MULTI-TRANSPUTER BASED ISOLATED WORD SPEECH RECOGNITION SYSTEM

By

Francis Cho-yiu Chik

(戚祖耀)

A Thesis

submitted in partial fulfillment of the requirements

for the Degree of Master of Philosophy

Division of Electronic Engineering

The Chinese University of Hong Kong

October 1996

Acknowledgement

Many people have made invaluable contributions to the successful completion of this project. I would like to take this opportunity to express my sincere thanks to their kindness and guidance. I wish to express my appreciation to Dr. T Lee, Dr. HC So, Mr. WK Lai and Mr. WK Lo who have shared their knowledge expertise and valuable experience with me. Especially, thanks to Dr. T. Lee for providing the source of Cantonese speech samples.

Finally, I would like to thank my supervisor, Dr. SM John Chiang, for his support and guidance in the past. Besides, I would like to gratefully acknowledge Prof. PC Ching for his support and encouragement to finish the research work.

Abstract

Automatic speech recognition has been a popular research topic. Both recognition accuracy and speed are the major concern. In our studies, template matching and statistical models for isolated word speech recognition are investigated. While template matching based on Dynamic Time Warping (DTW) can solve the problem of mis-alignment between two speech utterances, statistical model of Hidden Markov Model (HMM) has proved to be a more powerful tool for automatic speech recognition. In an HMM based system, the model topology and the input acoustic features affect the performance the most. 2-dimensional features have been explored with an aim to develop a new model, 2-dimensional HMM (2dHMM), with added flexibility and control for an isolated-word speaker-dependent speech recognition system. Short-time spectral information and tone (pitch information) are used as features in two different dimensions and the system is tested with Cantonese speech.

Experiments have been conducted to investigate the performance of the proposed 2dHMM speech recognition method. Comparisons have also been made among DTW, HMM and 2dHMM with 40 English words selected from TI-46 which is a speaker-dependent isolated word corpus, published by Texas Instruments. It is found that accuracy performances of HMM and 2dHMM based recognizers are better than one of DTW. By using a 20-English-word vocabulary selected from the TI-46 isolated word database, recognition accuracy tests are conducted for HMM and 2dHMM based recognizers only. The performances between them are similar from the results. By using Cantonese language, tests are conducted for recognition with tone feature. 5 pairs of utterances with same phonemes (Initial and Final) but tone are used to build the vocabulary. The result shows that 2dHMM based recognizer has rather good capability to correctly recognize these confused words. Further, the vocabulary of 80 Cantonese syllables are used for evaluation. Besides words with the same syllable but tone do not easily recognized correctly, it is observed that words with the only difference in the Final of the syllable are also easily mis-recognized.

A new architecture of HMM is proposed which is basically an 2dHMM and it models speech signals on two distinctive sets of features. Extensive experiments have been conducted to study the performance of the proposed 2dHMM speech recognition system using both English and Cantonese syllables. By using the spectral and pitch information for Cantonese language as the two feature sets, the 2dHMM speech model achieves an improvement in terms of recognition accuracy. Based on a transputer multi-processing platform, a farming model is developed for the implementation of the recognition algorithm. It is found that the model may be expanded to suit for a larger vocabulary without the modification on program codes in system design. It is expected the 2dHMM together with the farming model provides a platform for further exploration on feature type variations or even a Multi-dimensional HMM (MdHMM) system.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Automatic speech recognition and its applications

Automatic speech recognition is one of the more fascinating research areas today. It will no doubt be integral to many potential applications in which speech is used as direct command. In military application, it will be helpful for a pilot to control the aircraft by voice in hands/eyes-busy situations [1, 2, 3]. In such circumstances, pilots can perform missions, such as formation flying or low-level navigation, faster and more accurately when using spoken control over various avionics subsystems, as compared with keyboard and multifunction-button data entry. On the other hand, it will also be useful to assist the disables to manage simple tasks, such as controlling the movement of an electric wheel-chair [4]. This kind of wheel-chair could help those motion disable persons who cannot drive it manually nor with a joystick and enable them to enjoy a higher quality of life. In survey, man-machine interaction by voice is much preferred by many computer users. Mobile phone users are also able to enjoy voice dialing through a special service provided by the base station [5]. This may decrease the hazards of using mobile phones during driving since voice dialing simplifies their use. The MTR (Mass Transit Railway) Telephone Hot Line Service in Hong Kong has already incorporated a speech recognizer in their voice control enquiry system. There are many other examples that show speech recognition is moving to an important position in man-machine interaction technologies.

Speech recognition involves three processes as shown in Figure 1.1. They are extraction of acoustic parameters from the speech signal, calculation of matching score between the unknown utterance and an utterance in the vocabulary, and determination of the recognized utterance via a search among all possibilities. The first process

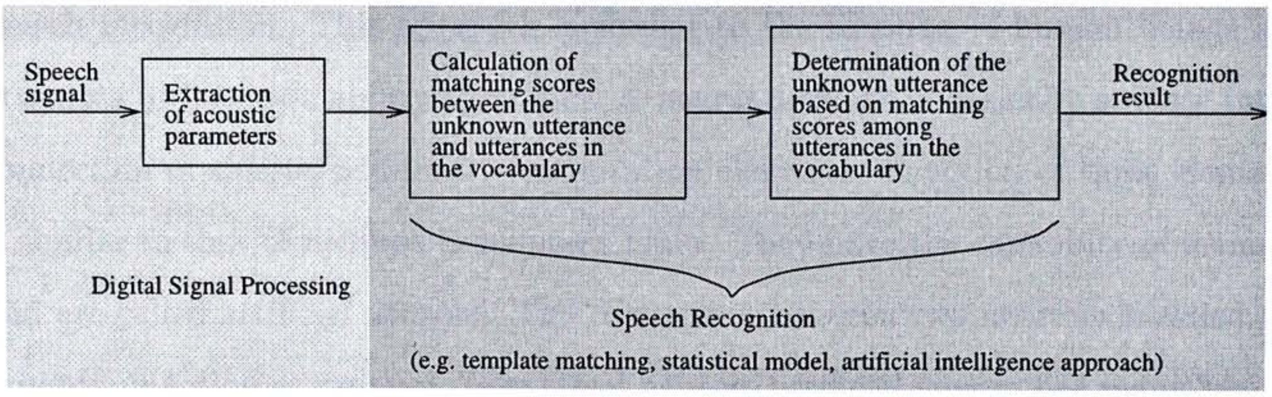
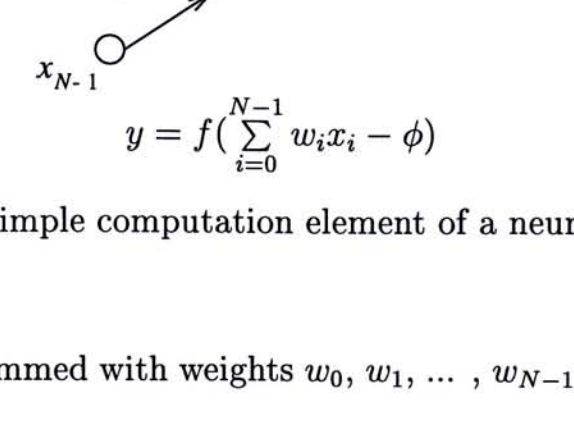

Speech signal
Extraction of acoustic parameters
Calculation of matching scores between the unknown utterance and utterances in the vocabulary
Determination of the unknown utterance based on matching scores among utterances in the vocabulary
Recognition result
Digital Signal Processing
Speech Recognition
(e.g. template matching, statistical model, artificial intelligence approach)

Figure 1.1: Processes involved in speech recognition.

deals with the properties of acoustic signal. In this stage, signal modeling techniques in parameterization of an analog speech signal are considered. These algorithms are intended to produce a "perceptually meaningful" parametric representation of the speech signal where the parameters emulate some of the behavior observed in the human auditory and perceptual systems. Of course, and perhaps more importantly, these algorithms are also designed to maximize recognition performance [6]. Based on the extracted parameters, the second and third processes follow and deal with the recognition method. Some existing recognition methods are template matching with Dynamic Time Warping (DTW) [7, 8, 9, 10], statistical model (Hidden Markov Model – HMM) [11, 12, 13] and artificial intelligence approaches (neural networks) [14, 15]. Before performing the job of recognition, training is always required. The training may involve a huge amount of computation, so in most case it is designed to be done off-line. In spite of considerable research, neural networks have not yet shown significantly better performance than HMM algorithms [16]. In this thesis, we do not intend to study neural network based speech recognition algorithms. However, for completeness, the fundamental concept of these methods are briefly introduced.

1.1.1 Artificial Neural Network (ANN) approach

Artificial Neural Network (ANN) is a kind of artificial intelligence approaches to speech recognition. This approach is similar to the learning of human beings and attempts to have the ability like them. A neural network is basically a dense interconnection of simple, nonlinear, computation elements. Function of these elements is similar to that of neurons in human's brain. They have the capability of memory and are called artificial neurons. The connection between two neurons is called the synaptic junction or synapse. Like a real neuron, artificial neuron has many inputs, but has only a single output, which can fan out to many other artificial neurons in the network. Figure 1.2 shows a simple computation element of a neural network and the function $f(x_i)$ is nonlinear. It is assumed that there are N inputs (x_0, x_1,



$$y = f\left(\sum_{i=0}^{N-1} w_i x_i - \phi\right)$$

Figure 1.2: Simple computation element of a neural network.

... , x_{N-1}) which are summed with weights w_0, w_1, ... , w_{N-1}, threshold ϕ, and then nonlinearly compressed to give the output y.

$$y = f(\sum_{i=0}^{N-1} w_i x_i - \phi), \tag{1.1}$$

where ϕ is an internal threshold or offset, and $f(x_i)$ is a nonlinearity of one of the following types:

1. hard limiter

$$f(x) = \begin{cases} +1, & x \leq 0 \\ -1, & x < 0 \end{cases} \tag{1.2}$$

or

2. sigmoid functions

$$f(x) = \tanh(\beta x), \quad \beta > 0 \tag{1.3}$$

$$f(x) = \frac{1}{1 + e^{-\beta x}}, \quad \beta > 0. \tag{1.4}$$

The sigmoid nonlinearities are most commonly used because their continuity and differentiability make mathematical analysis feasible.

For the whole network, the relative information transmitted between different artificial neurons can be reflected by a quantity called a weight (or connection strength). The weight on the connection from ith neuron to jth neuron is denoted by w_{ij}. The output y of a neuron corresponds to the firing frequency of it. It is complex and meaningful for the network to be built by large amount of individual computation element, although each element is simple. The weights in the network are obtained through learning. There are three standard and well known topologies. They are single/multilayer perceptrons, Hopfield (recurrent) networks and Kohonen (self-organizing) networks. Due to the nonlinearity in the network, the overall performance is not just like a simple addition among elements. This is one of the characteristics of a human's brain. The other property of ANN is that the whole network will not be serious affected by failure of small amount of computation elements. It likes brain's cells, they die everyday and would not affect the operation of the brain. An example of ANN model for speech recognition can be found in Appendix A.

1.2 Motivation

The purpose of this project is to explore the different isolated word speech recognition techniques and to develop an efficient and effective system with emphasis on both recognition accuracy and speed. Given the existing techniques, there are numerous ways to implement a speech recognition system. Since HMM has a solid theoretical basis and offers practical advantages, the work of this thesis will be based on this method. It is understood that the performance of an HMM based system depends on many factors, two of them are the topology of the models and the acoustic parameters. A good choice of model architecture and feature set can sensibly improve the recognition performance of the system [17]. We believe that it will be more effective if the model can accept two sets of features in different domains simultaneously. This approach will be explored in greater detail. A 2dHMM speech model is proposed and it is tested specifically with Cantonese speech. It is known that the dialect of Cantonese is a monosyllabic and tonal language. Since Cantonese has nine lexical tones, it is expected that this property could be used as a kind of acoustic feature to be modeled by the 2dHMM.

Performance of the speech recognition algorithm is one of the main concerns. Most traditional methods of speech recognition can be implemented in real time by a single processor [18]. Recently, recognition algorithms, such as HMM, have become so complex and computational intensive that a single processor system cannot achieve real time performance. By using several processors working together in a parallel processing fashion, it can theoretically increase the processing power in many folds thereby meeting the real time requirement. With the multi-processor environment, research works [19, 20] are carried out for the improvement of recognition time.

In our work, we also explore the application of a transputer based multi-processor system for performance improvement of our speech recognition scheme.

1.3 Background

1.3.1 Speech recognition

Research in automatic speech recognition by machine has been going on for over four decades. It is the process by which a computer maps an acoustic speech signal to text or to some form of abstract meaning of the speech. Basically, there are three different speech recognition systems. They are speaker dependent system, speaker independent system and speaker adaptive system. A speaker dependent system is developed to operate for a single speaker or within a group of designated speakers. A speaker independent system is developed to operate for any speaker. A speaker adaptive system is developed to adapt its operation to the characteristics of new speakers.

The complexity, processing requirements and the accuracy of an automatic speech recognition system depends greatly on the size of vocabulary. Some applications only require a few words (e.g. numbers only), others require very large dictionaries (e.g. dictation machines). Tens of words can be classified as small vocabulary. Hundreds of words can be classified as medium vocabulary. Thousands of words can be classified as large vocabulary.

Speech signals are generally classified as either an isolated words or continuous speech. An isolated-word recognition system operates on words which are separated by pauses. Unlike an isolated-word system, there are no pauses between single words in a continuous speech system. It operates on speech in which words are connected together. An isolated-word system is the simplest form of recognition because the occurrences of words are more consistent, while a continuous speech system is more complicated because of a variety of effects, such as production of each phoneme is affected by the production of surrounding phonemes, and similarly the start and end of words are affected by the preceding and succeeding words.

1.3.2 Parallel processing

Most existing speech recognition algorithm are very sophisticated and involve extensive computation. In order to enable real-time processing, we either design more computationally efficient algorithm or we have to resort to more powerful engine with parallel processing capability.

An instruction stream is a sequence of instructions as executed by the machine; a data stream is a sequence of data including input, partial, or temporary results, called for by the instruction stream [21]. In general, digital computers may be classified as:

- Single-Instruction, Single-Data computer (SISD)
- Single-Instruction, Multiple-Data computer (SIMD)
- Multiple-Instruction, Multiple-Data computer (MIMD)

Thus the computer shown in Figure 1.3 is called a Single-Instruction (because there is only one central processing unit (CPU) which executes one instruction at a time), Single-Data (because there is only one data store and one channel by which data is retrieved) SISD computer. SIMD (shown in Figure 1.4) and MIMD (shown in Figure 1.5) can be classified as a parallel processing.

The SIMD organization is the classic form of an array processor. All processing elements which controlled by a master controller are identical. The communication between processors is typically limited to nearest-neighbor links and is also controlled by the master controller. All processors in the array operate synchronously (i.e. they execute the same instruction at the same time).

Each processor of MIMD machines has its own independent instruction and data stream. No restrictions are placed on the synchronization of the separate processors in an MIMD configuration. Moreover, the communication between processors is defined as minimal. Almost all multiprocessor systems in existence today are classified as MIMD machines.

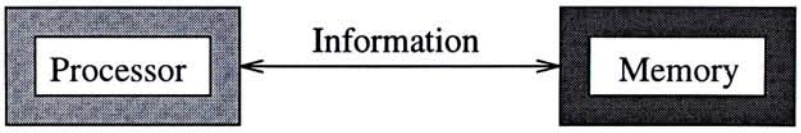


Figure 1.3: The three parts of an SISD computer are a CPU, memory, and a communication channel through which information passes.

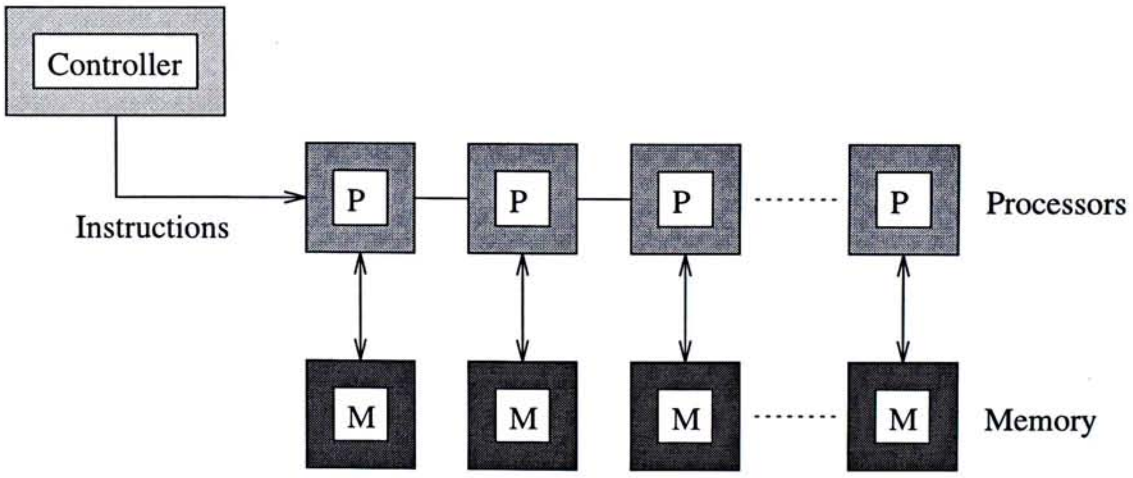


Figure 1.4: An SIMD computer executes one instruction at a time using many simple processors to process many data elements at once.

