Landmark Detection for Distinctive Feature-Based Speech Recognition

by

Sharlene Anne Liu

B.S., Massachusetts Institute of Technology
(1987)

M.S., Massachusetts Institute of Technology
(1990)

Submitted to the Department of
Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

May 1995

Author_____
Department of Electrical Engineering and Computer Science
May 8, 1995

Certified by_____
Kenneth N. Stevens
Clarence J. LeBel Professor of Electrical Engineering
Thesis Supervisor

Accepted by_____
Frederic R. Morgenthaler
Chairperson, Departmental Committee on Graduate Students

Landmark Detection for Distinctive Feature-Based Speech Recognition

by

Sharlene Anne Liu

Submitted to the Department of Electrical Engineering and Computer Science
on May 8, 1995 in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis is a component of a proposed knowledge-based speech recognition system which uses *landmarks* to guide the search for *distinctive features.* In an utterance, landmarks identify localized regions where the acoustic manifestations of the linguistically-motivated distinctive features are most salient. This thesis describes an algorithm for automatically detecting landmarks associated with segments having abrupt acoustics. As a consequence of landmark detection, the algorithm also provides hypotheses about the underlying broad phonetic class at each landmark. The algorithm is hierarchically-structured, and is rooted in linguistic and speech production theory. It uses several factors to detect landmarks: energy abruptness in five frequency bands and at two levels of temporal resolution, segmental duration, specific broad phonetic class constraints, and articulatory constraints.

Landmark detection experiments were performed on clean speech (including TIMIT), speech in noise, and telephone speech. On clean speech, the landmark detector performed relatively well, with a detection rate of about 90% if correct landmark type was required, and 94% if correct landmark type was not required. The insertion rate was 6%–9%. An analysis of the temporal precision of the landmark detector showed that a large majority of the landmarks were detected within 20 ms of the landmark transcription, and almost all were within 30 ms. For either speech in noise or telephone speech, performance understandably degraded due to the reduced information content in the speech signal.

For each set of experiments, the landmark detection algorithm was manually customized to the database using knowledge about speech and the operating environment. One consequence of this knowledge-driven approach is that there is no degradation in performance between what is typically called the "training" data set and the test data set. This approach also allows careful evaluation and further improvements to be made in a methodical manner.

Thesis Supervisor: Kenneth N. Stevens
Title: Clarence J. LeBel Professor of Electrical Engineering

Dedication:

To Mom and Dad

Acknowledgements

My stay at MIT, from entering as a freshman to graduating with a Ph.D., was dynamic and fulfilling. In particular, my doctoral studies were an enriching, learning experience. I would like to express my gratitude here to the people who were an integral part of this experience and who made it so pleasurable.

I am most grateful to my thesis advisor, Ken Stevens, for teaching me about speech and for providing an atmosphere in which learning is fun. His kindness, patience, and guidance formed the backbone to my research.

My thesis committee added breadth to my thesis. I thank Jim Glass for expanding my knowledge into the engineering aspects of speech recognition. It was a pleasure to discuss various aspects of phonology with Morris Halle. Thanks to Jon Allen for insightful comments on system-level issues.

The people in the Speech Communication Group are a friendly and generous bunch. They made coming into the office every morning something to look forward to! In particular, I would like to acknowledge Stefanie Shattuck-Hufnagel, who taught me about prosody. Her smile and amazing knack for understanding people were a great support. Thanks to Corine Bickley for reading a draft of my thesis and for providing a role model to work towards. I am grateful to Melanie Matthies and Seth Hall for helping me with computer issues, and to Charles Jankowski for giving me the telephone speech I needed.

Inseparably part of doing research is taking afternoon tea breaks. I would like to thank my friends for entertaining tea time discussions. Thanks to Marilyn Chen and Jeff Kuo for laughing with me, Mark Johnson for putting landmarks to use, Roeland Schaeffer for sharing my enthusiasm about seeing the world, Alice Turk for her knowledge on phonetics, and Yi Xu for his interesting views on the Mandarin language.

I am indebted to Mom and Dad, whose never-ending care gave me the ambition and the strength to succeed, and to my sisters, Joanne and Lynette, for growing with me, both professionally and personally, so that I have company every step of the way.

This research was financially supported by a grant from the National Science Foundation and by the Clarence J. LeBel fund.

Biographical Note

Sharlene Liu is originally from the tropical island of Taiwan. She immigrated to the United States with her parents and two sisters when she was 6 years old. She learned her ABC's in Washington state, and received the rest of her elementary school through high school education in the San Francisco Bay Area. Wanting to become an engineer, she went to the east coast to attend MIT. Her major in Electrical Engineering allowed her to dabble in a variety of topics. Her Bachelor's thesis was in digital design and her Master's thesis was in the area of electromagnetism. In the end, she concentrated her Ph.D. research in automatic speech recognition. She finds speech recognition research particularly interesting because it combines several of her interests: signal processing, linguistics, and bioengineering. She plans to make a career in this area. She was a student member of the Acoustical Society of America and of the Institute of Electrical and Electronics Engineers.

Interspersed throughout Sharlene's studies at MIT were diverse opportunities to work domestically and abroad. She had summer jobs at Gatan in California, IBM in Vermont, Tektronix in Oregon, and MIT Lincoln Lab in Massachusetts. In addition, she spent a year between her Master's and Ph.D. studies in the Jura Mountains of Switzerland working at Portescap, a micromechanical company with a history in watch-making. During her Ph.D. studies, she passed an enjoyable summer in Sydney, Australia (winter down under) at the National Acoustic Labs working on speech processing for hearing aid applications.

One of Sharlene's hobbies is learning languages and, in fact, she travels around the world in pursuit of this hobby. She also enjoys hiking in the mountains of the countries she visits.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 A knowledge-based approach

In a knowledge-based approach to speech recognition, knowledge about speech and the operating environment is explicitly built into the recognizer by the system designer. It is an attempt to model the human perception process. Since humans are the best speech recognizers known to us, a knowledge-based approach is a worthwhile endeavor. If successful, it should have the flexibility to work in many operating environments. A knowledge-based system is dedicated to processing speech (versus signals in general) and therefore is efficient in that sense. Because knowledge sources are explicitly incorporated, improvements to the system can be made in a directed, meaningful manner. Prosodic and segmental level knowledge can usually be added in a straightforward manner. The knowledge necessary to design such a system, however, needs to be comprehensive and the desired acoustic parameters need to be extractable.

Rather than explicitly specifying the knowledge used in a speech recognition system, a statistical approach builds speech models by training on speech data, thereby implicitly acquiring knowledge on its own. This automatic learning during the training phase makes statistical methods powerful. Much of our present ignorance about speech can be overcome in this way. Statistical methods have been successful for large-vocabulary, speaker-independent speech recognition (e.g. [Lee, 1989]). In the training phase, large amounts of data are used to cover all the possible variations of speech. Frequently-occurring speech units can be modeled well, but infrequently-occurring units will not be as robust. Because of their heavy reliance on data, statis-

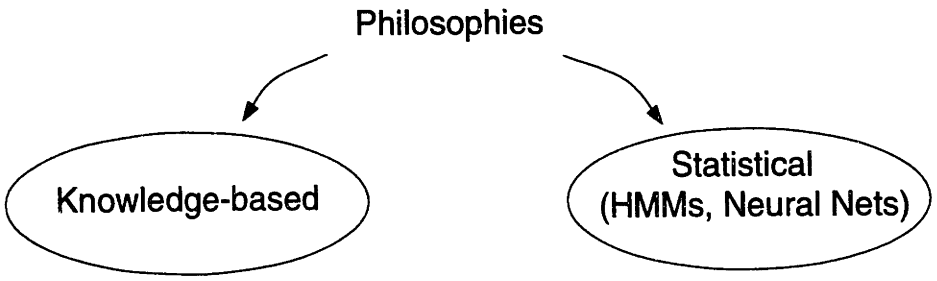

Figure 1.1: An illustration of the dichotomy in philosophies between the knowledge-based and statistical approaches.

tical methods do not generalize well to tasks for which they are not explicitly trained. For example, if the operating environment does not match the training environment (e.g. a different microphone is used, background noise is added, or speech is bandlimited to the telephone line), recognition accuracy can degrade severely. To accommodate a new operating environment, re-training or adaptation is usually required. In adverse conditions, like noisy or telephone quality environments, re-training does not necessarily help performance (e.g. [Das et al., 1993]).

Figure 1.1 illustrates the dichotomy in philosophies between a knowledge-based and a statistical approach. In reality, however, speech recognition systems use a combination of both approaches. In much of speech recognition research since the late 1980s, hidden Markov models (HMMs), a popular statistical tool, form the foundation of the system. Increasingly, artificial neural networks (ANNs), another statistical tool, are becoming popular as well. Hybrid HMM/ANN systems are being built. In these fundamentally statistical systems, knowledge sources are added whenever suitable. Examples of such knowledge sources are phone duration information, an auditory model front-end, or, more simply, the mel-frequency scale to approximate the frequency warping performed by the basilar membrane. The fundamental philosophy underlying the speech recognition system adopted in this thesis, however, is knowledge-based instead of statistical. The system will employ some statistics as a guide to making decisions but not to build the foundation of the system.

Figure 1.2 shows a block diagram of the proposed knowledge-based speech recognition system [Stevens et al., 1992b]. This system sets the framework for the research presented in this thesis. The first step is to locate *landmarks*, which are points in an utterance around which information about the underlying *distinctive*

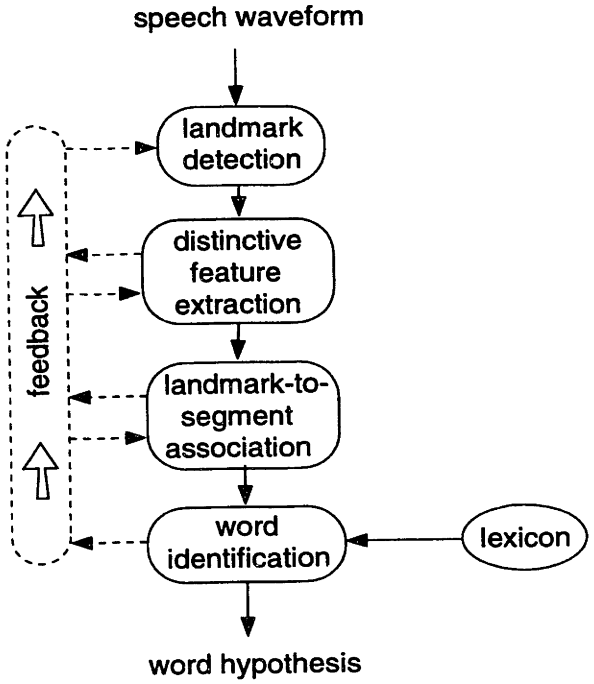

Figure 1.2: The proposed speech recognition system.

features may be extracted. Next, the distinctive features at a landmark are identified based on acoustic measurements made in the vicinity of the landmark. The landmarks and associated features are related to the underlying segments [1] and a sequence of segments is hypothesized. This sequence is matched to a lexicon whose words are directly defined in terms of features, and word hypotheses are made. Shown in dotted lines, feedback allows information from an advanced stage to be used to correct mistakes made at an earlier stage.

1.2 Distinctive features

The speech recognition system of Figure 1.2 has two noteworthy properties. The first is the chosen unit of speech, the distinctive feature. Distinctive features concisely describe the sounds of a language at a subsegmental level. They have a relatively direct relation to acoustics and articulation. They take on binary values and form a minimal set which can distinguish each segment from all others in a language

[1] A *segment* is used in this thesis to refer to a bundle of distinctive features which describe a speech sound and which have acoustic correlates. It does not refer to a physical slice of the acoustic signal.

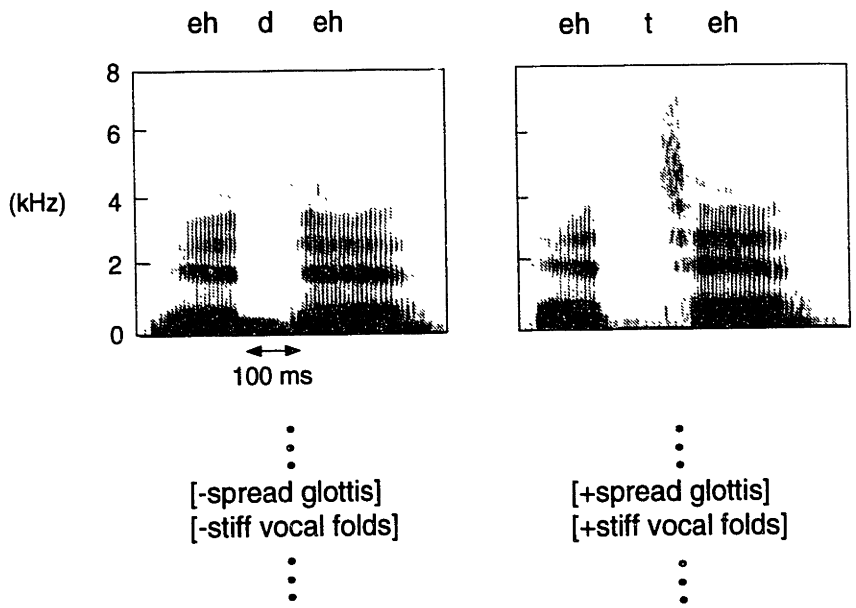

Figure 1.3: An illustration of the ease with which distinctive features describe phonemic contrast. The two segments, [d] and [t], differ only in the features [spread glottis] and [stiff vocal folds].

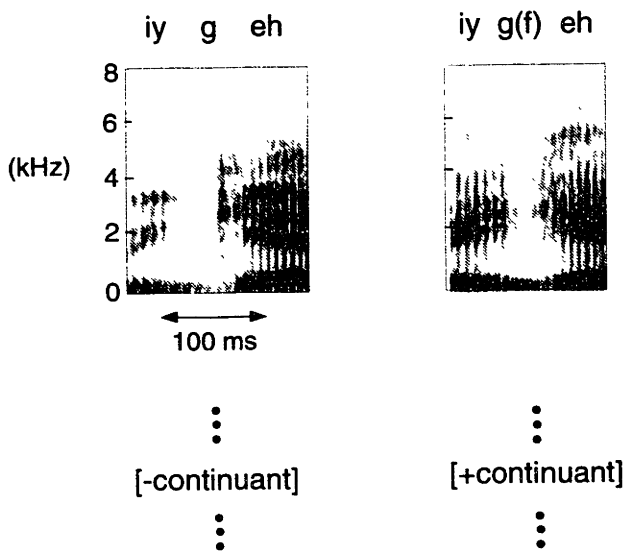

Figure 1.4: An illustration of the ease which distinctive features describe contextual variability. The feature [continuant] alone describes the difference between a canonical [g] and a velar fricative variation.

[Jakobson et al., 1952]. Figure 1.3 illustrates how distinctive features describe phonemic contrasts. In English, [d] and [t] are very similar segments; they differ only in that [d] is voiced and [t] is voiceless. In terms of distinctive features, [d] and [t] have the same features except for [spread glottis] and [stiff vocal folds]. These two segments are close in distinctive feature space, as dictated by their similarity. Another advantage of distinctive features is that they can concisely describe many of the contextual variations of a segment. These contextual variations could be due to individual speaking styles and phonological assimilation across word boundaries, for example. Figure 1.4 illustrates how distinctive features describe contextual variability with a minimal amount of modification. The canonical implementation of a [g] is with a complete closure; however, in casual speech frication noise may exist throughout the constriction due to an incomplete closure. In terms of distinctive features, this modification can be described by a change in the feature [continuant] alone. Phoneme or higher-level modeling (e.g. syllables, words) would have to have a separate model for each new modification, resulting in an explosion of the number of speech units required, as [Lee, 1989] experienced with triphone modeling.

In acoustic-phonetic processing, some researchers have shown a growing interest in using intermediate forms of representation between the speech signal and phones, rather than directly mapping from speech frames to phones. The hope among these researchers is that an intermediate form of representation with a grounding in speech knowledge will be able to capture the fine phonetic distinctions better than a direct mapping. For the intermediate representation, researchers have used acoustic features (e.g. [Phillips and Zue, 1992]), articulation-motivated representations (e.g. [De Mori and Flammia, 1993], [Deng and Sun, 1994]), and distinctive features (e.g. [Meng, 1991], [Eide, 1993]). Sometimes, hierarchy is imposed on the intermediate representation. For instance, [Windheuser, 1993] trained and tested a neural network using a set of features divided by consonant and vowel affiliation and noted that such a division gave better results than a set that was not divided. As another example, [Eide, 1993] used a two-stage process for phonetic identification: each phone was first grouped into broad phonetic classes and then the phone was identified using broad-class-dependent processing.

In a recognition system based on features, the lexicon is accessed directly in terms of bundles of distinctive features rather than phones, and there exists a hier-

	feature or articulator	v	o	t
articulator-free features	consonantal	+	−	+
	sonorant	−		−
	continuant	+		−
	strident	−		
	lateral			
primary articulator	lips	√		
	tongue blade			√
	tongue body			
articulator-bound features	round (lips)	−	+	
	anterior (blade)			+
	distributed (blade)			−
	high (body)		−	
	low (body)		−	
	back (body)		+	
	nasal (soft palate)			
	advanced tongue root (pharynx)		+	−
	constricted tongue root (pharynx)			
	spread (glottis)	−		
	constricted (glottis)	−		+
	stiff (vocal folds)			+
	slack (vocal folds)	+	+	

Table 1.1: List of distinctive features and primary articulators. An example of feature specification for segments of the word "vote" is given. An unreleased [t] is assumed.

archical relationship among these distinctive features. Table 1.1 lists the distinctive features used in the framework of this thesis [Stevens et al., 1992b]. They evolved from a set proposed by [Chomsky and Halle, 1968], who used articulatory terms to label the distinctive features, and provided their acoustic correlates. Following the formalism of [Halle and Stevens, 1991], the distinctive features in Table 1.1 are separated into *articulator-free* and *articulator-bound* features.[2]

An articulator-free feature is one which has no dedicated articulator to implement it. In other words, more than one articulator can implement it. For example, the articulator-free feature [sonorant], which describes whether or not there is pressure built up in the vocal tract, can be implemented by the lips, tongue blade, or tongue body. Other than the feature [consonantal], articulator-free features are relevant only for [+consonantal] segments. The articulator that implements [+consonantal] must also be the one that implements the rest of the articulator-free features. By their very nature, articulator-free features must always be specified in conjunction with the major articulator which is reponsible for their implementation.[3,4] Other, supporting articulators may be active in further defining the segment. The major and supporting articulators will technically be referred to as the *primary* and *secondary* articulators, respectively. From one point-of-view, only the articulators in the oral cavity can be primary articulators, i.e. lips, tongue blade, and tongue body.

In the process of landmark detection, hypotheses about some of the articulator-free features are made. The features which are affected by landmark detection are described below:

consonantal A [+consonantal] segment is formed with a tight constriction in the oral cavity, causing a rapid change in the spectrum. Stops, fricatives, nasals, and acoustically-abrupt /l/s are [+consonantal]; semivowels excluding acoustically-abrupt /l/s are [–consonantal]. In this thesis, "semivowel" will refer to /w,y,r/ and nonabrupt /l/, while "[l]" will refer to acoustically-abrupt /l/.

sonorant A [+sonorant] segment causes no pressure buildup in the vocal tract. Glot-

[2][Clements, 1985] originally pioneered the concept of feature geometry to hierarchically structure distinctive features.

[3][Sagey, 1986] first introduced the concept of a major articulator responsible for implementing a particular sound.

[4]By convention, a [–consonantal] segment has no major articulator associated with it.

tal vibration continues throughout the constriction, without a change in glottal amplitude from the adjacent vowel. Of the [+consonantal] segments, [l]s and nasals are [+sonorant], and all others are [–sonorant].

continuant A [–continuant] segment is formed with an incomplete closure in the midline of the vocal tract. The acoustic correlate is the continuation of noise throughout the segment. Of the [–sonorant] segments, fricatives are [+continuant], and all others are [–continuant].

Figure 1.5 shows two feature trees. These trees illustrate fundamental and practical constraints on the interdependence of the articulator-free features listed above. The leaves on the trees are the broad phonetic classes. An affricate is a compound segment consisting of a stop followed by a fricative. In tree (a), [sonorant] is placed above [continuant]. [–Continuant] is understood for a nasal and an [l]. From an algorithmic point-of-view, this arrangement is desirable. It separates the sonorant consonantal segments from the obstruent consonantal segments, and keeps a close relationship between fricatives and stops (weak fricatives sometimes become stop-like and velar stops sometimes become fricative-like in casual speech). Tree (a) is the one used for the formulation of the landmark detection algorithm. An alternative feature tree is shown in tree (b). [Continuant] is placed above [sonorant], and [–sonorant] is understood for a fricative. However, tree (b) is not used because it does not reflect the acoustic similarities between fricatives and stops or the acoustic dissimilarity between sonorant consonantal and obstruent consonantal segments.

In contrast to articulator-free features, an articulator-bound feature has a dedicated articulator responsible for implementing it. For example, the articulator-bound feature [round] can be implemented only by the lips. For any particular [+consonantal] segment, there are primary articulator-bound features, describing the configuration of the primary articulator, and secondary articulator-bound features, describing the configuration of the secondary articulators. Table 1.1 gives an example of feature specification for the word "vote". For [v], the lips are the primary articulator and the glottis and vocal folds are secondary articulators. The lips implement the articulator-free features [+consonantal], [–sonorant], [+continuant], and [–strident]. The primary articulator-bound feature, bound to the lips, is [–round]. The secondary articulator-bound features are [–spread] and [–constricted], bound to the glottis, and [+slack],

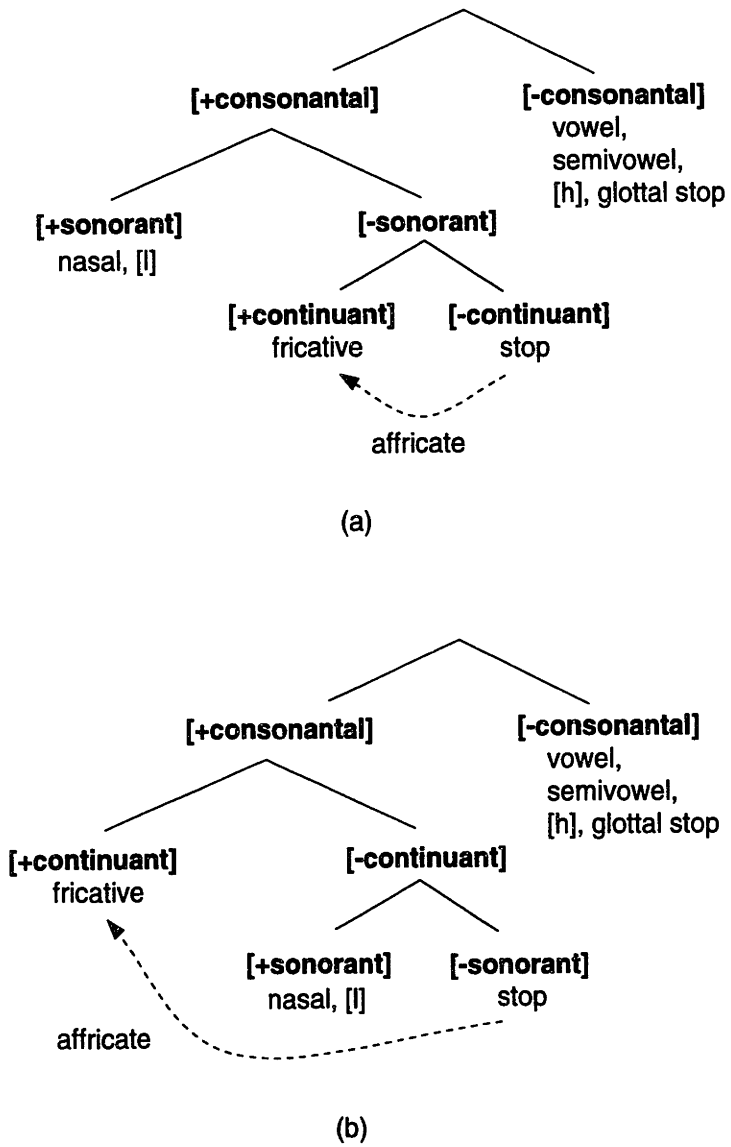

Figure 1.5: Two alternative feature trees for articulator-free features. (a) [Sonorant] is placed above [continuant]. This tree is the preferred one for landmark detection. In (b), [continuant] is placed above [sonorant].

bound to the vocal folds.[5] For any particular segment, only a subset of the features needs to be specified. The unspecified features, illustrated by the blanks in Table 1.1, are either irrelevant or redundant. Note that, because [v] and [t] are [+consonantal], further articulator-free features are specified and a primary articulator is checked, whereas, because [o] is [–consonantal], no further articulator-free features or primary articulator are specified. An approach to articulator-bound feature identification can be found in [Johnson, 1994].

1.3 Landmark detection

The second noteworthy property of the proposed speech recognition system is landmarks. They are a guide to the presence of underlying segments, which organize distinctive features into bundles. Landmarks are times in an utterance when the acoustic correlates of distinctive features are most salient. They mark perceptual foci and articulatory targets. [Stevens, 1985] has suggested that, for some phonetic contrasts, a listener focuses on landmarks to get the acoustic cues necessary for deciphering the underlying distinctive features. [Furui, 1986] and [Ohde, 1994] have made this same observation for Japanese syllables and children's speech, respectively. In an effort to mimic human perception, the proposed speech recognition system finds landmarks and then focuses subsequent processing on relevant signal portions, instead of treating each part of the signal as equally important. Based on the kind of landmark found, certain distinctive features will be relevant and others will not. This very directed approach minimizes the amount of processing necessary. Landmarks can also be found somewhat independently of timing factors, like speaking rate and segmental duration. On the other hand, landmarks can give timing information to aid in later processing. Figure 1.6 shows an example of landmarks: the vertical lines at the closure and release of a [d] are the landmarks. The bundles underneath each landmark symbolize the bundles of distinctive features that can be identified at each landmark.

Landmark detection is just one way to organize the speech waveform, as illustrated in Figure 1.7. Frame-based processing, often used in conjunction with HMMs, is another way. It is presently the most popular way of dividing up the speech

[5] For a description of the distinctive features in Table 1.1, see [Chomsky and Halle, 1968].

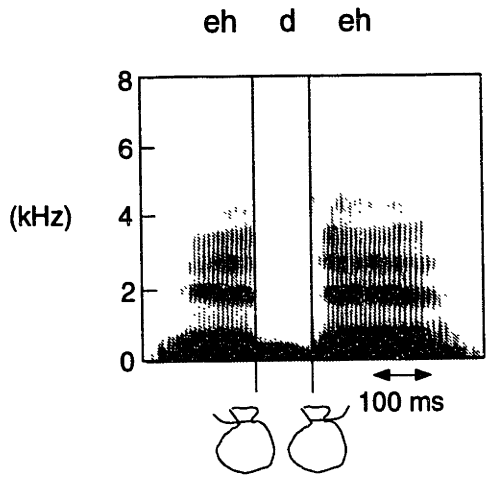

Figure 1.6: Examples of landmarks at a [d] closure and release, as designated by the vertical lines. The bundles underneath symbolize the bundles of distinctive features that can be found at each landmark.

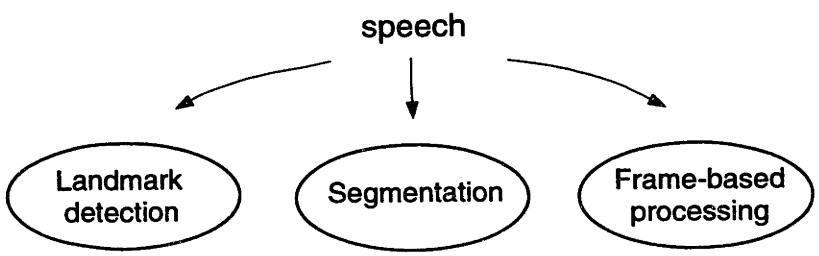

Figure 1.7: An illustration of the alternatives for organizing a speech waveform.

waveform. Whereas landmark detection focuses subsequent processing on relevant signal portions, frame-based processing homogeneously applies the same processing to each frame of speech. Each frame is typically 15–30 ms long, occurring every 5–10 ms. Phone hypotheses are directly assigned to each frame. Because segmental and prosodic level information can be acquired only after phone recognition, such information cannot aid in the phone recognition itself.

More structured than frame-based processing, segmentation is the process of placing boundaries in the speech waveform to delimit unequal-length, semi-steady-state, abutting regions, with each region corresponding to a phone or sub-phone unit.[6] Subsequent signal processing typically focuses on acquiring averages across these regions (e.g. [Gish and Ng, 1993]), although some attempt is made to focus on attributes near the boundaries (e.g. [Zue et al., 1990b], [Marcus, 1993]). In various tasks, [Flammia et al., 1992] and [Marcus, 1993] have found that a segmentation approach performs better or comparably to a frame-based approach while reducing the computational load in training and testing by a significant amount. In continuous speech recognition systems of the 1970s through the mid-1980s, segmentation was a popular method of organizing the speech waveform. Segmentation was compatible with acoustic-phonetic processing, which was then widely used (e.g. [Weinstein et al., 1975]). A problem arises with segmentation, however, when parts of the waveform do not have sharp boundaries, like those corresponding to diphthongs and semivowels. In order to accommodate different degrees of abruptness, either oversegmentation is required in order to find all the pieces of interest [Andre-Obrecht, 1988] or a multi-level representation is needed [Withgott et al., 1987], [Glass, 1988]. While many of the boundaries in segmentation are also the landmarks for a landmark-based system, the motivations for the two methods are quite different. Landmarks are the foci, not the boundaries. The problem with delimiting semivowels and diphthongs is avoided altogether by landmark detection, as will be explained in Chapter 2. Landmark detection is typically more hierarchical and involves more than one acoustic measure.

[6]Segmentation does not involve the further step of associating any phonetic significance to these pieces. The assignment of phonetic labels is done by a subsequent classification stage.

1.4 Thesis objective and outline

As the first step in the lexical access process, landmark detection is of primary importance. The most numerous types of landmarks are acoustically-abrupt. An estimate based on the TIMIT corpus, a database of phonetically-balanced sentences, shows that acoustically-abrupt landmarks comprise approximately 68% of the total number of landmarks in speech. These landmarks are often associated with consonantal segments, e.g. a stop closure or release. The objective of this thesis is to create a knowledge-based algorithm for automatically detecting acoustically-abrupt landmarks. The process of landmark detection provides information about the articulator-free features [consonantal], [sonorant], and [continuant]. The landmark detector is originally designed and tested on clean, broadband, phonetically-controlled speech. In order to be of practical use, however, the landmark detector should be able to work in a broader range of environments. In subsequent experiments, the landmark detector is adapted, in a knowledge-directed manner, to more phonetic contexts, a variety of speakers and accents, a different microphone, noise, and telephone speech.

The organization of this thesis is as follows. Chapter 2 describes landmarks in greater detail. Next, four landmark detection experiments, each involving a different database, are presented: clean speech (Chapter 3), noisy speech (Chapter 4), TIMIT (Chapter 5), and telephone speech (Chapter 6). For each experiment, the database used, the knowledge-based modifications to adapt the landmark detection algorithm to the speech characteristics of that database, and the results of landmark detection are presented. Finally, Chapter 7 summarizes the thesis, shows how the articulator-free features can be deduced from the landmark type, and gives suggestions on how the landmark detector can be improved.

Chapter 2

Landmarks

Landmarks are categorized into four groups: abrupt-consonantal, abrupt, nonabrupt, and vocalic. Figure 2.1 shows examples of these landmark groups for the utterance "The day was long". This chapter will describe these landmark groups. The focus of this thesis is on detecting the abrupt-consonantal and abrupt landmarks.

Phonologically, segments can be classified as [+consonantal] or [-consonantal]. A [+consonantal] segment involves a *primary articulator* forming a tight constriction in the midline of the vocal tract [Sagey, 1986]. Only the articulators in the oral cavity (i.e. lips, tongue blade, and tongue body) can be primary articulators. Segments not involving a tight constriction or implemented by articulators outside of the oral cavity

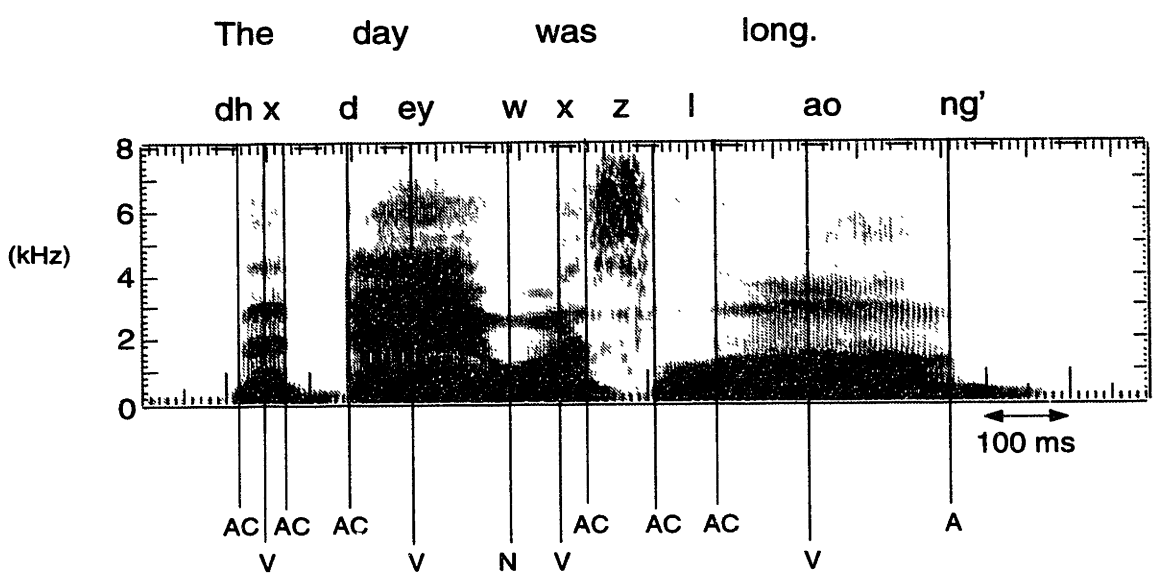

Figure 2.1: An illustration of landmarks. **AC** = abrupt-consonantal, **A** = abrupt, **N** = nonabrupt, **V** = vocalic.

(i.e. soft palate and glottis) are [–consonantal]. Speech is formed by a series of articulator narrowings and releases. The most salient of these narrowings and releases are acoustically-abrupt. An acoustically-abrupt constriction involving a primary articulator is typically tight and [+consonantal]. An *abrupt-consonantal* (**AC**) landmark marks the closure and another marks the release of one of these [+consonantal] constrictions. The clearest manifestation of an **AC** landmark is when the constriction occurs adjacent to a [–consonantal] segment. A pair of these landmarks, one on either side of the constriction, will be referred to as the *outer* **AC** landmarks. An example of a pair of outer **AC** landmarks is the [b] closure and release in "able". Other landmarks can occur within or outside of the pair of outer **AC** landmarks. These others are describe below.

A common sequence of landmarks is one in which the outer **AC** landmarks are governed by the same underlying segment and, thus, are implemented by the same articulator (e.g. the [b] closure and release in "able"). In consonantal clusters, however, the two outer **AC** landmarks are often not governed by the same articulator (e.g. the [p] closure and [d] release in "tap dance"). The release by the first articulator or the formation of the constriction by the second articulator may or may not be manifested in the acoustic signal. If the articulatory event is observable in the acoustic signal, then it is marked as an *intraconsonantal* **AC** landmark. In the "tap dance" example, if the [p] is released, then the [p] burst and the [d] closure are intraconsonantal **AC** landmarks. Figure 2.2 shows the location of an intraconsonantal **AC** landmark between the two outer **AC** landmarks.

The configuration of the glottis and the soft palate are articulatorily independent of the occurrence of an **AC** landmark. Just as for abrupt primary articulator movement, the movements of the glottis or the soft palate can independently cause an abrupt change in the acoustic signal. For example, as the glottis moves from a spread to a modal configuration when air is passing through, vocal-fold vibration begins. This onset of vocal-fold vibration is observed as a rapid change in the characteristics of the sound. Likewise, if the velopharyngeal port closes when the oral cavity is already closed, there is an abrupt increase in intraoral pressure, resulting in a reduction in the amplitude of glottal pulses. These abrupt changes in the sound caused by glottal or velopharyngeal activity but without accompanying primary articulator movement are labeled as *abrupt* (**A**) landmarks. **AC** and **A** landmarks differ

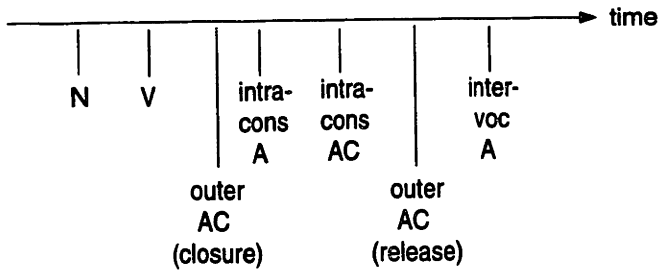

Figure 2.2: A possible positioning of landmarks with respect to a pair of outer **AC** landmarks. The sequence of landmarks shown is not necessarily a real sequence; rather, the location of a landmark is significant only in terms of where it is with respect to the pair of outer **AC** landamrks. Intraconsonantal **AC** and intraconsonantal **A** landmarks occur within the pair of outer **AC** landmarks. **N**, **V**, and intervocalic **A** landmarks occur outside of the pair of outer **AC** landmarks.

in that **A** landmarks do not involve primary articulator movement and therefore are [–consonantal]. Figure 2.2 shows two **A** landmarks. **A** landmarks can occur outside of a pair of outer **AC** landmarks (*intervocalic* **A** landmarks, e.g. the voice onset after the [p] burst in "paint") or within the pair (*intraconsonantal* **A** landmarks, e.g. the [m]-[dh] transition in "something"). Together **AC** and **A** landmarks comprise approximately 68% of the total number of landmarks.

For semivowels, the constriction that is formed is not narrow enough to create an abrupt change in the spectrum of the sound. The narrowing of the constriction, however, does reach some articulatory extremum out of which it gradually releases. The narrowing of the constriction usually causes a decrease in the first-formant frequency, F_1, and in the amplitude of the sound. If the consonant occurs between two vowels, a minimum in F_1 and in waveform amplitude denotes the narrowest point in the constriction (e.g. the middle of [w] in "away"). This point is a *nonabrupt* (**N**) landmark and occurs outside of a pair of outer **AC** landmarks, as shown in Figure 2.2. The narrowest point in the production of a semivowel can be coincident with an acoustically-abrupt part of speech; in such a case, the landmark is both an **N** and an acoustically-abrupt landmark (e.g. the [dw] release in "dwell"). **N** landmarks comprise approximately 3% of the total number of landmarks.

Finally, vowels have their own landmarks. When the vocal tract is at an open extremum for a vowel, a local maximum occurs in both F_1 and waveform amplitude (e.g. the middle of [ae] in "bat"). This point is a *vocalic* (**V**) landmark and occurs

	Phonetic category	*Landmark type*
Outer **AC**	+v fric closure	**g**(lottis)
	+v fricative release	"
	-v fricative closure	"
	-v fricative release	"
	flap closure	"
	flap release	"
	+v stop closure	"
	-aspirated stop release	"
	-v stop closure	"
	+aspirated stop release	**b**(urst)
	nasal closure	**s**(onorant)
	nasal release	"
	[l] closure	"
	[l] release	"
Intraconsonantal **AC**	stop closure	**b**(urst)
	stop release	"
	fricative closure	"
	affricate release	"
	nasal → fricative	**g**(lottis)
	fricative → nasal	"
Intraconsonantal **A**	velopharyngeal closure	**g**(lottis)
	velopharyngeal release	"
Intervocalic **A**	[ʔ] closure	**g**(lottis)
	[ʔ] release	"
	-v [h] onset	"
	-v [h] offset	"

Table 2.1: Landmarks listed by phonetic category and type.

outside of a pair of outer **AC** landmarks, as shown in Figure 2.2. **V** landmarks make up approximately 29% of the total number of landmarks.

For lexical access, a landmark-to-segment relation must be specified. Sometimes, a one-to-one relation holds, like the relation between a **V** landmark with a vowel. Sometimes, a two-to-one relation holds; for example, an intervocalic fricative can have a landmark at closure and a landmark at release. In some cases, a three-to-one relation holds, as in [p], which has one landmark at closure, one at burst, and one at voice onset. Or, there may be a one-to-two relation, as in "bright", where the landmark at the [b] release serves both [b] and [r].

For the purpose of evaluating the performance of a landmark detection algo-

rithm, the **AC** and **A** landmarks are classified into phonetic categories. Table 2.1 lists these categories. The outer **AC** landmarks are the closures and releases associated with fricatives, flaps, stops, nasals, and [l]s next to [-consonantal] segments. For example, the closure and release of the /b/ in /aba/ are outer **AC** landmarks. Flaps are classified as obstruents, although they are often too short to allow much pressure build-up. Eventually, they will need a specialized detector dedicated to finding them. Intraconsonantal **AC** and **A** landmarks are those in consonant clusters. For illustration, if the /d/ in "tadpole" is released, then the /d/ release is an intraconsonantal **AC** stop release and the /p/ closure is an intraconsonantal **AC** stop closure. In the word "ski", the end of /s/ at the /k/ closure is an intraconsonantal **AC** fricative closure. An affricate release as in /č/ of "church" is an intraconsonantal **AC** affricate release. The landmark between the /s/ and /m/ in "small" is a fricative $\rightarrow$ nasal landmark. The landmark between the /m/ and /z/ in "plums" is a nasal $\rightarrow$ fricative landmark. A velopharyngeal closure or release occurs for nasal/stop combinations. For example, the end of glottal vibration after the /n/ in "bending" is a velopharyngeal closure. The beginning of glottal vibration for the /m/ in "batman" is a velopharyngeal release. The intervocalic **A** landmarks are caused by the glottis. These are the onsets and offsets of glottal stops and aspirated consonants. The landmark type column of Table 2.1 will be discussed when the landmark detection algorithm is presented, in Section 3.2.

Chapter 3

LAFF: Clean speech

The first experiment on landmark detection was performed in a carefully controlled setting. The database used was of clean speech and contained words with few consonant clusters or syllables. The sentences were read naturally but carefully. Only a limited number of speakers and sentences were used in this controlled experiment. Later chapters will describe experiments done in less controlled environments. In this chapter, the database, the landmark detection algorithm, and the results of the experiment using clean speech will be presented.

3.1 LAFF database

3.1.1 Speech recording

The database used in this study will be called the LAFF database (Lexical Access From Features, [Stevens, 1986], [Klatt, 1989]). The utterances in the database were tape-recorded in a quiet room using an Electrovoice omnidirectional microphone dangling 25 cm in front of and 5 cm above the speaker's mouth. This placement was roughly equidistant to the nose, mouth, and throat, so the microphone could pick up signals from all three radiating sources. It was not placed in direct line of the mouth in order not to pick up the low-frequency puffs of air during aspiration. The microphone had a broadband frequency response, with 3 dB cutoff frequencies at 50 Hz and 18 kHz. In between these cutoff frequencies, the frequency response magnitude varied no more than ± 1 dB from a flat response. Sentences were read naturally and clearly. The recordings were passed through an anti-aliasing filter with a cut-off frequency of 7.5 kHz before being digitized at 16 kHz. The 7.5 kHz cut-off frequency

allowed relevant high-frequency frication noise in female speech to be captured. The signal-to-noise ratio (SNR) for speech recorded in this condition was about 30 dB. Section 4.1 describes how this SNR was measured.

3.1.2 Lexicon and data sets

A lexicon of 250 words was used to construct 100 syntactically-correct sentences. The words were mostly monosyllabic (69%), some bisyllabic (30%), and very few trisyllabic (1%). Fifteen percent of the words had consonant clusters (e.g. [sp] in "sport", [nd] in "and", or [fy] in "few"). Two data sets – development and test – were constructed. The development set was used during algorithm development: the design and parameter values of the algorithm were modified by hand based on knowledge accumulated from preliminary results with the development set. It consisted of 4 speakers (2 women, 2 men) speaking the first 20 of the 100 sentences. The test set was used as an independent set from the development set to see how well the algorithm generalized to new speakers and new sentences. It consisted of 2 new speakers (1 woman, 1 man) speaking the last 20 of the 100 sentences. The test and development set sentences are listed in Appendix A.

3.1.3 Labeling convention

Because a complete speech recognition system which employed landmark detection was not available, the landmark detector could not be evaluated in terms of a word recognition score. Instead, an intermediate level of evaluation was necessary. This intermediate level was a landmark detection score, which was highly dependent on the landmark labeling convention. Thus, care was taken to objectively label utterances with landmarks. In this section, the rules for landmark labeling are described in detail and then a comparison to the TIMIT labeling convention [Zue and Seneff, 1990] is presented.

Landmark labeling was done using phonological and acoustic rules. Three decisions had to be made: (1) the existence of an **AC** or **A** landmark, (2) the time placement of the landmark, and (3) the categorization of the landmark. Waveform and spectrogram displays and listening were used to guide landmark labeling.

An **AC** or **A** landmark exists if the underlying segment is sufficiently man-

ifested in the acoustic signal and is acoustically-abrupt. The underlying phonology of an utterance was used to guide the labeling. Most of the time, the underlying phonology was manifested in the speech signal. Sometimes, however, phonological targets were modified in the speech signal. For example, a [ð] following a nasal, as in "in the", often shows little evidence of diminished voicing, so instead of labeling it as a velopharyngeal closure followed by a fricative release, it was labeled simply as a sonorant consonantal release. Even if a phonological target was manifested in the speech signal, it may not have been acoustically-abrupt. A typical example is /l/, which can be acoustically-abrupt or not. An /l/ closure or release was considered acoustically-abrupt if 90% of its broadband energy transition occurred within 40 ms.

Regarding the time placement of a landmark, an **AC** or **A** landmark was put at the time of the articulator movement which caused the landmark. Time placement for most **AC** and **A** landmarks is straightforward because of the abrupt spectral change that takes place. The landmark for a voiced obstruent closure was placed at the disappearance of high-frequency formant energy in the spectrogram.

The landmarks were grouped into the phonetic categories listed in Table 2.1. A flap was defined to have a 35 ms or less closure interval; otherwise it was categorized as a [t] or [d]. This criterion is in accord with [Zue and Laferriere, 1979]'s acoustic study, in which they found that the average closure period of a flap is 26–27 ms, with a standard deviation of 10–12 ms. Outer **AC** landmarks associated with stop closures and fricatives were further divided into voiced and unvoiced. To avoid conflicting data on the acoustic manifestations of voicing, a voiced/voiceless decision was made based on the phonology rather than on the phonetics. Stop releases were labeled with one landmark if the voice onset time (VOT) was 20 ms or less. They were labeled with separate burst and voice onset landmarks if the VOT was more than 20 ms. Most voiced stops fall into the 20 ms or less category while most voiceless stops fall into the other category. This criterion agrees with [Lisker and Abramson, 1964]'s finding that English unvoiced aspirated stops in word-initial position have an average VOT of 28 ms or more and the voiced unaspirated counterparts have an average VOT of 17 ms or less. Affricates also have a 20 ms criterion: separate burst and fricative release landmarks were assigned only if the VOT exceeded 20 ms. Nasal onsets and offsets at utterance extrema were labeled as velopharyngeal releases and closures, respectively.

