USE OF WIDE CONTRIBUTION SIGNATURES FOR REAL-TIME GENERALIZED AUDIO RECOGNITION

by

Malay Kundu

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
MASTER OF SCIENCE
IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 1996

Signature of Author _____
Department of Electrical Engineering and Computer Science, May 1996

Certified by _____
Bernard Gold, Professor Emeritus of Electrical Engineering Thesis Supervisor

Certified by _____
Scott Diamond, Tektronix Company Supervisor Thesis Supervisor

Accepted by _____
Frederic R. Morgenthaler Chairman, Department Committee on Graduate Theses

USE OF WIDE CONTRIBUTION SIGNATURES

FOR

REAL-TIME GENERALIZED AUDIO RECOGNITION

by

Malay Kundu

Submitted to the Department of Electrical Engineering and Computer Science

on May 10, 1996, in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

ABSTRACT

A system has been designed for real-time recognition of arbitrary known audio segments as they occur within a continuous audio broadcast. The audio segments must be successfully recognized despite distortions such as noise, compression-decompression artifacts, and playback speed shift. The system attempts to achieve robust recognition by first representing the signal as a set of wide contribution signatures (WCS) and then breaking up the complex recognition task into a corresponding set of simpler, robust recognition tasks. Each of the simpler recognition tasks is designed with an easily satisfied broad acceptance region to ensure robustness. Although they are therefore likely to misfire and report false positive matches individually, the tasks each test inherently different properties of the signal (as represented by the different WCS's) and therefore do not misfire at the same time. Only when all of the recognition tasks are satisfied concurrently do they indicate a true positive match. Thus, by representing and recognizing the signal in terms of a set of *inherently different* wide contribution signatures, the system attempts to maintain both robustness as well as discrimination in recognition of generalized audio.

The system was tested with audio samples from radio, low-quality tape, cable television, and MPEG compressed digital audio. Recognition from radio and low-quality tape resulted in 71% and 83% detection rates despite noise and shift in playback speed. Detection rates with cable television and MPEG compressed audio were 100%.

Thesis Supervisors: Bernard Gold, Professor Emeritus of Electrical Engineering
Scott Diamond, Tektronix Company Supervisor

I would like to thank Scott Diamond, Rama Pailoor, Ganesh Rajan, Mike Coleman, and all the other engineers at Tektronix who put up with my many many questions and entertained all my crazy unconventional attempts at getting this recognition system to happen. I would like to also thank Professor Ben Gold for advising me from afar while I was at Tektronix and then helping me once again back here in Boston.

I owe a sincere debt of appreciation to my roommates Nick and Prashant who have had to listen to all my frustrations and late night typing as I was writing up this thesis.

And lastly, I'd like to thank my mother, father, little brother Mukul, and Baba Lokenath just because.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

This paper addresses the design of a system for real-time recognition of arbitrary known audio segments as they occur within a continuous audio broadcast. The audio segments must be successfully recognized despite distortions such as noise, compression-decompression artifacts, and playback speed shift.[1] The system presented in this paper attempts to achieve robust recognition by first representing the signal as a set of wide contribution signatures (WCS) and then breaking up the complex recognition task into a corresponding set of simpler, robust recognition tasks. Each of the simpler recognition tasks is designed with an easily satisfied broad acceptance region to ensure robustness. Although they are therefore likely to misfire and report false positive matches individually, the tasks each test inherently different properties of the signal (as represented by the different WCS's) and therefore do not misfire at the same time. Only when all of the recognition tasks are satisfied concurrently do they indicate a true positive match. Thus, by representing and recognizing the signal in terms of a set of *inherently different* wide contribution signatures, it may be possible to maintain both robustness as well as discrimination in recognition of generalized audio.

Motivation for design of a real-time generalized audio recognition system stems from the desire to automate tasks such as the testing of transmission systems and the collection of music royalties from radio stations. Currently, methods for characterizing television broadcast transmission and reception systems require insertion of the vendor's fixed test signal into the vertical sync region before the signal is transmitted; when the signal is received, the condition of the post-transmission signal can be compared with its known original state in order to characterize the transmission and reception systems. Rather than troubling the customer with having to insert the vendor's test

signal, a real-time generalized audio recognition system used at the reception end will be able to monitor the broadcast for the occurrence of a known commercial or television program which can then be compared with its pre-transmission recording to perform the characterization. In the area of music royalty collection, there is currently no efficient method for monitoring radio stations to catalog how many times they play any given song or commercial jingle, and so payments are made on an "honor-system" basis. There are vans that sporadically go to different cities to check up on radio stations, but this is by no means practical. Use of a generalized audio recognition system will clearly automate such a task.

1.1 Broadcast Issues

The primary challenges to successful recognition are set forth by three different forms of broadcasting. These include playback speed shift in radio transmission, audio companding in television transmission, and compression-decompression artifacts in digital audio transmission.

1.1.1 Radio

Radio stations are obligated to numerous advertisers to play commercials between songs within certain time slots. Since songs have no fixed standard durations as do television programs, radio stations are known to occasionally either speed up or slow down songs and/or commercials in order to fit everything into specified amounts of time so as to meet all their obligations.

The need to deal with shift in playback speed is perhaps one the most significant challenges to successful recognition. Recognition despite speed shift necessitates a high degree of robustness in that it requires the ability to ignore *absolute* discrepancy in favor of recognizing *relative* similarity.

---

[1] The most successful prior attempt at such recognition was implemented by use of neural nets, but like other methods, it handled signals that were played back slightly faster or slower with only limited success.

1.1.2 Television

Audio noise reduction has been incorporated on both the transmission and reception end of television broadcasting in the US[2] by development of the BTSC MTS (Multi-Channel Television Sound) audio transmission standard. (Crutchfield, 1985) The system affects both the dynamic range and the high frequency content of a signal by employing a combination of fixed preemphasis, spectral compression, and amplitude compression. The result is a dynamic range reduction of 2:1 for low frequencies and 3:1 for high frequencies. An audio signal with 40 dB dynamic range, for example, is therefore reduced to 15 to 20 dB for transmission.

Although a single companding process should leave the audio signal relatively free of artifacts, television programs undergo noise reduction of this sort 5 to 6 times before reaching the viewer. Although any resulting artifacts may not be objectionable by viewers, they are viewed as comparison discrepancies in the process of recognition.

1.1.3 Digital Audio

The utility of digital audio transmission is judged by its ability to transmit large amounts of audio information with minimal degradation of quality. In order to support transmission of large amounts of information over limited-bandwidth communication channels, the information must first be compressed. Furthermore, in order to compress information without producing audible artifacts, compression-decompression schemes must take into account psychoacoustic models of the human auditory system.

Although implemented differently by various compression schemes, all compression schemes employ a number of key concepts (Noll, 1993):

---

[2] BBC uses the NICAM (Near-Instantaneously Companded Audio Multiplex) system. (Noll, 1993)

- *Simultaneous Masking* is a frequency domain phenomenon exhibited by the human auditory system where one pure tone (the masker) can simultaneously make another tone inaudible if the second tone (the maskee) is weaker and nearby in frequency. Compression schemes take this phenomenon into account by deleting frequency components that will be masked and therefore not perceived by human hearing.

- *Temporal Masking* is a time domain phenomenon where a strong sound event masks other events that are weaker and nearby in time, both before and after the strong event. Compression schemes can take advantage of this phenomenon in the same way they do simultaneous masking.

- *Dynamic Bit Allocation* is the process by which compression schemes devote more or less or (in some cases) no resolution to different subbands of the frequency spectrum at different times.

Of the three compression concepts, dynamic bit allocation is the one most likely to produce audible artifacts. In the process of dynamic bit allocation, samples are broken into blocks of samples and then normalized such that the greatest magnitude in each block is normalized to unity. If a relatively silent part of the signal is followed by a sharp attack, such as by a percussion instrument, then the entire block will be normalized by the peak of the attack. Normalization for such a high peak will reduce the quantization resolution for the entire block and thus introduce considerable quantization noise into the originally silent part of the block. Instead, the post-decompression signal will contain a noncausal "pre-echo" before the attack. This can remedied by reduction of the block length such that the pre-echo is naturally pre-masked by auditory system (see temporal masking above), but the reduction in block length is undesirable because it consequently increases the bit rate of side information sent with each block.

Although the commercial implementation of compression systems mandates minimization of artifacts that are noticeable by humans, the artifacts can nevertheless be noticed by recognition systems. This poses a twofold situation for recognition -- recognition is facilitated by the fact that compression schemes are always attempting to reduce audible artifacts, but at the same time this requires that recognition systems be as robust to localized artifacts as humans.

2. Problem Statement

This paper addresses the design of an algorithm for a real-time generalized audio recognition system capable of identifying any given pre-recorded audio segment (referred to from now on as a "jingle") upon its occurrence within an audio broadcast despite distortions to the signal incurred during transmission.

2.1 System Requirements

Given a recording (pre-transmission) of a particular segment of audio[3], the system is required to be able to monitor a continuous audio broadcast (post-reception) for the occurrence of that segment.

Since the input audio stream is continuous and of indefinite length, the throughput of the recognition system must be at least as fast as the input information flow so as to avoid unbounded accumulation of unprocessed input signal. This implies that the system must be able to operate in real-time with only a limited delay.

Since the audio stream may be broadcast over the air, the recognition system must be able to handle noise in the signal. For the purpose of this paper, the noise floor will be assumed to be 40dB below the maximum amplitude of the signal.

As mentioned in Section 1.1.1, compression-decompression schemes may alter the audio signal in several ways. The signal may look more "clean" due to deletion of masked frequency components and masked time-domain events. Because of low resolution devoted to certain subbands at certain times (dynamic bit allocation), phase may be altered non-uniformly across different subbands in the process of compression-decompression. Despite these artifacts, though, the audio signal should

[3] The fact that the segment of audio being searched for has already been pre-recorded greatly simplifies the problem. This means that the system knows exactly what it is looking for, i.e., there are no issues of

sound almost the same to the human ear. Although the recognition system will be able to notice the artifacts (unlike the human ear), it must still maintain robustness to these artifacts and instead compare the overall signal.

The third form of distortion, as mentioned in Section 1.1.1, is shift of the playback speed. In terms of time and frequency domains, speeding up the signal will cause compression along the time axis and expansion along the frequency axis; and slowing down the signal will do the opposite. The recognition system must be robust to this type of warping and must recognize the signal regardless of it. For the purpose of this paper, we will assume the speed will vary by no more than ±10%.

In summary, the recognition system must satisfy the following requirements:

- generalized for all forms of wide-band audio
- finite-delay real-time processing
- robustness to deletion of the weaker frequency components
- robustness to deletion of the weaker time events
- robustness to phase alteration
- robustness to frequency expansion/compression accompanied by time compression/expansion

while also trying to maintain

- minimal computational expense
- minimal memory storage

---

determining "intent" such as with speech recognition. All of the distortion will be introduced to the signal only in the process of transmission, not in the process of signal formation.

3. Representational Limitations

3.1 Time Domain

A purely time domain representation lacks the robustness required for the recognition system because of three main drawbacks:

- synchronization is critical
- correct playback speed is critical
- correct phase is critical

All of the drawbacks associated with a purely time-domain representation of the signal stem from the fact that comparison of two signals can only be performed on a sample-for-sample basis. Time-domain representations of incoming signal and original signal may be compared, for example, by means of a cross-correlation, i.e., sliding multiplication and sum between the individual samples of one signal with those of the other. Because the instantaneous amplitude of the signal is so critical to such a comparison, even identical signals require sliding until the beginnings of the two signals are exactly aligned for a comparison resulting in a positive match. Moreover, in order that the instantaneous amplitudes match correctly, there must be no nonlinear phase shift between the two signals, even if they have identical frequency content. Lastly, since correct synchronization is very important, it is a necessity for positive comparison that the speeds of the two signals match exactly so as to avoid missynchronization resulting from the time lag between two signals at different playback speeds.

The drawbacks resulting as a consequence of sample-for-sample comparison make the purely time domain representation unsuitable for this recognition task. The issue of synchronizing the beginnings of the signals could be handled by sliding the signal one sample at a time until a positive

match, but this would be rather expensive requiring a segment-length's worth of comparisons (such as multiplications and additions for correlation) for each unit sample of sliding. The inability to handle nonlinear phase shift rules out successful recognition of signals output by frequency domain compression schemes such as ISO/MPEG Layers I, II, and III that allocate different bit rates to different frequency subbands dynamically. Lastly, the requirement of exact speed matching makes the time domain representation impractical for use with radio broadcasts where the playback speed is known to vary.

3.2 Frequency Domain

For the purpose of this discussion, frequency domain representations are being divided into two basic categories: subband filtered representations and Short-Time Fourier Transform representations. It should be noted that these two categories are not necessarily mutually exclusive as equally spaced subband filtering can be implemented by means of an STFT. Instead, the subband filtering section discusses the limitations imposed by the lower frequency resolution characteristic of subband filtered representations, whereas the STFT section discusses problem related specifically to the STFT regardless of high or low resolution.

3.2.1 Subband Filtering

One method of frequency domain representation is to implement a filterbank that separates the signal's frequency components into frequency subbands. The instantaneous powers (over some time window) of each of these filterbanks could then be compared between the template jingle and the incoming signal for the purpose of recognition. Since the power found from each filterbank disregards the phase of the signal, a subband representation is will not be affected by nonlinear phase alteration.

The one most significant limitation of a subband representation is the relatively wide bandwidth of the subband filters in the filterbank. The resulting low frequency resolution of the representation makes it non-robust to affects of playback speed shift. In terms of frequency, an increase in the playback speed causes a proportional increase in each of the frequencies as if stretching along a continuous frequency axis. Since the filterbank imposes discrete, low-resolution subbands upon the signal, each frequency component may or may not appear to change its frequency location (i.e., subband) depending on the amount of speed shift and the component's exact distance from the edge of its subband. Since neither of these factors can be known apriori, the subband filtered representation is not robust to the effects of playback speed shift.

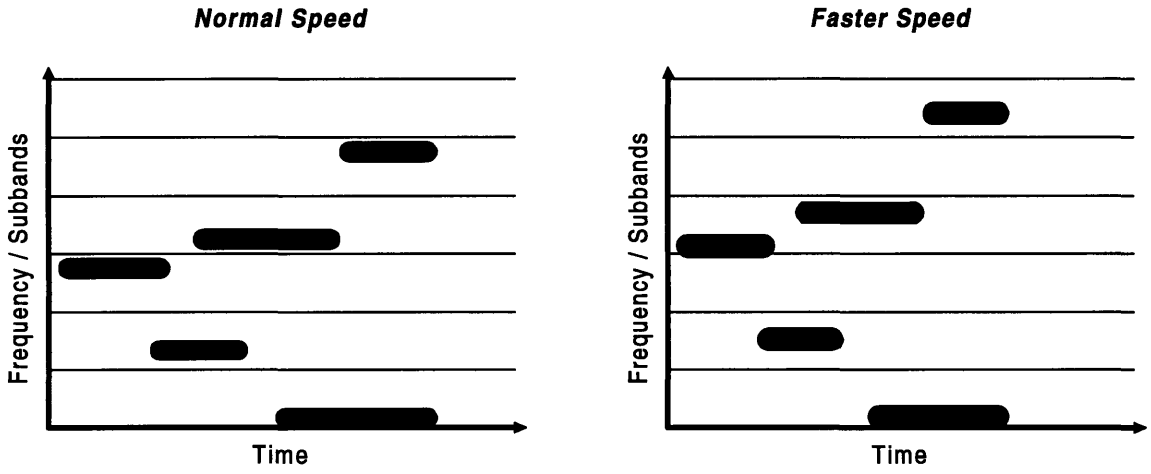

Figure 1: Speed Shift Effects on Low Resolution Subband Representation

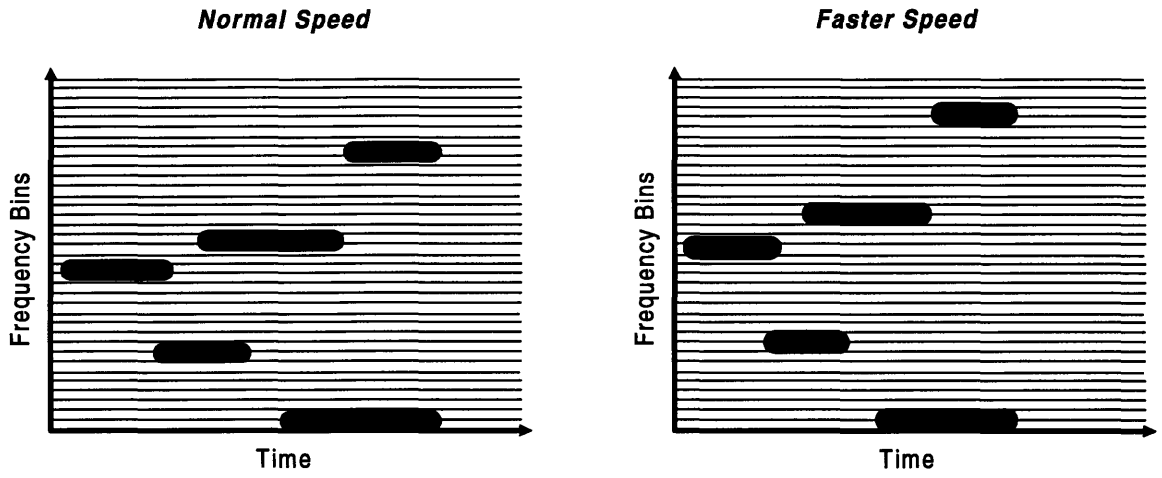

Figure 2: Speed Shift Effects on High-Resolution STFT Representation

3.2.2 Short-Time Fourier Transform

The Short-Time Fourier Transform (STFT) provides a useful frequency-domain representation because of its computational efficiency and potentially high resolution. Because it separates the signal into many different frequency bins, most compression-decompression artifacts remain isolated and do not affect the entire representation.[4] As demonstrated in

Figure 2, because of its relatively high resolution (as compared to subband filterbanks), the movement of frequency components due to playback speed shift should be readily apparent in the STFT representation.

The one major drawback of using the STFT for recognition purposes is the existence of cross-components. (Kadambe and Boudreaux-Bartels, 1992) Although the Fourier transform is linear by definition, its behavior is nonlinear in terms of either energy or phase individually.

$$|S_{(x1+x2)}(t,\omega)|^2 = |S_{x1}(t,\omega) + S_{x2}(t,\omega)|^2$$

$$= |S_{x1}(t,\omega)|^2 + |S_{x2}(t,\omega)|^2 + 2\,\mathrm{Re}\,[S_{x1}(t,\omega)S^*_{x2}(t,\omega)]$$

$$= |S_{x1}(t,\omega)|^2 + |S_{x2}(t,\omega)|^2 \quad \textbf{autocomponents}$$

$$+ 2|S_{x1}(t,\omega)|\,|S_{x2}(t,\omega)| \times \cos(\phi_{x1}(t,\omega) - \phi_{x2}(t,\omega)) \quad \textbf{cross-components}$$

As can be gathered from the above equation, the cross-components exist only when the time-frequency support of $S_{x1}(t,\omega)$ and $S_{x2}(t,\omega)$ overlap each other therefore making the $|S_{x1}(t,\omega)|\,|S_{x2}(t,\omega)|$ product nonzero. In such intersections between two transforms, the equation also implies that the cross-terms may have a magnitude as great as double the product of magnitudes of the two spectrograms. Lastly, according to the equation, the magnitude of the cross terms fluctuates as a function of the difference in center frequencies and center times.

The practical effects of cross terms are demonstrated in Figure 3, Figure 4, and Figure 5. All of these figures deal with the STFT representation of five pure tones (sine waves) at 200 Hz, 250 Hz, 500 Hz, 700 Hz, and 800 Hz each being ramped up in frequency by 100 Hz over the period of 4000 samples at a sampling rate of 44.1 kHz. The first four tones all have equal magnitude and the fifth tone at 800 Hz has double magnitude. The FFT's were taken every 64 samples and windowed with a 1024-tap Kaiser window with 80dB drop off. The ideal spectrogram in Figure 3 depicts a simple linear summation of the individual STFT *magnitudes* of the four pure tones. The actual spectrogram depicted in Figure 4, on the other hand, accounts for the nonlinear cross term effects caused by the summation of phase. In this example it can be seen that the cross term effects oscillate more rapidly in some regions (see 400 Hz - 900 Hz) than others (see 100 Hz - 300 Hz). Even in the region of slowest oscillation, the period of oscillation is roughly 1000 samples -- or one time window. With so much variation during one time window, even a small amount of mis-

[4] Compression schemes such as the ISO/MPEG Audio Coder compress different subbands separately and therefore produce artifacts that are localized in frequency.

synchronization will result in a significant difference error in the spectrogram. The difference error resulting from a 256-sample phase shift is demonstrated in Figure 5. As can be seen in the figure, the error caused by such a shift is present all throughout the spectrogram and reaches as high as 60% the magnitude of the original spectrogram itself.

Significant error due to slight missynchronization (as shown in Figure 5) is the major drawback to using the STFT for direct comparison of the incoming signal with the original template signal. This is a direct result of the presence of cross terms which are present not only in the STFT representation but in other time-frequency representations such as the Wigner distribution and wavelet transform as well. (Kadambe and Boudreaux-Bartels, 1992)

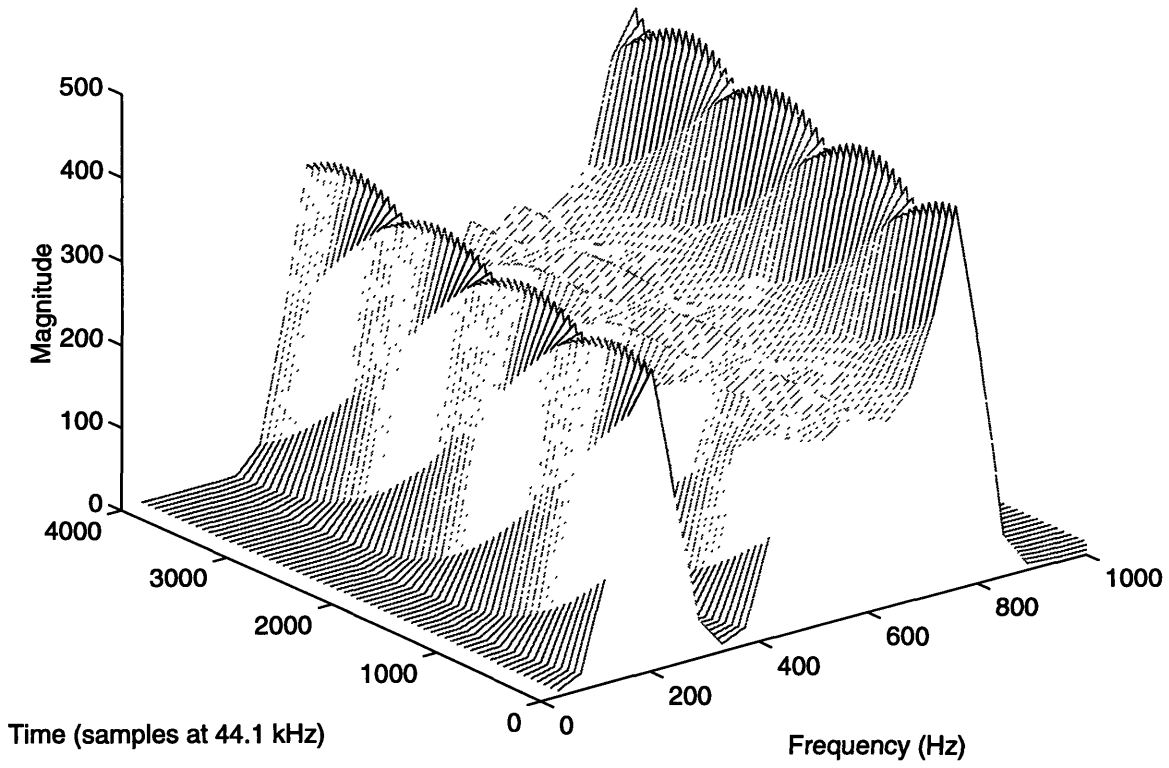

Figure 3: Ideal Spectrogram without Cross Terms

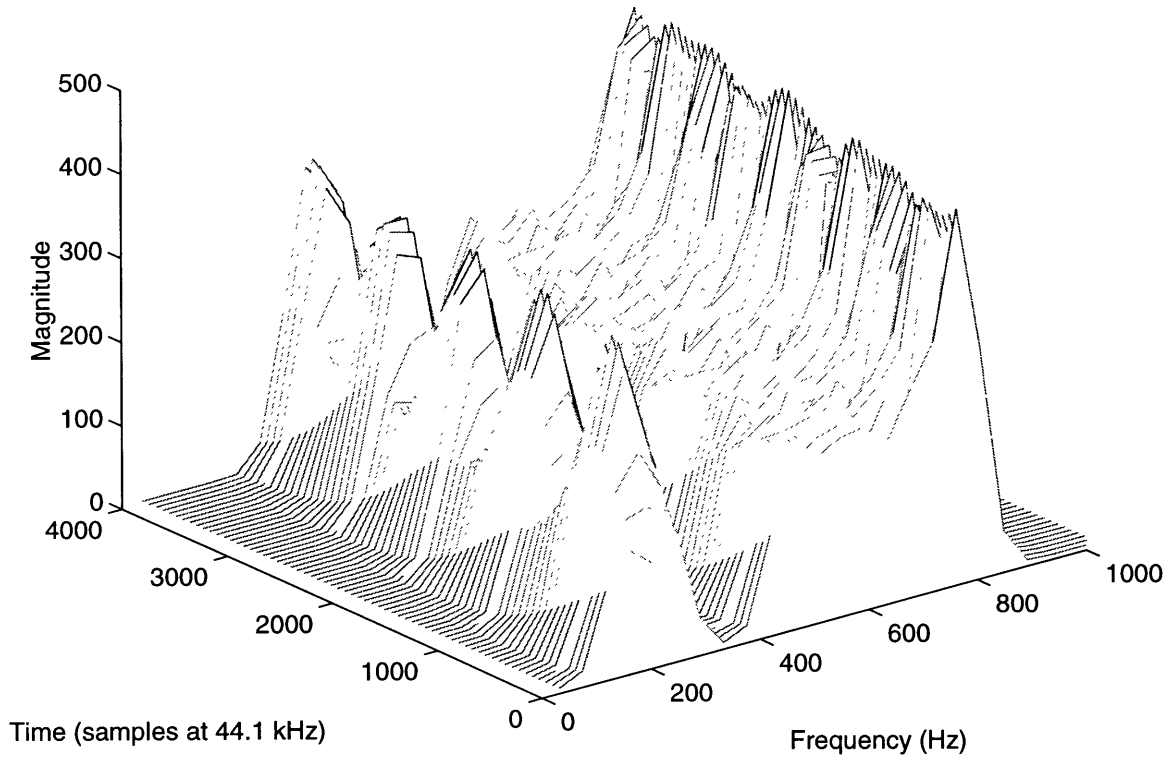

Figure 4: Actual Spectrogram including Cross Terms

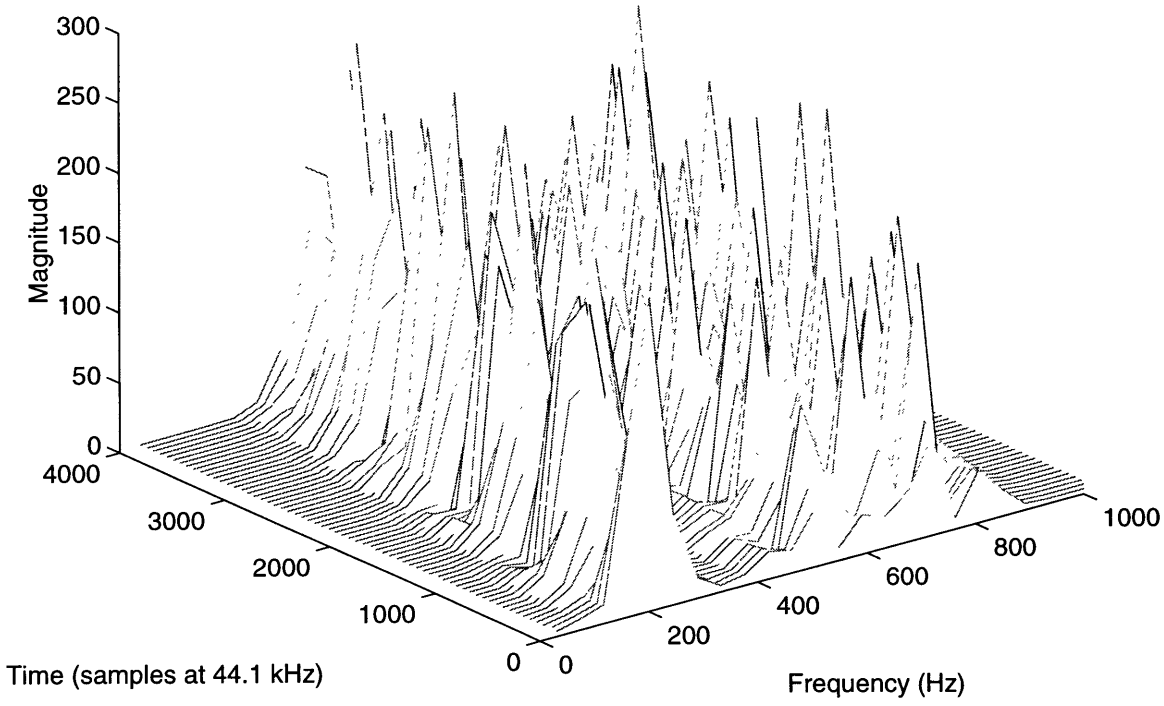

Figure 5: Error caused by 256-sample missynchronization delay

4. Wide Contribution Signatures

4.1 Motivation

The limitations associated with the representations discussed above stem mainly from the fact that they are not robust when used with sample-for-sample comparison of two signals. The purely time-domain representation compares each time sample of the original template jingle with each sample of the incoming signal; the need to compare the signals so closely in this fashion makes the representation non-robust to nonlinear phase shift. Likewise, the conventional STFT representation compares each frequency sample (FFT bin) of the template jingle with the frequency samples of the incoming signal; once again, such close comparison of the signals makes the representation non-robust to artifacts of its own cross terms as well as those of transmission.