The performance measurements are important for the effective use of any parallel computer [22]. These measurements are *efficiency* and *speedup*. The efficiency of a program running on a multi-processor system is:

$$\text{Efficiency} = \frac{\text{Time on a single processor}}{\text{Time on parallel processor} \times \text{Number of processors}}$$

The speedup of a parallel processing is the ratio of its processing time to that of a

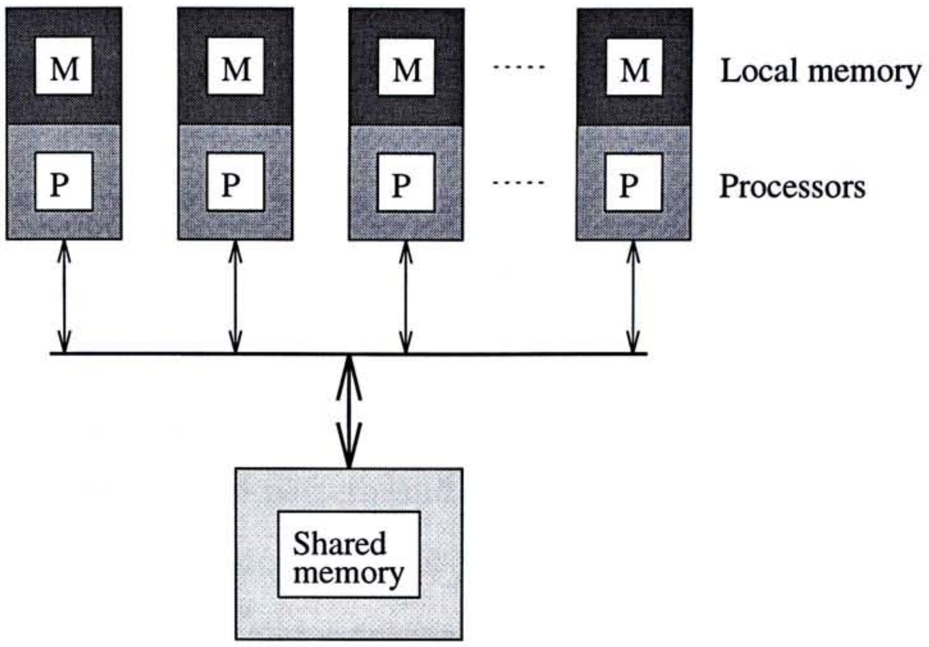


Figure 1.5: A practical shared-memory MIMD computer with a small local memory for data cache and program storage as well as a common memory.

single processing.

$$\text{Speedup} = \frac{\text{Time on a single processing}}{\text{Time on a parallel processing}}$$

On the other hand, the *speedup* of a parallel processing compared to a single processing is equal to the *efficiency* of the parallel processing times the number of processors in it.

There are two different types of architectural models in a multiprocessor system. One is a *tightly coupled* system and the other is a *loosely coupled* system. Tightly coupled systems communicate through a shared main memory. For loosely coupled systems, they do not generally encounter the degree of memory conflicts experienced by tightly coupled systems. However, the degree of coupling in such a system is very loose. Loosely coupled systems are usually efficient when the interactions between tasks are minimal. Tightly coupled systems can tolerate a higher degree of interactions between tasks without significant deterioration in performance. If high-speed or real-time processing is desired, tightly coupled multiprocessors may be used since loosely coupled multiprocessor may be too low for some applications that require fast response times [21].

1.3.3 Parallel architectures

There are three issues that must be considered for a parallel architecture. They are the granularity of the processing elements, the topology of the interconnections between processing elements and the distribution of control across the processing elements. Granularity refers to the power of each processing element in the architecture ranging from many single-bit processors to a few powerful general purpose ones. Topology refers to the pattern and density of the connections that exist between the processing elements. Control distribution is concerned with allocating tasks to processing elements and synchronizing their interactions. The parallel computer architecture space with these three variables as the axes is illustrated in Figure 1.6. Referring to this computer architecture space, the relative perspective of various

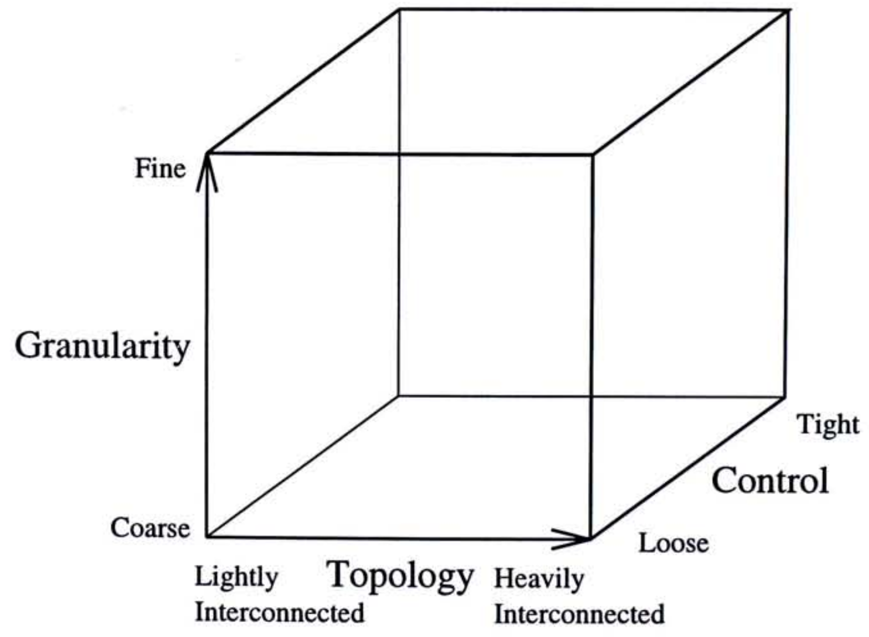


Figure 1.6: Organizational space of parallel systems.

architectures can be illustrated by showing their approximate position within the space. The criteria used to position the architectures are somewhat subjective and qualitative, because the various architectures are often so different in their structure and operations that a one-to-one comparison of their features is virtually impossible [23].

The organizational space that array processors of SIMD machines are situated in is shown in Figure 1.7. The granularity of the processing elements ranges from single-bit to whole-word, and their instruction set typically does not include program control instructions or sophisticated memory addressing mechanisms. The processing elements are considered finely grained processors. The topology of interconnection between the processing elements is fairly dense and highly synchronized. The operation of all processing elements is synchronized so that a single operation is performed on all data at once. The control for this action resides within the control unit at all times and is always synchronized with the main processor clock. Similarly, data transfers between processing elements must be orchestrated in such a way that all processing elements release and receive their data simultaneously to avoid access conflicts. Therefore, array processor control is defined tight.

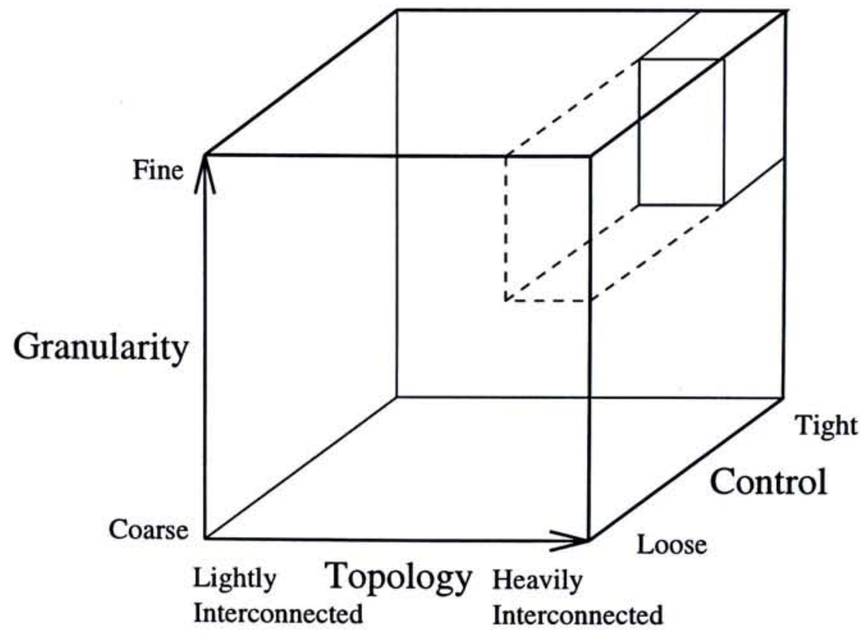


Figure 1.7: Array processor region in the organizational space.

The organizational space of interconnected network processors which may be in the form of SIMD-type operation or MIMD-type operation is shown Figure 1.8. Granularity ranges from fairly simple fine-grained processors to more sophisticated general-purpose processors. The topology is relatively lightly interconnected and each processor directly connects to at most several switches. Control in these types of systems varies greatly, from SIMD-type operation with synchronized data transfers between network stages to MIMD-type operation where the interconnections are packet-switched and communication occurs asynchronously.

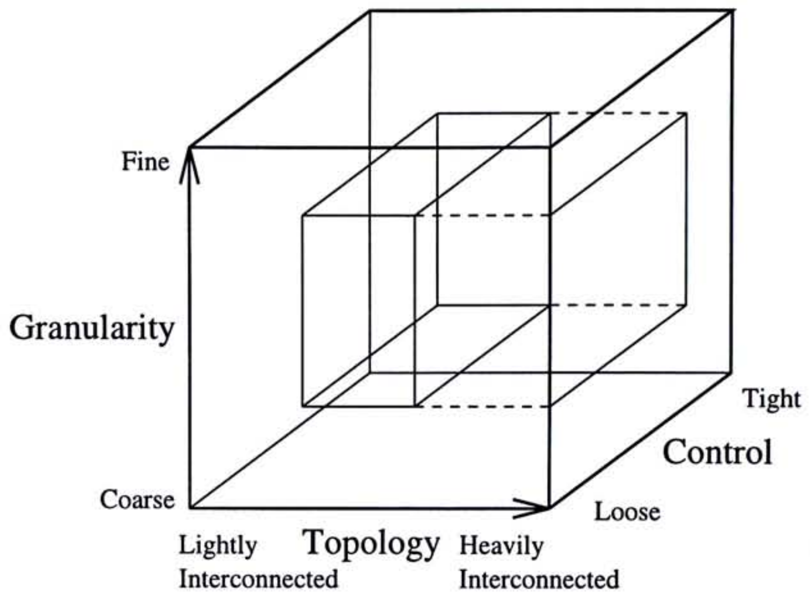


Figure 1.8: Organizational space of parallel systems.

1.3.4 Transputer

The transputer is essentially a single-chip microcomputer with a simple but powerful microprocessor, RAM, and input/output circuitries including four high speed serial communication channels. By using these channels, a number of transputers may be interconnected to form a composite system. All the channels of a transputer can operate simultaneously, unlike a system using a bus to link together several processors in which the speed of communication is limited by the overall capacity of the connecting bus. Since each transputer contains its own program counter and executes from its own local memory, parallel systems built using transputers are inherently MIMD machines.

1.4 Thesis outline

The thesis is organized into six chapters. Chapter 2, Speech Signal Pre-processing, explains the techniques used in the design of an isolated word speech recognition system. It describes acoustic pre-processing of speech signal including end-point detection and extraction of features. Chapter 3, Speech Recognition Methods, describes three types of speech recognition methods which are template matching with Dynamic Time Warping (DTW), Hidden Markov Model (HMM) and 2-dimensional Hidden Markov Model (2dHMM). Specifically, the theory and topology of 2dHMM are introduced. Chapter 4, Implementation, describes the system implementation for performance evaluation of the recognition algorithm. The properties of transputers including the architecture to be adopted in our implementation are described. The recognition algorithm has been decomposed into various sub-levels such that they can be processed in parallel simultaneously. Chapter 5, Experimental Result, presents the results of the experiments which illustrate the performance of various speech recognizers. The final chapter gives discussions and conclusions drawn from the project.

Chapter 2

Speech Signal Pre-processing

2.1 Determine useful signal

Spoken speech, even if the same word by the same speaker, can vary according to circumstances, mood, and environmental effects [24]. It is known that people tend to speak in different rates. For automatic speech recognition purpose, the difference in time must be aligned somehow. In addition, if the environment is noisy, then we need to determine when the word starts and finishes. Furthermore, the variation in amplitude would need to be normalized, otherwise we may find that speech recognition systems will only respond to the loudness of the speech rather than to the actual content of the speech signal.

To recognize spoken utterance, end-point detection is necessary to isolate the speech of interest from background noise for further processing (i.e. to be modeled or recognized). In practice, it is not easy to detect the uttered signal accurately so as to provide the "best" speech patterns for recognition. In an ideal case (i.e. no additive noise), the method used for determining the endpoints of isolated utterances is to consider the energy. The energy of the lowest-level speech sounds such as weak fricatives should still exceed the background noise energy and therefore can serve as a distinctive feature.

In search for a suitable algorithm for end-point detection, a scheme proposed by *L. R. Rabiner et al.*, 1975 [25] was adopted. Figure 2.1 shows the flow diagram of the endpoint-location algorithm. This method for end-point detection evaluates the energy and zero crossing rate of a frame. Comparing with other frames, useful signal frame is expected to contain higher energy. Zero crossing rate is auxiliary to locate the start of a week sound such as fricative (e.g. f, v, θ are fricatives) and plosive (e.g. /t/ and /p/ in "top").

2.1.1 End point detection using energy

Suppose a sampling frequency of 10 kHz and a window size (or a frame size) of 25.6 ms with 256 sampling points are used. The energy of a frame is defined as

$$E(n) = \sum_{i=0}^{255} |s(n+i)|, \tag{2.1}$$

where $s(n)$ are the sampled speech signals. To save computation time, a magnitude function is considered rather than a squared magnitude function. Further, the use of a magnitude can de-emphasize amplitude variations of speech. Therefore, compared with squared magnitude function, magnitude function can produce a smoother energy function. Having computed the energy function for the entire interval, $E(n)$, the peak energy (IMX) and the silence energy (IMN) are used to set the lower threshold (ITL) and the upper threshold (ITU).

$$I1 = 0.03 \times (IMX - IMN) + IMN \tag{2.2}$$

$$I2 = 4 \times IMN \tag{2.3}$$

$$ITL = \min(I1, I2) \tag{2.4}$$

$$ITU = 5 \times ITL. \tag{2.5}$$

$I1$ and $I2$ are parameters for the lower threshold in Equation (2.4). $I1$ is a level which is 3 percent of the peak energy (adjusted for the silence energy), whereas

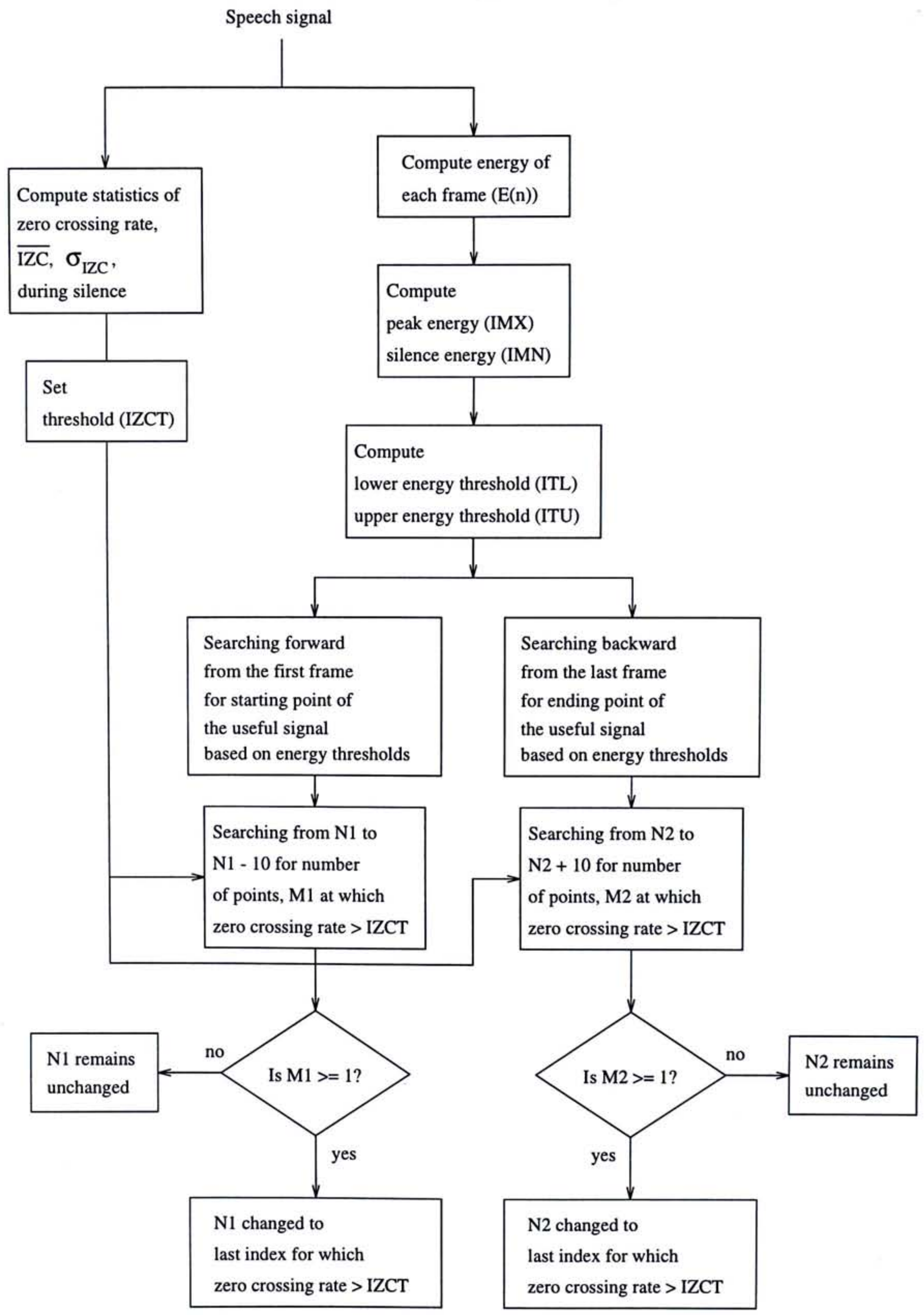

Speech signal
Compute statistics of zero crossing rate, $\overline{IZC}$, σ_{IZC}, during silence
Set threshold (IZCT)
Compute energy of each frame (E(n))
Compute peak energy (IMX) silence energy (IMN)
Compute lower energy threshold (ITL) upper energy threshold (ITU)
Searching forward from the first frame for starting point of the useful signal based on energy thresholds
Searching backward from the last frame for ending point of the useful signal based on energy thresholds
Searching from N1 to N1 - 10 for number of points, M1 at which zero crossing rate > IZCT
Searching from N2 to N2 + 10 for number of points, M2 at which zero crossing rate > IZCT
Is M1 >= 1?
Is M2 >= 1?
no
no
yes
yes
N1 remains unchanged
N2 remains unchanged
N1 changed to last index for which zero crossing rate > IZCT
N2 changed to last index for which zero crossing rate > IZCT

Figure 2.1: Flow diagram of endpoint detection algorithm.