In order to assess the effect of vowel reduction on landmark detection, vowels

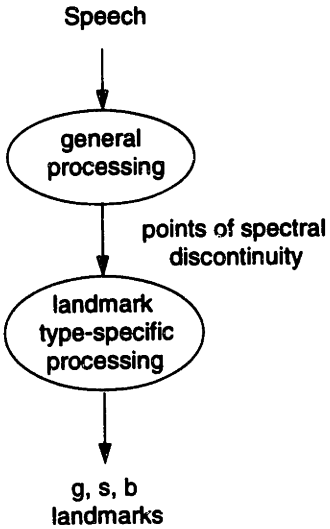

Figure 3.1: The landmark detection algorithm. Landmark types are **g**(lottis), **s**(onorant), **b**(urst).

next to **AC** and **A** landmarks were labeled as either reduced or unreduced. A reduced vowel is short in duration (typically less than 50 ms [Klatt, 1976]) and low in intensity (typically 12 dB or more down from a neighboring stressed vowel [Beckman, 1986]). Schwas by definition are reduced; syllabic nasals, syllabic /l/s, and some /ɚ/s are typically reduced. All other vowels, stressed or otherwise, were classified as unreduced.

The LAFF labeling conventions agree for the most part with the TIMIT labeling conventions for acoustically-abrupt points in the speech waveform [Zue and Seneff, 1990]. Like LAFF, TIMIT labels were motivated both by phonemics and acoustics. Abrupt acoustic changes were always marked. If no acoustic evidence existed for a certain phoneme, then no label was put there. Spectrogram displays and listening were also used to decide on labels. One difference between TIMIT and LAFF was the labeling of stop releases. Whereas only one label was placed at an unaspirated stop release in LAFF, two labels were placed in TIMIT, one for the burst and one for the voice onset, regardless of the VOT.

3.2 LAFF algorithm

Figure 3.1 shows a flow diagram of the algorithm. Speech input goes through a general processing stage, whose outputs feed a landmark type-specific processing stage. The

output of type-specific processing is a series of landmarks specified by time and type. In general processing, a spectrogram is computed and divided into six frequency bands. Then, a coarse- and a fine-processing pass are executed. In each pass, an energy waveform is constructed in each of the six bands, the derivative of the energy is computed, and peaks in the derivative are detected. Localized peaks in time are found by matching peaks from the coarse- and fine-processing passes. These peaks represent times of abrupt spectral change in the six bands. In type-specific processing, the localized peaks direct processing to find three types of landmarks. These three types are:

1. **g**(lottis), which marks a time when the vocal folds transition from freely vibrating to not freely vibrating or vice versa.

2. **s**(onorant), which marks sonorant consonantal closures and releases.

3. **b**(urst), which designates stop or affricate bursts and points where aspiration or frication ends due to a stop closure.

Table 2.1 associates landmarks in each phonetic category with a landmark type. The rest of this section presents the steps just outlined in greater detail.

The landmark detection algorithm was implemented in the C programming language. The source code is listed in Appendix C.

3.2.1 General processing

Figure 3.2 shows a block diagram of the general processing stage. A broadband spectrogram is computed with a 6 ms Hanning window every 1 ms. Each 6 ms frame is zero-padded out to 512 points before a *discrete Fourier transform* (DFT) is taken. The top panel of Figure 3.3 shows a spectrogram for the utterance: "The money is coming today". The spacing between points for the DFT is 31.2 Hz, so that formant peak amplitudes can be approximated reasonably well. The high frame rate allows quick acoustic changes to be monitored. Some acoustic changes happen very quickly, particularly the ones associated with obstruent segments as articulators move from one quantal state to another. The short Hanning window produces a broadband spectrum, which gives formant information but suppresses harmonic detail.

The resulting spectrogram is then divided into the following 6 frequency bands:

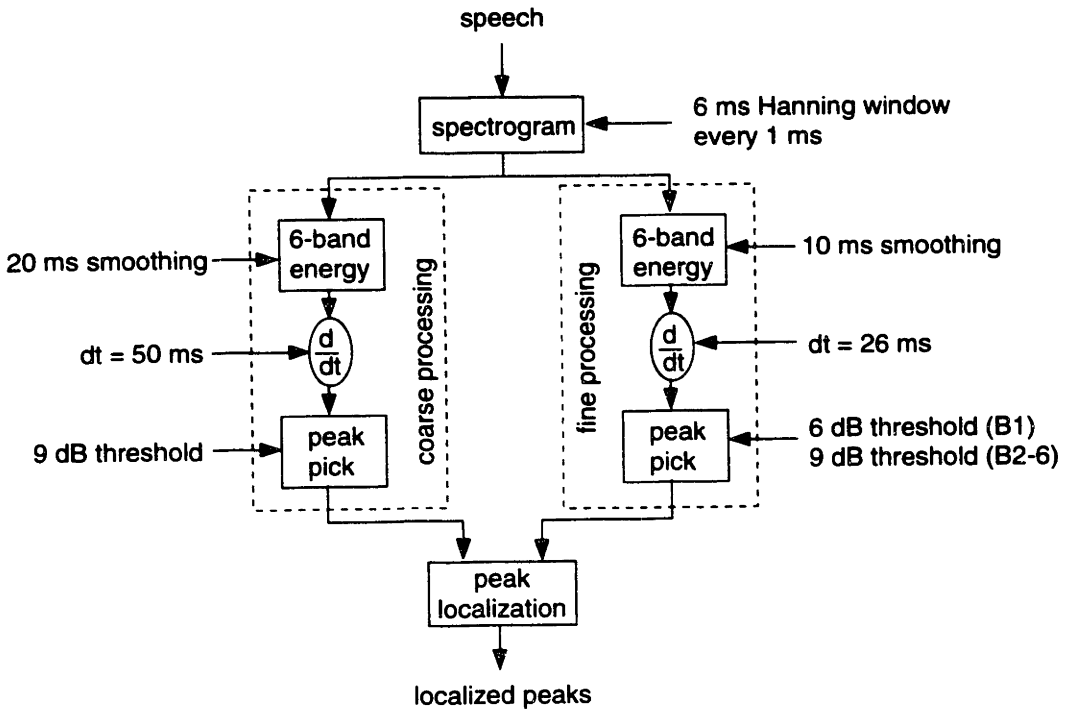

Figure 3.2: Details of general processing.

Band	1:	0.0 – 0.4 kHz
	2:	0.8 – 1.5
	3:	1.2 – 2.0
	4:	2.0 – 3.5
	5:	3.5 – 5.0
	6:	5.0 – 8.0 .

Band 1 monitors the presence or absence of glottal vibration. It does not extend beyond 400 Hz in order to avoid picking up low frequency burst energy. Closures and releases for sonorant consonants are detected using Bands 2–5. These bands approximate the frequency ranges for the spectral prominences of sonorant consonants. For intervocalic sonorant consonantal segments, the biggest spectral change occurs in the 0.8–2 kHz range due to the introduction of a zero in that range. In order to capture this change, Bands 2 and 3 span this range and are chosen to overlap. Each band is not guaranteed to contain one spectral prominence, but at least one band is expected to capture at least one spectral prominence. At a sonorant consonantal closure, Band 1 energy remains strong because glottal vibration continues; however, spectral prominences above F_1 show a marked abrupt decrease in energy because of increased acoustic losses, the introduction of pole-zero pairs, and the fall in F_1. The

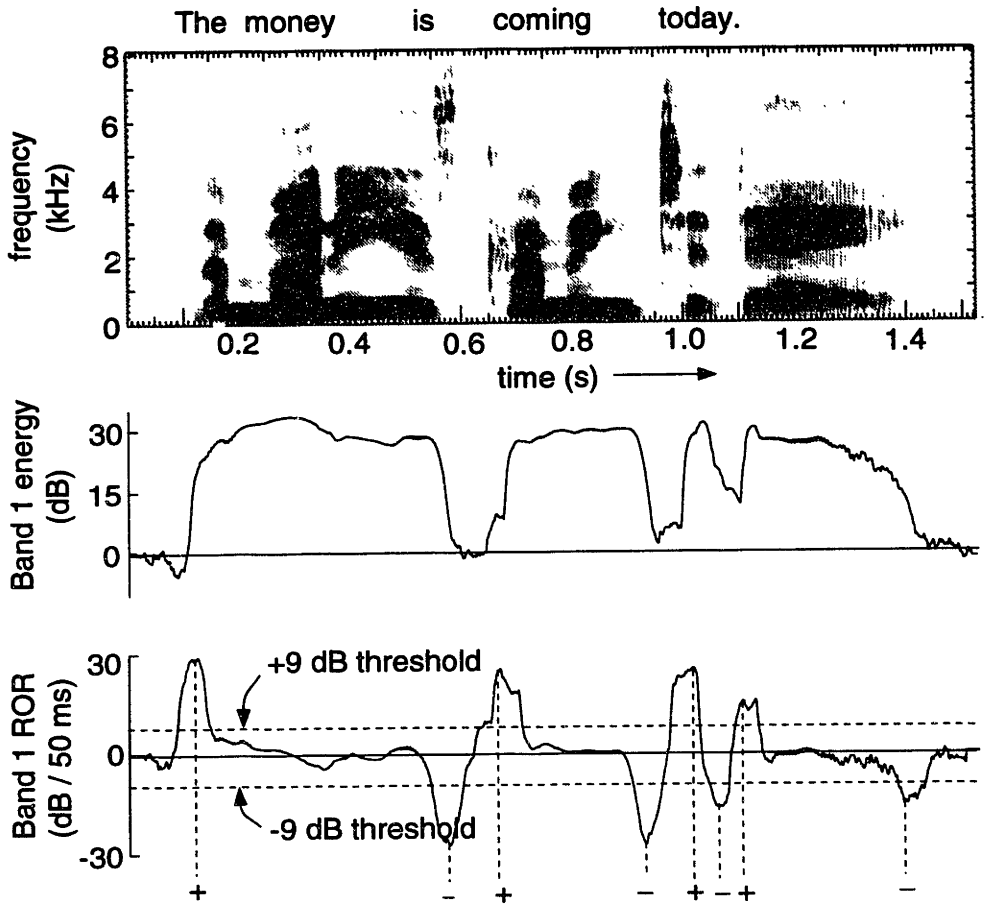

Figure 3.3: An illustration of general processing. The top panel shows a spectrogram for the utterance "The money is coming today". The middle panel is the Band 1 energy and the bottom panel is the Band 1 ROR, both from coarse processing. The two dotted horizontal lines are thresholds for peak picking. The peaks detected are shown with ± signs indicating the polarity of the peaks. Band 1 peaks are also likely candidates for **g**(lottis) landmarks.

onsets and offsets of aspiration and frication noise associated with stops, fricatives, and affricates can also be found from Bands 2–5. Noise energy will lie in at least one of these four bands. Band 6 spans the remaining frequency range up to 8 kHz, and is one of the bands used for silence detection for stops.

Following the spectrogram, energy changes in the 6 bands are found using a two-pass strategy, as indicated by the two parallel branches coming out of the spectrogram block in Figure 3.2. Both passes employ the same processing steps except that the first pass uses coarse parameter values to find the general vicinity of a spectral change, and the second pass uses fine parameter values to localize it in time. The processing strategy will be described with the first pass parameter values, and then the second pass parameter values will be given afterwards.

In the coarse-processing pass, an energy waveform in each of the 6 bands is calculated. The middle panel of Figure 3.3 gives an example of the Band 1 energy waveform. An energy waveform should be able to resolve abrupt acoustic changes due to sudden changes in formant frequency amplitudes, but ignore glottal pulse variations and noise fluctuations. To smooth out the unwanted characteristics, a 20 ms average of the squared magnitude of the STFT, centered about the time of interest, is computed every 1 ms. Within each band, the maximum in the smoothed spectrogram at each time is chosen to represent the energy in that band. Energy is then recorded in dB. Provided a band encompasses exactly one spectral prominence, picking the maximum energy in the band as a function of time is the same as following the spectral prominence amplitude in time.

Once the 6-band energy is computed, a 6-band *rate-of-rise* (ROR) is found by taking an overlapping dB first difference of the energy in each band. The ROR waveform of a band indicates how quickly the energy is changing in that band. Working with dB differences automatically considers relative values so that gain normalization is not necessary across utterances. The first difference is computed every 1 ms using a 50 ms time step, centered about the time of interest. The 50 ms time step is chosen to span energy transitions of abrupt closures and releases, including voiced obstruent closures, taking into account the 6 ms Hanning window and 20 ms smoothing. The third panel of Figure 3.3 shows the ROR calculated from the Band 1 energy waveform above it.

The positive and negative peaks in the ROR waveform are the points of

abrupt acoustic change in a band. A peak-picking algorithm (originally described by [Mermelstein, 1975]) was tailored to find the ROR ±peaks whose absolute value is greater than 9 dB. The 9 dB threshold is motivated by the difference in glottal source amplitude between an obstruent and a neighboring vocalic segment and by empirical evidence. Appendix B shows how this threshold was derived. The two dotted horizontal lines in the third panel of Figure 3.3 show the ±9 dB thresholds. The peaks that are detected by the peak-picker are shown with ± signs indicating the polarity of the peaks.

In the parallel fine-processing pass shown in Figure 3.2, some parameter values are modified in order to localize energy changes in time. A 10 ms smoothing interval is used on the spectrogram instead of 20 ms; a time step of 26 ms is used for the ROR calculation instead of 50 ms ; and the peak threshold for Band 1 is reduced from 9 dB to 6 dB. This 3 dB reduction is made to accommodate smaller peaks due to the reduction of the time step. The peak thresholds in Bands 2–6 are kept at 9 dB to prevent too many spurious peaks in those bands from occurring.

As shown in Figure 3.2, the ROR peaks resulting from the coarse and fine passes come together at a "peak localization" block. Here, ROR peaks from the coarse pass are used to guide the search for corresponding ROR peaks in the same band from the fine pass. Within ±30 ms of a coarse pass peak, the biggest fine pass peak (in absolute terms) with the same sign as the coarse pass peak is chosen as the localized peak. Localized peaks are the inputs into the landmark type-specific processing stage. The type-specific detectors for **g**, **s**, and **b** landmarks are explained next.

3.2.2 G(lottis) detector

A g(lottis) landmark pinpoints a time when glottal vibration turns on or off. The factors causing glottal vibration to cease are: a buildup of intraoral pressure due to a supraglottal constriction, vocal-fold spreading or glottal closure, or a reduction of subglottal pressure. To find **g** landmarks, all the localized Band 1 ROR peaks are initially candidates. For illustration, the Band 1 peaks in Figure 3.3 are the approximate times for possible **g** landmarks. (The peaks shown give only approximate times since they are from coarse-processing before peak localization.) A +peak indicates

the turning on of glottal vibration; a –peak indicates the turning off of glottal vibration. Band 1 peaks are paired with each other. When glottal vibration turns off, it must turn on some time later unless the utterance has ended. Thus, with the exception of the last –peak in an utterance, all the –peaks should be followed by a +peak. Peaks are inserted wherever necessary to satisfy this condition. The point of insertion is guided by the shape of the Band 1 energy contour. After the peaks are paired, a minimum vowel requirement is imposed on each +/– pair of **g** landmarks. A +/– pair of **g** landmarks should span at least the vowel part of a syllable. The minimum vowel is a schwa. The requirement is that Band 1 energy between a +/– pair of peaks must be no less than 20 dB below the maximum Band 1 energy in the utterance for at least 20 ms. The 20 dB threshold agrees with [Stevens, 1995b]'s vowel reduction study, in which he showed that the F_1 amplitude of a reduced vowel can regularly be as low as 17 dB below that of the pitch-accented vowel in the utterance, provided the reduced vowel is not devoiced. The 20 ms duration requirement is a lower bound on the length of a schwa, which can regularly be about 30 ms in duration [van Beinum, 1994]. If the region between the +/– pair of peaks does not satisfy the vowel requirement, then that pair is most likely due to some noise or burst and is deleted. Short bursts of energy due to creaky voicing or noise are candidates for deletion.

Most of the **g** landmarks are detected with the method just described. However, heavily voiced obstruents (e.g. voiced stops, voiced fricatives, flaps) do not exhibit much Band 1 energy change, so they can be missed by this method. If they are missed, then another detector which concentrates on high frequency spectral abruptness is invoked. This detector is very similar to the **s** detector. The next section describes the **s** detector, and then explains how a variation of it becomes a **g** detector for heavily voiced obstruents.

3.2.3 S(onorant) detector

An **s**(onorant) landmark is caused by the closure or release of a nasal or [l]. Figure 3.4 illustrates **s** landmark detection. It shows "The money is ..." fragment of the utterance from Figure 3.3. A –**s** landmark designates a closure and a +**s** landmark designates a release. As the vocal tract constricts for a sonorant consonant, energy in the F_2 to F_4 range decreases. At a release, this energy increases. If the constriction

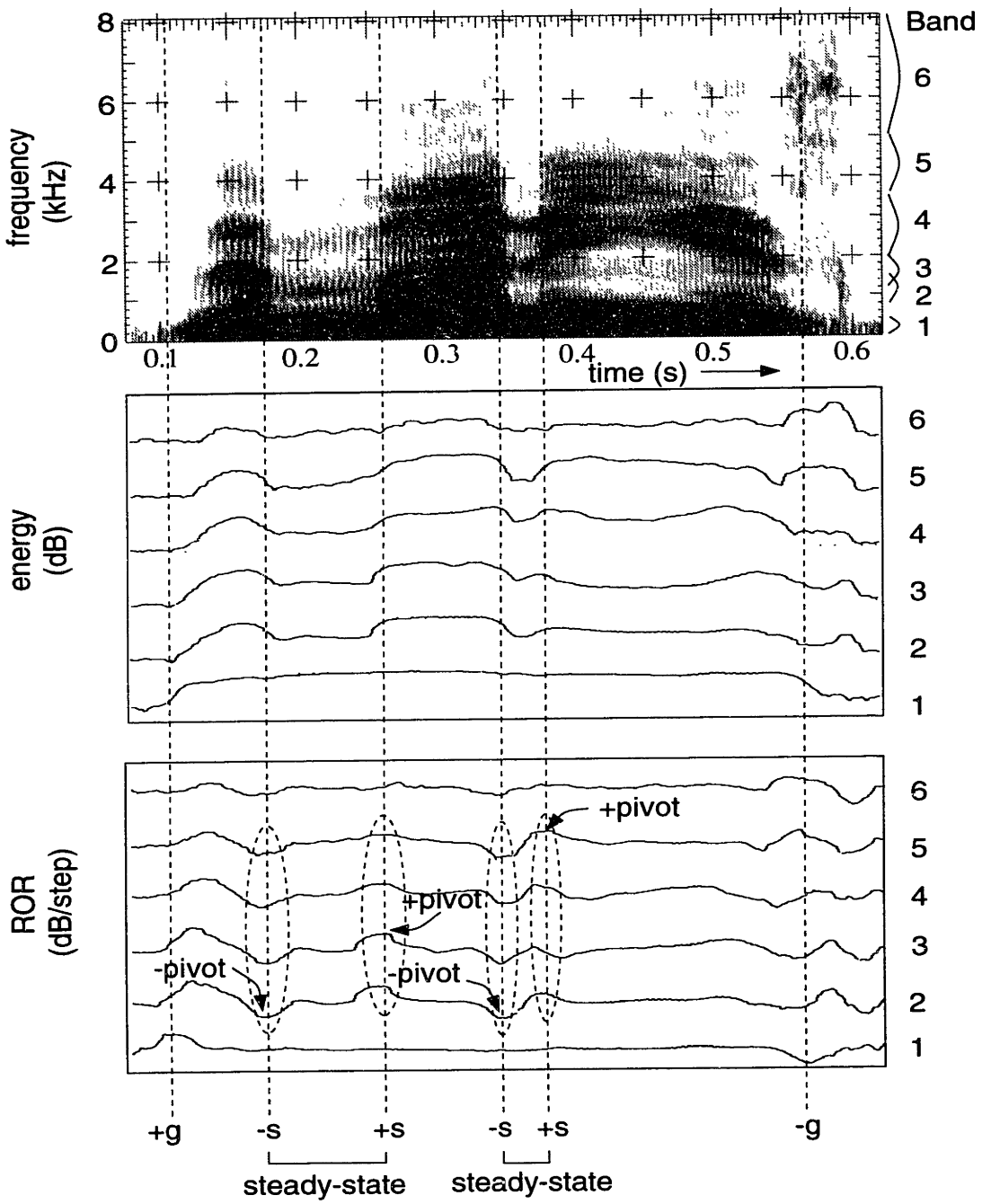

Figure 3.4: An illustration of s(onorant) landmark detection. "The money is ... " is shown. The top panel is the spectrogram. The middle panel is the 6-band energy waveform and the bottom panel is the 6-band ROR waveforms, both from fine processing. The energy waveform's vertical range is approximately 35 dB for Bands 1, 5, and 6, and 50 dB for Bands 2, 3, 4. Each ROR waveform's vertical range is approximately ±30 dB/time step.

is tight enough and occurs sufficiently quickly, the energy will change rapidly and simultaneously in all bands. During the constricted interval for a nasal or an [l], a primary articulator has made a complete closure, the vocal folds continue to vibrate, and the vocal-tract shape is relatively constant. Thus, the spectrum should remain relatively steady, especially at low frequencies.

To find **s** landmarks, only voiced regions, bounded by a **+g** landmark on the left and a **–g** landmark on the right, are considered. Within a voiced region, Bands 2–5 peaks having the same sign are grouped together. The biggest absolute peak in each group is designated the "pivot" and is a likely candidate for an **s** landmark. The pivot then has to pass a steady-state test and an abruptness test. The steady-state attribute is measured by examining the spectral magnitude in the 0–600 Hz range of the spectrogram; higher frequencies are not used because pole-zero movement may cause some variation in the higher frequencies. At closure and release, high-frequency abruptness is measured by checking for sufficient change in an energy signal calculated from the 1.3–8 kHz range of a pre-emphasized version of the spectrogram. This 1.3–8 kHz energy is used instead of the original 6-band energies in order to exclude false landmarks caused by semivowel formant movements. As a further measure of high-frequency abruptness, the Bands 2–5 peaks that were grouped together by sign earlier must occur somewhat coincidentally with the pivot in that group. The steady-state and abruptness tests that a pivot must pass are designed to exclude pivots caused by semivowels, which are generally not steady-state and not acoustically-abrupt.

A variation of the **s** detector just described becomes a **g** detector for heavily voiced obstruents. For this **g** detector, every aspect of the **s** detector stays the same except, instead of a steady-state requirement, a non-steady-state requirement is imposed. All the pivots that fail the steady-state test but pass the abruptness test are **g** landmarks.

3.2.4 B(urst) detector

Figure 3.5 illustrates **b**(urst) landmark detection. It is the "... is coming ... " portion of Figure 3.3. A **+b** landmark signifies an affricate or aspirated stop burst. The acoustic correlates for a **+b** landmark are a silence interval followed by a sharp increase in energy in high frequencies. Since **b** landmarks can occur only during

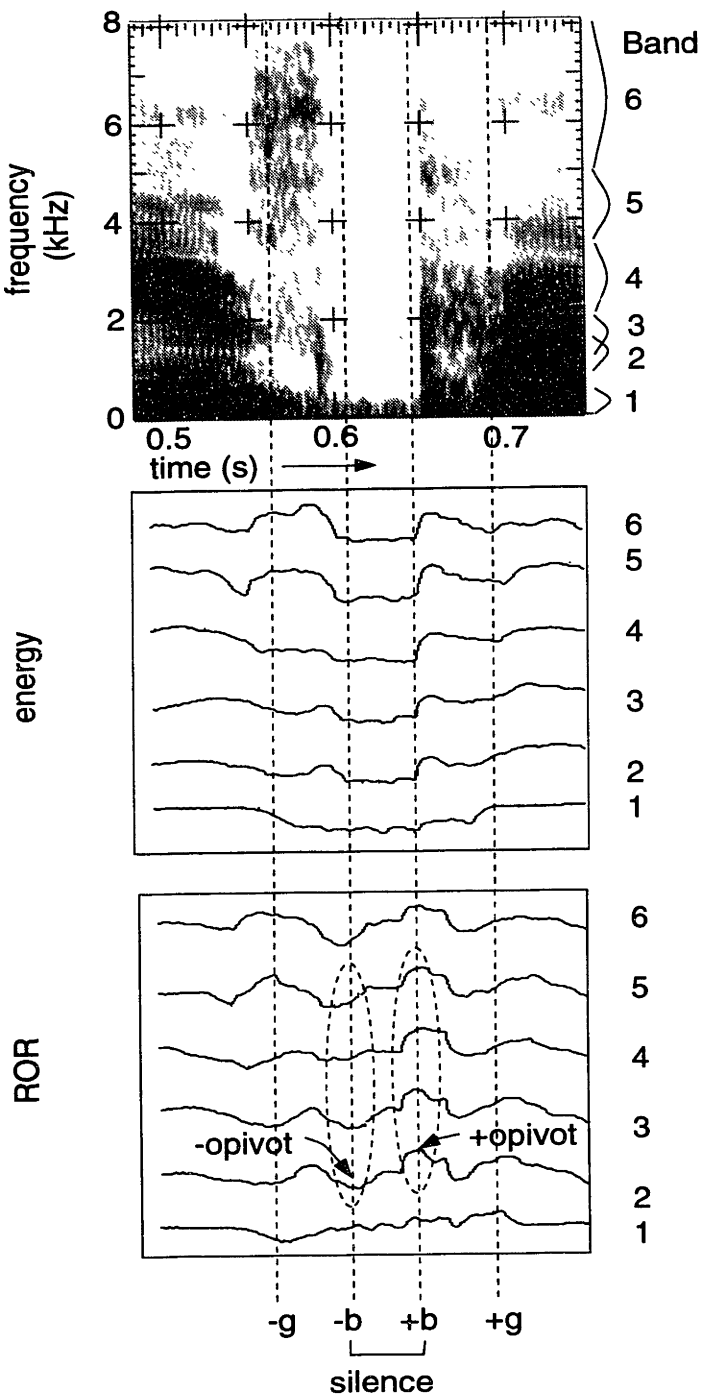

Figure 3.5: An illustration of b(urst) landmark detection. "... is coming ... " is shown. The top panel is the spectrogram. The middle panel is the 6-band energy waveform and the bottom panel is the 6-band ROR waveform, both from fine processing. Vertical ranges are the same as for Figure 3.4.

regions without glottal vibration, only the regions delimited by a **–g** on the left and a **+g** on the right are searched. First, opivots (for "obstruent pivots") are found in an analogous manner to pivots. An opivot is a candidate for a **b** landmark. Next, silence is measured around an opivot. For a +opivot, a silence interval must exist to the left. This silence period is measured with Bands 3–6 energy, using the background energy levels in each band as reference. Bands 1–2 are not used in order to allow voice bars of voiced obstruent closures to persist without upsetting the detection of an obstruent closure.

A **–b** landmark signifies the offset of frication or aspiration noise due to a stop closure. The acoustic correlates for a **–b** landmark are a sharp decrease in high frequency energy followed by a silence interval. This silence is measured using all the bands, including Bands 1–2, since a voice bar preceding the following stop consonant is unlikely in English.

3.3 LAFF results and discussion

Results of landmark detection are presented in terms of deletion, substitution, insertion, neutral, and detection rates. These rates were determined by comparing the output of the landmark detector with the labeled landmarks. A landmark was considered correctly detected if it was of the same sign and type (**g**, **s**, **b**) as the hand-labeled landmark, and was within ±30 ms of this hand-labeled landmark. Voiced obstruent closures were sometimes detected more than 30 ms beyond the oral closure because the voice bar could push the maximum Band 1 energy change to the right by tens of milliseconds. However, if a **–g** landmark was obviously due to the cessation of the voice bar, the voiced obstruent closure was considered detected. Beyond landmark detection, the articulator-bound feature processor will have to adjust the landmarks in time to suit its needs.

A deletion is a missed landmark. No landmark of the correct sign, regardless of type, was detected in the vicinity of the hand-labeled landmark. The deletion rate is calculated by dividing the number of deletions, D, by the number of labeled tokens in the category of interest, T_c:

$$\text{Deletion rate} = \frac{D}{T_c} \times 100\%. \tag{3.1}$$

A substitution is a landmark of the correct sign but wrong type. The substitution rate is found by dividing the number of substitutions, S, by T_c:

$$\text{Substitution rate} = \frac{S}{T_c} \times 100\%. \tag{3.2}$$

The detection rate can be deduced from the substitution and deletion rates. It represents the number of hand-labeled landmarks correctly identified by sign and type:

$$\text{Detection rate} = 100\% - \text{Substitution rate} - \text{Deletion rate}. \tag{3.3}$$

An insertion is a false landmark. It is not in the hand-labeled set and should not have been detected. The insertion rate is found by dividing the number of insertions, I, by T_c:

$$\text{Insertion rate} = \frac{I}{T_c} \times 100\%. \tag{3.4}$$

A landmark of incorrect sign found near a hand-labeled landmark, regardless of type, is an insertion.

A neutral landmark is also not in the hand-labeled set but, because it can be useful in acoustic-phonetic decoding, is not counted as an insertion. Neutral landmarks do not contribute to error. Examples of neutral landmarks are $\pm$**g** landmarks at creaks,[1] a –**s** landmark at the beginning of the voice bar of a voiced obstruent, and a +**b** landmark at an unaspirated stop burst. The neutral rate is computed by dividing the number of neutrals, N, by T_c:

$$\text{Neutral rate} = \frac{N}{T_c} \times 100\%. \tag{3.5}$$

Conventionally, the error rate refers to the sum of the deletion, substitution, and insertion rates. This rate will be called E_1:

$$E_1 = \text{Deletion rate} + \text{Substitution rate} + \text{Insertion rate}. \tag{3.6}$$

Note that the error rate can exceed 100% because there is no limit to the number of insertions the landmark detector can produce. E_1 is a conservative representation of the results, as it requires that the sign *and* the type of the landmark be correct in order not to add to the error. A substitution, though, is clearly not as serious an error as a deletion or an insertion. In some cases, a substitution would not even be

[1] Creaks have linguistic significance. For example, they can signal word boundaries [Umeda, 1978].

Landmark type	*# Tokens*	*Del.*	*Subs.*	*Ins.*	*Neut.*	E_1	E_2
g(lottis)	1052	1%	3%	7%	0%	11%	8%
s(onorant)	332	10%	2%	21%	9%	33%	31%
b(urst)	213	5%	1%	6%	33%	12%	11%
Total	1597	4%	2%	10%	7%	16%	14%

Table 3.1: Results of the LAFF development set, by landmark type.

Landmark type	*# Tokens*	*Del.*	*Subs.*	*Ins.*	*Neut.*	E_1	E_2
g(lottis)	486	2%	1%	2%	0%	5%	4%
s(onorant)	114	29%	1%	27%	10%	57%	56%
b(urst)	161	10%	2%	2%	8%	14%	12%
Total	761	8%	1%	6%	3%	15%	14%

Table 3.2: Results of the LAFF test set, by landmark type.

considered an error. To reflect this interpretation of the results, another error rate is defined:

$$E_2 = \text{Deletion rate} + \text{Insertion rate}. \tag{3.7}$$

E_2 excludes the substitution rate from the error. It requires that only the sign of the landmark be correct, regardless of type, in order for the landmark to not contribute to the error. This interpretation of the results does assume any type, and therefore gives less information about a landmark. E_2 weights deletions and insertions equally. In reality, however, a deletion could be much more serious than an insertion. In such a case, an even more accurate description of the error would be:

$$E_3 = \mathrm{K_d} \cdot \text{Deletion rate} + \mathrm{K_i} \cdot \text{Insertion rate} \tag{3.8}$$

where K_d and K_i are the weights for the deletion and insertion rates, respectively. These coefficients would have to be determined with cost functions for the specific application. Representation in terms of E_3 was not pursued in this thesis.

3.3.1 Overall results

Tables 3.1 and 3.2 show the results by landmark type for the development and test data sets, respectively. Figure 3.6 is a spectrogram showing some typical behavior of the landmark detector. For both data sets, **g** landmarks had the lowest deletion rate and did not degrade much from the development to the test set. **S** and **b** deletion rates, on the other hand, were somewhat higher, and did degrade from the

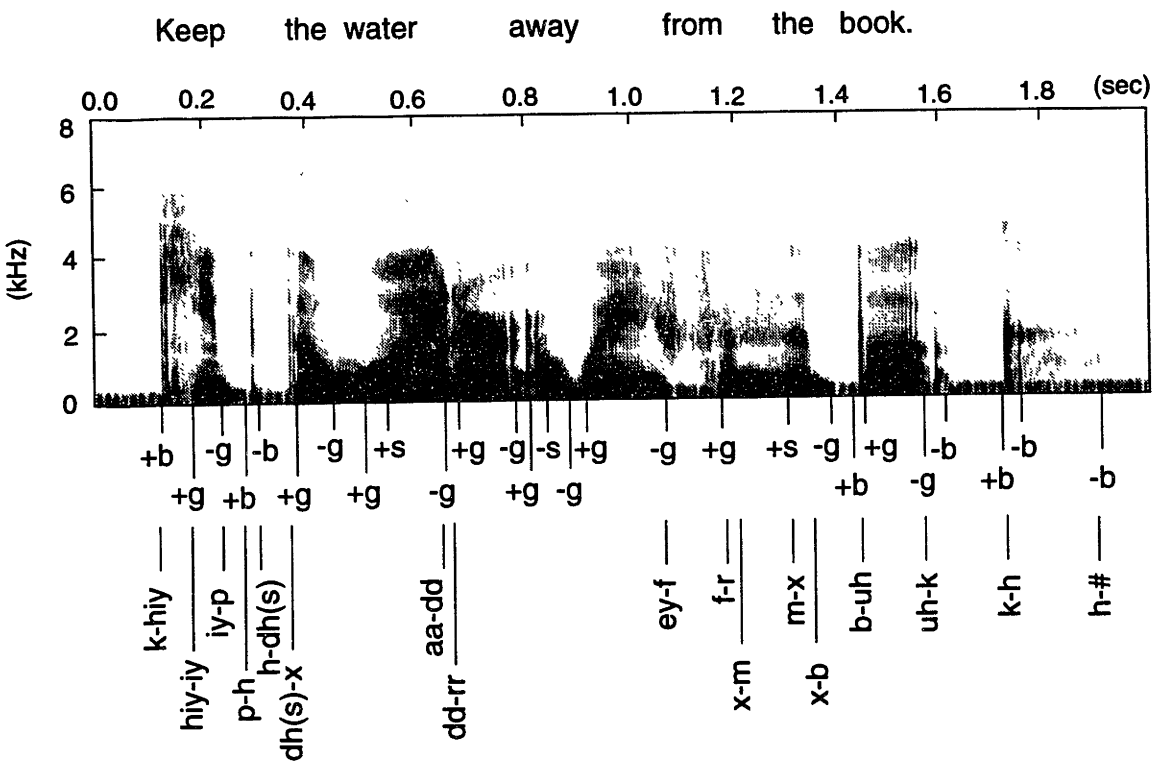

Figure 3.6: A spectrogram with the output of the landmark detector and the hand-labeled landmarks below.

Speaker	*# Tokens*	*Detection*
1	393	97%
2	405	95%
3	400	95%
4	399	90%

Table 3.3: Detection rates of the LAFF development set, by speaker.

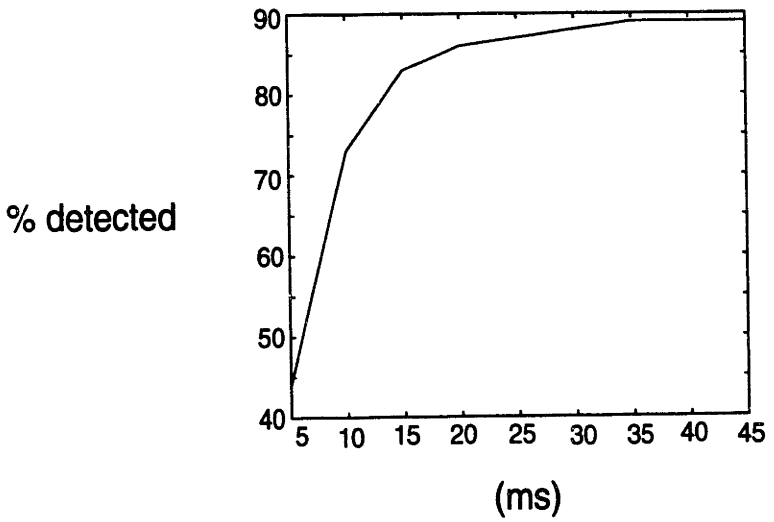

Figure 3.7: Temporal precision of the LAFF algorithm. The percentage of detected landmarks is plotted against the time difference from the hand-labeled transcription. These data were compiled from the results of the LAFF test set.

development to the test set. To determine the factors for this degradation, some extra evaluation and experiments were conducted. To explore the possibility of interspeaker variability, the overall detection rate of the development set was broken down by speaker. Table 3.3 shows this breakdown. As demonstrated by the wide difference in detection rates between speakers 1 and 4, interspeaker variability was a factor in the differences between the development and the test sets. To explore how well one speaker was represented by 20 utterances, speaker 2 re-recorded the sentences used in the development set. The overall detection rate for this second recording was 88%, which was a significant drop from 95%. Thus, the rates in either the development or test sets may not accurately represent the true rates for each speaker. These small experiments on interspeaker and intraspeaker variability show that the sample size in terms of the number of speakers and the number of sentences per speaker is too low to get an accurate estimate of the error rates. However, broad trends can still be demonstrated by these data sets. In the sections below, the results will be analyzed by phonetic context.

Figure 3.7 shows the temporal precision with which the landmark detector finds landmarks. 44% of the landmarks were detected within 5 ms of the hand-labeled transcription; 73% were within 10 ms; 83% were within 20 ms; and 88% were within 30 ms. A small percentage were beyond 30 ms, due to voiced obstruent closures. The

rest (9%) were either substituted or deleted. The role of a landmark is to identify approximate times around which further detailed processing needs to be performed to extract the desired acoustic cues. Depending on the acoustic cue to be extracted, the signal processing may be concentrated more to the left or more to the right of the hand-labeled landmark. For example, to find the presence of a voice bar at an obstruent closure, processing would be concentrated to the *right* of the hand-labeled landmark, which would be at the oral closure. To determine the place-of-articulation at this same landmark, processing would be concentrated to the *left* of the hand-labeled landmark. Since the precise placement of a landmark is dependent on the acoustic cue to be extracted, tuning the landmark detector to be more precise than it is at this stage would serve no real purpose. Nevertheless, even though the algorithm was not designed specifically to be precise, its time accuracy is still high.

3.3.2 G(lottis) landmarks

As demonstrated by Tables 3.1 and 3.2, the **g** detector was robust in most of the phonetic contexts it witnessed. Table 3.4 shows the detection rates of the development and test sets by phonetic category. The **g** deletions were due mostly to voiced obstruents. Voicing in the low frequencies reduced abruptness in Band 1 energy. Flaps were hard to detect because they were heavily voiced and had a short closure interval. If the closure interval was less than 20 ms, then the smoothing in the general processing stage of the landmark detection algorithm obscured the spectral discontinuity. In contrast to voiced obstruents, voiceless obstruents had a near 100% detection rate.

The bulk of the **g** insertions were due to semivowels. When a semivowel was implemented with a tight constriction, there was an abrupt weakening of high frequency energy, and sometimes low frequency energy as well. The ROR peaks that occurred were then picked up as **g** landmarks. Figure 3.6 shows that the two [w]s near 0.5 sec and 0.9 sec caused **g** insertions. There are two **g** neutrals at the creak at 0.8 sec.

3.3.3 S(onorant) landmarks

Compared to **g** landmarks, **s** landmarks were more difficult to detect. **S** landmarks relied on the energy change in Bands 2–5. In these bands, resonance peak amplitudes

	Phonetic category	*Landmark type*	*Detection rate (Development)*		*Detection rate (Test)*	
Outer **AC**	+v fric clos	**g**(lottis)	96%	(97)	96%	(47)
	+v fric rel	"	95%	(109)	98%	(58)
	−v fric clos	"	100%	(58)	*100%	(28)
	−v fric rel	"	100%	(107)	*100%	(26)
	flap clos	"	*88%	(26)	*88%	(8)
	flap rel	"	*85%	(27)	*75%	(8)
	+v stop clos	"	95%	(94)	98%	(48)
	+v stop rel	"	97%	(104)	95%	(44)
	−v stop clos	"	99%	(126)	100%	(62)
	−v stop rel	**b**(urst)	95%	(111)	89%	(72)
	nasal clos	**s**(onorant)	90%	(170)	72%	(53)
	nasal rel	"	90%	(105)	77%	(35)
	[l] clos	"	*71%	(21)	*33%	(9)
	[l] rel	"	81%	(36)	*76%	(17)
Intracons-onantal **AC**	stop clos	**b**(urst)	*100%	(20)	*30%	(10)
	stop rel	"	*95%	(19)	*100%	(10)
	fric clos	"	*100%	(8)	*90%	(29)
	affric rel	"	84%	(56)	95%	(40)
	nasal → fric	**g**(lottis)	*100%	(26)	*100%	(2)
	fric → nasal	"	*100%	(13)	*100%	(4)
Intracons-onantal **A**	velophar clos	**g**(lottis)	95%	(57)	*92%	(25)
	velophar rel	"	*25%	(4)	*100%	(8)
Inter-vocalic **A**	[ʔ] clos	**g**(lottis)	90%	(41)	*100%	(23)
	[ʔ] rel	"	100%	(51)	*100%	(23)
	−v [h] onset	"	–	(0)	–	(0)
	−v [h] offset	"	97%	(111)	97%	(72)
Total			94%	(1597)	91%	(761)

Table 3.4: Detection rates of the LAFF development and test sets, by phonetic categtory. An * next to a detection rate means that the number of tokens is less than 30 so the detection rate is unreliable. The number of tokens is given in parentheses.

	Left-reduced vCV		*Right-reduced VCv*		*Both-reduced vCv*		*Neither-reduced VCV*	
altogether	98%	(444)	87%	(367)	96%	(134)	89%	(390)
+v fric	100%	(41)	81%	(59)	100%	(13)	87%	(77)
+v stop	100%	(52)	80%	(49)	100%	(22)	95%	(59)

Table 3.5: Detection rates of the LAFF development set, by position with respect to reduced vowels. An example of each prosodic environment is given. The number of tokens is given in parentheses.

and resonance frequency ranges relied on phonetic context and can vary quite a bit. Most of the **s** deletions were next to semivowels, high vowels, back vowels, or reduced vowels. In these contexts, spectral change was small at the vowel/consonant transition. Table 3.4 shows that, of the **s** landmarks, nasals were detected better than [l]s. On average, [l]s were implemented less abruptly than nasals. The [m] closure at 1.2 sec in Figure 3.7 was missed by the **s** detector.

Most of the **s** insertions were due to semivowels. A semivowel's high frequency abruptness and low frequency steady-state voicing was mistaken for sonorant consonantal segments. In Figure 3.6, an **s** insertion occurred for each of the [w]s, at 0.55 sec and 0.83 sec. Another cause of **s** insertions was the nonsimultaneous change between high and low frequency energy at obstruent constrictions.

3.3.4 B(urst) landmarks

The **b** deletion rate was higher than the **g** deletion rate. Being low amplitude signals, bursts were often obscured by noise. For example, background noise or speech noise easily marred the silence interval preceding a burst. Also, the energy changes associated with **b** landmarks were sometimes too weak to cause an energy discontinuity. For example, the frication noise followed by a stop closure, as in the syllable boundary of "bathtub", sometimes did not cause a –opivot.

The **b** insertions were mainly due to extralingual noise during stop closures. A –**b** insertion just before 1.8 sec in Figure 3.7 marks the drop in energy after the [k] burst. A +**b** neutral marks a voiced stop burst for [b] at 1.4 sec, and another marks the creak at 1.6 sec.

3.3.5 The influence of vowel reduction

	Left-reduced vCV	*Right-reduced VCv*	*Both-reduced vCv*	*Neither-reduced VCV*
constriction duration	89 ± 13 ms (151)	63 ± 28 ms (111)	76 ± 21 ms (38)	67 ± 28 ms (106)
+v obstruent energy change	21 ± 5 dB (120)	16 ± 6 dB (144)	18 ± 5 dB (54)	17 ± 6 dB (144)

Table 3.6: Constriction duration and voiced obstruent low-frequency energy change at constriction, grouped by position with respect to reduced vowels. The number of tokens is given in parentheses.

Some insight into the effect of prosody on landmark detection can be gained by considering the position of a landmark in relation to reduced vowels. Reduced vowel effects on the development data are summarized in Table 3.5. Closure and release landmarks in left-reduced position (reduced vowel to the left, unreduced vowel to the right) had a higher detection rate than landmarks in right-reduced position (unreduced vowel to the left, reduced vowel to the right). This contrast was especially pronounced in voiced fricatives and voiced stops. Flaps were not counted because there were not enough tokens. Interestingly, when landmarks were in both-reduced position (vowels on both sides of the landmark are reduced), the detection rate was high. The neither-reduced position (vowels on both sides of the landmark are unreduced) had a detection rate intermediate between left-reduced and right-reduced.

The American English flapping rule says that alveolar stops are likely to be flapped in right-reduced environment. The finding that landmarks in left-reduced environment have a higher detection rate than landmarks in right-reduced environment suggests that voiced obstruents in general are likely to be reduced in right-reduced environment.

An acoustic analysis of the various prosodic environments shows why landmarks in right-reduced environment are harder to detect. One acoustic factor that affects landmark detection is constriction duration. The shorter the duration, the harder the landmark is to detect. The landmark detector relies on detecting energy change. If a constriction is too short, the energy change may be de-emphasized by the smoothing during the 6-band energy calculation. For voiced obstruents, voice bars de-emphasize the energy change in Band 1 even more. The first row in Table 3.6 shows the average constriction duration of singleton consonants in the four prosodic

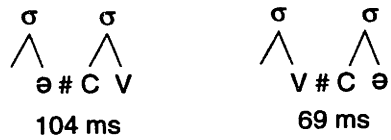

Figure 3.8: The effect of vowel reduction and syllable affiliation on consonant duration.

environments. The constriction duration in right-reduced environment is shortest, explaining in part why landmarks in this environment have the lowest detection rate. The constriction duration in left-reduced environment is longest, resulting in a higher detection rate. The durations in both-reduced and neither-reduced environments are in between, in agreement with the detection rates. Of relevance, [Turk, 1992] showed that, within a word, stop consonant constriction durations are shorter in left-stressed environments than in right-stressed environments.

In the above analysis, the syllable affiliation of the consonant was not taken into account. It has been hypothesized that reduction is syllable-affiliated, so that the effect of vowel reduction on a consonant is greater if that consonant belongs to the same syllable as the reduced vowel than if it belongs to a different syllable. To test this hypothesis, the constriction durations of Table 3.6 were grouped according to the word affiliation of the consonant. Consonants in word-medial position were not used because of the difficulty of deciding their syllable affiliation. The duration of consonants in left-reduced and right-reduced environments was noted when the consonant was affiliated with the right vowel. Fig. 3.8 illustrates the two cases considered. Consonants in left-reduced environment had an average duration of 104 $\pm$ 30 ms, while in right-reduced environment the average duration was 69 $\pm$ 25 ms. The average difference was 35 ms, which is bigger than the 26 ms difference when syllable affiliation was not considered. This finding supports the hypothesis that consonant reduction is affected not only by neighboring vowel reduction, but by the syllable affiliation of the consonant to the neighboring vowels as well.