Taking in to account this fundamental limitation of the above mentioned representations, the Wide Contribution Signatures representation seeks to compare signals on the basis of global characteristics instead.

4.2 Concept

The concept behind the use of Wide Contribution Signatures (WCS) is based on two fundamental premises:

- First of all, it is assumed that the global characteristics of a signal are relatively unaffected by localized perturbation. Use of a global characteristic for comparison thus affords robustness, but only at the expense of loss of discrimination. This is to say that the robustness is gained as a result of using a test that has a broad region of acceptance, i.e., a large set of possible signals may exhibit the same global characteristic and satisfy the comparison test.

- The second premise behind the concept of the WCS representation is that successful signal discrimination can be achieved by a battery of such broad-acceptance criterion given that they are *inherently different*[5]. As stated above, since each criterion is satisfied by comparison of a global characteristic, each criterion therefore has a broad region of acceptance. If each criterion is based on a global characteristic that is inherently different from the others, then the overlap between a set of such criterion will be relatively small.

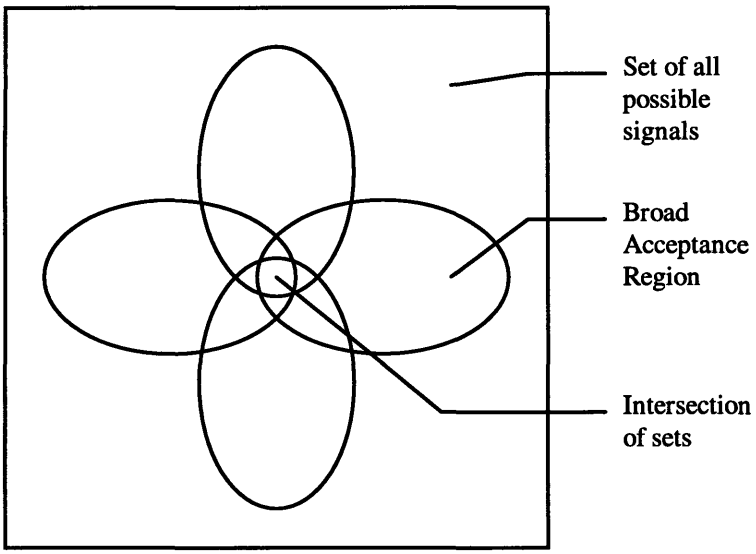

Figure 6: Wide Contribution Signatures concept

The Wide Contribution Signature representation may thus be capable of satisfactory signal discrimination while maintaining overall robustness since each test that the signal must satisfy is in itself quite robust.

4.3 Selection of Signatures

As stated above, the WCS representation requires that each signature be representative of contribution from all components of the signal as well as inherently different from the other

[5] *Inherent difference* is subject to interpretation, but is discussed with reference to this project in Section

signatures. Different types of signatures may be considered inherently different if they are computed from completely different types of representations of a signal (e.g., time domain and spectrogram representations) and/or from completely independent attributes of those representations. In order to meet the system requirement of real time as well as the design goal of minimal computational complexity, the signatures must also be computationally easy to compute as well as to compare. The four signatures selected are the volume envelope, center frequency, frequency spread, and zero-crossings.

4.3.1 Volume Envelope

The volume envelope is found by taking an average amplitude of the signal over a sliding window in the time domain. Since it is found from the time-domain representation of the signal, it takes into account all components of the signal (see Section 3.1); but since it is a measure of the average amplitude over the duration of an averaging window, it is unaffected by alteration of phase or localized time-domain artifacts.

The volume envelope is computed as the root mean squared amplitude of the signal over the duration of a rectangular time window. The duration of this time window is equivalent to the chunk duration (see Section 4.4.1). The chunk duration therefore is directly correlated with the sensitivity versus robustness tradeoff of the volume envelope signature. A shorter duration results in a faster changing, more sensitive signature; whereas a longer duration will be produce a slower changing but more robust signature.

After breaking up the volume envelope signature into its quickly changing "AC" and slowly changing "DC" components (see Section 4.4.2), the AC component may be normalized by the DC

4.3.

component. This will provide a signature that can be used for comparing the *shape* of the volume envelope rather than its magnitude. The resulting volume envelope signature will be immune to differences in overall volume and will facilitate comparison by means of a simple sliding subtraction.

4.3.2 Center Frequency

The center frequency as a function of time is found from the STFT of the signal. Since the center frequency is computed by taking into account the magnitudes of all of the frequency bins of the FFT, it avoids the susceptibility to localized artifacts found when using the conventional STFT (see Section 3.2.2). Instead, it represents a significant global characteristic of the signal that is both dynamic and robust.

Treating the normalized spectrum as a probability distribution, the center frequency can be found either by computing the mean frequency or the median frequency. Since the mean weighs frequency components near the ends of the spectrum more than those near the center, it will give unwanted weight to high frequency noise. The median, on the other hand, weighs all frequency components equally and therefore gives a better estimation of the center frequency.

The median is also computationally relatively simple. First, the amplitudes of all of the frequency bins must be summed. The median can then be found by taking the successive sum of all frequency bins until it reaches half the sum of the entire spectrum. Assuming that the center frequency is generally low in comparison to the 22 kHz bandwidth, only a sixteenth to an eighth of the bin amplitudes will have to be summed before the median frequency is found.

Since the result of the center frequency computation is a frequency without any information about magnitude or phase, it is in a way already "normalized" to the frequency scale of the STFT. Still, there are advantages to breaking the center frequency signature into its AC and DC components

(see Section 4.4.2). By ignoring the DC component and looking only at the AC component of the signature, recognition is robust to the addition of "DC-offsets" such as may arise from the addition of a stationary pure tone to the incoming signal.[6] Since the center frequency is inherently "normalized" as mentioned above, the AC center frequency signatures of the template jingle and incoming signal can then be compared by a simple sliding subtraction.

4.3.3 Frequency Spread

Our treatment of the normalized spectrum as a probability distribution function is incomplete without the second most important attribute of a PDF, a measure of the spread. Conventionally, this would be computed as the standard deviation of the PDF, but this will give more weight to the ends of the spectrum as compared to the center. The frequency spread will instead be computed by finding the distance between the medians of the two halves of the spectrum on either side of the center frequency.

In order to increase its robustness to noise, the DC component of this signature may be ignored. Looking at only the AC component will ignore any "DC-offset" which may occur due to noise without losing information regarding the *shape* of the signature. Since the spread is measured as a distance with reference to the frequency axis, it is inherently "normalized" and can easily be compared by means of sliding subtraction.

4.3.4 Zero Crossings

The zero-crossings as a function of time is a signal characteristic that has been found to be useful in pitch detection of speech signals. (Hess, 1983) Like the volume envelope, this characteristic is found over the duration of a window sliding over the time domain representation of the signal. As

[6] One such pure tone, for example, may be observed at 19 kHz in recordings from the radio as an artifact of the FM stereo carrier signal.

with the volume envelope, this makes it immune to effects of phase alteration and localized time-domain artifacts.

The zero-crossings are counted over the duration of a sliding time window. Since any count of zero-crossings will obviously be very prone to error due to noise in the signal, zero crossings are only counted if they surpass a noise threshold 40dB below the maximum recorded volume.

Since the number zero-crossings would undergo a "DC-offset" in the presence of either a pure tone or above-threshold random noise, improved robustness could be achieved by simply ignoring the DC component of the zero-crossing signature. Instead, only the AC component would be used for comparison. Since zero-crossings are counted on an absolute scale, the AC component is on the same scale and can be compared between the template jingle and incoming signal by means of a simple sliding subtraction.

4.4 Implementation Issues

Although the basic computational and comparison issues have been detailed above, there are a few other issues which must be addressed for practical implementation of the system.

4.4.1 Segmentation of Signals

The jingle and its template signatures can be thought of as being broken down into three levels of time duration. The largest of these is, of course, the entire jingle. The jingle can then be further broken down into "chunks" and then "time frames". Time frames are the length of time used to create each new signature sample, i.e., it may be on the order of 512 or 1024 samples such as an STFT window. Time frames are then grouped into chunks which may have duration on the order of a few seconds.

The segmentation of the signal into multiple levels is motivated by the compromise between confidence and robustness. Since longer audio segments take into account more information about the signal, use of these longer segments is less likely to result in a false positive match. In this way, by incorporating more *context* (i.e., behavior of the signal over an extended period of time), we can have more confidence in the resulting match/no-match decision.

The tradeoff of confidence in intermediate results may be a costly one in trying to recognize an entire jingle. If the jingle is broken down into small segments, then each of those segments must be matched one after another for a positive recognition of the jingle. If the first segment incorrectly results in a match decision, then hopefully the second segment would correctly declare no-match. If not the second, then hopefully the third, and so on. By introducing more room for error in intermediate match/no-match decisions, the possibility of prematurely progressing down a list of template segments is increased. This may lead the recognition algorithm on a "wild goose chase" trying to match the later segments of the template jingle while the first segments of the actual jingle may be just starting to come in.

Matching a longer segment, on the other hand, also requires a more exact match between the incoming jingle and the implicit assumptions of the template segment. The main implicit assumption is, namely, about the playback speed. Since it is unlikely that the *exact* playback speed of the incoming signal will be chosen for the template, the only way that a true positive match can occur is if the segment duration is short enough such that excessive time lag does not occur between the incoming signal segment and the template segment. Thus, a longer template segment, even if its speed is chosen close to the actual speed of the input signal, will be more likely to produce a false no-match decision.

It is for these reasons that a multilevel segmentation scheme is used for each jingle. The time-frame duration is too short to provide enough intermediate confidence. The jingle duration is too

long to provide robustness to different playback speeds, even if a close guess is made. It is therefore necessary to maintain a third level of duration, the chunk level, so as to establish a compromise between robustness and confidence.

4.4.2 AC/DC Analysis

As mentioned in Sections 4.3.1 through 4.3.4, there are many benefits to separating the quickly varying "AC" component of a signal from the slowly varying "DC" component of the signal. The important question which arises when implementing such an AC-DC separation is from which signature samples to compute the DC value for normalization. To demonstrate its practical relevance, this question will be addressed with respect to the normalization of the volume envelope signature.

If every sample amplitude in the window were normalized by one average DC value calculated only for that window, then the resulting volume envelope signature could be heavily dependent on exact chunk window synchronization between the template and incoming jingles. If, instead, each AC amplitudes were normalized by the average of a fixed number of signature amplitudes preceding it, then chunk window synchronization would not matter; the AC amplitudes of the template and incoming jingles would look the same despite window synchronization issues because each sample in incoming jingle would have been normalized by the same corresponding preceding signature samples in the jingle. As shown in Figure 7, since there are no signature samples (except silence) preceding the beginning of the template jingle and because unknown audio will precede the incoming jingle, each signature sample in the first chunk should be normalized by an average of a fixed number of signature samples *following* that sample.[7] In this way, if the jingle is in fact

[7] Likewise, each signature sample in the last chunk should be normalized by an average of the samples *preceding* that sample.

contained within the incoming signal, it will automatically be used to correctly normalize its volume envelope itself.

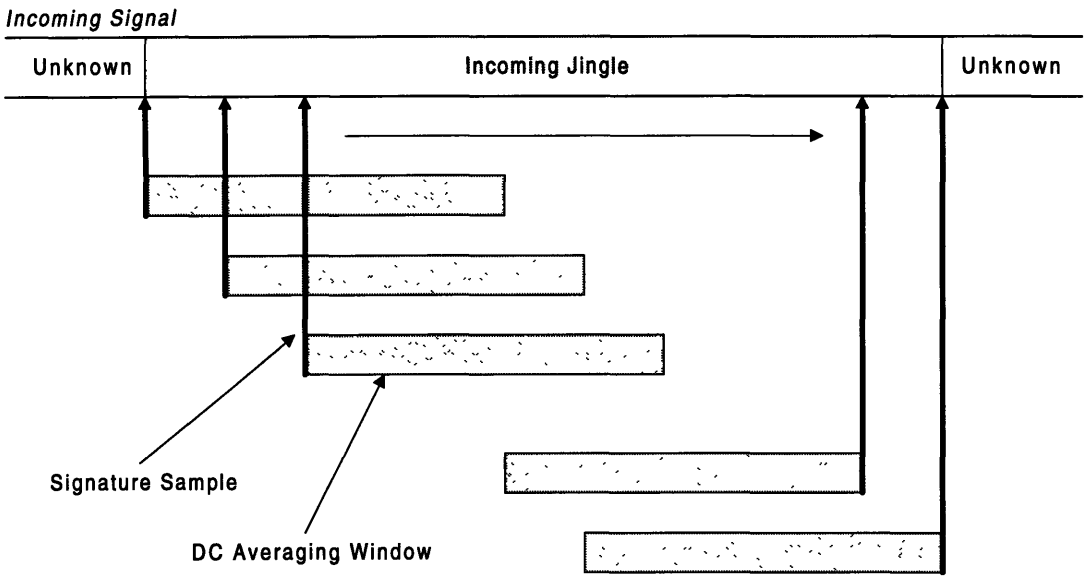

Figure 7: Placement of DC Averaging Window

Since the signal is normalized by the slowly varying (DC) component of the volume envelope, that DC information is lost, but the dynamic (AC) information of the signature is still intact. In addition, since the AC information has already been normalized, it can be compared between the template and incoming signatures by simple subtraction. Thus, by simply computing a sliding subtraction between the template and incoming volume envelope signatures, the presence of a specific chunk can be detected by searching for the minimized difference.

4.4.3 Playback Speed Synchronization

In the normal case, signatures are synchronized with each other by means of maintaining equivalent window lengths. For instance, the time windows employed by the volume envelope and zero-crossing signatures should be equivalent in length both to each other as well as to the STFT window used for finding the center frequency and spread signatures. In this way, all signatures will be found concurrently from the same portion of the signal, and each signature produces one signature sample per time frame.

In order to be able to successfully recognize an incoming jingle that has been speed shifted, it must be compared with a template that accounts for the approximate amount of speed shift. Given a limited range of possible speed shift (±10% as stated in section 2.1), a finite set of templates can be created to handle a set of different amounts of speed shift; if neighboring speeds are close enough together, then the intermediate speeds may also be successfully identified.

As shown in Figure 8, since each WCS is essentially a one dimensional function of time, it can be simply rescaled (along one or both axes) to represent a speed-shifted version of the original template jingle. This saves the complication of having to first resample the original jingle and then produce new sets of signatures for each different playback speed. Instead, for instance, the original volume envelope signature can be rescaled along the time axis to represent any desired amount of playback speed shift. Although all of the signatures must rescaled along the time axis, the center frequency, frequency spread, and zero-crossings must be rescaled along the y-axis as well. When the playback speed is increased, the frequencies of all signal components increase proportionally as do the number of zero-crossings. For this reason, once the time axis is rescaled, the values of all three of the above mentioned WCS's should be proportionally rescaled.

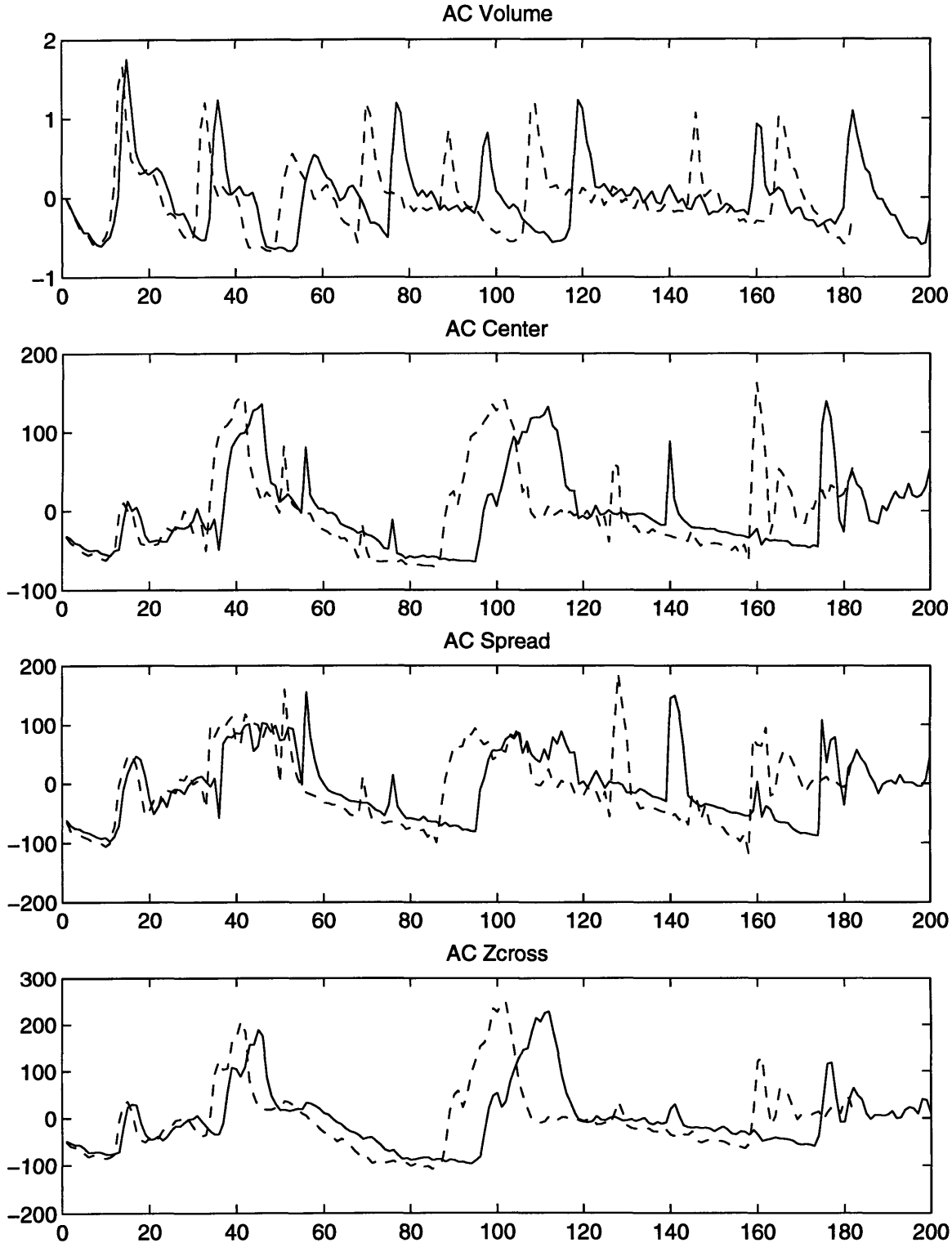

Figure 8: Rescaling of Signatures

4.4.4 Limiting Dynamic Range

Although the original jingle recording will have a great deal of dynamic range, the incoming signal will be noisy and will thus effectively have a lesser dynamic range. For this reason, it is necessary to take the SNR noise floor specification (40dB, in this case -- see Section 2.1) into account in the comparison of signatures. More specifically, signature samples should not be considered for comparison if they coincide with time frames where the volume envelope of the template falls below the noise floor specification. This will avoid consideration of signature samples that are likely to be skewed by noise, therefore introducing less error into the signature comparisons.

5. Implementation

The complete process of jingle detection is implemented in two stages as described in Figure 9. Having been given an original recording of the jingle to be detected, the first stage involves the preprocessing of the signal to prepare the system for real-time detection. The second stage involves use of the information produced from the first stage in conjunction with information about the incoming signal in order to detect the separate chunks of the jingle until the entire jingle is finally detected.

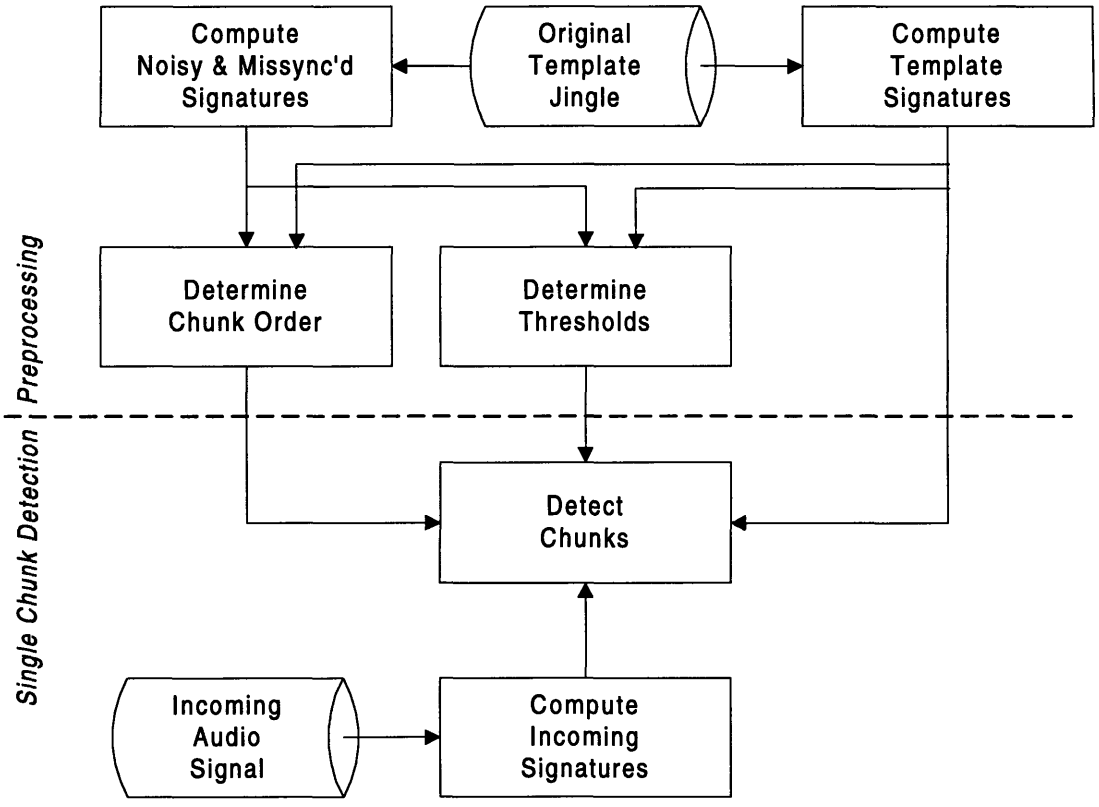

Figure 9: Entire Jingle Detection Algorithm

5.1 Preprocessing

The first stage of the system involves preprocessing of the given jingle. The purpose of this is to produce information that will aid detection of both separate chunks as well as the entire jingle. As outlined in Figure 9, this first involves signature formation for both the original template jingle as well as a noisy & missynchronized version of the jingle. The signatures are then used together to determine the desired order in which the chunks will be detected and the comparison thresholds for detection of each chunk.

5.1.1 Signature Formation

Formation of the signatures for both the original template jingle as well as an artificially noisy & missynchronized version of the template jingle are integral to the preprocessing stage of jingle detection. Computation of both sets of signatures is performed as described in Sections 4.3.1-4.3.4. This section addresses specifics regarding segmentation as well as the creation of the noisy & missynchronized (N&M) signal.

Segmentation

A general discussion regarding segmentation issues can be found in Section 4.4.1; this section discusses the specific choices made for actual implementation. The time frame is chosen to be 1024 samples in duration with 512 samples of overlap between successive frames. The chunk length is chosen to be 200 time frames.

The time frame length is equivalent to the time window length of the signatures and is therefore chosen according its utility to the computation of the signatures. A window duration of 1024 samples (at 44.1 kHz) means that the center frequency and frequency spread signatures will be based on 1024-pt FFT's providing good resolution. The 23 ms duration of the time window is also

long enough such that the volume envelope and zero-crossings signatures are robust to the individual oscillations of the waveform whereas the duration is short enough such that the signatures are sensitive to global change in the signal. Each time frame marks the passage of one such 1024-sample time window overlapped by 512 samples each time. Therefore a new time frame occurs once every 512 samples.

The chunk length is chosen to be 200 time-frames. Given a time frame sampling frequency of once every 512 samples (at 44.1 kHz), the duration of each chunk is therefore 2.3 seconds. While the choice of chunk length is somewhat arbitrary, it should remain on the order of a few seconds to provide context and consequently intermediate confidence as discussed in Section 4.4.1. The chunk length must also not be so great that the maximum possible speed difference between the incoming signal and pre-speed-shifted template will result in excessive time lag. Comparing the incoming signature chunks with template chunks pre-shifted in intervals of 2% (see Section 5.2.1), the maximum possible speed difference will be 1%; in the case of a 200 time-frame chunk length, this translates to a maximum time lag of only 2 time frames. Thus, the choice of a 200 time-frame chunk length provides intermediate confidence at the expense of minimal template signature time lag in handling speed-shifted incoming signals.

Noisy & Missynchronized Signatures

The creation of an artificially predistorted signal is necessary for the experimental determination of effects due to noisiness and missynchronization of the incoming signal. These effects are systematically used in the preprocessing stage to determine the ordering and thresholding of the chunks of the jingle.

The first effect, noisiness, is added to original jingle recording in accordance with the system SNR/noise floor specifications for the incoming signal. As mentioned in Section 2.1, the SNR is

specified to be 40 dB. Therefore random noise is added to the signal with a maximum possible amplitude 40 dB less than the maximum amplitude of the signal. In cases of extraordinarily quiet jingles, this maximum noise amplitude may require manual adjustment.

The second effect, missynchronization, refers to the incoming signatures being computed from a set of time frames that are shifted (by delay) with respect to the template time frames by a fraction of a time frame. Missynchronization occurs, for example, when the template signatures are computed from time windows at sample 0, 512, 1024, 1536, etc. of the jingle, whereas the incoming signatures are computed from sample 256, 768, 1280, 1792, etc. of the jingle. Since such missynchronization is likely to occur in practice, the artificially predistorted signal must take this into account by shifting synchronization by half a time frame. Therefore, given a time frame duration of 512 samples, the signal will be shifted 256 samples before computation of the signatures.

Once the original jingle has been artificially predistorted by the deliberate addition of noise and missynchronization (delay), a new set of signatures can be computed in the same manner as for the original jingle.

5.1.2 Ordering

The order in which the chunks are detected is of great importance to success of detecting the entire jingle. This is related to the "wild goose chase" phenomenon mentioned briefly in Section 4.4.1. In essence, the problem is this: If in attempting to detect the first chunk a false positive match occurs, then the system will move on to the second chunk. If yet another false positive match is to occur, then the system will incorrectly move on to the third chunk as well. In theory, eventually one or more of the chunks will return a true negative match therefore indicating that the incoming signal is not actually the jingle. But if the true first chunk were to actually occur within the incoming signal

during the time of the "wild goose chase", then it would not be noticed by the system because the system would already be trying to detect the second or third chunk. By the time the system would find that the second, third, or later chunks did not match, it would return to waiting for the first chunk only to have missed its earlier occurrence. Thus, the "wild goose chase" will have distracted the system from ever detecting the actual incoming jingle.