$I2$ is a level set at four times the silence energy. The lower threshold (ITL) is the minimum of these two conservative energy thresholds, and the upper threshold (ITL) in Equation (2.5) is five times the lower threshold.

To make a first guess of the start point and the end point, the algorithms shown in Figure 2.2 and Figure 2.3 are employed. The algorithm to locate the start point

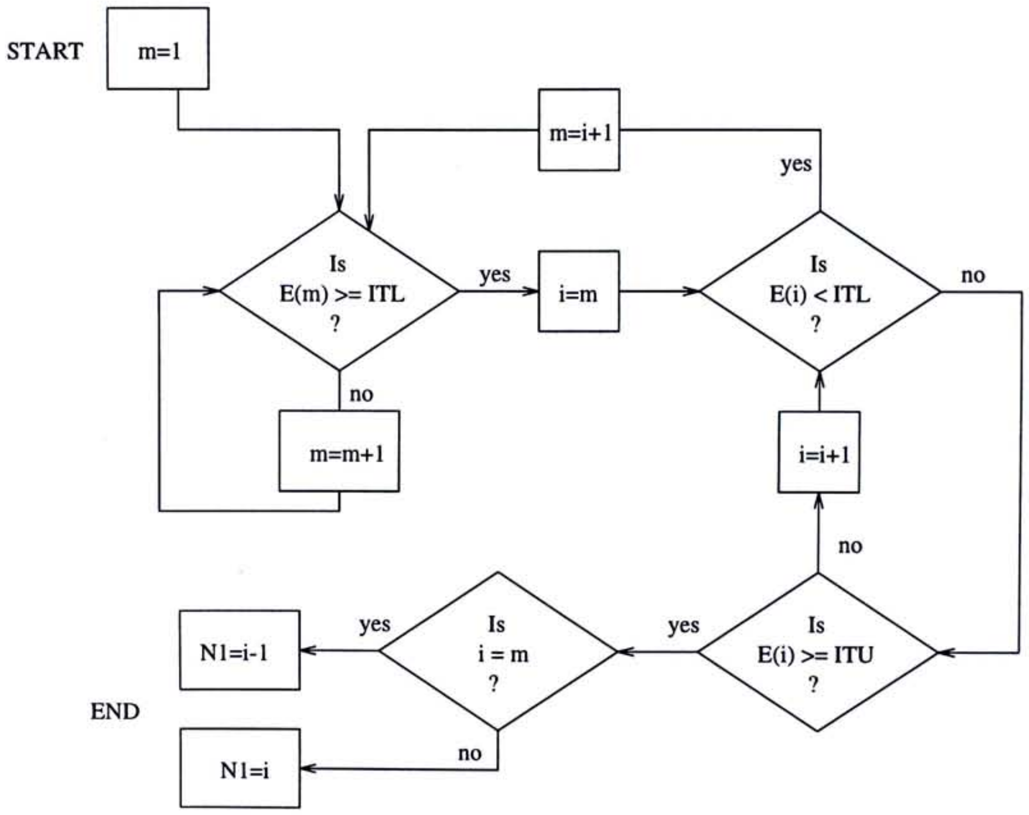


Figure 2.2: Estimation of the start point of a speech signal based on energy criterion.

begins by searching from the beginning of the interval until the lower threshold is exceeded. This point is preliminarily labeled as the beginning of the utterance unless the energy falls below ITL before it rises above ITU. These beginning and ending points are called $N1$ and $N2$ respectively.

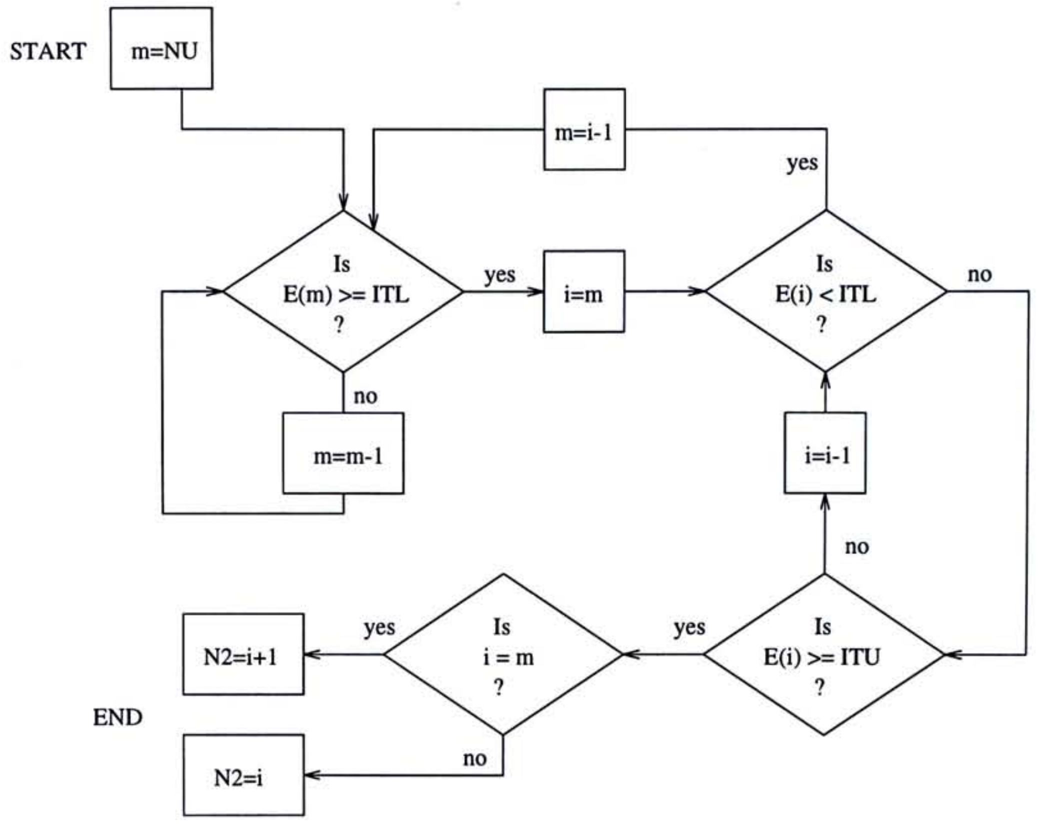


Figure 2.3: Estimation of the end point of a speech signal based on energy criterion.

2.1.2 End point detection enhancement using zero crossing rate

For unvoice sound, the signal energy would be small, therefore the energy criterion cannot accurately locate the end points of a speech segment. To alleviate the problem, the measurement of zero crossing rate may be used to enhance detection.

The zero crossing rate of a speech signal is defined as the number of zero crossings in a fixed time interval (25.6 ms). In most cases, it is a good way to indicate the presence or absence of unvoiced speech. It is assumed that the first 4 frames (102.4 ms) of the utterance do not contain speech. The statistics of the background silence can be measured during this interval. The average zero crossing rate $\overline{IZC}$ and standard deviation σ_{IZC} of the zero crossing rate are recorded and zero crossing threshold is then calculated by Equation (2.6).

$$IZCT = \min(FT, \overline{IZC} + 2\sigma_{IZC}) \tag{2.6}$$

where $IZCT$ is a zero crossing threshold and FT is a fixed threshold. The fixed threshold is set to be 63 crossings per 25.6 ms in our system. It is used as an upper bound preventing the threshold being set too large. Figure 2.4 illustrates the endpoint detection using energy criterion and enhanced with the use of zero crossing rate. The new beginning and ending points are $N1'$ and $N2'$.

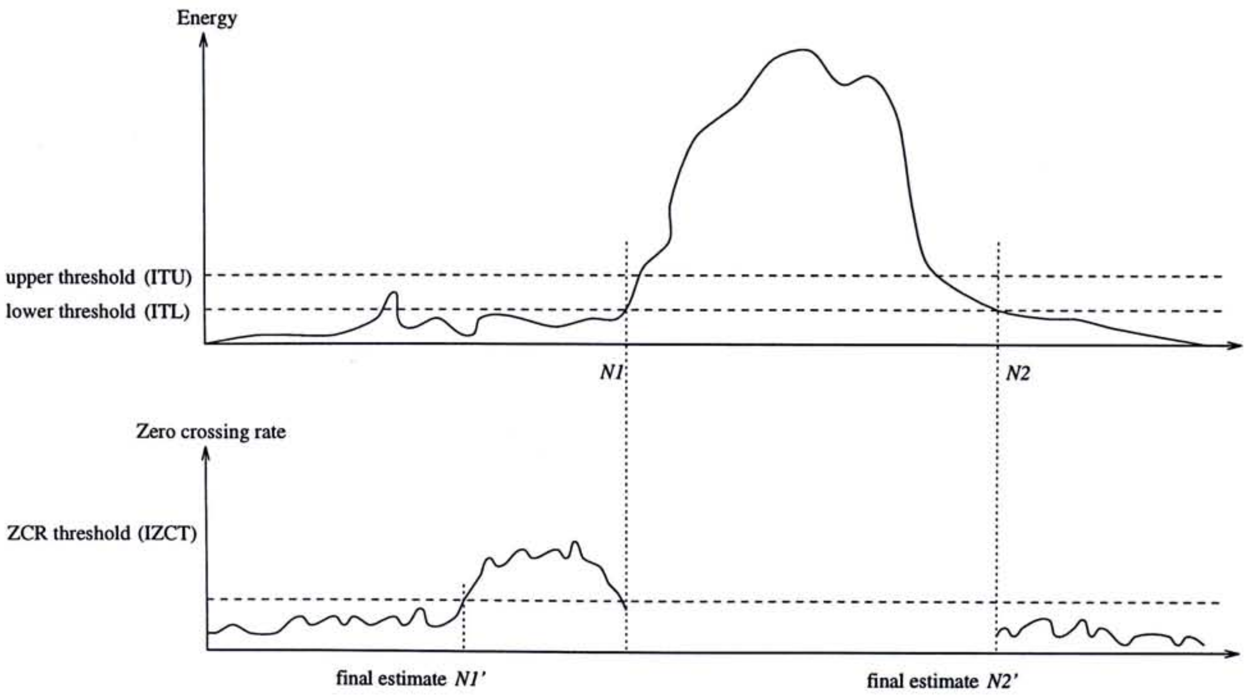


Figure 2.4: Endpoint detection using both energy and zero crossing rate of a sampled speech signal.

2.2 Pre-emphasis filter

To process the voice signal, it must first be converted into a discreted signal by transducer (microphone) and analog-to-digital converter. During analog-to-digital conversion, the microphone usually introduces undesired side effects, such as line frequency noise (50-Hz hum), loss of low- and high-frequency information, and non-linear distortion [6].

Speech samples used in this project can be obtained either from a pre-recorded CD-ROM through a data acquisition device — GRADIENT or an analog-to-digital TRAM (transputer module). For processing of human speech, 10 kHz sampling frequency (5 kHz bandwidth) is acceptable. After digitizing the signal, digital post-filtering is most often executed using a pre-emphasis filter. The motivation is to spectrally flatten the speech signal and to amplify important areas of the spectrum. Normally, the following filter function is used

$$H(z) = 1 + az^{-1} \tag{2.7}$$

where a should be [-1.0, -0.4]. The frequency responses of pre-emphasis filters are plotted in Figure 2.5.

2.3 Feature extraction

Having determined the useful signal from a speech waveform, as described in Section 2.1, a wide range of possibilities exists for parametrically representing the speech signal. This process is also called as feature extraction. Extraction of acoustic parameters from the speech waveform is a necessary step for all speech recognition systems.

In our work, we have identified a number of parameters to represent the speech signal, namely, spectral amplitude, LPC coefficients, cepstral coefficients, zero crossing rate, energy and pitch. The cepstrum analysis is considered as probably the most important parametric representation of speech and it is one of the dominant methods for speech recognition. Some examples of speech waveforms and their corresponding spectrograms are shown in Appendix B. It can be seen that similar speech waveforms can give quite distinct frequency plot in the spectrogram. The tone feature is

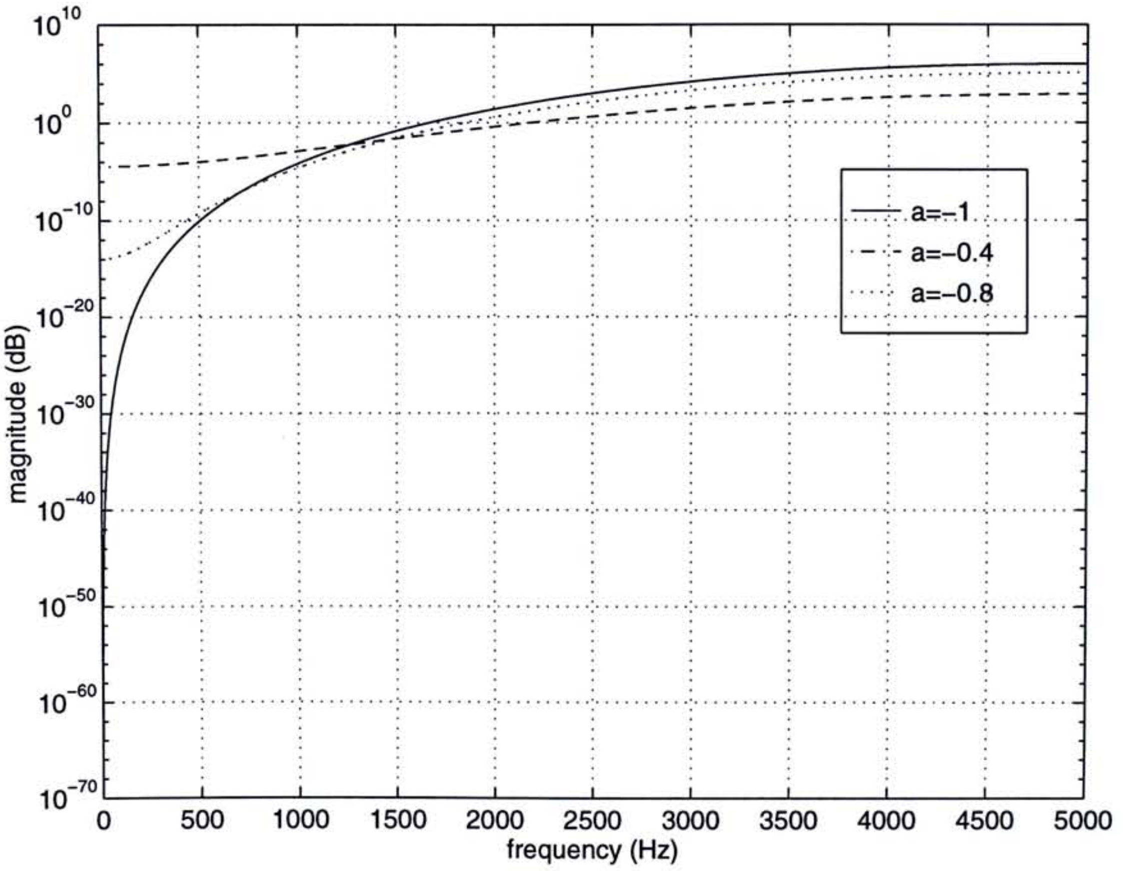


Figure 2.5: The frequency responses of common pre-emphasis filters.

used for tonal languages such as Cantonese and Mandarin. For the language of Cantonese, it is a monosyllabic and tonal language. Therefore tone feature is absolutely an essential element to be modeled for Cantonese speech.

