Optimizations for Automatic Speech Recognition Systems

Philippe Boucher
School of Computer Science
McGill University, Montreal

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of M.Sc in Computer Science.

Abstract

This thesis introduces an architecture for a generic automatic speech recognition (ASR) system which is easily adapted to specific tasks. It then expands the base system to include a Vector Quantizer (VQ) preprocessor and a speaker adaptation (SA) procedure. The VQ preprocessor reduces the amount of computation which must be done on multivariant gaussian mixtures contained within hidden Markov models (HMMs) in order to achieve a realtime speedup. The SA procedure increases the accuracy of the ASR system when the same speaker uses it for a period of time. Both improvements to the base architecture are then tested and evaluated using the TIMIT test set, an Air Traffic Control task set, and a single speaker test set for speaker adaptation.

Cette thèse présente une architecture pour un système pour la Reconnaissance Automatique de la Parole (RAP) qui peut être facilement adapté à une tâche spécifique. Ce système est alors agrandi pour comprendre un pré-processeur à base de quantification de vecteurs et une méthode d'adaptation à l'utilisateur. Le pré-processeur réduit le nombre de calculs nécessaire pour reconnaître la parole, ce qui améliore la vitesse du système. L'adaptation à l'utilisateur améliore l'exactitude du système. Ces deux modifications sont évaluées avec la base de données TIMIT, contenant des paroles de contrôleurs de trafic aérien, et une base de données d'un seul utilisateur pour vérifier l'adaptation.

Acknowledgments

I would like to thank Dr. Renato De Mori for his assistance and guidance, without which, this thesis wouldn't be. I would also like to thank Charles Snow and Michael Galler for their patient help, as well as everyone else in the Speech lab. Last but not least, the SOCS system staffers who do whatever it takes to keep the machines up and running no matter what and sometimes against great odds ;-)

Contents

List of Figures

Introduction

This thesis introduces an architecture for a generic speaker independent Automatic Speech Recognition (ASR) system called Roger. It is capable of being easily and quickly adapted to new tasks simply by updating a lexicon and a grammar description. This thesis then adds two improvements to the basic system, a vector quantizer preprocessor and speaker adaptation. The Vector Quantizer (VQ) preprocessor reduces the overall amount of computation required for recognition by predicting which gaussian distributions are likely to be useful for phoneme Hidden Markov Models and ignoring those gaussians that are not. The speaker adaptation system improves the overall accuracy of the recognizer over time for users who access the system for a period of time. This produces an ASR system which is both fast, accurate, and easily adapted to specific tasks. This document is divided into 7 chapters. The first 3 provide a basic overview of the theory behind standard speech recognition systems. The fourth introduces the Roger system, its interface and configuration. The fifth and sixth chapters describe vector quantization and mean adaptation. The seventh provides results showing the improvements obtained by adding vector quantization and speaker adaptation to Roger.

Chapter 1

Speech Generation

Before an Automatic Speech Recognition (ASR) system is described in detail, a brief overview is first presented on how speech is generated, which components and features are relevant to speech recognition, and how these can be represented. By understanding the structure of speech at the level at which it is generated, we can find proper ways to represent it and recognize it.

1.1 The Vocal Tract

Figure 1.1 is a simplified representation of the human vocal tract (adapted from [Gold89]). It consists of the region between the glottis and the lips. The nasal tract is the region which begins at the soft palate (velum) and ends at the nostrils. Lowering the soft palate links both tracts, thus enabling the speaker to produce nasal speech sounds. Speech is generated by first forcing air from the lungs through the windpipe. Tensing the vocal cords in the glottis to various degrees will cause them to vibrate during the flow of air, thus creating voiced sounds. The air flow is then forced through the larynx, oral cavity, and the nasal cavity. Transient sounds can be generated by a partial or complete closure of the vocal tract at various points.

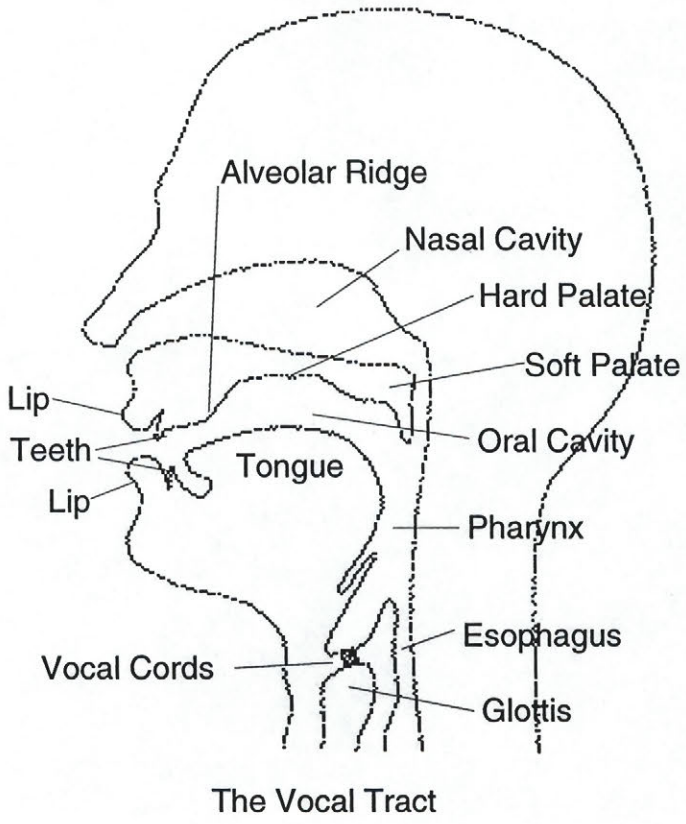

Figure 1.1: The human vocal tract [Gold89]

1.2 Phonemes

Phonemes are the smallest units of speech. They are the basic building blocks which make up the sounds of every spoken human language. Phonemes can be broken down into various groups based on their particular characteristics. These include the manner and place in which the phoneme is articulated as well as whether or not it is voiced. The two main groups are vowels and consonants, which differ in their mode of articulation. Consonants are generated when the airflow in the vocal tract is momentarily blocked or restricted, so that the noise is generated mostly when the closure is opened and the airflow is abruptly released. Vowels are generated by a continuous flow of air through the vocal tract and can be produced without interruption for significant lengths of time.

1.2.1 Consonants

In order to generate any kind of sound, the various components of the vocal tract must be set in some sort of configuration. The configuration of the vocal tract can be described using key positions, known as *places of articulation.* Altering these configurations will produce the following specific classes of sounds

1. *Labial* phonemes are produced by a partial or complete closure of the lips. If the sound involves both lips, then the sound is said to be *bilabial.* Sounds generated by using the lower lip and the upper teeth are called *labiodentals.* Examples of labial sounds are */p/*, */b/*, */v/*, and */f/*.

2. *Dental* sounds are produced by placing the tongue against or near the teeth. This produces sounds such as */s/* and */z/*. If the tongue is placed between the teeth, the sound is called *interdental.* The English sound */th/* is an interdental.

3. *Alveolar* sounds are generated by touching the tongue near the alveolar ridge which protrudes behind the upper front teeth. The English phonemes */t/* and

/d/ are alveolar sounds.

4. *Alveopalatal* and *Palatal* sounds are generated by placing the tongue on or near various positions in the roof of the mouth. An example phoneme is */sh/*.

5. *Velar* sounds are produced when the back of the tongue is placed on or near the velum. In English, such sounds tend to come at the beginning of a word, such as the */c/* in *cat*. If the lips are rounded during the production of the sound, it is called *labiovelar*. An example of this is the */w/* in *wet*.

6. *Uvular* sounds are made by placing the tongue on or near the uvula, the fleshy flap of tissue which hangs down from the velum. There are no such sounds in English, however, the European French */r/* is a uvular sound.

7. *Pharyngeal* sounds are produced by retracting the tongue or constricting the pharynx. These sounds are present in dialects of Arabic but not English.

8. *Glottal* sounds are made by closing or constricting the vocal chords and building up pressure behind them. An example of these is the */h/* sound at the beginning of such words as *heave* and *hog*.

The way in which air is pushed through the vocal tract will also affect the kind of consonant which is produced, depending on the current place of articulation being used by the speaker.

1. *Oral* vs *Nasal*. Sound can be generated by pushing air either solely through the oral cavity, or through both the oral and nasal cavities by lowering the velum.

2. *Stops* (sometimes referred to as plosives) are created by building up pressure behind a total constriction then releasing it. They can be voiced or unvoiced.

3. *Fricatives* are generated by a steady air flow through the vocal tract. This generates such unvoiced sounds as /s/ and /f/ or voiced sounds like /z/ and

/v/. Since fricatives require a partial constriction of the vocal tract in order to be generated, they are not classed as vowels.

4. *Affricates* are plosive/fricative pairs. They are usually modeled as single phonemes since the duration of the trailing fricative is relatively short.

5. *Liquids* (sometimes referred to as glides or semi-vowels) are produced by a constriction in the vocal tract smaller than that of vowels, but still large enough to not generate a fricative sound.

1.2.2 Vowels

Vowels are produced by opening the vocal tract and varying the position of the body of the tongue and the shape of the lips. There exists two major types of vowels in English, diphthongs and simple vowels. Diphthongs exhibit a distinct change in the sound of the phoneme over time. Simple vowels, however, are constant.

Vowel Articulation

Vowels are not articulated the same way as consonants. They are instead generated by positioning the tongue in a particular place and rounding the lips. If the tongue is placed high within the oral cavity, the vowels which are generated are classed as high. If the tongue is low within the oral cavity, the vowels generated are classed as low. If the tongue is in the middle, the vowel class is called mid. Similarly, the position of the tip of the tongue, front, central, or back also determines the vowel class. Vowels which are rounded are generated by placing the lips in an **O** shape. An */ow/* sound is therefore a mid-back rounded vowel. Nasal vowels are produced with a lowered velum so that air passes through the nasal cavity as well as the oral cavity

Figure 1.2 (adapted from [Dobr87]) shows the place (horizontal) and manner (vertical) of the articulation of the vowels of Canadian English. The trapezoid is a

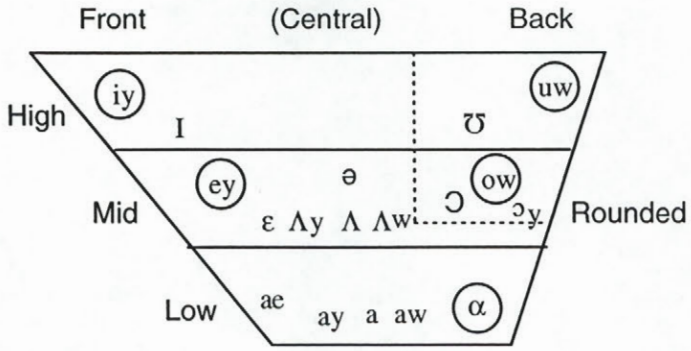

Figure 1.2: Vowel Articulation [Dobr87]

rough approximation of the space in which the tongue is able to move. Note that the lower the tongue goes in the oral cavity, the more restricted the space in which it can move. Vowels which are circled are lax, all others are tense. Lax vowels are sounds which are generated with little constriction of the tongue. Tense vowels require the musculature of the tongue to be tensed. Vowels within the dotted line are formed by a rounding of the lips during the vowel's production.

1.2.3 Phonetic features of consonants and vowels

Features are the units of phonological structure which make up the phonemes. They directly represent the phoneme and how it is generated. Each feature is independently controlled by the vocal tract. Linguists describe phonemes in term of a feature matrix. A + sign indicates that the feature is present in the particular phoneme, whereas a - shows the feature is not used. Certain features can be used to classify phonemes into groups called natural classes. Such classes will share one or more common feature. This method also permits comparisons between groups of phonemes. Allophones (variants of a standard phoneme) can be explained merely by changing the specification of one or more features for a given phoneme, not by substituting one phoneme for another. Current linguistic theory defines 24 distinct phonetic features which describe the whole of human phonological behaviour. The basic features of the

English language form a subset of those 24 and are arranged as follows:

1. Major Class features are used to classify phonemes into obstruents, vowels, glides, liquids and nasals. This set contains 3 specific features. *Consonantal* phonemes entail a major obstruction of the vocal tract. *Vocalic* phonemes are some vowels and liquids. *Sonorant* phonemes are the "singable" vowels, glides, liquids and nasals.

2. Laryngeal phonemes are those which are generated by changing the state of the larynx. *Voice* are those sounds which are actively voiced. *Spread glottis* are those consonants which are aspirated.

3. Place Features. These are *labial* , *round* , *coronal* , *anterior* , and *strident.*

4. *Dorsal* features represent the position of the body of the tongue. These include *high*, *back*, *low*, *tense* for certain vowels which are tensed instead of lax, and *reduced.*

5. The manner features are *Nasal*, where the velum is lowered, and *continuant*, where there is relatively free airflow through the oral cavity. The *lateral* feature includes only the varieties of */l/*. The *delayed release* feature is for all of the affricate consonants and nothing else.

Figure 1.3 contains a table adapted from [Dobr91] giving, for each consonant in Canadian English, the standard phonetic symbol, a word containing the sound, and a list of the phonetic features which create the particular sound. The feature groups are organized as follows: The *Major Class* features contain the consonantal, sonorant, and vocalic features. The second set, *Laryngeal* , contains the voiced, constricted glottis, and spread glottis features. The third set, *Place*, consists of the labial, round, coronal, anterior, and strident features. The fourth set are the *Dorsal* features: high and back. The fifth, *Manner*, are the nasal, continuant, lateral, and delayed release features.