In order to avoid falsely progressing into the chunk search sequence, the most *unique* chunks should be searched for first since they will be less likely to result in false positive matches. Uniqueness is defined as the ability of a chunk to stand out and be easily discriminated from other signals. Although uniqueness really means the ability to stand out from *all* other possible signals, such uniqueness is obviously impossible to determine experimentally. Instead it is assumed that, of the set of all possible signals, the signals with which any chunk may get most easily confused come from other parts of the same jingle. Therefore, the best way to rate each chunk's uniqueness is to determine how it stands out from the rest of its specific jingle. (See "Uniqueness" later in this section).

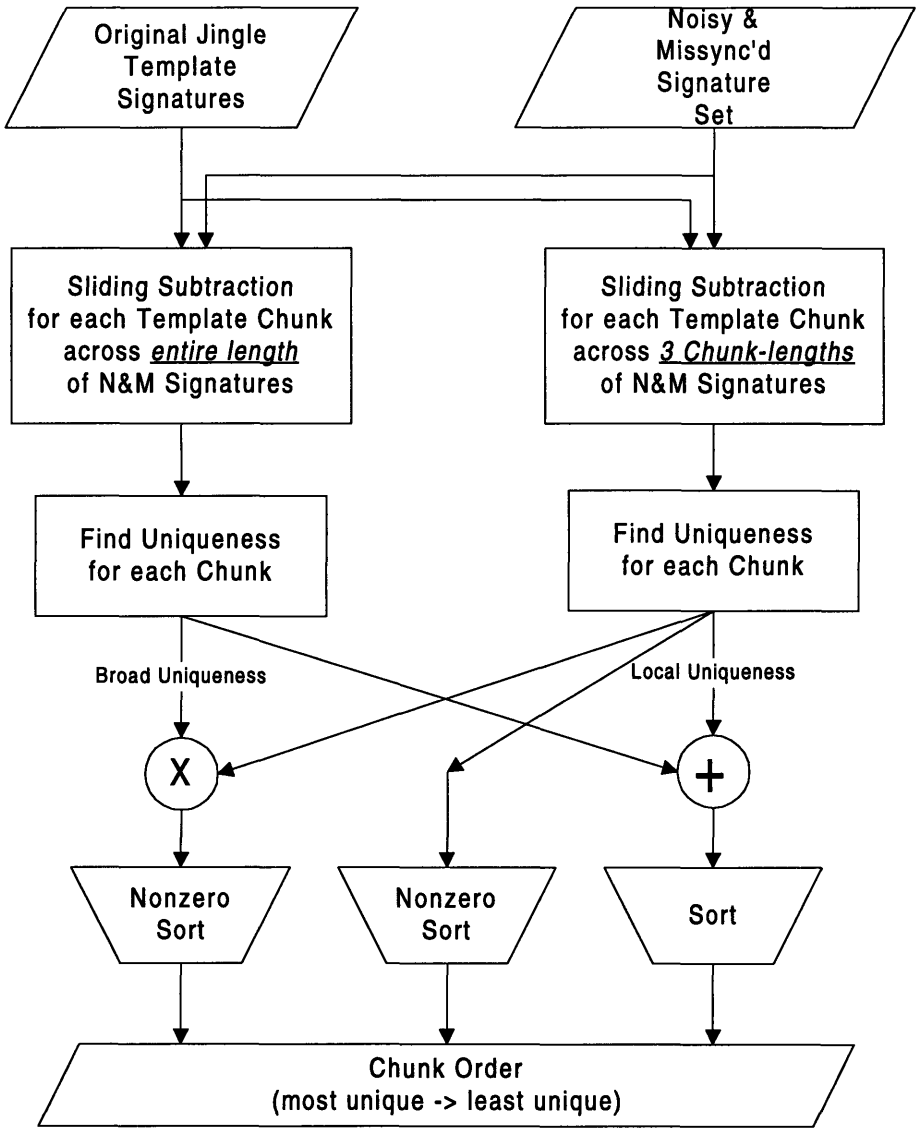

Figure 10: Chunk-Ordering Algorithm

The algorithm for ordering is described in Figure 10. First, sliding subtractions between the original template signatures and the noisy & missynchronized (N&M) signatures are fed into the uniqueness-finding algorithm described in the "Uniqueness" subsection later in this section. As depicted in Figure 11, sliding subtractions between each template chunk and the *entire* N&M

signature set are used for determining broad uniquenesses; whereas sliding subtractions between each template chunk and only three N&M chunk-lengths (the corresponding N&M chunk plus its two surrounding chunks) are used for determining the local uniquenesses. The chunks are then ordered according to their broad and local uniquenesses. First, the chunks that have both nonzero broad uniqueness and nonzero local uniqueness are sorted by the product of the two uniquenesses. Secondly, those chunks that had only nonzero local uniquenesses are then sorted by that uniqueness. Lastly, any remaining chunks that have not already been ordered will be ordered by sum of both uniquenesses.

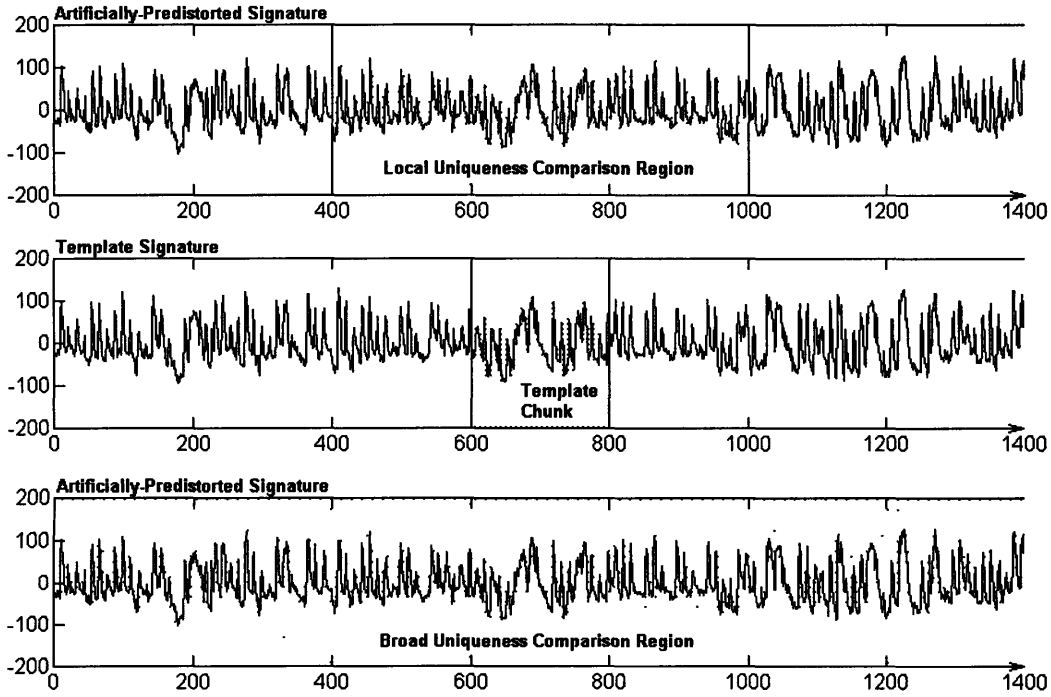

Figure 11: Comparison Regions for Local and Broad Uniqueness

Once the order has been established during pre-processing, the system can later selectively jump between the successively ordered chunks as long as the incoming signal signatures are stored in a continuous buffer for random access. In order to account for the finite length of the continuous

buffer, the chunk-ordering algorithm must only order one buffer-length of the template chunks at a time. (See Section 5.3.1)

Uniqueness

When the signatures of a template chunk are slid across and subtracted from the N&M signatures, the error/difference between the two (for any given signature) should be minimized at the point at which the template chunk has slid into place over the corresponding portion of the N&M signature (See Figure 12). Likewise, wherever the N&M signature is similar -- although not necessarily identical -- to the template chunk signature, there will also be another local minimum in the error. The greater the difference between the absolute minimum error and the other minima, the more easily the chunk in question can be discriminated. Therefore, the uniquess for a given WCS is measured as the difference in error between the absolute minimum error and the next lowest local minimum. If, however, the absolute minimum error produced from the sliding subtraction is not in the correct location, i.e., one of the other local minima is lower, then the uniquess is by definition

zero (see

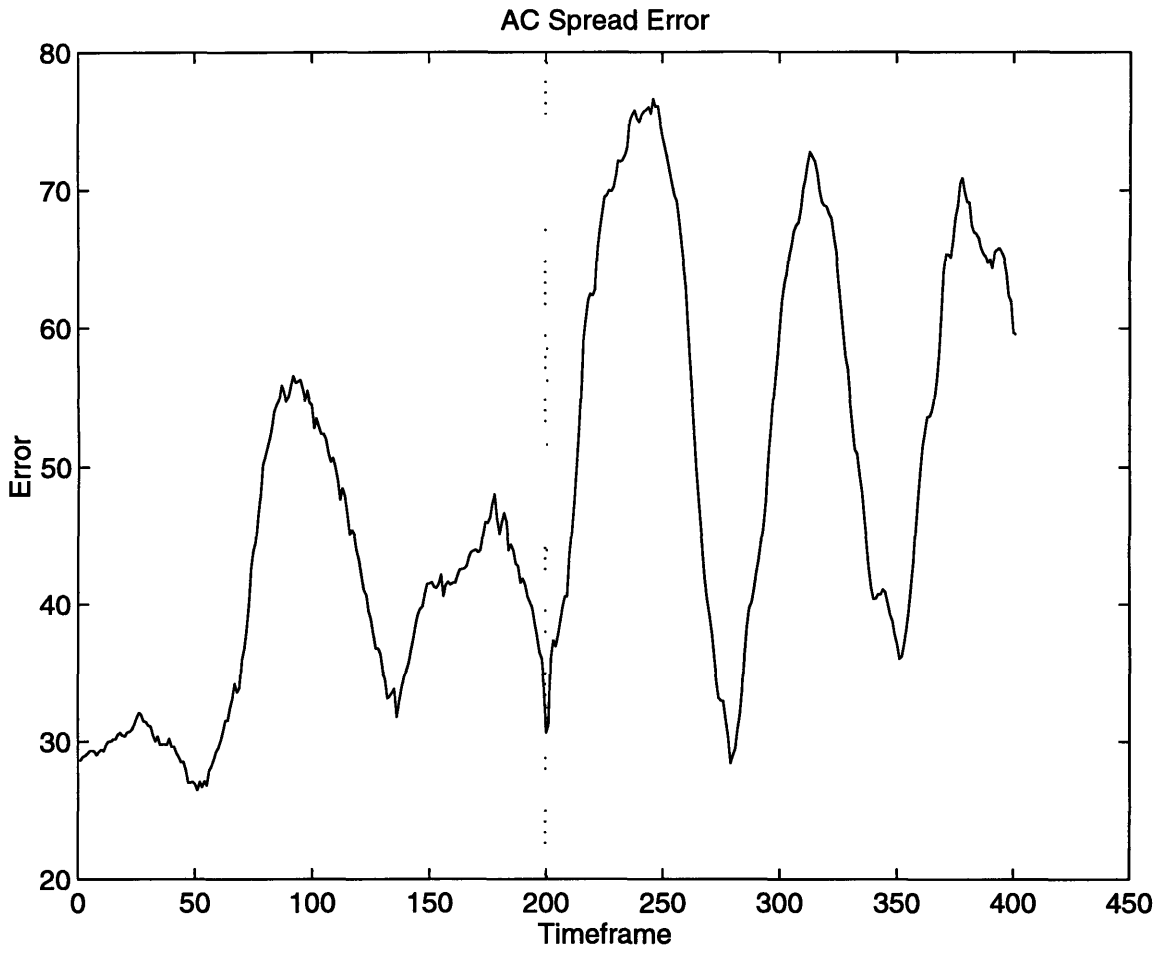

Figure 13). The composite uniqueness is the equal to the product of the four uniquenesses for the different WCS's.

Since the uniqueness is computed directly from the errors produced by the sliding subtractions, the inputs to these sliding subtractions determine the scope of the uniqueness. If the sliding subtraction is performed between a template chunk and the entire N&M signature set, then the resulting uniqueness will be with respect to the whole jingle; this is called broad uniqueness. Local uniqueness, on the other hand, computes sliding subtractions only between the template chunk and the region of the N&M signature set spanning from one chunk-duration before to one chunk-duration after the corresponding location of that chunk on the N&M signature set. (See Figure 11)

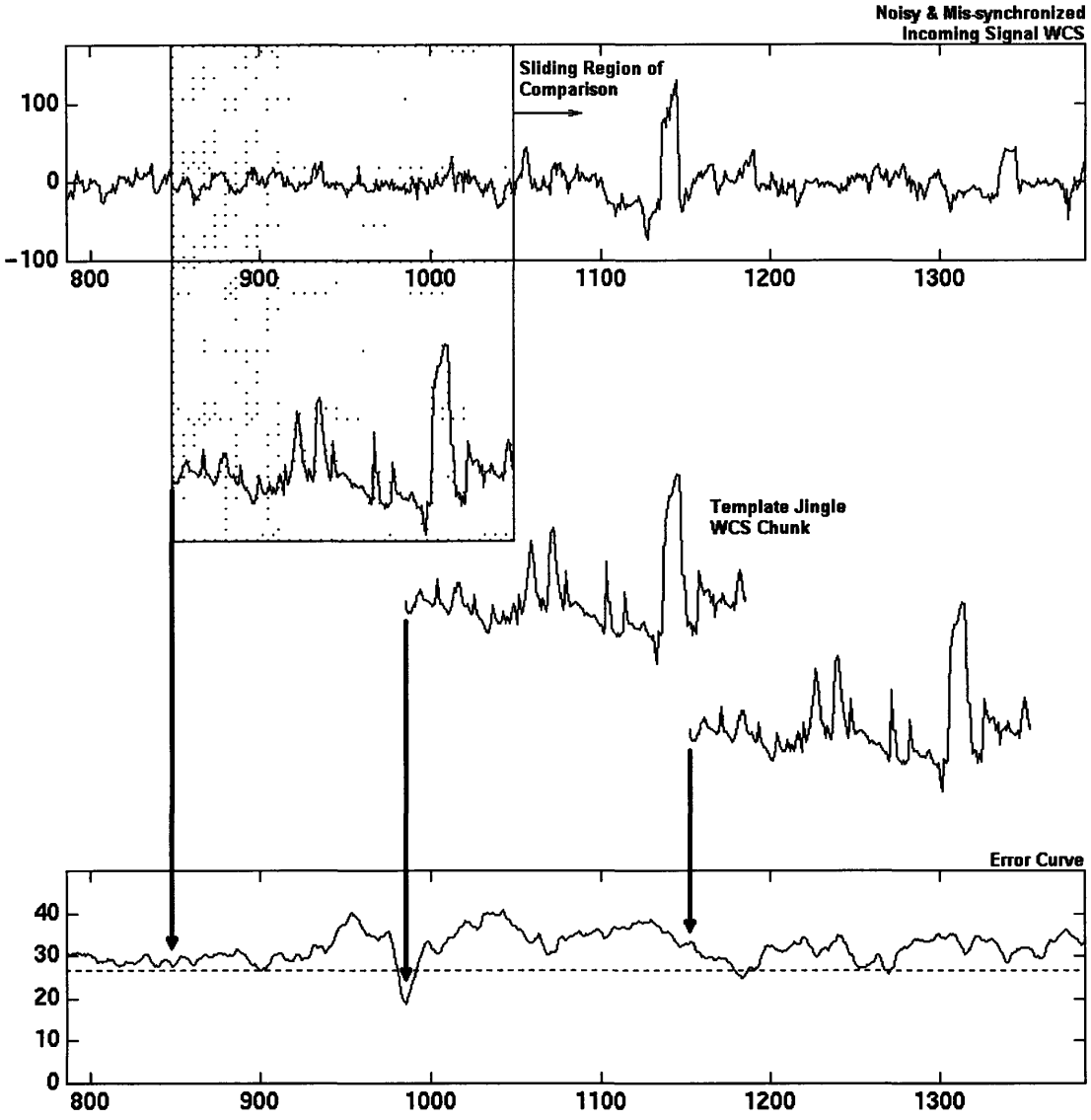

Figure 12: Sliding Subtraction and Resulting Error Curve

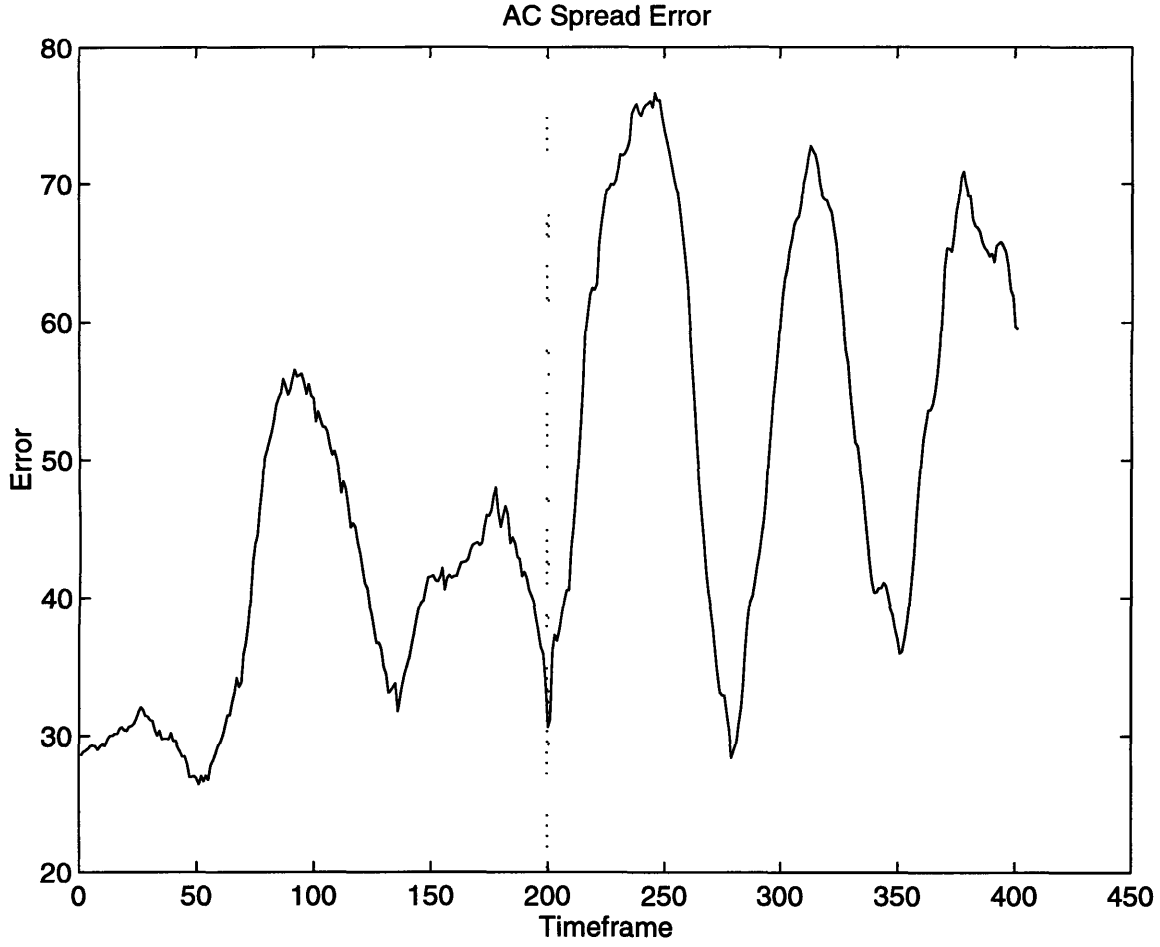

Figure 13: Zero Uniqueness Error Curve

Although there is a local minimum at the correct location of timeframe 200, it is not the absolute minimum. The chunk in question is therefore considered not unique.

5.1.3 Thresholding

As the final part of the system's pre-processing stage, the error thresholds must be determined for detection of each individual chunk. The threshold for each signature of each chunk is determined by the value of the Nth lowest point in the local sliding subtraction error curves. In the case that the lowest point in the error curve is not at the correct location (i.e., the case of zero uniqueness), the value of the Nth point greater than the correct local minimum is used as the threshold. The fixed number of points N essentially determines the size of the broad acceptance regions in Figure 6.

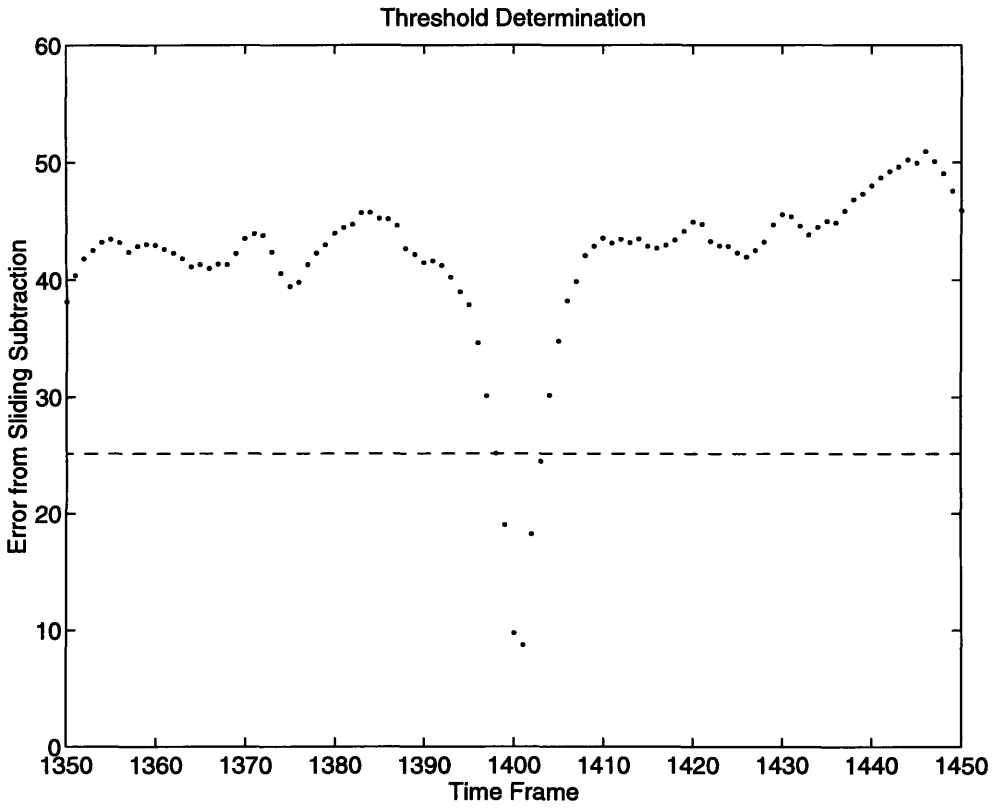

Figure 14: Threshold Determination from Error Curve (for N = 5)

5.2 Single Chunk Detection

Detection of a single chunk of the jingle within the incoming signal requires only a few basic steps (see Figure 15). First, the desired chunk signatures must be extracted from the template according to the next chunk in the chunk order. Once the appropriate chunk signatures are extracted, the volume signature is used in conjunction with the incoming signal signatures to approximate the possible speed. The speed hypothesis is then used to rescale the template chunk signatures for comparison by sliding subtraction. After the sliding subtraction, the thresholds determined during preprocessing will be applied to the error curves to determine whether or not the chunk exists.

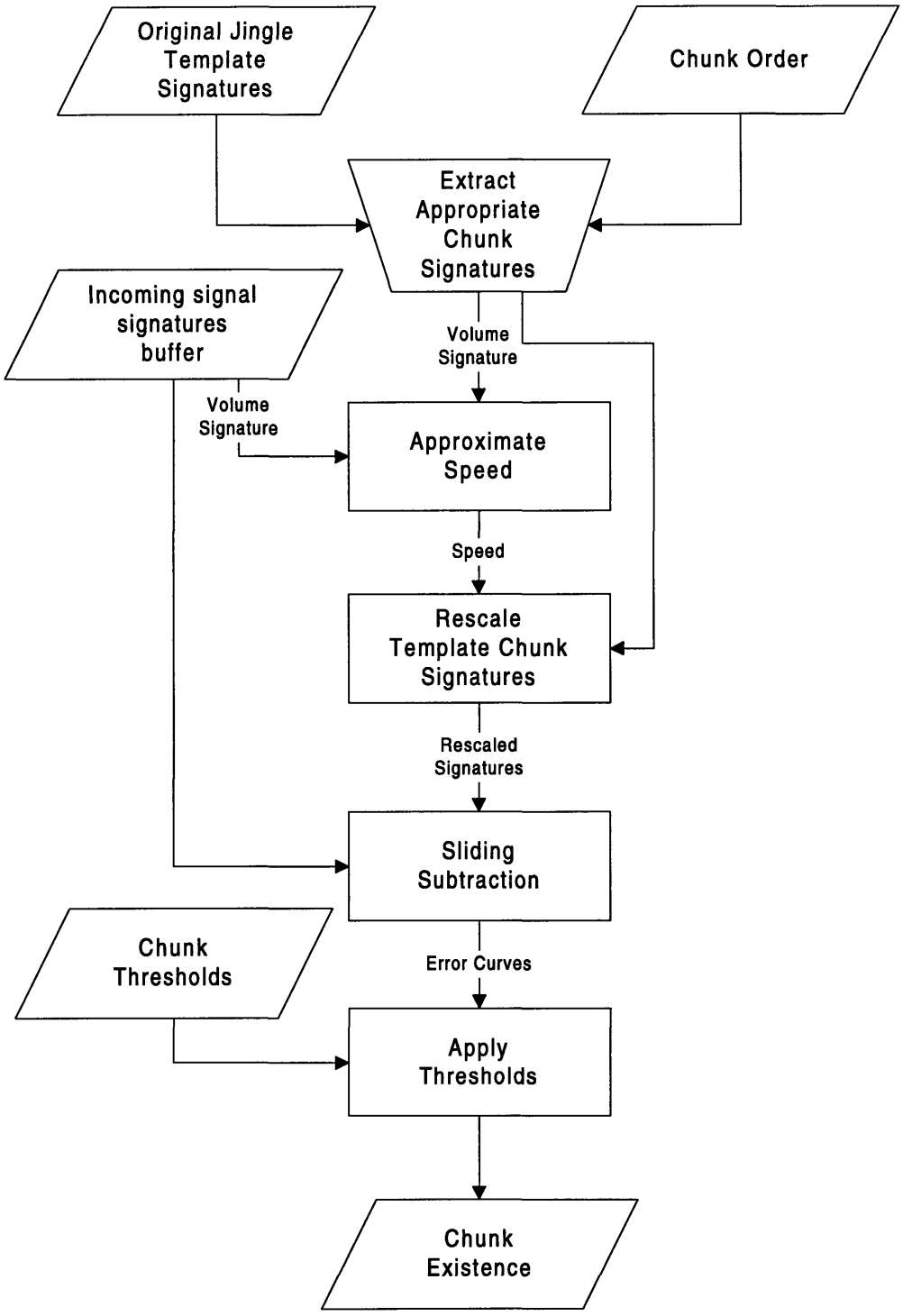

Figure 15: Single Chunk Detection Algorithm

5.2.1 Approximating Speed

The only way to guess the speed of playback is to rescale template chunk signatures for a set of different speeds while attempting to match each one with the signatures of the incoming signal. Since it is the most easily rescalable signature, the volume envelope signature is to be used for this purpose (see Section 4.3).

Given that the system must be able to handle speed shifts of ±10% (see Section 2.1), a speed approximation resolution of 2% is reasonable. Therefore, the volume envelope signature will be rescaled for every 2% between +10% and -10% speed shift, producing a total of 11 rescaled volume envelope signatures for each chunk.

Having produced the set of rescaled volume envelope signatures by the method described in Section 5.2.2, the signatures will then be compared with the volume envelope signature of the incoming signal. The speed corresponding to the signature producing the lowest absolute minimum error will be output as the best speed hypothesis.