Feature extraction may be considered as a kind of data compression technique. It reduces a large number of data to a few parameters. In our prototype system, spectral information is used as the main features. The speech waveform is a non-stationary signal. However, if a short segment (frame) of the signal is taken, it is assumed to be stationary. Thus, a speech segment can be represented by a set of model parameters. To deal with the non-stationary aspects of the signal, we will track the time variation of the model parameters of each small segment of speech. In order to avoid abrupt change between frames, overlapping segments are taken. To

2.3.1 Filter-bank spectrum analysis model

Fourier transform derived filter bank amplitudes is one of the spectral analysis methods [6] of speech signal. The discrete Fourier transform (DFT) [26] of a speech signal is defined as

$$S(f) = \sum_{n=0}^{N-1} s(n)e^{-j\omega nf}, \qquad f = 0, 1, ..., N-1 \tag{2.8}$$

where $\omega = \frac{2\pi}{N}$ and N is the number of sampled point. The rectangle window is used for each frame on taking DFT. Features are extracted by short time FFT (which is a fast algorithm of DFT) on every N sampled points with sampling frequency, f_s kHz (folding frequency is $\frac{f_s}{2}$ kHz) as a frame. After end-point detection, total number of sampled points are truncated to the multiple of N sampled points. Features of each frame are obtained by using FFT as shown in Figure 2.6. The feature vector contains 12 elements,

$$\text{frame feature} = (v_1, v_2, v_3, ..., v_{12}) \tag{2.9}$$

For each frame in the speech signal, spectral characteristics are extracted as distinc-

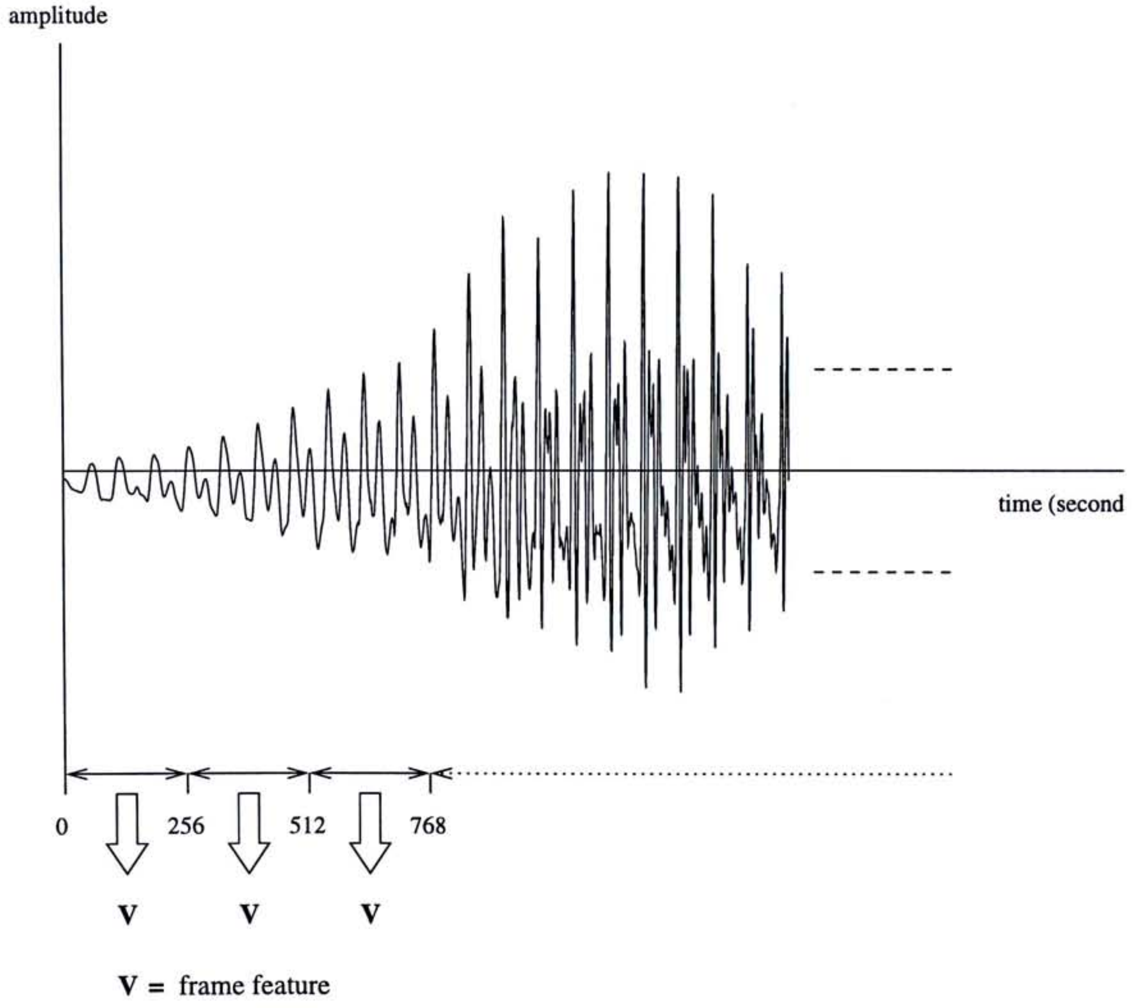


Figure 2.6: Feature extracted to a 12-dimensional vector by FFT on each frame.

tive features [27]. The frequency range of interest are divided into 12 spectral bands in which frequencies are mapped to a 12-dimensional vector as shown in Table 2.1. The widths of spectral bands are varied from 200 Hz at lower frequencies to 600 Hz at higher frequencies.

Table 2.1: Extracted frequencies are mapped to a 12-dimensional vector.

Spectral band	Frequency range (Hz)	Band width (Hz)
1	100-300	200
2	250-450	200
3	400-600	200
4	550-750	200
5	700-900	200
6	900-1200	300
7	1200-1500	300
8	1500-1800	300
9	1800-2300	500
10	2300-2800	500
11	2800-3400	600
12	3400-4000	600

Normalization

Normalization of a frame vector is accomplished by subtracting the average value for that frame from the value of their spectral bands. The normalized values are shifted around the zero level and the sum of those normalized values will be equal to zero. This method can eliminate the enormous difference between the large amplitude signal and small amplitude signal. Moreover, log scale is used for more accurate representation of the speech signal in the spectral domain. Therefore, element of the normalized frame vector is accomplished by subtracting the average log energy for that frame from the log of the energy in each of the spectral bands,

$$f_i = \log \hat{f}_i - \frac{\sum_{j=1}^{12} \log(\hat{f}_j)}{12}, \qquad \text{for} 1 \leq i \leq 12 \tag{2.10}$$

where f_i = normalized output of the i^{th} spectral band of the frame and $\hat{f}_i$ = energy in the i^{th} spectral band of the frame.

2.3.2 Linear Predictive Coding (LPC) coefficients

LPC coefficients are commonly extracted as a spectral feature vector from a short-time frame in speech signal. The idea is that a given speech sample at time n, $s(n)$, can be approximated as a linear combination of the past p speech samples, such that

$$s(n) = \sum_{i=1}^{p} a_i s(n-i) + Gu(n), \tag{2.11}$$

where the coefficients $a_1, a_2, ..., a_p$ are assumed constant over the frame, $u(n)$ is a normalized excitation and G is the gain of the excitation. The model of Equation (2.11) is shown in Figure 2.7. The linear combination of past speech samples as the

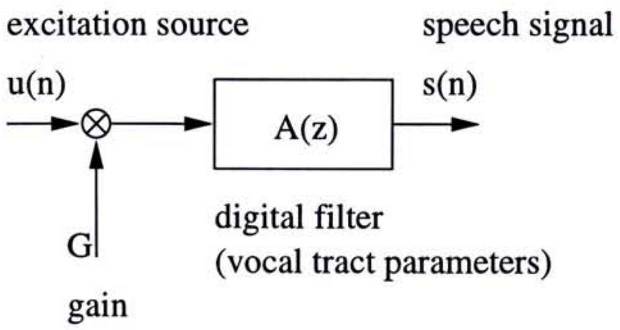


Figure 2.7: Linear prediction model of speech.

estimate $\tilde{s}(n)$ is defined as

$$\tilde{s}(n) = \sum_{i=1}^{p} a_i s(n-i) \tag{2.12}$$

Therefore, $a_1, a_2, a_3, ..., a_p$ are used as feature elements to represent the frame signal. The prediction error, $e(n)$, is defined as

$$e(n) = s(n) - \tilde{s}(n) = s(n) - \sum_{i=1}^{p} a_i s(n-i) \tag{2.13}$$

The set of predictor coefficients, $\{a_i\}$, needs to be determined in such a way that the following error is being minimized.

$$E = \sum_{m=0}^{N-1+p} e^2(m) \tag{2.14}$$

where N is the total sampling number in a frame, or

$$E = \sum_{m=0}^{N-1+p} (s(m) - \sum_{i=1}^{p} a_i s(m-i))^2. \quad (2.15)$$

To solve the predictor coefficients in Equation (2.15), E is first differentiated with respect to each a_i and the result is then set to zero,

$$\frac{\partial E}{\partial a_i} = 0, \qquad i = 1, 2, ..., p$$

Therefore,

$$r(x) = \sum_{i=1}^{p} a_i r(|x-i|), \qquad 1 \leq x \leq p \quad (2.16)$$

where the autocorrelation function, $r(\tau) = \sum_{m=0}^{N-1-\tau} s(m)s(m+\tau) \qquad \tau = 0, 1, 2, ..., p$
Equation (2.16) describes a set of p equations in p unknowns. They can be expressed in matrix form as

$$\begin{bmatrix} r(0) & r(1) & r(2) & ... & r(p-1) \\ r(1) & r(0) & r(1) & ... & r(p-2) \\ r(2 & r(1) & r(0) & ... & r(p-3) \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ r(p-1) & r(p-2) & r(p-3) & ... & r(0) \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ . \\ . \\ . \\ a_p \end{bmatrix} = \begin{bmatrix} r(1) \\ r(2) \\ r(3) \\ . \\ . \\ . \\ r(p) \end{bmatrix} \quad (2.17)$$

The $p \times p$ matrix of autocorrelation values is a Toeplitz matrix (symmetric with all diagonal elements equal) and hence can be solved efficiently through several well-known procedures such as the Levinson-Durbin's Recursive Algorithm [28, 29] which is used in our work. These $p+1$ autocorrelations are converted into LPC coefficients by the procedure involved as

Step (1) :

$E^{(0)} = r(0)$

Step (2) :

$k_i = \frac{r(i) - \sum_{j=1}^{i-1} a_j^{(i-1)} r(|i-j|)}{E^{(i-1)}}, \qquad 1 \leq i \leq p$

$$
\begin{aligned}
&a_i^{(i)} = k_i \\
&a_j^{(i)} = a_j^{(i-1)} - k_i a_{i-j}^{(i-1)}, \qquad 1 \le j \le i-1 \\
&E^{(i)} = (1 - k_i^2)E^{(i-1)}
\end{aligned}
$$

Step (3) :

back to **Step (2)**, for i from 1 to p

Step (4) :

$$a_j = a_j^{(p)}, \qquad 1 \le j \le p$$

2.3.3 Cepstral coefficients

The cepstral coefficients, which are the coefficients of the Fourier transform representation of the log magnitude spectrum, have been shown to be a more robust, reliable feature set for speech recognition than the LPC coefficients [24]. Cepstral coefficients, c_n can be derived directly from the LPC coefficient set. The recursion used is

$$
\begin{aligned}
c_0 &= \log \delta^2 \\
c_n &= a_n + \sum_{k=1}^{n-1} (\tfrac{k}{n}) c_k a_{n-k}, \qquad 1 \le n \le p \\
c_n &= \sum_{k=1}^{n-1} (\tfrac{k}{n}) c_k a_{n-k}, \qquad n > p
\end{aligned}
$$

where δ^2 is the gain term in the LPC model.

2.3.4 Zero crossing rate and energy

Zero crossing rate [30] can be used as a special feature parameter for speech signals. For each frame, zero crossing rate is counted. A collection of these counted numbers could be the characteristic of a speech signal. Referring to *Lau*, 1986 [31], these features can give quite a good representation of different words for a small vocabulary. Also, as described in Section 2.1.2, voiced and unvoiced sounds can be distinguished by using zero crossing rate. Short-time energy described in Section 2.1.1 can be used to separate a sampled speech signal from the background noise. Also, energy can be used to distinguish different speech signals since they have different energy profiles. The advantage of using zero crossing rate and energy is that they are derived directly from the speech signal and require less computation.

2.3.5 Pitch (fundamental frequency) detection

Pitch is defined as the fundamental frequency of quasi-stationary speech signal. Being a tonal language, the pronunciation of a Chinese character is characterized by its constituent phonemes and distinctive tone. Therefore, both phonemes and tone are essential to be considered as features in a Cantonese speech recognition system. This language, commonly used in Hong Kong and southeastern China, has nine lexical tones. Tones 1 to 6 are classified as non-entering tones and tones 7 to 9 as entering tones. Figure 2.8 shows their relative pitch contours of nine lexical tones. Basically, the fundamental frequency is calculated by the technique of auto-

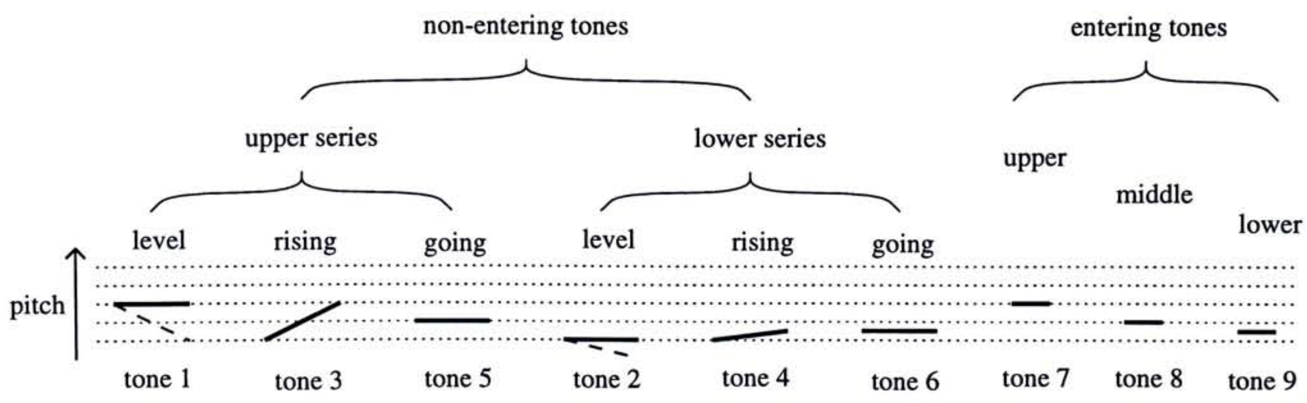


Figure 2.8: Relative pitch contours of nine lexical tones in Cantonese.

correlation [32]. The auto-correlation function of a windowed discrete time sequence $s(n)$ is defined as

$$r(\tau) = \sum_{n=0}^{N-1-\tau} s(n)s(n+\tau),$$

where τ is the time delay and N is the window size. When the time delay, τ, equals to the signal period or its multiple (harmonic), peaks occur for a periodic signal. However, the fundamental frequency of a signal can be found by searching the maximum peak value.

A well developed pitch detection algorithm proposed by T. Lee [33] is adopted for extracting the tone feature in Cantonese. The flow diagram for the pitch detection is shown in Figure 2.9. The pitch extraction method is a modified version of the

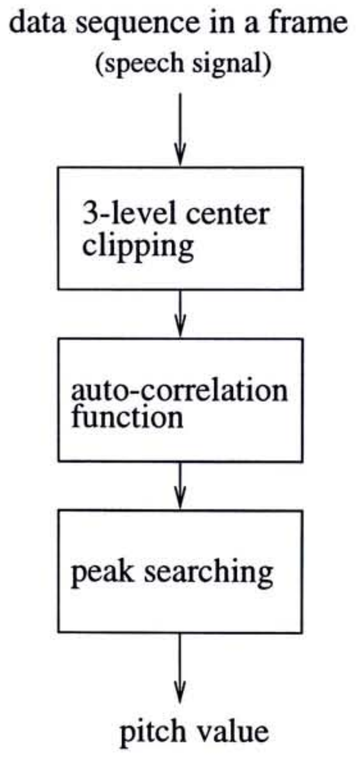


Figure 2.9: Flow diagram of pitch detection algorithm.

well-known 3-level center clipped algorithm [34, 35]. Referring to the flow diagram, the function of 3-level center clipping is used to enhance the edge of the pitch (fundamental frequency) for the auto-correlation function. The auto-correlation function can estimate the pitch value. In peak searching, sometimes the peak at the real pitch position does not have the largest amplitude and more than one outstanding peaks may exist. An effective peak searching method to deal with this problem is proposed by Y. H. Cheng [35] and modified by T. Lee [33]. By using this algorithm, examples of nine different lexical tones in Cantonese are extracted. The different profiles among them are shown in Figure 2.10. They are listed in Table 2.2.

Table 2.2: Nine Cantonese syllables with different tones.

Phonetic symbol		Tone
tɔ1	(多)	upper level (tone 1)
piu3	(表)	upper rising (tone 3)
pun5	(半)	upper going (tone 5)
fei2	(肥)	lower level (tone 2)
jyn4	(軟)	lower rising (tone 4)
tai6	(大)	lower going (tone 6)
pɐk7	(北)	upper entering (tone 7)
pak8	(百)	middle entering (tone 8)
hɔk9	(學)	lower entering (tone 9)

2.4 Discussions

A number of feature sets commonly used for speech processing were discussed as well as their mathematical formulation. The choice of feature for speech recognition is important since it can directly affect the recognition performance. If the algorithm for extracting features is rather complicated, the computational time will be longer. If elements in the feature set are similar, the recognition accuracy will be affected. In practical situation, the end-point detection is also important to locate an isolated uttered signal from the background. Recognition methods using these feature sets will be described in Chapter 3.