Symbol	Word	Major	Laryngeal	Place	Dorsal	Manner
p	spit	+−−	−−−	+−−+−	−−	−−−−
p^h	pit	+−−	−−+	+−−+−	−−	−−−−
t^h	tick	+−−	−−+	−−++−	−−	−−−−
t	stuck	+−−	−−−	−−++−	−−	−−−−
k^h	keep	+−−	−−+	−−−−−	++	−−−−
k	skip	+−−	−−−	−−−−−	++	−−−−
č	chip	+−−	−−−	−−+−+	−−	−−−+
ǰ	judge	+−−	+−−	−−+−+	−−	−−−+
b	bib	+−−	+−−	+−−+−	−−	−−−−
d	dip	+−−	+−−	−−++−	−−	−−−−
g	get	+−−	+−−	−−−−−	++	−−−−
f	fit	+−−	−−−	+−−+−	−−	−+−−
v	vat	+−−	+−−	+−−+−	−−	−+−−
θ	thick	+−−	−−−	−−++−	−−	−+−−
∂	though	+−−	+−−	−−++−	−−	−+−−
s	sip	+−−	−−−	−−+++	−−	−+−−
z	zap	+−−	+−−	−−+++	−−	−+−−
š	ship	+−−	−−−	−−+−+	−−	−+−−
ž	azure	+−−	+−−	−−+−+	−−	−+−−
h	hat	−−−	−−+	++−−−	++	−+−−
y	yet	−+−	+−−	−−−−−	+−	−+−−
w	witch	−+−	+−−	++−−−	++	−+−−
w	which	−+−	−−−	++−−−	++	−+−−
l	leaf	+++	+−−	−−++−	−−	−++−
m	moat	++−	+−−	+−−+−	−−	+−−−
r	rent	+++	+−−	−−++−	−−	−+−−
?	(Glottal stop)	−−−	−+−	−−−−−	−−	−−−−
n	note	++−	+−−	−−++−	−−	+−−−
η	sing	++−	+−−	−−−−−	++	+−−−

Figure 1.3: Standard Canadian English consonants

Symbol	Word	Major	Laryngeal	Place	Dorsal	manner
i	fee	-++	+	-	+--+-	+
I	fit	-++	+	-	+--+-	+
e^y	fate	-++	+	-	---+-	+
ε	let	-++	+	-	-----	+
ae	bat	-++	+	-	--+--	+
u	boot	-++	+	-	+++-+	+
U	book	-++	+	-	+++--	+
o^w	note	-++	+	-	+-+-+	+
a	saw	-++	+	-	-+++-	+
Δ	shut	-++	+	-	-+---	+
e	roses	-++	+	-	-+--+	+

Figure 1.4: Standard Canadian English vowels

Figure 1.4 shows the features of the English vowels. The features of the *Major* class are the same as those of figure 1.3. The *Laryngeal class* contains only the voice feature, and the *Manner* class has only the continuant feature. The *Place* class is represented by the features high, back, low, tense, and reduced.

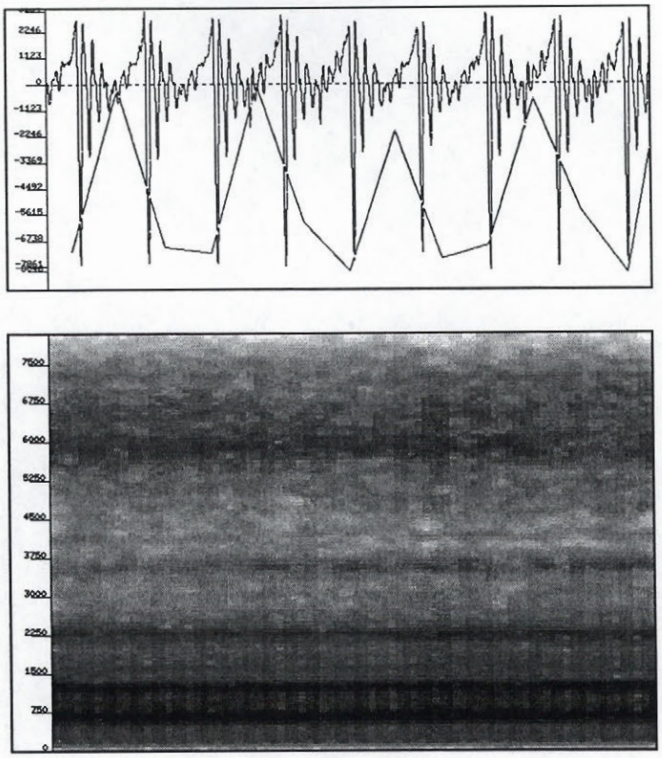

Figure 1.5: Waveform and spectrogram for /a/

1.3 Acoustic Features

The following represents examples of particular kinds of phonemes previously discussed. The spectrograms and waveforms contain features which are particular to each phonetic class. The curve which accompanies the waveform plot represents the amount of energy present in the speech sample (see Chapter 2). The spectrograms are scaled from 0Hz to 8000Hz and the waveforms are scaled from -32768 to 32768.

First, the spectrogram for a vowel will be shown, in this case, */a/*. As can be seen above, vowels are characterized by constant straight lines in the spectrogram, called *formants*. All vowels have formant structure. The positions of the formants and the distances between them will vary according to the phoneme itself and the voice pitch of the speaker.

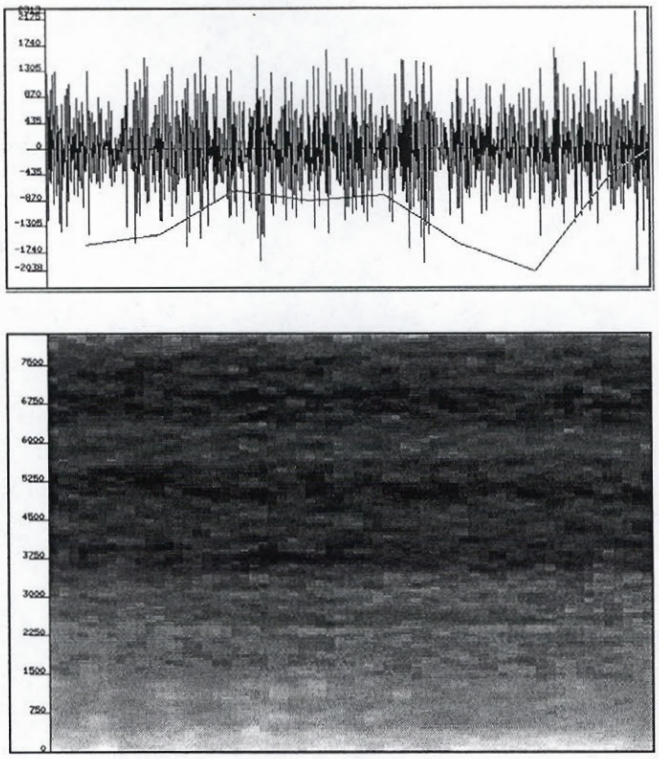

Figure 1.6: Waveform and spectrogram for /s/

We now look at the spectrograms for two fricatives. Figure 1.6 represents a spectrogram for the phoneme */s/*. Note that it is characterized by a broad band noise in the high frequency range only. This shows up as the dark layer in the upper area of the spectrogram.

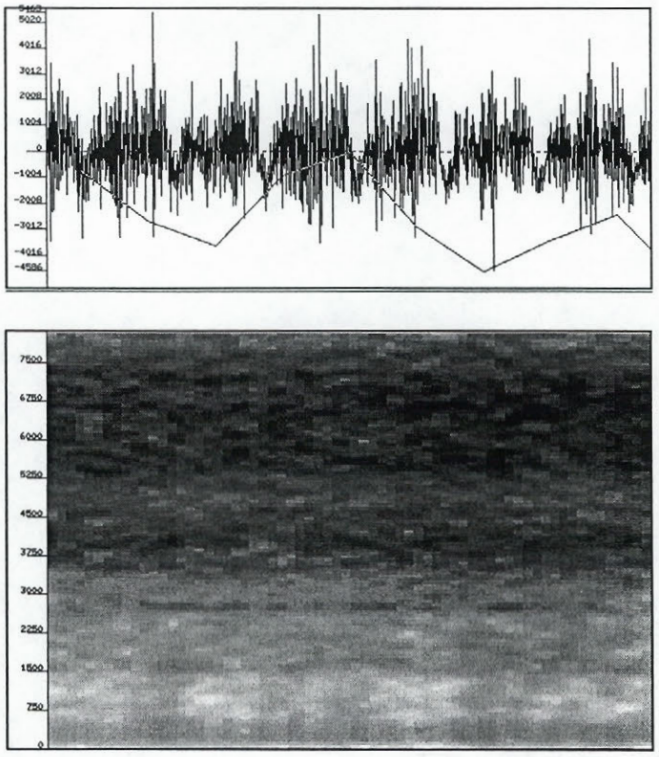

Figure 1.7: Waveform and spectrogram for /z/

This spectrogram represents the phoneme */z/*. Note that, like */s/*, it is characterized by a broad band high frequency noise component. However, you will see a small energy band underneath. This is caused by voicing and constitutes the only difference between */s/* and */z/*. Because of this, these phonemes are referred to as *cognates*.

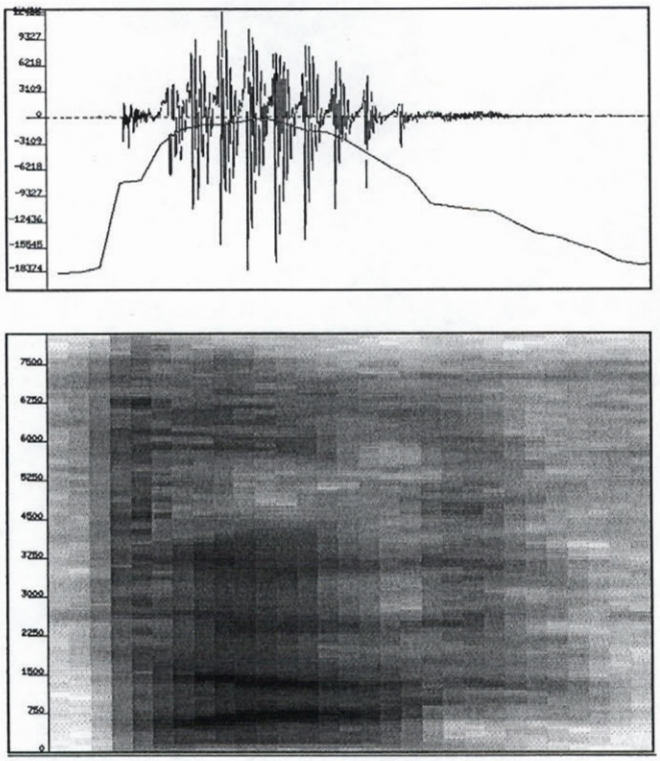

Figure 1.8: Waveform and spectrogram for /ta/

Figures 1.8 and 1.9 are examples of plosives. The above images for */ta/* first begin with a segment of silence, followed by a burst of energy for */t/* which generates two small bands of high frequency sound. The vowel sound, */a/*, produces the two formants, f_1 and f_2 near the bottom of the spectrogram.

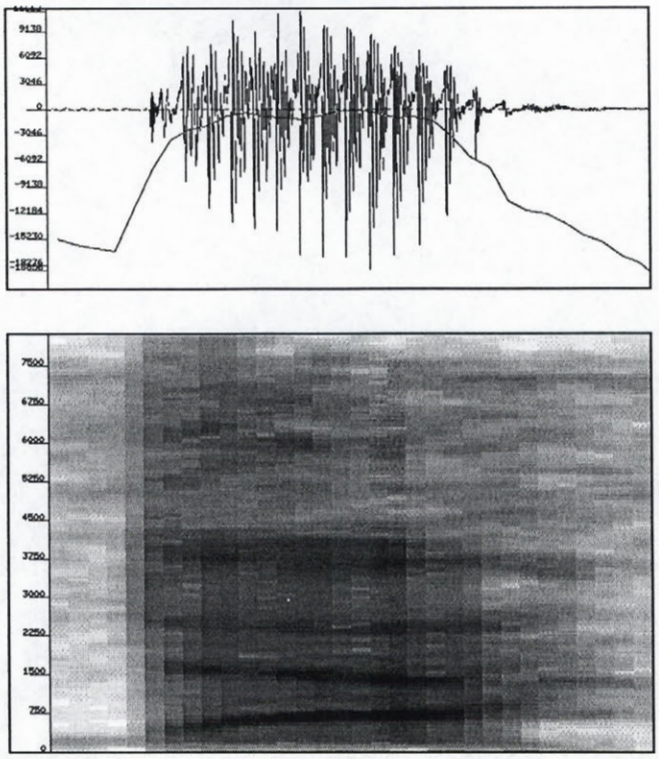

Figure 1.9: Waveform and spectrogram for /da/

The */da/* sound is similar to */ta/*, these plosives are cognates and differ only in that */d/* component is voiced.