In addition to constriction duration, the amount of energy change at closure and release also affects landmark detection. The bigger the change, the easier the detection, and vice versa. The change in the 20 ms–smoothed, Band 1 energy at closure and release was measured for all voiced obstruents. An energy change at closure was measured by subtracting the lowest energy level (in dB) during the con-

striction from the energy at a point directly preceding the closure transition in the vowel. At release, the measurement is made with the succeeding vowel. The second row in Table 3.6 shows that the energy change is 5 dB less, on average, for voiced obstruents in right-reduced environment than in left-reduced environment. The energy changes in the other prosodic environments were in between. This gradation in energy change is consistent with the landmark detector's performance in the four prosodic environments.

3.4 Chapter summary

This chapter described in detail the LAFF database, the landmark detection algorithm, and the results of running the algorithm on the LAFF database. Following acoustic and phonological rules, the utterances were carefully labeled with landmarks. The landmark detection algorithm was a hierarchical, knowledge-based algorithm employing abrupt spectral change and acoustic-phonetic information. The overall results of the algorithm consisted of an 8% deletion rate, a 1% substitution rate, and a 6% insertion rate. The error rate was 15% (14% not counting subsitutions). Broken down by landmark type, results showed that the **g** detector was the most robust, followed by the **b** detector, and finally by the **s** detector. Relying on a strong cue – voicing – the **g** detector had a 5% error rate (4% without substitutions). The **b** detector, being sensitive to random noise, had a 14% error rate (12% without subsitutions). Sensitive to vowel context, the **s** detector had a 57% error rate (56% without subsitutions). Semivowels and diphthongs were responsible for many of the insertions. Analyzed by reduced vowel environment, the detection rates of voiced obstruents showed that their landmarks were more likely to be missed in the left-unreduced-vowel/right-reduced-vowel environment than in other reduced vowel contexts. This prosodic context corresponds to the flapping environment for American English, suggesting that not only do alveolar stops but voiced obstruents in general tend to be reduced in this prosodic context. An acoustic analysis showed that constriction duration was shorter and low-frequency energy change was smaller in this prosodic context. The temporal precision with which the algorithm detected landmarks was examined. This temporal analysis showed that, although the algorithm was not originally tuned to be precise, it found most of the landmarks rather accurately. 44% of the landmarks were detected within

5 ms of the hand-labeled transcription, 73% were within 10 ms, 83% were within 20 ms, and 88% were within 30 ms. A few lay beyond 30 ms, due to voiced obstruent closures. The rest (9%) were either substituted or deleted.

The LAFF database is a carefully controlled database of clean, formal speech of four speakers. It was used primarily as an ideal setting for developing the main structure of the landmark detection algorithm. Because of the ideal recording environment and speaking style, the LAFF results may be unrealistic. Therefore, comparisons with other work is not attempted in this chapter. Instead, these comparisons will be delayed until Chapter 5, in which work with the TIMIT corpus is presented.

Chapter 4

Speech in noise

The study of how noise degrades a speech recognition system is relevant from a practical point of view. In any realistic situation, an automatic speech recognizer must be able to perform in background noise. This chapter will show how noise affects the landmark detection stage of the lexical access system. From an understanding of the noise characteristics, knowledge-based methods can be applied to the design of an algorithm specialized to work in a given signal-to-noise ratio (SNR).

In speech perception tests, [Miller et al., 1951] showed that human listeners' ability to recognize speech degraded with increasing noise. At 18 dB SNR, humans recognized isolated words correctly 80% of the time; at 0 dB SNR, the recognition rate decreased to 40%. For nonsense syllables, the recognition rates were 65% at 18 dB SNR and 40% at 0 dB SNR.

Machine recognition is also susceptible to noise. In an automatic speech recognition experiment, [Das et al., 1993] tested the IBM Tangora system, which is HMM-based, in several noisy environments. The recognition experiment was speaker-dependent using isolated words. The microphone used was omnidirectional. A subset of the results is presented in Table 4.1. A typical quiet office provided a 16 dB SNR

Test SNR	*Speaker*	*Error (%)*	
		Training SNR = 16 dB	*Training SNR = 10 dB*
16 dB	male	15%	50%+
	female	5%	50%+
10 dB	male	26%	35%
	female	50%+	50%+

Table 4.1: Error rates of a speaker-dependent, isolated-word recognition experiment using the IBM Tangora system at two SNRs.

environment, and a cafeteria with background babble provided a 10 dB SNR environment. When trained and tested in 16 dB SNR, the recognizer had a relatively low error rate. However, performance degraded severely when the recognizer was trained in 16 dB SNR but tested in 10 dB SNR. Surprisingly, when trained and tested in 10 dB SNR, the recognizer performed even worse. Performance was the worst when it was trained in 10 dB SNR and tested in 16 dB SNR. This example shows that, in noisy environments, a statistically-based system is sometimes unable to learn the speech information necessary to recognize speech because it gets confused between what is noise and what is speech. The IBM Tangora system shows that training and testing in the same environment does not always give the best performance when the environment is noisy. A knowledge-based system which concentrates only on the parts of speech that are not masked out by the noise may have an advantage over statistically-based systems in this case.

From these two examples, one can expect the performance of the landmark detector to degrade in increasing noise as well. One of the issues to be addressed in this chapter is: how quickly and in what manner does landmark detection degrade in increasing noise? Another issue is: can a knowledge-based algorithm, like the landmark detector, avoid the unexpected result of performing worse at an SNR it was designed for than at an SNR it was not designed for? In the following noise experiments, SNRs of 30 dB, 20 dB, 10 dB, and 0 dB were used. First, the algorithm customized for the clean utterances in the LAFF database was applied to the noisy speech. Based on the results, a minimalist algorithm, for very noisy signals, was designed and applied. This chapter will describe how the database of noisy speech was constructed, the details of the minimalist algorithm, and the results of running the LAFF and minimalist algorithms on the noisy speech database.

4.1 Database of speech in noise

The database of speech in noise was constructed by adding noise to the clean speech of the LAFF database, which was described in Section 3.1. The development set, consisting of 20 sentences $\times$ 4 speakers, was used in these noise experiments. The test set was not used because the development set was sufficient to show how landmark detection can degrade in increasing noise. Landmark labels for the utterances in this

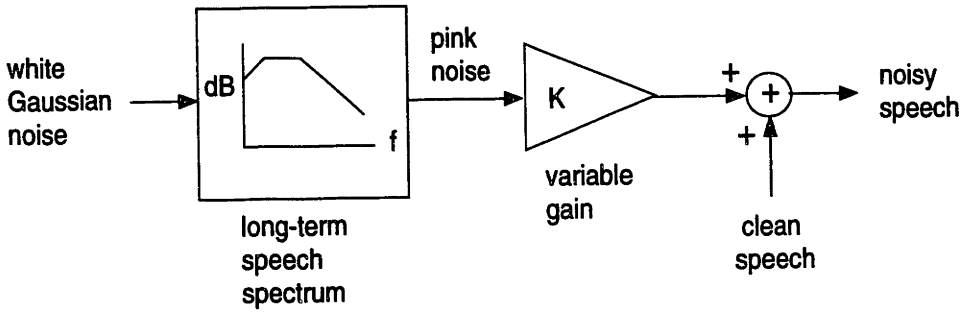

Figure 4.1: A block diagram of the scheme for adding noise to clean speech.

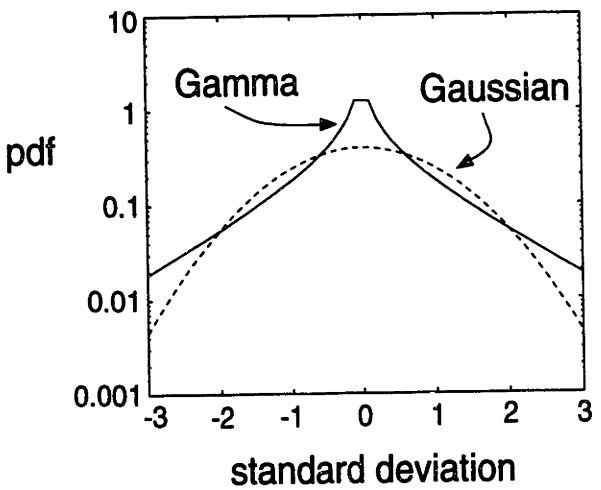

Figure 4.2: The Gamma PDF, approximating the PDF of speech, and the Gaussian PDF.

database were copied over from the LAFF database. Figure 4.1 shows a block diagram of the creation of noisy speech.

The added noise was made to resemble certain speech characteristics so that the simulated noisy environment approximated that of people talking in the background. First, white Gaussian noise was generated from a random noise generator. Figure 4.2 shows the probability distribution function (PDF) of white Gaussian noise overlaid with a Gamma PDF, which is a close approximation to the PDF of speech [Rabiner and Schafer, 1978]. The Gaussian PDF is given by

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{1}{2}\left(\frac{x}{\sigma}\right)^2} \tag{4.1}$$

where σ is the standard deviation. The Gamma PDF is given by

$$p(x) = \left(\frac{\sqrt{3}}{8\pi\sigma|x|}\right)^{1/2} e^{-\frac{\sqrt{3}|x|}{2\sigma}}. \tag{4.2}$$

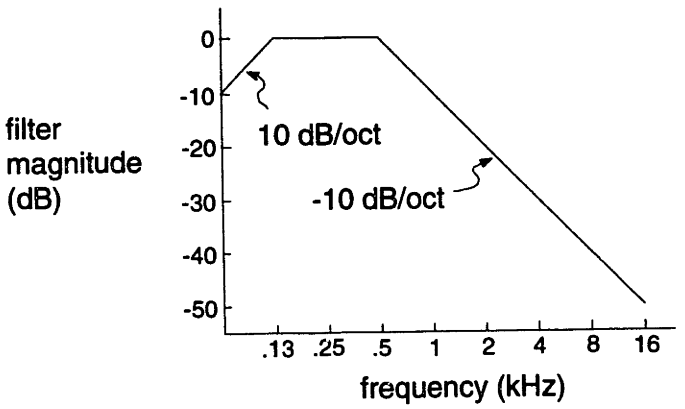

Figure 4.3: The magnitude of the transfer function approximating the long-term speech spectrum.

Although shown as finite, the Gamma function goes to infinity at zero. The Gaussian PDF is broader near zero and falls off more quickly than the PDF of speech. The white Gaussian noise was passed through a filter whose magnitude response approximated that of the long-term speech spectrum. Figure 4.3 shows the filter's response. It is flat from 125 Hz to 500 Hz, and falls off at 10 dB/octave on either side. This transfer function was derived from some measured long-term speech spectra, which can be found in [Rabiner and Schafer, 1978] and [Byrne et al., 1994]. The filtered noise, called pink noise, was then scaled by a factor, K, which was adjusted to attain the desired SNRs: 30 dB, 20 dB, 10 dB, and 0 dB. Finally, the scaled pink noise was added to the clean speech of the LAFF database to produce noisy speech.

Assuming that a speech signal has zero mean, the SNR of a recording is given by a ratio of variances [Rabiner and Schafer, 1978]:

$$\text{SNR} = \frac{\sigma_x^2}{\sigma_b^2 + (K\sigma_n)^2} \tag{4.3}$$

where σ_x^2 is the variance of the speech signal, σ_b^2 is the variance of the background noise level, and $(K\sigma_n)^2$ is the variance of the scaled additive noise. The variance of the speech signal, σ_x^2, was calculated from only the speech part of the original signal. Because recording levels were not equilibrated from speaker to speaker, σ_x^2 varied with speaker. Furthermore, because phonemic content varied from utterance to utterance, σ_x^2 of one utterance was not the same as the next within a speaker. For each speaker, a mean σ_x^2 was found by averaging over the variances of each individual utterance. The variance of the background noise level, σ_b^2, was calculated from the beginning

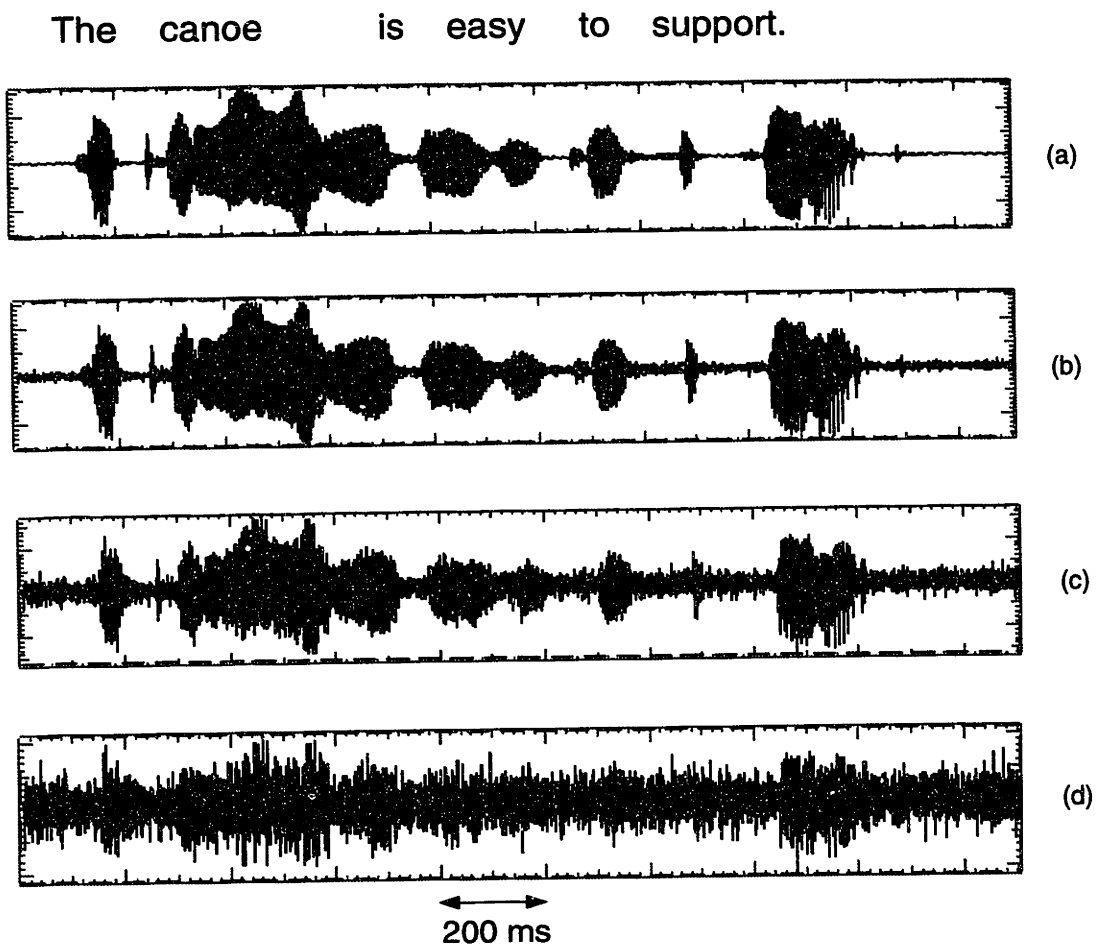

Figure 4.4: A speech waveform at several SNRs: (a) 30 dB, (b) 20 dB, (c) 10 dB, (d) 0 dB. The word transcription is written at the top.

of the recording to the time just before speech began. The variance of the additive noise, σ_n^2 was calculated from the generated Gaussian pink noise.

When no noise was added ($K = 0$), the SNR was the original SNR of the recording. For any one speaker, the SNR of each utterance deviated about ± 2 dB from that speaker's average SNR. The average original SNRs for the four speakers were found to be 30 dB, 25 dB, 29 dB, and 30 dB. The original SNR level will be listed as 30 dB from now on regardless of speaker, for convenience. K is nonzero only for the 20 dB, 10 dB, and 0 dB SNR signals.

Figure 4.4 shows the original 30 dB SNR speech waveform, and then the same waveform corrupted by pink noise to SNRs of 20 dB, 10 dB, and 0 dB. Figure 4.5 shows the corresponding spectrograms. At 30 dB SNR, all the spectral structure

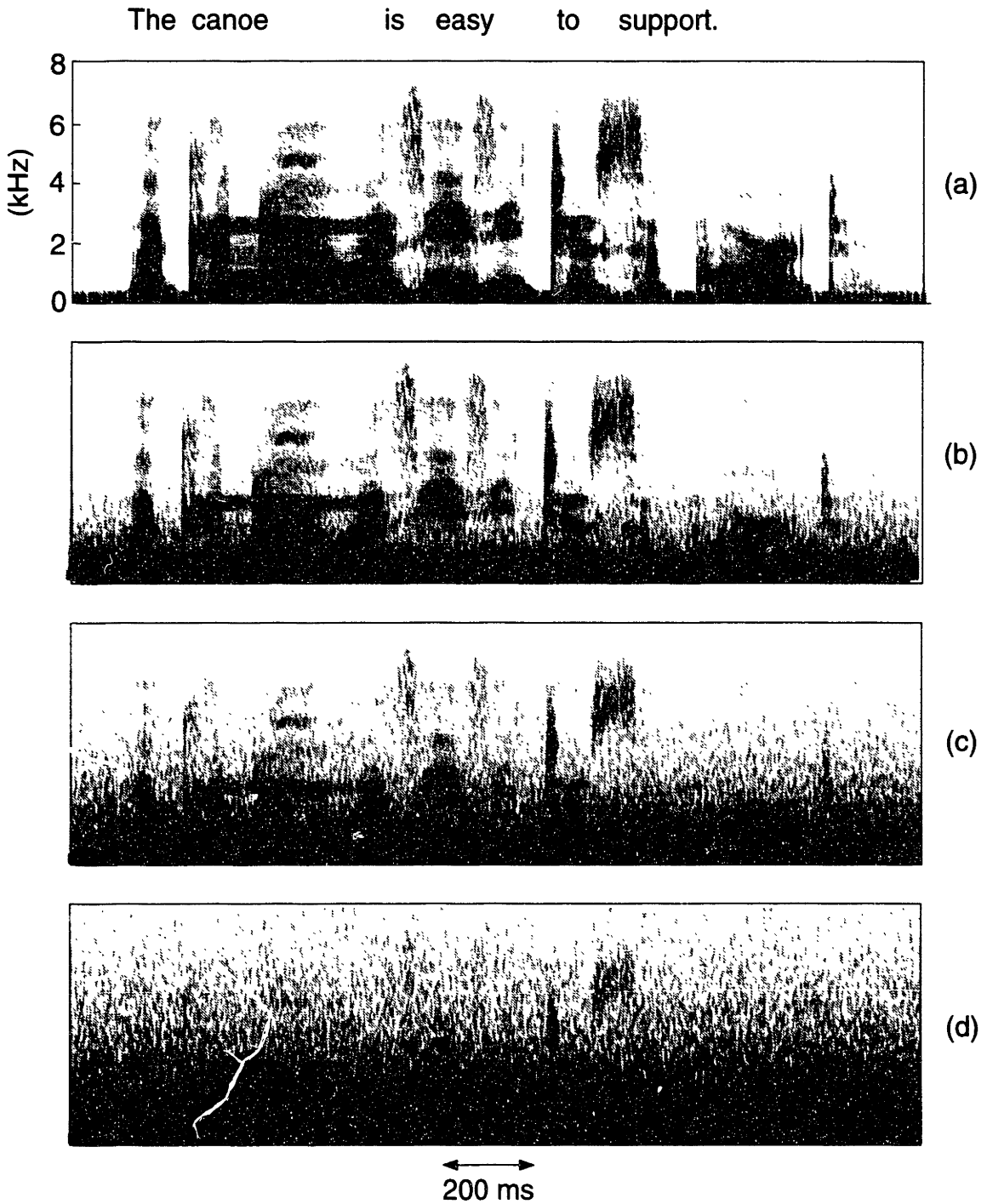

Figure 4.5: Spectrograms of the speech waveforms in Figure 4.4. SNRs shown are: (a) 30 dB, (b) 20 dB, (c) 10 dB, (d) 0 dB. The word transcription is written at the top.

shows up clearly. The spectral structure increasingly gets buried in noise as SNR decreases. By 0 dB SNR, almost all the consonant information is buried, leaving only some traces of pitch-accented vowels, aspirated stops, and strident fricatives.

4.2 Minimalist algorithm

In increasing noise, certain attributes of speech get masked out before others [Miller and Nicely, 1955]. To work optimally in noise, an algorithm must know the noise characteristics and the SNR of the operating environment, deduce which speech attributes are masked out and which are remaining, and then concentrate on finding the remaining attributes. Understandably, the information content of the output of such an algorithm is reduced, since noise masks out some of the desired information. This section presents a landmark detection algorithm designed to work in very noisy conditions (e.g. 0 dB SNR). Because of its simplicity, this algorithm will be called the "minimalist" algorithm.

The LAFF algorithm presented in Section 3.2 was hierarchical: first, it searched for the **g** landmarks; then it searched for the **s** and **b** landmarks in specific regions of the speech waveform delimited by the **g** landmarks. The LAFF algorithm was also stringent in its requirements: tests specific to **g**, **s**, and **b** landmarks were imposed on a landmark candidate before it could assume landmark status. The output was rich in information content: the sign and type of the landmark signified whether the point in the speech waveform was a closure or release, and whether there was a change in glottal vibration, sonorant consonantal constriction, or a burst of noise.

In the minimalist algorithm, this hierarchy and stringency are relaxed. The minimalist algorithm has the same general processing stage as the LAFF algorithm. A block diagram of general processing was shown in Figure 3.2. In brief, the algorithm calculates the spectrogram, and then, in a coarse and fine processing pass it calculates the 6-band energy and the 6-band ROR. It picks out the ROR peaks from the two passes and localizes them in time. The localized peaks are the inputs into the next stage, which is where the LAFF and minimalist algorithms differ. In the minimalist algorithm, the Band 1 peaks are automatically landmarks. Landmark pairing and a minimum vowel requirement are not imposed. Bands 2–5 peaks in the vicinity of each Band 1 peak are grouped with the Band 1 peak. Of the remaining Bands 2–5

SNR	*# Tokens*	*Del.*	*Subs.*	*Ins.*	*Neut.*	E_1	E_2
30 dB	1597	4%	2%	10%	7%	16%	14%
20 dB	"	7%	3%	10%	6%	20%	17%
10 dB	"	20%	8%	9%	5%	37%	29%
0 dB	"	66%	14%	8%	3%	88%	74%

Table 4.2: Overall results of running the LAFF algorithm on noisy speech.

peaks, pivots are found in a manner similar to the LAFF algorithm: the biggest peak in absolute terms is chosen as the pivot of each uni-sign block of peaks. No division is made between pivots of sonorant/voiced regions and opivots of obstruent/unvoiced regions. Each pivot is automatically a landmark. The criteria of steady-state, high-frequency abruptness, and silence are not imposed. The landmarks found with the minimalist algorithm are "generic" landmarks because they are unmarked by sign or type. The minimalist landmark just picks out the points in the speech waveform where there may be abrupt spectral change but does not elaborate further on these points.

4.3 Noisy speech results and discussion

Two experiments were carried out on noisy speech. The purpose of the first experiment was to see how a knowledge-rich algorithm would perform in increasing noise. The algorithm used was the LAFF landmark detector. The purpose of the second experiment was to see whether performance improved for certain SNRs if a less stringent algorithm was used. The second experiment involved the minimalist algorithm. This section presents results of these two experiments.

4.3.1 Results with the LAFF algorithm

In the first experiment, the criteria for a detection were the same as in Section 3.3. That is, the correctly detected landmark had to be within ±30 ms of the labeled landmark (+80 ms for a voiced obstruent closure), and be of the same sign and type as the labeled landmark. If it was of the wrong type but correct sign, it was a substitution. All other hand-labeled landmarks were deletions. Extra landmarks found by the landmark detector were either insertions or neutrals. Rates are still given by equations (3.1) through (3.7). Table 4.2 shows the overall results of running

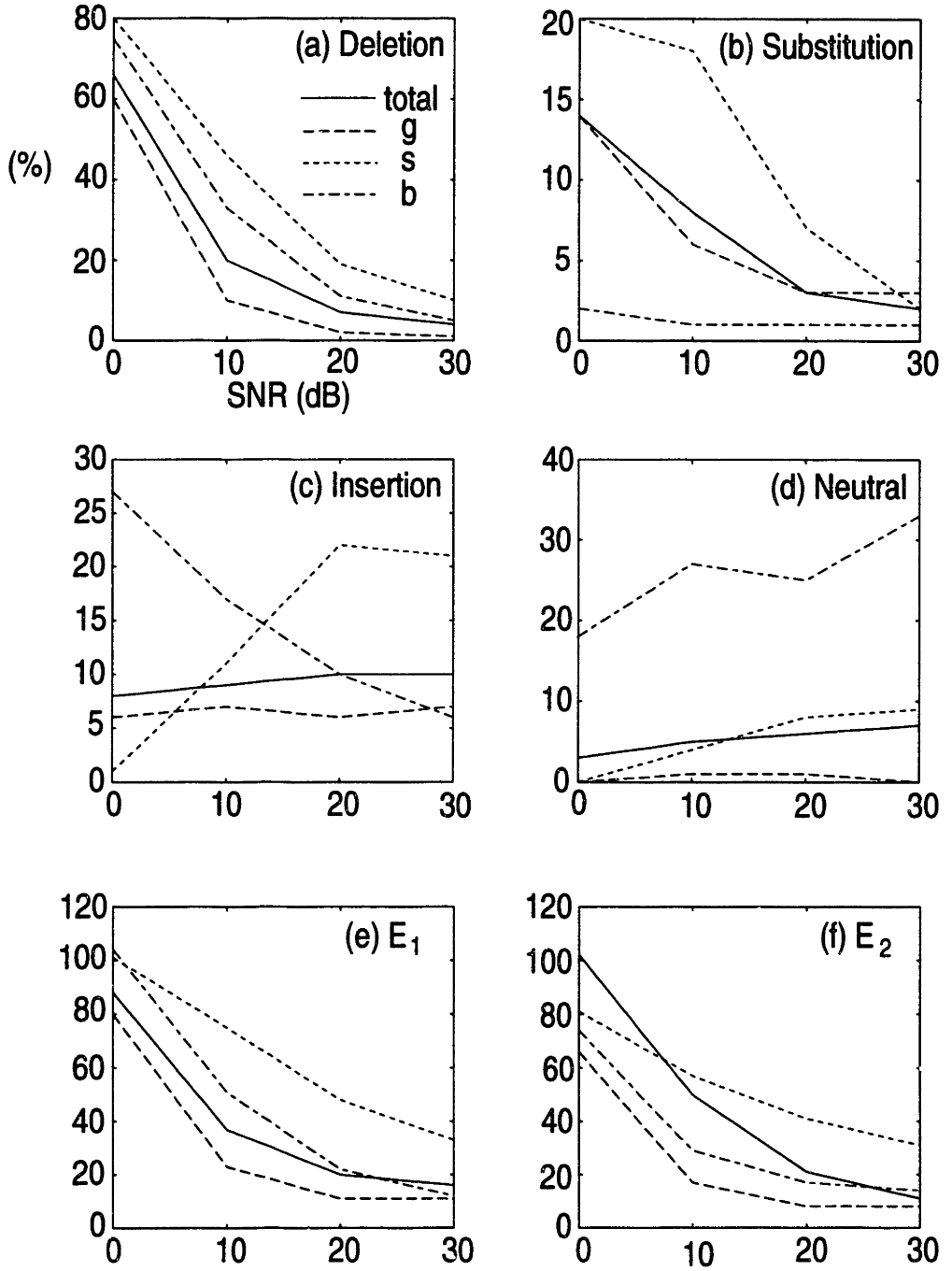

Figure 4.6: Results of the LAFF algorithm on noisy speech at four SNRs. (a) Deletion rate. (b) Substitution rate. (c) Insertion rate. (d) Neutral rate. (e) Error rate counting substitutions. (f) Error rate not counting substitutions.

SNR	*Type*	*# Tokens*	*Del.*	*Subs.*	*Ins.*	*Neut.*	E_1	E_2
30 dB	**g**(lottis)	1052	1%	3%	7%	0%	11%	8%
	s(onorant)	332	10%	2%	21%	9%	33%	31%
	b(urst)	213	5%	1%	6%	33%	12%	11%
	Total	1597	4%	2%	10%	7%	16%	14%
20 dB	**g**(lottis)	1052	2%	3%	6%	1%	11%	8%
	s(onorant)	332	19%	7%	22%	8%	48%	41%
	b(urst)	213	11%	1%	10%	25%	22%	21%
	Total	1597	7%	3%	10%	6%	20%	17%
10 dB	**g**(lottis)	1052	10%	6%	7%	1%	23%	17%
	s(onorant)	332	46%	18%	11%	4%	75%	57%
	b(urst)	213	33%	1%	17%	27%	51%	50%
	Total	1597	20%	8%	9%	5%	37%	29%
0 dB	**g**(lottis)	1052	60%	14%	6%	0%	80%	66%
	s(onorant)	332	80%	20%	1%	0%	101%	81%
	b(urst)	213	75%	2%	27%	18%	104%	102%
	Total	1597	66%	14%	8%	3%	88%	74%

Table 4.3: Results of running the LAFF algorithm on noisy speech, listed by landmark type.

the LAFF algorithm on speech at SNRs of 30 dB, 20 dB, 10 dB, and 0 dB. The solid lines in Figure 4.6 provide a graphical representation of these results. At 30 dB SNR, the speech is the original clean waveform with no noise added, so these results are the same as those in Section 3.3. Examining the results, one sees that the deletion rate increased as SNR decreased. At 20 dB SNR, the deletion rate was still relatively low. At 10 dB SNR, the deletion rate had increased but the majority of the landmarks were still being detected. At 0 dB SNR, the deletion rate was understandably high, since most of the spectral structure in speech was buried under noise. The substitution rate increased with decreasing SNR, as the delimitation of sonorant and obstruent regions by the **g** landmarks broke down and the **s** and **b** landmarks were not found in their usual regions. The insertion and neutral rates decreased with decreasing SNR, apparently because the speech cues that caused the insertions and neutrals at high SNRs became increasingly buried in noise at low SNRs. Overall, error rates increased with decreasing SNR.

In more detail, Table 4.3 breaks the results down by landmark type. The dashed lines in Figure 4.6 give a graphical representation of this breakdown. At every SNR level, **g** landmarks were the most robust of all three landmark types in

SNR	*# Tokens*	*Del.*	*Subs.*	*Ins.*	*Neut.*	E_2
30 dB	1597	2%	–	33%	5%	35%
20 dB	"	3%	–	29%	6%	32%
10 dB	"	15%	–	20%	3%	35%
0 dB	"	61%	–	6%	1%	67%

Table 4.4: Overall results of running the minimalist algorithm on noisy speech. By design, the substitution rate is zero.

terms of having the lowest deletion, insertion, and error rates. (One exception was at 0 dB SNR, when the **s** insertion rate was lower). The **g** deletion rate remained relatively low, down to 10 dB SNR. Voicing was a robust speech attribute in noise, as [Miller and Nicely, 1955] also noted with their experiments of speech in white noise. The **g** substitution rate increased substantially below 10 dB SNR as the pink noise masked out low frequency information and Bands 2–5 were increasingly relied upon to generate landmarks. The **g** insertion rate remained relatively low and steady as SNR varied.

Of all three landmark types, **s** landmarks were the most sensitive to noise. Below 20 dB SNR, the performance of the **s** detector degraded dramatically. Not only did the added pink noise mask out spectral abruptness in the mid-bands, but it also disrupted the steady-state portion during the sonorant consonantal segment. The **s** substitutions were due to the spectral abruptness in Bands 2–5 being picked up as **g** or **b** landmarks. In contrast to the deletion rate, the **s** insertion rate dropped below 20 dB SNR. The characteristics of semivowels and other segments causing **s** insertions at 30 dB SNR were masked out by noise below 20 dB SNR.

Less robust than the **g** detector but more so than the **s** detector, the **b** detector performed well when the SNR was more than 10 dB, but degraded quickly below that. **B** landmarks were sensitive to added noise, mainly because the added noise masked out abrupt changes in bursts and closures after bursts, but also because it disrupted the "silence" during closure. The **b** substitution rate remained insignificant at all SNRs. The **b** insertion rate was the only insertion rate that increased with decreasing SNR. These insertions were caused by the random fluctuations in the added pink noise.

4.3.2 Results with the minimalist algorithm

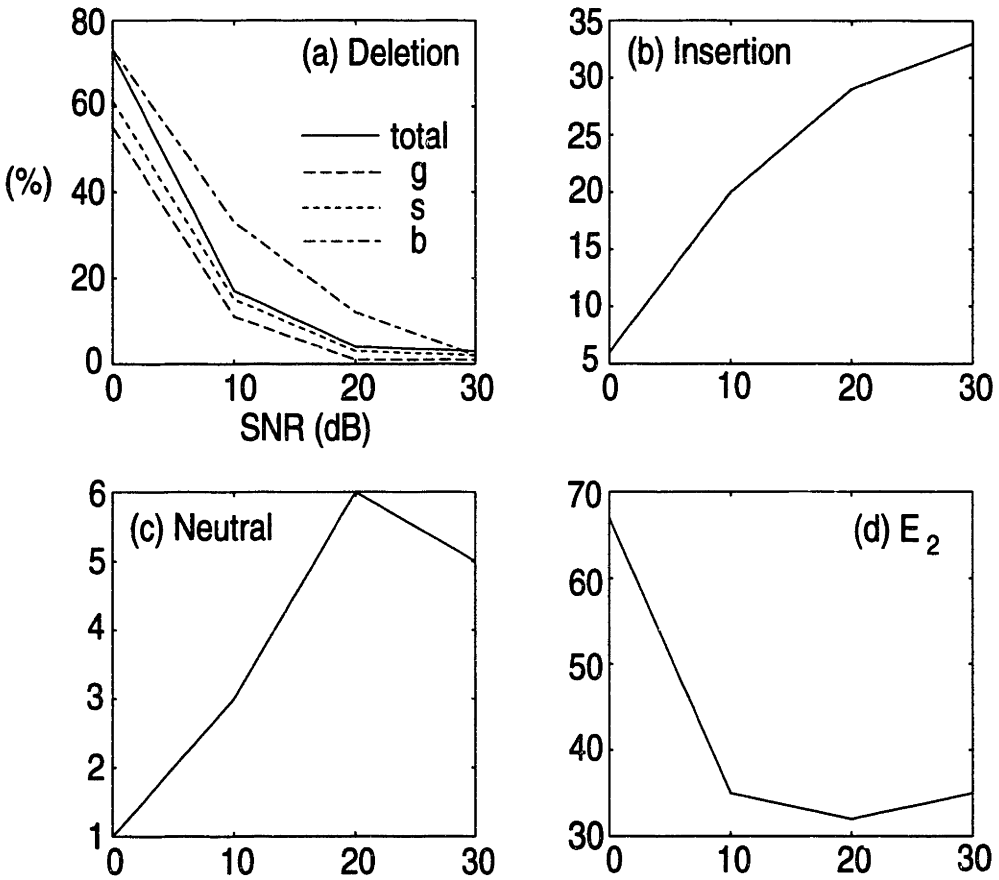

Figure 4.7: Results of the minimalist algorithm on noisy speech at four SNRs. (a) Deletion rate. (b) Insertion rate. (c) Neutral rate. (d) Error rate.

In the experiment involving the minimalist algorithm, the criteria for a detection were necessarily relaxed. Since the output of the minimalist algorithm was simply times of generic landmarks with no type or sign attached, a detection was merely a generic landmark within ±30 ms of the labeled landmark (+80 ms for a voiced obstruent closure). All other landmarks were insertions or neutrals. The substitution rate was zero by design. The insertion and neutral rates for the individual landmark types were undefined, since the generic landmarks did not specialize to any type. Table 4.4 shows the overall results of running the minimalist landmark detector on noisy speech. The solid lines in Figure 4.7 give a graphical representation of these results. The trend was the same as with the LAFF algorithm: as SNR went down, the deletion rate increased while the insertion rate decreased. However, the major difference was in the actual values of these rates. At all SNRs, the minimalist deletion rate was lower than LAFF's. This result was due to the relaxed criteria in the minimalist algorithm. However, the relaxed criteria also produced a higher insertion rate for SNRs 10 dB and above. The minimalist insertion rate sunk below the LAFF insertion rate at 0 dB SNR. In contrast, the LAFF insertion rate remained low and steady around 8–10% for all SNRs.

Table 4.5 shows the results of the minimalist algorithm broken down by landmark type. The dashed lines in Figure 4.7 are a graphical representation of this breakdown. The main difference here was that the **s** deletion rate was significantly lower for the minimalist algorithm than for the LAFF algorithm. The steady-state criterion in the LAFF algorithm was apparently too stringent for noisy speech.

Several conclusions are made concerning the effectiveness of the LAFF and minimalist algorithms in noise. Figure 4.8 shows the E_2 error rates for the LAFF and minimalist algorithms. E_2 is the error rate not counting substitutions. At 30 dB SNR, the LAFF E_2 rate was lower than LAFF's by 21%. Furthermore, the LAFF algorithm gave more information in its output than the minimalist algorithm. Therefore, the LAFF algorithm outperformed the minimalist detector at 30 dB SNR. When speech cues are not masked by noise, exploiting acoustic-phonetic information helped performance. At 0 dB SNR, the other extreme, the minimalist E_2 rate was lower than LAFF's, by 6%. When speech cues are masked by noise, imposing acoustic-phonetic constraints as LAFF did actually decreases performance; instead, the minimalist algorithm which looked for just the few remaining strong cues outperformed the LAFF

SNR	*Type*	*# Tokens*	*Del.*	*Subs.*	*Ins.*	*Neut.*	E_2
30 dB	**g**(lottis)	1052	1%	–	–	–	–
	s(onorant)	332	3%	–	–	–	–
	b(urst)	213	2%	–	–	–	–
	Total	1597	2%	–	33%	5%	35%
20 dB	**g**(lottis)	1052	1%	–	–	–	–
	s(onorant)	332	4%	–	–	–	–
	b(urst)	213	12%	–	–	–	–
	Total	1597	3%	–	29%	6%	32%
10 dB	**g**(lottis)	1052	11%	–	–	–	–
	s(onorant)	332	17%	–	–	–	–
	b(urst)	213	33%	–	–	–	–
	Total	1597	15%	–	20%	3%	35%
0 dB	**g**(lottis)	1052	55%	–	–	–	–
	s(onorant)	332	72%	–	–	–	–
	b(urst)	213	73%	–	–	–	–
	Total	1597	61%	–	6%	1%	67%

Table 4.5: Results of running the minimalist algorithm on noisy speech, listed by landmark type. By design, the substitution rate is zero, and the insertion, neutral, and error rates for individual landmark types are undefined.

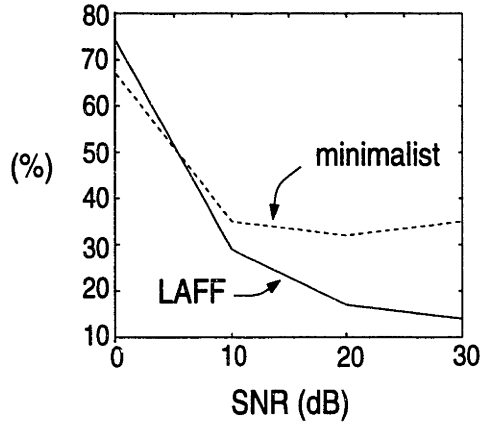

Figure 4.8: The error rates of the LAFF and minimalist algorithms as a function of SNR. The error rates do not count substitutions.

algorithm. At the intermediate SNRs of 20 dB and 10 dB, the LAFF algorithm arguably outperformed the minimalist algorithm, although optimum performance would be expected to be achieved with an algorithm of intermediate complexity between the LAFF and the minimalist algorithms.

4.4 Chapter summary

Landmark detection of speech in added noise was studied. The noise characteristics approximated those of the long-term speech spectrum. Two experiments were carried out. The first employed the LAFF algorithm, which was stringent and hierarchical in its search for acoustic-phonetic cues in the acoustic waveform. This algorithm worked best at a relatively high SNR of 30 dB, when much of the acoustic-phonetic information was still present in the acoustic waveform. At 0 dB SNR, this algorithm performed poorly. The second experiment employed the minimalist algorithm, which was a nonhierarchical, relaxed algorithm concentrating on only the strong cues that were least likely to be masked out by noise. This algorithm outperformed the LAFF algorithm at 0 dB SNR. At 30 dB SNR, however, it had a very high insertion rate. The results of these two experiments show that using knowledge about speech and the operating environment improves landmark detection. This result is in direct contrast to the IBM Tangora case (presented at the beginning of this chapter), in which training on the noisy operating environment actually degraded performance. In terms of the individual landmark types, **g** landmarks were robust down to 10 dB SNR, **b** landmarks were robust to between 10 dB and 20 dB SNR, and **s** landmarks were robust down to only 20 dB SNR.

To deal effectively with noise, a noise-cancelling front-end, like a multi-microphone array, could be added before the speech recognition algorithm to reduce the noise in the signal. The algorithm can then operate on the resulting signal, concentrating on the speech cues that are not masked out by the noise as before.

Chapter 5

TIMIT speech

In this chapter, an experiment with the TIMIT database is described. Several motivations exist for using TIMIT. First, TIMIT is a standard database in the speech community and, as such, provides a basis for comparison with other work. Second, the recording conditions for TIMIT were distinctly different from those for the LAFF database. Thus, TIMIT provides an opportunity to demonstrate systematic and knowledge-directed modifications to the landmark detection algorithm in accordance with this new recording environment. Third, because TIMIT is larger and more comprehensive than LAFF, landmark detection can be tested in a variety of phonetic contexts and speaker accents. This chapter will describe the TIMIT database and how it differs from the LAFF database. It will then explain how the landmark detection algorithm was customized to the TIMIT database. Finally, it will present the results of two landmark detection experiments using the TIMIT database.

5.1 TIMIT database

A description of the TIMIT database can be found in [Fisher et al., 1986] and [Zue et al., 1990a]. LAFF and TIMIT share some similarities: both were recorded in a quiet environment at 16 kHz, and both contain isolated, read utterances. However, TIMIT differs from LAFF in some important ways. TIMIT utterances are cleaner: the TIMIT utterances used have an average SNR of 42 dB while LAFF utterances have an SNR of 30 dB. Furthermore, TIMIT was recorded with a Sennheiser close-talking microphone while LAFF was recorded with an omnidirectional microphone. Because the Sennheiser microphone was directed toward the mouth, it received sounds radi-

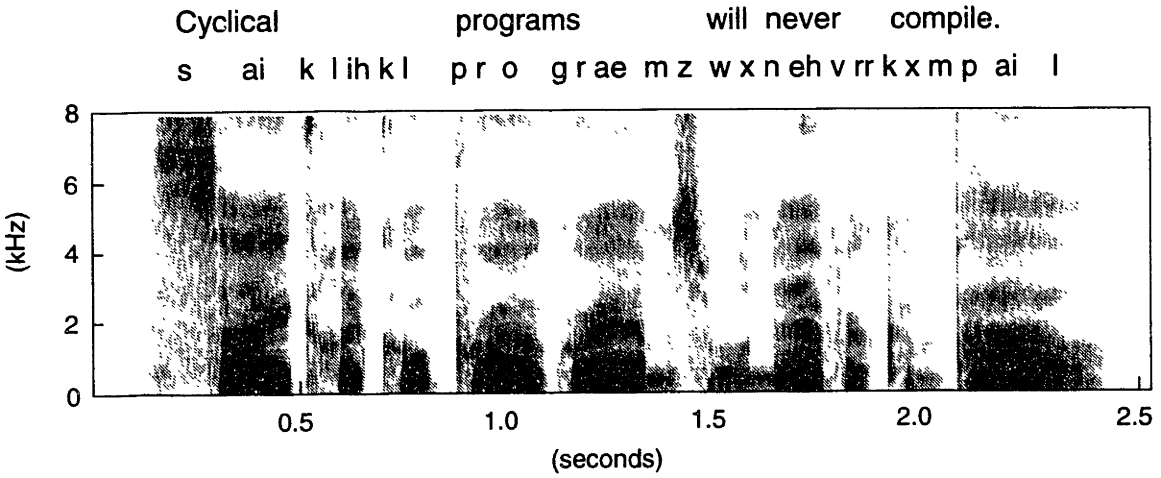

Figure 5.1: A spectrogram of a TIMIT utterance. The word transcription is given at the top.

ating from the mouth more strongly than sounds radiating from the nose or throat. Thus, a nasal sometimes appeared only as a weak voicing murmur on the spectrogram, and voice bars for voiced obstruent closures sometimes disappeared altogether. Figure 5.1 shows a spectrogram of a TIMIT utterance. Comparing it to the LAFF utterance in Figure 3.3, one can see that there is a reduction in amplitude during the nasals in the TIMIT utterance (at 1.4 sec, 1.6 sec, and 2.0 sec). The relatively close placement of the Sennheiser microphone to the mouth and its tilted frequency response results in a 2–4 dB/octave boost at frequencies above 2 kHz.[1] High-frequency energy is thus stronger than in LAFF utterances. The TIMIT anti-aliasing filter has a 3 dB cut-off frequency at 8 Hz with an almost vertical roll-off.[2] The signal near 8 kHz is thus aliased, but this effect is inconsequential for the purpose of landmark detection. In Figure 5.1, spectral energy at the high frequencies is more intense than in the LAFF utterance and extends clearly up to 8 kHz. The speakers in TIMIT spoke somewhat more casually than the speakers in LAFF; thus, voiced obstruents had a higher probability of being heavily voiced, schwas devoiced, and closures loosely formed. TIMIT utterances tended to be longer than LAFF utterances; a 1.5 second difference, for example, was not uncommon.

The TIMIT corpus is made up of 630 native American English speakers from 8 dialectal regions of the United States speaking 10 sentences each. The corpus is

[1] Personal communication with William Fisher.
[2] Personal communication with George Doddington.

composed of three groups of sentences. (1) The "sa" group is all 630 speakers speaking 2 calibration sentences. (2) The "sx" group was designed to be phonetically compact with emphasis on as complete a coverage of phonetic pairs as was practical. It consisted of 450 sentences each spoken by 7 speakers. (3) The "si" group is composed of 1890 randomly selected sentences, mostly from the Brown corpus [Kucera and Francis, 1967], each spoken by 1 speaker. For the experiment here, the "sx" sentences were used to form a development and test set. The development set is a subset of the TIMIT recommended training set while the test set is a subset of the TIMIT recommended test set. The development set consisted of 16 speakers (8 female, 8 male) speaking a total of 80 utterances. The sentences were chosen to not repeat and to be equally balanced among the 8 dialect regions of the United States. The test set was independent of the development set in terms of speakers and sentences. It consisted of 16 new speakers (8 female, 8 male), also spanning the 8 dialect regions, speaking a total of 48 utterances. Together, the development and test sets contained 684 distinct words, of which 46% are monosyllabic, 34% bisyllabic, 14% trisyllabic, 4% quadsyllabic, and 2% quinsyllabic. On average, the words in TIMIT has more syllables and consonant clusters than those in LAFF. Appendix A lists the sentences used in the development and test sets.

The landmark labels were derived from the phonetic labels provided with TIMIT. Some systematic changes were made to conform to the LAFF labeling conventions described in Section 3.1. Nasal flaps were replaced by [n]s. Labels for intervocalic semivowels were removed. Labels for nonabrupt /l/s were removed. For stop releases, if the burst and voice onset labels were less than 20 ms apart, they were collapsed into one stop release landmark. Fricative-to-fricative labels were removed. Almost all obstruent labels were kept, including heavily-voiced weak fricatives. The resulting landmark labels comprised 75% of the original TIMIT labels.