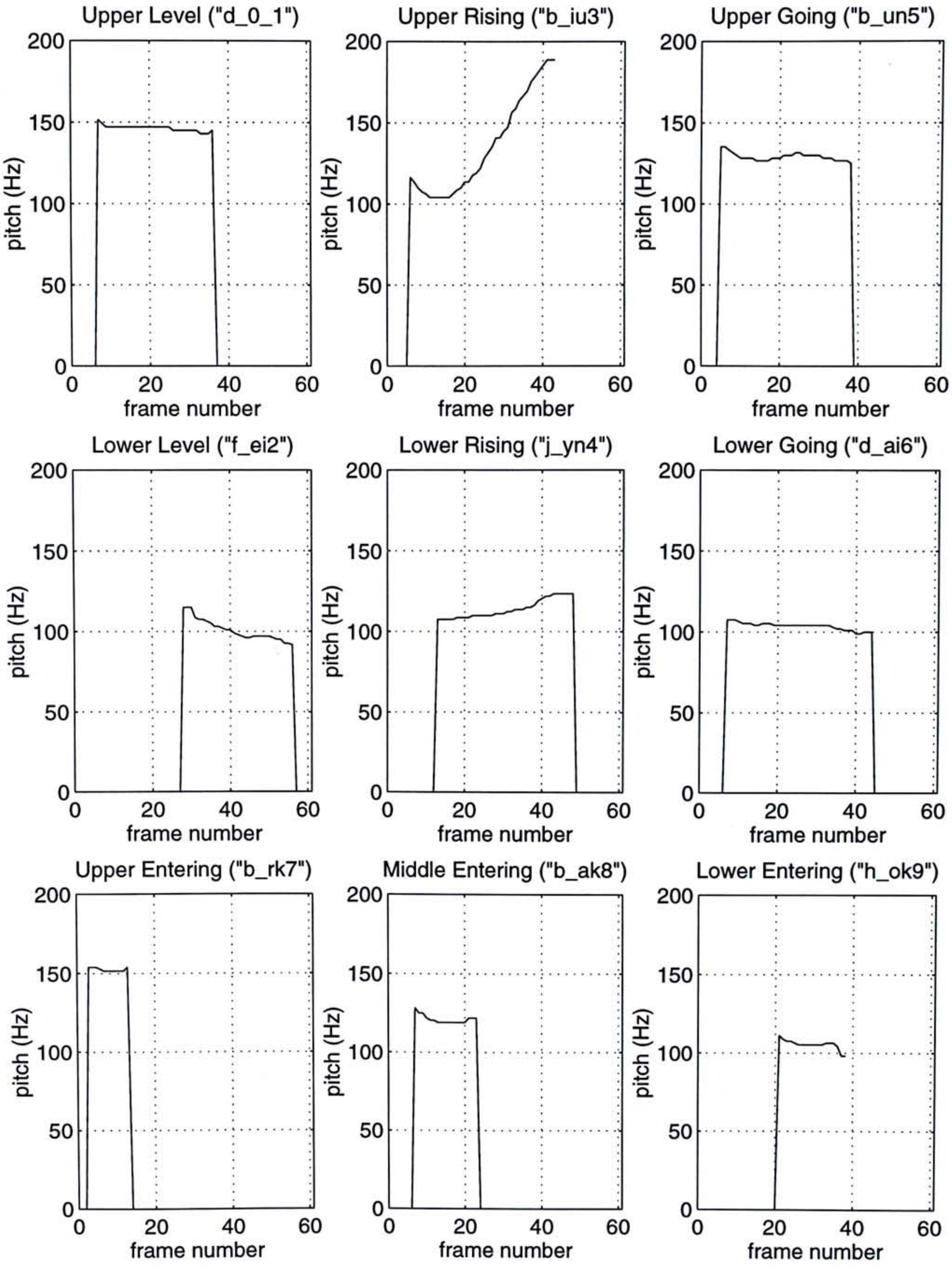


Figure 2.10: The profiles among nine tones in Cantonese language are extracted by the pitch detection algorithm.

Chapter 3

Speech Recognition Methods

3.1 Template matching using Dynamic Time Warping (DTW)

An acoustic model for a speech signal usually shows how it varies spectrally over the duration of the word. This is used as a template to describe a single typical way in which a word is pronounced. An unknown spoken word can then be identified by comparing it with all templates in the vocabulary and finding the best match. As described in the previous chapter, the filter-bank spectrum analysis model is considered as one of the effective parametric representations of speech signal. By using the parametric representation of the frequency time picture of a speech signal as a template for matching, a simple and effective speech recognition system can be realized. However, a word varies slightly from one pronunciation to another (e.g. it may be spoken quickly or slowly). In an attempt to handle this variation in speech recognition using template matching, Dynamic Time Warping (DTW) is used in the matching process [8, 9, 7]. The DTW is a pattern matching algorithm for computing the minimum distance between two patterns of different sizes with a nonlinear time normalization alignment. It can be formulated as a path finding problem over a finite grid formed from two patterns (Figure 3.1). A test pattern is denoted by

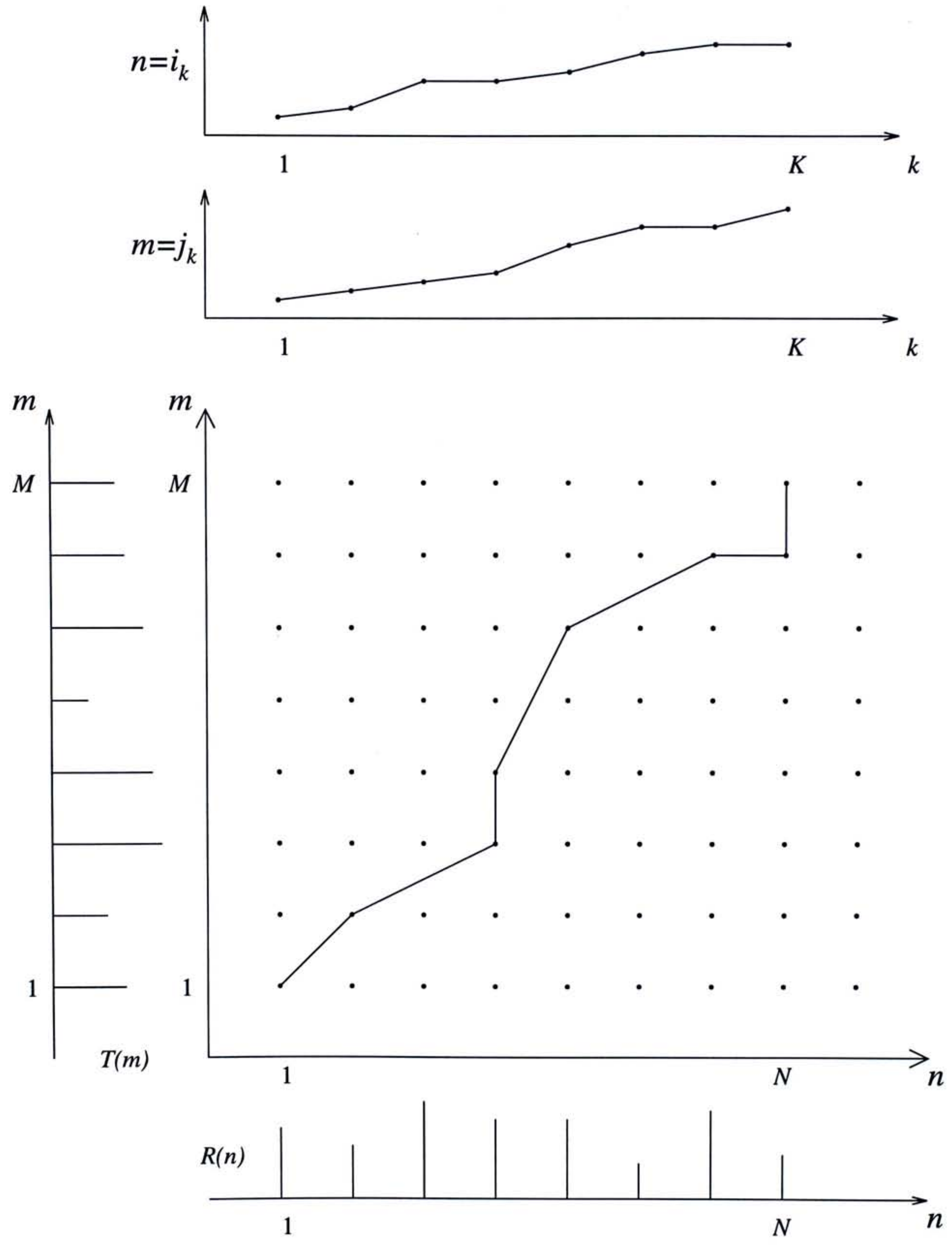


Figure 3.1: The matching between $R(n)$ and $T(m)$ by using DTW.

$T(m), m = 1, 2, ..., M$ placed on the y-axis and a reference template is denoted by $R(n), n = 1, 2, ..., N$ placed on the x-axis. A common time axis k is necessary that both time axes, m and n to be expressed as functions of time axis k. Therefore,

$$n = i_k, \quad k = \{1, 2, ..., K\}; \qquad m = j_k, \quad k = \{1, 2, ..., K\} \tag{3.1}$$

where K is the length of the common time axis k. An optimal warping function $c_k = (i_k, j_k) = (n, m)$ needs to be found to minimize the total distance function between two patterns.

$$D(R,T) = D(c_k) = \sum_{k=1}^{K} d(i_k, j_k) \tag{3.2}$$

where $d(i_k, j_k) = |T(m) - R(n)|^2$ is the local distance measured between the point j_k of the test pattern and the point i_k of the reference pattern. The matching pair should give the smaller value of the distance $D(R,T)$ between two patterns. For finding the best path in the $(n,\ m)$ plane, several factors of the algorithm must be considered, they are:

- **Endpoint constraints**

  The path is restricted to begin at the point $(n = 1, m = 1)$ and end at the point $(n = N, m = M)$, i.e.

  $i_1 = 1,\ j_1 = 1$ and $i_K = N,\ j_K = M$

- **Monotonicity constraints**

  $i_k \leq i_{k+1}$ and $j_k \leq j_{k+1}$

- **Local path constraints**

  Symmetric local path constraints are set to prevent excessive compression or expansion of the scale in the warping path. There are several local path constraints as proposed in [8]. Since the choice of local path constraint does not affect the accuracy very much. For simplicity, the Type II Constraint is chosen. It allows 3 possible predecessors. They are $(n-2, m-1), (n-1, m-1)$

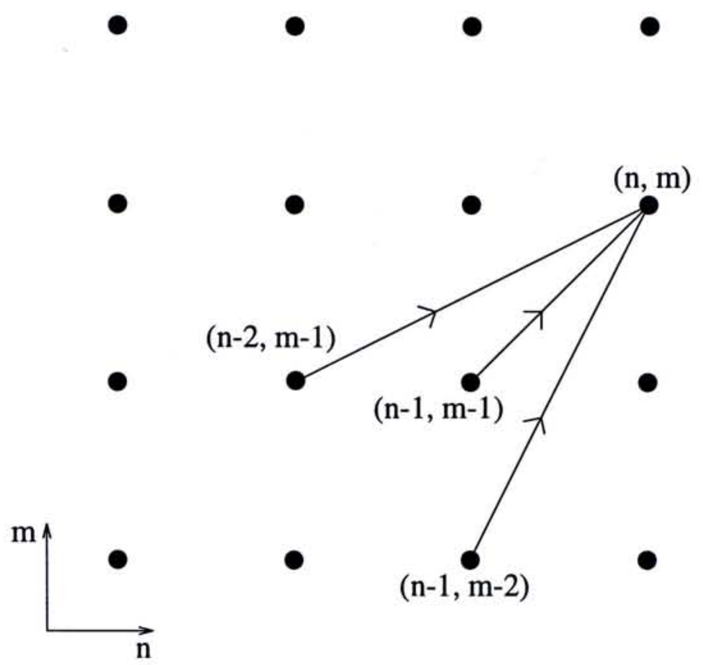


Figure 3.2: Local path constraints for DTW.

and $(n-1, m-2)$. Figure 3.2 shows three valid paths to the grid point (n, m).

- **Global path constraints**

  Since the local path constraints, certain parts of the (n, m) plan are excluded from the region which the optimal warping path can lie in. In Figure 3.2, the greatest slope ($\mathrm{E_{max}}$) for all 3 possible paths is 2 (2/1), the smallest slope ($\mathrm{E_{min}}$) is 0.5 (1/2). A set of relations can be obtained from the slope of warping function:

$$\mathrm{E_{min}} \le \frac{j_k - 1}{i_k - 1} \le \mathrm{E_{max}} \tag{3.3}$$

$$\mathrm{E_{min}} \le \frac{M - j_k}{N - i_k} \le \mathrm{E_{max}} \tag{3.4}$$

Equation (3.3) is used to limit the range of grid points that can be reached via a legal path from the point (1, 1). Similarly, Equation (3.4) is to limit the range to the point (N, M). By substituting $\mathrm{E_{max}}$=2 and $\mathrm{E_{min}}$=1/2 into Equations (3.3) and (3.4),

$$\frac{i_k + 1}{2} \le j_k \le 2i_k - 1 \tag{3.5}$$

$$M + 2(i_k - N) \leq j_k \leq \frac{i_k - N}{2} + M \tag{3.6}$$

- **Distance measure**

  The total distance along the path of length K,

$$D(i_k, j_k) = \frac{\sum_{k=1}^{K} d(i_k, j_k) W(k)}{N(W)} \tag{3.7}$$

where $W(k)$ is a weighting function of the k^{th} point of the path, and $N(W)$ is a normalization factor which is a function of the weighting function W. The searching process for the path which gives the minimum distance $\hat{D}$ is described by

$$\hat{D} = \min_{K, i_k, j_k} (D(i_k, j_k)) \tag{3.8}$$

The weighting function of Type C [8] is chosen. For each arc, there is a number which indicates the weight (Figure 3.3). This weighting method is based on the distance moved towards x direction. Therefore,

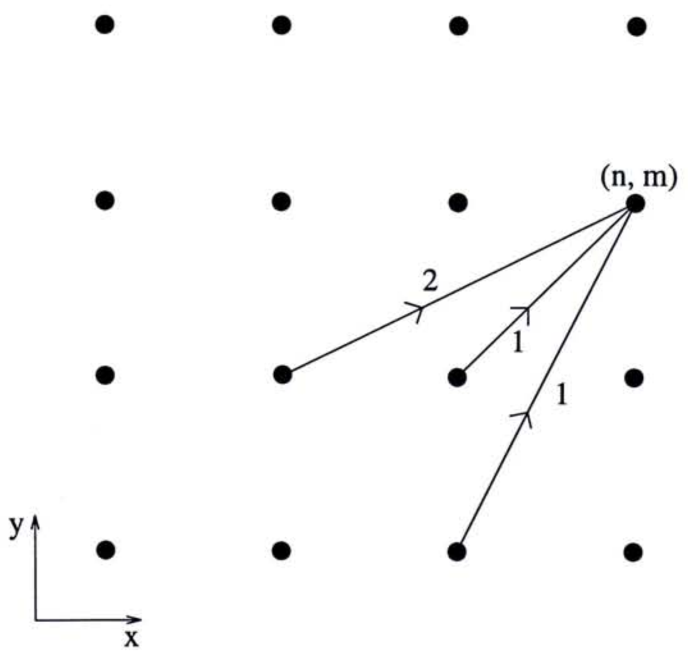


Figure 3.3: Weighting function for DTW.

$$W = i_k - i_{k-1} \tag{3.9}$$

Hence, the normalization function is,

$$N(W) = \sum_{k=1}^{K} W(k) = \sum_{k=1}^{K} (i_k - i_{k-1}) = i_K - i_0 = N \qquad (3.10)$$

- **Axis orientation**

  The test pattern and reference template are arbitrarily placed along the m and n axes. As suggested in [8], Type C weighting function should be combined with test pattern located in n axis (x direction) to improve accuracy.

3.2 Hidden Markov Model (HMM)

A hidden Markov model (HMM) is created from a large amount of data, statistically modeling the variation seen in that data. It is a finite-state statistical model, useful for modeling nonstationary signals whose time-varying characteristics may be described through a chain of statistical states. The assumption is based on the Markov process (Appendix D). There are some variations of HMM that are developed to improve the model itself. Some examples are continuous HMM [36, 37, 38], semi-continuous HMM [39], explicit-duration HMM [40], etc.

The basic HMM model parameters are $\lambda = (A, B, \pi)$, where A is a transition probability matrix, B is an output probability matrix, and π is an initial state probability vector. A typical speech modeling recognition system is shown in Figure 3.4. As the vocal tract is continuously variable, the speech sounds themselves become time varying [41] [42]. They are essentially non-stationary waveforms. However, by considering a small enough segment as a frame of the signal, the short time features can be obtained.