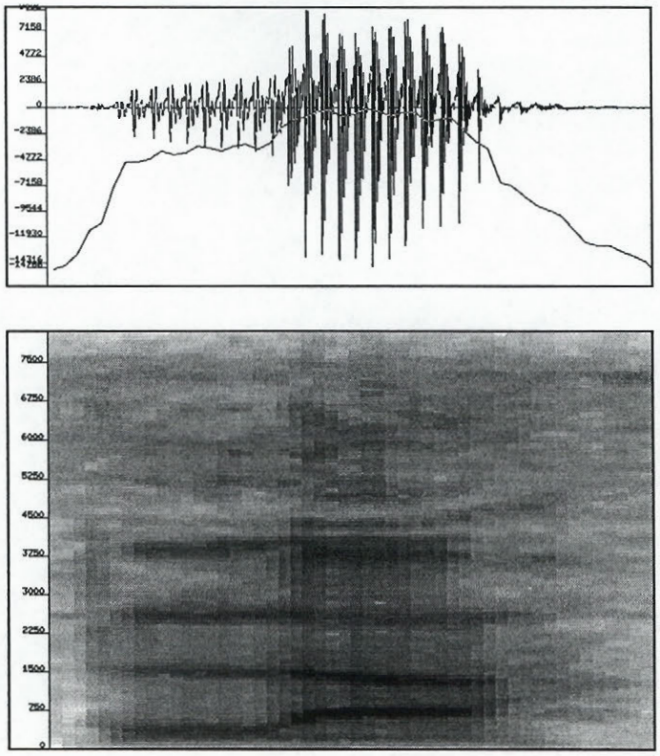

Figure 1.10: Waveform and spectrogram for /la/

The spectrogram for */la/* in Figure 1.10 demonstrates two continuous sounds, *l* and *a* and the transition from one set of formants to another. Note the transitions for the first two formants.

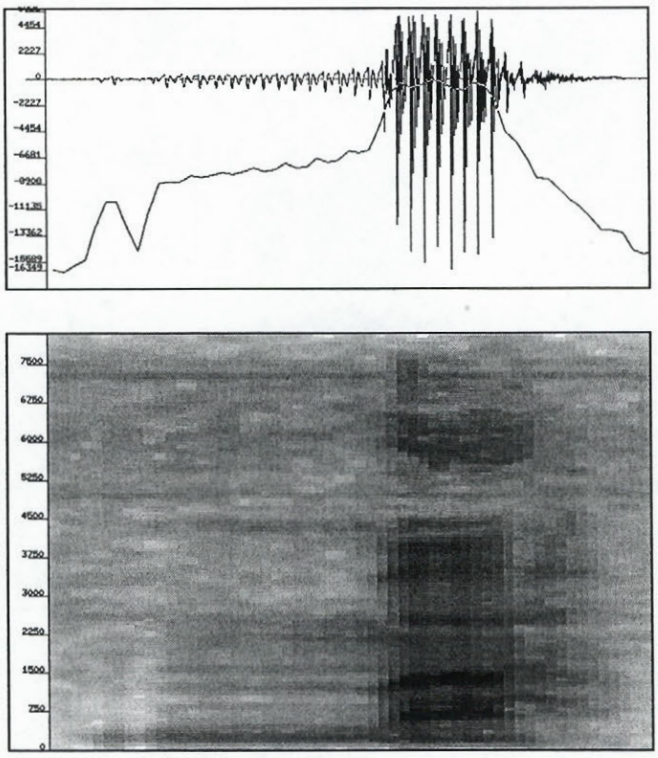

Figure 1.11: Waveform and spectrogram for /ma/

Lastly, Figure 1.11 shows a spectrogram for a nasal phoneme, */ma/*. One can notice that the first half of the spectrogram has many weak lines. This is caused by the nasalisation of the *m* sound. This corresponds to resonance of air through the nostrils. There is then a sharp transition as the velum gets closed. This can be seen by the sudden rise in amplitude in the waveform, as well as the appearance of dark areas on the spectrogram. The beginning of the */a/* sound can also be seen as voicing begins, indicated by the two dark formants in the bottom of the image.

Chapter 2

Feature Extraction

The purpose of feature extraction is to obtain information from a speech spectrum in order to recognize what was spoken. Cepstral coefficients are computed from the spectrum. These coefficients are used to represent the features a speech signal's spectrum, as shown in Chapter 1, in the form of a vectors for a sequence of discrete time intervals.

2.1 Data Acquisition

Data acquisition is required in order to input the speech signal from the user into the computer. This will usually involve a microphone and an analog to digital converter. These are the only components which might have to be added to a general purpose computer in order to do ASR work on it. The converter will transform sounds into a digital form which consists of a sequence of signed integers. The quality of the digitized sound depends on the sampling rate (higher is better) and the number of bits in the representation. Because the bandwidth of speech is at approximately 8kHz, the signal must be sampled at at least 16kHz in order to fully capture the signal. The greater the number of bits in the sample size, the higher the resolution

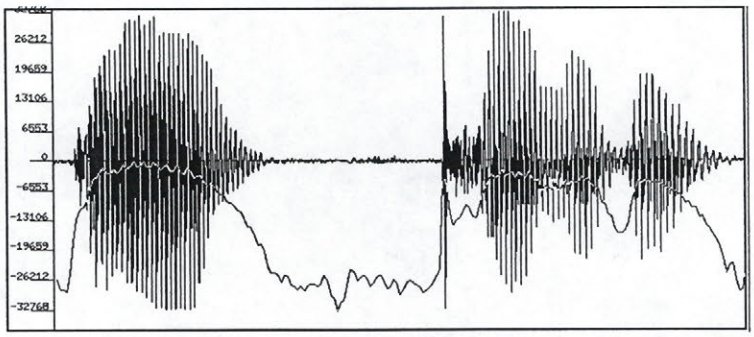

Figure 2.1: Sound waveform for the utterance *Air Canada*

of the sampling will be. Current systems, including Roger tend to use 16 bits as a standard quantization.

2.2 Acoustic Features

The initial sampling of a speech signal generates a vector of numbers representing the waveform of the signal. This represents the amplitude of the signal as time progresses. The waveform is of limited use, however, since two different people speaking the same word will generate different waveform plots. To make matters worse, the waveform is generally not the same for one person speaking the same word twice! In order to reduce variabilities during different pronunciations of the same word, a spectrum is computed from the waveform. For a given word, peaks and valleys within the spectrum will tend to maintain the same positions and can therefore be correlated to the utterance which produced the speech signal. Because of this, it is the spectrum which must be analyzed in order to recognize speech. A spectrogram is a plot of energy distributed over frequency and time, where darker areas represent a higher amount of energy. When trying to recognize speech, we wish to identify *acoustic cues*, specific components of the speech signal which characterize a phoneme. Theoretically, these cues can be found by analyzing frequency patterns in a spectrogram computed for an utterance. One must look for patterns which are constant for a given phoneme,

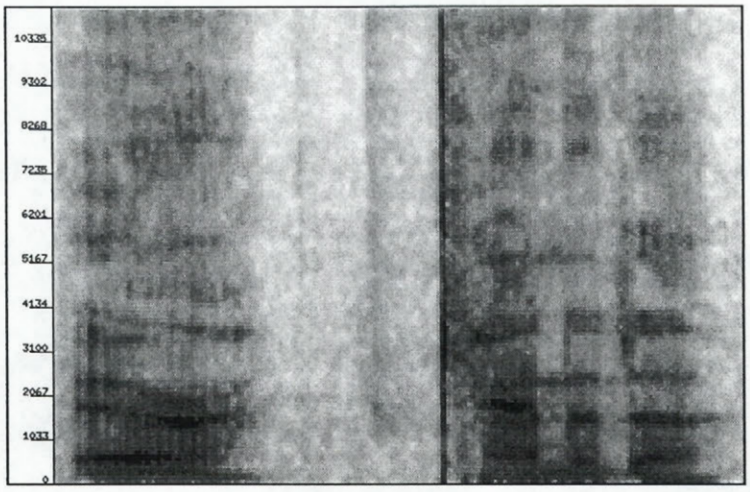

Figure 2.2: Spectrogram for the utterance *Air Canada*

these would be *invariant acoustic cues.* In the larger context of the entire spectrum of an utterance, these do not exist; researchers note that the spectrum for a phoneme changes depending on its context and its speaker. This is *acoustic variability* . One can look for acoustic cues for vowels in the *formants* which can be seen on a spectrogram as dark smeared lines of energy. The spectrogram for the vowel /a/ in Figure 1.5 provides a good example. An acoustic cue can be useful in a certain context. If a short silence, for example, is inserted between /s/ and /i/ a listener will usually perceive the sound *ski* whereas if a short silence is inserted between the sounds /s/ and /u/ the same listener will perceive *spu.* In the first case, the silence is an acoustic cue for a /k/ sound, yet in the second, the same silence cues a /p/ [Jusc86]. The frequencies of particular formants will also depend on the speaker. Generally, male speakers will generate a formant at a lower frequency than a female speaker for the same vowel.

2.3 Feature Extraction

From this point on, we will assume that the signal is represented as a sequence of signed 16 bit integer values, each of which represents an amplitude for a duration of 0.0000625 seconds (this from a 16kHz sampling rate).

2.3.1 Pre-Emphasis

The signal must first be pre-emphasized. This will amplify the spectral components above 1000 Hertz, where humans are known to be more perceptive. It will compensate for the attenuation of spectral peak energies at higher frequencies. This partially normalizes peak amplitudes [Pico93]. Pre-emphasis is performed using the following equation which is applied to each sample of a digitized utterance.

$$S_{pe}(i) = S(i) - \alpha S(i-1) \quad 1 \leq i \leq N_s \tag{2.1}$$

where $S(i)$ is the i^{th} sample in the signal, N_s is the number of samples in the complete signal, and α is some constant. A common value for α is 0.95.

2.3.2 Frames

Now that the samples have been pre-emphasized, they must be grouped into frames using a sliding window system. A frame contains a number of samples required to represent a particular length of time and will generally overlap with the previous and subsequent frames. The duration of the frame represents a tradeoff between accuracy of pitch characterization, typified by a larger frame, and rapid detection of spectral changes, which comes about by using a smaller frame. The Roger ASR system (see Chapter 4) generally uses a 20ms frame with a 10ms overlap. A Hamming window is used in order to filter the samples in a given frame [Rab93].

2.5 Windowing

In order to capture the dynamic cues of speech, the sampled signal must be analyzed in windows. The size of a window depends on a tradeoff. First and foremost, very small windows will reduce the accuracy of the DFT since there will be fewer samples available. A small window will be able to catch rapidly changing features in the speech signal, but features which require a significant length of time to be completely represented will not be easily spotted. A large window switches this around so that rapidly changing features of the spectrum become hard to notice, but slowly evolving ones stand out more readily. The Roger ASR system (see Chapter 4) uses a 20ms window with a 10ms window overlap. Several windows can be used in order to preprocess the sample frame. The simplest window has of course a rectangular shape:

$$w(n) = r(n) = \begin{cases} 1 & \text{for } 0 \leq n \leq N-1 \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

A window introduces distortions in the short-time spectra. It has been observed that the best results tend to come when a Hamming window is used. It resembles a gaussian which doesn't quite reach zero at its edges. It is specified using the following function:

$$w(n) = h(n) = \begin{cases} 0.54 - 0.46\cos(\frac{2\pi n}{N-1}) & \text{for } 0 \leq n \leq N-1 \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

Other window types are of course possible. A window function need only have a domain of 1 to n and its range must be between 0 and 1. Applying a window filter to the samples in a frame is nothing more than multiplying the value of the filter at position n with the value of sample n in the frame.

bility that if at time t the HMM is in state q_t, it will jump to state q_{t+1} at time $t+1$. Therefore:

$$a_{ij} = P(q_{t+1} = j | q_t = i) \quad 1 \leq i, j \leq N \tag{3.1}$$

If any state can be reached from any other state in the model, then we have the property that $a_{ij} > 0$ for all i and j. If a state can not be reached from another, then the probability of that transition is 0.

- Observation symbol probability distribution
  The Observation symbol probability distribution, $B = [b_j(k)]$, describes the probability that symbol k will be observed while the model is in state j. This can be expressed as

$$b_j(k) = P(O_t = v_k | q_t = j) \quad 1 \leq k \leq M; \quad 1 \leq j \leq N \tag{3.2}$$

  where O_t is the observation symbol at time t

- Initial state distribution The initial state distribution, $\Pi = \{\pi_1, \pi_2, ..., \pi_N\}$, is a list which indicates the probability that the HMM will start in state i

$$\pi_i = P(q_1 = i) \quad 1 \leq i \leq N \tag{3.3}$$

3.1.2 HMM training and the Baum-Welch Algorithm

In order to compute probabilities for a sequence of events, using HMMs, one can use the forward-backward procedure [Pico90]. The following equations illustrate how

this is done. We wish to have a probability measurement from a model G given a particular observation sequence O. This probability is the sum of the probabilities of all the paths within G which could have been taken to arrive at O. Essentially,

$$P(O|G) = \sum_{i=1}^{N} \alpha_T(i) \tag{3.4}$$

where

$$\alpha_1(i) = \pi_i b_i(O_1) \quad 1 \leq i \leq N \tag{3.5}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \quad 1 \leq t \leq T-1, \quad 1 \leq j \leq N \tag{3.6}$$

and

$$\beta_T(i) = 1 \quad 1 \leq i \leq N \tag{3.7}$$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \tag{3.8}$$

Hidden Markov Models can be trained iteratively to estimate their parameters. A Maximum Likelihood Estimation (MLE) procedure [Pico90] recomputes a new model which will improve the probability of the input vector given a previous model. An initial model can be established initially with random parameters. First, we must compute the expected number of times that the system will start in state S_i.

$$\check{\pi}_1 = \alpha_1(i)\beta_1(i) \tag{3.9}$$

Now we must compute the state transition probability matrix. For entry (i, j) the probability of going from state S_i to state S_j is the expected number of transitions from S_i to S_j divided by the total number of transitions from state S_i.

$$\check{a}_{ij} = \frac{\sum_{t=1}^{T-1} \alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)} \tag{3.10}$$

The next value to compute is the probability of observing a given symbol while in a given state. This is the expected number of times that we observe the symbol v_k while in the state S_j divided by the number of times we expect to be in state S_j.

$$\check{b}_j(k) = \frac{\sum_{t=1, O_t=v_k}^{T-1} \alpha_t(i)\beta_t(i)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)} \tag{3.11}$$

By using this method with a large training set, a set of phoneme Hidden Markov Models can be built which will be used for ASR.