5.2 TIMIT algorithm

In order to see how to modify the existing landmark detector to suit the TIMIT database, a trial run was executed using the LAFF landmark detector on the TIMIT utterances. The results of this trial run are presented in the next section. Performance, understandably, degraded compared to the LAFF experiment. Results

indicated that the landmark type-specific processing stage of the algorithm (see Figure 3.1) had to be modified. A description of these modifications follows.

5.2.1 G(lottis) detector

Because TIMIT utterances were longer than LAFF's, the difference in energy between a pitch-accented vowel and the end of a TIMIT utterance was bigger than in a LAFF utterance. The LAFF algorithm sometimes deleted **g** landmarks at the end of a TIMIT utterance because the the minimum vowel requirement was not satisfied. In order to correct for these erroneous deletions, two new acoustic measures are used, in place of Band 1 ROR peaks, as the primary indicators of **g** landmarks: Band 1 energy and zero-crossing rate (ZCR). Band 1 energy level is used to find the voiced regions of the waveform. Whenever Band 1 energy exceeds a certain threshold, the beginning of a voiced region is noted. This voiced region ends when Band 1 energy falls back below the threshold. When low frequency frication and aspiration noise contribute to Band 1 energy, the ZCR is used to adjust the boundaries of the voiced regions away from areas of high ZCR. Each resulting voiced region is then required to be at least of a certain length so that spurious signals due to creaks and bursts can be excluded. The boundaries of the voiced regions become **g** landmarks.

The parameter values for the above procedure are given below. Some of them are adapted from the minimum vowel requirement in the LAFF algorithm while others are empirical. For a voiced region, Band 1 energy is required to be above the following threshold: 25 dB below the maximum Band 1 energy in the utterance. Again, this drop is an upper bound on the difference between the F_1 amplitude of a pitch-accented vowel and that of a schwa. The LAFF algorithm used a 20 dB drop; the extra 5 dB is for the longer utterances in TIMIT. At the times when Band 1 energy crosses this threshold, if the ZCR is higher than 4000 crossings/sec, then the putative voiced region boundary is adjusted (to the left if Band 1 energy is falling; to the right if Band 1 energy is rising) until the ZCR falls below 4000 crossings/sec. This adjustment excludes sounds consisting predominantly of noise. The ZCR is calculated every 1 ms with 20 ms frames. A +/- pair of boundaries is deleted if it spans less than 20 ms. This durational requirement is the same as in LAFF.

The scheme using Band 1 energy and ZCR finds most of the **g** landmarks.

However, heavily voiced obstruents tend to be missed. Therefore, the localized Band 1 ROR peaks from the general processing stage are added to the **g** landmarks if these peaks fall within the voiced regions delimited by the first scheme. These ROR peaks do not necessarily occur in pairs. As in LAFF, peak insertion and deletion are imposed so that the final set of **g** landmarks are paired.

5.2.2 S(onorant) detector

The **s** detector is adjusted to suit the characteristics of the close-talking microphone. Because nasals are substantially reduced in amplitude due to the directionality of the microphone, the threshold for Band 1 ROR peak detection in the fine processing pass of the general processing stage is raised from 6 dB to 9 dB. The higher threshold makes nasals more likely to be detected as **s** than **g** landmarks.

Bands 2–5 pivots are found as before. Then the pivots are passed through the abruptness criterion. In the LAFF algorithm, abruptness was measured on a high-pass signal with a low-frequency cutoff of 1.3 kHz and pre-emphasized by 6 dB/octave. The 1.3 kHz cutoff was chosen so that the first spectral prominence in nasals did not mollify the high-frequency abruptness measure. For the TIMIT algorithm, the low-frequency cutoff is lowered to 1 kHz in order to capture changes in a broader spectrum; because TIMIT nasals are weak, the first spectral prominence in the nasal does not upset the high frequency abruptness measure. Since TIMIT utterances already have a 2–4 dB/octave boost above 2 kHz, the high-pass signal is pre-emphasized by only 3 dB/octave instead of the 6 dB/octave used for LAFF.

To allow for more variation in sonorant consonantal segment durations, the steady-state criterion is altered slightly to be more flexible. For intervocalic pivots, the steady-state region has to extend to a certain margin around the pivots, rather than be proportional to the distance between the two pivots. For pivots next to obstruents, the requirement remains the same as that of LAFF – a 20 ms steady-state minimum is required to be adjacent to the pivot.

5.2.3 B(urst) detector

The LAFF algorithm measured silence with respect to the background level. The background level was high enough that small variations in noise did not produce

large dB excursions. Because TIMIT has a lower noise floor, small variations in the background noise level can cause a big dB change. Therefore, the way in which the **b** detector measures silence is altered. For the TIMIT algorithm, the silence reference is chosen to be a dB drop from the maximum energy during the voiced portions of the signal. The voiced portions are already known from the ±**g** landmark pairs. This maximum energy is very likely a stressed vowel level.

In the LAFF algorithm, silence was measured with the energy waveforms in Bands 1–6. Each energy waveform was recorded with respect to its background level, causing unequal dB shifts in the bands. This decoupling of a band's energy with respect to the others made silence measurement unreliable. In the TIMIT algorithm, silence is measured using one high-pass energy signal with a low-frequency cutoff of 1.3 kHz. As before, the low frequencies are left out in order to avoid voice bars. Using one energy signal covering the entire frequency range of interest avoids the unequal dB shifts of individual bands. Silence regions are picked out whenever this high-pass energy signal drops below –40 dB with respect to the stressed vowel level. Only the unvoiced regions, delimited by pairs of –/+**g** landmarks, are searched. Opivots are found as before, and requirements on the length of a silence region and proximity to the opivots remain the same.

5.3 TIMIT results and discussion

In this section, the results of running the LAFF and TIMIT algorithms on the TIMIT database are presented. The results of the LAFF algorithm in fact provided the knowledge on how to construct the TIMIT algorithm.

5.3.1 Results with the LAFF algorithm

Table 5.1 summarizes the results of running the LAFF algorithm on the TIMIT development data. Compared to the results of running the LAFF algorithm on the LAFF test data (see Table 3.2), the overall performance on the TIMIT database, in all respects, degraded. For each landmark type, deletion, substitution, and insertion rates each increased. As was also seen in Sections 3.3 and 4.3, the **g** landmark detector was the most robust across the database change, followed by the **b** detector and then the **s** detector.

Landmark type	*# Tokens*	*Del.*	*Subs.*	*Ins.*	*Neut.*	E_1	E_2
g(lottis)	1315	5%	2%	4%	4%	11%	9%
s(onorant)	337	25%	23%	29%	4%	77%	54%
b(urst)	492	16%	8%	5%	9%	29%	21%
Total	2144	10%	7%	8%	5%	25%	18%

Table 5.1: Results of the TIMIT development set using the LAFF algorithm.

The **g** deletion rate increased partly from the **g** detector missing the reduced-amplitude, end-of-utterance voiced regions, and partly from missing voiced obstruents. The **g** insertion rate increased because the high SNR in TIMIT magnified the effects of low-frequency noises due to stop bursts and non-speech noises. Semivowel and diphthong transitions contributed to **g** insertions as before. The **g** neutral rate increased because there were more creaks in the TIMIT database than in the LAFF database.

There was a large increase in the **s** substitution rate. The main reason was that TIMIT nasals were much weaker than LAFF nasals. Many of the nasal landmarks were picked up as **g** instead of **s** landmarks. As before, the causes of **s** insertions were mainly semivowel/diphthong transitions and the transitions in and out of obstruent constrictions. These insertions were more frequent in TIMIT. The **s** neutral landmarks were due to creaks and voice bars.

The **b** deletion rate went up primarily because the method of determining a silence interval from the six frequency bands was unreliable, as mentioned in the previous section. Many stop closures were missed from slight fluctuations in the noise floor. These slight fluctuations also gave rise to sudden changes in energy during silence gaps, causing an increased insertion rate. As before, the **b** neutral landmarks were comprised of short affricate releases and creaks.

5.3.2 Results with the TIMIT algorithm

Table 5.2 shows the results of running the TIMIT algorithm on the TIMIT development data. Compared to Table 5.1, performance improved. Overall, the deletion and substitution rates were lower, while the insertion rate remained the same.

The **g** deletion rate decreased because the TIMIT algorithm was able to catch the end-of-utterance voiced regions effectively with its energy and ZCR measurements.

The **s** substitution rate went down because of the rise in Band 1 ROR threshold

Landmark type	*# Tokens*	*Del.*	*Subs.*	*Ins.*	*Neut.*	E_1	E_2
g(lottis)	1315	1%	2%	3%	4%	6%	4%
s(onorant)	337	25%	17%	23%	9%	65%	48%
b(urst)	492	4%	3%	10%	9%	17%	14%
Total	2144	6%	4%	8%	6%	18%	14%

Table 5.2: Results of the TIMIT development set using the TIMIT algorithm.

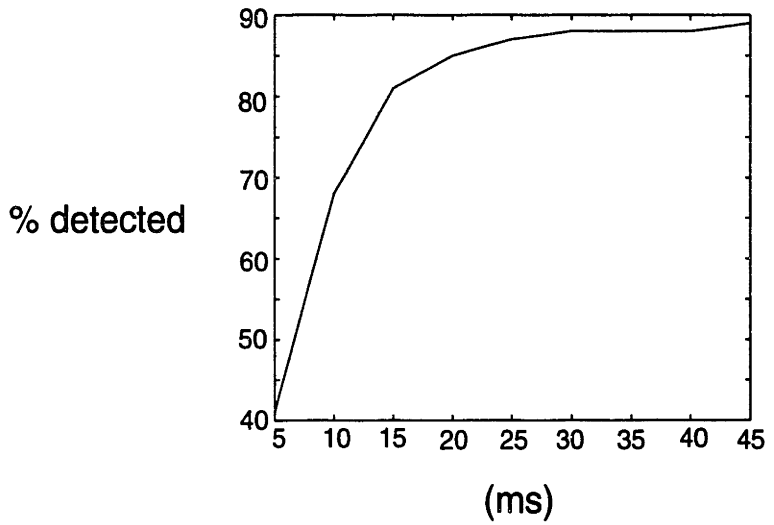

Figure 5.2: Temporal precision of the TIMIT algorithm. The percentage of detected landmarks is plotted against the time difference from the hand-labeled transcription. These data were compiled from the results of the TIMIT development set using the TIMIT algorithm. (fig:tresult2, 4x2.75).

from 6 dB to 9 dB. This threshold rise caused more nasals to be appropriately detected as **s** rather than **g** landmarks. The **s** insertions dropped because the **s** detector was fine-tuned to the new spectral characteristics of the TIMIT utterances, as described in Section 5.2.

The **b** deletion and substitution rates decreased because of the more sensitive method of determining silence. However, this sensitivity had a cost of increasing the **b** insertion rate.

Figure 5.2 shows the temporal precision of the TIMIT algorithm. 41% of the landmarks were detected within 5 ms of the hand-labeled transcription; 68% were within 10 ms; 85% were within 20 ms; and 88% were within 30 ms. A small percentage due to voiced obstruent closures lay beond 30 ms. The rest (10%) were either substituted or deleted. These values are approximately the same as those shown

Landmark type	*# Tokens*	*Del.*	*Subs.*	*Ins.*	*Neut.*	E_1	E_2
g(lottis)	798	2%	2%	3%	6%	7%	5%
s(onorant)	218	23%	17%	26%	12%	66%	49%
b(urst)	251	3%	3%	12%	11%	18%	15%
Total	1267	5%	5%	9%	8%	19%	14%

Table 5.3: Results of the TIMIT test set using the TIMIT algorithm.

in Figure 3.7 for the LAFF algorithm.

Table 5.3 shows the results of running the TIMIT algorithm on the TIMIT test data. The performance on the test data was strikingly similar to the performance on the development data. The similarity extended to the individual landmark types as well as the total. With a development set of just 16 speakers speaking a total of 80 utterances, the TIMIT algorithm was able to generalize to new speakers and sentences with no degradation in performance. This generalization was achieved with a few knowledge-directed modifications to the original LAFF algorithm. At an even more detailed level than landmark type, Table 5.4 shows the detection rates by phonetic category. The similarity in performance of the two data sets is manifested here as well, provided the number of tokens is large enough to make a good estimate of the detection rate. For the most part, landmark detection is a context-independent procedure, since it involves the identification of articulator-free features which are rather context-independent. The match between the development and test data, then, reflects this context-independency. In contrast, statistical algorithms, which rely on training, produce test results that are invariably inferior to training results (e.g. [Martens and Depuydt, 1991]). Part of the reason is that most statistical systems process on a frame-by-frame basis, so segment-level information like segment duration and the pairing of landmarks are not employed.

5.3.3 Complete test set

In the above experiment, the test set was a small subset of the complete test set provided with the TIMIT corpus. The test set used above was 48 utterances (16 speakers, 3 sx sentences per speaker). The justification for using such a small test set is the assumption that the test set is representative of all speakers and sentences. To examine the validity of this assumption, a larger test set was used. This larger test set is the complete TIMIT test set suggested by the National Institute of Standards and

	Phonetic category	*Landmark type*	*Detection rate (Development)*		*Detection rate (Test)*	
Outer **AC**	+v fric clos	**g**(lottis)	96%	(101)	95%	(58)
	+v fric rel	"	94%	(108)	93%	(60)
	−v fric clos	"	98%	(121)	100%	(78)
	−v fric rel	"	100%	(124)	100%	(76)
	flap clos	"	*88%	(24)	*65%	(20)
	flap rel	"	*88%	(24)	*65%	(20)
	+v stop clos	"	99%	(121)	100%	(73)
	+v stop rel	"	99%	(105)	100%	(83)
	−v stop clos	"	99%	(150)	99%	(86)
	−v stop rel	**b**(urst)	90%	(202)	92%	(109)
	nasal clos	**s**(onorant)	57%	(175)	71%	(115)
	nasal rel	"	56%	(113)	56%	(71)
	[l] clos	"	*53%	(19)	*44%	(9)
	[l] rel	"	63%	(30)	*30%	(23)
Intracons-onantal **AC**	stop clos	**b**(urst)	*93%	(28)	*89%	(9)
	stop rel	"	97%	(30)	*75%	(12)
	fric clos	"	95%	(115)	97%	(63)
	affric rel	"	91%	(117)	95%	(58)
	nasal → fric	**g**(lottis)	*96%	(28)	*88%	(16)
	fric → nasal	"	*100%	(13)	*100%	(11)
Intracons-onantal **A**	velophar clos	**g**(lottis)	89%	(64)	93%	(43)
	velophar rel	"	*91%	(22)	*100%	(9)
Inter-vocalic **A**	[ʔ] clos	**g**(lottis)	95%	(42)	*95%	(19)
	[ʔ] rel	"	100%	(45)	*100%	(21)
	−v [h] onset	"	*100%	(7)	*80%	(5)
	−v [h] offset	"	99%	(216)	98%	(120)
Total			90%	(2144)	89%	(1267)

Table 5.4: Detection rates of the TIMIT development and test sets using the TIMIT algorithm, listed by phonetic categtory. An * next to a detection rate means that the number of tokens is less than 30 so the detection rate is unreliable. The number of tokens is given in parentheses.

Landmark type	*# Tokens*	*Del.*	*Subs.*	*Ins. + Neut.*
g(lottis)	1412	6%	3%	7%
s(onorant)	427	45%	13%	23%
b(urst)	534	18%	4%	56%
Total	2373	16%	5%	21%

Table 5.5: Results of the TIMIT development set using the TIMIT algorithm, without hand-correction.

Landmark type	*# Tokens*	*Del.*	*Subs.*	*Ins. + Neut.*
g(lottis)	25121	6%	3%	10%
s(onorant)	8034	40%	16%	23%
b(urst)	8404	19%	5%	72%
Total	41559	15%	6%	25%

Table 5.6: Results of the complete NIST-suggested TIMIT test set using the TIMIT algorithm, without hand-correction.

Technology (NIST). It consists of 1344 utterances (168 speakers across all 8 dialect regions, each speaking 5 sx and 3 si sentences). In the interest of time, the hand labels and results of this complete test set were not hand-checked. The mapping from the TIMIT phonetic labels to landmark labels was performed automatically, so some inappropriate labels were left in the landmark transcription. For example, since TIMIT labels do not distinguish between acoustically-abrupt and acoustically-nonabrupt /l/s, all /l/s were labeled with landmarks, regardless of abruptness. Other kinds of extraneous transcription labels are a /nð/ combination realized as a dentalized nasal, and glide-like nasals. As part of the automatic result evaluation, the false landmarks were not separated into insertions and neutrals, and the extra-lingual false **b** landmarks were not excluded from the insertion + neutral rate.

The complete test results were compared to the development results when no hand-checking was used in the label-mapping and the result-checking. These results are shown in Tables 5.5 and 5.6. Note that the results between the development and complete test sets are roughly similar. The only big difference is in the **b** insertion rate. This difference is very likely due to longer pre-speech and post-speech intervals in the recordings of the complete test set. The background noise during these intervals cause **b** insertions. There is also a slight difference in the **s** deletion and substitution rates. Otherwise, the results are very similar. In most respects, then, the test results in Tables 5.3 and 5.4 are representative of the complete test set.

5.3.4 Comparison to related work

Other researchers have tried to perform tasks similar to landmark detection for such purposes as phonetic recognition, automatic phone label alignment, and concatenative speech synthesis. These tasks are often referred to as "segmentation". Segmentation divides the speech waveform into unequal length, somewhat steady-state, abutting pieces, with each piece corresponding to a phonetic or sub-phonetic unit. Direct comparison of landmark detection with segmentation is not possible because the philosophy and goals of the two tasks are different. Nevertheless, the results of some segmentation work will be presented to set the results of landmark detection in context. The comparisons are made with the TIMIT test results (Table 5.3) instead of with LAFF results because the TIMIT database is commonly used by the speech community and has wide phonetic and dialectal coverage.

Some researchers have employed a single-level segmenter to generate a single hypothesis of what they considered to be the phonetic boundaries in a speech waveform. Their goal is to maximize the detection of phonetic boundaries while minimizing insertions. Because the sounds of speech have varying levels of abruptness, with consonantal segments being the most abrupt, semivowels being less abrupt, and vowels being essentially steady-state, a single-level segmenter using one acoustic measure typically cannot increase detections without increasing insertions as well. An increase in detection rate directly translates into an increase in insertion rate. [Glass and Zue, 1986] have demonstrated this tradeoff. They used a segmentation algorithm which produced a boundary whenever the feature vector generated from the broadband signal changed sufficiently between two frames. Regardless of signal representation (hair cell response, critical band representation, or linear predictive coding), the segmentation error rate hovered around 30%. The hair cell response, which is the output of an auditory model and which Glass and Zue favored, gave a deletion rate of 23%, an insertion rate of 6%, and thus a total error of 29%. Their task required finding semivowel and diphthong boundaries, as well as voiced stop releases which could have a very short VOT. In order to compare their results to the landmark detection test results, the following adjustments on the landmark detection results had to be made:

1. In Glass' experiment, all the boundaries including semivowel and diphthong

transitions, were considered. Since the landmark experiment considered only acoustically-abrupt TIMIT labels, which comprised 75% of the complete set of labels, the remaining 25% were added to the number of tokens, making a total of 1690 tokens. The landmark error rates were then re-calculated with 1690 in the denominator.

2. The definition for "detection" was modified to include what was originally considered semivowel/diphthong insertions, creak insertions, and short stop burst neutrals.

3. Since Glass' experiment did not classify the boundaries, there was no substitution error. Thus, the substitution rate in the landmark experiment was counted in the new detection rate.

With these adjustments, the landmark error rate was 28% (deletion rate = 24%, insertion rate = 4%). However, this comparison is only a rough one because the landmark detector was not originally designed to find semivowels. If it had been, the deletion rate would be lower than 24%. Nevertheless, one can see that the landmark detector is at least as effective as the single-level segmenter of Glass and Zue when evaluated using their criteria. As another example, [Aubert, 1989] used an HMM to generate phonetic boundaries. The HMM hypothesized broad phonetic classes at each frame of the speech waveform, and every time the broad phonetic class changed, a phonetic boundary was generated there. The error rate was relatively low, at 12% (deletion rate = 9%, insertion rate = 3%), but the experiment was speaker-dependent.[3]

In yet another example, [Martens and Depuydt, 1991] used a multi-layer perceptron (MLP) to also categorize frames of the speech waveform into broad phonetic classes. Again, phonetic boundaries were placed whenever the broad phonetic class changed from one frame to the next. Their segmentation error rate was 23% (deletion rate = 11.4%, insertion rate = 11.5%), which is lower than the landmark detection error rate, but the vocabulary was restricted to the 10 Dutch digits.

In an attempt to overcome the conflicting goals of simultaneously reducing the

[3]The substitution rate presented in [Aubert, 1989] represents the rate of detecting a phonetic unit but misclassifying it during broad phonetic classification. In terms of segmentation, however, the substitution rate is counted as part of the detection rate.

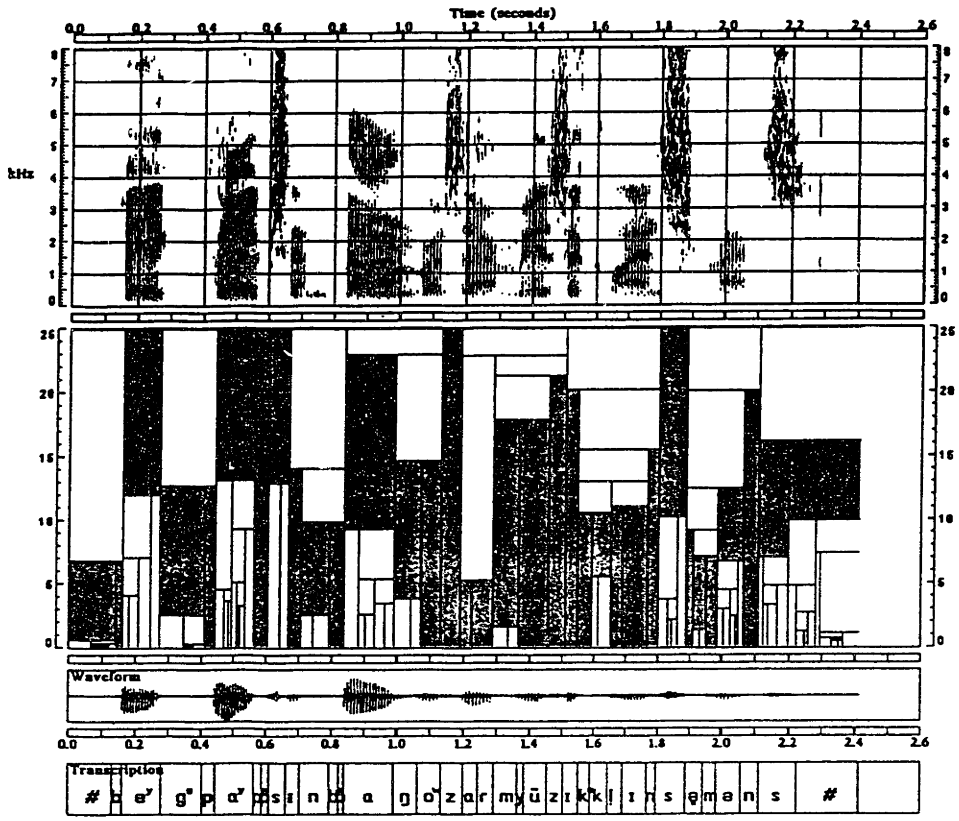

Figure 5.3: An illustration of multi-level segmentation of the utterance, "Bagpipes and bongos are musical instruments." The top panel is the spectrogram, the middle panel is the dendrogram, and the bottom panel is the hand-labeled phonetic transription. The best segmentation path is shaded in the dendrogram. The vertical dimension in the dendrogram corresponds to the Euclidean distance in feature space between two regions when they are merged. [Glass, 1988].

deletion and insertion rates met in single-level segmentation, some researchers have used multi-level segmentation, or segment networks. Multi-level segmentation examines the speech waveform at graded levels of resolution in order to capture the abrupt as well as nonabrupt phonetic boundaries. It generates boundaries at each level of resolution, resulting in many segmentation hypotheses. A unique segmentation is not given at the segmentation stage, but is determined only during the recognition stage. In order to score a multi-level segmenter, a hand-labeled phonetic transcription is aligned next to the multi-level segmentation, and the path which minimizes the error between the multi-level segmentation and the hand-labeled set is chosen. Based on this best path, which can include regions at any of the resolution levels, the deletion and insertion rates are calculated. In order to apply a multi-level segmenter to, say, phonetic recognition, post-processing needs to be performed to reduce the many segmentation hypotheses down to one hypothesis.

A notable algorithm for multi-level segmentation is the *dendrogram*, proposed by [Glass, 1988] and partly documented in [Glass and Zue, 1988]. The dendrogram is an example of an algorithm that requires no training and no parameter settings. Figure 5.3 shows a dendrogram with the shaded blocks being the best path through the multi-level segmentation. In Glass' work, the dendrogram was seeded at a fine temporal level and an agglomerative clustering technique produced segmentations of decreasing temporal resolution. The decision to merge neighboring regions was made based on the acoustic similarity of the broadband signal in the regions. Clustering ended when, at some level, one cluster spanned the entire utterance. One path was then chosen as the best path during a dynamic programming phase of minimizing the error between a hand-labeled phonetic transcription and the many layers of the dendrogram. The dendrogram resulted in a relatively low error rate of 9% (deletion rate = 4%, insertion rate = 5%). Because of the many segmentation hypotheses considered, this error rate can be viewed as a lower bound on the error rate of segmentation algorithms which rely on spectral variation as a function of time.

[Chigier and Brennan, 1988] have applied Glass' dendrogram to the practical problem of phonetic recognition. Out of computational considerations, they limited the average *depth* of the dendrogram to 1.8. They defined depth to mean the total number of dendrogram regions divided by the number of hand-labeled phonetic units. Again, the best path through the dendrogram was found by comparing to a

hand-labeled transcription. They achieved an error rate of 18% (deletion rate = 7%, insertion rate = 11%). For a depth of 3 and a restricted vocabulary of 10 digits, [Martens and Depuydt, 1991] have shown that the error rate can be as low as 4% (deletion rate = 3%, insertion rate = 1%). If the dendrogram were to be scored on the basis of a single-level segmenter, all the unused boundaries at all resolution levels, not just the best path, would have to be counted as insertions. The error rate would then be much higher than reported in each of these experiments and would increase with resolution depth.

5.4 Chapter summary

This chapter described landmark detection with the TIMIT database. The TIMIT database differs from the LAFF database in that TIMIT covers a wider range of dialects and phonetic contexts and it was recorded with a close-talking microphone having different spectral characteristics from the omnidirectional microphone used in LAFF. Knowledge-directed methods were used to adapt the LAFF algorithm to the TIMIT database. The adaptation produced a drop in error rate from 25% to 18% when substitutions were included, and from 18% to 14% when substitutions were not included. **G** and **b** detection was satisfactory. However, **s** detection was rather low because a large number of nasals, weak due to the mouth-directed recording microphone, were found by the **g** detector. With just 80 utterances spoken by 16 speakers in the development data, the TIMIT algorithm was able to generalize to new speakers and new sentences in the test data. The results of the development set and the unseen test set were strikingly similar. The small number of utterances in the development set to achieve equal performance with the test set is a trademark of knowledge-based engineering. An analysis of the temporal precision with which the landmark detector found landmarks showed that 41% were detected within 5 ms of the hand-labeled transcription, 68% were within 10 ms, 85% were within 20 ms, and 88% were within 30 ms. A small percentage due to voiced obstruent closures lay beyond 30 ms. The rest (10%) were either substituted or deleted. The temporal precision of the TIMIT algorithm is approximately the same as that for the LAFF algorithm. Although only a small test set was used, the results of the test set was shown to be representative across all dialects, speaking styles, and phonetic environments. A rough comparison

of landmark detection to a statistically-based segmentation algorithm showed that landmark detection is at least as effective as segmentation. Detailed comparison, however, was impossible because the philosophies and goals of landmark detection differ from those of segmentation. Integrating landmark detection and segmentation into a complete task (e.g. phonetic or work recognition) would be needed to make a fair comparison.

Chapter 6

NTIMIT: Telephone speech

As the telecommunications and computer industries merge, speech recognition applications increasingly involve telephone speech. Telephone speech presents a challenge because it is severely bandlimited compared to its broadband counterpart and can have a substantial amount of line noise added in. The transmission conditions, furthermore, are nonstationary, causing unpredictable effects to the speech signal. Some researchers have noted that task performance degrades between clean, broadband speech and telephone speech. In a phonetic recognition task, [Chigier and Leung, 1992] noted a consistent increase by a factor of 1.34 in phonetic classification error between broadband speech and telephone speech. In another study of phonetic classification, [Chang and Zue, 1994] also noted an increase in error, but their increase was by a factor of 1.7. Furthermore, they showed that, when training and test conditions did not match, performance degraded and this degradation was most pronounced when telephone speech was involved in either the training or the test data.

To illustrate how telephone speech affects landmark detection, experiments involving telephone speech were conducted. This chapter documents these experiments. First, a telephone speech database is presented. Then, a landmark detection algorithm specialized to telephone speech is developed. Finally, results of running the algorithm on the telephone speech are given and discussed.

6.1 NTIMIT database

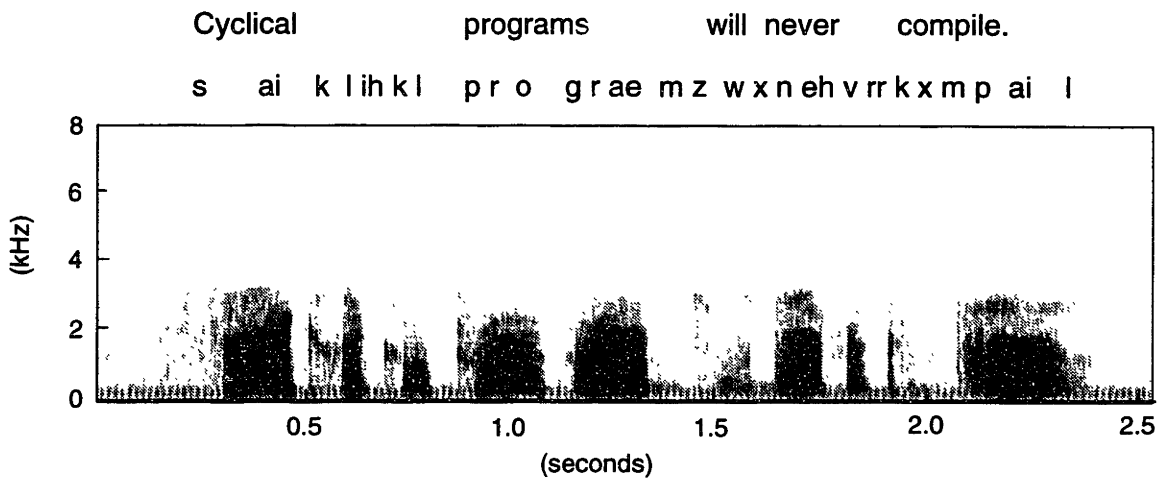

Figure 6.1: A spectrogram of an NTIMIT utterance. SNR = 19 dB for this utterance. The word transcription is given at the top.

The telephone speech came from NTIMIT, which stands for "network" TIMIT and is a telephone version of TIMIT. The NTIMIT utterances used in this experiment corresponded to the same utterances used in the TIMIT experiment. The same division of development and test data sets was kept. The landmark labels for NTIMIT were the same as those used for TIMIT. Details of how NTIMIT was created from TIMIT can be found in [Jankowski et al., 1990]. In brief, TIMIT utterances were transmitted through the NYNEX telephone network throughout New England, and the output collected at the end of the telephone line. Figure 6.1 shows a spectrogram of an NTIMIT utterance, which can be compared to its corresponding broadband TIMIT version shown in Figure 5.1. The output differs from the input in two important ways.

The first is bandlimitation: the resulting telephone speech is bandlimited to the range 300 Hz to 3.5 kHz. The bandwidth reduction can be seen in Figure 6.1: energy below 300 Hz and above 3.5 kHz is insubstantial. This bandwidth is about half of TIMIT's 8 kHz range. The 300 Hz cutoff presents problems for voicing and nasal detection. Voicing information can no longer be deduced from the 0–300 Hz range. Nasals, already weak from the close-talking microphone directivity, are even weaker with the low frequency cutoff. In Figure 6.1, the nasals at 1.4 sec, 1.6 sec, and 2.0 sec are so weak that they are almost buried under the background noise floor. Much of the time, the nasal murmur is absent altogether from the spectrogram. At

the other end of the spectrum, the 3.5 kHz high-frequency cutoff limits fricative noise information. This limitation is especially debilitating in the case of strident alveolar fricatives like [s,z], which have most of their frication noise above 3.5 kHz. Whereas [s,z] are clearly evident in the broadband case, they are completely cut off in the bandlimited case. In Figure 6.1, the maximum spectral prominence of the [s] at 0.2 sec and the [z] just before 1.5 sec are cut off, leaving only low-frequency, weak frication noise that is almost buried under the background noise floor.

The second effect of the telephone network is a reduced SNR. One contribution to the reduced SNR is telephone line noise, which can be substantial depending on the network path taken. The network paths in NTIMIT cover the entire region of New England. Another contribution to the reduced SNR is signal attenuation incurred during transmission. Of the NTIMIT utterances used, the average SNR is 29 dB, but the individual SNRs span a wide range from 14 dB to 44 dB. The utterance in Figure 6.1 has an SNR of 19 dB. In comparison, TIMIT's SNRs average 42 dB and have less variability. Already weak cues like weak fricatives and some stop bursts suffer the most from the reduced SNR. Their acoustic cues sometimes falls below the noise floor so that they don't even appear on the spectrogram. Comparing Figures 5.1 and 6.1, one can see that most of the stop bursts are noticeably weaker in NTIMIT than in TIMIT (e.g. the [k] at 0.7 sec).

6.2 NTIMIT algorithm

The NTIMIT algorithm is adapted from the TIMIT algorithm (see Section 5.2). Because of the differences between the TIMIT and NTIMIT utterances, the TIMIT algorithm has to be modified for the NTIMIT utterances. Specifically, the **g** and **b** landmark detectors are modified. An attempt was made to improve the **s** landmark detector as well, but was unsuccessful. This section describes the modifications, both successful and unsuccessful, to these landmark type-specific processors.

6.2.1 G(lottis) detector

Because of the reduced SNR in telephone speech, a structural change is made to the **g** detector. In the TIMIT algorithm, the first step was to find the voiced regions using zero-crossing rate (ZCR) and Band 1 energy information. Band 1 ROR peaks

were then added to the voicing boundaries to complete the set of **g** landmarks before pairing was imposed. In NTIMIT, this procedure is no longer viable because of the reduced SNR. The added noise makes the ZCR and Band 1 energy produce too many spurious landmarks. Therefore, for the NTIMIT algorithm, Band 1 ROR peaks are used alone to indicate **g** landmarks. Pairing proceeds as before. The **g** detector thus reverts back to LAFF's **g** detector.

Because of the 300 Hz low-frequency cutoff in telephone speech, a few parameter changes are made. Band 1's upper frequency limit is raised from 400 Hz to 800 Hz to include more harmonics. The widening of Band 1 causes an increased difference between the maximum Band 1 energy level and the Band 1 energy level of a schwa. Therefore, in the syllable requirement stage of **g** landmark verification, the voicing level is reduced from -25 dB to -30 dB with respect to the maximum Band 1 energy level. This represents a 10 dB drop from the original level used in the LAFF algorithm. Having very weak low frequency energy, nasals in telephone speech cause large ROR peaks at closure and release. In order to lessen the tendency to detect them as **g** landmarks, the Band 1 ROR peak threshold is raised from 9 dB to 12 dB. This represents a 6 dB increase from the original threshold used in the LAFF algorithm.

An autocorrelation technique of finding **g** landmarks was also tried, but was unsuccessful. In this technique, a short-time autocorrelation of the signal was computed with a frame length of 20 ms every 1 ms.[1] A periodic frame would have a high autocorrelation value and a noisy frame would have a low autocorrelation value. A threshold placed on the autocorrelation function could then pick out the voiced regions and exclude the aperiodic portions. The motivation behind using this technique for telephone speech is that, even if low frequency energy is cut off, the higher harmonics should still give an indication of periodicity. The problem with the autocorrelation function, however, was that signal energy dominated over periodicity. For example, high energy regions, like vowels or strident fricatives, would have a higher autocorrelation level than nasals. Normalization by the energy in each frame did not help, as frames with low energy level could produce a high normalized autocorrelation value. Thus, the autocorrelation technique was not used.

[1] In order to avoid the characteristic energy tapering at the ends of an autocorrelation, the autocorrelation over two frames was calculated first and then the result windowed, rather than windowing the signal first and then calculating the autocorrelation function.

6.2.2 S(onorant) detector

Because the telephone network does not affect the mid-frequency bands in the 0.8–3.5 kHz range in any predictable way, the **s** detector stays the same. The only modification is that Bands 5–6 are excluded in all processing since they are above the high-frequency cutoff of 3.5 kHz. Excluding Band 5 prevents it from introducing spurious pivots which can lead to **s** insertions.

An unsuccessful attempt was made to sensitize the **s** detector to subtle changes in energy. When sonorant consonantal segments are in high or back vowel context, their landmarks are often missed. To sensitize the **s** landmark detector, frequency bands finer than the coarse Bands 2–4 were used. In particular, every DFT component was made to represent one band, each covering 31.25 Hz. ROR peaks from any of these bands would signal an energy change. Pivot detection and landmark verification proceeded the same as before. Using this method, the **s** landmark detector was able to find pivots in high and back vowel context; however, it also found many other spurious **s** landmarks, mainly due to the mid-frequency energy transitions near obstruents. Thus, this method was not used.

6.2.3 B(urst) detector

In the **b** detector, the opivots are found as before. However, the silence detector is modified in order to accommodate the variable background noise level. The technique makes choosing the silence reference dependent on the amount of background noise in a given utterance. In the TIMIT algorithm, because the SNR was consistently high at 42 dB, measuring silence using a threshold which was offset from the highest energy level of the voiced regions was a reliable technique. In the LAFF algorithm, because the SNR was consistently lower at about 30 dB, a reliable technique was to use the background noise level as a reference for silence. However, because of the variable amount of noise in NTIMIT, using just one or the other of these two approaches is not a reliable measure for silence. Instead, an automatic hybrid technique is designed. First, the background noise level is measured with respect to the highest voiced energy level in the utterance. If this background level is lower than -40 dB from the highest voiced level, then the signal is relatively clean and an offset of -37 dB from the maximum vowel level is used as a maximum threshold for silence. If the background

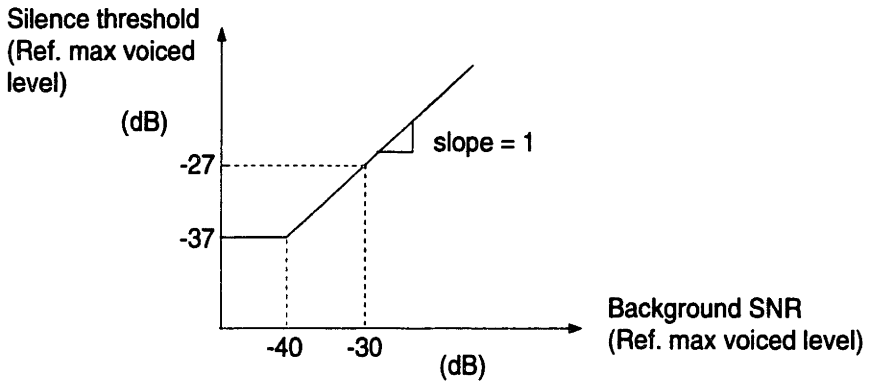

Figure 6.2: A graphical illustration of the hybrid method of choosing the silence threshold for the **b** detector in the NTIMIT algorithm.

level is higher than -40 dB, then the signal is relatively noisy and an offset of +3 dB from the background level is used as the maximum threshold for silence. Note that at exactly -40 dB from the maximum vowel level, either technique produces the same silence threshold (-37 dB), a desirable feature. The choice of a -40 dB breakpoint comes from TIMIT's silence measure. Figure 6.2 gives a graphical representation of the hybrid method for choosing the silence threshold.

6.3 NTIMIT results and discussion

Two landmark detection algorithms were applied to the NTIMIT database. First, the TIMIT algorithm was run on the telephone speech to give an indication of how to modify the landmark detection algorithm to suit the telephone speech. Second, the modified algorithm, which is the NTIMIT algorithm presented in the last section, was applied to the telephone speech. The results of both experiments are presented in this section.

6.3.1 Results with the TIMIT algorithm

Table 6.1 shows the results of running the TIMIT algorithm on the NTIMIT development data. Compared to the test results of running the TIMIT algorithm on the TIMIT database (see Table 5.3), performance degraded overall. The deletion, substitution, and insertion rates were all higher.[2] As had also been witnessed in Chapters 3,

[2]Because of the reduced amplitudes of the NTIMIT utterances, four utterances were not able to be processed using the TIMIT algorithm. The TIMIT utterance assumed a higher SNR than four

Landmark type	# *Tokens*	*Del.*	*Subs.*	*Ins.*	*Neut.*	E_1	E_2
g(lottis)	1250	3%	3%	20%	3%	26%	23%
s(onorant)	318	19%	41%	40%	7%	100%	59%
b(urst)	481	30%	21%	3%	13%	54%	33%
Total	2049	12%	13%	19%	6%	43%	31%

Table 6.1: Results of the NTIMIT development set using the TIMIT algorithm. (The number of tokens is less than in the corresponding TIMIT database because 4 utterances with exceptionally low SNR were not able to be processed with the TIMIT algorithm.)

4, and 5, **g** landmark detection was more robust than **s** and **b** detection.

The **g** deletion and substitution rates increased slightly because weak voiced regions representing schwas and nasals next to obstruents were significantly attenuated during telephone transmission. At the receiver, these voiced regions were so weak that some were not visible on the spectrogram. Thus, they were missed by the **g** detector. The **g** insertion rate increased by a large amount. The reason was that the NTIMIT average SNR was much lower than TIMIT's, causing low-frequency noise in NTIMIT to be misinterpreted as voiced regions.

The **s** deletion rate was actually 4% higher than in Table 5.3. The **s** substitution rate rose because the weak nasals in NTIMIT were captured by the **g** instead of the **s** detector. The **s** insertion rate increased because Band 5, which contained only background noise, caused spurious pivots to occur.

The **b** deletion rate increased significantly for two reasons. First, many attenuated bursts were too weak to cause an energy change in Bands 2–5, so they were missed altogether. Second, the TIMIT algorithm's definition of silence was too stringent for the telephone speech so that, even if a burst caused an energy change, the silence criterion may not have been met. The **b** substitution rate increased because the **b** landmarks were picked up as **g** landmarks. The **g** detector was very sensitive to small changes in energy, such as the low-frequency portion of bursts. The **b** insertion rate decreased for many of the same reasons that the **b** deletion rate increased.

6.3.2 Results with the NTIMIT algorithm

of the NTIMIT utterances had. The number of tokens is thus less than in the full development set.

Landmark type	*# Tokens*	*Del.*	*Subs.*	*Ins.*	*Neut.*	E_1	E_2
g(lottis)	1315	7%	4%	9%	1%	20%	16%
s(onorant)	337	19%	52%	22%	4%	93%	41%
b(urst)	492	24%	11%	9%	13%	44%	33%
Total	2144	13%	13%	11%	5%	37%	24%

Table 6.2: Results of the NTIMIT development set using the NTIMIT algorithm.

Table 6.2 shows the results of running the NTIMIT algorithm on the NTIMIT development data. Overall, the results do not differ much from Table 6.1, except that the insertion rate is reduced. The impact of the NTIMIT algorithm is more evident from examining the individual landmark types.

The **g** deletion rate actually increased, due to the de-sensitizing of the **g** detector to small variations in low-frequency energy. This de-sensitization was also the cause for a desired decrease in **g** insertions. Picking Band 1 ROR peaks, in general, is a less sensitive method of detecting **g** landmarks than using threshold crossings of the Band 1 energy. Raising the Band 1 ROR peak threshold for detection also contributed to the de-sensitization.

The **s** deletion rate remained the same while the substitution rate increased. The cause of rise in substitutions was the widening of the bandwidth of Band 1 from 0–400 Hz to 0–800 Hz. This widened bandwidth caused vowel-nasal transitions to have a more dramatic effect on Band 1 energy, since more harmonics are now involved. These Band 1 energy changes then caused **g** landmarks to occur at vowel-nasal transitions. Thus, the NTIMIT algorithm had more of a tendency to substitute **g** for **s** landmarks than the TIMIT algorithm. Although the widening of the Band 1 bandwidth seemed to have a negative impact on the performance of the **s** detector, it was a necessary modification because, otherwise, Band 1 would span only 100 Hz (300–400 Hz) and the Band 1 ROR signal would be noisy, causing many **g** insertions. The rest of the **s** landmarks were missed for the same reasons as for the TIMIT database: sonorant consonantal segments were often implemented in a glide-like manner so that there was not enough spectral abruptness. The **s** insertion rate was lower than in Table 6.1 because Band 5 was excluded from consideration during pivot detection.

The **b** deletion rate decreased because the **b** detector was able to adapt its silence threshold to the utterance-to-utterance variation in SNR. Nevertheless, deletion was still high because, as before, many of the bursts were so attenuated by the

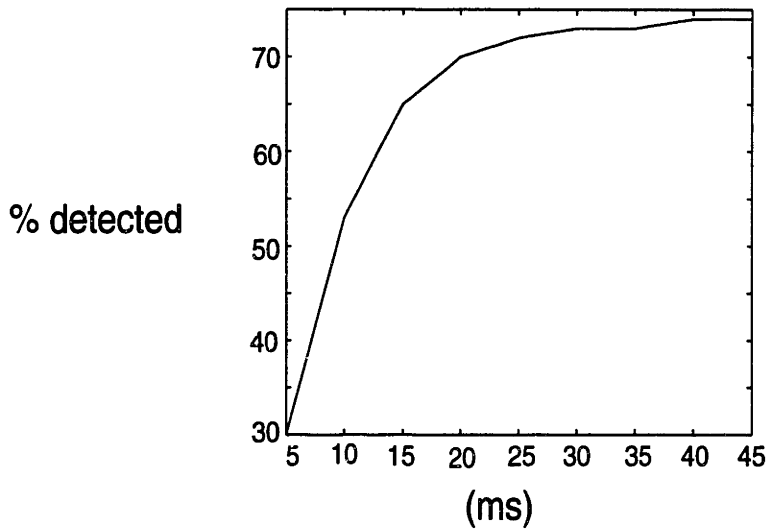

Figure 6.3: Temporal precision of the NTIMIT algorithm. The percentage of detected landmarks is plotted against the time difference from the hand-labeled transcription. These data were compiled from the results of the NTIMIT development set using the NTIMIT algorithm.

telephone line that they were buried under the background noise. The **b** substitution rate decreased because the **g** detector, responsible for most of the **b** substitutions before, is now de-sensitized so it responds less frequently to low-frequency burst energy. The **b** insertion rate increased because the silence detector is now finding silence intervals effectively. Since alveolar strident fricatives were often completely cut off by the 3.5 kHz limitation and silence replaced them, random noise below 3.5 kHz during these segments can now caused **b** insertions. Other causes of **b** insertions were spurious noise during stop closures and frication intervals, as was the case for the TIMIT database.