Thus speech modeling involves the analysis of the short time properties (features) of individual sounds. This short time vector is generally called an observation. Observations are typically measured once every 10-30 ms. Frame overlapping is usually employed to avoid abrupt change between frames. The short time properties of an

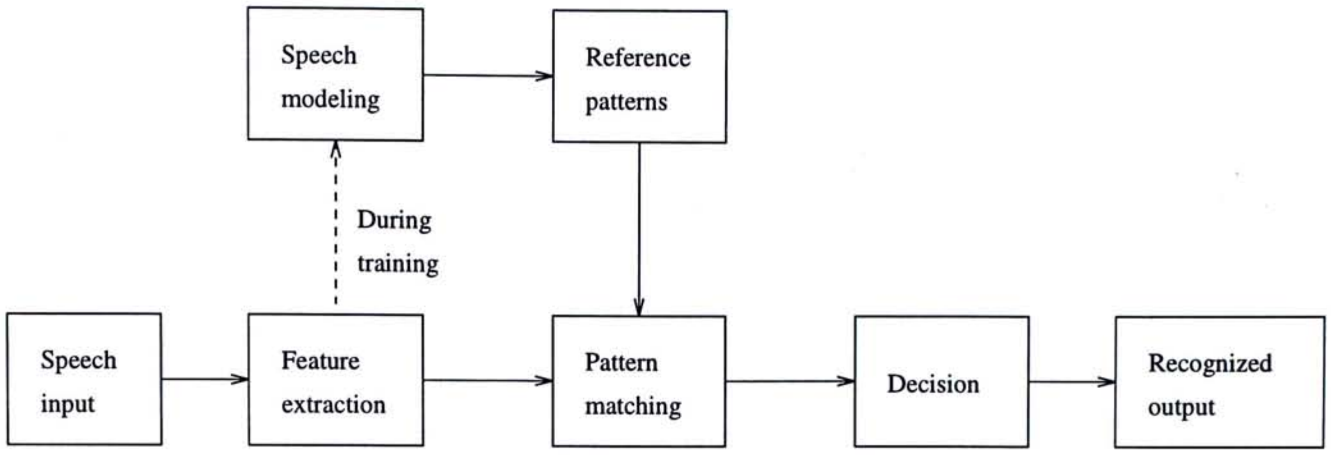


Figure 3.4: Block diagram of a typical speech modeling recognition system.

individual sound can be represented by a spectral measurement vector which can be obtained by using standard methods such as measurement of the discrete Fast Fourier Transform (FFT), Linear Predictive Coding (LPC) coefficients or cepstral coefficients. Let the feature vector at time t be o_t and the observed spectral sequence lasts from $t = 1$ to $t = T$, a deterministic sequence is then represented by $O = \{o_t\}_{t=1}^{T} \equiv \{o_1, o_2, ..., o_T\}$. HMMs are further classified into two types. They are Discrete Hidden Markov Model (DHMM) and Continuous Hidden Markov Model (CHMM). In our work, discrete HMMs are used since they require less computation than the CHMM. However much time is consumed in calculating the required codebook. The output of an DHMM are discrete probability distributions, therefore, Vector Quantization (VQ) is necessary to convert feature vectors of the speech waveform into a finite set of prototypes. For simplicity and computation efficient, DHMM is more practical to be used in the speech recognition system.

3.2.1 Vector Quantization (VQ)

The results of spectral analysis methods for a speech signal such as Fast Fourier Transform (FFT) or Linear Predictive Coding (LPC) are a series of vectors characterizing the time varying spectral characteristic of the speech signal. These vectors contain many similar information concerning the speech waveform. For speech recognition, the amount of information contained in the vector may be reduced by Vector Quantization (VQ). VQ is a source-coding (data compression) technique in communication and information theory. It is a procedure that encodes a vector into an index that is associated with an entry of a codebook. A given vector can be represented by one which is closest to a vector in a codebook. The codebook represents the underlying speech characteristic of the speech signals. Therefore, VQ can save the storage for spectral analysis information and reduce computation for determining the similarity of spectral analysis vectors. However, an inherent distortion is used in representing the actual vector and a codebook is needed. VQ is essential to DHMM.

A codebook can be trained by a large set of spectral analysis vectors v_i, $i = 1, 2, ..., L$. For the size of the VQ codebook being $M = 2^B$ vectors (B-bit codebook), L must much greater than M to be able to find the best set of M codebook vectors. In practice, it has been found that L should be at least $10M$ in order to train a VQ codebook that works reasonably well [24].

The vector quantization quality has a significant influence on the recognition rates. It is a tradeoff between processing time and quantization error. If a codebook size is very large, the quantization error can be reduced. However, it will take time to do the computation. Therefore the optimized codebook size should be chosen wisely to fit different purposes. In our works, the codebook size of 32 is always to be used in experiments.

VQ algorithm to generate a codebook

There are several ways to generate a codebook for VQ. Fundamentally, a set of L training vectors can be clustered or quantized into a set of M codebook vectors. This method is so called K-means clustering algorithm (or Lloyd algorithm). The result of designing a VQ codebook by showing the partitioning of a spectral vector space into distinct areas where can be represented by a centroid vector is illustrated in Figure 3.5.

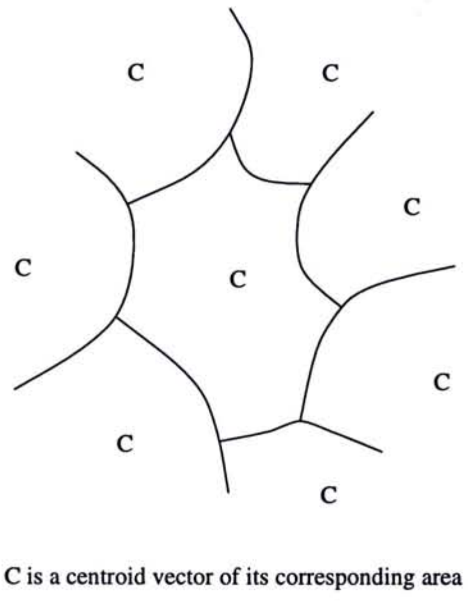


Figure 3.5: Partitioning of a vector space into different areas where they are represented by a centroid vector.

The one proposed by *Y. Linde et al.*, 1980 [43] was used in the project. The algorithm is a modification of K-means clustering algorithm. The advantage of this method is that the design starts with a 1-vector codebook and an M-vector codebook is produced by using a splitting technique on the code words and continuing the splitting process until the desired M-vector codebook is obtained.

The codebook is constructed in following procedure:

1. Design a codebook with size 1 (a vector), this vector is the centroid of the entire set of training vectors.

2. The size of the codebook is increased by splitting each current codebook y_n according to the Equation (3.11)

$$y_n^+ = y_n(1+\epsilon), \qquad y_n^- = y_n(1-\epsilon) \tag{3.11}$$

   where n varies from 1 to the current size of the codebook, and ϵ is a splitting parameter.

3. Finding the code word for each training vector in the current codebook that is closest (in terms of spectral distance) and vectors are assigned to their corresponding centroids.

4. Updating centroids by using code words that belong to their centroids.

5. Repeating steps (2), (3) and (4) until a codebook of size M is obtained.

3.2.2 Description of a discrete HMM

An HMM is uniquely defined by the following three sets of parameters:

- **π, the initial state-distribution vector**
  it defines the probability of starting the Markov chain in a given state.
- **A, the transition probability matrix**
  it defines the probability of jumping from one state, at any time t, to the next state, at time $t+1$.
- **B, the output probability matrix**
  it defines the probability of producing a given observation, in a given state, at any time t.

Model parameters are summarized by $\lambda = (\pi, A, B)$. The matrices A and B, whose entries are in [0,1], and whose elements sum to 1 in each of their rows, are stochastic matrices. Also for a vector π, all entries are in [0,1] and elements sum is equal to one. All the probabilities of characteristic sequences can be expressed in terms of the model parameters λ. For the left-to-right model, the sequence should start from state 1. Therefore $\pi_1 = 1$, otherwise $\pi_i = 0$.

3.2.3 Probability evaluation

For an observation sequence, the probability of its occurring can be obtained from a given model. For example, if an observation is $\{o_1, o_2, o_3, o_4, o_5\}$. From a three states left-to-right HMM as shown in Figure 3.6 with the following parameters:

$$\begin{aligned} A &= \{a_{ij}\} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \\ B &= \{b_j(k)\} = \begin{bmatrix} b_1(1) & b_1(2) & b_1(3) & b_1(4) \\ b_2(1) & b_2(2) & b_2(3) & b_2(4) \\ b_3(1) & b_3(2) & b_3(3) & b_3(4) \end{bmatrix}, \\ \pi &= [\pi_1, \pi_2, \pi_3]. \end{aligned}$$

The joint probability of an observation and state sequence, $\mathrm{P}(O, q|\lambda)$ is the product

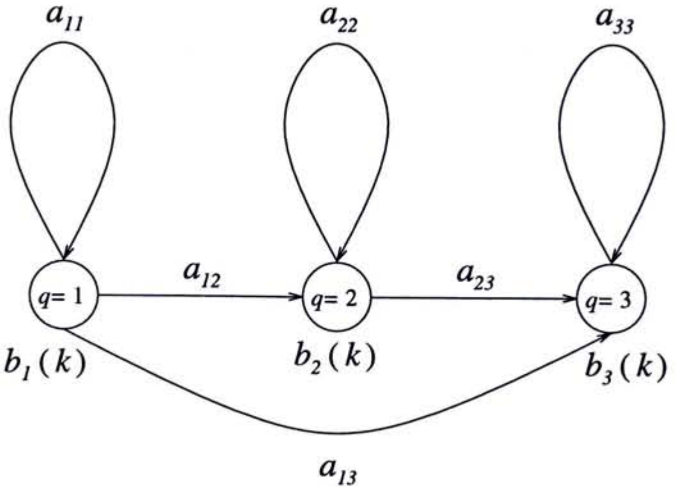


Figure 3.6: State diagram of a left-to-right hidden Markov model.

of the $\mathrm{P}(O|q, \lambda)$ and $\mathrm{P}(q|\lambda)$. The probability of the observation sequence, $\mathrm{P}(O|\lambda)$ is obtained by summing this joint probability over all possible state sequences, therefore,

$$\begin{aligned} \mathrm{P}(O|\lambda) &= \sum_{q_1,q_2,\ldots,q_5} \mathrm{P}(O,q|\lambda) \\ &= \sum_{q_1,q_2,\ldots,q_5} \mathrm{P}(O|q,\lambda) \times \mathrm{P}(q|\lambda) \\ &= \sum_{q_1,q_2,\ldots,q_5} b_{q_1}(o_1)b_{q_2}(o_2)\ldots b_{q_5}(o_5) \times \pi_{q_1}a_{q_1q_2}a_{q_2q_3}\ldots q_{q_4q_5} \\ &= \sum_{q_1,q_2,\ldots,q_5} \pi_{q_1}b_{q_1}(o_1)\ a_{q_1q_2}b_{q_2}(o_2)\ a_{q_2q_3}b_{q_3}(o_3)\ldots q_{q_4q_5}b_{q_5}(o_5) \end{aligned} \tag{3.12}$$

Hence, Equation (3.12) can be used to calculate the probability of an observation sequence. Initially, the sequence starting at time $t = 1$ in state q_1 with probability π_{q_1} generates the symbol o_1 with probability $b_{q_1}(o_1)$ in this state. Then the transition from time $t = 1$ to $t = 2$ is from state q_1 to state q_2 with probability $a_{q_1q_2}$. Also, a symbol o_2 is generated with probability $b_{q_2}(o_2)$. Similarly, the remain sequence follows until it reaches time $t = T$ from state q_{T-1} to state q_T and generate symbol o_T with probability $b_{q_T}(o_T)$. Finding of $\mathrm{P}(O|\lambda)$ from direct definition involves a lot of calculations. Totally, there are $(2T-1)N^T$ multiplications and $N^T - 1$ additions. It will not be practical to calculate the probability. However, there is an efficient way called forward algorithm that is capable to tackle the problem. Similarly, a backward algorithm calculating from backward exits to calculate the probability.

For forward algorithm, a forward induction procedure allows evaluation of the probability $\mathrm{P}(O|\lambda)$ to be carried out in the observation sequence with length T. For example, if there are four states in the model (i.e. $N = 4$) and the forward variable $\alpha_t(i)$ is defined as

$$\alpha_t(i) = \mathrm{P}(o_1, o_2, \ldots, o_t, q_t = i|\lambda)$$

which is the probability of the partial observation sequence up to time t and state $q_t = i$. Figure 3.7 shows a trellis structure implementation for the computation of $\alpha_t(i)$.

- **Forward algorithm**

Step 1 :

Initialize $\alpha_1(i)$

$$\alpha_1(i) = \pi_i b_i(o_1), \quad \text{for all states } i$$

($\pi_1 = 1$; otherwise $\pi_i = 0$ for a left-to-right model).

Step 2 :

Calculate $\alpha()$ along the time axis for $t = 2, ..., T$ and all states j

$$\alpha_t(j) = [\sum_i^N \alpha_{t-1}(i) a_{ij}] b_j(o_t) \tag{3.13}$$

Step 3 :

Probability is given by

$$\mathrm{P}(O|\lambda) = \sum_{i \in S_F} \alpha_T(i) \tag{3.14}$$

where $S_F = \{1, 2, ..., N\}$ is a set of final states.

- **Backward algorithm**

Step 1 :

Initialize $\beta_T(i)$

$$\beta_T(i) = 1$$

Step 2 :

Calculate β () along the time axis for $t = T - 1, ..., 1$ and all states j

$$\beta_t(j) = \sum_i a_{ji} b_i(o_{t+1}) \beta_{t+1}(i) \tag{3.15}$$

Step 3 :

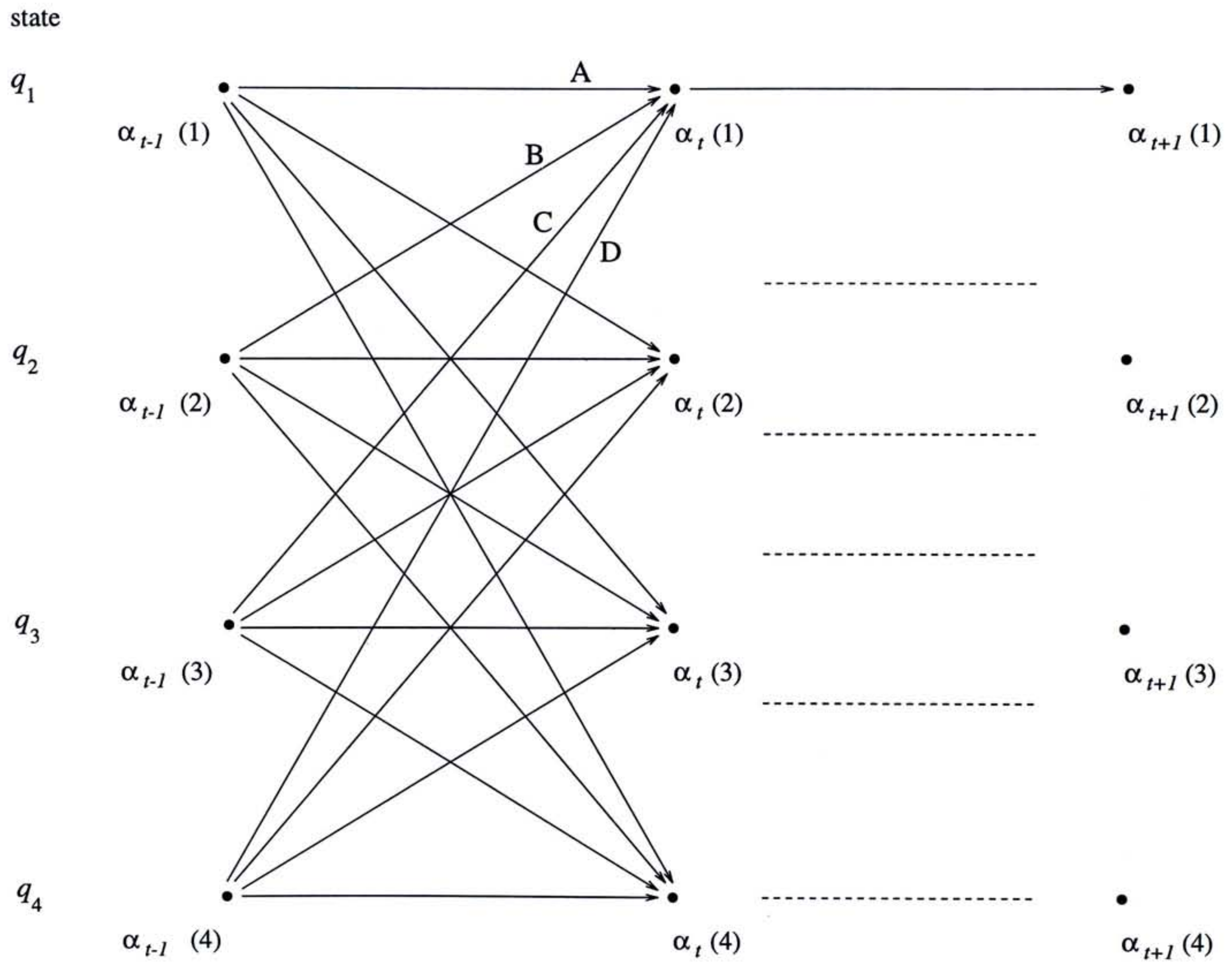
Probability is given by

$$\mathrm{P}(O|\lambda) = \sum_{i \in S_I} \pi_i b_i(o_1) \beta_1(i) \tag{3.16}$$

where $S_I = \{1\}$ is a set of final states.

Or

$$\mathrm{P}(O|\lambda) = \pi_1 b_1(o_1) \beta_1(i)$$



$$\mathrm{A} = a_{1,1}b_1(O_t)$$
$$\mathrm{B} = a_{2,1}b_1(O_t)$$
$$\mathrm{C} = a_{3,1}b_1(O_t)$$
$$\mathrm{D} = a_{4,1}b_1(O_t)$$

$$\alpha_t(1) = \mathrm{A}\ \alpha_{t-1}(1) + \mathrm{B}\ \alpha_{t-1}(2) + \mathrm{C}\ \alpha_{t-1}(3) + \mathrm{D}\ \alpha_{t-1}(4)$$

Figure 3.7: A trellis structure for the calculation of the forward partial probabilities $\alpha_t()$ of an HMM with 4-state.

The forward or backward algorithms can reduce tremendous computation and make the HMM method efficient. The calculated probability can be viewed as matching score to indicate how well an unknown observation sequence matches a given model.

3.2.4 Estimation technique for model parameters

We have described how $\mathrm{P}(O|\lambda)$ can be calculated. But we have not addressed how a speech signal (a word) can be modeled by HMM. For HMM modeling, it involves selecting the state size (topology of a model), and calculating the model parameters from an observation of a speech waveform. This is called training the model. An observation sequence of a speech signal that is used to train a model is called a training sequence. There is no analytical method to solve for the model parameters. However, an iterative method based on Maximum Likelihood (ML) can be used for obtaining a solution. For $\lambda = (A, B, \pi)$, its likelihood of $\mathrm{P}(O|\lambda)$ is locally maximized using an iterative procedure such as the Baum-Welch method [44] [45] [46].