3.2 Finite State Networks for Phonemes

In order to increase the accuracy of the recognizer and reduce the amount of required computation, a finite state network (FSN) of phonemes is used. This FSN is built by analyzing the grammar of the required task. Each transition in the network represents a phoneme model. During recognition, every node in the network is associated with a probability updated based on the input vector. When the end of utterance is reached, this network is traversed using backtracking so that the highest scoring path is found. This path represents the most likely phoneme sequence for the given utterance. By

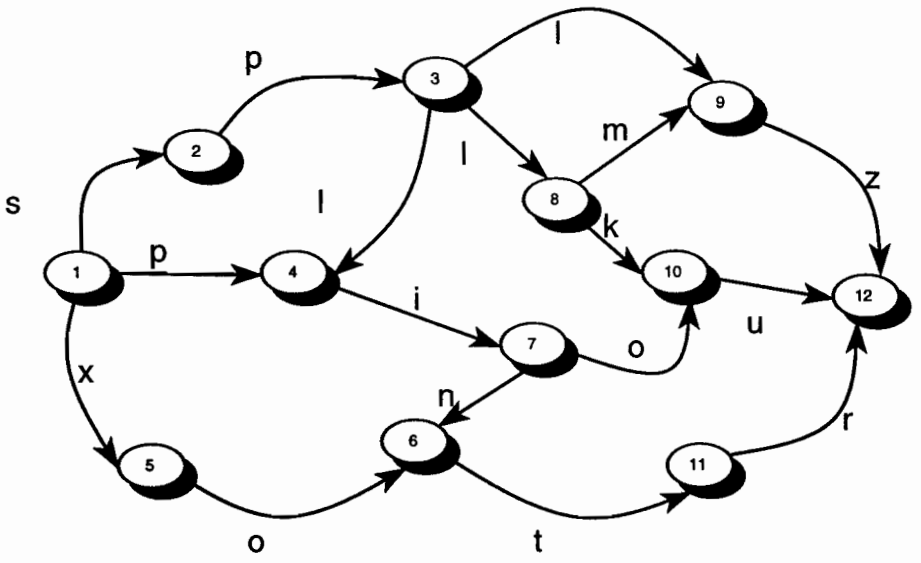

Figure 3.2: A simplified version of a finite state phoneme network [Gall92]

using such a network, the recognition is constrained to sentences accepted by the given grammar. The drawback to this is that any utterance will map into the grammar, so a speaker saying something which is unrelated to the task will still get a recognition which matches the task grammar. An alternative consists in considering an FSN with all the words in a lexicon and connecting each pair of words with an arc to which a bigram probability is associated.

The result of recognition is a lattice consisting of the most likely list of phonemes or words in the utterance. The *Viterbi Algorithm* [Vite67] is used in order to find the highest scoring sequence of phonemes or words. This algorithm is a dynamic programming method for maximum likelihood decoding.

3.3 Beam Search

Beam search is a non-admissible but effective search method used to speed up the computation time by speeding up the Viterbi search [Lowe76]. It works by restricting

the search to nodes which have a likelihood greater than some fraction of the maximum likelihood of a partial hypothesis in the search. If P_l^{max} is the maximum log-likelihood of a partial hypothesis, only those nodes whose log-likelihoods are greater than $P_l^{max} - \Delta$ are kept in the list of nodes to search. The parameter Δ is called the *beam width.* The smaller the beam, the faster the search since more nodes are pruned. However, a very small beam will remove nodes which would create highly probable paths because those nodes might have a poor local acoustic match, hence a low local likelihood. A proper beam width is therefore a tradeoff between speed and accuracy. Lee [Lee90] observes that the speed up is almost inversely linearly proportional to Δ.

ues and causes a rereading of the parameter file as specified in SA_PARM_FILE (see section 4.3). Once this is done, a RESET code is sent down the recognition stream so that the other processes can do the same.

- **samplerate** *CODEC* or *16* or *22* or *44*

  This command which must be issued before *mikeon* alters the recording sampling rate. It will currently accept CODEC (8kHz), 16, 22, or 44. The default is to leave it at 16 since all models in the system were trained for this rate.

- **save** *filename*

  This command will save the last complete utterance in a NeXT format sound file.

- **duration** *time*

  This command identifies the length of time that a signal must remain below the silence threshold in order to stop the automatic recording. The time is specified in half second blocks.

- **noise** *threshold*

  The noise threshold is the level which the signal must reach before Recorder will automatically start recording. The range is from 0 to 1 where 0 will automatically start recording as soon as *mikeon* is issued and 1 will never record anything.

- **calibrate**

  Causes the Recorder to recalibrate itself when the record parameters have been changed.

- **gain** *-20* to *20*

  Alters the current recording gain. For soft speakers, the gain should be set

higher, for loud speakers, it should be lower. The value must is interpreted in decibels.

- **playback**

  This command will use the system's native sound play command to playback the last complete recorded utterance.

- **line** *line* or *mike*

  This command determines the signal input source. Line is the system's linein, whereas mike is the microphone jack.

- **mode** *burst* or *realtime*

  The mode command controls the fashion in which the speech signal is delivered down the recognition stream. In burst mode, the Recorder will wait until the end of utterance is reached then send the entire signal in one burst. In realtime mode, buffers containing signal data are immediately sent down the stream as soon as they are available. The default mode is realtime.

- **rogermode** *on* or *off*

  Rogermode informs Recorder if the Roger interface is attached to the stream.

Recorder generates a standard output format on its *stdout* stream. Each packet of information consists of 32 bits. If the high order 16 bits are all zero, then the low order 16 bits represent an individual sample from the current signal being recorded. If the high order 16 bits are non zero, then the 32 bit sequence represents a control code. These are as follows (Actual numbers are not given for the sake of brevity)

- **RESET**

  This code forces all the processes which receive it to reset their parameters to default values and reread the parameter file.

- **SOU**

  This code indicates that there is now a new utterance being sent down the recognition stream.

- **NOP**

  This is a null control code. It should be ignored by anything that receives it.

- **EOU**

  This code indicates the end of the current utterance. When it is received by recognizer, it should begin backtracking in order to compute a complete recognition string.

- **QUIT**

  This code should cause everything to receive it to terminate gracefully.

- **MADAPT**

  MADAPT will cause the recognizer to update its HMMs based on the contents of the feature vector caches maintained for each Hidden Markov Model. It forces speaker adaptation on the current utterance.

- **INFO**

  This packet passes information from one process to another. The low order 16 bits of the code represent the particular type of information and the next 32 bits represent additional data, such as a parameter. These next 32 bits should not be used as signal data.

- **CALIBRATEON**

  Indicates to the feature extractor that the sample data which follows is recorded silence. This is used by the feature extractor (SA) in order to calibrate itself.

- **CALIBRATEOFF**

  This indicates the end of the recorded silence.

More codes can be added to Recorder and passed down the recognition stream. Regardless of whether or not a process recognizes one of these control codes, it must always pass it downstream to the next process without modifying it. This way a new command could be added to the recognizer and controlled by the Recorder without ever having to modify the feature extractor. Unrecognized control codes by default are ignored and passed on.

4.1.2 Feature Extraction

Feature extraction is performed by a program called *SA*. It will expect data on its *stdin* stream in the format described for the output of the Recorder process. SA will break the signal down into frames then compute various features from those frames. The features are then output to SA's *stdout* as a list of IEEE format floating point numbers. SA uses a special floating point number to indicate that what follows is actually a control code and not frame data. These floating point numbers will generally have the low order 16 bits all set to 1. SA will currently accept and process all control codes which can be output by Recorder with the exception of **MADAPT**. This unit implements the basic theory presented in chapter 2.

4.1.3 HMM Recognizer

The basic recognition engine is called *recognizer* and is written by Michael Galler and Philippe Boucher. Recognizer expects feature vectors from SA and uses them to update internal structures using Hidden Markov Models (HMM) in order to compute a recognition string for the original signal. Recognizer does not recognize any command line switches. The basic theory behind the recognition algorithm is described in chapter 3.

4.2 Interface

In addition to the parser provided by the Recorder module, Roger also has an X Windows based interface. This interface is written entirely in Tcl/Tk [Oust94][Oust93] and allows the user to control all aspects of the recognizer, including several additional tools in order to manipulate sound files, spectrograms, system audio parameters and HMMs. The interface attaches itself as two additional processes in the speech recognition stream. The Control window is attached directly to Recorder's parser and will in fact pass commands to Roger by placing text commands on Recorder's *stdin* stream. The Results window is attached to the stdout of the Recognizer process. It will parse and format the output of Recognizer, display it and pass the result further down the stream in case another process is attached to it.

4.2.1 Control Window

Roger's main control window is shown in Figure 4.2. The various functions which it can perform are described below. Each subsection represents one of the submenues associated with the entries on the menu bar (which contains Recorder, Sound, Parameters, and Help).

Recorder

This button creates a menu whose functions control all of the data acquisition parameters. These are as follows:

- **Settings**

  This option allows the user to alter the record parameters. *Noise Threshold* controls the loudness which a sound must initially have before it will cause the Recorder to automatically pick up the voice. The higher the threshold the louder the initial sound must be. At a threshold of 0, the Recorder will start

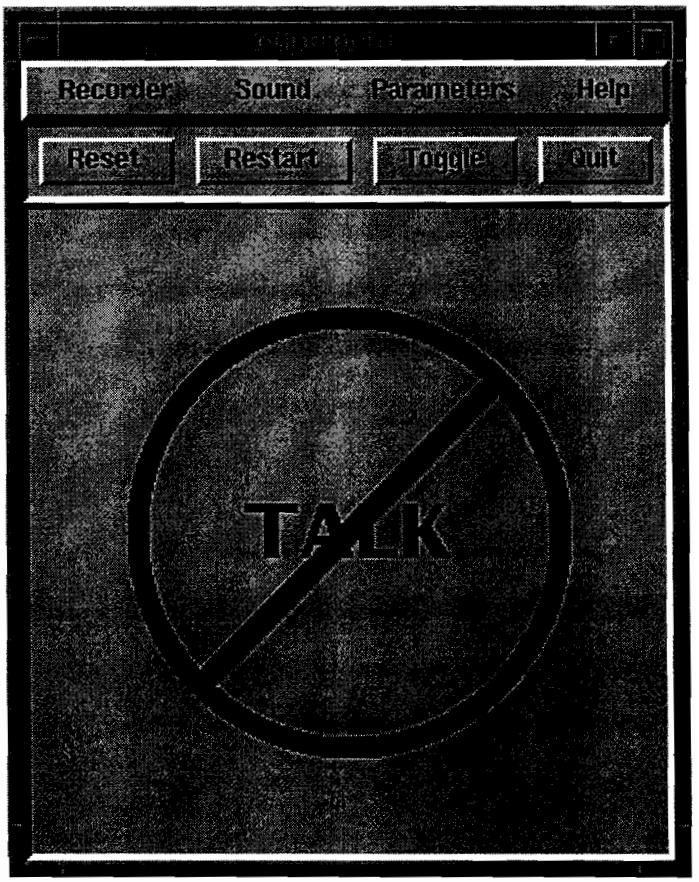

Figure 4.2: Roger's main control window

automatically, while at a threshold of 1, the Recorder will never pick anything up. The *Silence Threshold* indicates the level below which sound must lie for it to be considered silence. As before, a threshold of 1 would make any sound considered to be silent, while a threshold of 0 would cause any level of sound to be non silent. the *Duration Length* is the amount of time in half second intervals that the sound must be below the silence threshold before the the Recorder will turn itself off. *Gain Level* is the system's recoding gain, measured in db. For best results the thresholds should be kept low, with the silence threshold lower than the noise threshold. A recommended setting is 3% for the noise threshold, 2% for the silence threshold, a silence duration of 2 and a gain of 7. In noisy environments, the noise and silence thresholds should be raised. The gain also needs to be adjusted upwards for speakers with soft voices and downwards for loud speakers. Pressing the *Set* button will cause the system to calibrate its sound hardware using the new parameters.

- **Calibrate**

  This option will cause the system to recalibrate its sound hardware based on the current record parameters defined in Settings above. This operation is automatically done whenever Roger is launched, but should be repeated if a loud noise occurred during calibration.

- **Repeat**

  Repeat allows the user to repeat his or her last utterance by playing back the recorded sound into the recognition stream.

- **Feed**

  This option allows the user to feed a prerecorded sound file into the recognition stream. Currently, the only format supported is signed 16 bit linear sampling data, similar to the NeXT sound format without the header. The file will be

interpreted according to the current system sampling rate.

- **LineIn**

  This switches the input source to a device attached to the linein of the computer's audio hardware. When the input source is switched, the system should be recalibrated.

- **Microphone**

  This switches the input source to the microphone device attached to the computer. As for linein, the system has to be recalibrated when this option is used.

Sound

The Sound menu gives the user access to the stored sound files of Roger, as well as access to the native audio applets which are provided with the computer on which Roger is running. These functions are as follows:

- **Playback**

  This will play back the last recorded utterance using the computer's native sound play utility.

- **Save**

  This will save the last recorded utterance to a NeXT format sound file (.snd)

- **Sgram**

  Sgram will display a waveform and spectrogram of the last recorded utterance.

- **AControl**

  This will launch the audio control software native to the computer on which Roger is running.

- **AudioEditor**

  This will launch the native audio file editor of the computer on which Roger is running.