5.2.2 Rescaling Template Chunk Signatures

Wide Contribution Signatures by definition track characteristics of the signal that do not vary drastically from time frame to time frame. Because computations are sought to be minimized in the interest of a real-time implementation, the rescaling of chunk template signatures will be performed using zero-order hold rather than interpolation of new points (see Figure 16 and Figure 17). This means that the value at each time frame of the rescaled signature will be assigned as the un-interpolated value of the nearest corresponding time-frame of the unscaled signature. By exploiting this computationally inexpensive method of rescaling, rescaling of signatures can be performed instantaneously during each iteration of the single chunk detection.

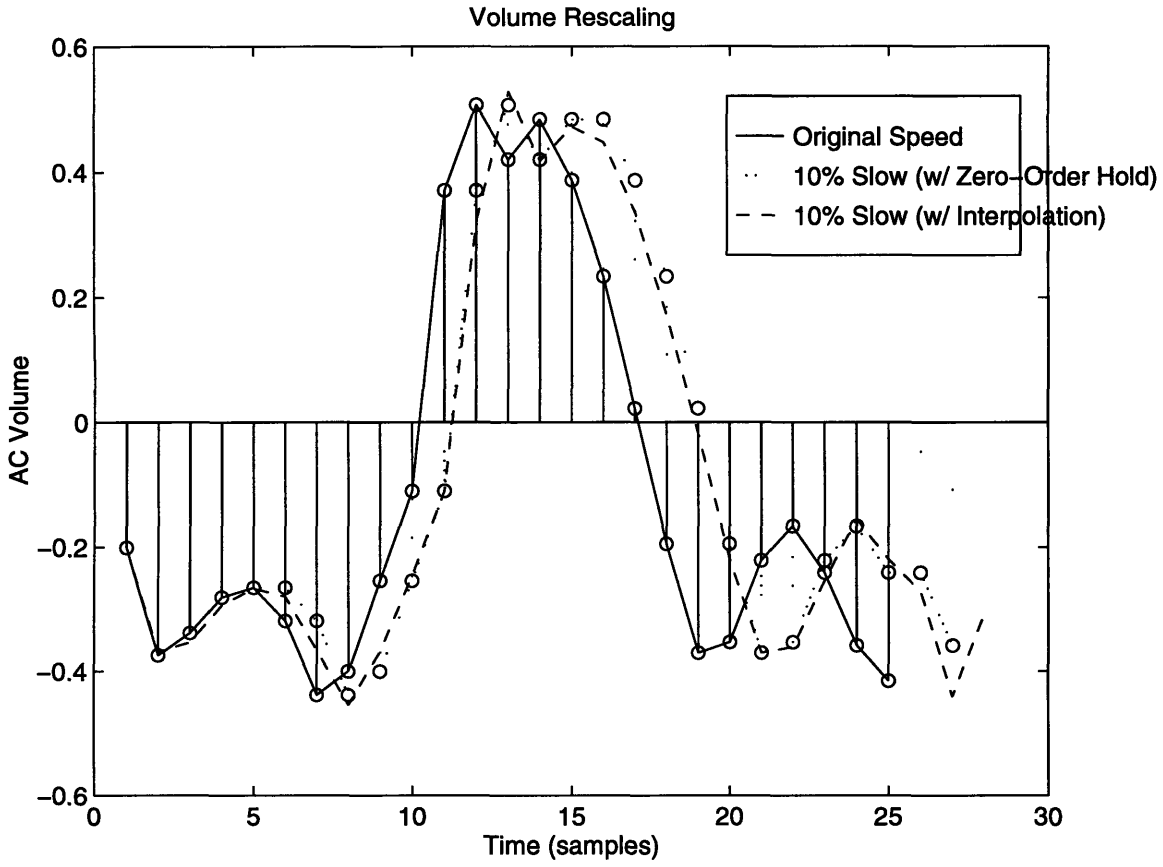

Figure 16: Time Axis Rescaling with Zero-Order Hold for Volume Signature

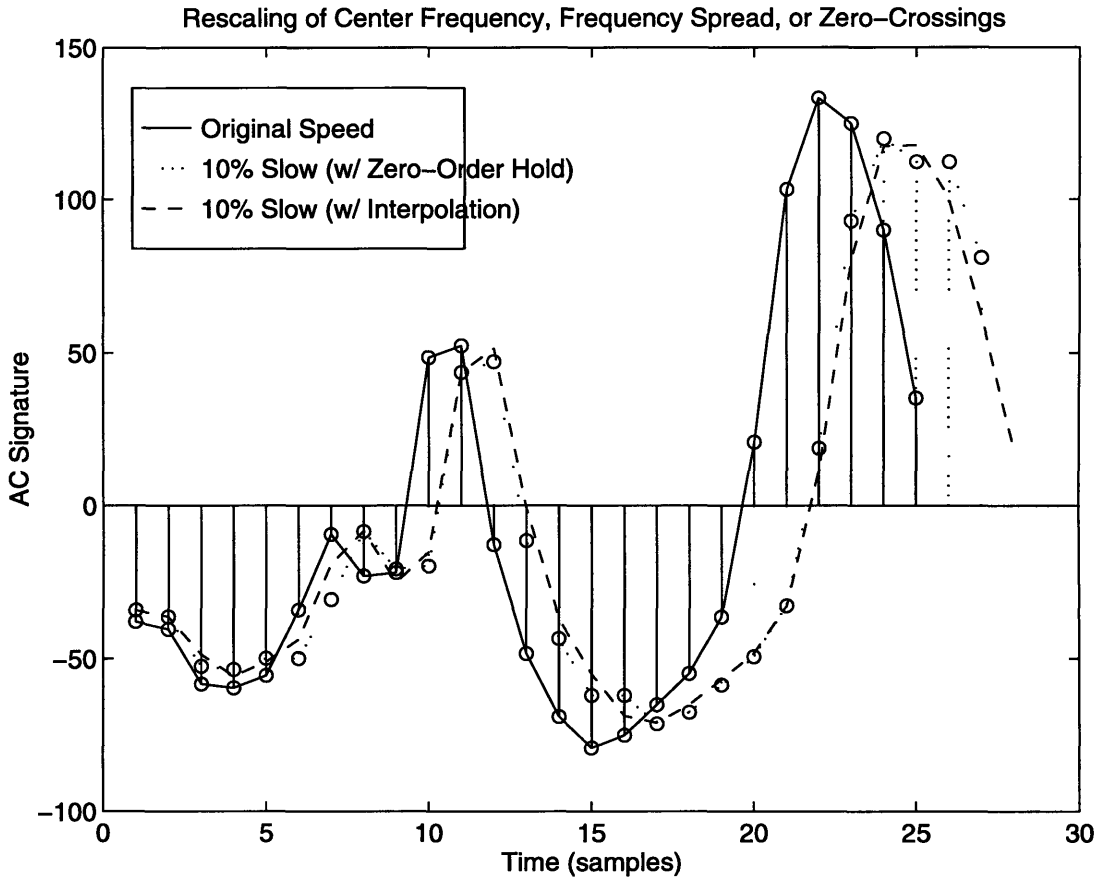

Figure 17: Time Axis Rescaling with Zero-Order Hold for Center Frequency, Frequency Spread, or Zero-Crossings

In terms of rescaling any chunk template signature for match with a given speed shift, this means that the rescaled template will be rescaled instantaneously as needed rather than storing an entire library of pre-speed-shifted template signatures.

5.2.3 Applying Thresholds

It is in the process of threshold application that the Wide Contribution Signature concept as visualized in Figure 6 is finally implemented. In the first stage of threshold application, the error curve for each signature of the given chunk is tested against the signature's corresponding precomputed threshold. Each time an error curve of a signature passes this test, it marks the

signal's acceptance into the broad acceptance region for that particular WCS. (Note that the breadth of each acceptance region has already been chosen in determination of the thresholds. See Section 5.1.3.) Given that the acceptance region of a threshold test is truly broad, a signal is likely to test positive on more than only the actual occurrence of the desired chunk. Therefore, the output of the initial threshold application to the error curve for one WCS will be a binary stream indicating where the error curve does and does not satisfy the threshold.

Since the signal must be a member of all four broad acceptance sets to be counted in the intersection of sets, a logical AND must be performed on the four binary streams to determine the possible occurrence and location of the desired chunk. If more than one location in a given error curve satisfies this, then the location at which the product of the error curves is minimum should be considered the actual location of the desired chunk. In the case in which the product of the errors is minimum at the expected location of the chunk (see Section 5.3.2) *without* satisfying all four thresholds, the chunk detection is considered semi-successful (see "Comparison and Detection" in Section 6.1.2).

Once the first chunk in the chunk order is found, the subsequent chunks should be sought for in the correct corresponding locations.

5.3 Entire Jingle Detection

5.3.1 Buffering

Buffering of the input signal WCS's is essential to the detection of an incoming jingle. Without an input buffer, the system would not be afforded the freedom to look for the most unique chunks first before having to look at the less unique chunks which may precede them chronologically.

The length of the buffer may be determined by considering two constraints. First of all, the minimum buffer length for the input WCS's should be at least as long as the template jingle WCS's, so that the all chunks are in the buffer even if the most unique chunk (the first to be matched) is the last chunk of the jingle. Since the overall average throughput of input and template comparison can be no slower than the rate at which new signature samples are being created (i.e., one per WCS per timeframe), the maximum buffer length is constrained to be small enough to allow for a full comparison of the buffered signatures with the template signatures within the span of one timeframe. In the case that the buffer cannot be made as long as the template jingle, the template jingle must be treated as a group of consecutive smaller sub-jingles.

Once the above constraints are met, the last consideration in determining the buffer length and/or maximum template length is the tradeoff between confidence and delay of positive detection. A greater buffer to template length ratio may provide more context (both preceding and following the jingle) and therefore increase the confidence of accurate detection of the jingle or sub-jingle (see similar discussion in Section 4.4.1). The only tradeoff in increasing the buffer length is the extra delay of waiting for more post-jingle context to enter the buffer before indicating a positive jingle detection.

5.3.2 Location Extrapolation

Location extrapolation is critical to the "intelligent" detection of the individual chunks. Once the most unique chunk (the first in the chunk order) is found, the relative location of all the other chunks may be guessed with some useful degree of accuracy. This allows the system to intelligently narrow down the regions of interest where it attempts to detect subsequent chunks. In the case where a specific chunk may fail to satisfy one of the four of the broad acceptance region

criteria, the apriori prediction of its expected location allows the system to check for a semi-successful detection as described in Section 5.2.3.

The region of interest as mentioned above is ascertained by computing the range of possible locations given the range of possible speed shift. In the case of this implementation, the regions of interested were computed for ±10% speed shift.

Since the expected location is used to determine if a chunk failing at least one of the broad acceptance criteria may still may be considered semi-successfully detected, the expected location must encompass a much tighter region. The expected location of a desired chunk is predicted according to the average speed of the signal and the location of the closest chunk that has already been successfully detected. Since the playback speed may shift during the course of the incoming signal, the average speed of the signal (as determined by the chunks already successfully detected) may not be an exact predictor of the location of a new chunk. In order to compensate for this, the location extrapolation must incorporate a tolerance that increases with increased distance between the chunk in question and the closest successfully detected chunk.[8] As described in Section 5.2.3, if the error curve is minimized within the region of tolerance about the predicted location of the desired chunk, then the detection is considered semi-successful.

[8] The tolerance used in this case was 1 timeframe tolerance per every 100 timeframes distance.

6. Results

The system has been tested on recognition of songs from different audio sources. The sources include radio, a low quality tape player[9], television, and digital audio. Radio and tape as sources provide the system with a practical test of robustness to noise and playback speed shift. Television tests the system for robustness to artifacts of companding, while digital audio tests robustness to artifacts of digital compression-decompression.

6.1 Radio

The sample used from the radio is the song "Into the Groove" by Madonna. The 27 sec radio sample is used as the input signal broken into 21 chunks, each 2.3 sec long and overlapped by 1.2 sec. A sample of the corresponding segment in the original CD recording is used as the template signal and similarly broken into 21 chunks. This test is used to demonstrate and justify the fundamental operation of the recognition system (as described in Chapter 5) as well as to assess its performance.

6.1.1 Preprocessing

In the preprocessing stage of the system, once the signature formation has been completed, the next step is to determine the order in which the chunks are to be detected. As described in Section 5.1.2, the system compares and sorts the chunks on the basis of their uniqueness values. Uniqueness is essentially a measure of how significantly the error drops when the original template signatures are aligned with the signatures of the artificially predistorted signal (see 5.1.1). Looking at Figures Figure 19 and Figure 20, for example, the chunks with a greater distance between the average and

[9] Although the system is not specifically intended for monitoring of low-quality tape output, this sample contains more noise and speed shift artifacts than even radio and was therefore chosen as a pertinent supplementary test.

minimum error may be considered more unique. As shown in Figure 18, the actual broad uniqueness values determined during preprocessing differ a great deal from chunk to chunk. The large range of uniqueness values clearly necessitates the reordering of the sequence in which the chunks are to be detected. The new chunk order is:

```
New Chunk Order: 12  13  17  18  7  15  16  14  11  19  8  20  4
6   2  21  10  5  3  1  9
```

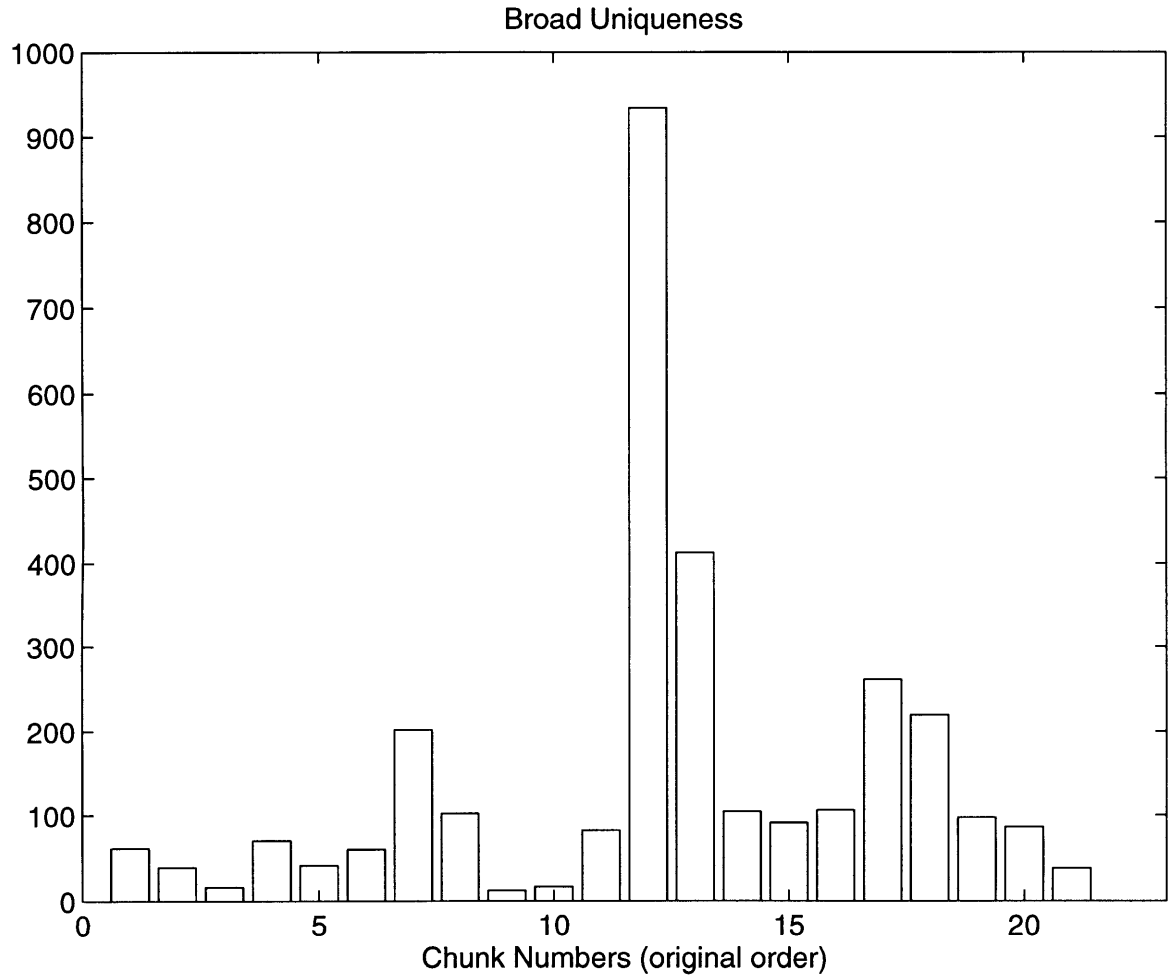

Figure 18: Broad Uniqueness Values

The next step in the preprocessing stage is the determination of thresholds. Figure 19 and Figure 20 show the AC Center Frequency signature errors for comparison of each template chunk with the corresponding chunk in either the artificially predistorted signal (see 5.1.1) or the actual input signal, respectively. As depicted in the two figures, there is great variation in the average errors of different chunks; in fact, in Figure 20, the average errors of several chunks are actually less than the minimum errors of other chunks. It is for this reason that the system "customizes" the error thresholds by determining them separately for each chunk rather than imposing one fixed threshold for the entire signal.

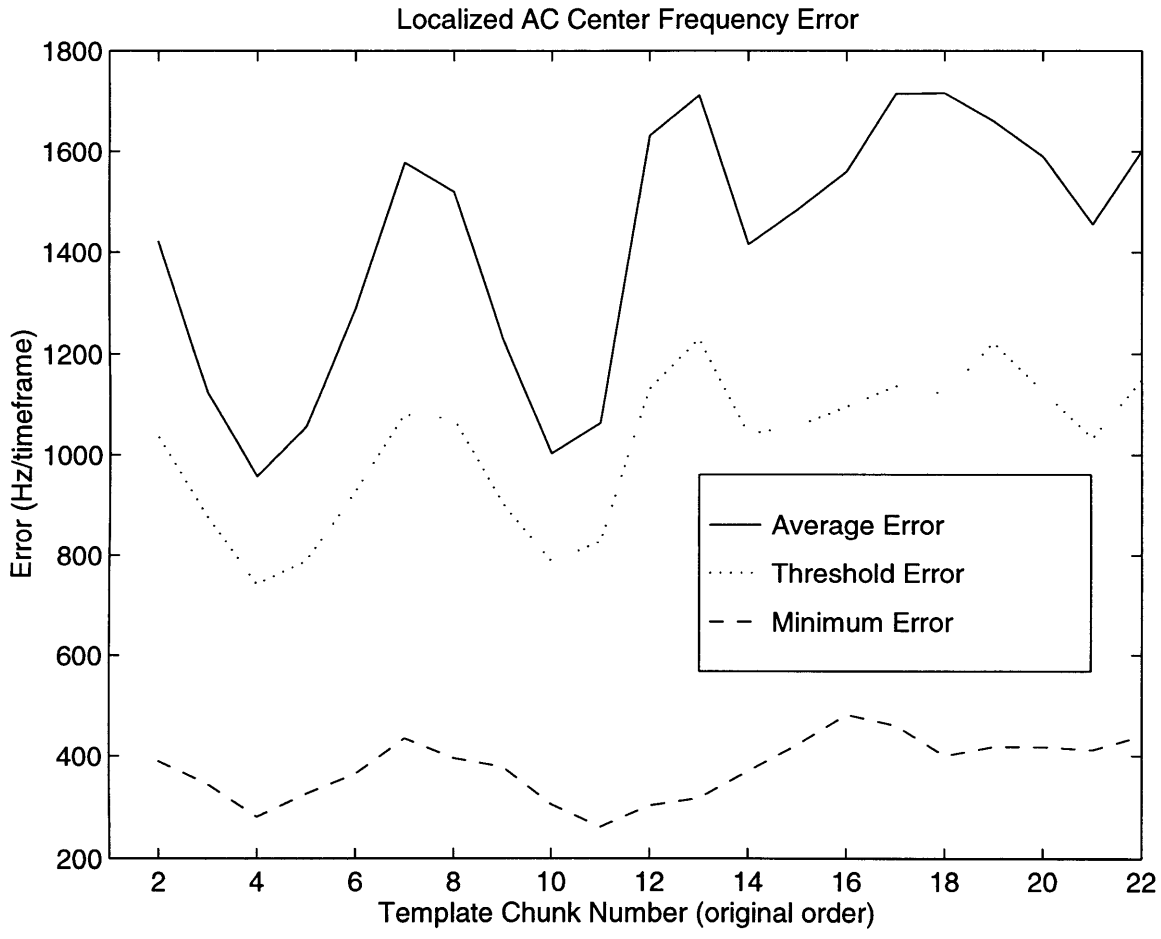

Figure 19: Preprocessing AC Center Frequency Errors

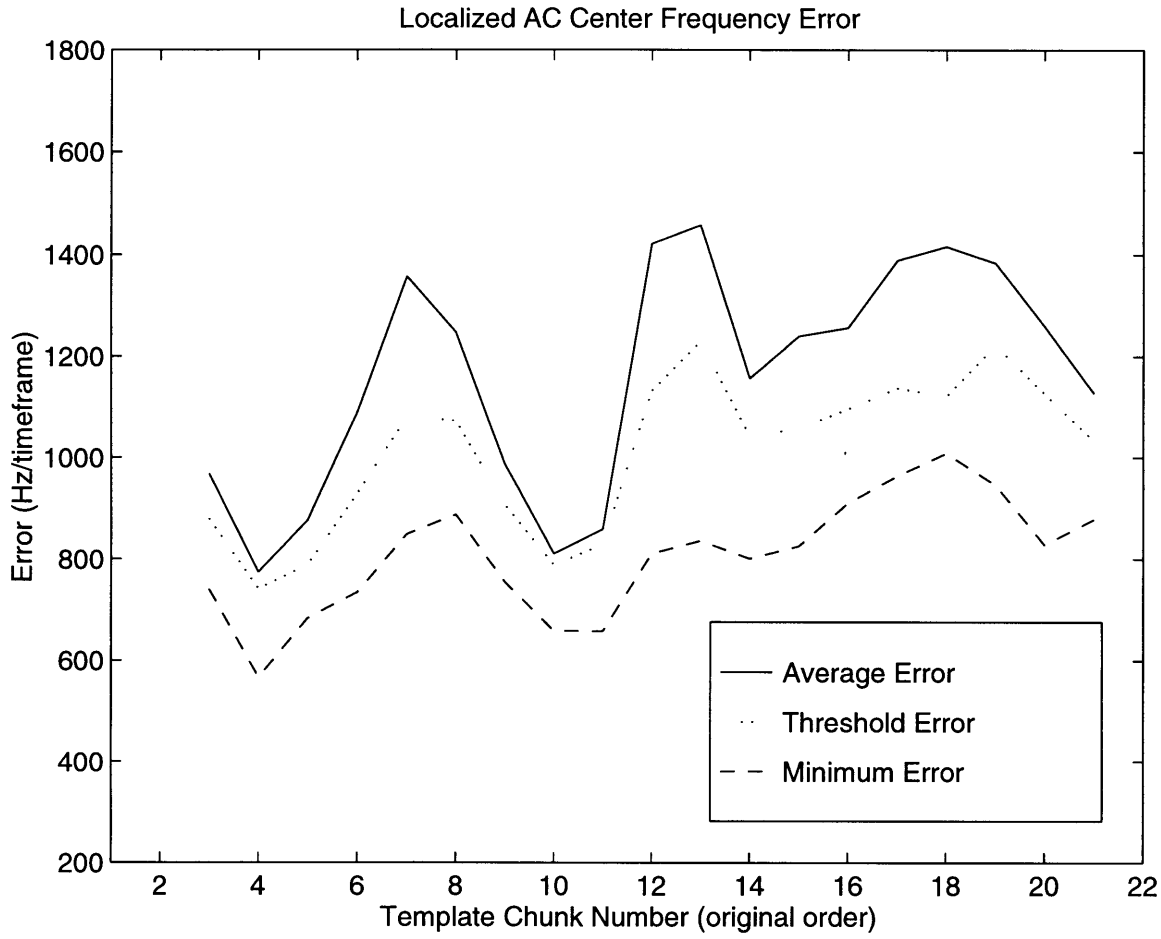

Figure 20: Actual AC Center Frequency Errors

Under close examination, Figure 19 and Figure 20 demonstrate the practical success of the system's thresholding procedures. Figure 19 depicts information about the errors produced when the recognition system compares the template signatures with the signatures of an artificially pre-distorted signal. The pre-distorted signal is created by adding enough random noise to the signal to artificially lower the SNR to only 40 dB. This is done because the 40 dB SNR is part of the system specifications and therefore determines the worst case scenario which the system should handle. In Figure 20, the average errors of each chunk are significantly less than those of Figure

19, thus indicating that the actual signal from the radio had less noise. The minimum errors in Figure 20, on the other hand, were significantly *greater* than those in Figure 19, thus indicating the effects of other unexpected artifacts of practical radio transmission and reception. Since the worst case scenario as shown in Figure 19 is used to determine the error thresholds, it may be surprising that the thresholds would have been set so low with respect to Figure 19's average errors that they could successfully handle cases with much less noise as in Figure 20. Furthermore, it may be even more surprising that the thresholds are set so high with respect to the *worst case's* minimum errors that they could successfully handle unexpectedly high minimum errors of the *better case*. It is for these reasons that the system's procedures for threshold determination (as described in Section 5.1.3) may be deemed successful.

6.1.2 Individual Chunk Detection

After the completion of the preprocessing stage, the recognition system enters the detection stage in which it continuously monitors the audio broadcast. Since the system has no apriori knowledge as to the time at which the desired jingle will be broadcast, the system attempts in vain to match the template to *everything* played on the radio. Therefore, in order to ideally test the system, the recognition system must attempt to match the template chunk signatures against the entire set of possible audio segments. Since different chunks are likely to be similar if they are from the same song, they are also consequently more confusable by the recognition system. The best feasible method for testing the recognition of individual chunks is, therefore, for the detection stage of the system to attempt to match each chunk of the template signal against each and every part of the corresponding test input signal itself.[10]

[10] Normally, once the first (i.e., most unique) chunk has been recognized, the recognition system will narrow its search for successive (i.e., less unique) chunks to the most probable region of the signal in the buffer. In order to best test the recognition system, however, each template chunk will be attempted to be recognized all throughout the entire input signal.

Playback Speed Issues

In the detection stage of the recognition of the song "Into the Groove", the system determined that the entire song was broadcast without any speed shift. For this reason, this particular test sample will not provide any interesting information regarding the first two steps of the detection stage, Approximating Speed and Rescaling Template Chunk Signatures (see Sections 5.2.1 and 5.2.2, respectively). Demonstration of these two steps is discussed later as part of the examination of the audio tape test sample.

Comparison and Detection

The third and most critical step in the detection stage involves the actual detection and isolation of individual chunks from the input signal buffer. This is usually to be done by a combination of location guessing and the localized application of the precomputed thresholds. For the purpose of testing recognition of individual template chunks against every input signal chunk, the thresholds will be applied to the entire input signal buffer rather than be localized.

Figure 21 shows the detection status of all the template chunks, reordered by descending uniqueness (i.e., beginning with Chunk 12 and ending with Chunk 9 as noted above). As can be seen from the figure, the first 14 most unique chunks were all detected either completely successfully or semi-successfully. A completely successful identification requires that each signature comparison test be simultaneously satisfied at the expected location of the chunk (except in the case of the first chunk[11]). An example of error curves and signature test evaluation flags for a completely successful identification are shown in Figure 22 and Figure 23. A semi-successful identification occurs when the location of the minimum product of errors coincides with the expected chunk location although all four of the errors are not below their respective thresholds.

An example of error curves and signature test evaluation flags for a semi-successful identification are shown in Figure 24 and Figure 25.

There are two concepts integral to the recognition system's method for individual chunk detection that are being tested -- the usefulness of Wide Contribution Signatures and the robustness of broad acceptance region signature comparison tests. The performance of the Wide Contribution Signatures (WCS) is well demonstrated in Figure 22 and Figure 24. These figures show the local error curves resulting from the sliding subtraction computed in the detection of the respective chunks. Although one figure shows the error curves for a completely successful chunk detection whereas the other shows error curves for a semi-successful detection, both figures demonstrate desired behavior on the part of the WCS's. In both figures, the error curves of all four signatures drop to their absolute minimum values simultaneously at the correct location[12] of the given chunk. This demonstrates the utility of each of the Wide Contribution Signatures for discriminating between chunks despite the fact that the WCS's represent general rather than specific characteristics.