- **Baum-Welch re-estimation algorithm**

  If the model parameters are known, the forward-backward algorithm can be used to evaluate probabilities produced by given model parameters and training sequence. Then, an estimation of original model parameters can be made based on current probabilities. $\mathrm{P}(O|\lambda)$ should be the same whether it is calculated from forward or backward probabilities, i.e.

$$\mathrm{P}(O|\lambda) = \sum_{i \in S_F} \alpha_T(i) = \sum_{i \in S_I} \pi_i b_i(O_1) \beta_1(i) = \sum_i \alpha_t(i) \beta_t(i). \tag{3.17}$$

  where S_I is a set of initial states and S_F is a set of final states.

  By using the forward-backward algorithm on such a model, the *posterior* probability of transitions $\gamma_t(ij)$, from state i to state j at time t, conditioned on the training sequence (O) and the model (λ) can be computed as

$$\begin{aligned}\gamma_t(i,j) &= \mathrm{P}(s_t = i, s_{t+1} = j|O,\lambda) \\ &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\mathrm{P}(O|\lambda)} \\ &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum\limits_{k\in S_F}\alpha_T(k)} \end{aligned} \tag{3.18}$$

The physical meaning of a_{ij} is the probability of the transition from state i to state j. Thus the ratio $\sum\limits_{t=1}^{T-1}\gamma_t(i,j)/\sum\limits_{t=1}^{T-1}\gamma_t(i)$ is an estimate of the probability a_{ij}. This ratio may be taken as a new estimate, $\overline{a}_{ij}$ of a_{ij}.

$$\begin{aligned}\overline{a}_{ij} &= \frac{\sum\limits_{t=1}^{T-1}\gamma_t(i,j)}{\sum\limits_{t=1}^{T-1}\sum\limits_{j}\gamma_t(i,j)} \\ &= \frac{\sum\limits_{t=1}^{T-1}\gamma_t(i,j)}{\sum\limits_{t=1}^{T-1}\gamma_t(i)}\end{aligned} \tag{3.19}$$

Similarly, the physical meaning of $b_j(k)$ is the probability of observation symbol v_k occurring in state j. This can be computed as the frequency of occurrence of observation symbol v_k relative to the frequency of occurrence of any observation symbol in state j. Summation of $\gamma_t(i)$ over the time index t is the expected number of times that state i is visited. $b_j(k)$ can be re-estimated as:

$$\overline{b}_j(k) = \frac{\sum\limits_{t\in O_t=v_k}\gamma_t(j)}{\sum\limits_{t=1}^{T}\gamma_t(j)} \tag{3.20}$$

It can be shown that either:

1. the initial model λ defines a critical point of the likelihood function, where new estimates equal old ones, or
2. model $\overline{\lambda}$ is more likely in the sense that $\mathrm{P}(O|\overline{\lambda}) \geq \mathrm{P}(O|\lambda)$, i.e. new model estimates are more likely to produce the given training sequence O.

Repeating the reestimation calculation by using $\overline{\lambda}$ in place of λ, $\mathrm{P}(O|\lambda)$ can be improved until a limiting point is reached. The result of this reestimation procedure is an ML estimate of the HMM, and the proof is shown in Appendix E.

3.2.5 State sequence for the observation sequence

The evaluated probability of an HMM in Section 3.2.3 does not explicitly involve the state sequence, however it is important in many applications to have the knowledge of the most likely state sequence for several reasons [13]. There are several possible ways to find the optimal state sequence. For example, maximization of $\mathrm{P}(q|O,\lambda)$ is equivalent to maximization of $\mathrm{P}(q,O|\lambda)$.

$$\mathrm{P}(q|O,\lambda) = \frac{\mathrm{P}(O,q|\lambda)}{\mathrm{P}(O|\lambda)}$$

It is because $\mathrm{P}(O|\lambda)$ is not involved in the optimization process. The optimal state sequence can be solved by the Viterbi algorithm [47] [48]. It is similar to the DTW algorithm (solved by dynamic programming) discussed in Section 3.1.

The Viterbi algorithm can be used in score evaluation. The forward and (or) backward algorithms can be used to calculate the probability $\mathrm{P}(O|\lambda)$ which is the summation of $\mathrm{P}(O,q|\lambda)$ over all possible state sequences. For Viterbi algorithm, the probability is obtained from maximum of $\mathrm{P}(O,q|\lambda)$ over all state sequences. Therefore, it is a special case of the forward-backward algorithm. The Viterbi algorithm is extremely efficient since it can operate in the logarithm domain using only additions; nevertheless, summation of $\mathrm{P}(O,q|\lambda)$ over all possible state sequences is used in our project. From experiments, the approach of maximizing $\mathrm{P}(O,q|\lambda)$ may not be better than $\sum\limits_{q}\mathrm{P}(O,q|\lambda)$, especially in reestimating model parameters although the probabilities obtained from forward-backward and Viterbi algorithms may be very close. In such cases, the forward-backward algorithm may work more robustly than the Viterbi algorithm [12]. In the forward-backward algorithm all the paths are taken into account, however the time consumed is not a critical factor in our cases.

3.3 2-dimensional Hidden Markov Model (2dHMM)

A 2-dimensional hidden Markov model (2dHMM) is designed to model two different feature sets of a speech waveform. The model consists of two statistically related HMMs. This configuration permits a more complete and an accurate characterization of the speech signal. An observation consists of a couple of acoustic parameters, the first model is a standard HMM (λ') and the second model is an HMM (λ'') whose parameters are probabilities functions of the state of the first model. In our work,

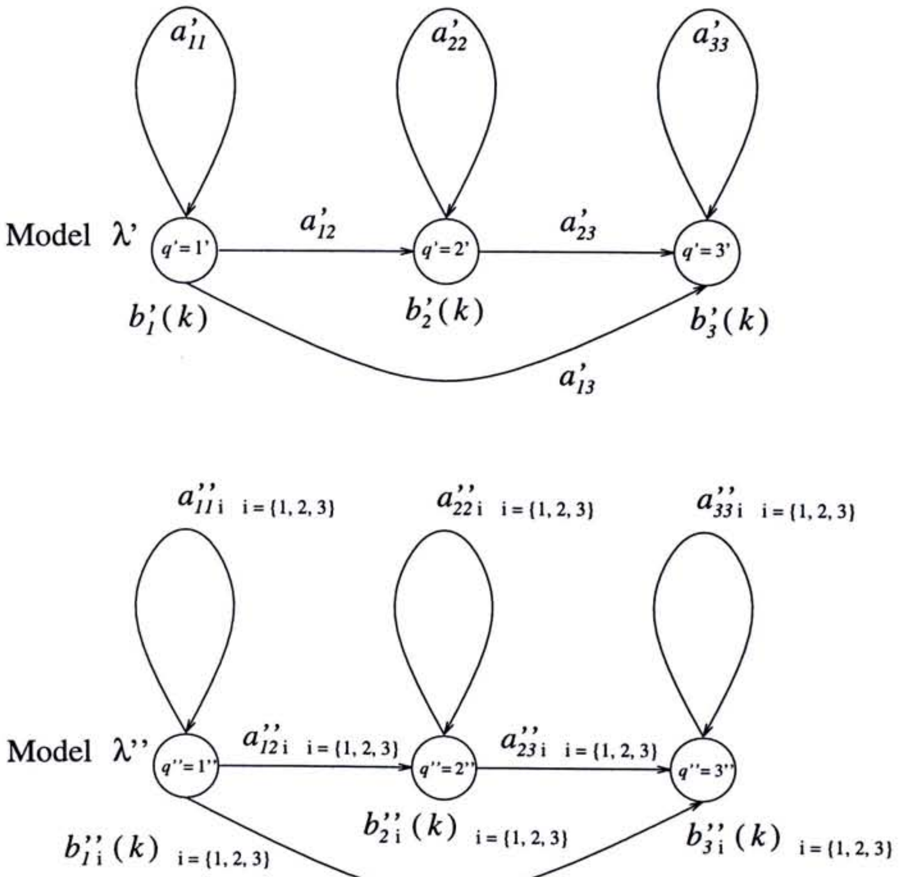


Figure 3.8: State diagram of an 2dHMM with 3-state in λ' and 3-state in λ''.

we adopted a discrete 2-dimensional HMM for modeling speech signal. Since the output probabilities are discrete probability distributions, vector quantization (VQ) is necessary to convert the continuous speech signal into a finite set of prototypes. The choice of the state size is to let the number of states correspond roughly to the number of sounds (phonemes) within the word [49]. Moreover, each word model is restricted to have the same number of states.

3.3.1 Calculation for a 2dHMM

For a couple of observation sequences,

$$O = \{o', o''\} = o_1^T \equiv \{(o'_1, o''_1), (o'_2, o''_2), ..., (o'_T, o''_T)\}$$

the probability can be obtained from a given model parameters (λ) which are organized by two correlated models (λ' and λ''). The idea of 2dHMM is based on the work of *F. Brugnara et al.*, 1991 [50]. The state notation for λ' is q' and that for λ'' is q''. The main difference between HMM and 2dHMM lies on the model topology. Therefore, given the 2dHMM, the joint probability of an observation sequence and the state sequence $\mathrm{P}(O, q|\lambda) = \mathrm{P}(o_1^T, q_1^T|\lambda)$ is defined as $\mathrm{P}(O', O'', q', q''|\lambda', \lambda'')$. For notations o_1^T and q_1^T, the lower index denotes the first element and the upper index denotes the last element. For example, $o_1^T = (o_1, o_2, \ldots, o_T) = ((o'_1\ o''_1), (o'_2\ o''_2), \ldots, (o'_T\ o''_T))$.

- **Parameter definitions**

  The model itself involves two correlated models, notations for them are defined as:

  for λ',

$$\begin{aligned}
a'_{i',j'} &\equiv \mathrm{P}(q'_t = j'|q'_{t-1} = i'), \\
&\qquad\qquad \text{for} \quad 1 \le i', j' \le N' \\
b'_{j'}(k') &\equiv \mathrm{P}(o'_t = v'_{k'}|q'_t = j'), \\
&\qquad\qquad \text{for} \quad 1 \le k' \le M'; \ \ 1 \le j' \le N' \\
\pi'_{i'} &\equiv \mathrm{P}(q'_1 = i'), \\
&\qquad\qquad \text{for} \quad 1 \le i' \le N'
\end{aligned}$$

($\pi'_{1'} = 1$; otherwise $\pi'_{i'} = 0$ for a left-to-right model).

and for λ'',

$$\begin{aligned}
a''_{i'',j'',j'} &\equiv \mathrm{P}(q''_t = j'' | q''_{t-1} = i'', q'_t = j'), \\
&\qquad \text{for} \;\; 1 \le i'', j'' \le N''; \;\; 1 \le j' \le N' \\
b''_{j''j'}(k'') &\equiv \mathrm{P}(o''_t = v''_{k''} | q''_t = j'', q'_t = j'), \\
&\qquad \text{for} \;\; 1 \le k'' \le M''; \;\; 1 \le j'' \le N''; \;\; 1 \le j' \le N' \\
\pi''_{i'',i'} &\equiv \mathrm{P}(q''_1 = i'' | q'_1 = i'), \\
&\qquad \text{for} \;\; 1 \le i'' \le N''; \;\; 1 \le i' \le N'
\end{aligned}$$

($\pi''_{1''} = 1$; otherwise $\pi''_{i''} = 0$ for a left-to-right model).

where N is the number of state in a model, M is the quantizational size for an observation in each state.

For example, the model parameters for the 2dHMM shown in Figure 3.8 are:

$$a' = \{a'_{i'j'}\} = \begin{bmatrix} a'_{1'1'} & a'_{1'2'} & a'_{1'3'} \\ a'_{2'1'} & a'_{2'2'} & a'_{2'3'} \\ a'_{3'1'} & a'_{3'2'} & a'_{3'3'} \end{bmatrix},$$

$$b' = \{b'_{j'}(k')\} = \begin{bmatrix} b'_{1'}(1) & b'_{1'}(2) & b'_{1'}(3) & b'_{1'}(4) \\ b'_{2'}(1) & b'_{2'}(2) & b'_{2'}(3) & b'_{2'}(4) \\ b'_{3'}(1) & b'_{3'}(2) & b'_{3'}(3) & b'_{3'}(4) \end{bmatrix},$$

$$\pi' = \{\pi'_{i'}\} = \begin{bmatrix} \pi'_{1'} & \pi'_{2'} & \pi'_{3'} \end{bmatrix}^{\mathrm{T}} \quad \text{(where T is transpose)},$$

$$a'' = \{a''_{i''j''i'}\} = \begin{bmatrix} a''_{1''1''1'} & a''_{1''2''1'} & a''_{1''3''1'} \\ a''_{2''1''1'} & a''_{2''2''1'} & a''_{2''3''1'} \\ a''_{3''1''1'} & a''_{3''2''1'} & a''_{3''3''1'} \\ a''_{1''1''2'} & a''_{1''2''2'} & a''_{1''3''2'} \\ a''_{2''1''2'} & a''_{2''2''2'} & a''_{2''3''2'} \\ a''_{3''1''2'} & a''_{3''2''2'} & a''_{3''3''2'} \\ a''_{1''1''3'} & a''_{1''2''3'} & a''_{1''3''3'} \\ a''_{2''1''3'} & a''_{2''2''3'} & a''_{2''3''3'} \\ a''_{3''1''3'} & a''_{3''2''3'} & a''_{3''3''3'} \end{bmatrix},$$

$$b'' = \{b''_{j''j'}(k'')\} = \begin{bmatrix} b''_{1''1'}(1) & b''_{1''1'}(2) & b''_{1''1'}(3) & b''_{1''1'}(4) \\ b''_{2''1'}(1) & b''_{2''1'}(2) & b''_{2''1'}(3) & b''_{2''1'}(4) \\ b''_{3''1'}(1) & b''_{3''1'}(2) & b''_{3''1'}(3) & b''_{3''1'}(4) \\ b''_{1''2'}(1) & b''_{1''2'}(2) & b''_{1''2'}(3) & b''_{1''2'}(4) \\ b''_{2''2'}(1) & b''_{2''2'}(2) & b''_{2''2'}(3) & b''_{2''2'}(4) \\ b''_{3''2'}(1) & b''_{3''2'}(2) & b''_{3''2'}(3) & b''_{3''2'}(4) \\ b''_{1''3'}(1) & b''_{1''3'}(2) & b''_{1''3'}(3) & b''_{1''3'}(4) \\ b''_{2''3'}(1) & b''_{2''3'}(2) & b''_{2''3'}(3) & b''_{2''3'}(4) \\ b''_{3''3'}(1) & b''_{3''3'}(2) & b''_{3''3'}(3) & b''_{3''3'}(4) \end{bmatrix},$$

$$\pi'' = \{\pi''_{i''i'}\} = \begin{bmatrix} \pi''_{1''1'} & \pi''_{1''2'} & \pi''_{1''3'} \\ \pi''_{2''1'} & \pi''_{2''2'} & \pi''_{2''3'} \\ \pi''_{3''1'} & \pi''_{3''2'} & \pi''_{3''3'} \end{bmatrix}.$$

(Note: the size of output probabilities of b' and b'' may be different.)

To compute the probability that an 2dHMM generates a particular pair of sequences of o_1^T, it can be obtained by summing the joint probability of the state sequences q_1^T and observation sequences o_1^T over all the possible couples of state sequences. Given the model parameters ($\lambda = \lambda'$, λ''), the joint probability of two observation sequences and state sequences is $\mathrm{P}(o_1^T, q_1^T|\lambda)$. The probability is

$$\begin{aligned}
\mathrm{P}(o_1^T|\lambda) &= \sum_{q_1^T} \mathrm{P}(o_1^T, q_1^T|\lambda) \\
&= \sum_{q_1^T} \mathrm{P}(o_1^T|q_1^T, \lambda) \times \mathrm{P}(q_1^T|\lambda) \\
&= \sum_{q_1^T} \pi'_{q'_1} \pi''_{q''_1, q'_1} \prod_{t=1}^{T-1} a'_{q'_t, q'_{t+1}} b'_{q'_{t+1}}(o'_{t+1}) a''_{q''_t, q''_{t+1}, q'_{t+1}} b''_{q''_{t+1}, q'_{t+1}}(o''_{t+1})
\end{aligned} \tag{3.21}$$

Therefore, direct calculation of $\mathrm{P}(O|\lambda)$ involves $(3T+2)N'^T N''^T$ multiplications and $(N'^T N''^T$ - 1) sums. In fact there are $N'^T N''^T$ possible couples of state sequences of length T and, for each of them, $(3T+2)$ multiplications and one sum are required. The computation is similar to HMM, it can be reduced by using the forward and backward algorithms. A trellis structure implementation for the computation of $\alpha_t(i', i'')$ with 2 states in model λ' and 2 states in model λ'' is shown in Figure 3.9.