Parameters

The Parameters menu is used to edit the values which are present in the parameter file SA_PARM_FILE (see section 4.3). It provides the following functions:

- **Edit**

  This option will popup a window which will display the current parameters as defined in the current parameter file. Double clicking on a parameter will bring up a second window which will allow you to edit it. Choosing the *Apply* button will cause Roger to be reinitialized using these new parameters but they will not be saved in the permanent parameter file. Choosing *Cancel* will of course cancel any changes which you have made to the current parameters

- **Save**

  This will save the current parameters held by the interface to the current parameter file in use.

- **Load**

  Allows the user to load a new parameter file and reinitialize Roger based on their values.

Help

This is a standard Help menu, its two functions are:

- **Help** This function will present a general help document on how to use Roger, the list of parameters that it understands, suggested settings for data acquisition, preparing grammars and transcription files, as well as a troubleshooting

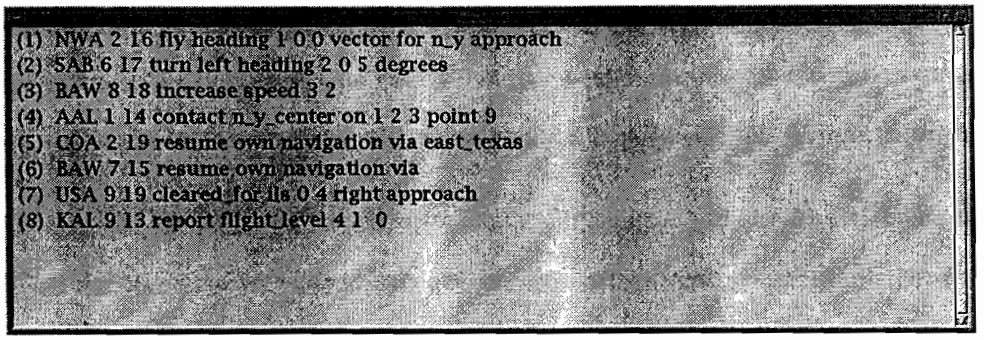

Figure 4.3: Roger's main recognition result window

guide in case the user hits some difficulty.

- **About** About will create a standard information window giving the name of the program, the version and revision numbers, as well as the list of authors.

4.2.2 Result Window

The Result window will display an enumerated list of all utterances which Roger has attempted to recognize, as the example in Figure 4.3 demonstrates. A slider on the right of the window allows the user to review all of the sentences. The interface will also output the recognized sentence string to stdout so that other processes can be attached to Roger and know what the recognition was. Common attachments at this point would be a speech synthesizer or a dialogue system so that responses can be spoken to the user.

4.3 Parameter settings

The parameters which are understood by the various components of Roger are described below. They can be accessed either by editing the library version of the pa-

rameter file, or, they can be altered during recognition using the Parameters submenu from the control window. These parameters control the behaviour of the recognition system following the theory set forth in the previous 2 chapters.

- **SampleRate**=*integer*

  The sampling rate at which audio data is being required. This should be set to the same value which was used to generate the models which will be loaded into the recognizer.

- **Advance**=*integer*

  The amount of time by which the window is to be moved ahead in order to generate the next frame.

- **HmmDictFile**=*filename*

  The path and file name of the phoneme dictionary file

- **HmmPrefix**=*directory*

  The directory where the HMM binary files are to be found

- **MaxSentLen**=*integer*

  This is the maximum number of labels or tokens which can be generated in the recognition of an utterance.

- **MaxObsLen**=*integer*

  This is the maximum number of frames in an utterance. Once an utterance reaches this number, a recognition up to that point will be automatically produced, regardless of whether or not there is more utterance to come.

- **FsnFile**=*filename*

  The path and file name of the task grammar file

- **RecBuffSize**=*integer*

  This is the maximum number of frames which will be passed to the Viterbi algorithm at one time.

- **BeamThreshold**=*real_number*

  The BeamThreshold is the initial threshold value used by the beam search in the recognition module.

- **FinalBeamThreshold**=*real_number*

  The FinalBeamThreshold is the final threshold value which is used by the beam search in the recognition module. Setting both beam threshold values to zero will cause the beam search to be disabled during recognition. Beam thresholds are usually set to be very small values. The higher the value of the threshold, the quicker the recognition will be, but at the cost of accuracy. The Final-BeamThreshold is only used when the system is on the last unit or phoneme of a word.

- **SegmentDump**=*integer_flag*

  Setting SegmentDump to 1 will cause a segmentation file to be written out at the end of each recognized utterance. The file will always be /tmp/save.seg. This file contains the the start and end *sample* numbers of each recognized token, as well as the token itself.

- **ThreshFile**=*filename*

  This file contains threshold values for model specific beam search.

- **SegFileList**=*filename*

  This specifies a list of segmentation files for the recognizer. This parameter is used mostly during a training run: It lists where the label files for a given sentence recording can be found, so that the recognized labels can be compared

with the actual labels of the utterance. This permits the system to automatically evaluate its performance.

- **TransFile**=*filename*

  The path and file name for the phoneme to word transcription list. This file consists of a token as it is referred to in the appropriate grammar file, followed by the phoneme models which make it up. There can be more than one representation for each token.

- **StatsFile**=*filename*

  This setting allows the user to specify a file in which statistics about the entire recognition run can be stored.

- **QuickStep**=*integer_flag*

  This controls which evaluation function is used for the Viterbi algorithm. A setting of 0 will cause all distributions of a transition to be used in scoring the transition, whereas a setting of 1 will cause the highest valued distribution to be used in order to score the transition. If you wish to use the vector quantizer optimization, this setting must be set to 1 (see chapter 5).

- **OffLine**=*integer_flag*

  OffLine recognition is done when the recognizer is used to run a large set of test utterances. Setting this flag to 1 therefore puts the recognizer in batch mode. Setting the flag to 0 or not specifying this flag will allow the recognizer to function with an interface and a human speaker.

- **NormFsnProbs**=*integer_flag*

  Normalizes the probabilities of the transitions in the fsn file

- **BestPath**=*integer_flag*

  Controls the behaviour of the Viterbi search

- **VqdFile**=*filename*

  This indicates which distribution usage file to use. This file works in conjunction with the vector quantizer so that the recognizer will know which distributions it should look at in order to evaluate each transition in the HMMs. The VqdFile must be specified.

- **VQCodeBook**=*filename*

  This file contains the list of codewords which the vector quantizer uses. You must be sure to specify a codebook which matches the VQD file you have selected. The Codebook file must be specified.

- **UsedDistributionList**=*filename*

  Specifies a file which contains a list of distributions to use during the evaluation of mixtures in the HMMs.

- **LatticeDump**=*integer_flag*

  This forces the recognizer to print out the phoneme lattice at the end of each completed recognition of an utterance.

- **AutoAdapt**=*integer_flag*

  Setting AutoAdapt to 1 will force the recognizer to update the model distribution means automatically after each utterance is recognized. If this setting is not specified or set to 0, then the model distribution means will only be updated when the recognizer module receives an ADAPT control packet. This packet can be sent either by clicking on the adapt button on the X windows interface or by entering the command adapt in the command line interface after a successful recognition.

- **SilThresh**=*integer*

  This silence threshold is used by Recorder in order to zero out long sections of a sampled speech signal. If the samples from the signal are below this value for

the number of samples indicated by **SilAvg** then they will all be changed to 0. If this setting is used, it must be accompanied by **SilAvg**.

- **SilAvg**=*integer*

  This setting determines the minimum number of samples which must lie below **SilThresh** in order to zero out the particular block of the signal. The signal will continue to be zeroed out until a sample is received was absolute value is greater than **SilThesh**. Note that under no circumstance will a sample which is greater than the threshold be canceled.

- **InputChannels**=*integer*

  This indicates the number of audio input channels. Currently, the Roger ASR system can only use a single input channel (mono).

- **SampleBits**=*integer*

  This is the size in bits of each sample which is received by the feature extractor. This value should stay set to 16 as it properly represents a signed short integer.

- **TimeOut**=*integer*

  The maximum length of time that the feature extractor will wait for a complete block of audio data to be sent from the Recorder process. If this timeout expires, then the feature extractor will use whatever sampled data it has received at that point.

- **WindowSize**=*integer*

  This is the length a window in milliseconds. The size in samples of a window depends on the sample rate, for example, a 20 ms window at a sample rate of 16kHz would contain 320 samples.

- **Coding**=*label*

  Coding describes the format of the sound data received by the feature extractor.

This setting should always stay at linear

- **PreEmph**=*real_value*

  This is the multiplier applied to samples in order to pre-emphasize them.

- **MelCoefs**=*integer*

  This is the number of Mel coefficients.

- **Filters**=*integer*

  This is the number of filters

- **Undersample**=*integer_flag*

  This setting is used when the Recorder is forced by the audio hardware to sample at a rate which is different than the one with which the models were trained.

- **DFTlen**=*integer*

  This is the size of the discrete Fourier transform, measured in distinct data points.

4.4 Datafiles

Roger requires a number of files in order to operate correctly. The most important ones are now described.

- **Model files**

  These files are the most important of Roger. They contain the model descriptions for each phoneme. There are currently two formats, one binary and the other text. The text format consists first of a header describing the name, size, and type of the HMM. This is followed by a list of transitions from one state to another. The format describes a complete transition per line, starting with

the initial state followed by the destination state (note that in this format, the states are labeled starting from 1, not 0). The probability of the transition is next, written in exponential notation. The last item on the line is a distribution index. Groups of transitions which have a common origin state and a common destination state form a mixture and are usually stored in sequence. The recognizer will try to read the number of transitions denoted by `Transitions` in the model header. Once the transition list is complete, the distributions must be added. First is the tag name `Distribution XXX` where `XXX` is the distribution index. Each distribution is merely a list of parameters represented by 2 exponential notation numbers. The first is the mean value of a gaussian, the second, its variance. There is one line per feature used in the models. In the current implementation of Roger, each model uses 26 features, therefore, every distribution list has 26 lines. The number of entries required per distribution is denoted by `DistribParam` in the header. A very simple text HMM file would look like:

```
Name            Sample
Type            Continuous
DistribParam    2
Size            2
Transitions     2 0
#From   To   Probability                 Distribution
1       1    2.63269614537241440e-02                1
1       2    1.89403992991330450e-02                1
2       2    1.49167526867538000e-02                2
Distribution 1
7.07089867052442810e-02  4.24948756519003010e+00
1.02406544091708970e+00  1.25856403969781900e+00
Distribution 2
-2.33432155436842520e+01  1.25886179241306650e+02
2.01287064721203680e+00  1.08598523588947710e+02
```

The binary HMM files follow a similar format. There is first a 1024 byte block which stores the name of the model. This is followed by a 4 byte integer which represents the type. For Continuous HMMs, this value must be set to 2. The

number of parameters per distribution is next, as well as the number of states and the number of transitions. Each of these is a 4 byte integer. After the header is the transition list, as in the text format this consists of the origin and destination states, each represented by an integer, the transition probability, a double, then the distribution index, an integer. The transition list is terminated by a 4 byte field which is unused. The next integer indicates the number of distributions which must be read. The distributions consist first of the list of mean values followed by the list of variance values. All are stored as doubles. All integers in this format are 4 bytes, signed, and big endian. All doubles used are in 8 byte IEEE format. The model distributions are generated with a separate system which uses the Baum-Welch training algorithm [Gall92]. The training is done using a database of speech utterances where the phonemes have been labeled. The particular database used for training Roger is TIMIT. The number of states present in a model is actually arbitrarily set as a parameter when training is begun. A complete description of how the models are generated is beyond the scope of this work.

- **Parameter file**

  This file controls the behaviour of Roger by setting all parameters and values described below. This file is specified to the system by setting the environment variable SA_PARM_FILE to the file and path name required before launching Roger. Parameters are set by entering the parameter name in the file, followed by an = and a value. Comments are denoted by a ; character, anything between this and a newline is ignored. The following example is the file used by the demonstration version of Roger for the ATS Air Traffic Control Task.

```
SampleRate=16000          ; samples per second
InputChannels=1           ; number of input channels
SampleBits=16             ; bits per sample
```

```
TimeOut=10                  ; timeout for something
WindowSize=20               ; width of window in milliseconds
Advance=10                  ; window advance in milliseconds
Coding=linear               ; encoding of samples
PreEmph=0.95                ; pre-emphasis factor
MelCoefs=12                 ; number of mel coefficients
Filters=24                  ; number of filters
MaxObsLen=800               ; max frames per utterance
MaxSentLen=200              ; max labels per output string
DFTlen=512                  ; Size of Discrete Fourier Transform
Undersample=0
RecBuffSize=20              ; max frames per call to viterbi
HmmDictFile=phonemes.dic    ; phoneme list
FsnFile=ATS
TransFile=ATS.tran          ; transcription file
BeamThreshold=1.0e-46       ; beam search parameters for ATS task
FinalBeamThreshold=1.0e-30 ; beam search parameters for ATS task
HmmPrefix=/continuous/phonemes/
QuickStep=1                 ; '1' algorithm for quick viterbi steps
OffLine=1
BestPath=1
```

- **Phoneme Dictionary File**

  This file must be specified as an entry in the parameter file. It indicates to Roger which HMM files must be loaded into the system. It must have a header describing the number of models, the type, the size of each distribution in the models and the actual model names.

```
Name            55
Type            Continuous
DistribParam    26
Models {
        garbage
        aa
        ae
        ah
        ao
}
```

- **Grammar file**

  The grammar file describes where and when phoneme models are likely to appear. It is generated from another utility. Its purpose is to constrain the

recognition run so that it will be faster and more accurate within the context of the desired task for which Roger has been set. The grammar file describes at which positions recognition tokens can appear. It is used in conjunction with the transcription file, so that the structure of the individual recognition tokens (like **UA** in the example below) can be known.

```
sil.(
        ((AC.sil+AC).(3+3.sil).(9+9.sil).(1+1.sil))
        +((AC.sil+AC).(8+8.sil).(5+5.sil).(9+9.sil))
        +((AC.sil+AC).(8+8.sil).(7+7.sil).(8+8.sil))
        +((AC.sil+AC).(8+8.sil).(7+7.sil).(9+9.sil))
        +((UA.sil+UA).(8+8.sil).(8+8.sil).(5+5.sil))
        +(tailnumber)
).sil
```

The above file fragment is a piece of the ATS task grammar which represents a silence, followed by an airline designation and 3 digits. In this particular example, AC is *Air Canada* and UA is *United*, which is also described below in the transcription file example. The notation of the grammar is specific to Roger. Units are surrounded by parenthesis. A + sign represents an OR and a . represents an AND. ANDs have precedence over ORs.