The second key concept being tested is the robustness gained by use Broad Acceptance Region (BAR) signature comparison tests. The ideal performance of BAR tests is demonstrated by the completely successful chunk detection shown in Figure 22 and Figure 23. Figure 23 shows the flags indicating the locations at which the four error curves shown in Figure 22 satisfy the comparison tests by dropping below their respective error thresholds. As the flags indicate, false positive matches can and do occur (such as with the AC Volume, Center, and Zero-Crossings signatures) but not at concurrent locations. The flags are therefore concurrently set only at the correct location of the given chunk.

---

[11] Since there is no apriori information about when the signal may occur, there is no expected location for the first chunk to be detected.

[12] The correct *relative* chunk location in the plots of local behavior is timeframe 200.

The semi-successful detection shown in Figure 24 and Figure 25 demonstrate a less ideal performance of the BAR tests. As indicated by the flags in Figure 25, the comparison tests are never satisfied by the AC Frequency Spread signature; therefore, despite correct alignment of flags for comparisons of the other signatures, the ANDed flags are never set and thus preempt a completely successful detection. It should be noted that of the six semi-successful detections, five of them failed the AC Frequency Spread comparison test. Thus, rather than implying that the BAR tests are not robust, the occurrence of semi-successful detections may simply indicate the need for broader acceptance regions for comparing AC Frequency Spread signatures.

A count of contiguous flags set during the detection of the different chunks is shown in Figure 26. As the figure demonstrates, each comparison test flag is set off many times due to its broad acceptance region. The number of ANDed flags, however, is always only one (with the exception of the non-unique chunk that was last in the chunk order) -- thus indicating that although the acceptance regions are broad, the intersection of the regions is rather small. In summary, the flag counts in Figure 26 therefore demonstrate that the Wide Contribution Signatures meet their goal of being inherently different representations while the Broad Acceptance Region comparison tests meet their goal of being an effective way to implement them.

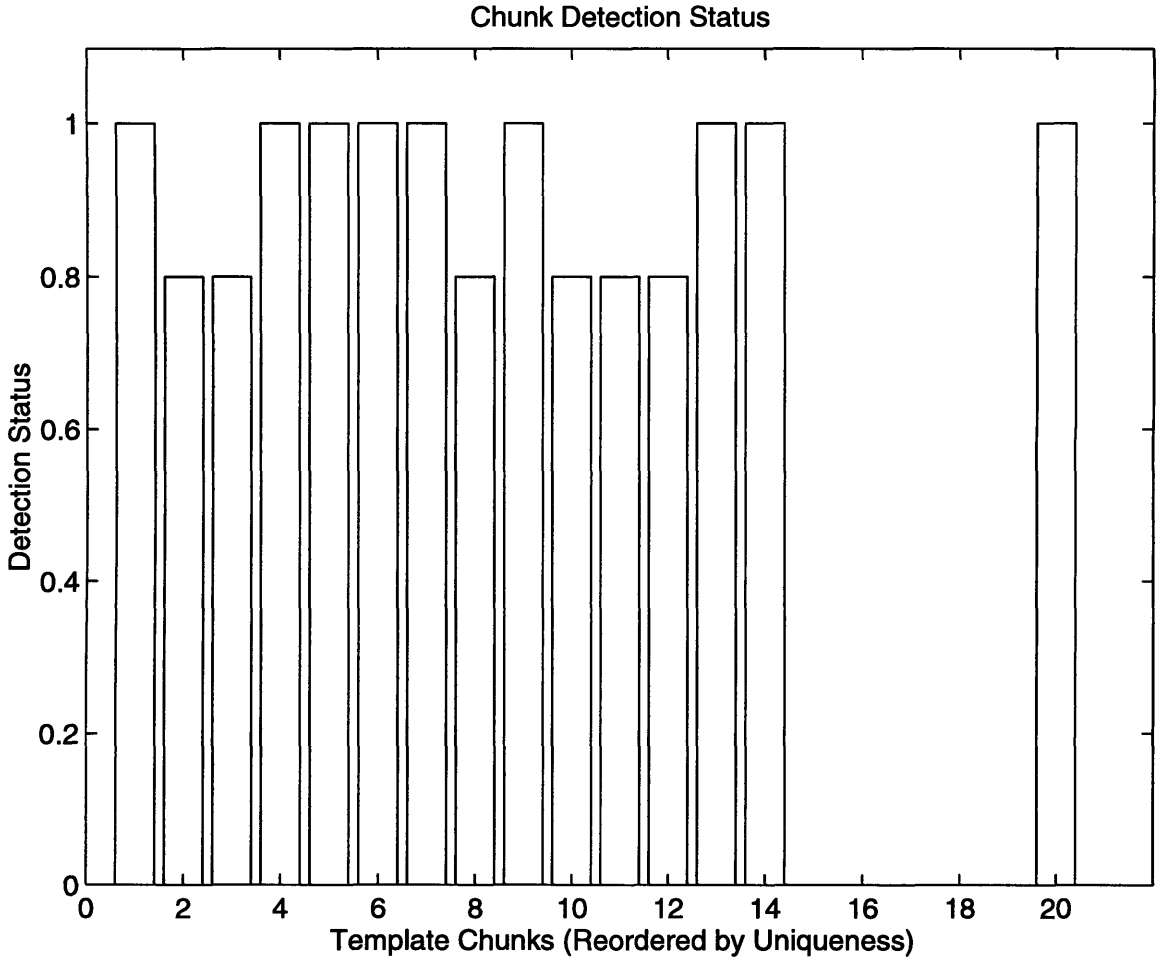

Figure 21: Detection Status of Individual Chunks

Detection status 1.0 indicates a completely successful detection;
detection status 0.8 indicates a semi-successful detection.

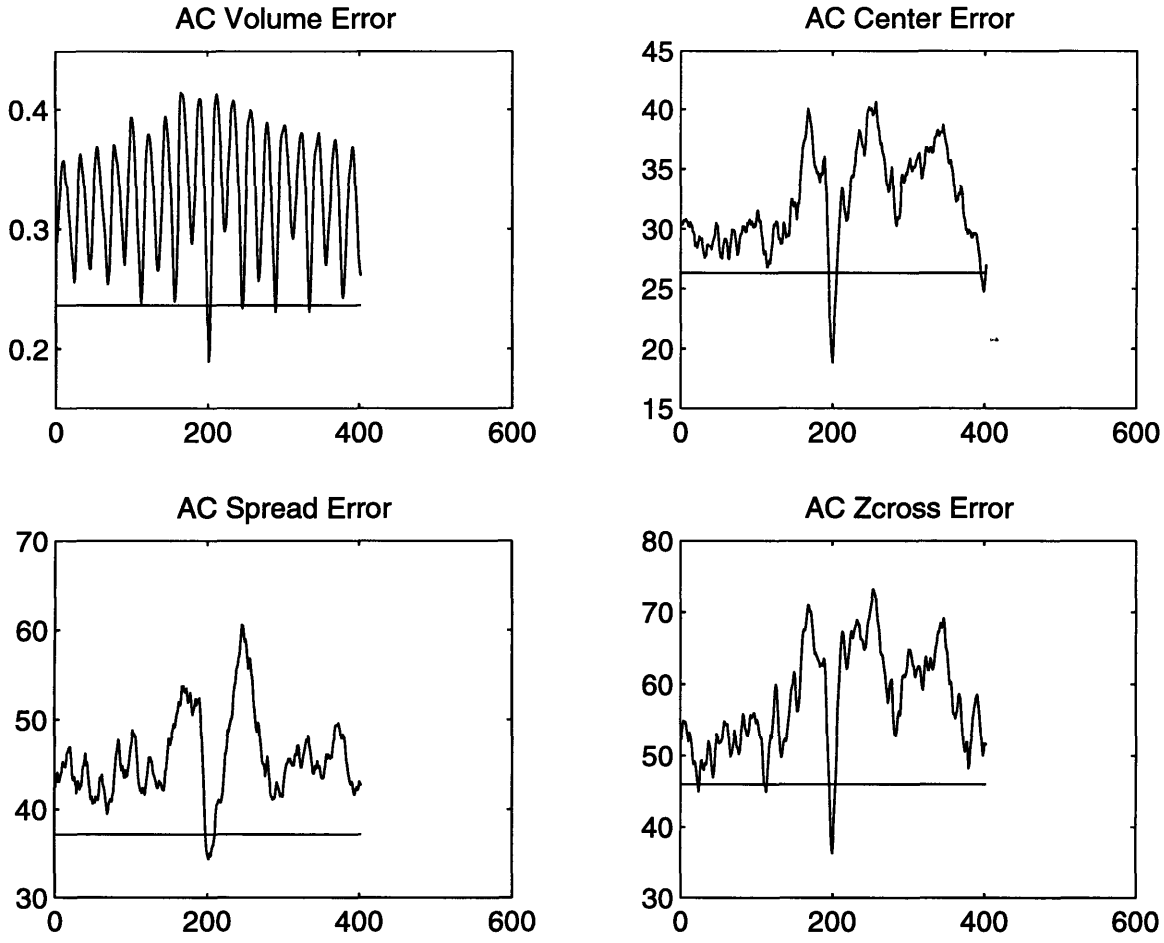

Figure 22: Error Curves of Successful Detection of Chunk 1 (new order)

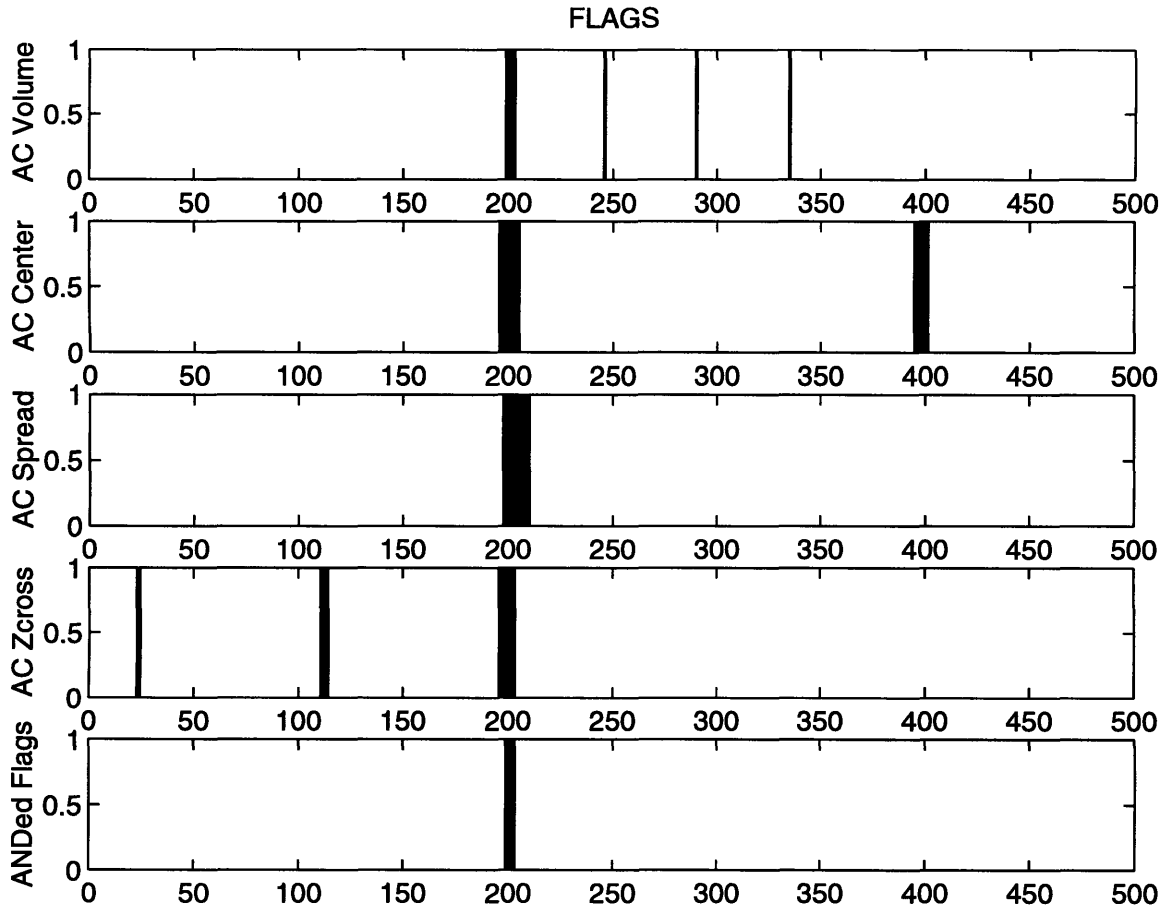

Figure 23: Flags of Successful Detection of Chunk 1 (new order)

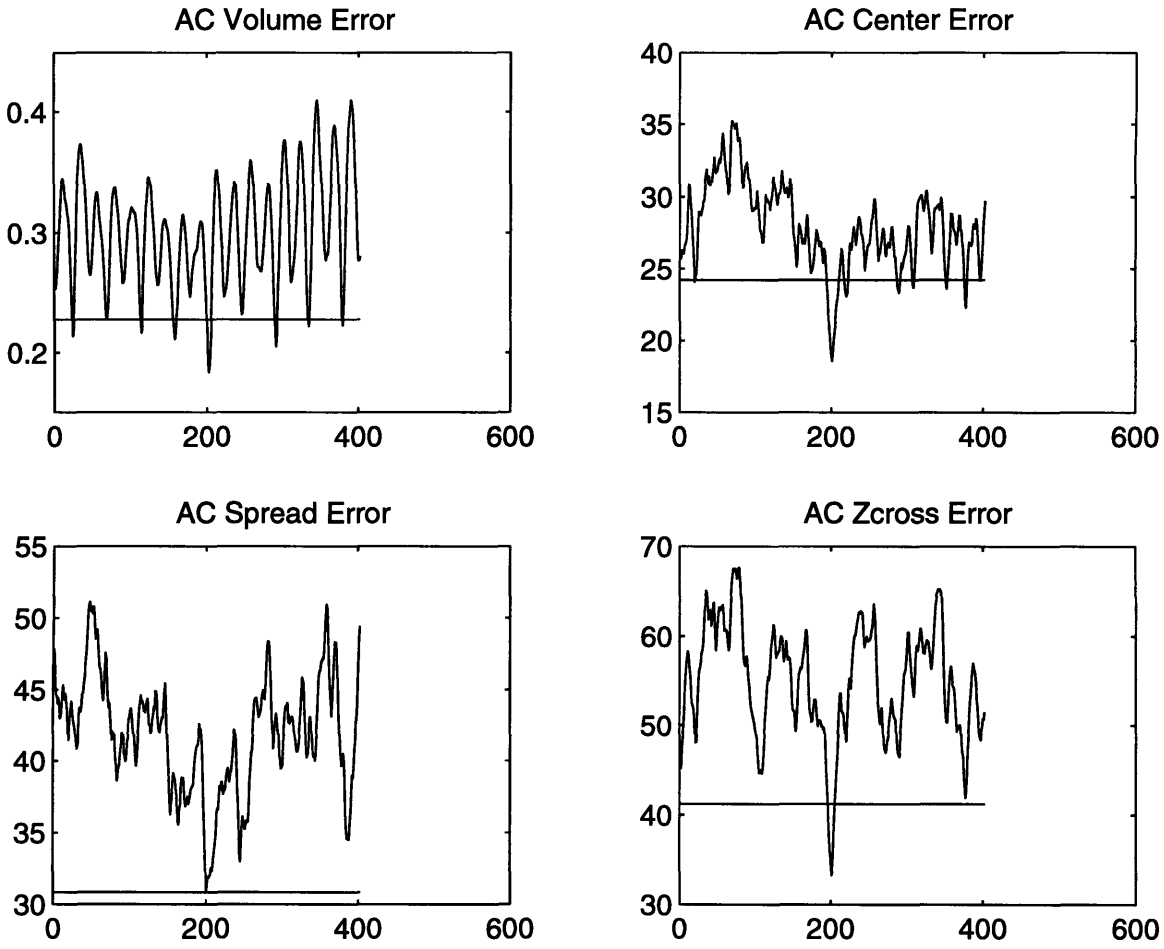

Figure 24: Error Curves of Semi-Successful Detection of Chunk 8 (new order)

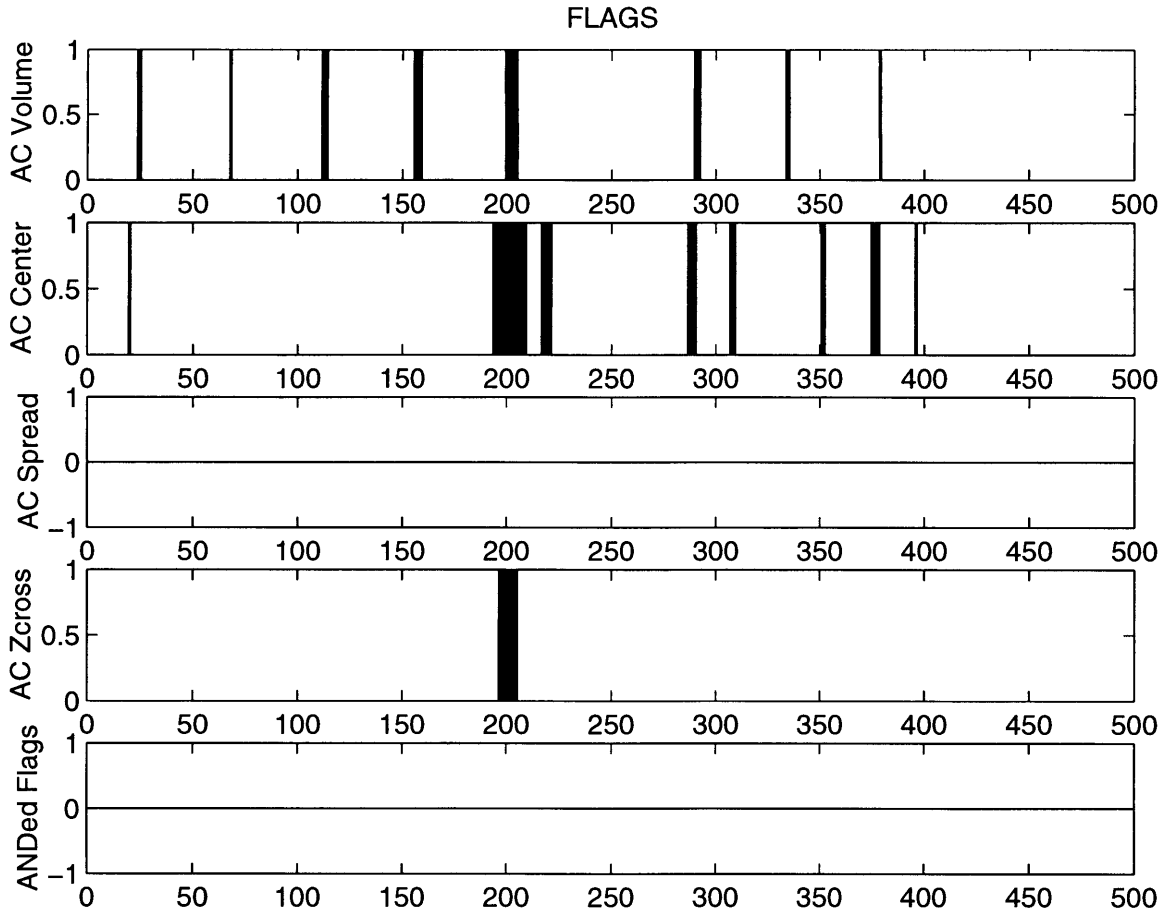

Figure 25: Flags of Semi-Successful Detection of Chunk 8 (new order)

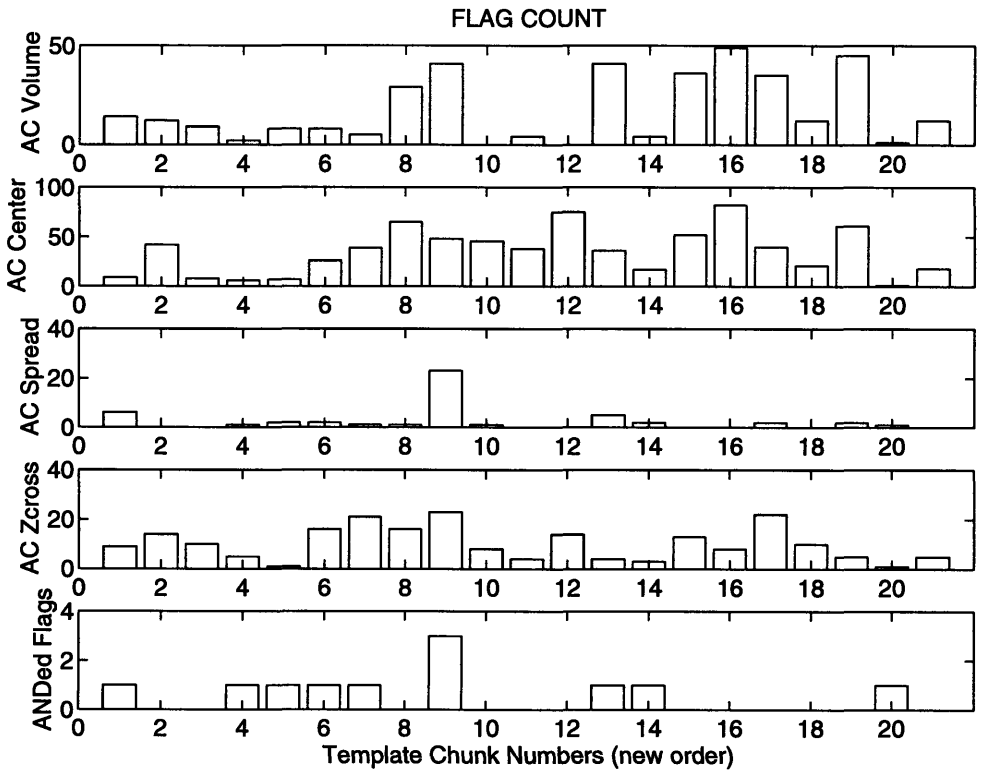

Figure 26: Flag Counts

6.1.3 Entire Jingle Detection

The crucial component in extending the individual chunk detection methods to detection of the entire jingle is the ability to guess the location in the input buffer at which to expect the next chunk (see Section 5.3.2). Although this test input sample did not challenge the recognition system with playback speed shift, the results shown in Figure 27 demonstrate excellent extrapolation of chunk location.

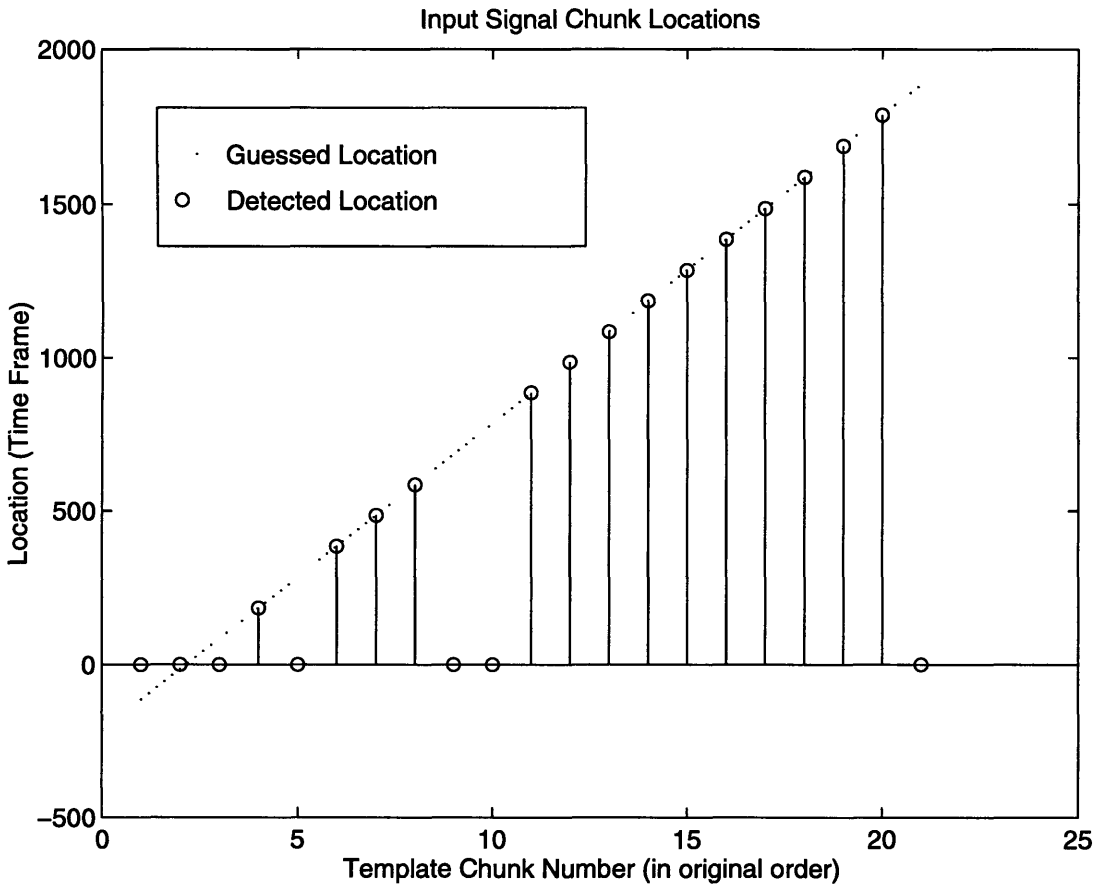

Figure 27: Extrapolated and Detected Chunk Locations

6.2 Audio Tape

The sample taken from a low quality audio tape player is of the song "King of Pain" by Police. The 18 sec audio tape sample is used as a the input signal and broken into 14 chunks, each 2.3 sec long and overlapped by 1.2 sec. A 18 sec sample of the corresponding segment in the original CD recording is used as the template signal and similarly broken into 14 chunks. This test is used to

demonstrate the performance of the recognition system in handling an input signal with multiple shifts of playback speed.

As seen in Figure 28, the recognition system proves itself to be quite successful in handling the audio tape input signal. Out of 12 chunks in the new chunk detection order (2 of the 14 chunks are not considered unique), only 2 less unique chunks were not detected. And out of the 10 detections, the five most unique chunks resulted in detections that were completely successful.

Such success in individual chunk detection is made possible only by a combination of accurate speed approximation and robustness to speed approximation error. As shown in Figure 29, the system consistently guessed the speed correctly to within 2% of the actual speed. For the more unique chunks, the speed was guessed to within 1%; but for less unique chunks, such as chunks 3 and 6, the speed approximation error was as high as 2%. For the more unique chunks, the comparison tests are robust enough to withstand a 2% speed approximation error (see Figure 31) but fail with speed approximation errors above 4% (see Figure 32).

Since the semi-successful detection of the less unique chunks hinges upon the minimum signature comparison errors coinciding with the expected location of the chunk, the accurate guessing of that expected location is very critical to the detection of those chunks. The guessed and actual locations as well as the associated errors are shown in Figure 33 and Figure 34, respectively. As seen in Figure 34, the error for guessed location is notably greater for the less unique chunks (Chunks 3-7); this is probably due to the greater variation of the speed between these chunks. It is interesting to note, however, that despite the greater discrepancies between guessed and actual locations, the guessed locations of Chunks 3 to 7 were all considered close enough to the actual locations for semi-successful detection. This is a direct result of the fact that a tolerance is built in to the location extrapolation algorithm (see Section 5.3.2) for guessing the location for a chunk such as

Chunk 4 by using information about a chunk as far away as Chunk 10[13]. The fact that Chunks 3-7 could be semi-successfully detected despite the significant fluctuation in playback speed demonstrates the successful operation of the location extrapolation algorithm.