Figure 6.3 shows the temporal precision with which the NTIMIT algorithm finds landmarks. 30% of the landmarks were detected within 5 ms of the hand-labeled transcription; 53% were within 10 ms; 70% were within 20 ms; and 73% were within 30 ms. About 1% were detected beyond 30 ms. The rest (26%) were either substituted or deleted. These percentages are lower than for the TIMIT algorithm (Figure 5.2). The lower percentages reflect not a lower temporal precision of the NTIMIT algorithm compared to the TIMIT algorithm, but the lower detection rate for the NTIMIT database. Provided a landmark is detected, the NTIMIT algorithm's ability to detect landmarks with temporal precision remains about the same as the

Landmark type	*# Tokens*	*Del.*	*Subs.*	*Ins.*	*Neut.*	E_1	E_2
g(lottis)	798	8%	3%	7%	4%	18%	15%
s(onorant)	218	21%	52%	23%	5%	96%	44%
b(urst)	251	22%	12%	5%	17%	39%	27%
Total	1267	14%	13%	9%	7%	36%	23%

Table 6.3: Results of the NTIMIT test set using the NTIMIT algorithm.

TIMIT algorithm's ability, as demonstrated by the similar shapes of the curves in Figure 5.2 and Figure 6.3.

Table 6.3 shows the results of running the NTIMIT algorithm on the NTIMIT test data. As was the case between the TIMIT development and test data, the NTIMIT development and test results are almost identical. In fact, the error rates in Table 6.3 are 1% lower than in Table 6.2. By phonetic category, Table 6.4 shows that the detection rates between the development and test sets are also similar, provided the number of token is large enough to provide a good estimate. Again, a knowledge-based algorithm was able to generalize to new accents and new phonetic environments from just a small sample of speaking styles and sentences.

6.4 Chapter summary

In this chapter, landmark detection on telephone speech was described. The NTIMIT database was used. Two notable differences between telephone speech and its broadband counterpart are: (1) bandlimitation to 0.3–3.5 kHz and (2) reduced SNR. These effects have the greatest impact on reduced vowels, nasals, bursts, and alveolar strident fricatives. Using knowledge-based methods, a landmark detection algorithm was customized to telephone speech. As was witnessed in Chapter 5, knowledge-based algorithms can perform equally well for development and test data. Landmarks in telephone speech are harder to detect than in clean, broadband speech. The error rate went up from 19% to 35% if substitutions were counted, and 14% to 22% of substitutions were not counted. This error increase is expected given the reduction in information between the original speech and the telephone speech. An analysis of the temporal precision with which the NTIMIT algorithm detected landmarks showed that the NTIMIT algorithm is just about as accurate as the TIMIT algorithm at placing a landmark in time, given that it has detected the landmark.

	Phonetic category	*Landmark type*	*Detection rate (Development)*		*Detection rate (Test)*	
Outer **AC**	+v fric clos	**g**(lottis)	92%	(101)	90%	(58)
	+v fric rel	"	84%	(108)	88%	(60)
	−v fric clos	"	93%	(121)	88%	(78)
	−v fric rel	"	90%	(124)	95%	(76)
	flap clos	"	*88%	(24)	*70%	(20)
	flap rel	"	*79%	(24)	*70%	(20)
	+v stop clos	"	97%	(121)	96%	(73)
	+v stop rel	"	95%	(105)	95%	(83)
	−v stop clos	"	95%	(150)	94%	(86)
	−v stop rel	**b**(urst)	71%	(202)	78%	(109)
	nasal clos	**s**(onorant)	27%	(175)	30%	(115)
	nasal rel	"	19%	(113)	23%	(71)
	[l] clos	"	*53%	(19)	*11%	(9)
	[l] rel	"	60%	(30)	*30%	(23)
Intracons-onantal **AC**	stop clos	**b**(urst)	*61%	(28)	*44%	(9)
	stop rel	"	63%	(30)	*50%	(12)
	fric clos	"	66%	(115)	67%	(63)
	affric rel	"	52%	(117)	50%	(58)
	nasal → fric	**g**(lottis)	*61%	(28)	*69%	(16)
	fric → nasal	"	*69%	(13)	*73%	(11)
Intracons-onantal **A**	velophar clos	**g**(lottis)	50%	(64)	44%	(43)
	velophar rel	"	*55%	(22)	*56%	(9)
Inter-vocalic **A**	[ʔ] clos	**g**(lottis)	81%	(42)	*95%	(19)
	[ʔ] rel	"	93%	(45)	*100%	(21)
	−v [h] onset	"	*71%	(7)	*80%	(5)
	−v [h] offset	"	96%	(216)	92%	(120)
Total			74%	(2144)	73%	(1267)

Table 6.4: Detection rates of the NTIMIT development and test sets using the NTIMIT algorithm, listed by phonetic categtory. An * next to a detection rate means that the number of tokens is less than 30 so the detection rate is unreliable. The number of tokens is given in parentheses.

Chapter 7

Summary and conclusions

7.1 Thesis summary

In this thesis, a landmark detection algorithm was presented. Landmark detection is the first step in a proposed knowledge-based speech recognition system based on identifying distinctive features. The landmark detection algorithm was designed to find the abrupt-consonantal and abrupt landmarks. It incorporated measures of spectral abruptness and acoustic-phonetic information in a knowledge-based, hierarchical fashion to detect landmarks. The algorithm classified these landmarks into three types: **g**(lottis), **s**(onorant), and **b**(urst).

Four sets of experiments were conducted. The error rates of these experiments are shown in the bar graphs of Figure 7.1. The first graph shows the E_1 error rate, which is sum of the deletion, substitution, and insertion rates. The second graph shows the E_2 error rate, which is the deletion plus insertion rates. The third graph shows just the deletion rates. Each bar is labeled with the database used in the experiment followed by the algorithm in parentheses. The experiments involving the noisy speech database are labeled with the SNR in dB. The error rates are displayed in three different ways to reflect the different levels of severity of the three types of errors: deletion, substitution, and insertion. Figure 7.1a assumes that all three errors are equally detrimental. Figure 7.1b assumes that only the deletions and insertions are deleterious. Figure 7.1c assumes that only the deletions are undesirable. Note that the noise experiments were done on the LAFF development set, so the error rates for the 30 dB SNR experiment using the LAFF algorithm are not necessarily equal to the error rates for the LAFF test set using the LAFF algorithm.

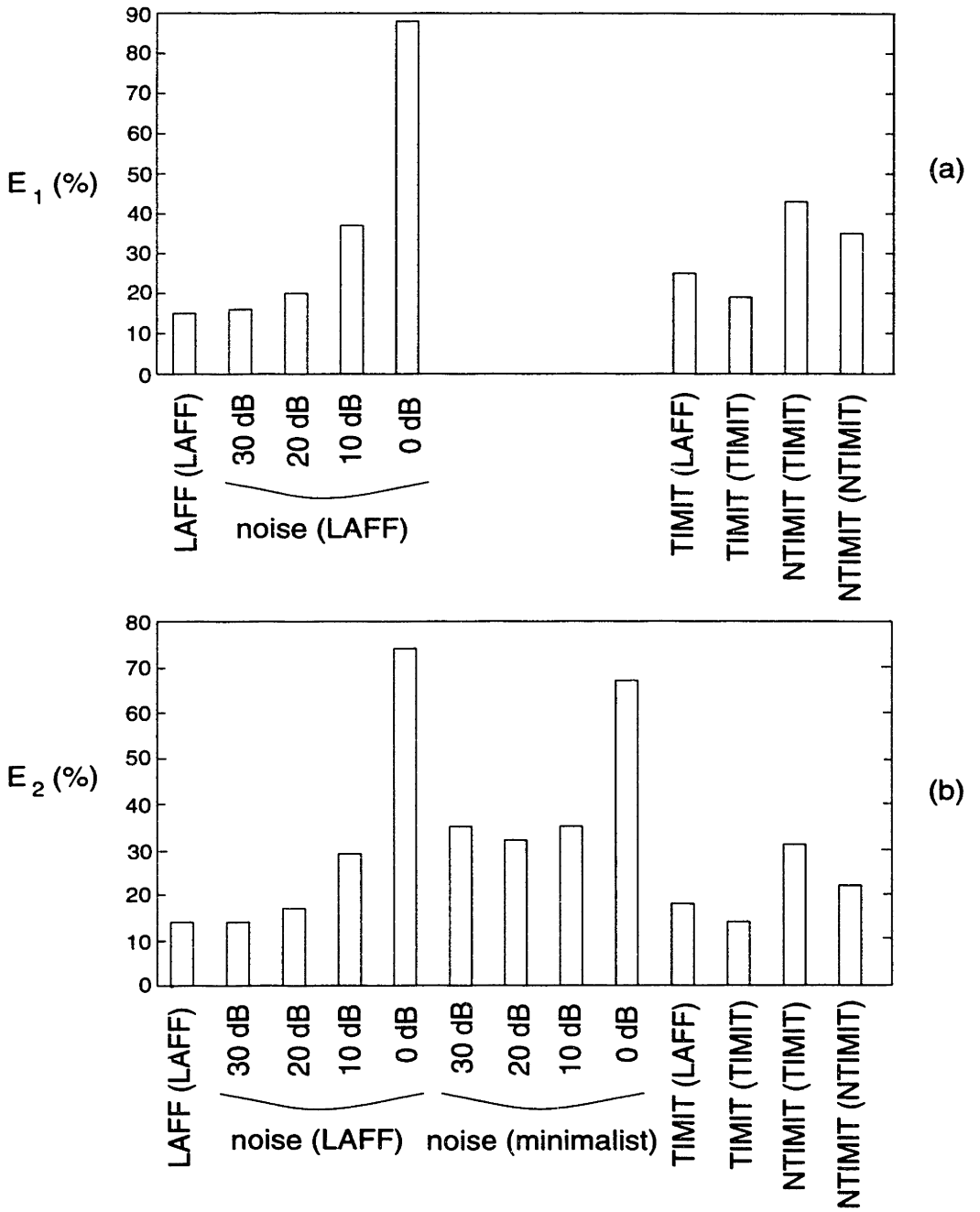

Figure 7.1: Error rates for the landmark detection experiments presented in this thesis. Each bar is labeled with the database used in the experiment and the algorithm in parentheses. The experiments involving the noisy speech database are labeled with the SNR in dB. (a) E_1 error rates are computed as the sum of the deletion, substitution, and insertion rates. (b) E_2 error rates are computed as the sum of the deletion and insertion rates. (Continued on next page).

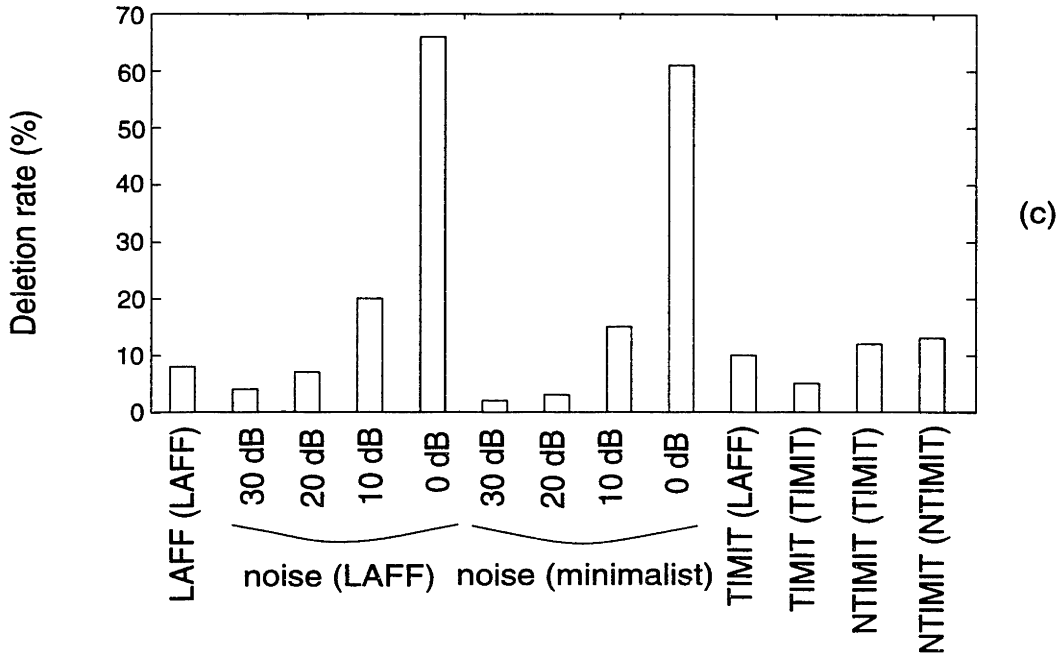

Figure 7.1: (Continued). (c) Deletion rates.

7.1.1 LAFF: clean speech

The first set of experiments involved the LAFF database, which is a controlled database of clean and clear speech of four speakers recorded with a broadband, omnidirectional microphone (see Chapter 3). Because of the favorable environment, the error rates were relatively low. Most of the deletions and insertions were due to **s** landmarks, which are sensitive to vowel context. The effect of the left and right reduced vowel environment on landmark detection was studied. Although the average detection rate for voiced obstruent landmarks was 95% or higher, those in right-reduced vowel context were missed more often than those in left-reduced vowel context. An acoustic analysis showed that voiced obstruents tended to have less of a low-frequency energy change, at closure and release, in right-reduced environment than in left-reduced environment. Furthermore, the duration of singleton consonants tended to be shorter in right-reduced environment than in left-reduced environment. These findings are in agreement with the American English flapping rule, which says that alveolar stops in right-reduced environment tends to be flapped more often than in left-reduced environment. The results of the prosodic analysis in this thesis suggests that the American English flapping rule could be extended to include a specification

on the reduction of all voiced obstruents.

7.1.2 Speech in noise

The second set of experiments involved speech in noise (see Chapter 4). A minimalist landmark detection algorithm was developed for operation in very noisy environments. This minimalist algorithm relied only on spectral abruptness to detect landmarks and used no higher level knowledge. Both the original LAFF algorithm and the minimalist algorithm were applied to the noisy speech. As the signal-to-noise ratio (SNR) decreased, the error rate went up, as expected. **G** landmarks were the most robust in noise, followed by **b** landmarks, and then by **s** landmarks. As shown in Figure 7.1c, the minimalist algorithm is able to detect more landmarks than the LAFF algorithm, but at the expense of not classifying landmarks into **g**, **s**, and **b** types and finding more insertions. The minimalist algorithm had a lower E_2 error rate at 0 dB SNR while the LAFF algorithm had the lower E_2 error rate at 10 dB, 20 dB, and 30 dB SNRs. This switch in performance suggests that, at a given SNR, an algorithm performs optimally when it searches for only those speech cues which are not masked by noise and that, if it tries to involve cues which are masked by noise, it performs sub-optimally. By design, the E_1 error rate is not defined for the minimalist algorithm because there are no substitutions.

7.1.3 TIMIT

The third set of experiments involved the TIMIT database (see Chapter 5). TIMIT is also a database of clean speech but it differs from LAFF in that TIMIT has wider phonetic and dialectal coverage and was recorded with a close-talking microphone having spectral characteristics different from an omnidirectional microphone. Knowledge-directed modifications were made to the original LAFF algorithm to customize it to the TIMIT database. These modifications decreased both the E_1 and E_2 error rates. This experiment demonstrates that knowledge-based methods can realize an improvement in performance when recording conditions are changed, provided that the correct parameter values are used in the algorithm. This improvement was realized using relatively few development utterances (80) and speakers (16).

7.1.4 NTIMIT: telephone speech

The fourth and final set of experiments involved the NTIMIT database (see Chapter 6). The NTIMIT is a telephone version of TIMIT. Increasingly, telephone speech is becoming a popular area of application. Telephone speech is challenging because it is severely bandlimited and has reduced SNR levels. Again, knowledge-based modifications were made to the TIMIT algorithm to create an NTIMIT algorithm suited to the NTIMIT database. Both E_1 and E_2 decreased as a result of the modifications. Again, only 80 utterances and 16 speakers were used in the development data to achieve the error reduction. As expected, error rates are higher than for the TIMIT database, since speech cues are compromised during telephone transmission.

7.1.5 Temporal precision

An analysis of the temporal precision of the landmark detection algorithm showed that most of the landmarks were detected within the vicinity of the hand-labeled transcription. For clean speech (LAFF and TIMIT), about 40% of the landmarks were detected within 5 ms of the hand-labeled transcription, 70% were within 10 ms, 85% were within 20 ms, and 90% were within 30 ms. Less than 1% were beyond 30 ms. The rest (about 10%) were either substituted or deleted.

7.2 Conclusions

7.2.1 The use of knowledge

One advantage of a knowledge-based approach is that improvements to the algorithm can be made in a directed manner. A deletion rate reduction does not have to be directly coupled with an insertion rate increase. This de-coupling of the deletion and insertion rates was illustrated in the TIMIT and NTIMIT experiments. In the TIMIT experiment, the TIMIT algorithm was constructed in a knowledge-directed manner based on the results of running the LAFF algorithm on the TIMIT database. As a result of the modified algorithm, the E_2 error rate dropped from 18% to 14%, as shown in Figure 7.1b. Similarly, the NTIMIT algorithm was designed based on the results of running the TIMIT algorithm on the NTIMIT database, and the E_2 error rate dropped from 31% to 24%.

A comparison to the related task of segmentation shows that landmark detection performs at least as well as traditional approaches to segmentation. In most previous work on segmentation, boundaries were found by detecting points of acoustic dissimilarity between two nearby frames. In general, little speech knowledge was employed, so a deletion rate reduction was directly coupled to an insertion rate increase.

Another advantage of a knowledge-based approach is that the development data set, traditionally termed the "training" set, need not be very large, and generalization to unseen data can still be achieved. This phenomenon was illustrated by the TIMIT and NTIMIT experiments. In both experiments, the development set consisted of 16 speakers speaking a total of 80 utterances. This set size was large enough to achieve no difference between the development and test results. In contrast, largely statistically-based systems that rely heavily on training to acquire the requisite "knowledge" invariably have a degradation in performance between the training and test sets.

7.2.2 Errors in landmark detection

Because landmark detection is a critical first step in the lexical access system, some way must be devised to take care of the errors in landmark detection. Of the errors, the insertions can be removed when further processing for distinctive features in the vicinity of the insertion determines that the insertion is not a valid landmark. Many of these insertions are due to semivowel and diphthong transitions; therefore, they are a clue to the placement of nonabrupt and vocalic landmarks. In the LAFF experiment, for example, the test set had an insertion rate of 6%; 3% was due to semivowel and diphthong transitions, 1% was due to the nonsimultaneous energy transitions between high and low frequencies near obstruents, 0.5% was due to nonspeech noise during stop closures, and 1.5% was due to other miscellaneous effects.

In a sense, the substitutions are not real errors, since they point to a desired landmark but with the wrong type assignment. Again, further processing can determine the true type of a substitution. In the LAFF experiment, for example, the substitution rate was a low 1%, mainly **g** landmarks substituted by **s** landmarks at heavily voiced obstruents, and **b** landmarks substituted by **g** landmarks when the low frequency energy in the burst was strong. In the other clean speech experiment, using

TIMIT, the test set gave a higher substitution rate (5%) because of the close-talking microphone characteristics.

The deletions are the most serious kinds of error. The deletion rates are given in Figure 7.1c. For clean, broadband speech, the deletion rate is under 10%. In the LAFF experiment, for example, the deletion rate was 8%. Part of the deletion rate is due to mislabeling. Mislabeling may account for up to 2% in the deletion rate. For example, the [n] closure in "want" may be coincident with the velopharyngeal port closure. If these two events are labeled with separate landmarks, however, then a deletion is likely to result. Even though no −**s** landmark is detected at the [n] closure, the presence of nasality in the preceding vowel would indicate an underlying nasal segment. Also segments which are phonologically [+consonantal] but realized nonabruptly may be mislabeled with abrupt-consonantal landmarks. This type of mislabeling may occur most often for nasals and /l/s, which may be implemented in a glide-like manner. To some extent, the neutral category of landmarks reduces errors caused by mislabeling by not counting certain superfluous landmarks found by the landmark detector as errors when decisions about landmark labeling is ambiguous. More careful labeling by a phonetician looking at multiple cues may help reduce the number of deletions due to mislabeling.

Another kind of deletion comes from not finding heavily voiced obstruents, flaps, nasals, and [l]s. In the LAFF experiment, 1% in the deletion rate was from missing heavily voiced obstruents and flaps, and 2% was from missing nasals and [l]s. The landmarks of these segments are hard to find because energy abruptness in the expected frequency bands may be compromised by a voice bar, the vowel context, or the speed of closure and release. These deletions can potentially be captured by a nonabrupt landmark detector, which puts a landmark near the energy minimum of a constriction for semivowels. Further analysis in the area, to find nasality based on pole-zero pairs for example, may then be able to identify the underlying segment.

Bursts and the cessation of speech noise due to a following stop closure accounted for some deletions. In the LAFF experiment, 1% out of the 8% deletion rate was from not finding sufficient silence during closure. This error can be taken care of by using a broadband measure for silence rather than using individual bands, as was done in the TIMIT algorithm. Another 1% in the deletion rate was due to not finding sufficient energy abruptness at the bursts or closures. A better SNR, attained

possibly through noise-cancelling pre-processing, could remove these deletions.

More knowledge than is currently used may be added to improve performance. For example, if the duration between two detected landmarks is too long, suggesting that a landmark was missed in between, then the ROR peak threshold could be lowered to detect more subtle spectral changes in that region. Prosodic information could also help. As mentioned in Section 3.3, obstruent landmarks following an unreduced vowel and preceding a reduced vowel are more likely to be missed than landmarks in other reduced vowel environments. Providing the landmark detector with the reduced vowel environment would allow tailoring of parameter values to suit the prosodic environment. For example, the ROR peak threshold could be reduced in right-reduced environment. In utterance-final syllables, the landmark detector misses some landmarks because it is not tuned to work in an environment with lowered fundamental frequency, decreased waveform intensity, and a lengthened syllable. If the utterance endpoint were provided, then parameter values could be modified for the end of the utterance. More generally, prosodic phrase boundaries in running discourse could aid the landmark detector. Examples of how the algorithm could be modified are, in the syllable before a prosodic phrase boundary, the smoothing interval could be lengthened from 20 ms to 30 ms, the ROR peak threshold could be reduced 3 dB, and the minimum vowel requirement could be relaxed to allow a lower intensity signal.

7.2.3 Adaptation to environment and speaker

In this thesis, the adaptation of the landmark detection algorithm to a new operating environment (i.e. LAFF to noisy speech, LAFF to TIMIT, TIMIT to NTIMIT) was done manually. Eventually, a fully-independent system would have to have an environment sensor at the beginning to decide on the operating environment conditions, and then the parameters of the landmark detector could be automatically adapted to the operating environment. Such a sensor would estimate the background noise level, speaking level, microphone frequency response, and possibly microphone directionality.

Just as human listeners go through an adaptation phase when listening to someone new, the landmark detector could improve performance by adapting to a

particular speaker and speaking style. An estimate of the speaker's vocal tract length and glottal characteristics could be obtained from a sample utterance before recognition begins. Then frequency ranges of the bands could be adjusted to suit a particular speaker's formant ranges. A pre-emphasis factor could be used to compensate for breathy voicing. The smoothing length could be adjusted to fit the speaker's fundamental frequency range. Speaking rate information could be used to set the range for how closely or far apart two landmarks can occur with respect to one another.

7.2.4 Articulator-free features

As mentioned in Chapter 1, in the process of landmark detection, inferences about some of the articulator-free features at a landmark can be made. The articulator-free features involved were shown in Figure 1.5. Researchers employing a segmentation approach to speech analysis have shown how segmentation and broad phonetic classification can be intertwined [Martens and Depuydt, 1991], [Eide, 1993]. Similarly, in the process of landmark detection, constraints on the articulator-free features, and thus broad phonetic class, arise. An **s** landmark carries with it the features [+consonantal] and [+sonorant]. A **b** landmark carries with it the features [+consonantal], [–sonorant], and [–continuant]. A **g** landmark is ambiguous, since it could be [+consonantal] or [–consonantal]. If a stop closure or release is causing the **g** landmark, for example, then the landmark would carry the features [+consonantal], [–sonorant], and [–continuant]; however, if an [h] or glottal stop is causing the landmark, then it would carry the feature [–consonantal]. Further analysis to detect any formant movement around the **g** landmark would resolve this ambiguity.

As was described in Chapter 2, outer **AC** landmarks usually come in closure-release pairs, except at the beginning and end of an utterance. Once the value of the feature [consonantal] is ascertained, the outer **AC** landmarks can be identified and paired to each other. This information will aid in identifying singleton consonants and consonant clusters.

7.2.5 Nonabrupt and vocalic landmarks

The landmark detector described here finds abrupt and abrupt-consonantal landmarks. Algorithms to detect the nonabrupt landmarks, for semivowels, and the

vocalic landmarks, for vowels, still need to be developed. The landmark detector presented in this thesis can help find the remaining landmarks. For example, the **g** landmarks already delimit the voiced regions within which the nonabrupt and vocalic landmarks have to occur. Pairs of pivots found by the **s** detector could be surrounding a nonabrupt landmark, especially if these pivots fail the steady-state and abruptness criteria.

7.2.6 Lexical access

Once the articulator-free features are completely identified around each landmark, further processing can be customized to the broad phonetic environment to identify the articulator-bound features at each landmark. Following feature extraction, the landmarks with their feature bundles need to be collapsed together in some cases and expanded in other cases so that a one-to-one relation exists between a segment and a bundle of features. An example in which two feature bundles at two landmarks would have to be collapsed into one feature bundle to represent one segment is the closure and release landmarks of an intervocalic [b]. An example in which one feature bundle at one landmark would have to be expanded into two bundles is the [br] release in "bright", in order to account for the two segments [b] and [r]. Procedures would have to be introduced to account for the different ways one could pronounce a word. One approach would be to construct a pronunciation network for each word having multiple pronunciations. This pronunciation network would be described at the distinctive feature level, since one of the advantages of distinctive features is their ability to describe modifications concisely. Another way to describe modification is by identifying all modifiable features in the lexicon, and then during lexical access, those features are optional. Modification can be due to speaking style ("either" as [i ð ɚ] or [a y ð ɚ]), casual speech ("in the" as [ih n x] where the [n] is dentalized), or word boundary assimilation ("gas shortage" as [g ae sh ao r t x dj]), among other effects. Once the pronunciation network is in place or the modifiable features are marked, lexical access using a sequence of feature bundles is a straightforward process.

Appendix A

Sentences in databases

A.1 LAFF and noise databases

A.1.1 Development set

1. Which year were you lazy?
2. Few can follow the zoo.
3. I write a lasso and then they debate.
4. Take caution with the small baby.
5. Below the city is a cookie.
6. A sudden shake can leave nothing other than an ache.
7. Catch an ape with her shoe again.
8. A big potato is something to deny.
9. Keep the water away from the book.
10. She can sing.
11. Catch a balloon before Mary does.
12. He saw the woman in the water.
13. Today we will keep something from Arlene.
14. The canoe is easy to support.
15. I have an even number of them.
16. It began when Jake said he was able to pay the money.
17. Tom and Arlene appear to support the lazy pig.
18. We get into something every day.
19. Follow that cab before you leave.
20. A few busy women began with no money.

A.1.2 Test set

81. Any other day would be easy.
82. Mary likes to suppose she can leave.
83. The money is coming today.
84. Did you go to the zoo?
85. Can't you take a bus?
86. What would you do without Arlene?
87. Could you take a sudden loss?
88. The bat you gave him is lazy.
89. I will add up the number of leaky pails.
90. What you do is up to you.
91. I can't shake this toe.
92. A small canoe is coming tonight.
93. Never appear for a day at a time.
94. The busses show a loss for the day.
95. Did you look at Jake or at Tom?
96. Did she blow up the balloon?
97. Too much caution can put you in a bad state.
98. Would you like some cookies?
99. Debby wants to seal up the cookies.
100. My cousin can never remember what a nasal is.

A.2 TIMIT and NTIMIT databases

A.2.1 Development set

1. Bright sunshine shimmers on the ocean.
2. Carl lives in a lively home.
3. Although always alone, we survive.
4. Critical equipment needs proper maintenance.
5. Biblical scholars argue history.
6. Academic aptitude guarantees your diploma.
7. The prowler wore a ski mask for disguise.

8. Chocolate and roses never fail as a romantic gift.
9. Challenge each general's intelligence.
10. Upgrade your status to reflect your wealth.
11. Cliff's display was misplaced on the screen.
12. Doctors prescribe drugs too freely.
13. It's impossible to deal with bureaucracy.
14. Primitive tribes have an upbeat attitude.
15. Flying standby can be practical if you want to save money.
16. The Thinker is a famous sculpture.
17. Masquerade parties tax one's imagination.
18. Birthday parties have cupcakes and ice cream.
19. The cartoon features a muskrat and a tadpole.
20. The emperor had a mean temper.
21. Publicity and notoriety go hand in hand.
22. Cyclical programs will never compile.
23. Correct execution of my instructions is crucial.
24. The previous speaker presented ambiguous results.
25. The water contained too much chlorine and stung his eyes.
26. The drunkard is a social outcast.
27. They remained lifelong friends and companions.
28. Who took the kayak down the bayou?
29. The diagnosis was discouraging; however, he was not overly worried.
30. Does Creole cooking use curry?
31. She encouraged her children to make their own Halloween costumes.
32. Almost all colleges are now coeducational.
33. Would a tomboy often play outdoors?
34. Draw each graph on a new axis.
35. Even I occasionally get the Monday blues!
36. How permanent are their records?
37. You always come up with pathological examples.
38. Clear pronunciation is appreciated.

39. The courier was a dwarf.
40. The hallway opens into a huge chamber.
41. Does Hindu ideology honor cows?
42. Reading in poor light gives you eyestrain.
43. By eating yogurt, you may live longer.
44. The essay undeniably reflects our view ably.
45. Do you have the yellow ointment ready?
46. Draw every outer line first, then fill in the interior.
47. The jaw operates by using antagonistic muscles.
48. Are holiday aprons available to us?
49. John's brother repainted the garage door.
50. In the long run, it pays to buy quality clothing.
51. Should giraffes be kept in small zoos?
52. The meeting is now adjourned.
53. Those answers will be straightforward if you think them through carefully first.
54. Steve collects rare and novel coins.
55. Al received a joint appointment in the biology and the engineering departments.
56. The big dog loved to chew on the old rag doll.
57. Gus saw pine trees and redwoods on his walk through Sequoia National Forest.
58. The dark, murky lagoon wound around for miles.
59. The frightened child was gently subdued by his big brother.
60. I'll have a scoop of that exotic purple and turquoise sherbet.
61. The new suburbanites worked hard on refurbishing their older home.
62. The moisture in my eyes is from eyedrops, not from tears.
63. George seldom watches daytime movies.
64. I assume moisture will damage this ship's hull.
65. Will you please confirm government policy regarding waste removal?
66. She always jokes about too much garlic in his food.
67. Please shorten this skirt for Joyce.
68. Tim takes Sheila to see movies twice a week.
69. We'll serve rhubarb pie after Rachel's talk.

70. A huge power outage rarely occurs.
71. Valley Lodge yearly celebrates the first calf born.
72. Those who teach values first abolish cheating.
73. Movies never have enough villains.
74. A crab challenged me, but a quick stab vanquished him.
75. The overweight charmer could slip poison into anyone's tea.
76. Shell shock caused by shrapnel is sometimes cured through group therapy.
77. Don't look for group valuables in a bank vault.
78. First add milk to the shredded cheese.
79. Spherical gifts are difficult to wrap.
80. Roy ignored the spurious data points in drawing the graph.

A.2.2 Test set

1. Project development was proceeding too slowly.
2. Kindergarten children decorate their classrooms for all holidays.
3. Special task forces rescue hostages from kidnappers.
4. Call an ambulance for medical assistance.
5. He stole a dime from a beggar.
6. The emblem depicts the Acropolis all aglow.
7. The misquote was retracted with an apology.
8. Objects made of pewter are beautiful.
9. Artificial intelligence is for real.
10. December and January are nice months to spend in Miami.
11. Tradition requires parental approval for under-age marriage.
12. The clumsy customer spilled some expensive perfume.
13. The bungalow was pleasantly situated near the shore.
14. The sound of Jennifer's bugle scared the antelope.
15. Who authorized the unlimited expense account?
16. Destroy every file related to my audits.
17. Serve the coleslaw after I add the oil.
18. Withdraw all phony accusations at once.

19. Straw hats are out of fashion this year.
20. Why buy oil when you always use mine?
21. They enjoy it when I audition.
22. Military personnel are expected to obey government orders.
23. Michael colored the bedroom wall with crayons.
24. The morning dew on the spider web glistened in the sun.
25. The small boy put the worm on the hook.
26. How good is your endurance?
27. It's healthier to cook without sugar.
28. The viewpoint overlooked the ocean.
29. Are you looking for employment?
30. Highway and freeway mean the same thing.
31. To further his prestige, he occasionally reads the Wall Street Journal.
32. Alice's ability to work without supervision is noteworthy.
33. The oasis was a mirage.
34. Cory and Trish played tag with beach balls for hours.
35. The tooth fairy forgot to come when Roger's tooth fell out.
36. Planned parenthood organizations promote birth control.
37. Jeff thought you argued in favor of a centrifuge purchase.
38. If Carol comes tomorrow, have her arrange for a meeting at two.
39. Shaving cream is a popular item on Halloween.
40. Cheap stockings run the first time they're worn.
41. A chosen few will become Generals.
42. If people were more generous, there would be no need for welfare.
43. The cranberry bog gets very pretty in Autumn.
44. Please dig my potatoes up before frost.
45. A big goat idly ambled through the farmyard.
46. The fog prevented them from arriving on time.
47. Lori's costume needed black gloves to be completely elegant.
48. Bob papered over the living room murals.

Appendix B

Derivation of the 9 dB peak threshold

The 9 dB threshold is roughly the minimum amount of difference between the energy in a segment having a diminished glottal source amplitude and a neighboring vowel. This threshold is justified below. The glottal source amplitude, G, is dependent on the subglottal pressure, P_s, and the supraglottal pressure, P_m, [Stevens et al., 1992a]

$$G = K(P_s - P_m)^{3/2}. \tag{B.1}$$

K is a proportionality factor. During a vowel, $P_m = 0$ so the glottal source amplitude becomes

$$G_{vowel} = K(P_s)^{3/2}. \tag{B.2}$$

During the production of an obstruent, P_m becomes positive and the glottal source amplitude diminishes. In the case of a voiced fricative, in order for the glottis to continue vibrating, P_m must be no more than $P_s/2$. Substituting this maximum limit in for P_m in (B.1), one gets that the glottal source amplitude during a voiced fricative is

$$G_{vfric} = K\left(\frac{P_s}{2}\right)^{3/2}. \tag{B.3}$$

From (B.2) and (B.3), the ratio of glottal source amplitudes between the production of a vowel and a voiced fricative is

$$\frac{G_{vowel}}{G_{vfric}} = 2^{3/2}.$$

In dB, this is

$$20\log(2^{3/2}) = 9 \text{ dB}.$$

The voiced fricative case was chosen as a worst case. For other segments, the complete closure of a supraglottal articulator (e.g. stops, affricates) and/or a complete cessation of glottal vibration (e.g. voiceless segments, glottal stops) causes the change in glottal source amplitude to be even more dramatic than 9 dB. Beyond the glottis, changes in vocal tract shape also affect the energy measured at the microphone. At the release of a consonant, which involves the opening of a primary articulator, the first formant frequency, F_1, goes up, leading to an increase in F_1 amplitude [Stevens, 1995a]. Assuming that higher formant frequencies remain constant, the movement of F_1 upward causes an even larger increase in higher frequency amplitudes than the increase in F_1 amplitude. The effect of the vocal tract shape, then, augments the 9 dB change in Band 1 and usually in the higher bands as well. Beyond the vocal tract, radiation attenuates the acoustic signal by 1/r, where r is the distance from the mouth to the microphone; however, this 1/r factor is the same for all sounds and therefore does not affect the ratio between a vowel and any other segment. The difference in losses in the vocal tract (viscosity, yielding throat walls, thermal effects, throat radiation) between a vowel and an obstruent are ignored [Portnoff, 1973]. The 9 dB threshold is a lower bound for ROR peak-picking; the above list of factors show that, in general, ROR peaks associated with segments having a diminished glottal source amplitude in relation to a neighboring vowel are well above 9 dB.

For sonorant consonants, glottal source amplitude remains at the level of the neighboring vowel, so all the energy change occurs in Bands 2–6 and is due to vocal tract movements. The first resononce frequency moves down at the closure of a sonorant consonant, and pole-zero pairs are introduced. Most of the time, these two factors are enough to cause a 9 dB change in Bands 2–5 energy. However, some sonorant consonants are particularly prone to be missed by the 9 dB rule. For example, if the neighboring vowel of a nasal is a high vowel, then F_1 is already close to the target position of the nasal ($F_1 \cong 300$ Hz for nasals and [i, u] [Fujimura, 1962, Peterson and Barney, 1952]). In these cases, ROR peaks in Bands 2–5 may not surpass 9 dB.

Most of the ROR peaks are well above 9 dB, often in the 20 dB to 40 dB range. In practice, the 9 dB threshold works well. A lower threshold, 6 dB, was tried, and was found to raise the true detection rate for **g** landmarks by 3%. This improvement in detection rate, however, was accompanied by an increase of 2% in

the number of inserted **g** landmarks. The increased number of inserted **g** landmarks marred the detection of **s** and **b** landmarks, which depend on the **g** landmarks. The detection rates for **s** and **b** landmarks actually went down as a result of the lowered threshold: by 6% for **s** landmarks, and by 1% for **b** landmarks. A higher threshold, 12 dB, was also tried, but this threshold caused a 5% decrease in the detection rate for **g** landmarks. Although the insertion rate for **g** landmarks correspondingly decreased (by 3%), the degradation in the detection rate of **g** landmarks substantial decreased the detection rates of **s** and **b** landmarks as well: by 3% for **s** landmarks, and by 6% for **b** landmarks. As further empirical corroboration for the 9 dB threshold, [Stevens et al., 1992a], in their study, used a similar threshold (10 dB) between the glottal vibration amplitude of a fricative and a neighboring vowel as measured in the acoustic waveform.

Appendix C

Source code for LAFF algorithm

This appendix lists the C source code for the LAFF landmark detection algorithm and its supporting functions. Figure C.1 shows a flow diagram of the algorithm, with the inputs and outputs to various C programs. The initial input is the sampled data file (*.sd). Two spectrograms are computed from *.sd using the Entropic ESPS **sgram()** program [Entropic Speech, 1994]:

```
sgram -o 9 -E 0 -S 1 -w 6 *.sd *.sgram
sgram -o 9 -E 1 -S 1 -w 6 *.sd *.sgram1.
```

The parameters used to calculate the first spectrogram (*.sgram) are a 9th order DFT, no pre-emphasis, frames occurring every 1 ms, and a 6 ms long Hamming window. The parameters to the second spectrogram (*.sgram1) are the same except there is a pre-emphasis factor of 6 dB/octave. The pre-emphasized spectrogram is passed into the function **hpror()**, which generates the first derivative of a high-pass energy signal (*.hpror). Then, *.sgram and *.hpror are the inputs to the landmark detection algorithm, **lm()**. The output of **lm()** is an ASCII ESPS-compatible label

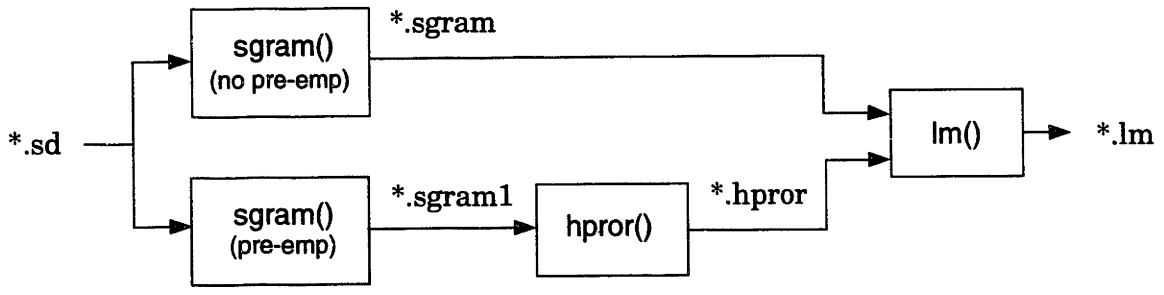

Figure C.1: A flow diagram of the landmark detection algorithm.

file specifying the times and types of the landmarks. This appendix lists the source code for **lm()** and **hpror()**.