- **Transcription File**

  This file is a dictionary which will convert from a list of phonemes to a word token, thus making it easier to use the output of the recognizer since it will be simpler to read and multiple word pronunciations can be given the same final word token.

```
NW n ao r th w eh s tcl t
UA jh uw n ey tcl t iy dcl d
UA y uw n ay tcl t iy dcl d
Patrol pcl p ae tcl t r ow l
Patrol pcl p ih tcl t r ow l
9 n ao ay n ih ix r
```

The above file fragment represents the utterances *Northwest*, *United*, *United*, *Patrol*, *Patrol*, and *Niner*. The phonetic symbols used come from the TIMIT speech database.

Chapter 5

Vector Quantization for Selecting a Gaussian in Mixture

5.1 Introduction

Vector Quantization (VQ) is a commonly used method to reduce the amount of redundancy in a set of data [Gray84]. In imaging, it is used to reduce the number of colors in a picture while minimizing the distortion that this reduction would cause. This allows pictures to be displayed on systems with a small colour palette, as well as permitting the image to be compressed with minimal alteration to its appearance. In speech applications, it can be used to represent a set of audio samples using either a single value or a vector. Likewise, it can be used to represent a set of extracted features from a speech signal using a single value or vector, as is often done with discrete HMM systems. Vector quantization a method which can be used to cluster data. Using VQ will significantly reduce a system's storage requirement for the input data that it must analyze. The drawback to this however is that the information thus stored will have some level of distortion which is inversely proportional to the number of vectors to which the Vector Quantizer can map it's input. This list of vectors is

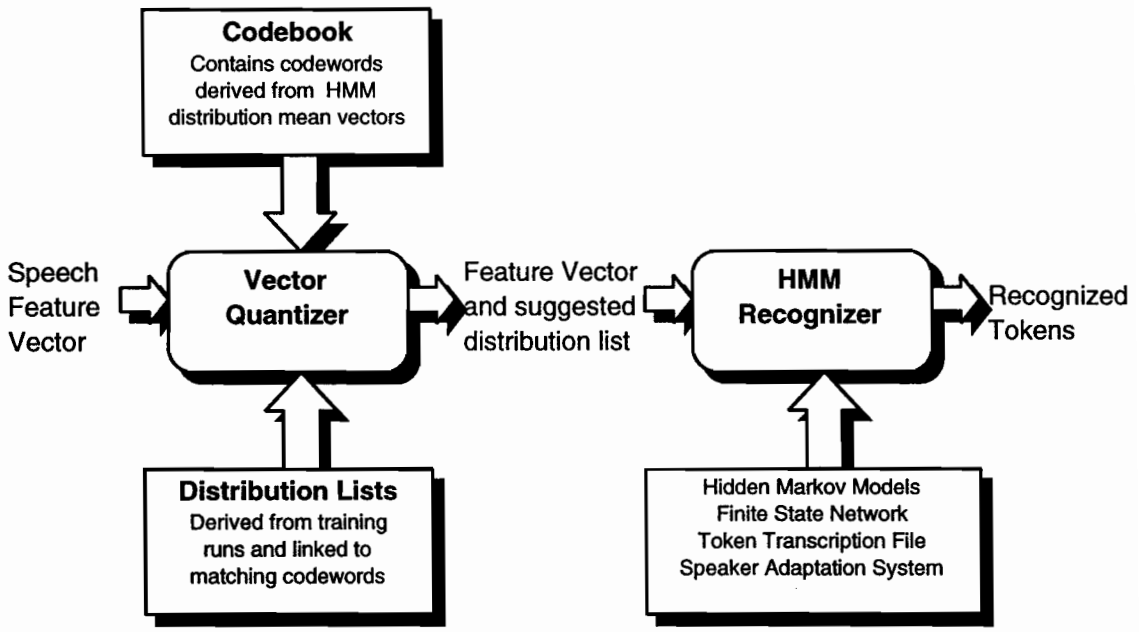

Figure 5.1: Structure of the Vector Quantizer preprocessor

referred to as the quantizer's codebook. The output of VQ is one of the vectors which are contained in that list. The amount of computation required to process data and determine the similarity of two vectors is much smaller. There is also the advantage that a discrete representation of the data (ie, by codeword) makes the data easier to manipulate. In order for a vector quantizer to have real usefulness, it usually will require a large codebook. This will induce its own overhead both for storage and for computation time required to search it. A properly designed codebook is then a tradeoff between the competing attributes of size, speed, and distortion, the cost of which must be compared to the gain that it gives over a more brute force approach to dealing with the input data. Small codebooks are easy to store and quick to search but have a large amount of distortion. Similarly, a large codebook will have a low amount of distortion but will be more difficult to store and time consuming to search.

5.2 Requirements

In order to build a proper codebook, a large set of training vectors is required. This data should match as closely as possible the kind of data that the final vector quantizer will be required to process. All cases of the data should be covered in the training set. Rabiner suggests that the training set should be at least 10 times as large as the desired codebook size [Rab93]. A measure of similarity between two data vectors is required so that the vectors can be clustered as well as classified into unique codebook entries. The similarity function is usually nothing more than a distance function between two vectors. It must have the following general form:

$$d(u,v) = \begin{cases} 0 & \text{if } u = v \\ \geq 0 & \text{otherwise} \end{cases} \tag{5.1}$$

A centroid computation procedure is required. The codewords are merely the centroids of each vector cluster which make up the quantizer. Finally, a classification procedure is necessary in order to match an arbitrary input vector to a codeword. This is usually accomplished with a nearest neighbour search. A fairly standard way of building codebooks which fit the above requirements is to use the Binary Split algorithm, which is based on the K-means algorithm.

5.2.1 K-Means Algorithm

K-Means is the basic VQ codebook building procedure [Gray84]. It requires the following definitions: M is the set of cluster codewords (the actual codebook). L is the set of training vectors. R is some arbitrary distance threshold which is used to terminate the procedure. This value depends entirely on the data being analyzed and the desired accuracy of the final codebook.

The K-Means algorithm now requires the following steps in order to build a general codebook.

1. Initialize the codebook by taking M vectors at random from the set of training vectors. Each of these vectors will act as the initial codeword of a vector cluster.

2. For each vector in the training set, use a nearest neighbour search in order to assign each one to the cluster to which it is closest.

3. For each cluster, generate its new codeword using the centroid computation procedure on the set of vectors assigned to it.

4. Compute the average distance between the set of training vectors and each's respective cluster codeword. If this distance is above some arbitrary threshold, re-iterate the procedure from step 2.

The K-Means algorithm will often generate a codebook with less than M codewords, since it is possible for no vectors to be grouped into a specific cluster. If this happens, then a new cluster can be generated depending on whatever criteria the user of the codebook wishes to use or the cluster can simply be discarded. An alternative is to use the Binary Split algorithm which guarantees to generate a codebook of the desired size.

5.2.2 Binary Split

1. Build a 1 cluster codebook. This is merely applying the centroid computation procedure to the entire vector training set.

2. Double the current size of the codebook. For each vector in the codebook, create two new codewords using the following rule:

$$y_{+n} = y_n(1 + \epsilon) \tag{5.2}$$

$$y_{-n} = y_n(1 - \epsilon) \tag{5.3}$$

ϵ is a constant value used to split the codeword in two. Its size should be fairly small [Rab93] between 0.01 and 0.05.

3. Apply the K-Means algorithm to the current codebook in order to optimize the current set of codewords to the data set.

4. Continue from step 2 until the codebook has reached the desired size M.

5.3 Distance and Distortion Measures

The distortion measure represents the cost or change which occurs when one goes from the input vector to the quantizer's output vector or codeword [Gray84]. In practice, the performance of a vector quantizer is represented using the average distortion over time in the long term. A good system will have a low average distortion, ie, the limit in Equation 5.4 is minimized.

$$\lim_{n\to\infty} \frac{1}{n} \Sigma_{i=0}^{n-1} d(X_i, Y) \tag{5.4}$$

There are various ways of computing the actual distortion from an input vector to a codeword. The particular method used will vary depending on the type of data being quantized. An example measure is as follows:

- The squared error distortion measure

  This computation can be applied to k-dimensional Euclidean vectors. It is merely the square of the distance between the two vectors

$$\begin{aligned} d(u,v) &= ||u-v||^2 \\ &= \Sigma_{i=0}^{k-1}(u_i - v_i)^2 \end{aligned} \tag{5.5}$$

Input dependent weights can also be used and have proved useful [Gray84] When using this type of distortion measure, it is common to represent the performance of the system using the signal to noise ratio.

$$SNR = \frac{10\log_{10} E(u^2)}{E[d(u,v)]} \tag{5.6}$$

A large SNR corresponds to a small average distortion and vice versa.

Rabiner describes a large set of other distance and distortion measures, such as the The Itakura-Saito distortion, likelihood-distortions, the COSH distortion and Cepstral distances.

5.4 Application of Vector Quantization to the Roger ASR system

We now describe in detail how the vector quantization preprocessor (VQPP) was added to the Roger ASR system. Roger uses continuous HMMs where each transition from one state to the next is represented by a set of one or more multivariant gaussians. This set of gaussians is referred to as the mixture for that transition. When evaluating the score for a particular transition, one will usually sum the score of each individual gaussian in the mixture given the observation in order to arrive at some value. However, an analysis of this sum reveals that most of the value comes from one or two gaussians in a particular mixture. We can therefore increase the

speed of the computation with almost no loss of accuracy by merely computing the log-likelihood scores of each individual gaussian and assigning it as the score for the mixture given a feature vector. This method compresses the computation for an individual gaussian and removes the need to expand and sum the values. However, all gaussians must still be evaluated individually, yet most are thrown away. This can be extremely wasteful as some of the models used by Roger may have over 40 gaussians per mixture!

First a codebook needs to be built. The raw data for it is extracted from the gaussian mean vectors found in the set of HMMs used by the speech recognition system itself. These vectors are generally representative of the kinds of feature vectors that one would expect from sampled speech. Codewords are then derived using the previously described Binary Split algorithm. The size of the codebook is determined experimentally. Now that we have a proper codebook, a special training speech recognizer is used. This recognizer maintains a record of each gaussian in each transition in each HMM which the system uses, indexed by codeword. These records will count the number of times that a particular gaussian has been used during training. As each feature vector is pushed through the recognizer, it is associated to the appropriate codeword in the VQPP's codebook. Recognition is then done normally on it, but once the winning gaussians and models are known for that vector, the appropriate records for those gaussians in the vector's codeword are incremented. At the end of a training run, their records are then output. From this data, a list of suggested gaussians per codeword per model can be generated given a minimum number of times that a gaussian must have been used in order to be included.

Now that we have a suggested gaussian list and a codebook, we can do normal recognition runs. The feature vector is first preprocessed in order to find its appropriate codeword. This codeword is then used in conjunction with the list of active models at the time in order to find which gaussians need to be evaluated for a particular transition, given the gaussian list. The recognizer then only evaluates those

gaussians given the input vector and keeps the maximum score for the particular transition. The experiments described in chapter 7 demonstrate that approximately 66% of the amount of computation can thus be removed with minimal loss of accuracy.

Chapter 6

Adaptation

6.1 Introduction

One way to improve the accuracy of a speech recognition system is to be able to make it adapt to the various speakers that it encounters. This is useful for Speaker Independent (SI) systems which will still be primarily used by the same user over an extended period of time, such as air traffic control simulators. On a system which uses continuous distribution HMMs, an easy way to do this is to dynamically adapt the distribution means of the models to the feature vectors extracted from the user's speech signal.

6.2 Mean Adaptation

Speaker adaptation can be done easily and without supervision by adapting the mean vectors of multivariant gaussians. In order to adapt the models to a given speaker, each transition from one state to another is associated with a cache. This cache holds the feature vectors which caused that transition to "win" at a particular point in the duration of the signal. At the end of an utterance, an average mean vector for each

cache is computed and the associated gaussian's mean vector is then updated using

$$\mathbf{U} = \sum_{i=1}^{N} \mathbf{u}_i \tag{6.1}$$

and

$$\check{\mathbf{X}} = (1 - \alpha)\mathbf{X} + \alpha\mathbf{U} \tag{6.2}$$

where $\mathbf{X}$ is the gaussian's original mean vector, $\check{\mathbf{X}}$ is the new mean vector, and $\mathbf{U}$ is the mean vector computed from the contents of the gaussian's cache. $\mathbf{u}_i$ is a vector from the cache and N is the size of the cache. α is the *adaptation parameter.* It is generally very small, around 0.01. Selecting a particular value for this variable represents a tradeoff between retaining knowledge in the model and quickly acquiring new information from the environment. Shinoda and Watanabe [Shin94] use a similar method and obtain a decrease in the error rate from 15.5% to 8.7% using unsupervised adaptation. Dragon Systems inc [Drag94] use the following algorithm in order to adapt models to speakers:

1. Assign a relevance R for the original models in the system.
2. For each frame, the recognition module will compute the likelihood that the frame was generated by the component i.
3. This probability is added to the accumulator for the counts, C_i.
4. This probability times the frame data is added to the accumulator for the means, f_i
5. New means can then be computed for the models at any time using the following formula:

$$U_i = \frac{Ru_i + f_i}{R + C} \tag{6.3}$$

Where u_i is a mean vector. Other parameters in the models can be adapted in the same way.