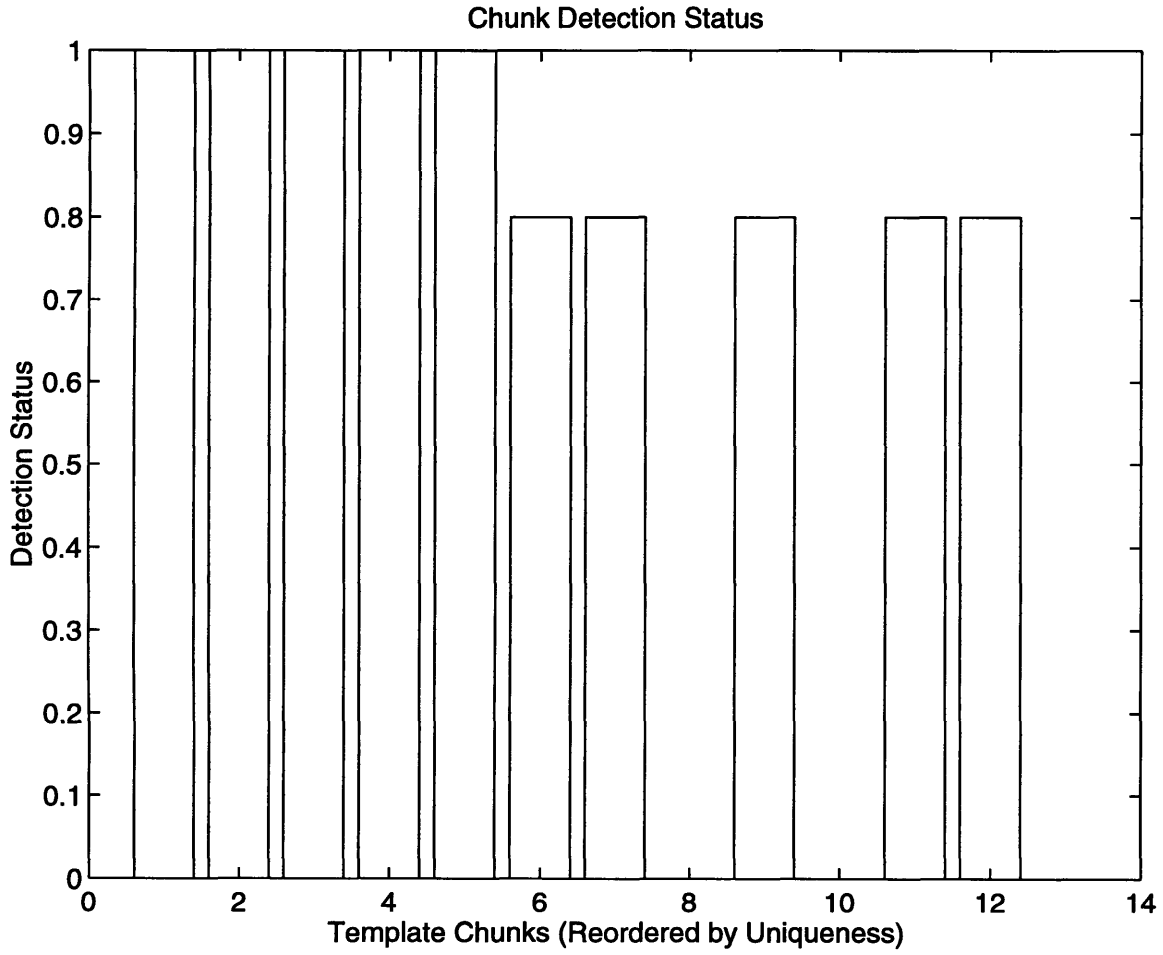

Figure 28: Detection Status of Individual Chunks of Audio Tape Sample

```
New Chunk Order:  10  11  13  12  14  4  3  2  5  1  6  7
```

[13] Chunk 10 is the closest chunk to Chunk 4 that would have been detected prior to Chunk 4. Chunk 10 would have been detected before Chunk 4 because it precedes Chunk 4 in the `New Chunk Order`.

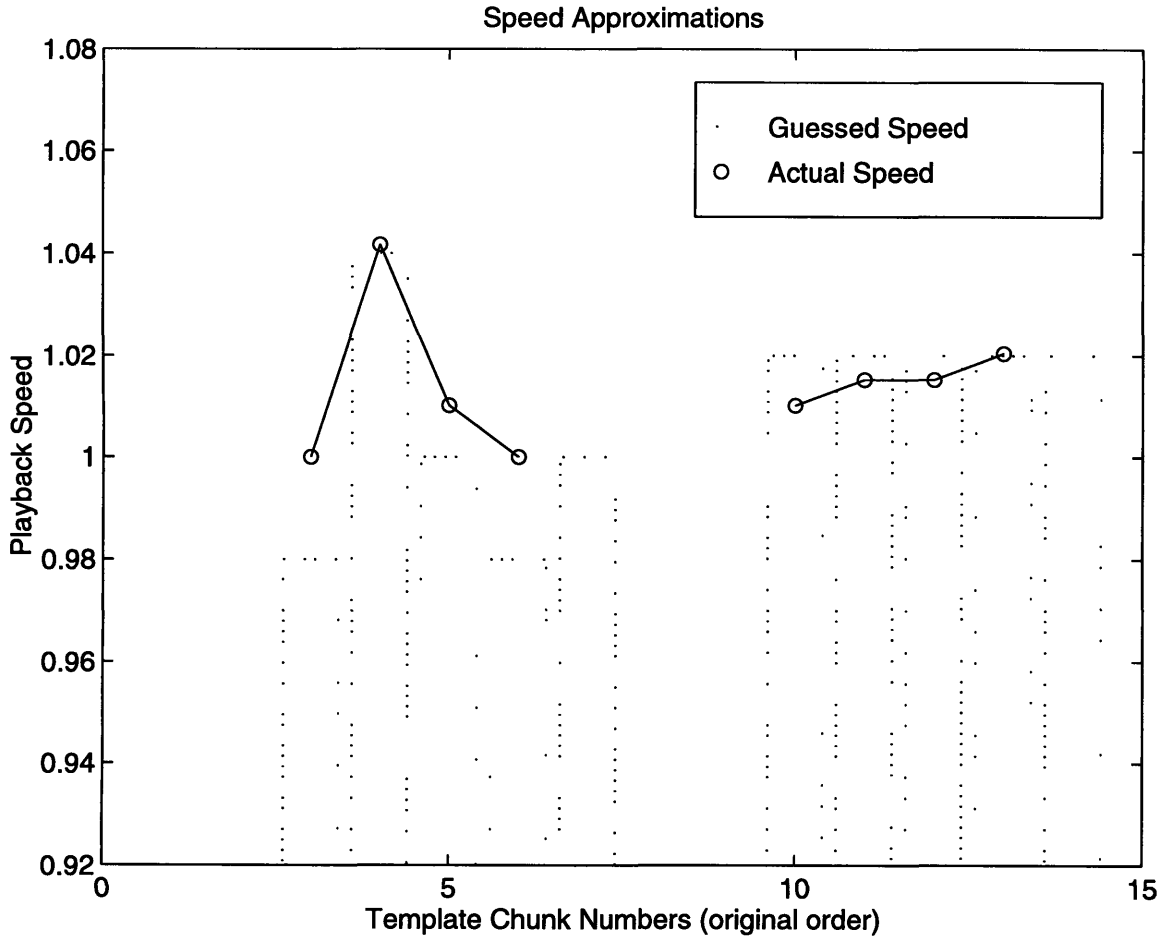

Figure 29: Speed Approximations

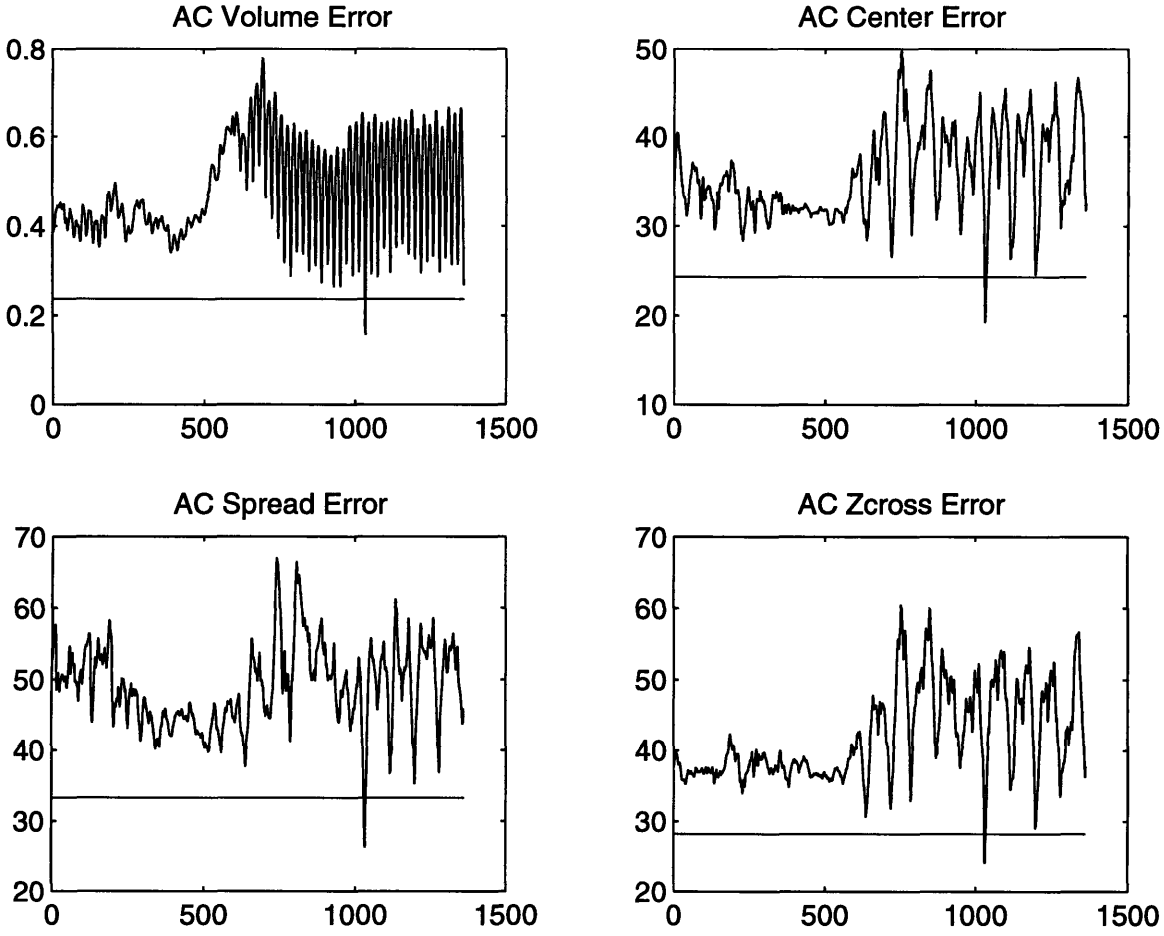

Figure 30: Error Curves for No Speed Approximation Error

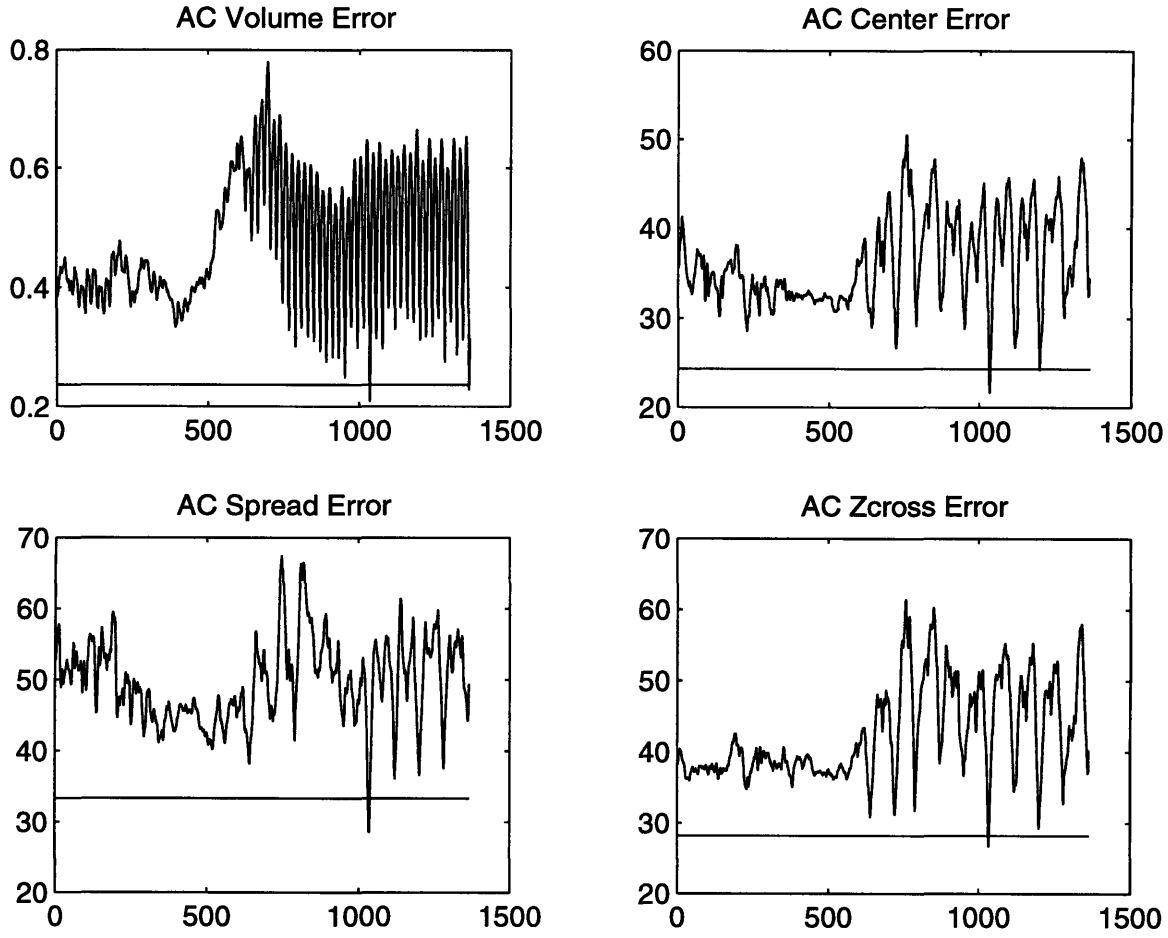

Figure 31: Error Curves for 2% Speed Approximation Error

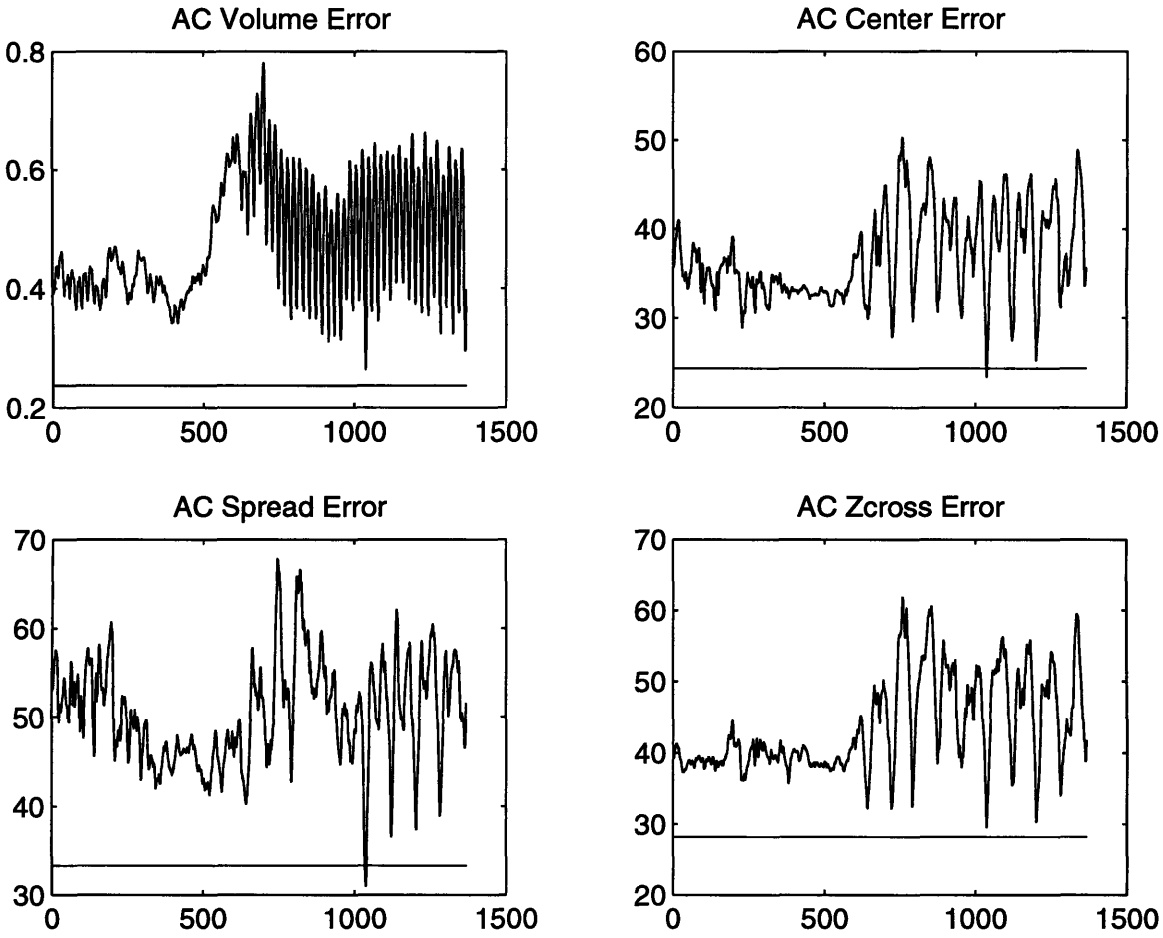

Figure 32: Error Curves for 4% Speed Approximation Error

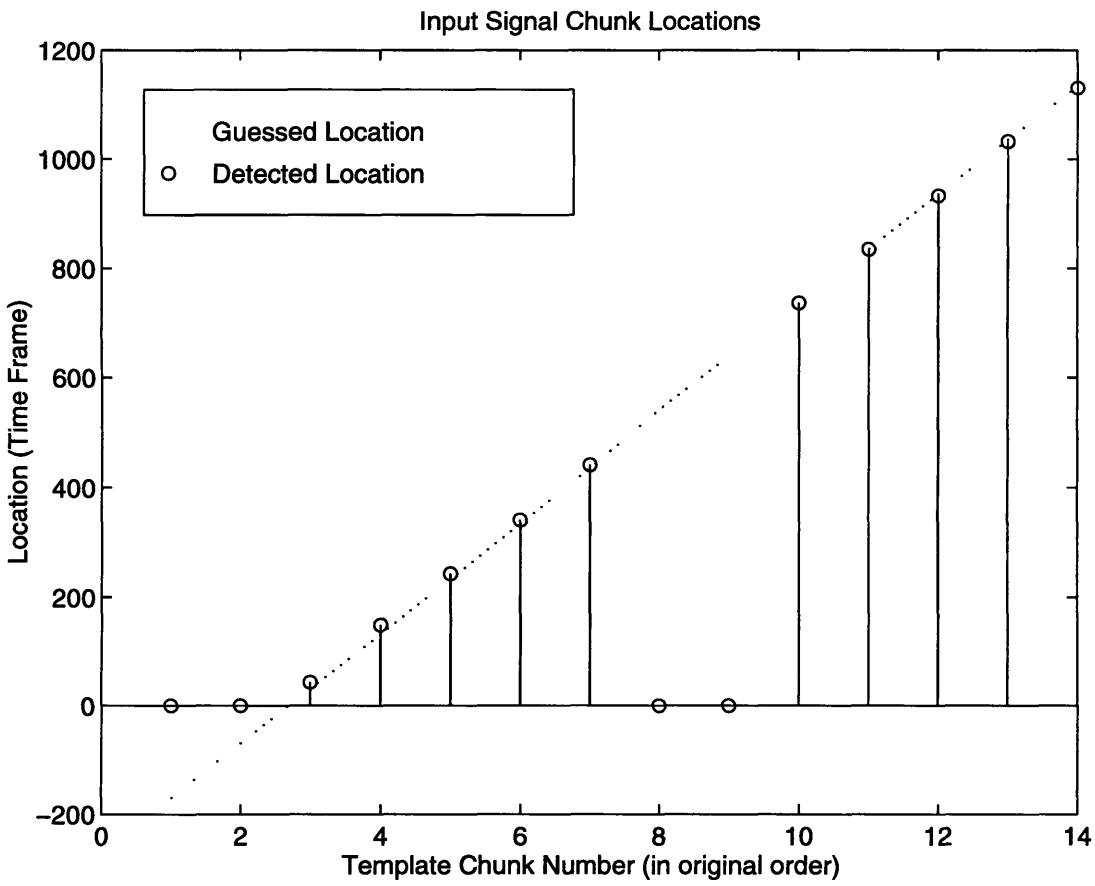

Figure 33: Guessed and Detected Chunk Locations

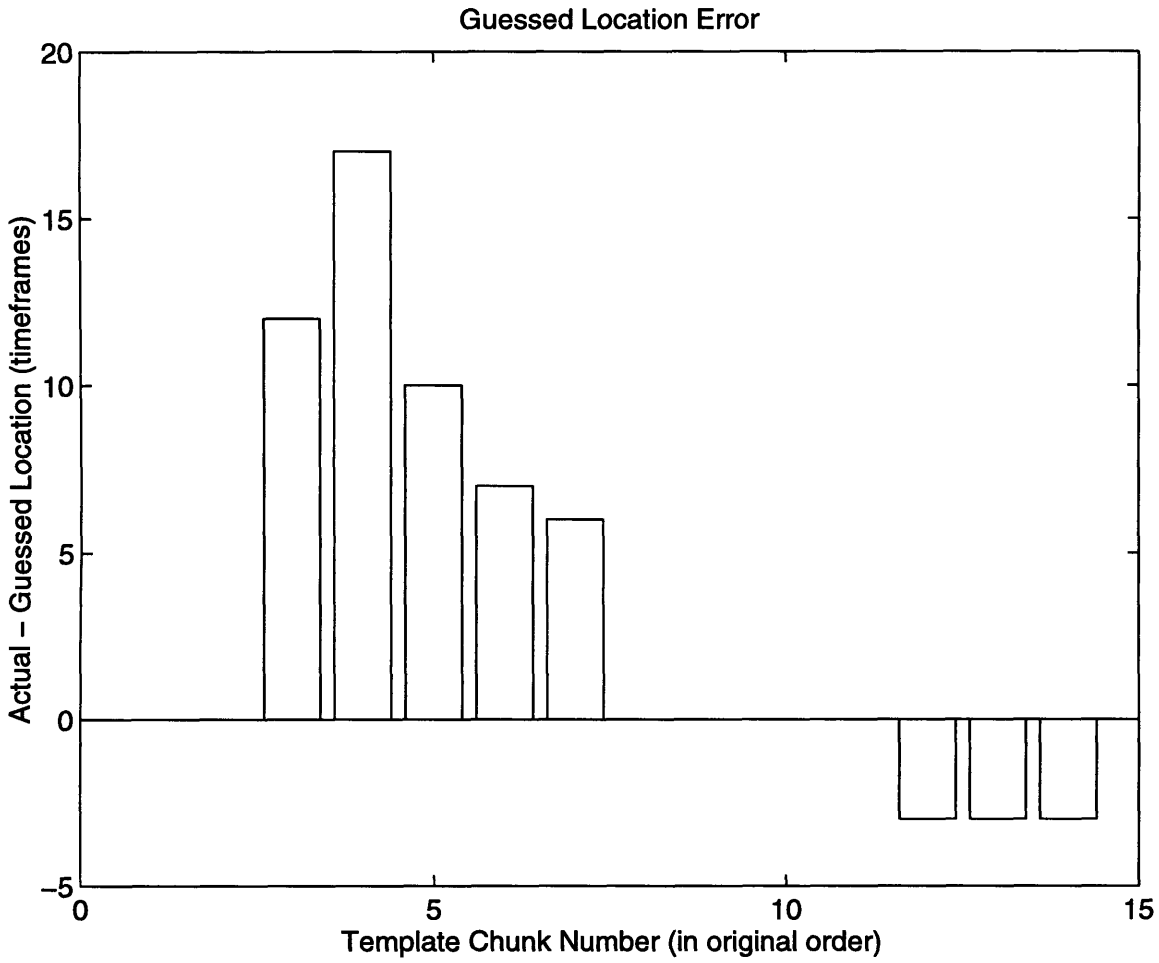

Figure 34: Guessed Location Errors

6.3 Television

The sample taken from cable TV transmission of the song "Living in Danger" by Ace of Base. The 24.3 sec television audio sample is used as a the input signal and broken into 19 chunks, each 2.3 sec long and overlapped by 1.2 sec. A 23.8 sec sample of the corresponding segment in the original CD recording is used as the template signal and similarly broken into 19 chunks. This test

is used to demonstrate the performance of the recognition system in handling an input signal has undergone standard television BTSC compansion.

The recognition system proves itself to be quite effective in handling the artifacts of compansion. 100% of the chunks were detected completely successfully. Also, the speed and locations were guessed with complete accuracy as well.

6.4 MPEG Digital Audio

The sample taken of ISO/MPEG compressed audio is of the song "Into the Groove" by Madonna. The 27 sec post compression-decompression digital audio sample is used as a the input signal and broken into 22 chunks, each 2.3 sec long and overlapped by 1.2 sec. A sample of the corresponding pre-compression segment in the original CD recording is used as the template signal. This test is used to demonstrate the performance of the recognition system in handling an input signal containing characteristic artifacts (such as the pre-echoes shown in Figure 35) of digital compression-decompression systems.

The recognition system proves itself to be very effective in handling compression-decompression artifacts. Despite an average minimum AC Center Frequency error of 378 Hz, an average Zero-Crossing discrepancy of 18%, and an average AC volume error of 116%, 100% of the chunks were detected completely successfully. The playback speed and chunk locations were guessed with complete accuracy as well.

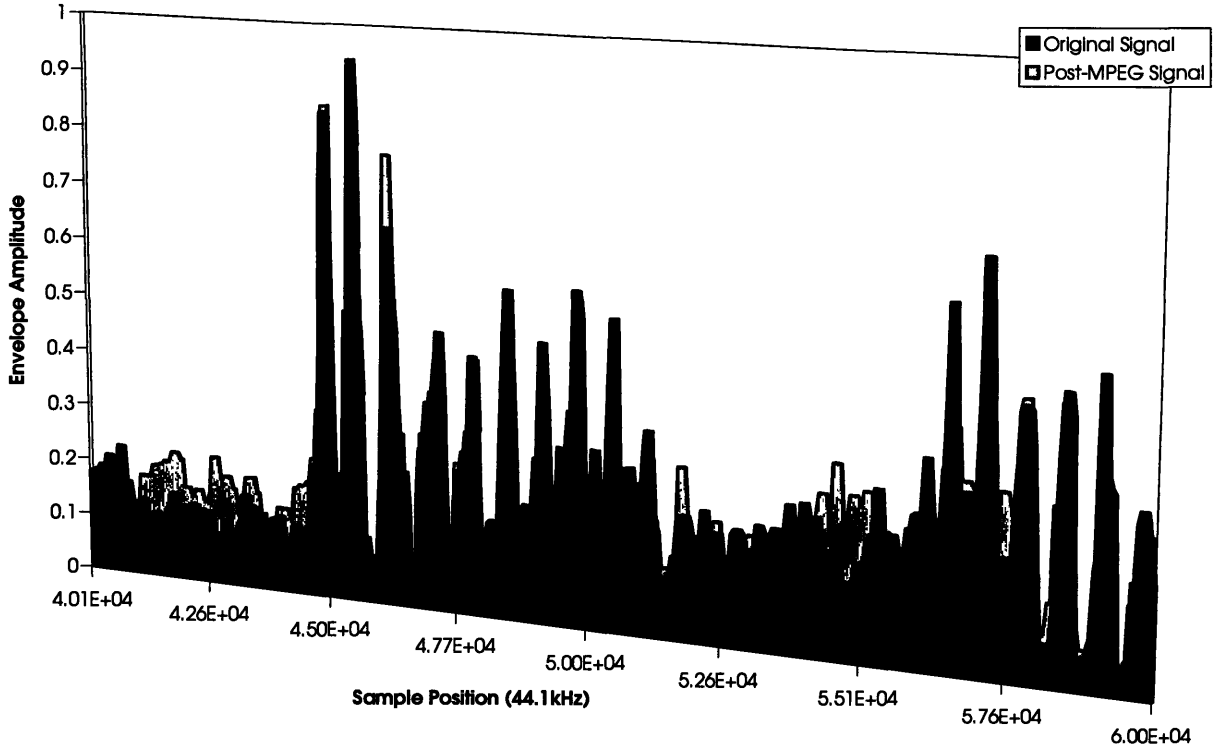

Figure 35: MPEG Compression Artifacts

7. Discussion and Conclusions

Of the four different types of audio samples tested, the television and digital audio samples were recognized most successfully, but the radio sample and the low-quality tape player samples provide the most useful information about the system. These results obtained validate several integral design concepts of the recognition system while also identifying areas for further improvement.

