[v];
            tmp += -1.525 - 0.175 * bark_v - 0.5 - 4 * tonality[v] -
0.1*bark_v * tonality[v];
```

```
            tmp += 17 * (dz + 1) - (0.4*L[v] + 6);
            sum += pow(10, tmp/10);
}

        lw_b = get_i_bark(bark_u - 1);
        up_b = get_i_bark(bark_u);
        up_b = min(up_b, SIZE);
        lw_b = max(lw_b, 0);

        for(v = lw_b; v < up_b; v++){
            bark_v = get_bark(v);
            dz = (float) abs( bark_u - bark_v);
            tmp = L[v];
            tmp += -1.525 - 0.175 * bark_v - 0.5 - 4 * tonality[v] -
0.1*bark_v * tonality[v];
            tmp += (0.4*L[v] + 6) * dz;
            sum += pow(10, tmp/10);
}

        lw_b = get_i_bark(bark_u);
        up_b = get_i_bark(bark_u + 1);
        up_b = min(up_b, SIZE);
        lw_b = max(lw_b, 0);

        for(v = lw_b; v < up_b; v++){
            bark_v = get_bark(v);
            dz = (float) abs( bark_u - bark_v);
            tmp = L[v];
            tmp += -1.525 - 0.175 * bark_v - 0.5 - 4 * tonality[v] -
0.1*bark_v * tonality[v];
            tmp += - 17 * dz;
            sum += pow(10, tmp/10);
```

```
}



      lw_b = get_i_bark(bark_u + 1);
      up_b = get_i_bark(bark_u + 8);
      up_b = min(up_b, SIZE);
      lw_b = max(lw_b, 0);


      for(v = lw_b; v < up_b; v++){
            bark_v = get_bark(v);
            dz = (float) abs( bark_u - bark_v);
            tmp = L[v];
            tmp += -1.525 - 0.175 * bark_v - 0.5 - 4 * tonality[v] -
0.1*bark_v * tonality[v];
            tmp += -1*(dz - 1)*(17 - 0.15*L[v]) - 17;
            sum += pow(10, tmp/10);
}


    th[u] = 10*log10(sum);



  }


} /** end thresh **/
```

## Appendix B: Voice Metric Table (Table 4.1)

| SNR Index | Voice Metric |
|---|---|
| 0 | 2 |
| 1 | 2 |
| 2 | 2 |
| 3 | 2 |
| 4 | 2 |
| 5 | 2 |
| 6 | 2 |
| 7 | 2 |
| 8 | 2 |
| 9 | 2 |
| 10 | 2 |
| 11 | 3 |
| 12 | 3 |
| 13 | 3 |
| 14 | 3 |
| 15 | 3 |
| 16 | 4 |
| 17 | 4 |
| 18 | 4 |
| 19 | 5 |
| 20 | 5 |
| 21 | 5 |
| 22 | 6 |
| 23 | 6 |
| 24 | 7 |
| 25 | 7 |
| 26 | 7 |
| 27 | 8 |
| 28 | 8 |
| 29 | 9 |
| 30 | 9 |
| 31 | 10 |
| 32 | 10 |
| 33 | 11 |
| 34 | 12 |
| 35 | 12 |
| 36 | 13 |
| 37 | 13 |
| 38 | 14 |
| 39 | 15 |
| 40 | 15 |
| 41 | 16 |
| 42 | 17 |
| 43 | 17 |
| 44 | 18 |
| 45 | 19 |
| 46 | 20 |
| 47 | 20 |
| 48 | 21 |
| 49 | 22 |
| 50 | 23 |
| 51 | 24 |
| 52 | 24 |
| 53 | 25 |
| 54 | 26 |
| 55 | 27 |
| 56 | 28 |
| 57 | 28 |
| 58 | 29 |
| 59 | 30 |
| 60 | 31 |
| 61 | 32 |
| 62 | 33 |
| 63 | 34 |
| 64 | 35 |
| 65 | 36 |
| 66 | 37 |
| 67 | 37 |
| 68 | 38 |
| 69 | 39 |
| 70 | 40 |
| 71 | 41 |
| 72 | 42 |
| 73 | 43 |
| 74 | 44 |
| 75 | 45 |
| 76 | 46 |
| 77 | 47 |
| 78 | 48 |
| 79 | 49 |
| 80 | 50 |
| 81 | 50 |
| 82 | 50 |
| 83 | 50 |
| 84 | 50 |
| 85 | 50 |
| 86 | 50 |
| 87 | 50 |
| 88 | 50 |
| 89 | 50 |

# Appendix C: Voice Activity Detection Algorithm

```
update_flag = FALSE
/* the update_flag is set when the voice metric sum is very low,
indicating that the SNR in the frame is low */

if  (VM_SUM  =<  UPDATE_THLD){
      update_flag = TRUE
      update_cnt  =  0
}

/* This logic takes care of the situation when the SNR may be high but
the spectrum isn't changing very much. This case will occur when the
noise increases in power. The SNR will be high and so the above check
will not catch the noise. So, a check is added to see if the spectrum
does not change much compared to the long term average spectrum for
several frames. Also, a check is needed for the case of the total energy
being below the noise floor. This is so that the spectrum is not
averaged for really small energies. */

else if ((E_tot > NOISE_FLOOR_DB) and (long_term_dev < DEV_THLD))
{
      update_cnt = update_cnt + 1                   /* count number of
times the

spectral  deviation is small */

      if( update_cnt  >= UPDATE_CNT_THLD)       /* if the spectral
deviation stays
                                              small for many frames set
                                              the update flag */
      update_flag = TRUE

}

/* some hysterisis logic is also needed in order to prevent the
update_cnt from creeping up over a long period of time. This will ensure
that the update_flag is set only when the spectral deviation is small
for a number of frames that are fairly continous */

if(update_cnt == last_update_cnt)    /* if the spectral
                                       deviation isn't
                                              high, => the update_cnt is
                                              the same, increase the
                                              hysterisis count */
      hyster_cnt = hyster_cnt + 1
else
      hyster_cnt = 0                        /* if the update_cnt has
                                       changed i.e. this
                                              frame has low long
                                       term deviation */

last_update_cnt = update_cnt         /* update the last
                                       update count */

if(hyster_cnt > HYSTER_CNT_THLD)     /* if the spectral
                                       deviation is small
                                              for a number of
                                       continous frames
```

```
                                                            start the update
                                                            count from scratch
                                                            again */
        update_cnt = 0
```

the values of the thresholds are used as in [1] are as follows: UPDATE_THLD = 35, NOISE_FLOOR_DB = 0, DEV_THLD = 28, UPDATE_CNT_THLD = 50, HYSTER_CNT_THLD = 6.