C.1 LM.C: landmark detection algorithm

Lm.c is the main program, which calls supporting functions. This section lists the following files, in order:

include.h
define.h
global.h
lm.c
sgram_read.c
energy.c
energy_write.c
ror.c
ror_write.c
mpeak.c
mermel.c
localize.c
order_pk.c
copy_pk.c
pgroup.c
verify_g.c
copy_B1pk.c
vocalic.c
write_B1pk.c
tag.c
find_pivot.c
uni_sign.c
verify_s.c
compare_spec.c
abruptness.c
rules.c
burst.c
find_opivot.c
nopivotsinbetween.c
write_pk.c

```
/****************************** INCLUDE.H ******************************
   Included files.
*/

#include <stdio.h>
```

```
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include <esps/esps.h>
#include <esps/fea.h>
#include <esps/feaspec.h>
#include <esps/ftypes.h>

/****************************** DEFINE.H ******************************
   Parameter values used in lm.c .
*/

/* energy.c */

#define WAVELEN 5000  /* Max length of waveform  (ms) */
#define FCOMP 256     /* Number of freq comp = recsize - 1, (512/2) */
#define BANDS 6        /* Number of freq bands */

#define F0 0000.0     /* Frequency band limits (Hz).  B1 */
#define F1  400.0
#define F2  800.0     /* B2 */
#define F3 1500.0
#define F4 1200.0     /* B3 */
#define F5 2000.0     /* B4 */
#define F6 3500.0     /* B5 */
#define F7 5000.0     /* B6 */
#define F8 8000.0

#define TLENd 20      /* Smoothing interval in each band for 1st pass (ms) */
#define TLENl 10      /* Smoothing interval in each band for 2nd pass (ms) */

/* ror.c */

#define DTd 50  /* Time step for ROR in each band for 1st pass (ms) */
#define DTl 26  /* Time step for ROR in each band for 2nd pass (ms) */

/* mpeak.c, mermel.c */

#define PEAK_THRESd 9.0 /* Mermel: threshold for 1st pass (B1-6),
   2nd pass (B2-6)*/
#define PEAK_THRESl 6.0 /* Mermel: B1 threshold for 2nd pass. */

/* localize.c */

#define LMTIME 0.030  /* Max separation time between vec1 and vec2 peaks */

/* order_pk.c */

#define PEAK struct peak  /* ROR peak data structure */
PEAK{
  double time;  /* Time of pk (s) */
  double val;   /* Value of pk (dB) */
  int band;     /* Band of pk */
```

```
  char *tag;     /* Tag of pk (obs, pivot, copivot, sstag) */
  int lm;        /* 0 = not lm; 1 = g lm; 2 = s lm; 3 = b lm; 4 = v lm */
  int magnet;    /* Pivot magnet w/ which pk is assoc. (copivot, sstag only) */
  int pair;      /* Index denoting B1 pk or pivot with which pivot is paired,
   or left unpaired. (pivot and opivot only) */
  int pos;  /* Pos of B1-5 pks (0 = unprd, 1 = iv, 2 = nas->obs, 3 = obs->nas
       4 = utt-final, 5 = utt-init) */
  int stag;      /* Binary: 1 if there are sstags associated with pivot */
  float hpror;   /* Hpror value in vicinity of pivot (dB) */
  int sslen;     /* Length of ss region associated with pivot (ms) */
  int marg;      /* Margin between pivot and beginning of ss region (ms) */
  float error;   /* Debug: max error summed across f (dB) */
  float oerror; /* Debug: max error at one f sample (dB) */
  float tref;    /* Debug: time of reference spectrum for ss determination (s)*/
  int sil;       /* Preceding silence length (opivot only) (ms) */
  int burst;     /* Time between opivot and preceding sil (opivot only) (ms) */
};

/* verify_obs.c */

#define ROWS 400 /* Max num peaks in one band; max num silence intervals */
#define CONSON_T 150  /* Max allowed time for a consonant region (ms) */
#define DELETE_T 50   /* Max allowed time bet min and pk for deletion (ms) */
#define VOCALIC_LEVEL 20.0  /* Drop from max in B1 eng for voc/son reg (dB) */
#define VOCALIC_LEN 20  /* Min length for a voc/son region (ms) */

#define B1PEAK struct B1peak    /* B1 ROR peak data structure */
B1PEAK{
  double time;   /* Time of pk (s) */
  double val;    /* Value of pk (dB) */
  char *tag;     /* Tag of pk (original, inserted, deleted) */
};

/* tag.c, order_pk.c, find_pivot.c */

#define LT_OBS_OOZE 0.035   /* Time ooze to left of B1 -pk (s) */
#define RT_OBS_OOZE 0.020   /* Time ooze to right of B1 +pk (s) */
#define LAST_OBS_OOZE 0.050 /* Time ooze to left of last B1 pk (s) */
#define LT_SON_OOZE 0.035   /* Time ooze to left of pivot (s) */
#define RT_SON_OOZE 0.020   /* Time ooze to right of pivot (s) */

#define BLOCK 0.070     /* Block of time around a pivot for sstags (s) */

#define IV_REG 0.150      /* Max time for pairing an iv nasal (s) */
#define NAS_OBS_REG 0.150 /* Max time for pairing a nas->obs nasal (s) */
#define OBS_NAS_REG 0.100 /* Max time for pairing an obs->nas nasal (s) */
#define LAST_REG 0.200    /* Max time for pairing an utt-end nasal (s) */

/* verify_son.c, compare_spec.c, son_lm_check.c */

#define UNPRD_TREF 30     /* Ref t after unprd IV pvt for ss comp (ms) */
#define NAS_OBS_TREF 30   /* Ref t after -pvt in nas->obs for ss comp (ms) */
#define OBS_NAS_TREF 20   /* Ref t before +pvt in obs->nas for ss comp (ms) */
```

```
#define UNPRD_SSLEN 35   /* Min sslen nec for unprd iv ss region (ms) */
#define IV_SS_RATIO 0.70 /* Min (sslen / tot len) nec for iv ss region */
#define NAS_OBS_SSLEN 20 /* Min sslen nec for nas->obs ss region (ms)*/
#define OBS_NAS_SSLEN 20 /* Min ss length nec for obs->nas ss region (ms)*/

#define UNPRD_MARG 25    /* Max margin allowed for unprd iv pivot (ms) */
#define NAS_OBS_MARG 25  /* Max margin allowed for nas->obs pivot (ms) */
#define OBS_NAS_MARG 15  /* Max margin allowed for obs->nas pivot (ms) */

#define IV_ERROR 80.0     /* Max abs error, summed across f, for ss (dB) */
#define NAS_OBS_ERROR 70.0
#define OBS_NAS_ERROR 50.0
#define IV_OERROR 6.0  /* Max abs error, at one f sample, for ss (dB) */
#define NAS_OBS_OERROR 6.0
#define OBS_NAS_OERROR 6.0

#define IV_SSTAG 0.040      /* Max time between sstag and iv pivot (s) */
#define NAS_OBS_SSTAG 0.070 /* Max time between sstag and nas->obs pivot (s) */
#define OBS_NAS_SSTAG 0.070 /* Max time between sstag and obs->nas pivot (s) */

#define UNPRD_REG 50     /* Length to check for ss'ness of an unprd lm (ms) */
#define COMP_F 600.0   /* Hi freq cut off for ss comparison of spectra (Hz) */

#define HPROR_OOZE 12  /* Time ooze around pivot to look for hpror max (ms) */
#define HPROR_LIMIT 9.0  /* Min hpror value near pivot for abruptness (dB) */

/* burst.c, find_opivot.c */

#define NAS_OOZE 0.010    /* Nas ooze beyond opivot for nas->obs opivots (s) */
#define VOT 0.020         /* Min VOT for distinct burst and VO (s) */
#define BURST_OOZE 0.030  /* Burst ooze.  Also min time between 2 bursts (s) */
#define NAS_BURST 0.200   /* Max time between AC S -lm and burst (s) */
#define SILTIME 5         /* Min silence length at closure (ms) */
#define BURSTTIME 30      /* Max time between silence and burst (ms) */
#define SIL0 10.0         /* B1 silence threshold at closure (dB). */
#define SIL1 10.0         /* B2 silence threshold at closure (dB). */
#define SIL2 10.0         /* B3 silence threshold at closure (dB). */
#define SIL3 10.0         /* B4 silence threshold at closure (dB). */
#define SIL4 10.0         /* B5 silence threshold at closure (dB). */
#define SIL5 10.0         /* B6 silence threshold at closure (dB). */

/****************************** GLOBAL.H ********************************
   Global parameters in lm.c .
*/

char *progName = "lm";
int debug_level = 0;

/******************************** LM.C **********************************
Finds landmarks.  Has "g", "s", "b", and the new "v" modules to find lm's.
"v" lm's are missed "g" lm's but detected based on higher band ROR peaks.

Input:  *.sgram -- spectrogram (FEA_SPEC)
        *.hpror -- high pass ROR, generated from hpror.c (FEA)
```

```
Output: *.agram -- smoothed spectrogram (FEA_SPEC)
        *.energy1 -- 1st pass: energy in 6 bands (FEA)
        *.energy2 -- 2nd pass: energy in 6 bands (FEA)
        *.ror1 -- 1st pass: ROR in 6 bands (FEA)
        *.ror2 -- 2nd pass: ROR in 6 bands (FEA)
*.peak1 -- coarse pass: B1-6 peaks grouped across bands (ASCII)
*.peak2 -- fine pass: B1-6 peaks grouped across bands (ASCII)
*.peak3 -- localized: B1-6 peaks grouped across bands (ASCII)
*.B1 -- B1 peaks with deleted and inserted pks marked (ASCII)
*.tag -- B1-5 pks tagged with obs, (co)opivot, (co)pivot, sstag (ASCII)
*.pivot -- B1-5 pks attached with details re. ss, hpror, etc. (ASCII)
*.lm -- g, s, b landmarks (label file)

Usage:  lm  <firstname of sgram file>

***/

#include "include.h"
#include "define.h"
#include "global.h"

main(argc, argv)
     int argc;
     char *argv[];
{
/*** Program variables ***/

/* Sgram header info */

  void sgram_read();      /* Reads sgram header info */
  int sgramlength;        /* Length of sgram file */
  int recsize;            /* Record size of sgram file */
  double sf;              /* Sampling frequency in sgram file (Hz) */
  double record_freq;     /* Record frequency of sgram file (Hz) */
  double start_time;      /* Start time of sgram file (s) */

/* Sgram smoothing and energy computing. */

  void energy();      /* Smooths and generates multi-channel energy */
  void energy_write(); /* Writes out *.energy file */
  float sil[BANDS];     /* Vector of silence levels of all bands. */
  float eng[BANDS*WAVELEN];  /* Array of energy values in each band */

/* ROR. */
  void ror();              /* Calculates the ROR of energy in a band. */
  void ror_write();        /* Writes out *.ror file */
  float rorarray[BANDS*WAVELEN]; /* 5-band ROR array */

/* Peak picking. */
  void mpeak();       /* Picks peaks using Mermelstein's algorithm. */
  int start;          /* Index for vector[] */
  double vector1[ROWS*BANDS*2];  /* Detection */
  double vector2[ROWS*BANDS*2];  /* Sensitized peaks */
```

```
                    /* Peak times and values.  Bands one after the other */

/* Landmark localization */
  void localize();
  double vector3[ROWS*BANDS*2];  /* Localized peaks */

/* Peak grouping. */
  void order_pk();                /* Arranges peaks by time only */
  void pgroup();                  /* Arranges peaks by time and band */
  PEAK pk_seq[BANDS*ROWS];        /* Peak sequence ordered by time */
  char tagstring[BANDS*ROWS][21]; /* Space for tag field of pk_seq */
  int k;                          /* Index for pk_seq */
  int tot_num_pk;                 /* Total number of peaks in (B1)pk_seq */

/* G landmarks */
  void verify_g();                /* Verifies B1 peaks (g landmarks) */
  void write_B1lm();              /* Writes out *.B1 file */
  B1PEAK B1pk_seq[ROWS];          /* B1 peak sequence */
  B1PEAK B1pk_seq2[ROWS];
  char B1tagstring[ROWS][21];     /* Space for tag field of B1pk_seq */
  char B1tagstring2[ROWS][21];    /* Space for tag field of B1pk_seq */
  int tot_num_B1pk;               /* Total number of B1 peaks in B1pk_seq */

/* S landmarks */
  void tag();                /* Tags vec3 peaks */
  void verify_s();           /* Verifies s lm's for s-s and abruptness */

/* B landmarks */
  void burst();              /* Finds vls stop and affric bursts */

/* Write out landmark and pivot info into output files */
  void write_lm();           /* Writes out *.lm and *.pivot files */

/*** Program body ***/

/* Check command line arguments and set processing parameters */

  if (argc < 2){
    printf("Usage:  lm  <firstname of sgram file>\n");
    exit(-1);
  }

/* Read in header info from sgram file */

  sgram_read(argv, &sgramlength, &recsize, &sf, &record_freq, &start_time);

/*** First pass:  coarse ***/

/* Smooth sgram and generate energy.  Input: *.sgram.  Output: *.agram,
   eng[], sil[]. */

  energy(argc, argv, TLENd, sil, eng);
  energy_write(argc, argv, ".energy1", eng);
```

```
/* Find ROR in each band.  Input:  eng[].  Output:  ror[]. */

  ror(eng, sgramlength, DTd, rorarray);
  ror_write(argc, argv, ".energy1", ".ror1", rorarray);

/* Find peaks of ror waveforms in each frequency band.  Generates a
   vector of peak times and values, each band separated by no. peaks in band.
   Input:  ror*.  Output:  vector1[] */

  start = 0;
  mpeak(rorarray, 0, sgramlength, record_freq, start_time, PEAK_THRESd,
        PEAK_THRESd, &start, vector1);
  mpeak(rorarray, 1, sgramlength, record_freq, start_time, PEAK_THRESd,
PEAK_THRESd, &start, vector1);
  mpeak(rorarray, 2, sgramlength, record_freq, start_time, PEAK_THRESd,
PEAK_THRESd, &start, vector1);
  mpeak(rorarray, 3, sgramlength, record_freq, start_time, PEAK_THRESd,
PEAK_THRESd, &start, vector1);
  mpeak(rorarray, 4, sgramlength, record_freq, start_time, PEAK_THRESd,
PEAK_THRESd, &start, vector1);
  mpeak(rorarray, 5, sgramlength, record_freq, start_time, PEAK_THRESd,
PEAK_THRESd, &start, vector1);

/*** Second pass:  fine ***/

/* Smooth sgram and pick max energy.  Input: *.sgram.  Output: *.agram,
   *.energy2, sil[], eng[]. */

  energy(argc, argv, TLEN1, sil, eng);
  energy_write(argc, argv, ".energy2", eng);

/* Find ROR in 6 bands.  Input:  eng[].  Output:  ror[], *.ror2 */

  ror(eng, sgramlength, DT1, rorarray);
  ror_write(argc, argv, ".energy2", ".ror2", rorarray);

/* Find PEAKs of ROR waveforms in each frequency band.  Generates a
   vector of peak times and values.  Input:  ror* vectors.  Output:
   vector2[] */

  start = 0;
  mpeak(rorarray, 0, sgramlength, record_freq, start_time, PEAK_THRES1,
PEAK_THRES1, &start, vector2);
  mpeak(rorarray, 1, sgramlength, record_freq, start_time, PEAK_THRESd,
PEAK_THRESd, &start, vector2);
  mpeak(rorarray, 2, sgramlength, record_freq, start_time, PEAK_THRESd,
PEAK_THRESd, &start, vector2);
  mpeak(rorarray, 3, sgramlength, record_freq, start_time, PEAK_THRESd,
PEAK_THRESd, &start, vector2);
  mpeak(rorarray, 4, sgramlength, record_freq, start_time, PEAK_THRESd,
PEAK_THRESd, &start, vector2);
  mpeak(rorarray, 5, sgramlength, record_freq, start_time, PEAK_THRESd,
PEAK_THRESd, &start, vector2);
```

```
/* Use vector1 and vector2 to localize lm's.  Input: vector1[], vector2[].
   Output: vector3[]. */

  localize(argv, vector1, vector2, vector3);

/* Convert vectors to matrix format and output to files.  These peaks are
   purely the ROR (localized) peaks from Mermelstein's peak picking;  involves
   no modification due to verification.  Input: vector1,2,3.
   Output: *.peak1,2,3 . */

  for (k = 0; k < BANDS*ROWS; k++) /* Allocate space for tag field of pk_seq */
    pk_seq[k].tag = tagstring[k];

  order_pk(vector1, 6, pk_seq, &tot_num_pk);
  pgroup(argv, pk_seq, tot_num_pk, ".peak1", 0.0005);

  order_pk(vector2, 6, pk_seq, &tot_num_pk);
  pgroup(argv, pk_seq, tot_num_pk, ".peak2", 0.0005);

  order_pk(vector3, 6, pk_seq, &tot_num_pk);
  pgroup(argv, pk_seq, tot_num_pk, ".peak3", 0.0005);

/*** G landmarks ***/

/* Verify B1 landmarks by:
   pairing up unpaired B1 lm's; and deleting +/- pairs that don't
   encompass voc/son regions.  Updates vector3's B1 with valid lm's, and
   outputs a file with B1 peaks inserted, deleted, or original.
   Input:  vector3[].
   Output:  vector3[] (updated), *.B1
*/

  for (k = 0; k < ROWS; k++){   /* Allocate space for tag field of B1pk_seq. */
    B1pk_seq[k].tag = B1tagstring[k];
    B1pk_seq2[k].tag = B1tagstring2[k];
  }

  verify_g(vector3, eng, record_freq, start_time, sgramlength,
     B1pk_seq, B1pk_seq2, &tot_num_B1pk);
  write_B1pk(argv, B1pk_seq2, tot_num_B1pk);

/*** S and V landmarks ***/

/* Tag B1-5 peaks from vec3.  Input: vector3.  Output: pk_seq
   (B1-5, tagged with obs, pivot, copivot, sstag), tot_num_pk, *.tag. */

  order_pk(vector3, 5, pk_seq, &tot_num_pk);
  tag(argv, pk_seq, tot_num_pk);

/* Verify pivots as s lm's by using steady-state, abruptness, and staggered
   peaks criteria of sonorant regions.  If a pivot passes abruptness and
   staggered peaks criteria but not ss criterion, then count as a v lm.
   Input: pk_seq[], *.agram.  Output: pk_seq (tagged with lm, ss info, pos).
*/
```

```
  verify_s(argv, pk_seq, tot_num_pk);

/*** B landmarks ***/

/* Detect fric and aspiration bursts and closures.  Input:
   pk_seq, eng.  Output: pk_seq (tagged with bursts). */

  burst(pk_seq, tot_num_pk, eng, sgramlength, start_time, record_freq);

/* Write g, s, b, and v lm's into lm file.  Write out B1 pk and (o)pivot
   descriptors into pivot file   Input: pk_seq.  Output: *.lm, *.pivot */

  write_pk(argv, pk_seq, tot_num_pk);
}

/*************************** SGRAM_READ.C ******************************
   Reads in header info from sgram file
*/

#include "include.h"
#include "define.h"

void sgram_read(argv, sgramlength, recsize, sf, record_freq, start_time)
     char *argv[];          /* In: command line arguments */
     int *sgramlength;      /* Out: length of sgram file */
     int *recsize;          /* Out: record size of sgram file */
     double *sf;            /* Out: sampling frequency of sd file */
     double *record_freq;   /* Out: record frequency of sgram file */
     double *start_time;    /* Out: start time of sgram file */
{
  extern char *progName;      /* Main prog name */
  char sgramfilename[30];     /* Sgram file name */
  struct header *inputHd;     /* Sgram file header */
  FILE *fpin;                 /* Sgram file pointer */

/* Open sgram input file and read header info. */

  strcpy(sgramfilename, argv[1]);
  eopen(progName, strcat(sgramfilename, ".sgram"), "r",
FT_FEA, FEA_SPEC, &inputHd, &fpin);
  *sgramlength = inputHd->common.ndrec;
  *recsize = get_fea_siz("re_spec_val", inputHd, (short *)NULL, (long **)NULL);
  *sf = get_genhd_val("sf", inputHd, 1.0);
  *record_freq = *get_genhd_d("record_freq", inputHd);
  *start_time = *get_genhd_d("start_time", inputHd);

/* Close shop */

  free_header(inputHd);
  fclose(fpin);
}

/*************************** ENERGY.C ********************************
```

```
   Smooths sgram and compute energy in each band
   from spectrogram.  Records eng[] wrt beginning silence.
   In: *.sgram.
   Out: *.agram, eng[] (dB).
*/

#include "include.h"
#include "define.h"

void energy(argc, argv, tlen, sil, eng)
     int argc;          /* In: number of command line arguments */
     char *argv[];      /* In: command line arguments */
     int tlen;          /* In: time interval over which to smooth (ms) */
     float *sil;        /* Out: vector of silence levels in each band (dB)*/
     float *eng;        /* Out: energy, one band after another (dB) */
{
  extern char *progName;              /* Main prog name */
  char sgramfilename[30], agramfilename[30];  /* Input, output file names */
  char *iname;                        /* Input file name */
  struct header *inputHd, *outputHd;  /* File headers */
  struct feaspec *i_rec, *o_rec;      /* Input record from sgram file (dB). */
  float *i_ptr, *o_ptr;               /* Pointer to input record. */
  FILE *fpin, *fpout;                 /* File pointers */
  int sgramlength;                    /* Length of sgram file */
  double sf;                          /* Sampling frequency of sd file */
  double *linear=(double *)calloc(FCOMP*30, sizeof(double));   /* smoothing*/
  double *t_av = (double *)calloc(FCOMP, sizeof(double));      /* smoothing*/
  float max;         /* Maximum smoothed energy in freq band at that time. */
  int fn1[BANDS], fn2[BANDS];      /* Frequency limits for bands (indices). */
  int i, j, k;       /* Generic indices. */
  int band;          /* Band index */
  int cirptr;        /* Circular pointer, for smoothing. */
  int start, end;    /* Start and end indices for processing sgram wave */

/* Open sgram input file, read header info, and allocate input buffer */

  strcpy(sgramfilename, argv[1]);
  eopen(progName, strcat(sgramfilename, ".sgram"), "r",
FT_FEA, FEA_SPEC, &inputHd, &fpin);
  sgramlength = inputHd->common.ndrec;
  sf = get_genhd_val("sf", inputHd, 1.0);
  i_rec = allo_feaspec_rec(inputHd, FLOAT);

/* Convert frequency band limits (Hz) to indices. */

  fn1[0] = (int)(F0 * 2.0 * (float)FCOMP/sf);
  fn2[0] = (int)(F1 * 2.0 * (float)FCOMP/sf);
  fn1[1] = (int)(F2 * 2.0 * (float)FCOMP/sf);
  fn2[1] = (int)(F3 * 2.0 * (float)FCOMP/sf);
  fn1[2] = (int)(F4 * 2.0 * (float)FCOMP/sf);
  fn2[2] = (int)(F5 * 2.0 * (float)FCOMP/sf);
  fn1[3] = (int)(F5 * 2.0 * (float)FCOMP/sf);
  fn2[3] = (int)(F6 * 2.0 * (float)FCOMP/sf);
  fn1[4] = (int)(F6 * 2.0 * (float)FCOMP/sf);
```

```
  fn2[4] = (int)(F7 * 2.0 * (float)FCOMP/sf);
  fn1[5] = (int)(F7 * 2.0 * (float)FCOMP/sf);
  fn2[5] = (int)(F8 * 2.0 * (float)FCOMP/sf);

/*** Find silence levels from utt beginning, for eng[]. ***/

/* Read in first 30 ms worth of records from sgram, convert from dB to energy,
   compute first time average. */

  for (j = 0; j < FCOMP; j++)
    *(t_av + j) = 0.0;

  for (cirptr = 0; cirptr < 30; cirptr++){
    get_feaspec_rec(i_rec, inputHd, fpin);
    i_ptr = i_rec->re_spec_val;

    for (j = 0; j < FCOMP; j++){
      *(t_av + j) += (*(linear + cirptr*FCOMP + j) =
              exp(.23026* (*(i_ptr+j))));
    }
  }

/* Smooth first 40 frames of sgram and collect silence level at frame 40. */

  cirptr = 0;
  for (i = 30; i <= 40; i++){

    /* Read in next record, average it in, update cirptr. */
    get_feaspec_rec(i_rec, inputHd, fpin);
    i_ptr = i_rec->re_spec_val;
    *(t_av + j) -= *(linear + cirptr*FCOMP + j);
    *(t_av + j) += (*(linear + cirptr*FCOMP +j)
    = exp(.23026* *(i_ptr+j)));
    cirptr++;
  }

  /* Record silence level in each band at 40th frame */
  for (band = 0; band < BANDS; band++){
    max = 0.0;
    for (j = fn1[band]; j < fn2[band]; j++)
      if (max < *(t_av + j))
max = *(t_av + j);
    sil[band] = 10.0*log10(max * (float)tlen / 30.0);
  }

/* Close shop */

  free_fea_rec(i_rec);
  free_header(inputHd);
  fclose(fpin);

/*** Now compute and record agram, and eng[] ref. silence levels. ***/

/* Open sgram input file and allocate input buffer. */
```

```
  strcpy(sgramfilename, argv[1]);
  iname = eopen(progName, strcat(sgramfilename, ".sgram"), "r",
FT_FEA, FEA_SPEC, &inputHd, &fpin);
  i_rec = allo_feaspec_rec(inputHd, FLOAT);

/* Open agram output file, create output file header, and allocate output
   buffer. */

  strcpy(agramfilename, argv[1]);
  eopen(progName, strcat(agramfilename, ".agram"), "w",
FT_FEA, FEA_SPEC, &outputHd, &fpout);

  outputHd = copy_header(inputHd);
  add_source_file(outputHd, iname, inputHd);
  outputHd->common.tag = NO;
  (void) strcpy(outputHd->common.prog, progName);
  outputHd->variable.refer = inputHd->variable.refer;
  add_comment(outputHd, get_cmd_line(argc, argv));
  (void)update_waves_gen(inputHd, outputHd, 1.0, 1.0);
  write_header(outputHd, fpout);

  o_rec = allo_feaspec_rec(outputHd, FLOAT);

/* Read in first tlen worth of records, convert to linear,
   compute first time average. */

  for (j = 0; j < FCOMP; j++)
    *(t_av + j) = 0.0;

  for (cirptr = 0; cirptr < tlen; cirptr++){
    get_feaspec_rec(i_rec, inputHd, fpin);
    i_ptr = i_rec->re_spec_val;
    for (j = 0; j < FCOMP; j++){
      *(t_av + j) += (*(linear + cirptr*FCOMP + j) =
              exp(.23026* (*(i_ptr+j))));
    }
  }

/* Convert t_av to dB and record in output record. */

  for (j = 0; j < FCOMP; j++){
    if (*(t_av+j) <= 0.0)
      *(t_av+j) = 0.00001;
    o_rec->re_spec_val[j] = (float)(10.0*log10(*(t_av+j)));
  }

/* Pad beginning of agram file with tlen/2 points long of its first
   record value.  */

  for (i = 0; i < tlen/2; i++)
    put_feaspec_rec(o_rec, outputHd, fpout);

/* Compute 5 band energy, subtracting off silence levels found above,
```

```
   and record in eng[]. */

  for (band = 0; band < BANDS; band++){
    max = 0.0;
    for (j = fn1[band]; j < fn2[band]; j++)
      if (max < *(t_av + j))
max = *(t_av + j);
    for (i = 0; i < tlen/2; i++)
      eng[band*sgramlength + i] = 10.0*log10(max) - sil[band];
  }

/* Process entire waveform. */

  cirptr = 0;
  start = tlen/2;
  end = sgramlength - tlen/2;

  for (i = start; i < end; i++){

/* Read in next record, average it in, update cirptr. */

    get_feaspec_rec(i_rec, inputHd, fpin);
    i_ptr = i_rec->re_spec_val;

    for (j = 0; j < FCOMP; j++){
      *(t_av + j) -= *(linear + cirptr*FCOMP + j);
      *(t_av + j) += (*(linear + cirptr*FCOMP +j)
      = exp(.23026* *(i_ptr+j)));
    }
    cirptr++; if (cirptr == tlen) cirptr = 0;

/* Convert to dB and record in output record.  Write record to agram file. */

    for (j = 0; j < FCOMP; j++){
      if (*(t_av+j) <= 0.0)
*(t_av+j) = 0.00001;
      o_rec->re_spec_val[j] = (float)(10.0*log10(*(t_av+j)));
    }
    put_feaspec_rec(o_rec, outputHd, fpout);

/* Compute 5 band energy, subtracting off silence levels found above,
   and record in eng[]. */

    for (band = 0; band < BANDS; band++){
      max = 0.0;
      for (j = fn1[band]; j < fn2[band]; j++)
if (max < *(t_av + j))
  max = *(t_av + j);
      eng[band*sgramlength + i] = 10.0*log10(max) - sil[band];
    }
  }

/* Pad end of agram with tlen/2 points of last value */
```

```
  for (i = end; i < end + tlen/2; i++)
    put_feaspec_rec(o_rec, outputHd, fpout);

/* Pad end of eng[] with tlen/2 points of last value */

  for (i = end; i < end + tlen/2; i++)
    for (band = 0; band < BANDS; band++){
      eng[band*sgramlength + i] = eng[band*sgramlength + end - 1];
  }

/* Close shop */

  free_fea_rec(i_rec);   free_fea_rec(o_rec);
  free_header(inputHd);  free_header(outputHd);
  fclose(fpin);          fclose(fpout);

  cfree(linear);
  cfree(t_av);
}

/************************** ENERGY_WRITE.C ****************************
   Writes 6 band energy array to energy (FEA) file.
   In:  eng[].
   Out:  energy file.
*/

#include "include.h"
#include "define.h"

void energy_write(argc, argv, out_ext, eng)
     int argc;           /* In: number of command line args */
     char *argv[];       /* In: command line args */
     char *out_ext;      /* In: extension for output energy file */
     float *eng;         /* In: 6 band energy array */
{
  extern char *progName;                    /* Main prog name */
  char *iname;                              /* File name */
  char sgramfilename[30], energyfilename[30]; /* File names */
  struct header *inputHd, *outputHd;        /* File headers */
  FILE *fpin, *fpout;                       /* File pointers */
  struct fea_data *o_rec;                   /* Output record */
  float *o_ptr[BANDS];                      /* Output record pointers */
  int sgramlength;                          /* Length of sgram file */
  int i;                                    /* Time index */
  int band;                                 /* Band index */

/* Open sgram input file and read header info, for purposes of creating
   output energy file header */

  strcpy(sgramfilename, argv[1]);
  iname = eopen(progName, strcat(sgramfilename, ".sgram"), "r",
FT_FEA, FEA_SPEC, &inputHd, &fpin);
  sgramlength = inputHd->common.ndrec;
```

```
/* Open energy output file, create output file header, and allocate output
   buffer. */

  strcpy(energyfilename, argv[1]);
  eopen(progName, strcat(energyfilename, out_ext), "w",
NONE, NONE, &outputHd, &fpout);

  outputHd = new_header(FT_FEA);
  add_source_file(outputHd, iname, inputHd);
  outputHd->common.tag = NO;
  (void) strcpy(outputHd->common.prog, progName);
  outputHd->variable.refer = NULL;
  add_comment(outputHd, get_cmd_line(argc, argv));
  (void) add_fea_fld("energy1", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void) add_fea_fld("energy2", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void) add_fea_fld("energy3", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void) add_fea_fld("energy4", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void) add_fea_fld("energy5", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void) add_fea_fld("energy6", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void)update_waves_gen(inputHd, outputHd, 1.0, 1.0);
  write_header(outputHd, fpout);

  o_rec = allo_fea_rec(outputHd);
  o_ptr[0] = (float *) get_fea_ptr(o_rec, "energy1", outputHd);
  o_ptr[1] = (float *) get_fea_ptr(o_rec, "energy2", outputHd);
  o_ptr[2] = (float *) get_fea_ptr(o_rec, "energy3", outputHd);
  o_ptr[3] = (float *) get_fea_ptr(o_rec, "energy4", outputHd);
  o_ptr[4] = (float *) get_fea_ptr(o_rec, "energy5", outputHd);
  o_ptr[5] = (float *) get_fea_ptr(o_rec, "energy6", outputHd);

/* Assign values to o_ptr's and write out to energy file */

  for (i = 0; i < sgramlength; i++){
    for (band = 0; band < BANDS; band++)
      *o_ptr[band] = eng[band*sgramlength + i];
    put_fea_rec(o_rec, outputHd, fpout);
  }

/* Close shop */

  fclose(fpin);
  fclose(fpout);
}

/****************************** ROR.C ********************************
   Finds the rate of rise of energy by taking overlapping
   time differences.
   In: eng[].
```

```
   Out: rorarray[].
*/

#include "include.h"
#include "define.h"

void ror(eng, sgramlength, dt, rorarray)
     float *eng;              /* In: 6-band energy array */
     int sgramlength;         /* In: length of sgram file */
     int dt;                  /* In: dt for the band */
     float *rorarray;         /* Out: array of B1-6 ror values */
{
  int i;               /* Time index */
  int begin, end;      /* Begin and end indices for computing rorarray */
  int band;            /* Band index */

/* Pre zero pad rorarray with dt/2 pts to make it a symmetric diff. */

  for (i = 0; i < dt/2; i++)
    for (band = 0; band < BANDS; band++)
      rorarray[band*sgramlength + i] = 0.0;

/* Process entire waveform. */

  begin = dt/2;
  end = sgramlength - dt/2;

  for (i = begin; i < end; i++)
    for (band = 0; band < BANDS; band++)
      rorarray[band*sgramlength + i] =
eng[band*sgramlength + i + dt/2] - eng[band*sgramlength + i - dt/2];

/* Post zero pad rorarray with dt/2 pts to make rorarray as long as
   eng[] */

  for (i = end; i < end + dt/2; i++)
    for (band = 0; band < BANDS; band++)
      rorarray[band*sgramlength + i] = 0.0;
}

/*************************** ROR_WRITE.C ******************************
   Writes ror vectors in all bands to one *.ror (FEA) file.
   In:  rorarray[].
   Out:  ror file.
*/

#include "include.h"
#include "define.h"

void ror_write(argc, argv, in_ext, out_ext, rorarray)
     int argc;            /* In: number of command line arguments */
     char *argv[];        /* In: command line arguments */
     char *in_ext;        /* In: extension for input energy file */
     char *out_ext;       /* In: extension for output ROR file */
```

```
    float *rorarray;     /* In: 6-band ROR array */
{
  extern char *progName;                    /* Main prog name */
  char *iname;                              /* File name */
  char energyfilename[30], rorfilename[30]; /* File names */
  struct header *inputHd, *outputHd;        /* File headers */
  FILE *fpin, *fpout;                       /* File pointers */
  struct fea_data *o_rec;                   /* Output record */
  float *o_ptr[BANDS];                      /* Output record pointers */
  int sgramlength;                          /* Length of sgram file */
  int i;                                    /* Time index */
  int band;                                 /* Band index */

/* Open energy input file and read header info, for purposes of creating
   output ror file header */

  strcpy(energyfilename, argv[1]);
  iname = eopen(progName, strcat(energyfilename, in_ext), "r",
FT_FEA, NONE, &inputHd, &fpin);
  sgramlength = inputHd->common.ndrec;

/* Open ror output file, create output file header, and allocate output
   buffer. */

  strcpy(rorfilename, argv[1]);
  eopen(progName, strcat(rorfilename, out_ext), "w",
NONE, NONE, &outputHd, &fpout);

  outputHd = new_header(FT_FEA);
  add_source_file(outputHd, iname, inputHd);
  outputHd->common.tag = NO;
  (void) strcpy(outputHd->common.prog, progName);
  outputHd->variable.refer = NULL;
  add_comment(outputHd, get_cmd_line(argc, argv));
  (void) add_fea_fld("ror1", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void) add_fea_fld("ror2", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void) add_fea_fld("ror3", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void) add_fea_fld("ror4", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void) add_fea_fld("ror5", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void) add_fea_fld("ror6", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void)update_waves_gen(inputHd, outputHd, 1.0, 1.0);
  write_header(outputHd, fpout);

  o_rec = allo_fea_rec(outputHd);
  o_ptr[0] = (float *) get_fea_ptr(o_rec, "ror1", outputHd);
  o_ptr[1] = (float *) get_fea_ptr(o_rec, "ror2", outputHd);
  o_ptr[2] = (float *) get_fea_ptr(o_rec, "ror3", outputHd);
  o_ptr[3] = (float *) get_fea_ptr(o_rec, "ror4", outputHd);
```

```
  o_ptr[4] = (float *) get_fea_ptr(o_rec, "ror5", outputHd);
  o_ptr[5] = (float *) get_fea_ptr(o_rec, "ror6", outputHd);

/* Assign values to o_ptr's and write out to *.ror */

  for (i = 0; i < sgramlength; i++){
    for (band = 0; band < BANDS; band++)
      *o_ptr[band] = rorarray[band*sgramlength + i];
    put_fea_rec(o_rec, outputHd, fpout);
  }

/* Close shop */

  fclose(fpin);
  fclose(fpout);
}

/****************************** MPEAK.C *********************************
   Picks +/- peaks using Mermelstein's algorithm for
   one band's waveform.  First segments waveform into +/- regions, then
   recursively applies Mermelstein's algorithm within each region.
   In: rorarray[].
   Out:  times and values of peaks.
*/

#include "include.h"
#include "define.h"

void mpeak(rorarray, band, sgramlength, record_freq, start_time, peak_thres,
   dip_thres, start, vector)
     float *rorarray;    /* In: 6-band ROR array */
     int band;           /* In: current band */
     int sgramlength;    /* In: length of sgram file */
     double record_freq; /* In: record frequency of sgram file */
     double start_time;  /* In: start time of sgram file */
     float peak_thres;   /* In: peak threshold for picking */
     float dip_thres;    /* In: dip threshold for picking */
     int *start;         /* In: starting index for vector[]
    Out: modified to ending index for vector[] */
     double *vector;     /* Out: vector of times and values of peaks */
{
  int mermel();        /* Recursive algorithm to find peaks; returns npeak. */
  int i, j;
  int boundary[WAVELEN];  /* Boundaries delimiting regions of same sign. */
  int numregions;         /* Number of regions, each of same sign. */
  float dip;              /* Max difference = dip, in a mermel segment. */
  int dip_loc;            /* Location of dip. */
  float peak[ROWS];       /* Peak values of band. */
  int peak_loc[ROWS];     /* Peak locations of band. */
  int npeak = 0;          /* Number of peaks in band. */
  int ror_start, ror_end; /* Start and end indices for 1 band in rorarray */

/* Find starting index of each region of same +/- sign, and record in
   boundary[]. */
```

```
  ror_start = band*sgramlength;
  ror_end = (band+1)*sgramlength;
  boundary[0] = ror_start;
  j = 1;
  for (i = ror_start + 1; i < ror_end; i++)
    if (rorarray[i-1] * rorarray[i] < 0.0){
      boundary[j] = i;
      j++;
    }
  boundary[j] = ror_end;
  numregions = j;

/* Within each region, apply Mermelstein's algorithm. */

  for (j = 0; j < numregions; j++)
    npeak = mermel(rorarray, band, boundary[j], boundary[j+1]-1,
   npeak, peak_thres, dip_thres, sgramlength, peak, peak_loc);

/* Output is vector of peak times and peak values.  Peaks already in order.
   Record number of peaks at the top of the band of the vector. */

  vector[2 * *start] = (double)npeak;   /* Number of peaks */
  vector[2 * *start + 1] = 0.0;        /* Zero fill */

  for (i = 1; i < npeak + 1; i++){
    vector[2*(i + *start)] = (double)(peak_loc[i-1] - ror_start)/record_freq
      + start_time;
    vector[2*(i + *start)+1] = peak[i-1];
  }
  *start += npeak + 1;
}

/****************************** MERMEL.C ********************************
    Locates peaks using Mermelstein's algorithm.
    In:  rorarray[] segment.
    Out:  a peak value and time.
*/

#include "include.h"
#include "define.h"

int mermel(rorarray, band, start, end, npeak, peak_thres, dip_thres,
   sgramlength, peak, peak_loc)
     float *rorarray;    /* Input: 5-band ror array */
     int band;           /* Input: current band */
     int start, end;     /* Input: start and end indices of ror segment */
     int npeak;          /* Input: number of peaks so far */
     float peak_thres;   /* Input: peak threshold for picking */
     float dip_thres;    /* Input: dip threshold for picking */
     int sgramlength;    /* Input: length of sgram file */
     float *peak;        /* Output: peak values */
     int *peak_loc;      /* Output: peak locations */
{
```

```
  int i;
  float max = 0.0;     /* max in mermel segment */
  int max_loc;         /* location of max */
  float hull[WAVELEN];  /* convex hull of mermel segment */
  float diff;          /* difference between convex hull and mermel seg */
  float dip = 0.0;     /* max diff in mermel seg */
  int dip_loc;         /* location of max diff */

/* Find max location. */

  for (i = start; i <= end; i++){
    if (fabs(max) < fabs(rorarray[i])){
      max = rorarray[i];
      max_loc = i;
    }
  }

/* Make convex hull for left side. */

  hull[0] = rorarray[start];
  for (i = start+1; i <= max_loc; i++){
    if (fabs(rorarray[i]) > fabs(hull[i-1 - start]))
      hull[i - start] = rorarray[i];
    else
      hull[i - start] = hull[i-1 - start];
    diff = fabs(hull[i - start] - rorarray[i]);
    if (diff > dip){
      dip = diff;
      dip_loc = i;
    }
  }

/* Now make convex hull for right side. */

  hull[end - start] = rorarray[end];
  for (i = end-1; i > max_loc; i--){
    if (fabs(rorarray[i]) > fabs(hull[i+1 - start]))
      hull[i - start] = rorarray[i];
    else
      hull[i - start] = hull[i+1 - start];
    diff = fabs(hull[i - start] - rorarray[i]);
    if (diff > dip){
      dip = diff;
      dip_loc = i;
    }
  }

/* Recurse if dip > dip_thres; otherwise if peak > peak_thres record
   peak and peak location of mermel segment. */

  if (dip > dip_thres){    /* Recurse.*/
    npeak = mermel(rorarray, band, start, dip_loc, npeak, peak_thres,
   dip_thres, sgramlength, peak, peak_loc); /*Left side*/
    npeak = mermel(rorarray, band, dip_loc, end, npeak, peak_thres,
```

```
   dip_thres, sgramlength, peak, peak_loc);  /*Right side*/
  }
  else if (fabs(max) >= peak_thres){      /* Record peak and peak location. */
    peak[npeak] = max;
    peak_loc[npeak] = max_loc;
    npeak++;
  }
  return(npeak);
}

/***************************** LOCALIZE.C ******************************
   Localizes lm's using vector1 (coarse) and vector2
   (fine).  Record in vector3 (localized).  Format of vector*[] is one
   band's peaks right after the other, with the top of each band
   preceded by number of peaks in that band.  Picks the largest vec2
   pk in vicinity of vec1 pk, not the closest vec2 pk.
*/

#include "include.h"
#include "define.h"

void localize(argv, vector1, vector2, vector3)
     char *argv[];          /* In: command line args */
     double *vector1;       /* In: 1st pass detected peaks */
     double *vector2;       /* In: 2nd pass sensitized peaks */
     double *vector3;       /* Out: localized peaks */
{
  int i1, i2, i3, j1, j2, j3; /* Indices for vec1, vec2, vec3, rspctvly */
  int band;                 /* Band counter */
  float separation;         /* Time separation between vec1 and vec2 peaks */
  float max;                /* Biggest vec2 pk in vicinity of vec1 pk */
  int num_pk1, num_pk2, num_pk3;  /* Number peaks in vectors */

/*    In each band, and for each vector1 peak in each band, find vec2 pk
   that is closest to and has same sign as vec1 pk.  Record this vec2
   peak in vec3.
      If no vec2 peak within LMTIME of vec1 peak, skip vec1 peak.
      If 2 vec1 peaks are within LMTIME of the same vec2 peak,
   then use the vec1 pk that's closer to vec2.
*/

  i1 = i2 = i3 = 0;
  j3 = 0;

  for (band = 0; band < BANDS; band++){    /* For each band */
    num_pk1 = (int) vector1[2*i1];      /* Number pks in vec1 */
    num_pk2 = (int) vector2[2*i2];      /* Number pks in vec2 */
    num_pk3 = num_pk1;                  /* Initialize num_pk3 to num_pk1 */
    vector3[i3*2+1] = 0.0;       /* Fill in 0.0 in dB slot next to num_pk */
    j3 = i3 + 1;
    for (j1 = i1+1; j1 < i1+1+num_pk1; j1++){  /* For each vec1 pk in band */

      /* Go thru every vec2 pk and record biggest vec2 pk, within the
 vicinity of vec1 pk, in vec3 */
```

```
      max = 0.0;
      for (j2 = i2+1; j2 < i2+1+num_pk2; j2++){
separation = fabs(vector1[j1*2] - vector2[j2*2]);
if (separation <= LMTIME &&
    vector1[j1*2 + 1] * vector2[j2*2 + 1] > 0.0 &&
    fabs(vector2[j2*2 + 1]) > max){
  max = fabs(vector2[j2*2 + 1]);
  vector3[j3*2] = vector2[j2*2];
  vector3[j3*2+1] = vector2[j2*2+1];
}
      }

      /* If no matching vec2 pk, skip over that vec1 pk */
      if (max == 0)
num_pk3--;

      /* If 2 vec1 pks for 1 vec2 pk, record vec2 pk only once */
      else if (j3 != i3+1 && vector3[(j3-1)*2] == vector3[j3*2])
num_pk3--;

      else
j3++;

    }
    vector3[i3*2] = num_pk3;
    i1 += num_pk1 + 1;
    i2 += num_pk2 + 1;
    i3 += num_pk3 + 1;
  }
}

/***************************** ORDER_PK.C ******************************
   Time orders peaks in vec (arranged by band and then time)
   to a sequence of peaks (arranged by time only).  Fills in time, val,
   and band fields.
*/

#include "include.h"
#include "define.h"

void order_pk(vec, tot_bands, pk_seq, ptot_num_pk)
     double *vec;       /* In: vector of peaks ordered by band and time */
     int tot_bands;     /* In: total number of bands to be ordered */
     PEAK *pk_seq;      /* Out: peak sequence ordered by time only */
     int *ptot_num_pk;  /* Out: total number of peaks in pk_seq */
{
  void copy_pk();        /* Copies fields in one son_pk to another */
  int i, j, k;
  int k_so_far;          /* Growing index used during ordering of pk_seq */
  int band;              /* Band index, 0-5 */
  int num_pk[BANDS];      /* Number of peaks in each band */
  PEAK pk_seq2[BANDS*ROWS];   /* Temp pk sequence used during pk ordering */
  char tagstring2[BANDS*ROWS][21]; /* Space for tag field of pk_seq2 */
```

```
/* Allocate space for tag field of pk_seq2 */

  for (k = 0; k < tot_bands*ROWS; k++)
    pk_seq2[k].tag = tagstring2[k];

/* Find number of peaks for each band. */

  i = 0;
  *ptot_num_pk = 0;
  for (band = 0; band < tot_bands; band++){
    num_pk[band] = (int) vec[2*i];
    i += num_pk[band] + 1;
    *ptot_num_pk += num_pk[band];
  }

/* Copy B1 from vec into pk_seq. */

  for (k = 0; k < num_pk[0]; k++){
    pk_seq[k].time = vec[(k+1)*2];
    pk_seq[k].val = vec[(k+1)*2 + 1];
    pk_seq[k].band = 0;
  }
  k_so_far = num_pk[0];
  i = num_pk[0] + 1;

/* Insert rest of peaks, one at a time in a time-sequential manner,
   into pk_seq.  Use pk_seq2 to temporarily store sequence. */

  for (band = 1; band < tot_bands; band++){
    for (j = i+1; j < i+1 + num_pk[band]; j++){   /* j is vec[]'s index */
                          /* i is vec[]'s place holder at top of each band */

      /* Copy pk_seq into pk_seq2 up to t < t of current pk */
      k = 0;
      while (pk_seq[k].time <= vec[j*2] && k < k_so_far){
copy_pk(&pk_seq[k], &pk_seq2[k]);
k++;
      }

      /* Insert new peak into pk_seq2 */
      pk_seq2[k].time = vec[j*2];
      pk_seq2[k].val = vec[j*2 + 1];
      pk_seq2[k].band = band;
      k_so_far++;

      /* Copy rest of pk_seq into pk_seq2 */
      while (k < k_so_far){
copy_pk(&pk_seq[k], &pk_seq2[k+1]);
k++;
      }

      /* Copy pk_seq2 back to pk_seq */
      for (k = 0; k < k_so_far; k++)
copy_pk(&pk_seq2[k], &pk_seq[k]);
```

```
    }
    i += num_pk[band] + 1;
  }
}

/****************************** COPY_PK.C ********************************
   Copies peak structure 1 into peak structure 2.  Fields in a
   peak sequence are:  time, value, band, tag, lm, magnet, pair, stag,
   hpror, sslen, margin.
*/

#include "include.h"
#include "define.h"

void copy_pk(pk_struct1, pk_struct2)
     PEAK *pk_struct1, *pk_struct2;
{
  (*pk_struct2).time = (*pk_struct1).time;
  (*pk_struct2).val = (*pk_struct1).val;
  (*pk_struct2).band = (*pk_struct1).band;
  strcpy((*pk_struct2).tag, (*pk_struct1).tag);
  (*pk_struct2).lm = (*pk_struct1).lm;
  (*pk_struct2).magnet = (*pk_struct1).magnet;
  (*pk_struct2).pair = (*pk_struct1).pair;
  (*pk_struct2).pos = (*pk_struct1).pos;
  (*pk_struct2).stag = (*pk_struct1).stag;
  (*pk_struct2).hpror = (*pk_struct1).hpror;
  (*pk_struct2).sslen = (*pk_struct1).sslen;
  (*pk_struct2).marg = (*pk_struct1).marg;
  (*pk_struct2).error = (*pk_struct1).error;
  (*pk_struct2).oerror = (*pk_struct1).oerror;
  (*pk_struct2).tref = (*pk_struct1).tref;
  (*pk_struct2).sil = (*pk_struct1).sil;
  (*pk_struct2).burst = (*pk_struct1).burst;
}

/****************************** PGROUP.C *********************************
   Records pk_seq ordered by time and band into a *.peak# file.
*/

#include "include.h"
#include "define.h"

void pgroup(argv, pk_seq, tot_num_pk, out_ext, tgroup)
     char *argv[];      /* In: command line args */
     PEAK *pk_seq;      /* In: peak sequence */
     int tot_num_pk;    /* In: total number of peaks in pk_seq */
     char *out_ext;     /* In: output filename extension */
     float tgroup;      /* In: time tolerance to group peaks */
{
  FILE *fp1;               /* Output file pointer */
  char peakfilename[30];   /* Output file name */
  int i, j, k;             /* Indices for matrix (i,j) and pk_seq (k) */
```

```
  float mat[ROWS * (BANDS+1)];  /* Matrix of peaks ordered by time and band */
  int tot_num_rows;       /* Total number of rows used in matrix */

/* Initialize matrix to 0.0.  i is the row index; j is the column index. */

  for (i = 0; i < ROWS; i++)
    for (j = 0; j < BANDS+1; j++)
      mat[i*(BANDS+1) + j] = 0.0;

/* Sequence pk_seq in time and fill in matrix ordered by time and band.
   k is the index for pk_seq. */

  /* Put in first value of pk_seq into matrix */
  mat[0] = pk_seq[0].time;
  mat[pk_seq[0].band + 1] = pk_seq[0].val;

  /* Scroll thru rest of pk_seq */
  i = 0;
  for (k = 1; k < tot_num_pk; k++){
    if (pk_seq[k].time - mat[i*(BANDS+1)] > tgroup){
      i++;
      mat[i*(BANDS+1)] = pk_seq[k].time;
    }
    mat[i*(BANDS+1) + pk_seq[k].band + 1] = pk_seq[k].val;
  }
  tot_num_rows = i + 1;

/* Write matrix into *.peak# file. */

  strcpy(peakfilename, argv[1]);
  fp1 = fopen(strcat(peakfilename, out_ext), "w");

  for (i = 0; i < tot_num_rows; i++){
    fprintf(fp1, "%2.3f ", mat[i*(BANDS+1)]);
    for (j = 1; j < BANDS+1; j++)
      fprintf(fp1, "%4.0f ", mat[i*(BANDS+1) + j]);
    fprintf(fp1, "\n");
  }

/* Close shop. */

  fclose(fp1);
}

/****************************** VERIFY_G.C ********************************
   Verifies B1 landmarks:

   (1) Pairs up unpaired ones if there's a valid place to insert
   lm according to B1 energy; otherwise, chooses the left or right lm to
   delete based on a proximity measurement.  Does not insert pk's beyond
   an existing pk.  Makes sure first B1 pk is a +pk and last is a -pk.

   (2) Syllable requirement:  checks for sufficient energy and length between
   a pair of +/- lm's.
```

```
   In: vector, B1pk_seq.
   Out: vector (updated), B1pk_seq2.
*/

#include "include.h"
#include "define.h"

void verify_g(vector, eng, record_freq, start_time, sgramlength,
B1pk_seq, B1pk_seq2, tot_num_B1pk)
     double *vector;     /* In/out: vector of peaks */
     float *eng;         /* In: B1-6 energy array */
     double record_freq; /* In: record frequency of sgram file */
     double start_time;  /* In: start time of sgram file */
     int sgramlength;    /* In: length of sgram file */
     B1PEAK *B1pk_seq;   /* In: B1 peak sequence */
     B1PEAK *B1pk_seq2;  /* Out: B1 peak sequence, verified */
     int *tot_num_B1pk;  /* Out: total number of B1 peaks */
{
  void copy_B1pk();    /* Copies field of one B1pk_seq struct to another */
  int vocalic();       /* Determines if a given region is voc/son */
  int i, i2, i3;       /* Indices to vector and vector_tmp */
  int j;               /* Index to eng */
  int k;               /* Index to B1pk_seq */
  int k2;              /* Index to B1pk_seq2 */
  int band;            /* Band index */
  int num_pk;          /* Number of peaks in a band */
  int left, right;     /* Lt and rt indices to energy valley */
  float min, max;      /* Min and max B1 energy values in search space */
  int j_min, j_max;    /* Indices to min and max energy values */
  float thres;         /* Threshold calc'd from % (max - min) for lm insert */
  int pk_index;        /* Time index of current pk */
  int prev_pk_index;   /* Time index of previous pk */
  int next_pk_index;   /* Time index of next pk */
  int inserted_pk_index;  /* Time index of a previously inserted +pk */
  int left_k, right_k; /* Indices to B1pk_seq delimiting son/voc reg */
  double vector_tmp[ROWS*BANDS*2]; /* Temp vector to store vector values */

/* Put B1 peaks of vector into B1pk_seq, a vector of B1 peak structures.
   Initialize fields of B1pk_seq. */

  num_pk = (int)vector[0];        /* Number of peaks in B1 */
  for (k = 0; k < num_pk; k++){
    B1pk_seq[k].time = vector[(k + 1)*2];
    B1pk_seq[k].val = vector[(k + 1)*2 + 1];
    B1pk_seq[k].tag = "original";
  }

/* First pk:  if +pk, simply copy.  If -pk, then insert a +pk before it.
   Don't delete first pk at this stage.  Record temp results in B1pk_seq2. */

  k = k2 = 0;
  if (B1pk_seq[0].val > 0.0){        /* +pk: simply copy */
    copy_B1pk(&B1pk_seq[0], &B1pk_seq2[0]);
```

```
    k++; k2++;
  }
  else{                          /* -pk: insert +pk before it */
    next_pk_index = (int)((B1pk_seq[0].time - start_time) * record_freq);
    j = j_max = next_pk_index;     /* Search bkwd from -pk for max */
    max = eng[j_max];
    while (j > 0 && max - eng[j] < PEAK_THRES1){
      if (max < eng[j]){
max = eng[j];
j_max = j;
      }
      j--;
    }
    j_min = j;              /* Search bkwd from current j for min */
    min = eng[j_min];
    while (j > 0){
      if (min > eng[j]){
min = eng[j];
j_min = j;
      }
      j--;
    }
    thres = 0.5 * (max - min) + min;      /* 50% value */
    j = j_max;
    while (eng[j] > thres && j > j_min)
      j--;
    B1pk_seq2[k2].time = (float)j/record_freq + start_time;   /* Ins +pk */
    B1pk_seq2[k2].val = 1.0;
    B1pk_seq2[k2].tag = "inserted";
    k2++;
  }


/* Insert counterparts to unpaired B1 peaks. Search through B1pk_seq for
   -pk -> +pk pairs.  Delete unpaired peaks if energy diff not big enuf;
   use proximity rule to decide which lm on either side of missed lm to
   delete.  */

  while (k < num_pk){

/* If the final pk is a -pk, then simply copy from B1pk_seq to B1pk_seq2 */

    if (k == num_pk - 1 && B1pk_seq[k].val < 0.0){
      copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);
      k++; k2++;
    }

/* If the next 2 pk's are -pk's, look for appropriate place to insert
   a +pk in between.  If no appropriate place is found, decide which -pk
   to delete. */

    else if (B1pk_seq[k].val < 0.0 && B1pk_seq[k+1].val < 0.0){
      pk_index = (int)((B1pk_seq[k].time - start_time) * record_freq);
      next_pk_index = (int)((B1pk_seq[k+1].time - start_time) * record_freq);
```

```
      if (next_pk_index - 20 - pk_index > CONSON_T)
right = pk_index + CONSON_T;
      else
right = next_pk_index - 20;
      j = j_min = pk_index;          /* Search fwd from 1st -pk for min */
      min = eng[j_min];
      while (j < right && eng[j] - min < PEAK_THRES1){
if (min > eng[j]){
  min = eng[j];
  j_min = j;
}
j++;
      }
      j_max = j;               /* Search fwd from current j for max */
      max = eng[j_max];
      while (j < right){
if (max < eng[j]){
  max = eng[j];
  j_max = j;
}
j++;
      }
      if (max - min >= PEAK_THRES1){      /* Insert +pk if big enuf diff */
thres = 0.5 * (max - min) + min;       /* 50% value */
j = j_min;
while (eng[j] < thres && j < j_max)
  j++;
copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);    /* Copy -pk */
k++;  k2++;
B1pk_seq2[k2].time = (float)j/record_freq + start_time; /* Ins +pk */
B1pk_seq2[k2].val = 1.0;
B1pk_seq2[k2].tag = "inserted";
k2++;
      }
      else{              /* Else decide which -pk to delete */
j_max = next_pk_index;      /* Search bkwd from 2nd -pk for max */
max = eng[j_max];
if (next_pk_index - 20 - pk_index > CONSON_T)
  left = next_pk_index - CONSON_T;
else
  left = pk_index + 20;
while (j > left && max - eng[j] < PEAK_THRES1){
  if (max < eng[j]){
    max = eng[j];
    j_max = j;
  }
  j--;
}
j_min = j;                 /* Search bkwd from current j for min */
min = eng[j_min];
while (j > left){
  if (min > eng[j]){
    min = eng[j];
    j_min = j;
```

```
  }
  j--;
}