7.1 Conceptual Issues

The single most critical concept in the effective use of wide contribution signatures (WCS's) is the idea that the WCS's may be assumed to be *inherently different.* As demonstrated in Figure 26, each successfully detected chunk[14] produces only *one* ANDed flag a piece despite multiple flag occurrences for the individual broad acceptance region (BAR) tests. This experimentally confirms the concept of inherent difference by showing that although each individual BAR test could be satisfied at a number of different locations in the incoming signal, all four BAR tests could be satisfied only at the one correct location in the signal. Thus, although BAR tests demonstrated the breadth of their acceptance regions, the intersection of the regions was nevertheless small and discriminating.

[14] With the exception of the one non-unique chunk, which produced multiple ANDed flags.

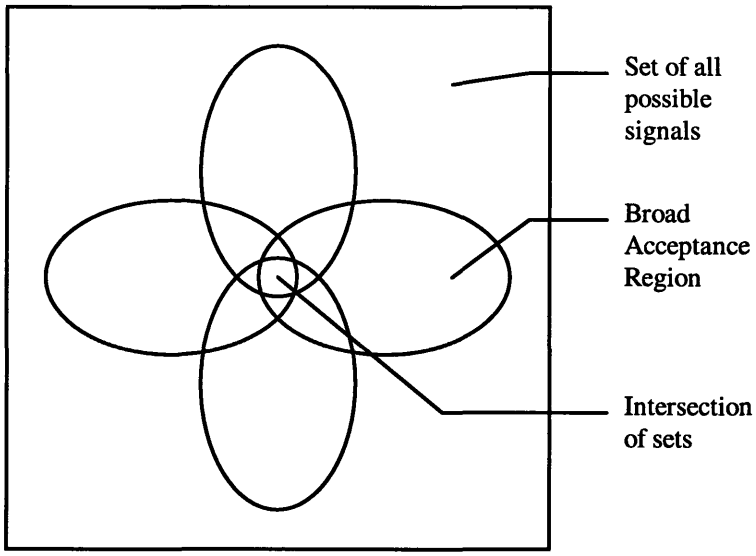

Figure 36: Wide Contribution Signatures concept

7.2 Implementation Issues

7.2.1 Threshold Determination

As discussed in Section 6.1.1, Figure 19 and Figure 20 demonstrate the success of the threshold determination system. First of all, Figure 20 experimentally validates the need for using different error thresholds for different chunks rather than using the same fixed threshold for every chunk. One of the assumptions crucial to the proper preprocessing of the signal is the assumption that original pre-transmission signal may be artificially predistorted in such a way that effectively simulates the signal distortion due to actual transmission. The validity of this assumption is confirmed by comparing Figure 19 with Figure 20; not only do the error curves of the artificially predistorted signal resemble their experimental counterparts, more importantly they also produce effective error thresholds. This success validates the concept that error thresholds "customized" individually for each chunk are necessary as well as practical.

Given that there were never more than only one ANDed flag for each successful test, there is no reason to believe that the individual broad acceptance regions used are of excessive breadth. Furthermore, as evidenced by the semi-successful detection exemplified in Figure 24 and Figure 25, the thresholds used for the BAR tests may in fact be too constricting. Therefore, it may be reasonable to increase the breadth[15] of the individual broad acceptance regions so long as the final discrimination of the ANDed flags is not compromised. This would result in more completely successful detections replacing detections that are currently only semi-successful.

7.2.2 Uniqueness-based Ordering of Chunks

As explained in Section 5.1.2, attempting to detect the chunks in the wrong order may result in a "wild goose chase" phenomenon. In order to avoid this, the system attempts to detect the most *unique* chunks first. Uniqueness is determined empirically by use of the error curves produced by the artificially predistorted input signal as detailed in the Uniqueness subsection of Section 5.1.2. The most unique chunks are to be detected first since these should be the most easily detectable. The results of testing with the four different types of audio broadcasts all consistently showed more successful detections for chunks that were rated more unique. Thus, the algorithm for determination of uniqueness and the concept of ordering based on it have been proved successful.

7.2.3 Speed Approximation and Signature Rescaling

As discussed in Section 6.2, the success in recognition of the audio tape signal reflects the success of playback speed approximation. Furthermore, since the approximated speed is used to produce rescaled versions of the original template signatures for comparison with the input signatures, the

[15] Increasing the breadth of the acceptance regions may be accomplished by raising the error thresholds of the BAR tests. Since the greater thresholds would be incorporated into the preprocessing, the uniqueness determination and chunk-ordering would adjust accordingly.

success of speed approximation reflects on the success of the zero-order hold signature rescaling method as well.

Although the input signal did not test the full range of ±10% speed shift, the results shown in Figure 29 do show robustness to as much as 2% error in speed approximation. This measure of robustness is useful in determining the desired percentage spread of between the successive speeds with which speed approximation subsystem will attempt to match the incoming signal. Since, however, the percentage spread used in this implementation was also coincidentally 2%, a smaller percentage spread – thus higher speed approximation resolution – must be used to accurately determine the measure of robustness.

7.2.4 Location Extrapolation

As described in Sections 5.2.3 and 6.1.2, it is possible to deem a detection semi-successful if the product of the four BAR test errors is found to be minimized at the expected location of a given chunk. Computation of the expected location is performed by the location extrapolation algorithm which takes into account all the previously successfully detected chunks as well as tolerances to accommodate for intermediate speed shifts between chunks. As discussed in Section 6.2, the semi-successful detection of certain chunks despite significant fluctuation in playback speed demonstrates the success of the location extrapolation algorithm. The consequent ability for the system to perform semi-successful detections in place of failed detections affords extra overall robustness to the system.

7.3 Overall Conclusions

The system has effectively proven the utility of properly implemented wide contribution signatures as a means of generalized audio recognition. The two issues most critical to a useful reliable implementation have to do with the implementation of inherent difference of the signatures and

proper ordering of detection. Both of these issues have proven to be both realizable as well as effective.

The current implementations of the underlying subsystems satisfactorily support the implementations of WCS's and proper ordering, but as discussed in Section 7.2, there is room for further improvement. The idea of producing error thresholds by means of artificial experimentation in the preprocessing stage proves to be practical, but the overall tightness of the thresholds may need to be lessened. The notion that signature rescaling may be implemented by a type of zero-order hold rescaling obviously reaps the practical benefits of saved computation time, but its robustness should be further tested to determine the optimal speed resolution for the speed approximation subsystem. Lastly, although it cannot be effectively tested since it operates at the lowest level of the system, the idea of comparing all signatures by sliding *subtractions* versus sliding multiplies has shown no adverse side-effects while obviously providing great amounts of saved computation time.

Lastly, it is important to note that the critical measure of success is the overall robustness of the system. As noted in Sections 6.3 and 6.4, the artifacts associated with cable television and digital audio do not pose any considerable difficulty in recognition of the signal. As detailed in Sections 6.1 and 6.2, between completely successful and semi-successful detections, all of the more unique chunks were detected leaving only the less unique chunks (17% to 25% of the total number of chunks) undetected[16] despite the presence of noise and playback speed fluctuation. This level of robustness for individual chunk detection proves that, in conjunction with the heuristics mentioned

[16] After making the threshold changes of Section 7.2.1, more of the semi-successful detections should become fully successful.

in Section 7.4, the methods developed in this paper lay the foundation for implementation of a robust real-time system for recognition of audio segments within an audio broadcast.

8. Future Development

Future development on individual chunk detection should focus on the improvement of existing signatures or the creation of new types of inherently different wide contribution signatures. As far as improving upon the existing types of signatures, simple changes in the underlying representations may bring about much better performance. For instance, simply changing the frequency center and frequency spread signatures to be based on a wavelet-based spectral representation rather than an STFT-based representation will cause the signatures to act more like human hearing by giving greater importance to lower frequencies while paying less attention to high frequency noise. As far as the development of completely new inherently different signatures, this will require the most ingenuity but also has the potential to advance system performance most substantially.

Finally, although it was not the purpose of this paper to describe a fully implemented recognition system, it should be noted that all the key concepts necessary to build one have been demonstrated. In order to extend these concepts to implement a fully realized system, it is still necessary to create heuristics to decide how to react to different states of intermediate detections. The heuristics will need to address questions such as:

- How early in the chunk detection order can the system detect a semi-successful or failed detection and still continue attempting to match the rest of the jingle?
- How many consecutive semi-successful detections are allowed?
- How many chunks need to be successfully detected in order to deem the entire jingle successfully detected?

Is the state of detection of the entire jingle a binary output or is it a confidence value based on the states of the individual chunk detections?

Once such heuristics are in place, the system can extend individual chunk detection to detection of the complete audio segment from any audio broadcast.

9. Appendix - Matlab Test Code

Following is an alphabetical listing of the actual Matlab code used to perform the tests used in this thesis. Some of the "M-files" listed act as scripts whereas others act as general functions. The highest level script is JINGLE_DETECT.M from which all the other scripts and functions are executed. CREATE_TEMPLATE.M and CREATE_SIGNAL.M are scripts used to produce the signatures used by JINGLE_DETECT.M.

Although the recognition system itself is designed to be able to monitor a continuous, indefinite length incoming audio stream using the input buffer as only temporary storage, the test code loads the entire incoming signal into the buffer. The test code therefore currently requires that the length of the test incoming signal take into account the memory limitations imposed by the system on which Matlab is being run. The more practical input buffer described in Section 5.3.1 can be implemented by simply deferring loading of the entire incoming signal in CREATE_SIGNAL.M to load it part by part in ASSIGN_BUFFER.M instead.

9.1 ACDC.M

```
function [ac, dc] = acdc(prevregion, region, nextregion,
dc_window);

% [ac, dc] = acdc(prevregion, region, nextregion, dc_window)
%   - dc_window is the two-sided (i.e., zero-position
%       in center) window used to weight points for the dc
average
%   - prevregion and nextregion are used with the dc_window when
it
%       needs to access points outside of region.  If either
prevregion
%       or nextregion is unneccessary (i.e., for one-sided
averaging),
%       then it can be set to zero length
%   - the ac is NOT normalized with respect to the dc average


dc = timeavg([prevregion region nextregion], dc_window);
dc =
dc([length(prevregion)+1:length(prevregion)+length(region)]);
ac = region - dc;
```

9.2 APPLY_THRESHS.M

```
% ********************************************
% *****  A P P L Y   T H R E S H O L D S  *****
% ********************************************
%
% Input: chunk_num, chunk, errs_thresh_xxxxx's
% Output: all flags, existence, position, all flag_counts


% Rename error thresholds for chunk of interest

acvol_thresh = errs_thresh_acvol(chunk_num);
accen1_thresh = errs_thresh_accen1(chunk_num);
accen2_thresh = errs_thresh_accen2(chunk_num);
aczcross_thresh = errs_thresh_aczcross(chunk_num);


% Establish flags for satisfaction of different error thresholds

flag_acvol = onesthresh(-acvol_thresh, -err_acvol);
flag_accen1 = onesthresh(-accen1_thresh, -err_accen1);
flag_accen2 = onesthresh(-accen2_thresh, -err_accen2);
flag_aczcross = onesthresh(-aczcross_thresh, -err_aczcross);

flags = flag_acvol & flag_accen1 & flag_accen2 & flag_aczcross;


% Establish flags taking into account satisfaction of other
flags
%    within a +/-1 timeframe radius

maybe_flag_acvol = zeros(1,length(err_acvol) + 2);  % one extra
sample in front
maybe_flag_accen1 = zeros(1,length(err_acvol) + 2); % and one
extra at the end
maybe_flag_accen2 = zeros(1,length(err_acvol) + 2);
maybe_flag_aczcross = zeros(1,length(err_acvol) + 2);

[y, minpos_acvol] = min(err_acvol);
[y, minpos_accen1] = min(err_accen1);
[y, minpos_accen2] = min(err_accen2);
[y, minpos_aczcross] = min(err_aczcross);

maybe_flag_acvol(minpos_acvol + 1 + [-1 0 1]) = [1 1 1];
maybe_flag_accen1(minpos_accen1 + 1 + [-1 0 1]) = [1 1 1];
maybe_flag_accen2(minpos_accen2 + 1 + [-1 0 1]) = [1 1 1];
maybe_flag_aczcross(minpos_aczcross + 1 + [-1 0 1]) = [1 1 1];

maybe_flags = maybe_flag_acvol & maybe_flag_accen1 &
maybe_flag_accen2 & maybe_flag_aczcross;
maybe_flags = maybe_flags([2:length(flags)+1]);  %takes one off
the front
```

```
                                % and one off the end


err_all = err_acvol .* err_accen1 .* err_accen2 .* err_aczcross;
[y, minpos_all] = min(err_all);

rel_position_min = max([1 (position_min - min(buff_range) +
1)]);
rel_position_max = min([length(err_all) (position_max -
min(buff_range) + 1)]);


% Completely successful detection

if sum(flags([rel_position_min:rel_position_max])) > 0
  flag_errs = flags .* err_all;
  flag_errs(find(flag_errs == 0)) = +inf .*
ones(size(find(flag_errs == 0)));
  flag_errs([ [1:rel_position_min]
[rel_position_max:length(flag_errs)] ]) = +inf .* ones(size([
[1:rel_position_min] [rel_position_max:length(flag_errs)] ]));
  [y, rel_position] = min(flag_errs);
  existence = 1;
  position = rel_position + min(buff_range) - 1;
  disp(sprintf('Chunk #%d EXISTS at %d',chunk_num,position));


% Almost completely successful detection

elseif sum(maybe_flags([rel_position_min:rel_position_max] + 1))
> 0
  maybe_flag_errs = maybe_flags .* err_all;
  maybe_flag_errs(find(maybe_flag_errs == 0)) = +inf .*
ones(size(find(maybe_flag_errs == 0)));
  maybeflag_errs([ [1:rel_position_min]
[rel_position_max:length(flag_errs)] ] + 1) = +inf .*
ones(size([ [1:rel_position_min]
[rel_position_max:length(flag_errs)] ]));
  [y, rel_position] = min(maybe_flag_errs);
  existence = 0.9;
  position = rel_position + min(buff_range) - 1 - 1;   %
subtracting one extra to make up for maybeflag range extension
  disp(sprintf('Chunk #%d MAY EXIST at %d',chunk_num,position));


% Semi-Successful detection

elseif (pos_guess > 0 & abs(minpos_all - pos_guess) <
(pos_tolerance * abs(chunk_num - old_chunk_num)) )
  existence = 0.8;
  position = minpos_all + min(buff_range) - 1;
  disp(sprintf('Chunk #%d SEEMS TO EXIST at %d (i.e., minimizes
error in correct location)',chunk_num,position));
```

```
% Failed detection

else
  existence = 0;
  position = 0;
  disp(sprintf('Chunk #%d DOES NOT EXIST.',chunk_num))
end


[minflag_val(chunk_num),minflag_pos(chunk_num)] = min(err_acvol
.* err_accen1 .* err_accen2 .* err_aczcross);


flag_count_acvol(chunk_num) = flag_counter(flag_acvol);
flag_count_accen1(chunk_num) = flag_counter(flag_accen1);
flag_count_accen2(chunk_num) = flag_counter(flag_accen2);
flag_count_aczcross(chunk_num) = flag_counter(flag_aczcross);
flag_counts(chunk_num) = flag_counter(flags);
```

9.3 ASSIGN_BUFFER.M

```
% ***************************************
% *****  A S S I G N   B U F F E R  *****
% ***************************************
%
% Returns fully described input signal buffer for a given
buff_range
%
% Input: buff_range, input signal signatures
% Output: buff_vol, buff_acvol, buff_accen1, buff_accen2,
%         buff_aczcross, buff_len


buff_vol = sig_vol(buff_range);
buff_acvol = sig_acvol(buff_range);
buff_accen1 = sig_accen1(buff_range);
buff_accen2 = sig_accen2(buff_range);
buff_aczcross = sig_aczcross(buff_range);

buff_len = length(buff_range);
```

9.4 ASSIGN_CHUNK.M

```
% ************************************
% *****  A S S I G N   C H U N K  *****
% ************************************
%
% Returns fully described template chunk for a given chunk_range
%
% Input: chunk_range, template signatures
% Output: chunk_vol, chunk_acvol, chunk_accen1, chunk_accen2,
%         chunk_aczcross, chunk_len


chunk_vol = temp_vol(chunk_range);
chunk_acvol = temp_acvol(chunk_range);
chunk_accen1 = temp_accen1(chunk_range);
chunk_accen2 = temp_accen2(chunk_range);
chunk_aczcross = temp_aczcross(chunk_range);

chunk_len = length(chunk_range);
```

9.5 BIN2MAT_LEFT.M

```
function matsound = bin2mat_left(filename,len)

% matsound = bin2mat_right(filename,len);
%
% Produces a matlab floating point vector of the LEFT channel
%   of 16-bit stereo .aiff audio file of length len where len
%   can be inf or a limited number of bytes


[fid, message] = fopen(filename)

headerjunk = fread(fid,200,'char');       % Reads in first 200
bytes

matsound = fread(fid,len,'short',2);      % Reads in 16-bit LEFT
track only

matsound = matsound / (2^15);             % Converts to floating
pt. format
%matsound = matsound / (2^14);             % Converts to
floating pt. format
```

9.6 BIN2MAT_RIGHT.M

```
function matsound = bin2mat_right(filename,len)

% matsound = bin2mat_right(filename,len);
%
% Produces a matlab floating point vector of the RIGHT channel
%   of 16-bit stereo .aiff audio file of length len where len
%   can be inf or a limited number of bytes


[fid, message] = fopen(filename)

headerjunk = fread(fid,202,'char');       % Reads in first 202
bytes

matsound = fread(fid,len,'short',2);      % Reads in 16-bit
RIGHT track only

matsound = matsound / (2^15);             % Converts to floating
pt. format
%matsound = matsound / (2^14);             % Converts to
floating pt. format
```

9.7 CENTER.M

```
function center_pos = center(spectrogram)

% centerpos = center(spectrogram)
%
% Finds centers of each column in spectrogram
%   where spectrogram should be non-negative everywhere


t_len = size(spectrogram,2);
bins = size(spectrogram,1);
center_pos = zeros(1,t_len);


for tf = 1:t_len% for each timeframe of spectrogram

  weights = [0; spectrogram([1:bins],tf); 0];      %NOTE: Added
2 zero row as spacer
  tot_weight = sum(weights);

  if tot_weight == 0
    center_pos(tf) = (bins + 1) / 2;
  else

    y = 1;
    left_weight = 0;
    right_weight = tot_weight - weights(y);

    while left_weight < right_weight,
      y = y + 1;
      left_weight = left_weight + weights(y-1);
      right_weight = tot_weight - weights(y) - left_weight;
    end

    left_pos = y-1;   % last tf at which left_weight <
right_weight
    left_pos_weight = left_weight;

    y = left_pos;
    left_weight = left_pos_weight - weights(y);
    right_weight = tot_weight - weights(y) - left_weight;

    while left_weight <= right_weight,
      y = y + 1;
      left_weight = left_weight + weights(y-1);
      right_weight = tot_weight - weights(y) - left_weight;
    end

    right_pos = y;    % first tf at which right_weight <
left_weight
    right_pos_weight = right_weight + weights(y);
```

```
    if ( (left_pos_weight == 0) & (right_pos_weight == 0) )
      center_pos(tf) = (left_pos + right_pos) / 2;
    else
      center_pos(tf) = (left_pos*left_pos_weight +
right_pos*right_pos_weight) / (left_pos_weight +
right_pos_weight);
    end

    center_pos(tf) = center_pos(tf) - 1;  %NOTE: Subtract bottom
spacer bin

  end

end
```

9.8 CENTER2.M

```
function cen2 = center2(spectrogram,cen1);

% cen2 = center2(spectrogram,cen1);
%
% Finds spread for each timeframe of spectrogram
%   by finding the distance between the
%   centers of the two regions on either side of cen1


bins = size(spectrogram,1);
t_len = size(spectrogram,2);
cen1 = round(cen1);

for tf=1:t_len,
  cen2_1(tf) = center(spectrogram([1:cen1(tf)],tf));
  cen2_2(tf) = cen1(tf) +
center(spectrogram([cen1(tf):bins],tf)) - 1;
end

cen2 = cen2_2 - cen2_1;


% If desired, I could even look at cen2_1 and cen2_2 separately
```

9.9 COMPUTE_ERRORS.M

```
% ****************************************
% *****  C O M P U T E   E R R O R S  *****
% ****************************************
%
% Input: fully defined chunk, fully defined buffer
% Output: all error curves


% Compute sliding subtraction error curves
err_acvol = subtract_cutoff(spchunk_acvol, buff_acvol,
spchunk_vol, temp_vol_noise_thresh);
err_accen1 = subtract_cutoff(spchunk_accen1, buff_accen1,
spchunk_vol, temp_vol_noise_thresh);
err_accen2 = subtract_cutoff(spchunk_accen2, buff_accen2,
spchunk_vol, temp_vol_noise_thresh);
err_aczcross = subtract_cutoff(spchunk_aczcross, buff_aczcross,
spchunk_vol, temp_vol_noise_thresh);

% Find actual average errors for each chunk
avg_act_err_acvol(chunk_num) = mean(err_acvol);
avg_act_err_accen1(chunk_num) = mean(err_accen1);
avg_act_err_accen2(chunk_num) = mean(err_accen2);
avg_act_err_aczcross(chunk_num) = mean(err_aczcross);

% Find actual minimum errors for each
min_act_err_acvol(chunk_num) = min(err_acvol);
min_act_err_accen1(chunk_num) = min(err_accen1);
min_act_err_accen2(chunk_num) = min(err_accen2);
min_act_err_aczcross(chunk_num) = min(err_aczcross);

% Find actual difference between mimimum and threshold error
thresh_pass_acvol(chunk_num) = min_act_err_acvol(chunk_num) -
errs_thresh_acvol(chunk_num);
thresh_pass_accen1(chunk_num) = min_act_err_accen1(chunk_num) -
errs_thresh_accen1(chunk_num);
thresh_pass_accen2(chunk_num) = min_act_err_accen2(chunk_num) -
errs_thresh_accen2(chunk_num);
thresh_pass_aczcross(chunk_num) =
min_act_err_aczcross(chunk_num) -
errs_thresh_aczcross(chunk_num);
```

9.10 CREATE_SIGNAL.M

```
% ********************************************
% *******  C R E A T E   S I G N A L  **********
% ********************************************

disp('*******  C R E A T E   S I G N A L  **********')
disp('Remember to assign SIGNAL_NAME and JINGLE_NAME')
disp('Reset MAX_AMPL and VOL_ and ZC_NOISE_THRESH if necessary')


fftwin =  kaiser(1024, 7.8573);   % 1024-pt Kaiser window for
80db dropoff
dc_window = [zeros(1,40) 1 ones(1,40)] / 41;
dc_len = 40;     % Number of points actually used in averaging
          % (not necessarily length of dc_window)
sig_max_ampl = max(signal)  %should remember max of preceding
broadcast
sig_fft_thresh = 0.3/1.6 * sig_max_ampl;  %noise floor for -40db
noise
sig_zc_noise_thresh = 0.01 * sig_max_ampl;
sig_vol_noise_thresh = 0.01 * sig_max_ampl;


% Note: Don't necessarily have to use same dc window for
everything

sig_len = floor( (length(signal) - 1024) / 512 + 1 );

sig_vol = volume(signal,1024,512);
[sig_acvol, sig_dcvol] = acdc([], sig_vol, [], dc_window);
sig_acvol = sig_acvol ./ sig_dcvol;     % Normalize AC

sig_zcross =
zerocross(signal([1:sig_len*512+512]),1024,512,sig_zc_noise_thre
sh);
[sig_aczcross, sig_dczcross] = acdc([], sig_zcross, [],
dc_window);


sig_fft = win_fft(signal,1024,fftwin,512);
clear signal
sig_mag = abs(sig_fft);
clear sig_fft
sig_mag = sig_mag([1:512],[1:sig_len]);
sig_mag = sig_mag .* onesthresh(sig_fft_thresh,sig_mag);


sig_cen1 = center(sig_mag);
[sig_accen1, sig_dccen1] = acdc([], sig_cen1, [], dc_window);

sig_cen2 = center2(sig_mag,sig_cen1);
[sig_accen2, sig_dccen2] = acdc([], sig_cen2, [], dc_window);
```

```
sig_len = sig_len - dc_len;
sig_mag = sig_mag([1:512],[1:sig_len]);
sig_vol = sig_vol([1:sig_len]);
sig_dcvol = sig_dcvol([1:sig_len]);
sig_acvol = sig_acvol([1:sig_len]);
sig_cen1 = sig_cen1([1:sig_len]);
sig_dccen1 = sig_dccen1([1:sig_len]);
sig_accen1 = sig_accen1([1:sig_len]);
sig_cen2 = sig_cen2([1:sig_len]);
sig_dccen2 = sig_dccen2([1:sig_len]);
sig_accen2 = sig_accen2([1:sig_len]);
sig_zcross = sig_zcross([1:sig_len]);
sig_dczcross = sig_dczcross([1:sig_len]);
sig_aczcross = sig_aczcross([1:sig_len]);


sig_good_points = onesthresh(sig_vol_noise_thresh, sig_vol);
sig_good_points = onesthresh(2.9, timeavg(sig_good_points,[1 1
1]));

sig_accen1 = sig_accen1 .* sig_good_points;
sig_accen2 = sig_accen2 .* sig_good_points;
sig_aczcross = sig_aczcross .* sig_good_points;
```

9.11 CREATE_TEMPLATE.M

```
% ********************************************
% *******  C R E A T E   T E M P L A T E  ******
% ********************************************

disp('*******  C R E A T E   T E M P L A T E  ******');
disp('Remember to assign TEMPLATE_NAME and JINGLE_NAME');


fftwin =  kaiser(1024, 7.8573);   % 1024-pt Kaiser window for
80db dropoff
dc_window = [zeros(1,40) 1 ones(1,40)] / 41;
dc_len = 40;     % Number of points actually used in averaging
          % (not necessarily length of dc_window)
temp_max_ampl = max(template)
temp_fft_thresh = 0.3/1.6 * temp_max_ampl;  %noise floor for -
40db noise
temp_zc_noise_thresh = 0.01 * temp_max_ampl;


% Note: Don't necessarily have to use same dc window for
everything


temp_len = floor( (length(template) - 1024) / 512 + 1 );

temp_vol = volume(template,1024,512);
[temp_acvol, temp_dcvol] = acdc([], temp_vol, [], dc_window);
temp_acvol = temp_acvol ./ temp_dcvol;     % Normalize AC

temp_zcross =
zerocross(template([1:temp_len*512+512]),1024,512,temp_zc_noise_
thresh);
[temp_aczcross, temp_dczcross] = acdc([], temp_zcross, [],
dc_window);


temp_fft = win_fft(template,1024,fftwin,512);
clear template
temp_mag = abs(temp_fft);
clear temp_fft
temp_mag = temp_mag([1:512],[1:temp_len]);
temp_mag = temp_mag .* onesthresh(temp_fft_thresh,temp_mag);


temp_cen1 = center(temp_mag);
[temp_accen1, temp_dccen1] = acdc([], temp_cen1, [], dc_window);

temp_cen2 = center2(temp_mag,temp_cen1);
[temp_accen2, temp_dccen2] = acdc([], temp_cen2, [], dc_window);
```

```
temp_len = temp_len - dc_len;
temp_mag = temp_mag([1:512],[1:temp_len]);
temp_vol = temp_vol([1:temp_len]);
temp_dcvol = temp_dcvol([1:temp_len]);
temp_acvol = temp_acvol([1:temp_len]);
temp_cen1 = temp_cen1([1:temp_len]);
temp_dccen1 = temp_dccen1([1:temp_len]);
temp_accen1 = temp_accen1([1:temp_len]);
temp_cen2 = temp_cen2([1:temp_len]);
temp_dccen2 = temp_dccen2([1:temp_len]);
temp_accen2 = temp_accen2([1:temp_len]);
temp_zcross = temp_zcross([1:temp_len]);
temp_dczcross = temp_dczcross([1:temp_len]);
temp_aczcross = temp_aczcross([1:temp_len]);


temp_vol_noise_thresh = 0.01 * temp_max_ampl;    % -40db noise
threshold
temp_good_points = onesthresh(temp_vol_noise_thresh, temp_vol);
temp_good_points = onesthresh(2.9, timeavg(temp_good_points,[1 1
1]));
temp_accen1 = temp_accen1 .* temp_good_points;
temp_accen2 = temp_accen2 .* temp_good_points;
temp_aczcross = temp_aczcross .* temp_good_points;
```