Using this method, Dragon Systems reports a drop in the percentage word error from 20.1% to 15.2% for unsupervised adaptation and a further drop to 13.6% for supervised adaptation on their evaluation test corpus.

6.3 Other Methods

There are other methods for adapting speaker independent recognition systems to a given environment or speaker. One of these is the REALISE system [Taka94] which attempts to deal with environmental factors in the spectrum of a signal. It will try to remove additive and multiplicative noise in a signal by comparing the spectra of a single test utterance from the training environment and a reference utterance. All incoming utterances are then adjusted to compensate using the results of the test. Additive noise is background noise from the environment itself, whereas multiplicative noise is caused by linear filters such as microphones, transmission channels, and different vocal tracts. REALISE assumes that a signal $x(t)$ is first corrupted by a linear filter $a(t)$, then by an uncorrelated additive noise $b(t)$. The spectrum of what the recognizer then sees is described by

$$Y(\omega) = A(\omega)X(\omega) + B(\omega) \tag{6.4}$$

where $A(\omega)$, $X(\omega)$, and $B(\omega)$ are the spectra of $a(t)$, $x(t)$, and $b(t)$. Given test and reference utterances, REALISE tries to estimate $A(\omega)$ and $B(\omega)$ without supervision.

Results on a demi-syllable HMM based Japanese recognizer show that a single reference utterance was enough to raise the word recognition accuracy from 55.2% to 77.2%. Another technique is called cepstral high-pass filtering. It is very robust and has a minimal computation cost [Herm91]. The RASTA method [Herm93] involves applying a high-pass filter to a log-spectral representation of speech feature vectors. Cepstral means normalization can be high-pass filtered simply by subtracting the short-term average of the speech feature vectors from the incoming speech feature vectors. These methods are intended to compensate directly for the unknown effects of linear filtering which can occur to the speech signal as it is being acquired.

6.4 Generalized Adaptation

When performing adaptation in an ASR system, we would like to be able to generalize the information that we gather to gaussian mixtures which have not yet been encountered in the input signal. There are many ways to do this. Maximum Likelihood Linear Regression (MLLR) is a method which allows adaptation to be performed with a very limited set of data by extending its knowledge about adapted gaussians to previously unseen gaussians. The following overview of MLLR is adapted from a paper by Leggetter and Woodland [Legg95]. During adaptation, the mean of a gaussian, u is mapped to an unknown speaker adapted mean, $\check{u}$, using a linear regression-based transform based on the adaptation data:

$$\check{u} = Wv \tag{6.5}$$

where W is an $nX(n+1)$ transformation matrix and v is the extended mean vector $v = [1, u_1, u_2, ..., u_n]^T$. In order for the method to be effective with small amounts of adaptation data, each regression matrix is associated with a set of state distributions and estimated from the combined data. This way, distributions which were

not present in the adaptation data can be updated given what was already seen of other distributions in the same set. This also permits this method to work without supervision. After the transformation, the probability density function of a state j generating an observation vector of O of size n is

$$b_j = \frac{1}{(2\pi)^{\frac{n}{2}} |\sum_j|^{\frac{1}{2}}} e^{-\frac{1}{2}(O-W_j v_j)' \sum_j^{-1} (O-W_j v_j)} \tag{6.6}$$

We now need to estimate the MLLR matrices for each regression class. These matrices must maximize the likelihood of the adaptation data. The probability of being in state j while generating the observation sequence O at time t is then

$$\gamma_j(t) = \frac{f(O|\theta_t = j|\lambda)}{f(O|\lambda)} \tag{6.7}$$

where $f(O|\theta_t = j|\lambda)$ is the likelihood of being in state j at time t and creating the observation sequence O. $f(O|\lambda)$ is the overall probability that the model λ will generate the observation O. The matrix W_j can now be computed column by column using

$$w_i = G_i^{-1} z_i \tag{6.8}$$

where z_i is the i^{th} column of the matrix

$$Z = \sum_{t=1}^{T} \sum_{t=1}^{R} \gamma_{j_r}(t) \sum_{j_r}^{-1} o_t v'_{j_r} \tag{6.9}$$

and G_i is given by

$$G_i = \sum_{r=1}^{R} c_{ii}^{(r)} v_{j_r} v'_{j_r} \tag{6.10}$$

where $c_{ii}^{(r)}$ is the i^{th} diagonal element of the matrix

$$C^{(r)} = \sum_{t=1}^{T} \gamma_{j_r}(t) \sum_{j_r}^{-1} \tag{6.11}$$

These equations have been derived by Leggetter and Woodland in [Legg94]. The above represents one MLLR iteration. The probabilities can be further updated by using more iterations. It is also possible to alter these equations so that adaptation can be done incrementally, without an explicit reference to the time component. Now that the regression matrices are available, the regression classes must be defined. There are two basic methods to do this. The classes can either be predetermined based on the mount of adaptation data available or they can be generated dynamically. In the first case, a clustering method applied to the gaussian mixtures is required. This requires that the amount of adaptation data is known in advance. The second case, however, does not have that limitation. The mixture components are arranged in a tree structure where the leaves of the tree are actual mixtures and the internal nodes represent groupings of those mixtures. The tree is then used to get the most specific set of regression classes, given the current amount of adaptation data. By using MLLR, Leggetter reports a 40% decrease in word error rate on the Spoke S3 test for native English speakers ([Legg94]). Another method for generalizing adaptation is to use transformation parameters obtained through the maximum likelihood criterion, combined with Bayesian techniques, as is done in SRI's speech recognition system By tying mixtures together, gaussians can be adapted for which no observations have yet been made. In their paper, Digalakis and Neumeyer [Diga95] fully describe such a method of estimating parameters based on the Expectation-Maximization (EM)

algorithm [Demp77]. Once the parameters have been estimated, the transformations must be tied in order to efficiently adapt gaussians. A tree based method as described previously is sufficient for this. Bayesian techniques are now applied so that limited adaptation data can be combined with prior knowledge. This technique will only modify gaussians in a speaker independent system if they are likely to have generated adaptation data.

Chapter 7

Experiments and Results

Vector quantization of gaussians in a mixture was introduced in order to reduce the amount of computation required in order to get proper recognition with a minimal loss in accuracy. All methods applied were specially designed for Hidden Markov models which use a set of continuous distributions to represent a single transition between states

Mean adaptation was introduced to increase the recognition accuracy. As the application was for a relatively small vocabulary (approximately 500 words) in a system used by the same speaker for an extended period of time, mean adaptation was considered adequate for the purpose. The application in question is a simulated air traffic control (ATC) system.

7.1 Test and Training Sets

Three test and training sets were used in the context of these experiments. The first two sets were created at the Mcgill Speech Lab

7.1.1 ATS Task Test Set

The ATS Task Training Set (ATTS) is a set of 135 sentences with a total of 1109 labels. It contains 8 different speakers from various nationalities who at one time or another have been members of the McGill SOCS Speech Lab, thus the English used is somewhat accented. Each speaker has 17 utterances which conform to a subset of the standard international ATC grammar which is used in the demo version of Roger. The sentences given for the Personal Training Set are representative of the kinds of utterance which conform to this grammar.

7.1.2 Personal Training Set

In order to test out the speaker adaptation system, a special training set was created using a single speaker and the following sentences, which are derived from the Air Traffic Control simulation task.

```
AAL 1 11 descend and maintain flight_level 1 9 0
ACA 1 12 fly heading 1 5 0 degrees
EIN 3 13 turn left 2 4 0 degrees
AMX 3 14 increase speed 1 8 0 knots
AVA 5_15 cleared_for ils 0 4 left approach
COA 2 16 resume own navigation via solberg
DAL 4 17 affirmative 5 thousand
JAL 6 18 report reaching flight_level 2 9 0
NWA 1 19 contact n_y_center on 1 2 0 point 9
SAB 5 11 traffic 12 o'clock 5 miles northbound airbus at 1 0 thousand
BAW 7 12 roger contact tower 1 2 3 point 9
USA 1 13 climb 2 flight_level 3 5 0 altimeter 2 9 9 2
UAL 7 14 maintain heading 0 4 0
UAL 7 14 maintain heading 0 4 0
VRG 2 15 turn right heading 1 4 5
KAL 8_16 reduce speed 2 2 0
AAL 9 17 cleared_for visual 0 4 left approach
ACA 9 18 resume own navigation via J 1 7 4
EIN 4 19 affirmative flight_level 1 8 0
AMX 4 11 report reaching flight_level 1 8 0
AVA 6 12 contact n_y approach on 1 2 4 point 9
COA 3 13 traffic 7 o'clock 10 miles southbound boeing 7 50 7 at 2 7 0
DAL 5 14 roger contact tower 1 2 0 point 9
JAL 7 15 descend and maintain 1 1 thousand
NWA 2 16 fly heading 1 0 0 vector for n_y approach
```

```
SAB 6 17 turn left heading 2 0 5 degrees
BAW 8 18 increase speed 3 2 0
USA 9 19 cleared_for ils 0 4 right approach
UAL 8 11 resume own navigation via wavey
VRG 3 12 affirmative flight_level 3 9 0 ident
KAL 9 13 report flight_level 4 1 0
AAL 1 14 contact n_y_center on 1 2 3 point 9
ACA 1 15 traffic 8 o'clock 9 miles eastbound m d 80 at flight_level 3 3 0
EIN 3 16 roger contact tower 1 2 4 point 9
AMX 4 17 resume own navigation via bermuda
AVA 5 18 resume own navigation via champ
COA 2 19 resume own navigation via east_texas
DAL 4 11 resume own navigation via carmel
JAL 6 12 resume own navigation via bergh
NWA 1 13 resume own navigation via dixie
SAB 5 14 resume own navigation via j_f_k
BAW 7 15 resume own navigation via modena
USA 1 16 resume own navigation via linnd
UAL 7 17 resume own navigation via carmel
VRG 2 18 resume own navigation via coyle
KAL 8 19 resume own navigation via flann
AAL 9 11 resume own navigation via daner
EIN 4 12 resume own navigation via J 6 0
```

7.1.3 TIMIT

The TIMIT speech corpus is a database of recorded and labeled utterances which has been collected by Texas Instruments and MIT. The corpus consists of 6300 sentences, where 10 sentences were spoken by 630 speakers from 8 major dialect regions in the United States. The database is split with 70% of the recordings from males, the remaining 30% from females. For the purposes of building the VQ preprocessor's distribution lists, the TIMIT test set was used. This set consists of a group of 192 sentences and has been designed so that every phoneme is used extensively. The test set contains 24 speakers representing the 8 dialect zones.

7.2 Experiments

7.2.1 Vector Quantization Experiments

These experiments were intended to measure the amount of computation reduction and resultant accuracy loss from using various codebooks. First a codebook must be generated. These are usually created by obtaining sequences of expected data and clustering them into a desired number of nodes. In this case, data was extracted from the distribution mean vectors of the HMMs themselves. The resulting codebooks represent clusters of the HMM's distribution mean vectors. Once a codebook is generated, a special version of the recognizer is used to build up a distribution list for it. Feature vectors are pushed through the vector quantizer, then evaluated normally using HMM's. The distributions which were found to be useful in evaluating the feature vector are then associated with the codeword to which the feature vector was mapped. Figure 7.1 shows the results of applying the VQ preprocessor. The codebooks where all generated using the standard HMMs for the recognizer. The training set used to generate the list of gaussians to associate with each codeword was TIMIT and the results come from testing the system on the ATS Task Test Set. The table is organized as follows: **Size** represents the number of codewords in the VQ codebook. **Cutoff** represents the minimum number of times that a gaussian needs to have been used before it is considered for inclusion in the suggested gaussian list. **RedEval** is the average percentage reduction in the number of gaussians which had to be evaluated when compared to the baseline for the test set. **RedUsed** is the average percentage reduction in the number of gaussians which where used when compared to the baseline of the test set. **String Accuracy** represents the correct string accuracy rate over the test. In order for a string to be considered correct, all the labels which make it up must have been correctly recognized. **Unit Accuracy** is the percentage of unit labels which were correctly recognized and were in the

Size	Cutoff	RedEval	RedUsed	String Accuracy	Unit Accuracy	Percent Correct	Correct Labels	Error Labels
Baseline (no codebook)				88.15	97.57	97.93	1086	27
16	0	49.0865	51.5368	86.67	97.39	97.57	1082	29
16	25	60.2797	62.4355	85.93	97.39	97.57	1082	29
16	50	63.7942	65.8746	83.70	96.84	97.11	1077	35
16	75	65.7311	67.7333	84.44	96.93	97.20	1078	34
16	100	67.3302	69.2347	85.19	97.20	97.39	1080	31
32	0	52.9405	55.3571	86.67	97.39	97.57	1082	29
32	25	65.1098	67.0854	84.44	97.02	97.20	1078	33
32	50	67.9829	69.7624	84.44	96.93	97.20	1078	34
32	75	69.8534	71.4914	85.93	97.02	97.29	1079	33
32	100	71.1080	72.5694	84.44	96.93	97.29	1079	34
64	0	56.3389	58.6742	86.67	97.39	97.57	1082	29
64	25	68.4937	70.2329	84.44	96.93	97.11	1077	34
64	50	71.4220	72.8729	85.19	96.93	97.29	1079	34
64	75	73.0277	74.2603	82.22	96.21	96.66	1072	42
64	100	74.1801	75.2132	81.48	96.12	96.48	1070	43
128	0	59.8202	62.0545	85.93	97.29	97.48	1081	30
128	25	72.0754	73.4713	85.93	97.11	97.29	1079	32
128	50	74.4654	75.5147	84.44	96.57	96.93	1075	38
128	75	75.9559	76.7243	81.48	95.67	96.12	1066	48
128	100	76.9669	77.5000	77.78	95.22	95.58	1060	53
256	0	62.0863	64.1596	85.93	97.29	97.48	1081	30
256	25	98.5243	98.2676	00.00	00.00	00.00	0	1109
256	50	98.5243	98.2676	00.00	00.00	00.00	0	1109
256	75	98.5243	98.2676	00.00	00.00	00.00	0	1109
256	100	98.5243	98.2676	00.00	00.00	00.00	0	1109