if (next_pk_index - j_min > DELETE_T){    /* Delete 1st -pk */
  copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);
  B1pk_seq2[k2].tag = "deleted";
  k++;  k2++;
}
else{                                     /* Delete 2nd -pk */
  copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);    /* Copy 1st -pk */
  k++;  k2++;
  copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);    /* Delete 2nd -pk */
  B1pk_seq2[k2].tag = "deleted";
  k++;  k2++;
  if (k < num_pk){
    copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);  /* Copy next +pk */
    k++;  k2++;
  }
}
      }
    }

/* If the next pk is a +pk, look for appropriate place to insert a -pk
   before it.  If no appropriate place is found, decide which +pk to delete. */

    else if (B1pk_seq[k].val > 0.0){
      prev_pk_index = (int)((B1pk_seq[k-1].time - start_time) * record_freq);
      pk_index = (int)((B1pk_seq[k].time - start_time) * record_freq);
      if (pk_index - prev_pk_index - 20 > CONSON_T)
left = pk_index - CONSON_T;
      else
left = prev_pk_index + 20;
      j = j_min = pk_index;         /* Search bkwd from +pk for min */
      min = eng[j_min];
      while (j > left && eng[j] - min < PEAK_THRES1){
if (min > eng[j]){
  min = eng[j];
  j_min = j;
}
j--;
      }
      j_max = j;            /* Search bkwd from current j for max */
      max = eng[j_max];
      while (j > left){
if (max < eng[j]){
  max = eng[j];
  j_max = j;
}
j--;
      }
      if (max - min >= PEAK_THRES1){     /* Insert +pk if big enuf diff */
thres = 0.5 * (max - min) + min;                    /* 50% value */
j = j_min;
while (eng[j] < thres && j > j_max)
```

```
  j--;
B1pk_seq2[k2].time = (float)j/record_freq + start_time;  /* Ins -pk */
B1pk_seq2[k2].val = -1.0;
B1pk_seq2[k2].tag = "inserted";
k2++;
copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);    /* Copy +pk */
k++;  k2++;
      }
      else{                              /* Else decide which +pk to delete */
j_max = prev_pk_index;          /* Search fwd from 1st +pk for max */
max = eng[j_max];
if (pk_index - 20 - prev_pk_index > CONSON_T)
  right = prev_pk_index + CONSON_T;
else
  right = pk_index - 20;
while (j < right && max - eng[j] < PEAK_THRES1){
  if (max < eng[j]){
    max = eng[j];
    j_max = j;
  }
  j++;
}
j_min = j;
min = eng[j_min];
while (j < right){
  if (min > eng[j]){
    min = eng[j];
    j_min = j;
  }
  j++;
}
if (j_min - prev_pk_index > DELETE_T){     /* Delete 2nd +pk */
  copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);
  B1pk_seq2[k2].tag = "deleted";
  k++;  k2++;
}
else{                                      /* Delete 1st +pk */
  B1pk_seq2[k2-1].tag = "deleted";         /* Delete 1st +pk */
  copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]); /* Copy 2nd +pk */
  k++;  k2++;
}
      }
    }

/* If the next 2 pk's (-/+) are too far apart, insert 2 pk's (+/-)
   in between. */

    else if (B1pk_seq[k+1].time - B1pk_seq[k].time >
     (float)CONSON_T/ record_freq * 5.0/3.0){
      pk_index = (int)((B1pk_seq[k].time - start_time) * record_freq);
      next_pk_index = (int)((B1pk_seq[k+1].time - start_time) * record_freq);

      /* Insert a +pk after the -pk */
      j = j_min = pk_index;              /* Search fwd from -pk for min */
```

```
      min = eng[j_min];
      right = pk_index + CONSON_T;
      while (j < right && eng[j] - min < PEAK_THRES1){
if (min > eng[j]){
  min = eng[j];
  j_min = j;
}
j++;
      }
      j_max = j;                /* Search fwd from current j for max */
      max = eng[j_max];
      while (j < right){
if (max < eng[j]){
  max = eng[j];
  j_max = j;
}
j++;
      }
      if (max - min >= PEAK_THRES1){    /* Insert +pk if big enuf diff */
thres = 0.5 * (max - min) + min;  /* 50% value */
j = j_min;
while (eng[j] < thres && j < j_max)
  j++;
inserted_pk_index = j;
copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);   /* Copy -pk */
k++; k2++;
B1pk_seq2[k2].time = (float)j/record_freq + start_time;  /* Ins +pk */
B1pk_seq2[k2].val = 1.0;
B1pk_seq2[k2].tag = "inserted";
k2++;
      }
      else{                                        /* Else delete -pk */
copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);
B1pk_seq2[k2].tag = "deleted";
k++;  k2++;
      }

      /* Insert a -pk before the +pk */
      j = j_min = next_pk_index;        /* Search bkwd from +pk for min */
      min = eng[next_pk_index];
      if (next_pk_index - inserted_pk_index - 20 > CONSON_T)
left = next_pk_index - CONSON_T;
      else
left = inserted_pk_index + 20;
      while (j > left && eng[j] - min < PEAK_THRES1){
if (min > eng[j]){
  min = eng[j];
  j_min = j;
}
j--;
      }
      j_max = j;                /* Search bkwd from current j for max */
      max = eng[j_max];
      while (j > left){
```

```
if (max < eng[j]){
  max = eng[j];
  j_max = j;
}
j--;
      }
      if (max - min >= PEAK_THRES1){   /* Insert +pk if big enuf diff */
thres = 0.5 * (max - min) + min;         /* 50% value */
j = j_min;
while (eng[j] < thres && j > j_max)
  j--;
B1pk_seq2[k2].time = (float)j/record_freq + start_time;  /* Ins -pk */
B1pk_seq2[k2].val = -1.0;
B1pk_seq2[k2].tag = "inserted";
k2++;
copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);   /* Copy +pk */
k++;  k2++;
      }
      else{                              /* Else delete +pk */
copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);
B1pk_seq2[k2].tag = "deleted";
k++;  k2++;
      }
    }

/* Otherwise copy -pk -> +pk pairs from B1pk_seq to B1pk_seq2. */

    else{
      copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);       /* Copy -pk */
      k++;  k2++;
      copy_B1pk(&B1pk_seq[k], &B1pk_seq2[k2]);       /* Copy +pk */
      k++;  k2++;
    }
  }

/* Last pk:  if it was a +pk, insert a -pk after it.  Don't delete last pk
   at this stage. */

  if (B1pk_seq2[k2-1].val > 0.0){
    pk_index = (int)((B1pk_seq2[k2-1].time - start_time) * record_freq);
    j = j_max = pk_index;          /* Search fwd from +pk for max */
    max = eng[j_max];
    while (j < sgramlength - 10 && max - eng[j] < PEAK_THRES1){
      if (eng[j] > max){
max = eng[j];
j_max = j;
      }
      j++;
    }
    j_min = j;             /* Search fwd from current j for min */
    min = eng[j_min];
    while (j < sgramlength - 10){
      if (eng[j] < min){
min = eng[j];
```

```
j_min = j;
      }
      j++;
    }
    thres = 0.5 * (max - min) + min;   /* 50% value */
    j = j_max;
    while (eng[j] > thres && j < j_min)
      j++;
    B1pk_seq2[k2].time = (float)j/record_freq + start_time;  /* Ins -pk */
    B1pk_seq2[k2].val = -1.0;
    B1pk_seq2[k2].tag = "inserted";
    k2++;
  }
  *tot_num_B1pk = k2;

/* Syllable requirement:  Check each undeleted +/- pair in B1pk_seq2 for
   sufficient B1 energy level and duration.  Delete pairs if insufficient
   energy.  */

  k = 0;
  while (k < *tot_num_B1pk){
    while (!strcmp("deleted", B1pk_seq2[k].tag))
      k++;
    left_k = k;
    k++;
    while (!strcmp("deleted", B1pk_seq2[k].tag))
      k++;
    right_k = k;
    k++;
    if (!vocalic(B1pk_seq2, left_k, right_k, eng, start_time, record_freq,
 sgramlength)){
      B1pk_seq2[left_k].tag = "deleted";
      B1pk_seq2[right_k].tag = "deleted";
    }
  }

/* Update vector with undeleted B1 lm's in B1pk_seq2. */

  /* Copy all of vector to vector_tmp */
  i = 0;
  for (band = 0; band < BANDS; band++){
    num_pk = (int) vector[2*i];
    for (i2 = i; i2 < i + num_pk + 1; i2++){
      vector_tmp[2*i2] = vector[2*i2];
      vector_tmp[2*i2+1] = vector[2*i2+1];
    }
    i = i2;
  }

  /* Copy original or inserted peaks of B1pk_seq2 to top of vector */
  i = 1;
  for (k = 0; k < *tot_num_B1pk; k++){
    if (strcmp("deleted", B1pk_seq2[k].tag)){
      vector[2*i] = B1pk_seq2[k].time;
```

```
      vector[2*i + 1] = B1pk_seq2[k].val;
      i++;
    }
  }
  vector[0] = (double)(i-1);     /* Num of B1 lm's */
  vector[1] = 0.0;
  i3 = i;

  /* Copy B2-6 pk's from vec_tmp to vec */
  i = (int) vector_tmp[0] + 1;          /* Skip vec_tmp's B1 pk's */
  for (band = 1; band < BANDS; band++){
    num_pk = (int) vector_tmp[2*i];
    for (i2 = i; i2 < i + num_pk + 1; i2++){
      vector[2*i3] = vector_tmp[2*i2];
      vector[2*i3 + 1] = vector_tmp[2*i2 + 1];
      i3++;
    }
    i = i2;
  }
}

/***************************** COPY_B1PK.C ********************************
   Copies B1 peak structure 1 into B1 peak structure 2.
*/

#include "include.h"
#include "define.h"

void copy_B1pk(B1pk_struct1, B1pk_struct2)
     B1PEAK *B1pk_struct1, *B1pk_struct2;
{
  (*B1pk_struct2).time = (*B1pk_struct1).time;
  (*B1pk_struct2).val = (*B1pk_struct1).val;
  strcpy((*B1pk_struct2).tag, (*B1pk_struct1).tag);
}

/***************************** VOCALIC.C **********************************
   Checks a given region to see if it's a voc/son region.  Uses B1 energy
   level and length sufficiency condition.  Energy level sufficiency measured
   wrt max B1 level in utt.   Returns 1 if voc/son; 0 otherwise.
*/

#include "include.h"
#include "define.h"

int vocalic(B1pk_seq, left_k, right_k, eng, start_time, record_freq,
    sgramlength)
     B1PEAK *B1pk_seq; /* In: B1 peak sequence */
     int left_k;       /* In: B1pk_seq index marking left side of region */
     int right_k;      /* In: B1pk_seq index marking right side of region */
     float *eng;       /* In: 6 band energy (dB) */
     double start_time;  /* In: start time of sgram file (s) */
     double record_freq; /* In: record freq of sgram file (s) */
     int sgramlength;    /* In: length of sgram file (ms) */
```

```
{
  int j;               /* Index to eng */
  int left;            /* Index to eng delimiting left side of region */
  int right;           /* Index to eng delimiting right side of region */
  int left_vocalic;    /* Left boundary of son/voc region */
  int right_vocalic;   /* Right boundary of son/voc region */
  int max_vocalic_len; /* Maximum voc/son length (ms) */
  int vocalic_len;     /* Voc/son length (ms) */
  float max;           /* Max energy level in utt (dB) */
  float vocalic_level; /* Min vocalic level for a syllable (dB) */

/* Find max B1 energy level in utt */

  max = 0.0;
  for (j = 0; j < sgramlength; j++)
    if (max < eng[j])
      max = eng[j];

  vocalic_level = max - VOCALIC_LEVEL;   /* A certain dB below max */

/* Compute left and right indices of region under consideration */

  left = (int)((B1pk_seq[left_k].time - start_time) * record_freq);
  right = (int)((B1pk_seq[right_k].time - start_time) * record_freq);
  max_vocalic_len = 0;
  j = left;

/* Check syllable level and duration sufficiency */

  while (j < right){
    while (eng[j] < vocalic_level && j < right)
      j++;
    left_vocalic = j;
    while (eng[j] >= vocalic_level && j < right)
      j++;
    right_vocalic = j;
    vocalic_len = right_vocalic - left_vocalic;
    if (vocalic_len > max_vocalic_len)
      max_vocalic_len = vocalic_len;
  }
  if (max_vocalic_len >= VOCALIC_LEN)
    return(1);
  else
    return(0);
}

/************************** WRITE_B1PK.C ********************************
   Writes out detailed info of B1pk_seq pks to *.B1 file.
*/

#include "include.h"
#include "define.h"

write_B1pk(argv, B1pk_seq, tot_num_B1pk)
```

```
    char **argv;          /* In: argument list */
    B1PEAK *B1pk_seq;     /* In: B1 peak sequence */
    int tot_num_B1pk;     /* In: total number B1 pks */
{
  extern char *progName;  /* Program name */
  FILE *fp1;              /* Output file pointer */
  char B1filename[30];    /* Output B1 file name */
  int k;

/* Open *.B1 file and write out detailed field info of B1 pks */

  strcpy(B1filename, argv[1]);
  fp1 = fopen(strcat(B1filename, ".B1"), "w");
  fprintf(fp1, "%s B1 file.\n", argv[1]);
  fprintf(fp1, "Contains detailed field info on B1 pks.\n\n");

  for (k = 0; k < tot_num_B1pk; k++){
    fprintf(fp1, "%2.3f %2.0f ", B1pk_seq[k].time, B1pk_seq[k].val);
    if (strcmp("original", B1pk_seq[k].tag))
      fprintf(fp1, "%s\n", B1pk_seq[k].tag);
    else
      fprintf(fp1, "\n");
  }

  fclose(fp1);
}

/******************************** TAG.C ***********************************
   Tags peaks in B1-5 as "obs", "pivot", "copivot", or "sstag".
   Outline of steps:

      1. Look only within non-obs regions, delimited by B1 pks.
      2. Remove from consideration B2-5 pks too close to B1 pk.
      3. Of the remaining B2-5 pks, search for biggest B2-5 pk in time until
         a B2-5 pk changes sign.  These biggest pks are labeled "pivot".
 Regions of one B2-5 sign are called blocks.
      4. Group neighboring B2-5 pks with this biggest pk, labeling them
         "copivot".  B2-5 pks not close enuf but not too far away are
 labeled "stag".
      5. If there are remaining B2-5 pks in the block after grouping,
         the biggest pk of the remaining pks is found and step 4 is repeated.
 Repeat step 5 until all B2-5 pks in the block are accounted for.
      6. Repeat steps 3-5 until all the blocks in the non-obs regions are
         accounted for.
      7. Repeat steps 2-6 until all the non-obs regions are searched.
      8. The pivots are the proposed s lm's.
      9. Pair up pivots with "g" lm's or other pivots, or leave unpaired.
     10. Fill in position field.
*/

#include "include.h"
#include "define.h"

void tag(argv, pk_seq, tot_num_pk)
```

```
    char *argv[];       /* In: argument list */
    PEAK *pk_seq;       /* In: tagged peak sequence */
    int tot_num_pk;     /* In: total number of peaks in pk_seq */
{
  void find_pivot();    /* Finds and tags pivots, copivots, and sstags */
  FILE *fp1;            /* Output file pointer */
  char tagfilename[30]; /* Output tag file name */
  int i, j, k;
  int k_last_obs;       /* Index of pk_seq to last B1 -lm */
  float left_obound, right_obound;  /* Lft & rt t bounds of obs reg (s) */
  int left_sbound, right_sbound;    /* Lft & rt t bounds of son reg (index) */

/* Initialize some fields:
     .tag = "untagged"
     .pos = 0
     .lm = 1 for B1 pks, and 0 otherwise
*/

  for (k = 0; k < tot_num_pk; k++){
    pk_seq[k].tag = "untagged";
    pk_seq[k].pos = 0;
    if (pk_seq[k].band == 0)
      pk_seq[k].lm = 1;
    else
      pk_seq[k].lm = 0;
  }

/* Indicate position of first B1 +lm */

  k = 0;
  while(pk_seq[k].band != 0)
    k++;
  pk_seq[k].pos = 1;

/* Find last B1 -pk, for later special utt-end processing */

  k = tot_num_pk - 1;

  while (pk_seq[k].band != 0)
    k--;
  k_last_obs = k;
  pk_seq[k].pos = 6;     /* Mark last B1 -lm */

/* Tag B1-5 peaks in and near obs region as "obs".  An obs plus
   neighboring ooze region is delimited by left_obound and
   right_obound (s).  Left and right obs ooze are different lengths.
   Last obs region has longer preceding ooze than medial obs
   regions.
*/
  k = 0;
  left_obound = 0.0;
  while (1){

    /* Specify right obs boundary (+B1 peak) */
```

```
    while (pk_seq[k].band != 0 && k < tot_num_pk)
      k++;
    if (k == tot_num_pk)
      right_obound = pk_seq[k-1].time + 0.001; /* .001 compensates for inacc */
    else if (k == tot_num_pk + 1)
      right_obound = pk_seq[k-2].time + 0.001;
    else
      right_obound = pk_seq[k].time + RT_OBS_OOZE;
    k++;

    /* Tag peaks in and near obs region with "obs". */
    j = 0;
    while (j < tot_num_pk){
      if (left_obound <= pk_seq[j].time && pk_seq[j].time <= right_obound)
pk_seq[j].tag = "obs";
      j++;
    }

    /* Break out of loop once all the peaks have been checked. */
    if (k >= tot_num_pk)
      break;

    /* Specify left obs boundary (-B1 peak) */
    while (pk_seq[k].band != 0)
      k++;
    if (k == k_last_obs){
      left_obound = pk_seq[k].time - LAST_OBS_OOZE;
      if (left_obound < right_obound)
left_obound = right_obound;
    }
    else
      left_obound = pk_seq[k].time - LT_OBS_OOZE;
    k++;
  }

/* Delimit untagged regions.  Recursively find pivots, copivots, and
   sstags in each untagged region.  */

  k = 0;
  while (k < tot_num_pk){
    if (!strcmp("untagged", pk_seq[k].tag)){
      left_sbound = k;
      k++;
      while (!strcmp("untagged", pk_seq[k].tag))
k++;
      right_sbound = k;
      find_pivot(left_sbound, right_sbound, pk_seq);
    }
    k++;
  }

/* Pair each pivot up with: (1) another pivot, if intervocalic,
   (2) a B1 lm, if a nasal-obs combo, or (3) leave unpaired, if
   other side was too gradual.  Fill in pair field and pos field of pk_seq. */
```

```
  for (k = 0; k < tot_num_pk; k++){
    if (!strcmp("pivot", pk_seq[k].tag)){
      if (pk_seq[k].val > 0.0){   /* +pivot: Search bkwd for B1 pk or -pivot */
j = k - 1;
while (strcmp("pivot", pk_seq[j].tag) && pk_seq[j].band != 0)
  j--;
if (!strcmp("pivot", pk_seq[j].tag) &&
    pk_seq[k].time - pk_seq[j].time <= IV_REG &&
    pk_seq[j].val < 0.0){
  pk_seq[k].pair = j;
  pk_seq[k].pos = 3;          /* Prd iv pivot */
}
else if (pk_seq[j].band == 0 &&
 pk_seq[k].time - pk_seq[j].time <= OBS_NAS_REG){
  pk_seq[k].pair = j;
  pk_seq[k].pos = 5;          /* obs->nas pivot */
}
else                          /* Unprd pivot */
  pk_seq[k].pos = 2;
      }
      else{                       /* -pivot: search fwd for B1 pk or +pivot */
j = k + 1;
while (strcmp("pivot", pk_seq[j].tag) && pk_seq[j].band != 0)
  j++;
if (!strcmp("pivot", pk_seq[j].tag) &&
    pk_seq[j].time - pk_seq[k].time <= IV_REG &&
    pk_seq[j].val > 0.0){
  pk_seq[k].pair = j;         /* Prd iv pivot */
  pk_seq[k].pos = 3;
}
else if (pk_seq[j].band == 0){
  if (pk_seq[j].pos == 6 &&    /* Last pivot */
      pk_seq[j].time - pk_seq[k].time <= LAST_REG){
    pk_seq[k].pair = j;
    pk_seq[k].pos = 6;
  }
  else if (pk_seq[j].pos == 0 &&   /* nas->obs pivot */
   pk_seq[j].time - pk_seq[k].time <= NAS_OBS_REG){
    pk_seq[k].pair = j;
    pk_seq[k].pos = 4;
  }
  else
    pk_seq[k].pos = 2;        /* Unprd pivot */
}
else                          /* Unprd pivot */
  pk_seq[k].pos = 2;
      }
    }
  }

/* For each pk in pk_seq, write descriptors out to *.tag file. */

  strcpy(tagfilename, argv[1]);
```

```
  fp1 = fopen(strcat(tagfilename, ".tag"), "w");

  for (k = 0; k < tot_num_pk; k++){
    fprintf(fp1, "%2.3f %2.0f %d %s ", pk_seq[k].time, pk_seq[k].val,
    pk_seq[k].band, pk_seq[k].tag);
    if (!strcmp("obs", pk_seq[k].tag))        /* obs */
      fprintf(fp1, "\n");
    else{
      if (!strcmp("pivot", pk_seq[k].tag)){   /* pivot */
if (pk_seq[k].pos != 2)       /* paired pivot */
  fprintf(fp1, "paired with %2.3f\n", pk_seq[pk_seq[k].pair].time);
else
  fprintf(fp1, "unpaired\n");         /* paired pivot */
      }
      else                                    /* sstag, copivot */
fprintf(fp1, "magnet at %2.3f\n", pk_seq[pk_seq[k].magnet].time);
    }
  }
  fclose(fp1);
}

/*************************** FIND_PIVOT.C ******************************
   Recursively find uni_sign blocks in the given voiced
   region. Tags pivots, copivots, and sstags.  Labels pivot magnet
   index of copivots and sstags.
*/

#include "include.h"
#include "define.h"

void find_pivot(left_sbound, right_sbound, pk_seq)
     int left_sbound, right_sbound;  /* In: pk_seq index limits for son reg
under consideration */
     PEAK *pk_seq;                   /* In/Out: Peak sequence */
{
  int uni_sign();     /* Checks for uni-sign and intervening B1 pks */
  int k;              /* Index of pk_seq */
  int k_pvt;          /* Index pointing to a pivot */
  float son_ooze;     /* Son ooze to left or right of pivot */
  int new_left_sbound, new_right_sbound;  /* Limits used in recursive call */

/* Search and tag pivot, copivots, and sstags. */

  /* Search for pivot and tag it. */
  k_pvt = left_sbound;
  for (k = left_sbound + 1; k < right_sbound; k++)
    if (fabs(pk_seq[k].val) > fabs(pk_seq[k_pvt].val) &&
pk_seq[k].val * pk_seq[k_pvt].val > 0.0 &&
k != k_pvt)
      k_pvt = k;
  pk_seq[k_pvt].tag = "pivot";

  /* Search for copivots around pivot and tag them.  Left son ooze is longer
     than right son ooze. */
```

```
  for (k = left_sbound; k < right_sbound; k++){
    if (pk_seq[k].time < pk_seq[k_pvt].time)
      son_ooze = LT_SON_OOZE;
    else
      son_ooze = RT_SON_OOZE;
    if (fabs(pk_seq[k].time - pk_seq[k_pvt].time) <= son_ooze &&
k != k_pvt &&
!strcmp("untagged", pk_seq[k].tag) &&
uni_sign(k, k_pvt, pk_seq)){
      pk_seq[k].tag = "copivot";
      pk_seq[k].magnet = k_pvt;
    }
  }

  /* Search for sstags around pivot and tag them. */
  for (k = left_sbound; k < right_sbound; k++)
    if (fabs(pk_seq[k].time - pk_seq[k_pvt].time) <= BLOCK &&
!strcmp("untagged", pk_seq[k].tag) &&
uni_sign(k, k_pvt, pk_seq)){
      pk_seq[k].tag = "sstag";
      pk_seq[k].magnet = k_pvt;
    }

/* Recurse if there are any untagged peaks left between left_sbound and
   right_sbound. */

  /* Search to left of pivot for any untagged peaks */
  for (k = left_sbound; k < k_pvt; k++)
    if (!strcmp("untagged", pk_seq[k].tag)){
      new_left_sbound = k;
      k++;
      while (!strcmp("untagged", pk_seq[k].tag))
k++;
      new_right_sbound = k;
      find_pivot(new_left_sbound, new_right_sbound, pk_seq);
    }

  /* Search to right of pivot for any untagged peaks */
  for (k = k_pvt + 1; k < right_sbound; k++)
    if (!strcmp("untagged", pk_seq[k].tag)){
      new_left_sbound = k;
      k++;
      while (!strcmp("untagged", pk_seq[k].tag))
k++;
      new_right_sbound = k;
      find_pivot(new_left_sbound, new_right_sbound, pk_seq);
    }
}

/***************************** UNI_SIGN.C ********************************
   Checks if the peaks in between and including 2 given points
   in pk_seq are all of the same sign; and checks if there are any inter-
   vening B1 peaks.  Returns 1 if all same sign and no B1 peaks;
   0 otherwise.
```

```
*/

#include "include.h"
#include "define.h"

int uni_sign(k1, k2, pk_seq)
     int k1, k2;     /* In: left and right indices for region to be examined */
     PEAK *pk_seq;   /* In: peak sequence */
{
  int k, k_tmp;      /* Pk_seq indices */

  if (k1 > k2){           /* Swap indices if k1 > k2 */
    k_tmp = k2;
    k2 = k1;
    k1 = k_tmp;
  }

  for (k = k1; k < k2; k++)    /* Check for any sign changes and B1 pks */
    if (pk_seq[k].val * pk_seq[k+1].val < 0.0 ||
pk_seq[k].band == 0)
      return(0);

  return(1);
}

/****************************** VERIFY_S.C ******************************
   Verifies an s lm candidate by checking for ss in middle of
   son region, abruptness at edges of region, and staggered peaks.  If
   a pivot passes all criteria except ss criterion, then label as a v lm.
   Input: pk_seq[], *.agram.
   Output: pk_seq (tagged with lm, ss info, pos).
*/

#include "include.h"
#include "define.h"

void verify_s(argv, pk_seq, tot_num_pk)
     char *argv[];       /* Input: argument list */
     PEAK *pk_seq;       /* Input: peak sequence */
                         /* Output: pk_seq lm field specified */
     int tot_num_pk;     /* Input: total number of peaks in pk_seq */
{
  float compare_spec();  /* Compares spectra for s-s determination */
  float abruptness();    /* Checks abruptness of proposed AC S lm's */
  int rules();           /* Puts proposed AC S lm's thru a series of tests */
  extern char *progName; /* Program name */
  char agramfilename[30];/* Input agram file name */
  struct header *inputHd;/* Input header */
  FILE *fpin;            /* Input file pointer */
  struct feaspec *i_rec; /* Input record */
  float *i_ptr;          /* Pointer to input record's re_spec_val */
  double sf;             /* Sampling frequency (Hz) */
  int recsize;           /* Record size, in frequency */
  double start_time;     /* Start time (s) */
```

```
  double record_freq;    /* Record frequency (1 kHz) */
  int sgramlength;       /* Length of sgram file */
  int i;                 /* Index of *.agram records */
  int j;                 /* Index in time in s-s regions */
  int k;                 /* Index for pk_seq */
  int fn1;               /* Frequency cutoff index for ss comparisons */
  int fn;                /* Frequency index */
  int already_checked;   /* 1 if an AC S lm has already been checked for ss */
  int left, right;       /* Times of peaks encompassing ss region */
  int middle;            /* Middle pt to start ss comparison (ref 0) */
  int left_ss, right_ss; /* Ss region boudaries (ref left) */
  int margin, left_margin, right_margin; /* Margins between ss bound and pk */
  int sslen;         /* Length of ss region */
  float l_error, r_error;   /* Error summed across f (dB)... */
  float error_limit;     /* ... and its limit (dB) */
  float l_oerror, r_oerror; /* Max error at one f index (dB) ... */
  float oerror_limit;    /* ... and its limit (dB) */
  float spec[200*20];    /* Agram values, 200 ms x 20 fcomps */
  int spec_i;            /* Index of spec[] */
  float hpror_val;       /* Value of hpror peak near peak */
  int pivot_index;       /* Time index of pivot occurrence */

/* Open *.agram file for reading and allocate space for input record. */

  strcpy(agramfilename, argv[1]);
  eopen(progName, strcat(agramfilename, ".agram"), "r",
FT_FEA, FEA_SPEC, &inputHd, &fpin);
  i_rec = allo_feaspec_rec(inputHd, FLOAT);
  i_ptr = i_rec->re_spec_val;
  sgramlength = inputHd->common.ndrec;
  recsize = get_fea_siz("re_spec_val", inputHd, (short *)NULL, (long **)NULL);
  sf = get_genhd_val("sf", inputHd, 1.0);
  record_freq = *get_genhd_d("record_freq", inputHd);
  start_time = *get_genhd_d("start_time", inputHd);

/* Compute upper frequency cutoff index for ss comparisons. */

  fn1 = (int)(COMP_F * 2.0 * (float)(recsize-1)/sf) + 1;

/* Delimit region in which to check for ss'ness.  Specify a middle point
   (ref wrt left index) at which to start checking. */

  i = 0;
  for (k = 0; k < tot_num_pk; k++){
    if (!strcmp("pivot", pk_seq[k].tag)){
      already_checked = 0;

      /* Specify left, right, and middle indices. */

      if (pk_seq[k].pos == 2){          /* Unprd pivot */
if (pk_seq[k].val < 0.0){
  left = (int)((pk_seq[k].time - start_time) * record_freq);
  right = left + UNPRD_REG;
  middle = UNPRD_TREF;
```

```
}
else{
  right = (int)((pk_seq[k].time - start_time) * record_freq);
  left = right - UNPRD_REG;
  middle = right - left - UNPRD_TREF;
}
error_limit = IV_ERROR;
oerror_limit = IV_OERROR;
      }
      else if (pk_seq[k].pos == 3 && pk_seq[k].val < 0.0){    /* iv -pivot */
left = (int)((pk_seq[k].time - start_time) * record_freq);
right = (int)((pk_seq[pk_seq[k].pair].time - start_time)*record_freq);
middle = (right - left)/2;
error_limit = IV_ERROR;
oerror_limit = IV_OERROR;
      }
      else if (pk_seq[k].pos == 4 || pk_seq[k].pos == 6){ /* nas->obs pivot */
left = (int)((pk_seq[k].time - start_time) * record_freq);
right = (int)((pk_seq[pk_seq[k].pair].time - start_time)*record_freq);
middle = NAS_OBS_TREF;
if (middle > (int)(0.5 * (float)(right - left)))
  middle = (right - left)/2;
error_limit = NAS_OBS_ERROR;
oerror_limit = NAS_OBS_OERROR;
      }
      else if (pk_seq[k].pos == 5){                    /* obs->nas pivot */
left = (int)((pk_seq[pk_seq[k].pair].time - start_time)*record_freq);
right = (int)((pk_seq[k].time - start_time) * record_freq);
middle = right - left - OBS_NAS_TREF;
if (middle < (int)(0.5 * (float)(right - left)))
  middle = (right - left)/2;
error_limit = OBS_NAS_ERROR;
oerror_limit = OBS_NAS_OERROR;
      }
      else
already_checked = 1;            /* iv +pivot */

/* Read agram values of "ss" region into spec[]. */

      if (!already_checked){

/* Skip through records in *.agram until left index is reached */
while (i < left){
  get_feaspec_rec(i_rec, inputHd, fpin);
  i++;
}

/* Read records from left to right indices, inclusive, into spec[] */
spec_i = 0;
while (i <= right){
  get_feaspec_rec(i_rec, inputHd, fpin);
  for (fn = 0; fn < fn1; fn++)
    spec[spec_i*fn1 + fn] = *(i_ptr + fn);
  i++;
```

```
  spec_i++;
}

/* Check for ss by comparing spectra in proposed region.  Use middle spectrum
   as reference. */

j = middle;
l_error = 0.0;
l_oerror = 0.0;
while (j > 0 && l_error < error_limit && l_oerror < oerror_limit){
  j--;
  l_error = compare_spec(spec, fn1, j, middle, &l_oerror);
}
left_ss = j + left;

j = middle;
r_error = 0.0;
r_oerror = 0.0;
while (j < right - left && r_error < error_limit &&
       r_oerror < oerror_limit){
  j++;
  r_error = compare_spec(spec, fn1, middle, j, &r_oerror);
}
right_ss = j + left;

left_margin = left_ss - left;
right_margin = right - right_ss;
sslen = right_ss - left_ss;

/* Check abruptness by looking for hpror peak in vicinity of pivot.  Then
   pass peak thru a series of tests to determine validity of a lm.  Tag
   nas->obs pvt lm's with .pos = 4. */

if (pk_seq[k].pos == 2){              /* Unprd +/- pvt */
  pivot_index = (int)((pk_seq[k].time - start_time) * record_freq);
  hpror_val = abruptness(argv, sgramlength, pivot_index,
 pk_seq[k].val);
  pk_seq[k].hpror = hpror_val;
  pk_seq[k].sslen = sslen;
  pk_seq[k].tref = (float)(middle + left)/record_freq + start_time;
  if (pk_seq[k].val < 0.0){           /* Unprd -pvt */
    pk_seq[k].marg = left_margin;
    pk_seq[k].error = l_error;
    pk_seq[k].oerror = l_oerror;
  }
  else{                               /* Unprd +pvt */
    pk_seq[k].marg = right_margin;
    pk_seq[k].error = r_error;
    pk_seq[k].oerror = r_oerror;
  }
  pk_seq[k].lm = rules(k, pk_seq, tot_num_pk);
}
else if (pk_seq[k].pos == 3){        /* IV pivot */
  pivot_index = (int)((pk_seq[k].time - start_time) * record_freq);
```

```
  hpror_val = abruptness(argv, sgramlength, pivot_index,
 pk_seq[k].val);
  pk_seq[k].hpror = hpror_val;
  pk_seq[k].sslen = sslen;
  pk_seq[k].tref = (float)(middle + left)/record_freq + start_time;
  pk_seq[k].marg = left_margin;
  pk_seq[pk_seq[k].pair].marg = right_margin;
  pk_seq[k].error = l_error;
  pk_seq[k].oerror = l_oerror;
  pk_seq[k].lm = rules(k, pk_seq, tot_num_pk);

  pivot_index = (int)((pk_seq[pk_seq[k].pair].time - start_time) *
      record_freq);
  hpror_val = abruptness(argv, sgramlength, pivot_index,
 pk_seq[pk_seq[k].pair].val);
  pk_seq[pk_seq[k].pair].hpror = hpror_val;
  pk_seq[pk_seq[k].pair].sslen = sslen;
  pk_seq[pk_seq[k].pair].tref =
    (float)(middle + left)/record_freq + start_time;
  pk_seq[pk_seq[k].pair].error = r_error;
  pk_seq[pk_seq[k].pair].oerror = r_oerror;
  pk_seq[pk_seq[k].pair].lm =
    rules(pk_seq[k].pair, pk_seq, tot_num_pk);
}
else if (pk_seq[k].pos == 4 ||        /* nas->obs pivot */
 pk_seq[k].pos == 6){
  pivot_index = (int)((pk_seq[k].time - start_time) * record_freq);
  hpror_val = abruptness(argv, sgramlength, pivot_index,
 pk_seq[k].val);
  pk_seq[k].hpror = hpror_val;
  pk_seq[k].sslen = sslen;
  pk_seq[k].tref = (float)(middle + left)/record_freq + start_time;
  pk_seq[k].marg = left_margin;
  pk_seq[k].error = l_error;
  pk_seq[k].oerror = l_oerror;
  pk_seq[k].lm = rules(k, pk_seq, tot_num_pk);
}
else{                                 /* obs->nas pivot */
  pivot_index = (int)((pk_seq[k].time - start_time) * record_freq);
  hpror_val = abruptness(argv, sgramlength, pivot_index,
  pk_seq[k].val);
  pk_seq[k].hpror = hpror_val;
  pk_seq[k].sslen = sslen;
  pk_seq[k].tref = (float)(middle + left)/record_freq + start_time;
  pk_seq[k].marg = right_margin;
  pk_seq[k].error = r_error;
  pk_seq[k].oerror = r_oerror;
  pk_seq[k].lm = rules(k, pk_seq, tot_num_pk);
}
      }
    }
  }

/* Close shop. */
```

```
  free_header(inputHd);
  free_fea_rec(i_rec);
  fclose(fpin);
}

/************************** COMPARE_SPEC.C ****************************
   Compares 2 spectral slices and returns the error, summed across
   frequency band of interest.
*/

#include "include.h"
#include "define.h"

float compare_spec(spec, fn1, j1, j2, oerror)
     float *spec;   /* In: spectral array from *.agram */
     int fn1;       /* In: frequency cutoff index */
     int j1, j2;    /* In: Indices in time of spectra to compare, j1 < j2 */
     float *oerror; /* Out: max error at one f index */
{
  int fn;                  /* Frequency index and cutoff */
  float error;             /* Total absolute error summed across f */
  float diff;              /* Error at one f index */

/* Compare the 2 spectra in f up to fn1.  Get the two spectral slices and
   compare them in f up to fn1.  Keep track of max f sample error in
   the f range searched.  */

  error = 0.0;
  *oerror = 0.0;
  for (fn = 0; fn < fn1; fn++){
    diff = fabs(spec[j1*fn1 + fn] - spec[j2*fn1 + fn]);
    error += diff;
    if (*oerror < diff)
      *oerror = diff;
  }
  return(error);
}

/************************** ABRUPTNESS.C ****************************
   Checks the abruptness of high pass energy change at a pivot.  Returns
   the max value of hpror near the pivot.
*/

#include "include.h"
#include "define.h"

float abruptness(argv, sgramlength, pivot_index, pivot_val)
     char *argv[];        /* In: argument list */
     int sgramlength;     /* In: length of sgram file */
     int pivot_index;     /* In: time index of pivot */
     float pivot_val;     /* In: pivot peak value */
{
  extern char *progName;  /* Program name */
```

```
  char hprorfilename[30]; /* Hpror file name */
  struct header *inputHd; /* Input file header */
  FILE *fpin;             /* Input file pointer */
  struct fea_data *i_rec; /* Input data record */
  float *i_ptr;           /* Input record pointer */
  int left, right;        /* Time indices delimiting region to check */
  int i;
  float max;              /* Maximum hpror value in delimited region (dB) */

/* Open *.hpror file for reading and allocate input record buffer */

  strcpy(hprorfilename, argv[1]);
  eopen(progName, strcat(hprorfilename, ".hpror"), "r",
FT_FEA, NONE, &inputHd, &fpin);
  i_rec = allo_fea_rec(inputHd);

/* Scroll through hpror until pivot vicinity is reached.  Then look for
   max hpror value. */

  left = pivot_index - HPROR_OOZE;
  right = pivot_index + HPROR_OOZE;

  for (i = 0; i < left; i++)    /* Skip values up to left index */
    get_fea_rec(i_rec, inputHd, fpin);

  max = 0.0;                    /* Find max in region around pivot */
  for (i = left; i <= right; i++){
    get_fea_rec(i_rec, inputHd, fpin);
    i_ptr = i_rec->f_data;
    if (fabs(max) < fabs(*i_ptr) && pivot_val * *i_ptr > 0.0)
      max = *i_ptr;
  }

/* Close shop. */

  free_fea_rec(i_rec);
  free_header(inputHd);
  fclose(fpin);
  return(max);
}

/****************************** RULES.C ********************************
   Checks whether a given pivot can be an s or v lm.  Factors
   considered:  abruptness, steady-state length, steady-state margin,
   and any associated sstags.  If not ss but abrupt and passes sstag
   test, then v lm.
*/

#include "include.h"
#include "define.h"

int rules(k1, pk_seq, tot_num_pk)
     int k1;              /* In: pk_seq index of pivot under examination */
     PEAK *pk_seq;        /* In: pk sequence */
```

```
     int tot_num_pk;      /* In: total number of peaks in pk_seq */
{
  int k;                  /* Index of pk_seq */
  int num_sstag;          /* Number of sstags associated with pivot */
  int ss_satisfied;       /* Binary:  1 if ss criterion satisfied */
  float ss_ratio;         /* Ss ratio for IV nasal verification */

/* Count number of sstags associated with pivot */

  num_sstag = 0;
  for (k = 0; k < tot_num_pk; k++)
    if (!strcmp("sstag", pk_seq[k].tag) && pk_seq[k].magnet == k1)
      num_sstag++;

/* Fill in stag field of pk_seq.  Stag field = 0 if there is no more
   than 1 sstag associated with pivot (2 sstags for utt-end -pivot);
   stag field = 1 otherwise. */

  pk_seq[k1].stag = 0;
  if (pk_seq[k1].pos == 6){      /* Utt-end -pivot */
    if (num_sstag > 2)
      pk_seq[k1].stag = 1;
  }
  else{                          /* Utt-medial pivot */
    if (num_sstag > 1)
      pk_seq[k1].stag = 1;
  }

/* Apply ss criterion, which depends on pivot position. */

  ss_satisfied = 0;
  if (pk_seq[k1].pos == 2 &&                    /* Unprd pivot */
      pk_seq[k1].sslen >= UNPRD_SSLEN &&
      pk_seq[k1].marg <= UNPRD_MARG)
    ss_satisfied = 1;
  else if (pk_seq[k1].pos == 3){                /* Iv pivot */
    ss_ratio = (float)pk_seq[k1].sslen / (float)(pk_seq[k1].marg +
       pk_seq[pk_seq[k1].pair].marg + pk_seq[k1].sslen);
    if (ss_ratio >= IV_SS_RATIO)
      ss_satisfied = 1;
  }
  else if ((pk_seq[k1].pos == 4 || pk_seq[k1].pos == 6) &&
   pk_seq[k1].sslen >= NAS_OBS_SSLEN &&  /* Nas->obs pivot */
   pk_seq[k1].marg <= NAS_OBS_MARG)
    ss_satisfied = 1;
  else if (pk_seq[k1].pos == 5 &&               /* Obs->nas pivot */
   pk_seq[k1].sslen >= OBS_NAS_SSLEN &&
   pk_seq[k1].marg <= OBS_NAS_MARG)
    ss_satisfied = 1;

/* Apply all the rules to determine if the pivot is an AC S lm */

  if (fabs(pk_seq[k1].hpror) > HPROR_LIMIT &&
      pk_seq[k1].stag == 0){
```

```
    if (ss_satisfied)
      return(2);      /* s lm */
    else if (pk_seq[k1].pos == 3 || pk_seq[k1].pos == 2) /* IV or unprd pvt */
      return(4);      /* v lm */
    else
      return(0);      /* Neither s nor v lm */
  }
  else
    return(0);        /* Neither s nor v lm */
}

/****************************** BURST.C ********************************
   Finds the voiceless stop and affricate releases and closures.
   Searches only in obstruent regions: looks for B2-5 peaks.  The pair
   field of opivot is the preceding -B1 lm, except when opivot is
   utt-init, in which case it's the succeeding +B1 lm.

   A silence interval is required before a release, and after a
   closure.  For +opivot, a silence interval is defined to be B2-6
   energy close to utt-init energy levels.  For -opivot, B1-6 are
   considered.

   In: pk_seq.
   Out: pk_seq (tagged with burst info).
*/

#include "include.h"
#include "define.h"

void burst(pk_seq, tot_num_pk, eng, sgramlength, start_time, record_freq)
     PEAK *pk_seq;      /* In/Out: Peak sequence in B1-5 */
     int tot_num_pk;    /* In: Total number of peaks in pk_seq */
     float *eng;        /* In: Energy vector of all bands (dB) */
     int sgramlength;   /* In: Length of sgram file */
     float start_time;  /* In: Start time of sgram file (s) */
     float record_freq; /* In: Record frequency of sgram file (Hz) */
{
  void find_opivot();   /* Recursively finds opivots, coopivots */
  int nopivotsinbetween(); /* Checks for (o)pivots in bet AC S -lm and burst */
  int j;                /* Index to eng[] */
  int k;                /* Index to pk_seq[] */
  float left_obound;    /* Left boundary of obs region + a margin (s) */
  float right_obound;   /* Right boundary of obs region + a margin (s) */
  int neg_B1pk_k;   /* Pk_seq index to preceding B1 -pk (+pk for utt-init) */
  int pos_B1pk_k;   /* Pk_seq index to succeeding B1 +pk (-pk for utt-final) */
  int opvt_time;        /* Time of opivot (index) */
  int left_search;      /* Time to search backwards from +opivot for sil */
  int right_search;     /* Time to search forward from -opivot for sil */
  int left_sil;         /* Left boundary of silence interval (index) */
  int right_sil;        /* Right boundary of silience interval (index) */
  int sil_length;       /* Length of silence interval (index) */
  int bursttime;        /* Time between opivot and preceding silence (ms) */
  int nas_obs_final;    /* Binary: 1 if there's a nasal in utt-final pos */
```

```
/* Determine if there's an utt-final nasal for later use with utt-final
   processing. */

  nas_obs_final = 0;
  k = tot_num_pk - 1;
  while (k > 0){
    if (pk_seq[k].lm == 2 && pk_seq[k].pos == 6){
      nas_obs_final = 1;
      break;
    }
    k--;
  }

/* Find any utt-init opivots and coopivots.  Specify left_obound and
   right_obound before first B1 +pk. The right B1 +pk index is passed
   as pair field to find_opivot(), (since there's no left B1 -pk). */

  k = 0;
  while (pk_seq[k].band != 0)
    k++;

  left_obound = 0.0;
  right_obound = pk_seq[k].time - VOT;
  if (right_obound < 0.0)
    right_obound = 0.0;
  find_opivot(left_obound, right_obound, pk_seq, tot_num_pk, k, k);

/* Recursively find opivots and coopivots within each utt-med and the
   utt-final obs region.  Pair field of +opivot is the preceding B1
   -pk's index (except for utt-init, which is then succeeding B1 +pk);
   pair field of -opivot is succeeding B1 +pk's index (except for
   utt-final, which is then preceding B1 -pk).  Magnet field of
   coopivot is opivot's index.  */

  while (pk_seq[k].pos != 6){    /* Not utt-final B1 -lm */
    k++;
    while (pk_seq[k].band != 0)
      k++;
    neg_B1pk_k = k;
    if (pk_seq[k].pos == 4 ||                    /* Nas->obs B1 -lm */
(pk_seq[k].pos == 6 && nas_obs_final))
      left_obound = pk_seq[k].time - NAS_OOZE;  /* Burst may be bef B1 -lm */
    else                                           /* Non nas->obs B1 -lm */
      left_obound = pk_seq[k].time;
    if (pk_seq[k].pos == 6){
      right_obound = (float)sgramlength/ record_freq + start_time;
      pos_B1pk_k = k;
    }
    else{
      k++;
      while (pk_seq[k].band != 0)
k++;
      right_obound = pk_seq[k].time - VOT;
      pos_B1pk_k = k;
```

```
    }
    find_opivot(left_obound, right_obound, pk_seq, tot_num_pk, neg_B1pk_k,
pos_B1pk_k);
  }

/* Tag nasal->opivot iv and utt-final +opivots for special treatment.
   In particular, don't require a silence interval for these opivots. */

  for (k = 0; k < tot_num_pk; k++){
    if (!strcmp("opivot", pk_seq[k].tag) && pk_seq[k].val > 0.0){
      j = k - 1;
      while (j > 0){
if (pk_seq[j].lm == 2 &&
    (pk_seq[j].pos == 4 || pk_seq[j].pos == 6) &&
    pk_seq[k].time - pk_seq[j].time < NAS_BURST &&
    nopivotsinbetween(j, k, pk_seq, tot_num_pk)){
  if (pk_seq[j].pos == 4)  /* Nas->obs opivot */
    pk_seq[k].pos = 4;
  else                     /* Utt-final nas->obs opivot */
    pk_seq[k].pos = 6;
  break;
}
j--;
      }
    }
  }

/* Look for silence intervals before each +opivot.  If it's a nasal->opivot
   opivot or an utt-init opivot, don't require any silence.  */

  for (k = 0; k < tot_num_pk; k++){
    if (!strcmp("opivot", pk_seq[k].tag) && pk_seq[k].val > 0.0){
      opvt_time = (int)((pk_seq[k].time - start_time) * record_freq);
      j = opvt_time;
      if (pk_seq[pk_seq[k].pair].pos != 1)    /* Not utt-init B1 +lm */
        left_search = (int)((pk_seq[pk_seq[k].pair].time - start_time) *
    record_freq) - 20;
      else{                                   /* Utt-init B1 + lm */
pk_seq[k].pos = 1;
left_search = 0;
      }
      while (pk_seq[k].lm == 0 && j > left_search){
while (j > left_search &&
       (eng[2*sgramlength + j] > SIL2 ||
eng[3*sgramlength + j] > SIL3 || eng[4*sgramlength + j]>SIL4 ||
eng[5*sgramlength + j] > SIL5))
  j--;
right_sil = j;
while (j > left_search &&
       (eng[2*sgramlength + j] < SIL2 &&
eng[3*sgramlength + j] < SIL3 && eng[4*sgramlength + j]<SIL4 &&
eng[5*sgramlength + j] < SIL5))
  j--;
left_sil = j;
```

```
sil_length = right_sil - left_sil;
bursttime = opvt_time - right_sil;
if (pk_seq[k].pos != 4 && pk_seq[k].pos != 6 && pk_seq[k].pos != 1){
                    /* Not nas->obs opivot or utt-init opivot */
  if (sil_length >= SILTIME && bursttime <= BURSTTIME)
    pk_seq[k].lm = 3;
}
else        /* Nas->obs and utt-init opivot: don't require silence */
  pk_seq[k].lm = 3;
pk_seq[k].sil = sil_length;
pk_seq[k].burst = bursttime;
      }
    }
  }

/* Look for silence interval after each -opivot.  Require silence in B1.
   Don't require any silence for utt-final opivot. */

  for (k = 0; k < tot_num_pk; k++){
    if (!strcmp("opivot", pk_seq[k].tag) && pk_seq[k].val < 0.0){
      opvt_time = (int)((pk_seq[k].time - start_time) * record_freq);
      j = opvt_time;
      if (pk_seq[pk_seq[k].pair].pos != 6)   /* Not utt-final B1 -lm */
right_search = (int)((pk_seq[pk_seq[k].pair].time - start_time) *
    record_freq) + 20;
      else{                                  /* Utt-final B1 -lm */
pk_seq[k].pos = 6;
right_search = sgramlength;
      }
      while (j < right_search &&
     (eng[2*sgramlength + j] > SIL2 ||
      eng[3*sgramlength + j] > SIL3 || eng[4*sgramlength + j] > SIL4 ||
      eng[5*sgramlength + j] > SIL5 || eng[j] > SIL0))
j++;
      left_sil = j;
      while (j < right_search &&
     (eng[2*sgramlength + j] < SIL2 &&
      eng[3*sgramlength + j] < SIL3 && eng[4*sgramlength + j] < SIL4 &&
      eng[5*sgramlength + j] < SIL5 && eng[j] < SIL0))
j++;
      right_sil = j;
      sil_length = right_sil - left_sil;
      bursttime = left_sil - opvt_time;
      if (pk_seq[k].pos != 6){            /* Not utt-final opivot */
if (sil_length >= SILTIME && bursttime <= BURSTTIME)
  pk_seq[k].lm = 3;
      }
      else                     /* Utt-final opivot: don't require silence */
pk_seq[k].lm = 3;
      pk_seq[k].sil = sil_length;
      pk_seq[k].burst = bursttime;
    }
  }
```

```
/* Remove any -opivot lm's if too close to B1 pks */

  for (k = 0; k < tot_num_pk; k++){
    if (pk_seq[k].val < 0.0 && pk_seq[k].lm == 3){
      for (j = 0; j < tot_num_pk; j++)
if (pk_seq[j].time >= pk_seq[k].time - VOT &&
    pk_seq[j].time <= pk_seq[k].time + VOT &&
    pk_seq[j].band == 0){
  pk_seq[k].lm = 0;
}
    }
  }
}

/************************** FIND_OPIVOT.C ******************************
   Recursively finds opivots within obs regions.  Tags coopivots
   around opivots.  Fills in pair field of +opivot with preceding B1
   -pk's index, if not utt-initial opivot.  Fills in pair field of
   -opivot with succeeding B1 +pk's index, if not utt-final opivot.
   Fills in magnet field of coopivot with index of opivot.
*/

#include "include.h"
#include "define.h"

void find_opivot(left_obound, right_obound, pk_seq, tot_num_pk, neg_B1pk_k,
 pos_B1pk_k)
     float left_obound;     /* In: Left boundary of obs region (s) */
     float right_obound;    /* In: Right boundary of obs region (s) */
     PEAK *pk_seq;          /* In/Out; Peak sequence */
     int tot_num_pk;        /* In: Total number of peaks in pk_seq */
     int neg_B1pk_k;        /* In: Preceding B1 -pk of obs region */
     int pos_B1pk_k;        /* In: Succeeding B1 +pk of obs region */
{
  int k;          /* Peak sequence index */
  int k_max;      /* Peak sequence index of +opivot in obs region */
  float max;      /* Max +opivot peak value in obs region (dB) */
  int k_min;      /* Peak sequence index of -opivot in obs region */
  float min;      /* Min -opivot peak value in obs region (dB) */

/* Look for +/- opivots in obs region */

  min = max = 0.0;
  for (k = 0; k < tot_num_pk; k++){
    if (pk_seq[k].time > left_obound && pk_seq[k].time < right_obound &&
!strcmp("obs", pk_seq[k].tag) && pk_seq[k].band != 0){
      if (pk_seq[k].val > max){
max = pk_seq[k].val;
k_max = k;
      }
      else if (pk_seq[k].val < min){
min = pk_seq[k].val;
k_min = k;
      }
```

```
    }
  }

/* If there's an opivot in the obs region, identify it and tag
   coopivots around it.  Pair field of +opivot is preceding B1 -pk's
   index (except for utt-init, which is then succeeding B1 +pk); pair
   field of -opivot is succeeding B1 +pk's index (except for
   utt-final, which is then preceding B1 -pk).  Magnet field of
   copivot is opivot's index. */

  if (max > 1.0){
    pk_seq[k_max].tag = "opivot";
    pk_seq[k_max].pair = neg_B1pk_k;
    for (k = 0; k < tot_num_pk; k++)
      if (fabs(pk_seq[k].time - pk_seq[k_max].time) <= BURST_OOZE &&
  pk_seq[k].time > left_obound && pk_seq[k].time < right_obound &&
  !strcmp("obs", pk_seq[k].tag) && pk_seq[k].val > 0.0 &&
  pk_seq[k].band != 0){
pk_seq[k].tag = "coopivot";
pk_seq[k].magnet = k_max;
      }
  }

  if (min < -1.0){
    pk_seq[k_min].tag = "opivot";
    pk_seq[k_min].pair = pos_B1pk_k;
    for (k = 0; k < tot_num_pk; k++)
      if (fabs(pk_seq[k].time - pk_seq[k_min].time) <= BURST_OOZE &&
  pk_seq[k].time > left_obound && pk_seq[k].time < right_obound &&
  !strcmp("obs", pk_seq[k].tag) && pk_seq[k].val < 0.0 &&
  pk_seq[k].band != 0){
pk_seq[k].tag = "coopivot";
pk_seq[k].magnet = k_min;
      }
  }

/* If there are remaining "obs" peaks in the obs region, recurse */

  for (k = 0; k < tot_num_pk; k++)
    if (pk_seq[k].time > left_obound && pk_seq[k].time < right_obound &&
!strcmp("obs", pk_seq[k].tag) && pk_seq[k].band != 0)
      find_opivot(left_obound, right_obound, pk_seq, tot_num_pk, neg_B1pk_k,
  pos_B1pk_k);
}

/************************ NOPIVOTSINBETWEEN.C ****************************
   Checks to see if there are any pivots or opivots in between an AC S -lm
   and an opivot.  Allows for only one B1 lm in between.  Returns 1 if all
   conditions satisfied; 0 otherwise.
*/

#include "include.h"
#include "define.h"
```

```
int nopivotsinbetween(k1, k2, pk_seq, tot_num_pk)
     int k1;          /* In: Index of pk_seq to s -lm */
     int k2;          /* In: Index of pk_seq to opivot */
     PEAK *pk_seq;    /* In: Peak sequence */
     int tot_num_pk; /* In: Total number of peaks in pk_seq */
{
  int k;                /* Index to pk_seq */
  int num_B1_lm;        /* Number of B1 lm's in between AC S lm and opivot */

  num_B1_lm = 0;

  for (k = k1 + 1; k < k2; k++){
    if (!strcmp("pivot", pk_seq[k].tag) ||
(!strcmp("opivot", pk_seq[k].tag) && pk_seq[k].val > 0.0))
      return(0);
    else if (pk_seq[k].band == 0){
      num_B1_lm++;
      if (num_B1_lm > 1)
return(0);
    }
  }

  return(1);
}

/***************************** WRITE_PK.C ******************************
   Writes out pk_seq lm's to *.lm label file.  Then writes
   out detailed field info of all B1 peaks and B2-5 (o)pivots to
   *.pivot file.
*/

#include "include.h"
#include "define.h"

write_pk(argv, pk_seq, tot_num_pk)
     char *argv[];         /* In: argument list */
     PEAK *pk_seq;         /* In: B1-4 pks */
     int tot_num_pk;       /* In: total number of peaks in pk_seq */
{
  extern char *progName;   /* Program name */
  FILE *fp1;               /* Output file pointer */
  char lmfilename[30];     /* Output lm label file name */
  char pivotfilename[30]; /* Output pivot file name */
  int k;                   /* Pk_seq index */

/* Open *.lm label file and write out AC lm's. */

  strcpy(lmfilename, argv[1]);
  fp1 = fopen(strcat(lmfilename, ".lm"), "w");
  fprintf(fp1, "signal %s\ntype 0\ncolor -1\n", argv[1]);
  fprintf(fp1, "comment Created using %s\n", progName);
  fprintf(fp1, "separator ;\nnfields 1\n#\n");

  for (k = 0; k < tot_num_pk; k++){
```

```
    if (pk_seq[k].lm == 1){
      if (pk_seq[k].val < 0.0)
fprintf(fp1, "%2.3f -1 -g\n", pk_seq[k].time);
      else
fprintf(fp1, "%2.3f -1 +g\n", pk_seq[k].time);
    }
    else if (pk_seq[k].lm == 2){
      if (pk_seq[k].val < 0.0)
fprintf(fp1, "%2.3f -1 -s\n", pk_seq[k].time);
      else
fprintf(fp1, "%2.3f -1 +s\n", pk_seq[k].time);
    }
    else if (pk_seq[k].lm == 3){
      if (pk_seq[k].val > 0.0)
fprintf(fp1, "%2.3f -1 +b\n", pk_seq[k].time);
      else
fprintf(fp1, "%2.3f -1 -b\n", pk_seq[k].time);
    }
    else if (pk_seq[k].lm == 4){
      if (pk_seq[k].val > 0.0)
fprintf(fp1, "%2.3f -1 +v\n", pk_seq[k].time);
      else
fprintf(fp1, "%2.3f -1 -v\n", pk_seq[k].time);
    }
  }
  fclose(fp1);

/* Open *.pivot file and write out detailed field info of B1 lm's and
   B2-5 pivots. */

  strcpy(pivotfilename, argv[1]);
  fp1 = fopen(strcat(pivotfilename, ".pivot"), "w");
  fprintf(fp1, "%s pivot file\n", argv[1]);
  fprintf(fp1, "Contains detailed field info on selected B1-5 pks\n\n");

  for (k = 0; k < tot_num_pk; k++){
    if (pk_seq[k].lm == 1)
      fprintf(fp1, "%2.3f 1 %2.0f %d\n",
      pk_seq[k].time, pk_seq[k].val, pk_seq[k].band);
    else if (!strcmp("pivot", pk_seq[k].tag)){
      if (pk_seq[k].pos != 2){          /* Paired pivot */
fprintf(fp1, "%2.3f %d %2.0f %d, pivot, stag = %d, ",
pk_seq[k].time, pk_seq[k].lm, pk_seq[k].val, pk_seq[k].band,
pk_seq[k].stag);
fprintf(fp1, "sslen = %d, margin = %d, hpror = %2.0f,\n",
pk_seq[k].sslen, pk_seq[k].marg, pk_seq[k].hpror);
fprintf(fp1, "    pair = %2.3f, error = %2.0f, oerror = %2.0f, ",
pk_seq[pk_seq[k].pair].time, pk_seq[k].error,
                pk_seq[k].oerror);
        fprintf(fp1, "tref = %2.3f\n", pk_seq[k].tref);
      }
      else{                             /* Unpaired pivot */
fprintf(fp1, "%2.3f %d %2.0f %d, pivot, stag = %d, ",
pk_seq[k].time, pk_seq[k].lm, pk_seq[k].val, pk_seq[k].band,
```

```
pk_seq[k].stag);
fprintf(fp1, "sslen = %d, margin = %d, hpror = %2.0f,\n",
pk_seq[k].sslen, pk_seq[k].marg, pk_seq[k].hpror);
fprintf(fp1, "    unpaired, error = %2.0f, oerror = %2.0f, ",
pk_seq[k].error, pk_seq[k].oerror);
fprintf(fp1, "tref = %2.3f\n", pk_seq[k].tref);
      }
    }
    else if (!strcmp("opivot", pk_seq[k].tag))
      fprintf(fp1, "%2.3f %d %2.0f %d, opivot, sil = %d, burst = %d\n",
      pk_seq[k].time, pk_seq[k].lm, pk_seq[k].val, pk_seq[k].band,
      pk_seq[k].sil, pk_seq[k].burst);
  }
  fclose(fp1);
}
```