9.12 DETERMINE_THRESHS.M

```
% ******************************************************
% *****  D E T E R M I N E   T H R E S H O L D S  *****
% ******************************************************
%
% Compares each template signature chunk with surrounding region
of
%    corresponding artificially pre-distorted signal signatures
%
% Input: Output of errors4 (used earlier by order_chunks)
% Output: Average, Minimum, and Threshold error
%         for each chunk of each signature


errs = errs_acvol;
errs_thresh = zeros(1,size(errs,1));
avg_error = zeros(1,size(errs,1));
min_error = zeros(1,size(errs,1));
for n = 1:size(errs,1)
  sub_range = [start_sub(n):end_sub(n)];
  dip_pos = find(sub_range == (n-1)*err_win_jump+1 );
  errs_thresh(n) = threshold(errs(n,sub_range), thresh_width,
dip_pos);
  avg_error(n) = mean(errs(n,sub_range));
  min_error(n) = min(errs(n,sub_range));
end
errs_thresh_acvol = errs_thresh;
avg_pre_err_acvol = avg_error;
min_pre_err_acvol = min_error;


errs = errs_accen1;
errs_thresh = zeros(1,size(errs,1));
avg_error = zeros(1,size(errs,1));
min_error = zeros(1,size(errs,1));
for n = 1:size(errs,1)
  sub_range = [start_sub(n):end_sub(n)];
  dip_pos = find(sub_range == (n-1)*err_win_jump+1 );
  errs_thresh(n) = threshold(errs(n,sub_range), thresh_width,
dip_pos);
  avg_error(n) = mean(errs(n,sub_range));
end
errs_thresh_accen1 = errs_thresh;
avg_pre_err_accen1 = avg_error;
min_pre_err_accen1 = min_error;


errs = errs_accen2;
errs_thresh = zeros(1,size(errs,1));
avg_error = zeros(1,size(errs,1));
min_error = zeros(1,size(errs,1));
for n = 1:size(errs,1)
```

```
  sub_range = [start_sub(n):end_sub(n)];
  dip_pos = find(sub_range == (n-1)*err_win_jump+1 );
  errs_thresh(n) = threshold(errs(n,sub_range), thresh_width,
dip_pos);
  avg_error(n) = mean(errs(n,sub_range));
end
errs_thresh_accen2 = errs_thresh;
avg_pre_err_accen2 = avg_error;
min_pre_err_accen2 = min_error;



errs = errs_aczcross;
errs_thresh = zeros(1,size(errs,1));
avg_error = zeros(1,size(errs,1));
min_error = zeros(1,size(errs,1));
for n = 1:size(errs,1)
  sub_range = [start_sub(n):end_sub(n)];
  dip_pos = find(sub_range == (n-1)*err_win_jump+1 );
  errs_thresh(n) = threshold(errs(n,sub_range), thresh_width,
dip_pos);
  avg_error(n) = mean(errs(n,sub_range));
end
errs_thresh_aczcross = errs_thresh;
avg_pre_err_aczcross = avg_error;
min_pre_err_aczcross = min_error;
```

9.13 ERRORS4.M

```
% ERRORS4  Provides local error curves in segments that
%               are 3 err_win_lens long

% Inputs: acvol, accen1, accen2, aczcross, temp_vol,
err_win_len, err_win_jump, vol_noise_thresh
% Outputs: errs_acvol, errs_accen1, errs_accen2, errs_aczcross


errors_method = 'errors4';

disp('Change error functions if necessary.');

t_len = min([size(temp_accen1,2) size(sig_accen1,2)]);          %
no. of timeframes
n_len = floor( (t_len - err_win_len) / err_win_jump + 1 );  %
no. of chunks
sub_len = t_len - err_win_len + 1;

errs_acvol = zeros(n_len,sub_len);
errs_accen1 = zeros(n_len,sub_len);
errs_accen2 = zeros(n_len,sub_len);
errs_aczcross = zeros(n_len,sub_len);


% ww = region of interest of template signature
% sig_ww = region of interest of input signal signature

n = 0;
for t = err_win_len:err_win_jump:t_len
  n = n + 1;                        % chunk number
  win_start = (t-err_win_len+1);
  preproc_location(n) = win_start;   % PREPROC_LOCATION
  disp(win_start);
  ww = [win_start:t];
  sig_ww = [ max([50 (win_start-err_win_len)]) :
min([(t+err_win_len) t_len]) ];
  % REAL STUFF DOESN'T START TILL SAMPLE 50 !!!
  sub_len = length(sig_ww) - err_win_len + 1;
  start_sub(n) = min(sig_ww);
  end_sub(n) = start_sub(n) + sub_len - 1;
  sub_range = [start_sub(n):start_sub(n) + sub_len - 1];

%  temp_acvol_slp = slopesum(temp_acvol(ww), good_points(ww));
%  temp_accen1_slp = slopesum(temp_accen1(ww), good_points(ww));
%  temp_accen2_slp = slopesum(temp_accen2(ww), good_points(ww));
%  temp_aczcross_slp = slopesum(temp_aczcross(ww),
good_points(ww));

  errs_acvol(n,[sub_range]) = subtract_cutoff(temp_acvol(ww),
sig_acvol(sig_ww), temp_vol(ww), temp_vol_noise_thresh);
```

```
  errs_accen1(n,[sub_range]) = subtract_cutoff(temp_accen1(ww),
sig_accen1(sig_ww), temp_vol(ww), temp_vol_noise_thresh);
  errs_accen2(n,[sub_range]) = subtract_cutoff(temp_accen2(ww),
sig_accen2(sig_ww), temp_vol(ww), temp_vol_noise_thresh);
  errs_aczcross(n,[sub_range]) =
subtract_cutoff(temp_aczcross(ww), sig_aczcross(sig_ww),
temp_vol(ww), temp_vol_noise_thresh);


% "Power" of AC template signatures may serve as a measure of
%     volatility in the signature and may thus be used in the
%     future to aid determination of thresholds.

  temp_acvol_pwr(n) =
volume(temp_acvol(ww),err_win_len,err_win_jump); % USING
VOLUME, NOT WEIGHTED_POWER
  temp_accen1_pwr(n) =
weighted_power(temp_accen1(ww),err_win_len,err_win_jump,temp_vol
(ww),temp_vol_noise_thresh);
  temp_accen2_pwr(n) =
weighted_power(temp_accen2(ww),err_win_len,err_win_jump,temp_vol
(ww),temp_vol_noise_thresh);
  temp_aczcross_pwr(n) =
volume(temp_aczcross(ww),err_win_len,err_win_jump); % USING
VOLUME, NOT WEIGHTED_POWER

end
```

9.14 ERROR4B.M

```
% ERRORS4B Provides error curves for ENTIRE length of signal

% Inputs: acvol, accen1, accen2, aczcross, temp_vol,
err_win_len, err_win_jump, vol_noise_thresh
% Outputs: errs_acvol, errs_accen1, errs_accen2, errs_aczcross


disp('Change error functions if necessary.');

t_len = min([size(temp_accen1,2) size(sig_accen1,2)]);          %
no. of timeframes
n_len = floor( (t_len - err_win_len) / err_win_jump + 1 );   %
no. of chunks
sub_len =  t_len - err_win_len + 1;


errs_acvol = zeros(n_len,sub_len);
errs_accen1 = zeros(n_len,sub_len);
errs_accen2 = zeros(n_len,sub_len);
errs_aczcross = zeros(n_len,sub_len);


% ww = region of interest of template signature
% sig_ww = region of interest of input signal signature

sig_ww = [1:t_len];              % t_len can maybe changed to
size(sig_accen1,2)
n = 0;
for t = err_win_len:err_win_jump:t_len     % t_len can maybe
changed to size(temp_accen1,2)
  n = n + 1;                     % chunk number
  win_start = (t-err_win_len+1);
  disp(win_start);
  ww = [win_start:t];
  start_chunk(n) = win_start;
  end_chunk(n) = t;
  start_sub(n) = 1;
  end_sub(n) = sub_len;


  errs_acvol(n,[1:sub_len]) = subtract_cutoff(temp_acvol(ww),
sig_acvol(sig_ww), temp_vol(ww), temp_vol_noise_thresh);
  errs_accen1(n,[1:sub_len]) = subtract_cutoff(temp_accen1(ww),
sig_accen1(sig_ww), temp_vol(ww), temp_vol_noise_thresh);
  errs_accen2(n,[1:sub_len]) = subtract_cutoff(temp_accen2(ww),
sig_accen2(sig_ww), temp_vol(ww), temp_vol_noise_thresh);
  errs_aczcross(n,[1:sub_len]) =
subtract_cutoff(temp_aczcross(ww), sig_aczcross(sig_ww),
temp_vol(ww), temp_vol_noise_thresh);
```

```
% "Power" of AC template signatures may serve as a measure of
%     volatility in the signature and may thus be used in the
%     future to aid determination of thresholds.

  temp_acvol_pwr(n) =
volume(temp_acvol(ww),err_win_len,err_win_jump); % USING
VOLUME, NOT WEIGHTED_POWER
  temp_accen1_pwr(n) =
weighted_power(temp_accen1(ww),err_win_len,err_win_jump,temp_vol
(ww),temp_vol_noise_thresh);
  temp_accen2_pwr(n) =
weighted_power(temp_accen2(ww),err_win_len,err_win_jump,temp_vol
(ww),temp_vol_noise_thresh);
  temp_aczcross_pwr(n) =
volume(temp_aczcross(ww),err_win_len,err_win_jump); % USING
VOLUME, NOT WEIGHTED_POWER

end
```

9.15 FIND_UNIQUE.M

```
% FIND_UNIQUE   Find uniqueness values for each chunk
%
% Input: acvol, accen1, accen2, aczcross, pure_vol, err_win_len,
err_win_jump, vol_noise_thresh
% Output: uniqueness


t_len = size(sig_accen1,2);    % length of input signal
signature in timeframes

for n = 1:size(errs_acvol,1)   % for each chunk
  sub_range = [start_sub(n):end_sub(n)];
  dip_pos = find(sub_range == (n-1)*err_win_jump+1 );      %
Expected pos of error dip
  disp(sprintf('\nMinPos should be at: %d', dip_pos));

  err_acvol = errs_acvol(n,sub_range);
  err_accen1 = errs_accen1(n,sub_range);
  err_accen2 = errs_accen2(n,sub_range);
  err_aczcross = errs_aczcross(n,sub_range);


  % If actual pos of minimum coincides with expected pos (w/in 3
timeframes),
  %   then uniqueness = difference between two lowest local
minima

  [minvals, minpos] = minpeaks(err_acvol); % find local minima
  mins = sort(minvals);
  minpos_error = abs( minpos(find(minvals == mins(1))) - dip_pos
);
  if minpos_error > 3
    disp(sprintf('  acvol minpos_error = %d',minpos_error));
    uniqueness_acvol(n) = 0;
  else
    uniqueness_acvol(n) = mins(2) - mins(1);
  end


  [minvals, minpos] = minpeaks(err_accen1);
  mins = sort(minvals);
  minpos_error = abs( minpos(find(minvals == mins(1))) - dip_pos
);
  if minpos_error > 3
    disp(sprintf('  accen1 minpos_error = %d',minpos_error));
    uniqueness_accen1(n) = 0;
  else
    uniqueness_accen1(n) = mins(2) - mins(1);
  end
```

```
  [minvals, minpos] = minpeaks(err_accen2);
  mins = sort(minvals);
  minpos_error = abs( minpos(find(minvals == mins(1))) - dip_pos
);
  if minpos_error > 3
    disp(sprintf('  accen2 minpos_error = %d',minpos_error));
    uniqueness_accen2(n) = 0;
  else
    uniqueness_accen2(n) = mins(2) - mins(1);
  end


  [minvals, minpos] = minpeaks(err_aczcross);
  mins = sort(minvals);
  minpos_error = abs( minpos(find(minvals == mins(1))) - dip_pos
);
  if minpos_error > 3
    disp(sprintf('  aczcross minpos_error = %d',minpos_error));
    uniqueness_aczcross(n) = 0;
  else
    uniqueness_aczcross(n) = mins(2) - mins(1);
  end

end

uniqueness_sum = uniqueness_acvol + uniqueness_accen1 +
uniqueness_accen2 + uniqueness_aczcross;

uniqueness = uniqueness_acvol .* uniqueness_accen1 .*
uniqueness_accen2 .* uniqueness_aczcross;
```

9.16 FLAG_COUNTER.M

```
function flag_count = flag_counter(flaglist);

% flag_count = flag_counter(flaglist)
%
% Counts number contiguous flag-set regions in flaglist


flag_len = length(flaglist);
flag_ons = onesthresh(0, flaglist([2:flag_len]) -
flaglist([1:flag_len-1]) );
flag_count = sum(flag_ons);

if flaglist(1) == 1
  flag_count = flag_count + 1;
end
```

9.17 GUESS_SPEED.M

```
% ************************************
% *****  G U E S S   S P E E D *******
% ************************************
%
% Input: fully defined chunk, buff_vol
% Output: speed, fully defined rescaled spchunk (speed-shifted
chunk)


slowchunk_len = round(chunk_len / 0.90) - 1;  % the slowest
chunk will be
slowerr_len = buff_len - slowchunk_len + 1;   %    the longest
also.


spchunk_acvol = zeros(1,slowchunk_len);          % Intialize
variables
err_acvols = zeros(11,slowerr_len);

% Match rescaled volume signatures to determine speed

min_err = +inf;
y = 0;
for sp = 0.90:0.02:1.10;
  y = y + 1;
  spchunk_len = round(chunk_len / sp) - 1;  % "-1" to not go
past chunk_acvol index limit
  spchunk_acvol([1:spchunk_len]) =
chunk_acvol(round([1:spchunk_len]*sp));
  spchunk_vol([1:spchunk_len]) =
chunk_vol(round([1:spchunk_len]*sp));
  err_len = buff_len - spchunk_len + 1;
  err_acvols(y,[1:err_len]) =
subtract_cutoff(spchunk_acvol([1:spchunk_len]), buff_acvol,
spchunk_vol([1:spchunk_len]), temp_vol_noise_thresh);
  if min(err_acvols(y,[1:err_len])) < min_err
    min_err = min(err_acvols(y,[1:err_len]));
    speedbin = y;
    speed = sp;
  end
end

disp(sprintf('Jingle Speed: %d',speed));

spchunk_len = round(chunk_len / speed) - 1;


% Initialize Speed-shifted chunk signatures

clear spchunk_vol spchunk_accen1 spchunk_accen2 spchunk_aczcross
spchunk_accen1 = zeros(1,spchunk_len);
```

```
spchunk_accen2 = zeros(1,spchunk_len);
spchunk_aczcross = zeros(1,spchunk_len);
spchunk_vol = zeros(1,spchunk_len);
spchunk_acvol = zeros(1,spchunk_len);


% Zero-Order Hold type rescaling of template chunks

spchunk_accen1([1:spchunk_len]) = speed *
chunk_accen1(round([1:spchunk_len]*speed));
spchunk_accen2([1:spchunk_len]) = speed *
chunk_accen2(round([1:spchunk_len]*speed));
spchunk_aczcross([1:spchunk_len]) = speed *
chunk_aczcross(round([1:spchunk_len]*speed));

spchunk_vol([1:spchunk_len]) =
chunk_vol(round([1:spchunk_len]*speed));
spchunk_acvol([1:spchunk_len]) =
chunk_acvol(round([1:spchunk_len]*speed));
```

9.18 JINGLE_DETECT.M

```
% ************************************************************
% **////////////////////////////////////////////////////////**
% **//////  J  I  N  G  L  E     D  E  T  E  C  T  ///////**
% **////////////////////////////////////////////////////////**
% ************************************************************

% Load Template and Input Signal Data

  %template = bin2mat....
  %create_template
  %clear template
  %save...

  %signal = bin2mat...
  %create_signal
  %clear signal
  %save...

  %signal = bin2mat...
  %create_signal
  %clear signal
  %save...

%=======================

load ../jingles/livingindanger_cd_template.mat
load ../jingles/livingindanger_ND_sig.mat


err_win_len = 200;    % Window length (in timeframes) for
sliding subtraction
err_win_jump = 100;   % Shift between starting pos's of
successive windows
order_chunks          % Establish chunk_order according to
uniqueness

thresh_width = 10;       % Width of error dip (N=10) for determ
of thresh's
determine_threshs

save ../data/livingindanger_preproc.mat chunk_order
errs_thresh_accen1 errs_thresh_accen2 errs_thresh_acvol
errs_thresh_aczcros errs_accen1 errs_accen2 errs_acvol
errs_aczcross

load ../jingles/livingindanger_mtv_signal.mat


% Find Trigger Chunk
```

```
chunk_detect_map = zeros(size(chunk_order));     % Initialize
stats
existence = 0;
minflag_pos = zeros(size(chunk_order));
minflag_val = zeros(size(chunk_order));

while existence == 0,
n = 1;                % Trigger chunk = Most unique chunk
  chunk_range =
[start_chunk(chunk_order(n)):end_chunk(chunk_order(n))];
  assign_chunk;

  pos_guess = -1;
  buff_range = [1:sig_len];
  assign_buffer;

  guess_speed;

  compute_errors;

  chunk_num = chunk_order(n);
  pos_tolerance = 5;
  apply_threshs;

end


chunk_detect_map(chunk_order(1)) = 1; %Update stats for detected
trigger chunk
old_position = position;
old_chunk_num = chunk_num;
old_speed = speed;


% Find Rest of the Chunks

for n = 2:length(chunk_order)

  chunk_num = chunk_order(n);
  chunk_range = [start_chunk(chunk_num):end_chunk(chunk_num)];
  assign_chunk;

  pos_guess = round(old_position + (chunk_num - old_chunk_num) *
err_win_jump * old_speed);
  buff_start = max([1 (pos_guess - err_win_len)]);
  buff_end = min([sig_len (pos_guess+ 2 * err_win_len)]);
  buff_range = [buff_start:buff_end];
  assign_buffer;

  guess_speed;

  compute_errors;

  chunk_num = chunk_order(n);
  pos_tolerance = 5;
```

```
  apply_threshs;
  disp(sprintf('        Guessed at %d',pos_guess));
  disp(sprintf('        Minimum error at %d\n',minpos_all));


  if existence == 1        % Update stats for fully successful
detection
    old_position = position;
    old_chunk_num = chunk_num;
    old_speed = speed;
  end

  chunk_detect_map(chunk_num) = existence;

end


confidence = mean(chunk_detect_map);
```

9.19 MINPEAKS.M

```
function [val, pos] = minpeaks(errfunc);

% [val, pos] = minpeaks(errfunc);
%
% Returns the values and positions of the
%   local minima of an error curve


avgwindow = ones(1,41) / 41;

[ac, dc] = acdc([], errfunc, [], avgwindow);

t_len = length(errfunc);

n = 0;
flag = +1;
for t = 1:t_len,
  if ac(t) < 0
    if flag == +1
      n = n+1;
      startpos = t;
      flag = -1;
    end
  elseif flag == -1     % and ac(t) >= 0
    [val(n), i] = min(errfunc([startpos:t]));
    pos(n) = startpos + i - 1;
    flag = +1;
  end
end
```

9.20 ONESTHRESH.M

```
function onesmap = onesthresh(level,Z)

% onesmap = onesthresh(level,Z)
%
% Makes all values above the threshold level
%   go to one and all else goes to zero.

onesmap = spones(abs(Z - level) + (Z-level));

onesmap = full(onesmap);
```

9.21 ORDER_CHUNKS.M

```
% ***********************************
% ****  O R D E R   C H U N K S  ****
% ***********************************
%
% Input: err_win_len, err_win_jump, ac template signatures, ac
signal signatures
% Output: chunk_order, error curves produced by errors4.m


errors4b   %Compares each template chunk with ENTIRE input
signal
find_unique
long_uniqueness = uniqueness;

errors4   %Compares each template chunk with surrounding region
of input signal
find_unique
short_uniqueness = uniqueness;


% Find Broad Uniqueness

uniqueness = long_uniqueness .* short_uniqueness;

[y, unique_chunks] = sort(uniqueness);

% Reverse uniqueness list into descending order
unique_chunks = unique_chunks([length(unique_chunks):-1:1]);
y = y([length(y):-1:1]);

unique_chunks = unique_chunks(find(y > 0));   % Keeps only Non-
Zero Uniqueness


% Find Local Uniqueness

short_uniqueness(unique_chunks) = zeros(size(unique_chunks));

[y onlyshort_unique_chunks] = sort(short_uniqueness);

onlyshort_unique_chunks =
onlyshort_unique_chunks([length(onlyshort_unique_chunks):-1:1]);
y = y([length(y):-1:1]);

onlyshort_unique_chunks = onlyshort_unique_chunks(find(y>0));



uniqueness_sum(unique_chunks) = zeros(size(unique_chunks));
uniqueness_sum(onlyshort_unique_chunks) =
zeros(size(onlyshort_unique_chunks));
```

```
[y, notunique_chunks] = sort(uniqueness_sum);

notunique_chunks = notunique_chunks([length(notunique_chunks):-
1:1]);
y = y([length(y):-1:1]);

notunique_chunks = notunique_chunks(find(y > 0));


% Produce chunk_order

chunk_order = [ unique_chunks onlyshort_unique_chunks];
number_of_chunks = length([ unique_chunks
onlyshort_unique_chunks notunique_chunks]);
```

9.22 SUBTRACT_CUTOFF.M

```
function err = subtract_cutoff(chunk, buffer, chunk_vol,
vol_noise_thresh);

% err = subtract_cutoff(chunk, buffer, chunk_vol,
vol_noise_thresh);
%
% Returns error curve resulting from sliding subtraction of
chunk & buffer
%    signatures.  Only chunk timeframes having volume greater
than the
%    vol_noise_thresh will be considered.

good_pts = onesthresh(vol_noise_thresh, chunk_vol);

chunk_len = length(chunk);
buffer_len = length(buffer);
tt = 0;
for t = chunk_len:buffer_len,
  tt = tt + 1;
  err(tt) = sum( abs(chunk - buffer([t-chunk_len+1:t])) .*
good_pts);
end

err = err / sum(good_pts);
```

9.23 THRESHOLD.M

```
function thresh = threshold(err, points, dip_pos);

% thresh = threshold(err, points, dip_pos)
%
%   finds thresh level below which there is the given number of
points
%   of the amplitude distribution between the thresh level and
%   the err amplitude at dip_pos (where pos is hopefully the
lowest point)


err_len = length(err);

[sorted_err, i] = sort(err);

dip_i = find(i == dip_pos);
if dip_i ~= 1
  disp(sprintf('%d is not the lowest pt., it is %d.. off by
%d',dip_pos,i(1),dip_pos-i(1)));
end

thresh = sorted_err(min([(dip_i + points) err_len]));
```

9.24 TIMEAVG.M

```
function tfsx2 = timeavg(tfsx,tri)

% tfsx2 = timeavg(tfsx,tri)
%
% Smooths out any 1 or 2 dimensional tfsx in one dimension (the
time dimension
%    for spectrograms) by using the averaging window tri.
%
% NOTE: since tri contains the weighting for the "averaging"
%       it is unlike convolution in that it is "not-flip &
slide"


tri([length(tri):-1:1]) = tri;  %Pre-flip for flip in conv

oldlen = size(tfsx,2);
newlen = oldlen + length(tri) - 1;

for freq = 1:size(tfsx,1),
  tfsx2(freq,[1:newlen]) = conv(tfsx(freq,[1:oldlen]),tri);
  %tfsx2(freq,[1:oldlen]) =
filtfilt(tri,1,tfsx(freq,[1:oldlen]));
  % filtfilt is DANGEROUS, see help-pages -- it does extra
filtering.
end

newextra = (newlen - oldlen) / 2;

tfsx2 = tfsx2([1:size(tfsx2,1)],[1+newextra:newlen-newextra]);
```

9.25 VOLUME.M

```
function vol = volume(signal, win_len, win_jump);

% vol = volume(signal, win_len, win_jump)
%
% Produces a signature of the RMS volumes over
%   windows of length win_len spread win_jump apart

sig_len = length(signal);

vol_len = floor((sig_len - win_len)/ win_jump) + 1;
vol = zeros(1,vol_len);

tf = 0;
for t = win_len:win_jump:sig_len
  tf = tf+1;
  vol(tf) = sqrt( sum(signal([t-win_len+1:t]).^2) / win_len );
end
```

9.26 WEIGHTED_POWER.M

```
function pwr = weighted_power(signature, win_len, win_jump,
volume, vol_noise_thresh);

% pwr = weighted_power(signature, win_len, win_jump, volume,
vol_noise_thresh)
%
% Returns "power" of each chunk of given signature


signature = signature .* onesthresh(vol_noise_thresh, volume);

sig_len = length(signature);

pwr_len = floor((sig_len - win_len)/ win_jump) + 1;
pwr = zeros(1,pwr_len);

tf = 0;
for t = win_len:win_jump:sig_len
  tf = tf+1;
  pwr(tf) = sqrt( sum(signature([t-win_len+1:t]).^2) / win_len
);
end
```

9.27 WIN_FFT.M

```
function ffts = win_fft(signal, fft_len, window, win_jump);

% ffts = win_fft(signal, fft_len, window, win_jump);
%
% Produces a set of ffts of signal filtered by a window of
%   length fft_len each spread win_jump apart


signal_len = length(signal);
win_len = length(window);

if (win_len > fft_len)
  time_aliasing_flag = 1;
  disp('Time Aliasing');
  if (win_len/fft_len ~= floor(win_len/fft_len))
    disp('Window length should be a multiple of the fft
length!')
  end
end

tf_len = floor((signal_len - win_len)/win_jump) + 1;
ffts = zeros(fft_len,tf_len);

tf = 0;
for t = win_len:win_jump:signal_len
  disp(t)
  tf = tf + 1;
  sig = signal([t-win_len+1:t]) .* window;

  if time_aliasing_flag
    for n = 1:fft_len
      new_sig(n,1) = sum(sig([n:fft_len:win_len]));
    end
    sig = new_sig;
  end

  ffts([1:fft_len],tf) = fft(sig,fft_len);
end
```

9.28 ZEROCROSS.M

```
function zcross =
zerocross(signal,win_len,win_jump,noise_thresh);

% zcross = zerocross(signal,win_len,win_jump);
%
% Produces a signature of the numbers of zero-crossings over
%   windows of length win_len spread win_jump apart


signal_len = length(signal);

zcross_len = floor((signal_len - win_len)/ win_jump) + 1
zcross = zeros(1,zcross_len);

flag = zeros(size(win_len));

tf = 0;
for t = win_len:win_jump:signal_len
  disp(t)
  tf = tf+1;
  sig = signal([t-win_len+1:t]);
  signs = sign(sig);
  zc = 0;
  flag = 0;
  for tt = 1:win_len
    if abs(sig(tt)) > noise_thresh
      if abs(signs(tt) - flag) == 2
     zc = zc + 1;
      end
      flag = signs(tt);
    end
  end
  zcross(tf) = zc;
end



%  for tt = 1:win_len
%    if signs(tt) ~= 0
%      if abs(signs(tt) - flag) == 2
%    zc = zc + 1;
%      end
%      flag = signs(tt);
%    end
%  end
```

10. References

Crutchfield, E. B. *National Association of Broadcasters Engineering Handbook,* Washington, DC: Dept. of Science and Technology, National Association of Broadcasters, 1985.

S. Kadambe and G.F. Boudreaux-Bartels, "A Comparison of the Existence of "Cross Terms" in the Wigner Distribution and the Squared Magnitude of the Wavelet Transform and the Short Time Fourier Transform," *IEEE Transactions on Signal Processing,* vol. 40, no. 10, pp. 2498-2517, Oct. 1992.

Hess, Wolfgang. *Pitch Determination of Speech Signals: Algorithms and Devices,* New York: Springer-Verlag, 1983.

Noll, Peter. "Wideband Speed and Audio Coding," *IEEE Communications Magazine,* pp. 34-44, November 1993.