Figure 7.1: Vector Quantization results

right position in a sentence. **Percent Correct** represents the number of correctly recognized labels which were in the right sequence but might have had incorrect labels in between the correct ones. **Correct Labels** is the total number of correctly recognized labels in the test. **Error Labels** represents the number of incorrectly recognized labels which were generated during the test. The speech recognizer can generate 3 types of errors. The first is *substitution* where one label is replaced by another. The second is *insertion* where an incorrect label is inserted somewhere in the recognition string. The last is a *deletion*, where a label is overlooked completely and thus not output. Because of these possibilities, the number of correct labels and the number of error labels do not necessarily sum to the total set of true labels in the test set. Now if we look at the table, we can analyze the performance of the preprocessor. Essentially, any codebook will reduce the amount of computation by at least 50%. If we use codebooks with a higher cutoff of gaussians, then the amount of reduction in computation increases, at a cost of some accuracy. The larger codebooks perform increasingly worse with higher cutoffs because the gaussians are more spread across each codeword, they are used less often for each entry. This explains why a codebook of 256 entries is completely useless even with a cutoff of 25. As can be seen, the selection of a proper codebook amounts to a tradeoff between accuracy and computation reduction. However, the size of the codebook itself is also an issue, as it must be searched every time an feature vector is processes. Therefore, a smaller codebook is always preferred. Increasing the cutoff reduces the number of gaussians which are in the suggested gaussian list for each codeword, thus reducing the amount of computation even more, although at a loss of some accuracy.

7.2.2 Mean Adaptation Experiments

In order to test out dynamic mean adaptation to a specific speaker, a data set containing a a large number of utterances from a single speaker is required.

Adapt Parm	String Accuracy	Unit Accuracy	Percent Correct	Correct Labels	Error Labels	Percent Improve
0.0 (baseline)	48.94	86.29	87.19	388	60	0.0
0.00625	46.81	88.31	88.99	396	52	2.34
0.01250	48.94	91.91	92.58	412	36	6.52
0.0140625	48.94	91.91	92.58	412	36	6.52
0.015625	48.94	91.91	92.58	412	36	6.52
0.01875	44.68	91.24	92.13	410	39	5.74
0.025	40.43	89.93	90.83	406	45	4.21
0.05	38.30	88.59	89.71	401	51	2.67

Figure 7.2: Automatic Speaker Adaptation results

A data set of sentences spoken by the author was used in order to test out the effectiveness of the speaker adaptation system. The first test consists simply of running the sentence list through the recognizer in order to get a base measure for the accuracy which can be achieved, then, the set is run again through the recognizer using different adaptation parameters in order to see if there is an increase or decrease in the overall accuracy.

The table in figure 7.2 shows the results of speaker adaptation. The table headings are the same as those in figure 7.1 with the addition of **Adapt Parm** which is the adaptation parameter, the α in equation 6.2. **Percent Improve** is the percentage increase in unit accuracy over the baseline given the particular adaptation parameter value. As has been mentioned in chapter 6, the goal is to find a parameter value which optimizes the accuracy by using the right amount of information from the feature vectors while keeping in mind the current state of the gaussian means in the model. In effect, one can zero in on the ideal parameter which in the case of the test run for Figure 7.2 is 0.0140625. Experiments were run with permutations of the sentence lists. In all cases, the results were similar. The improvements were almost identical to the numbers presented in Figure 7.2 and the ideal parameter remains approximately 0.014. The string accuracy of this test is low because the speaker was a francophone using American English models.

Because of the particular application, an ATC simulator, where the speaker will use it for an extended period of time, there is no generalization required for gaussians that have not yet been seen.

Chapter 8

Conclusions and Future Work

An architecture has been designed and implemented with a suitable interface for a generic speech based application called Roger. The current application of Roger is that of an interface for an Air Traffic Control (ATC) simulator. The system is designed, however, so that new applications can be quickly and efficiently designed once the required vocabulary and grammar are known. The system's multi-process stream design allows concurrent processing of data as it becomes available. This also permits more complicated applications to be built simply by adding new processes in the stream. As long as the communication protocol is known, no other knowledge of the internals of each module is required.

By adding a VQ preprocessor to the HMM based ASR system, a significant amount of computation can be removed with extremely minimal loss of accuracy. This reduction of computation can be translated into a real time speedup, thus improving the overall response time of the system. Another possibility is to give the ASR more complex or bigger models. This would increase the accuracy of the system and it would still be possible to maintain the speed as it was before the addition of the VQ preprocessor. As the VQ system currently stands, it could be further improved by exploring different internal structures for the codebook. A tree based approach to storing codewords,

for example, could make the search to find a matching codeword for a feature vector much faster.

Unsupervised speaker adaptation adds a simple yet powerful method of increasing the accuracy of the ASR system. It is especially useful in light of the demo application developed by the McGill Speech Lab, an Air Traffic Control simulator interface where a single speaker will be using the system for an extended period of time. Because of this kind of usage, a generalization system for adaptation, as described in chapter 6, was deemed unnecessary. For further development of more general applications in speech, however, the theory described should be implemented.

Bibliography

[Algo] T. H. Cormen, C. E. Leiserson, R. L. Rivest, "Introduction to Algorithms", MIT Electrical Engineering and Computer Science Series, MIT Press, Cambridge, Massachusetts, 1990

[AlpBer] Paul van Alphen, Dick R. van Bergem, "Markov Models and Their Application in Speech Recognition"

[Aust92] S. Austin, R. Schwartz, P. Placeway, "Speech Recognition Using Segmantal Neural Nets", *Proceedings of the IEEE ASSP conference*, pages I-625-628, San Francisco, March 1992

[BBN94] C. Lapre, F. Kubala, R. Schwartz, J. Makhoul, "Speaker Adaptation for Non-Native Speakers", *ARPA Spoken Language Technology Workshop*, March 8, 1994

[Bocc93] Enrico Bocchieri, "A study of the Beam-Search Alogorithm for Large Vocabulary Continuous Speech Recognition and Methods for Improved Efficiency", Proceedings of Eurospeech 1993, pages 1521-1524

[Bocc93] Enrico Bocchieri, "Vector Quantization for the Efficient Computation of Continuous Density Likelihoods", *IEEE reprint number 0-7803-0946-4/93*, 1993

[ColeEtAl] R. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, V. Zue, "Survey of the State of the Art in Human Language Technology", Draft, sponsored by the NSF and *the Directorate XIII-E of the Commission of European Communities*, August 3, 1995

[Demp77] A.P. Dempster, N.M. Laird, D.B. Rubin, "Maximum Likelihood Estimation from Incomplete Data", *Journal of the Royal Statistical Society (B)*, Volume 39, pages 1 - 39, 1977

[Diga95] V. Digalakis, L. Neumeyer, "Fast Speaker Adaptation Using Constrained Estimation of Gaussian Mixtures", SRI International, Speech Technology and Research Laboratory, Menlo Park, CA, 1995

[Dobr87] W. O'Grady, M. Dobrovolsky, "Contemporary Linguistic Analysis", 1st Edition, Copp Clark Pitman Ltd., 1987

[Dobr91] W. O'Grady, M. Dobrovolsky, "Contemporary Linguistic Analysis", 2nd Edition, Copp Clark Pitman Ltd., 1919

[Drag94] J. Orloff, J. Baker, L. Gillick, R. Roth, F. Scattone, "Speaker Adaptation", *ARPA Spoken Language Technology Workshop*, 1994

[Gall92] Michael Galler, "Improving Phoneme Models for Speaker-Independent Automatic Speech Recognition", Master's thesis, Mcgill University, School of Computer Science, October 1992

[Gold89] E. Bruce Goldstein, "Sensation and Perception", Wadsworth, Inc, 3rd edition, 1989

[Gray84] Robert M. Gray, "Vector Quantization", *IEEE ASSP Magazine*, April 1984, pages 4 - 29

[Herm91] H. Hermanski, N. Morgan, A. Bayya, P. Kohn, "Compensation for the effects of the communication channel in auditory-like analysis of speech", *Proceedings of Eurospeech '91*, pages 1367 - 1370, Genova Italy, 1991

[Herm93] H. Hermanski, N.Morgan, H.G. Hirsch, "Recognition of speech in additive and convolutional noise based on RASTA signal spectral processing", *Proceedings of ICASSP '93*, pages 83 - 86, Minneapolis, MN, 1993

[Jusc86] P. Jusczyk, "Speech Perception", *Handbook of Perception and Human Performance*, "Wiley Ltd.", 1986

[Lamp86] L. Lamport, "LATEX reference manual", Addison-Wesley Publishing Company, 1986

[Lee90] C.H. Lee, L.R. Rabiner, R. Pieraccini, J.G. Wilpon, "Accoustic Modeling for Large Vocabulary Speech Recognition", *Computer Speech and Language*, vol 4, no 2, April 1990

[Legg94] C.J. Leggetter, P.C. Woodland, "Speaker Adaptation Using Linear Regression", *Technical Report CUED/FINFENG/TR.181.*, Cambridge University Engineering Department, June 1994

[Legg95] G.J. Leggetter, P.C. Woodland, "Flexible Speaker Adaptation using Maximum Likelihood Linear Regression", Cambridge University Engineering Department, UK, 1995

[Lowe76] B.T. Lowerre, "the HARPY Speech Recognition System", PhD dissertation, Department of Computer Science, Carnagie-Mellon University, Pittsburgh, PA, 1976

[Makh95] John Makhoul, Richard Schwartz, "State of the Art in Continuous Speech Recognition", Book chapter, pages 165 - 198

[Norm91] Yves Normandin, "Hidden Markov Models, Maximum Mutual Information Estimation, and the Speech Recognition Problem", PhD Thesis, Department of Electrical Engineering, McGill University, March 1991.

[NumC] W. H. Press, B. P. Flannery, S. A. Teukolsky, W.T. Vetterling, "Numerical Recipes is C - The Art of Scientific Computing", Press Syndicate of the University of Cambridge, 1988

[Oust93] John Ousterhout, "Tk", Motif-like toolkit for Tcl, freeware software extension for tcl available from ftp://ftp.cs.berkeley.edu/ucb/tcl/tk3.6.tar.Z, Version 3.6, University of California, November 23, 1993

[Oust94] John Ousterhout, "Tcl - Tool Command Language", freeware software package available from ftp://ftp.cs.berkeley.edu/ucb/tcl/tcl7.3.tar.Z, Version 7.3, University of California, June 25, 1994

[Pico90] Joseph Picone, "Continuous Speech Recognition Using Hidden Markov Models", *IEEE ASSP Magazine*, July 1990, pages 26-41

[Pico93] Joseph W. Picone, "Signal Modeling Techniques in Speech Recognition", *Proceedings of the IEEE*, Volume 81, No. 9, pages 1215 - 1247, September 1993

[Rab89] L. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", Proceedings of the IEEE, Volume 77, No. 2, pages 257 - 286, February 1989

[Rab93] L. Rabiner, B. Juang, "Fundamentals of Speech Recognition", PTR Prentice-Hall Inc., 1993

[Rena92] S. Renals, N. Morgan, M. Cohen, H. Franco, "Connectionist Probability Estimation in the Decipher Speech Recognition System", *Proceedings of the IEEE ASSP conference*, pages I 601-603, San Francisco, March 1992

[Ricc93] Enrico Bocchieri, Giuseppe Riccardi, "Use of the Forward-Backward Search for Large Vocabulary Recognition with Continuous Observation Density HMM's", 1993 IEEE ASR Workshop

[Saga94] Jun-Ichi Takahashi, Shigeki Sagayama, "Telephone Line Characteristic Adaptation Using Vector Field Smoothing Technique", *Proceedings of the ICSLP conference*, pages S18-2.1 - S18-2.4, Yokohama, 1994

[Shau87] D. O'Shaughnessy, "Speech Communication - Human and Machine", Addison-Wesley Series in Electrical Engineering: Digital Signal Processing, Addison-Wesley Publishing Company, 1987

[Shin94] Koichi Shinoda, Takao Watanabe, "Unsupervised Speaker Adaptation for Speech Recognition Using Demi-Syllable HMM", *Proceedings of the ICSLP conference*, pages S09-2.1 - S09-2.4, Yokohama, 1994

[Taka94] K. Takagi, H. Hattori, T. Watanabe, "Speech Recognition with Rapid Environment Adaptation by Spectrum Equalization", *proceedings of the ICSLP conference*, pages S18-10.1 - S18-10.4, Yokohama, 1994

[Vite67] A.J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", *IEEE Transactions on Information Theory*, Volume 13, number 2, April 1967

[Zava95] G. Zavaliagkos, R. Schwartz, J. Makhoul, "Adaptation Algorithms for BBN's Phonetically Tied Mixture System", BBN Systems and Technologies, Cambridge, MA, 1995