C.2 HPROR.C: high-pass ROR

Hpror.c is the main program, which calls supporting functions. This section lists the following files, in order:

include.h
define.h
global.h
hpror.c
sgram_read.c
energy_sum.c
energy_write.c
ror.c
ror_write.c

```
/***************************** INCLUDE.H ******************************
   Included files.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <strings.h>

#include <esps/esps.h>
#include <esps/fea.h>
#include <esps/feaspec.h>
#include <esps/ftypes.h>

/***************************** DEFINE.H *******************************
   Parameter values used in hpror.c
*/
```

```
/* energy.c */

#define WAVELEN 10000  /* Max length of waveform  (ms) */
#define FCOMP 256      /* Number of freq comp = recsize, (512/2 + 1) */

#define F0 1000.0      /* Frequency band limits (Hz) */
#define F1 8000.0

#define TLEN 10       /* Smoothing interval (ms) */

/* ror.c */

#define DT 26  /* Time step for ROR in each band for localization (ms) */

/****************************** GLOBAL.H ******************************
   Global parameters in hpror.c .
*/

char *progName = "hpror";
int debug_level = 0;

/****************************** HPROR.C ******************************
   Finds the high-pass frequency ROR for abruptness verification in s lm
   detection.

   In:  a pre-emphasized sgram -- spectrogram (FEA_SPEC)
   Out: *.hpenergy -- high-pass energy (FEA)
        *.hpror -- ROR of hpenergy (FEA)

   Usage:  hpror  <fullname of sgram file> [ Fco ]
*/

#include "include.h"
#include "define.h"
#include "global.h"

main(argc, argv)
     int argc;
     char *argv[];
{
/*** Program variables ***/

/* Sgram header info */

  void sgram_read();      /* Reads sgram header info */
  int sgramlength;        /* Length of sgram file */
  int recsize;            /* Record size of sgram file */
  double sf;              /* Sampling frequency in sgram file */
  double record_freq;     /* Record frequency of sgram file */
  double start_time;      /* Start time of sgram file */

/* Input arguments */
```

```
  float fco;              /* Lower frequency cutoff (Hz) */

/* Energy smoothing. */

  void energy_sum();   /* Function to smooth and generate energy */
  void energy_write(); /* Function to write out *.hpenergy file */
  char firstname[30];  /* First name of sgram file */
  float sil;           /* Silence level in band */
  float eng[WAVELEN];  /* Energy vector in band */
  int i;

/* ROR. */
  void ror();          /* Function to calculate the ror of energy in band */
  void ror_write();    /* Function to write out *.hpror file */
  float rorwave[WAVELEN];  /* ROR vector in band */

/*** Program body ***/

/* Check argument list */

  if (argc < 2){
    printf("Usage:  hpror  <fullname of sgram file> [ Fco ]\n");
    exit(-1);
  }

  if (argc == 3)
    fco = atof(argv[2]);
  else
    fco = F0;

/* Read in header info from sgram file */

  sgram_read(argv, &sgramlength, &recsize, &sf, &record_freq, &start_time);

/* Smooth energy in each band.  Input: *.sgram.  Output: sil, eng[],
   *.hpenergy */

  i = 0;
  while (*(argv[1] + i) != '.')
    i++;
  strncpy(firstname, argv[1], i);

  energy_sum(argv, sgramlength, recsize, sf, fco, F1, TLEN, sil, eng);
  energy_write(argc, argv, firstname, ".hpenergy", sgramlength, eng);

/* Find ROR in each band.  Input:  eng[].  Output:  rorwave[], *.hpror */

  ror(eng, sgramlength, DT, rorwave);
  ror_write(argc, argv, firstname, ".hpenergy", ".hpror", rorwave,
    sgramlength);
}

/****************************** SGRAM_READ.C ******************************
  Reads in header info from sgram file.
```

```
*/

#include "include.h"
#include "define.h"

void sgram_read(argv, sgramlength, recsize, sf, record_freq, start_time)
     char *argv[];          /* In: input arguments */
     int *sgramlength;      /* Out: length of sgram file */
     int *recsize;          /* Out: record size of sgram file */
     double *sf;            /* Out: sampling frequency of sd file */
     double *record_freq;   /* Out: record frequency of sgram file */
     double *start_time;    /* Out: start time of sgram file */
{
  extern char *progName;      /* Main prog name */
  struct header *inputHd;     /* Sgram file header */
  FILE *fpin;                 /* Sgram file pointer */

/* Open sgram input file and read header info. */

  eopen(progName, argv[1], "r", FT_FEA, FEA_SPEC, &inputHd, &fpin);
  *sgramlength = inputHd->common.ndrec;
  *recsize = get_fea_siz("re_spec_val", inputHd, (short *)NULL, (long **)NULL);
  *sf = get_genhd_val("sf", inputHd, 1.0);
  *record_freq = *get_genhd_d("record_freq", inputHd);
  *start_time = *get_genhd_d("start_time", inputHd);

/* Close shop */

  free_header(inputHd);
  fclose(fpin);
}

/*************************** ENERGY_SUM.C ***************************
   Smooths and sums energy in specified band from spectrogram.
   Records results wrt beginning silence. tlen variable within band.
*/

#include "include.h"
#include "define.h"

void energy_sum(argv, sgramlength, recsize, sf, f1, f2, tlen, sil, eng)
     char *argv[];       /* In: command line arguments */
     int sgramlength;    /* In: length of sgram file */
     int recsize;        /* In: number of frequency components in a record */
     double sf;          /* In: sampling frequency of sd file */
     float f1, f2;       /* In: frequency band limits (Hz) */
     int tlen;           /* In: tlen */
     float sil;          /* Out: silence level */
     float *eng;         /* Out: energy vector */
{
  extern char *progName;    /* Main prog name */
  struct header *inputHd;   /* File headers */
  struct feaspec *i_rec;    /* Input record from sgram file (dB). */
  float *i_ptr;             /* Pointer to input record. */
```

```
  FILE *fpin;              /* File pointers */
  double *linear=(double *)calloc(FCOMP*TLEN, sizeof(double));/* smoothing*/
  double *t_av = (double *)calloc(FCOMP, sizeof(double));      /* smoothing*/
  float sum;          /* Sum of smoothed energy in freq band at that time. */
  int fn1, fn2;       /* Frequency limits for band (indices). */
  int i, j, k;        /* Generic indices. */
  int cirptr;         /* Circular pointer, for smoothing. */
  int start, end;     /* Start and end indices for processing sgram wave */

/*** Find silence levels from utt beginning. ***/

/* Open sgram input file and allocate record buffer */

  eopen(progName, argv[1], "r", FT_FEA, FEA_SPEC, &inputHd, &fpin);
  i_rec = allo_feaspec_rec(inputHd, FLOAT);

/* Convert frequency band limits (Hz) to indices. */

  fn1 = (int)(f1 * 2.0 * (float)FCOMP/sf);
  fn2 = (int)(f2 * 2.0 * (float)FCOMP/sf);

/* Read in first tlen worth of records from sgram, convert to linear,
   compute first time average. */

  for (j = fn1; j < fn2; j++)
    *(t_av + j) = 0.0;

  for (cirptr = 0; cirptr < tlen; cirptr++){
    get_feaspec_rec(i_rec, inputHd, fpin);
    i_ptr = i_rec->re_spec_val;
    for (j = fn1; j < fn2; j++){
      *(t_av + j) += (*(linear + cirptr*recsize + j) =
              exp(.23026* (*(i_ptr+j))));
    }
  }

/* Process first 40ms of waveform and collect silence level at 40ms mark. */

  cirptr = 0;
  for (i = 10; i <= 40; i++){
    if (i == 40){
      sum = 0.0;
      for (j = fn1; j < fn2; j++)
sum += *(t_av + j);
      sil = 10.0*log10(sum);
    }

/* Read in next record, average it in, update cirptr. */

    get_feaspec_rec(i_rec, inputHd, fpin);
    i_ptr = i_rec->re_spec_val;

    for (j = fn1; j < fn2; j++){
      *(t_av + j) -= *(linear + cirptr*recsize + j);
```

```
      *(t_av + j) += *(linear + cirptr*recsize +j)
                  = exp(.23026* *(i_ptr+j));
    }
    cirptr++; if (cirptr == tlen) cirptr = 0;
  }

/* Close shop */

  free_fea_rec(i_rec);
  free_header(inputHd);
  fclose(fpin);

/*** Now compute and record energy levels, subtracting off silence levels. ***/

/* Open sgram input file and allocate record buffer. */

  eopen(progName, argv[1], "r", FT_FEA, FEA_SPEC, &inputHd, &fpin);
  i_rec = allo_feaspec_rec(inputHd, FLOAT);

/* Read in first tlen worth of records, convert to linear,
   compute first time average. */

  for (j = fn1; j < fn2; j++)
    *(t_av + j) = 0.0;

  for (cirptr = 0; cirptr < tlen; cirptr++){
    get_feaspec_rec(i_rec, inputHd, fpin);
    i_ptr = i_rec->re_spec_val;
    for (j = fn1; j < fn2; j++){
      *(t_av + j) += (*(linear + cirptr*recsize + j) =
              exp(.23026* (*(i_ptr+j))));
    }
  }

/* Pad beginning of energy files with tlen/2 - 1 points long of their first
   values, subtracting off silence levels found above.  (The tlen/2 th point
   is filled in in the main loop.) */

  sum = 0.0;
  for (j = fn1; j < fn2; j++)
    sum += *(t_av + j);

  for (i = 0; i < tlen/2 - 1; i++)
    eng[i] = 10.0*log10(sum) - sil;

/* Process entire waveform. */

  cirptr = 0;
  start = tlen/2 - 1;
  end = sgramlength - tlen/2 - 1;

  for (i = start; i <= end; i++){

/* Find maximum energy in frequency band, and record relative values
```

```
   wrt utt-init silence level in eng vector.  */

    sum = 0.0;
    for (j = fn1; j < fn2; j++)
      sum += *(t_av + j);
    eng[i] = 10.0*log10(sum) - sil;

/* Read in next record, average it in, update cirptr. */

    get_feaspec_rec(i_rec, inputHd, fpin);
    i_ptr = i_rec->re_spec_val;

    for (j = fn1; j < fn2; j++){
      *(t_av + j) -= *(linear + cirptr*recsize + j);
      *(t_av + j) += (*(linear + cirptr*recsize +j) =
              exp(.23026* *(i_ptr+j)));
    }
    cirptr++; if (cirptr == tlen) cirptr = 0;
  }

/* Pad end of waveform with tlen/2 points of last value */

  for (i = end + 1; i < end + tlen/2 + 1; i++)
    eng[i] = eng[end];

/* Close shop */

  free_fea_rec(i_rec);
  free_header(inputHd);
  fclose(fpin);

  cfree(linear);
  cfree(t_av);
}

/***************************** ENERGY_WRITE.C ****************************
   Writes energy vectors in all bands to one energy (FEA) file.
*/

#include "include.h"
#include "define.h"

void energy_write(argc, argv, firstname, out_ext, sgramlength, eng)
     int argc;           /* In: number of command line args */
     char *argv[];       /* In: command line args */
     char *firstname;    /* In: firstname of sgram file */
     char *out_ext;      /* In: extension for output energy file */
     int sgramlength;    /* In: length of sgram file */
     float *eng;         /* In: energy vector */
{
  extern char *progName;                    /* Main prog name */
  char *iname;                              /* File name */
  char energyfilename[30];                  /* File name */
  struct header *inputHd, *outputHd;        /* File headers */
```

```
  FILE *fpin, *fpout;                          /* File pointers */
  struct fea_data *o_rec;                      /* Output record */
  float *o_ptr;                                /* Output record pointer */
  int i;

/* Open sgram input file and read header info, for purposes of creating
   output energy file header */

  iname = eopen(progName, argv[1], "r", FT_FEA, FEA_SPEC, &inputHd, &fpin);

/* Open energy output file, create output file header, and allocate output
   buffer. */

  strcpy(energyfilename, firstname);
  eopen(progName, strcat(energyfilename, out_ext),  "w",
NONE, NONE, &outputHd, &fpout);
  outputHd = new_header(FT_FEA);
  add_source_file(outputHd, iname, inputHd);
  outputHd->common.tag = NO;
  (void) strcpy(outputHd->common.prog, progName);
  outputHd->variable.refer = NULL;
  add_comment(outputHd, get_cmd_line(argc, argv));
  (void) add_fea_fld("hpenergy", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void)update_waves_gen(inputHd, outputHd, 1.0, 1.0);
  write_header(outputHd, fpout);

  o_rec = allo_fea_rec(outputHd);
  o_ptr = (float *) get_fea_ptr(o_rec, "hpenergy", outputHd);

/* Assign value to o_ptr and write out to energy file */

  for (i = 0; i < sgramlength; i++){
    *o_ptr = eng[i];
    put_fea_rec(o_rec, outputHd, fpout);
  }

/* Close shop */

  fclose(fpin);
  fclose(fpout);
}

/********************************* ROR.C ********************************
   Finds the rate of rise by taking overlapping time differences.
   Works on one band only.
*/

#include "include.h"
#include "define.h"

void ror(eng, sgramlength, dt, rorwave)
     float *eng;              /* In: energy wave of current band */
     int sgramlength;         /* In: length of sgram file */
```

```
     int dt;                  /* In: dt for the band */
     float *rorwave;          /* Out: array of ror values in the band */
{
  int i;
  int begin, end;      /* Begin and end indices for computing rorwave */

/* Pre zero pad rorwave with dt/2 pts to make it a symmetric diff. */

  for (i = 0; i < dt/2; i++)
    rorwave[i] = 0.0;

/* Process entire waveform. */

  begin = dt/2;
  end = sgramlength - dt/2;

  for (i = begin; i < end; i++)
    rorwave[i] = eng[i + dt/2] - eng[i - dt/2];

/* Post zero pad rorwave with dt/2 pts to make rorwave as long as
   energywave */

  for (i = end; i < end + dt/2; i++)
    rorwave[i] = 0.0;
}

/****************************** ROR_WRITE.C ******************************
   Writes ror vector to *.ror (FEA) file.
*/

#include "include.h"
#include "define.h"

void ror_write(argc, argv, firstname, in_ext, out_ext, rorwave, sgramlength)
     int argc;        /* In: number of command line arguments */
     char *argv[];    /* In: command line arguments */
     char *firstname; /* In: firstname of sgram file */
     char *in_ext;    /* In: extension for input energy file */
     char *out_ext;   /* In: extension for output ROR file */
     float *rorwave;  /* In: ROR vector */
     int sgramlength; /* In:  length of energy file */
{
  extern char *progName;                    /* Main prog name */
  char *iname;                              /* File name */
  char energyfilename[30], rorfilename[30]; /* File names */
  struct header *inputHd, *outputHd;        /* File headers */
  FILE *fpin, *fpout;                       /* File pointers */
  struct fea_data *o_rec;                   /* Output record */
  float *o_ptr;                             /* Output record pointer */
  int i;

/* Open energy input file and read header info, for purposes of creating
   output ror file header */
```

```
  strcpy(energyfilename, firstname);
  iname = eopen(progName, strcat(energyfilename, in_ext), "r",
FT_FEA, NONE, &inputHd, &fpin);

/* Open ror output file, create output file header, and allocate output
   buffer. */

  strcpy(rorfilename, firstname);
  eopen(progName, strcat(rorfilename, out_ext), "w",
NONE, NONE, &outputHd, &fpout);

  outputHd = new_header(FT_FEA);
  add_source_file(outputHd, iname, inputHd);
  outputHd->common.tag = NO;
  (void) strcpy(outputHd->common.prog, progName);
  outputHd->variable.refer = NULL;
  add_comment(outputHd, get_cmd_line(argc, argv));
  (void) add_fea_fld("hpror", 1L, 0, (long *) NULL, FLOAT, (char *)NULL,
     outputHd);
  (void)update_waves_gen(inputHd, outputHd, 1.0, 1.0);
  write_header(outputHd, fpout);

  o_rec = allo_fea_rec(outputHd);
  o_ptr = (float *) get_fea_ptr(o_rec, "hpror", outputHd);

/* Assign values to o_ptr and write out to *.ror */

  for (i = 0; i < sgramlength; i++){
    *o_ptr = rorwave[i];
    put_fea_rec(o_rec, outputHd, fpout);
  }

/* Close shop */

  fclose(fpin);
  fclose(fpout);
}
```

Bibliography

[Andre-Obrecht, 1988] Andre-Obrecht, R. (1988). A new statistical approach for the automatic segmentation of continuous speech signals. *IEEE Trans. ASSP*, 36:29–40.

[Aubert, 1989] Aubert, X. L. (1989). Fast look-ahead pruning strategies in continuous speech recognition. *Proc. ICASSP*, pages 659–662.

[Beckman, 1986] Beckman, M. E. (1986). *Stress and Non-stress Accent.* Foris Publications Holland, Dordrecht, The Netherlands.

[Byrne et al., 1994] Byrne, D., Dillon, H., Tran, K., Arlinger, S., Wilbraham, K., Cox, R., Hagerman, B., Kei, J., Lui, C., Kiessling, J., Kotby, M., Nasser, N., Kholy, W., Nakanishi, Y., Oyer, H., Powell, R., Stephens, D., Meredith, R., Sirimanna, T., Tavartkiladze, G., Frolenkov, G., Westerman, S., and Ludvigsen, C. (1994). An international comparison of long-term average speech spectra. *J. Acoust. Soc. Am.*, 96(4):2108–2120.

[Chang and Zue, 1994] Chang, J. and Zue, V. (1994). A study of speech recognition system robustness to microphone variations: experiments in phonetic classification. *Proc. ICSLP*, pages 995–998.

[Chigier and Brennan, 1988] Chigier, B. and Brennan, R. A. (1988). Broad class network generation using a combination of rules and statistics for speaker independent continuous speech. *Proc. ICASSP*, pages 449–452.

[Chigier and Leung, 1992] Chigier, B. and Leung, H. C. (1992). The effects of signal representations, phonetic classification techniques, and the telephone network. *Proc. ICSLP*, pages 97–100.

[Chomsky and Halle, 1968] Chomsky, N. and Halle, M. (1968). *The Sound Pattern of English.* Harper and Row, New York, NY.

[Clements, 1985] Clements, G. N. (1985). The geometry of phonological features. *Phonology Yearbook*, 2:223–250.

[Das et al., 1993] Das, S., Bakis, R., Nadas, A., Nahamoo, D., and Picheny, M. (1993). Influence of background noise and microphone on the performance of the IBM Tangora speech recognition system. *Proc. ICASSP*, 2:71–74.

[De Mori and Flammia, 1993] De Mori, R. and Flammia, G. (1993). Speaker-independent consonant classification in continuous speech with distinctive features and neural nets. *J. Acoust. Soc. Am.*, 94(6):3091–3103.

[Deng and Sun, 1994] Deng, L. and Sun, D. (1994). A statistical approach to automatic speech recognition using the atomic speech units constructed from overlapping articulatory features. *J. Acoust. Soc. Am.*, 95(5):2702–2719.

[Eide, 1993] Eide, E. (1993). *A linguistic feature representation of the speech waveform.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA. Dept. of EECS.

[Entropic Speech, 1994] Entropic Speech, I. (1994). *Entropic signal processing system.*

[Fisher et al., 1986] Fisher, W. M., Doddington, G. R., and Goudie-Marshall, K. M. (1986). The DARPA speech recognition research database: specifications and status. *DARPA Speech Recognition Workshop Proceedings*, pages 93–99.

[Flammia et al., 1992] Flammia, G., Dalsgaard, P., Andersen, O., and Lindberg, B. (1992). Segment based variable frame rate speech analysis and recognition using spectral variation function. *Proc. ICSLP*, pages 983–986.

[Fujimura, 1962] Fujimura, O. (1962). Analysis of nasal consonants. *J. Acoust. Soc. Am.*, 34(12):1865–1875.

[Furui, 1986] Furui, S. (1986). On the role of spectral transition for speech perception. *J. Acoust. Soc. Am.*, 80(4):1016–1025.

[Gish and Ng, 1993] Gish, H. and Ng, K. (1993). A segmental speech model with applications to word spotting. *Proc. ICASSP*, 2:447–450.

[Glass, 1988] Glass, J. R. (1988). *Finding acoustic regularities in speech: applications to phonetic recognition.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA.

[Glass and Zue, 1986] Glass, J. R. and Zue, V. W. (1986). Signal representation for acoustic segmentation. *Proc. First Australian Conference on Speech Science and Techonology*, pages 124–129.

[Glass and Zue, 1988] Glass, J. R. and Zue, V. W. (1988). Multi-level acoustic segmentation of continuous speech. *Proc. ICASSP*, pages 429–432.

[Halle and Stevens, 1991] Halle, M. and Stevens, K. N. (1991). Knowledge of language and the sounds of speech. In Sundberg, J., Nord, L., and Carlson, R., editors, *Symposium on Music, Language, and Brain*, pages 1–19. MacMillan, London.

[Jakobson et al., 1952] Jakobson, R., Fant, G., and Halle, M. (1952). Preliminaries to speech analysis. Technical Report 13, MIT Acoustics Laboratory.

[Jankowski et al., 1990] Jankowski, C., Kalyanswamy, A., Basson, S., and Spitz, J. (1990). NTIMIT: A phonetically balanced, continuous speech, telephone bandwidth speech database. *Proc. ICASSP*, pages 109–112.

[Johnson, 1994] Johnson, M. (1994). Automatic context-sensitive measurement of the acoustic correlates of distinctive features at landmarks. *Proc. ICSLP*, pages 1639–1642.

[Klatt, 1976] Klatt, D. H. (1976). Linguistic uses of segmental duration in English: Acoustic and perceptual evidence. *J. Acoust. Soc. Am.*, 59(5):1208–1221.

[Klatt, 1989] Klatt, D. H. (1989). Review of selected models of speech perception. In Marslen-Wilson, W., editor, *Lexical Representation and Process*, chapter 6, pages 169–226. MIT Press, Cambridge, MA.

[Kucera and Francis, 1967] Kucera, H. and Francis, W. N. (1967). *Computational Analysis of Present-Day American English.* Brown University Press, Providence, Rhode Island.

[Lee, 1989] Lee, K.-F. (1989). *Automatic speech recognition: The development of the SPHINX system.* Kluwer Academic Publishers, Norwell, MA.

[Lisker and Abramson, 1964] Lisker, L. and Abramson, A. S. (1964). A cross-language study of voicing in initial stops: Acoustical measurements. *Word*, 20:385–422.

[Marcus, 1993] Marcus, J. (1993). Phonetic recognition in a segment-based HMM. *Proc. ICASSP*, 2:479–482.

[Martens and Depuydt, 1991] Martens, J. P. and Depuydt, L. (1991). Broad phonetic classification and segmentation of continuous speech by means of neural networks and dynamic programming. *Speech Communication*, 10:81–90.

[Meng, 1991] Meng, H. (1991). The use of distinctive features for feature based speech recognition. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA. Dept. of EECS.

[Mermelstein, 1975] Mermelstein, P. (1975). Automatic segmentation of speech into syllabic units. *J. Acoust. Soc. Am.*, 58:880–883.

[Miller et al., 1951] Miller, G. A., Heise, G. A., and Lichten, W. (1951). The intelligibility of speech as a function of the context of the test materials. *J. Experimental Psychology*, 41(5):329–335.

[Miller and Nicely, 1955] Miller, G. A. and Nicely, P. E. (1955). An analysis of perceptual confusions among some English consonants. *J. Acoust. Soc. Am.*, 27:338–352.

[Ohde, 1994] Ohde, R. N. (1994). The developmental role of acoustic boundaries in speech perception. *J. Acoust. Soc. Am.*, 96(5, pt. 2):3307.

[Peterson and Barney, 1952] Peterson, G. E. and Barney, H. L. (1952). Control methods used in a study of the vowels. *J. Acoust. Soc. Am.*, 24(2):175–184.

[Phillips and Zue, 1992] Phillips, M. and Zue, V. (1992). Automatic discovery of acoustic measurements for phonetic classification. *Proc. ICSLP*, pages 795–798.

[Portnoff, 1973] Portnoff, M. R. (1973). A quasi-one-dimensional digital simulation for the time-varying vocal tract. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA. Dept. of EECS.

[Rabiner and Schafer, 1978] Rabiner, L. R. and Schafer, R. W. (1978). *Digital Processing of Speech Signals.* Prentice Hall, Englewood Cliffs, New Jersey.

[Sagey, 1986] Sagey, E. (1986). *The representation of features and relations in non-linear phonology.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA. Dept. of Linguistics and Philosophy.

[Stevens, 1985] Stevens, K. N. (1985). Evidence for the role of acoustic boundaries in the perception of speech sounds. In Fromkin, V., editor, *Phonetic Linguistics: Essays in Honor of Peter Ladefoged*, pages 243–255. Academic Press, New York.

[Stevens, 1986] Stevens, K. N. (1986). Models of phonetic recognition II: an approach to feature-based recognition. *Proceedings of Montreal symposium on speech recognition*, pages 67-68.

[Stevens, 1995a] Stevens, K. N. (1995a). Acoustic phonetics. Book in progress.

[Stevens, 1995b] Stevens, K. N. (1995b). Prosodic influences on glottal waveform: preliminary data. *J. Acoust. Soc. Am.*, 97(5). Presented at the 129th meeting of the Acoustical Society of America.

[Stevens et al., 1992a] Stevens, K. N., Blumstein, S. E., Glicksman, L., Burton, M., and Kurowski, K. (1992a). Acoustic and perceptual characteristics of voicing in fricatives and fricative clusters. *J. Acoust. Soc. Am.*, 91:2979–3000.

[Stevens et al., 1992b] Stevens, K. N., Manuel, S. Y., Shattuck-Hufnagel, S., and Liu, S. (1992b). Implementation of a model for lexical access based on features. *Proc. ICSLP*, pages 499–502.

[Turk, 1992] Turk, A. (1992). The American English flapping rule and the effect of stress on stop consonant durations. *Working Papers of the Cornell Phonetics Laboratory*, pages 103–134.

[Umeda, 1978] Umeda, N. (1978). Occurrence of glottal stops in fluent speech. *J. Acoust. Soc. Am.*, 64(1):88–94.

[van Beinum, 1994] van Beinum, F. J. K. (1994). What's in a schwa? *Phonetica*, 51:68–79.

[Weinstein et al., 1975] Weinstein, C. J., McCandless, S. S., Mondshein, L. F., and Zue, V. W. (1975). A system for acoustic-phonetic analysis of continuous speech. *IEEE Trans. ASSP*, 23:54–67.

[Windheuser, 1993] Windheuser, C. (1993). Phonetic features for spelled letter recognition with a time delay neural network. *Proc. Eurospeech*, pages 1489–1492.

[Withgott et al., 1987] Withgott, M., Bagley, S. C., Lyon, R. F., and Bush, M. (1987). Acoustic-phonetic segment classification and scale-space filtering. *Proc. ICASSP*, pages 860–863.

[Zue et al., 1990a] Zue, V., Seneff, S., and Glass, J. (1990a). Speech database development at MIT: TIMIT and beyond. *Speech Communication*, 9:351–356.

[Zue et al., 1990b] Zue, V. W., Glass, J. R., Goodine, D., Leung, H., Phillips, M., Polifroni, J., and Seneff, S. (1990b). Recent progress on the SUMMIT system. *Third DARPA Speech and Natural Language Workshop.*

[Zue and Laferriere, 1979] Zue, V. W. and Laferriere, M. (1979). Acoustic study on medial /t,d/ in American English. *J. Acoust. Soc. Am.*, 66:1039–1050.

[Zue and Seneff, 1990] Zue, V. W. and Seneff, S. (1990). Transcription and alignment of the TIMIT database. In Fujisaki, H., editor, *Recent Research Toward Advanced Man-Machine Interface through Spoken Language*, pages 464–473. Steering Group of the Priority Area Research, Tokyo.

THESIS PROCESSING SLIP

FIXED FIELD ill _____ name _____

index _____ biblio _____

▶ COPIES (Archives) Aero Dewey (Eng) Hum

Lindgren Music Rotch Science

TITLE VARIES ▶☐ _____

_____

_____

NAME VARIES ▶☐ _____

_____

IMPRINT (COPYRIGHT) _____

▶COLLATION 190 l (Eng. copy is p)

▶ADD. DEGREE _____ ▶ DEPT. _____

SUPERVISORS: _____

_____

_____

_____

_____

_____

_____

_____

_____

NOTES:

cat'r. date

page J136

▶DEPT E.E.

▶YEAR 1995 ▶DEGREE Ph.D.

▶NAME LIU, Sharlene Anne

_____