- **Forward algorithm**

  The forward function is defined as the probability (at a given instant t) of having observed a partial sequence o_1^t, while the models λ' and λ'' are in states i' and i'' respectively.

$$\begin{aligned}
\alpha_t(i', i'') &= \mathrm{P}(q_t = (i', i'')), & t = 1 \\
\alpha_t(i', i'') &= \mathrm{P}(q_t = (i', i''), o_1^t = y_1^t), & 2 \le t \le T
\end{aligned} \tag{3.22}$$

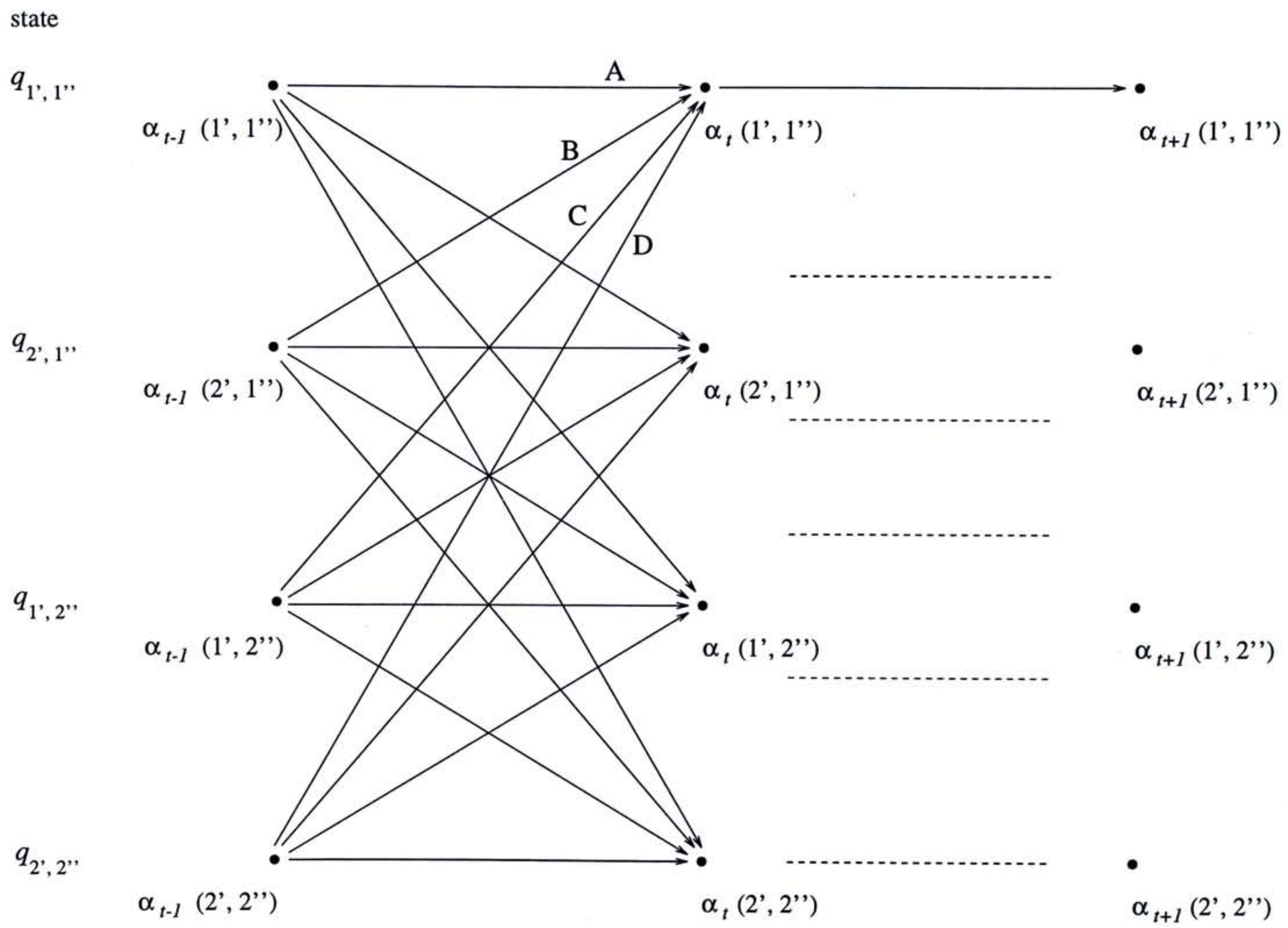
  or

$$\begin{aligned}
\alpha_1(i', i'') &= \pi'_{i'} \pi''_{i'', i'}, & t = 1 \\
\alpha_t(i', i'') &= \sum_{j', j''} \alpha_{t-1}(j', j'') a'_{j', i'} b'_{i'}(o'_t) a''_{j'', i'', i'} b''_{i'', i'}(o''_t), & 1 < t \le T
\end{aligned} \tag{3.23}$$

- **Backward algorithm**

  Similarly, the backward function is defined as the probability (at a given instant t) of observing the remaining part of the sequence o_{t+1}^T, given that λ' and λ'' are in states i' and i'' respectively.

$$\begin{aligned}
\beta_t(i', i'') &= \mathrm{P}(o_{t+1}^T | q_t = (i', i'')), & 1 \le t \le T-1 \\
\beta_t(i', i'') &= \mathrm{P}(q_t = (i', i'')), & t = T
\end{aligned} \tag{3.24}$$



$$\begin{aligned}
\mathrm{A} &= a'_{1',1'} b'_{1'}(O'_t)\ a''_{1'',1'',1'} b''_{1''}(O''_t) \\
\mathrm{B} &= a'_{2',1'} b'_{1'}(O'_t)\ a''_{1'',1'',1'} b''_{1''}(O''_t) \\
\mathrm{C} &= a'_{1',1'} b'_{1'}(O'_t)\ a''_{2'',1'',1'} b''_{1''}(O''_t) \\
\mathrm{D} &= a'_{2',1'} b'_{1'}(O'_t)\ a''_{2'',1'',1'} b''_{1''}(O''_t)
\end{aligned}$$

$$\alpha_t(1',1'') = \mathrm{A}\ \alpha_{t-1}(1',1'') + \mathrm{B}\ \alpha_{t-1}(2',1'') + \mathrm{C}\ \alpha_{t-1}(1',2'') + \mathrm{D}\ \alpha_{t-1}(2',2'')$$

Figure 3.9: A trellis structure for the calculation of the forward partial probabilities $\alpha_t()$ of an 2dHMM with 2-state in λ' and 2-state in λ''.

or

$$\beta_T(i', i'') = 1$$

$$\beta_t(i', i'') = \sum_{j',j''} a'_{i',j'} b'_{j'}(o'_{t+1}) a''_{i'',j'',j'} b''_{j'',j'}(o''_{t+1}) \beta_{t+1}(j', j''), \qquad 1 \leq t \leq T-1 \tag{3.25}$$

Same as HMM in Section 3.2.4, Baum-Welch method can locally maximized the model's likelihood $\mathrm{P}(o_1^T|\lambda)$. Therefore the reestimation of a', a'', b' and b'' can be found by this maximum likelihood (ML) method.

- **Transition Probabilities**

  By using the forward-backward algorithm on the model, the *posterior* probability of transitions $\gamma(i', i'', j', j'')$ from state i' to state j' of λ' and state i'' to state j'' of λ'' is defined as,

$$\begin{aligned} \gamma(i', i'', j', j'') &\equiv \mathrm{P}(q_t = i, q_{t+1} = j | o_1^T, \lambda) \\ &= \alpha_t(i', i'') a'_{i',j'} b'_{i',j'}(o'^{(k)}_{t+1}) a''_{i'',j'',j'} b''_{i'',j'',j'}(o''^{(k)}_{t+1}) \end{aligned} \tag{3.26}$$

The reestimation of $a'_{i',j'}$ is a ratio. The numerator of it is the expected number of times, given the observation set, that the hidden process q' undertakes the transition from i' to j'; the denominator represents the expected number of times, given the observation set, that the process q' is in state i'. Thus, the estimation of the probability $a'_{i',j'}$ is,

$$\bar{a}'_{i',j'} = \frac{\sum_{t=1}^{T} \sum_{i'',j''} \gamma_t(i', i'', j', j'')}{\sum_{t=1}^{T} \sum_{j',i'',j''} \gamma_t(i', i'', j', j'')} \tag{3.27}$$

and the estimation of the probability $a''_{i'',j'',i'}$ is,

$$\bar{a}''_{i'',j'',i'} = \frac{\sum_{t=1}^{T} \sum_{i'} \gamma_t(i', i'', j', j'')}{\sum_{t=1}^{T} \sum_{i',j''} \gamma_t(i', i'', j', j'')} \tag{3.28}$$

- **Discrete Output Probabilities**

  In the discrete case, the reestimation formula for the model λ' is,

$$\bar{b}'_{j'}(y') = \frac{\sum_{t=1}^{T} \delta(y', y'_t) \sum_{i',i'',j''} \gamma_t(i', i'', j', j'')}{\sum_{t=1}^{T} \sum_{i',i'',j''} \gamma_t(i', i'', j', j'')} \tag{3.29}$$

  where

$$\delta(m,n) = \begin{cases} 1 & \text{if } m = n \\ 0 & \text{otherwise} \end{cases}$$

  The output probabilities for λ'' are estimated as,

$$\bar{b}''_{i'',j'',j'}(y'') = \frac{\sum_{t=1}^{T} \delta(y'', y''_t) \sum_{i',i''} \gamma_t(i', i'', j', j'')}{\sum_{t=1}^{T} \sum_{i',i''} \gamma_t(i', i'', j', j'')} \tag{3.30}$$

3.4 Discussions

Template matching with Dynamic Time Warping (DTW) is the first generation of technology used for speech recognition. However, the use of DTW involves massive computation in the matching process. There are also limitations in using DTW for matching with large vocabularies, continuous speech, and speaker independence since a single template can not describe the full variability of the pronunciations. An alternate approach which may solve the shortcomings is a statistical method – the Hidden Markov Model (HMM) [51]. This is a second generation of technology that put the variability in the model instead of the matching process.

An HMM can be used to create word models from examples of full words, or to create them as phonetic models by putting together models of the phonemes composing the word. If the speech from which the word models were created comes from many speakers, the recognition system becomes speaker-independent. Some systems with a speaker-independent model can be made to adapt to an individual's speech characteristics by post training.

Discrete HMM models a single set of feature from the speech waveform. It is anticipated that if more features are used for modeling, high recognition rate might be obtained. It can be imagined that a speech signal will be more distinct from others if it is represented by various kinds of features in different domains. For example, if we want to recognize the way where we go, one method is to remember the properties of the road which could be a corner, traffic light, special building, etc. The chance to remember the way we want to go will increase, if we can observe more than one kind of properties at the same time. Therefore, if two different types of feature sets are extracted that can characterize a speech signal in different perspective, undoubtedly they can provide more information about the signal. The standard HMM theory is, therefore, extended to 2dHMM.

The auxiliary HMM (λ') is used to weight the main HMM (λ'') for calculating its model parameters. The probability of a state at a particular time of an auxiliary HMM is used to modulate the main HMM. This approach can enhance the modeling of the speech samples. For two feature sets of an 2dHMM to be the same, the auxiliary HMM in 2dHMM can be treated as a dummy which provides its state observation for the main HMM. Since the feature kind of this dummy HMM is the same as the main HMM, therefore, it is fed by the same observation sequence. This approach can be used to contrast a standard HMM. For example, two utterances with the same pronunciation generated at different time by a speaker (or different speakers) are to be modeled by 2dHMM. Therefore two statistically models are merged into a single model, 2dHMM. In this methodology, there should be improvement within limited speech training samples for 2dHMM comparing with HMM. It is because the training number in 2dHMM is virtually more than that of HMM with the same training number. Essentially, we allow inter-training of two different observation sequences in 2dHMM. However, two different kinds of feature should be used instead of one kind of feature in order to enhance the model further. Normalization of two speech samples is necessary for training. The technique for it can be achieved by using DTW to minimize error of phone duration between two signals.

Chapter 4

Implementation

4.1 Transputer based multiprocessor system

Although systems with more than one microprocessor offer an attractive method of increasing the processing power of a microcomputer, problems arise because of the need for communication among the microprocessors. Because of the need to share the workload between the microprocessors, the communication overheads detract from the potential improvement which might be expected from using several microprocessors. Hence adding extra processors may lead to a decrease in performance [52]. The transputer represents a solution to this problem in which communication with similar devices is designed into the circuit at a fundamental level. There are three main types of first-generation transputers:

- the 32 bit transputers with a floating point unit (known as the T8xx),
- the 32 bit transputers without a floating point unit (known as the T4xx)
- and the 16 bit transputers (known as the T2xx).

All of them have the same architecture and essentially similar instruction sets. Details of T800 transputer used for our work may be found in Appendix G. This 32 bit floating point transputer is designed for numerical applications. It has 4 kbytes of internal memory, and a floating-point unit (FPU) which is capable of operating to the IEEE-754 specification on 32- and 64-bit numbers.

Transputer systems can be designed to be scalable, since a transputer system is naturally modular so that a design may be able to use one, two or hundreds of transputers. With carefully design, the performance of a product can be increased by adding more transputers without redesigning the software or interfaces. The INMOS standard hardware modules are called TRAMs (TRAnsputer Modules). They are produced by a range of manufacturers and all fit into standard board slots.

4.1.1 Transputer Development System (TDS)

The majority of transputer will probably end in embedded computer systems, where the transputers act as the controlling processors for a device such as a laser printer. These transputers will be completely under the control of the application program. It is very unlikely that they will be used with any underlying operating system [53]. However, for development purpose in our work, a server program that communicates with the transputer system resides in a host computer providing user interface. For changing to a different host, all that required to port to a new host is a relatively simple server program. The server model like this also has the advantage that developers can continue to use the facilities of a familiar operating system, such as its command language and utility programs. The INMOS Transputer Development System (TDS) is shown in Figure 4.1. The principal language provided with the TDS was occam. Besides occam, software toolset of C, Fortran, Ada and Pascal are available from different sources (vendors). The 'iserver' command in the toolset can load programs onto transputers and transputer boards. It loads the bootable file onto the transputer network and activates the host file server that provides communication with the host.

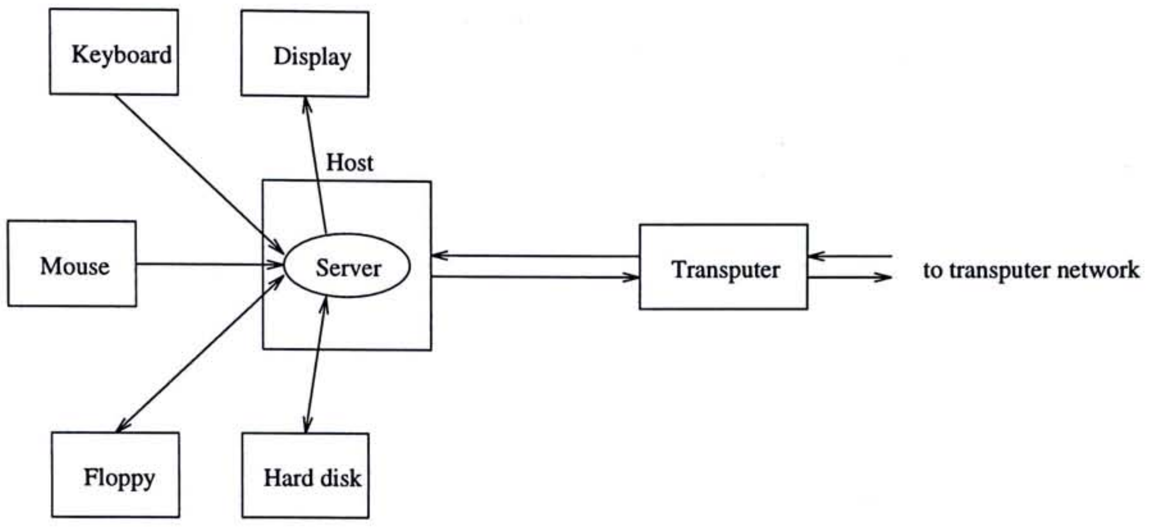


Figure 4.1: The INMOS TDS: the host is connected to the transputer network by a single link.

4.1.2 System architecture

At board level, a transputer system is based on a mother board with slots for plug in TRAM modules. A host computer is required to interface to a mother board to provide user interface and other peripheral resources. With a host computer and a TRAM mother board, a user is able to construct a multi-transputer system by mounting TRAMs onto the mother board. Each member of the TRAM family consists of one transputer and 1, 2, 4 or 8 mega bytes of memory. Figure 4.2 shows a 1 Mbyte DRAM TRAM which is the smallest size of TRAM that measures about 3.5 inches long by 1 inch wide. This is called a Size 1 TRAM. The size of larger

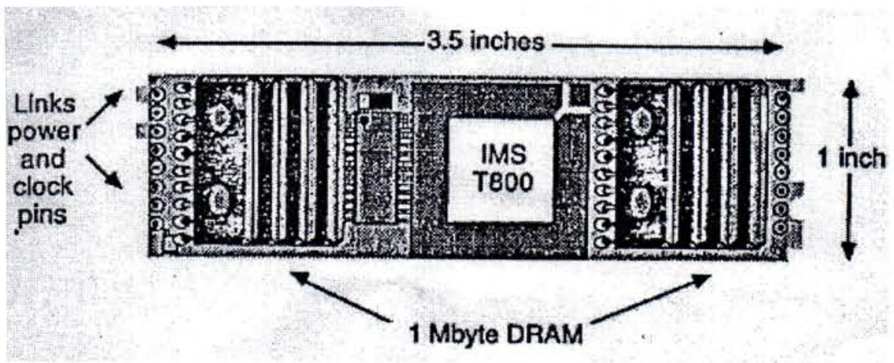


Figure 4.2: Compact transputer system size (IMS T800).

TRAMs is always a multiple of the Size 1 TRAM. For example, the Size 4 TRAM is about the size of four Size 1 TRAMs together. The Size 1 TRAM has 16 pins which plug into sockets on the motherboard. TRAMs are marked in one corner (pin 1) for orientation purposes.

Each mother board has a software control cross bar switch for connecting TRAMs in configurations specified by the user. For each TRAM, four channels are provided for communication with other TRAMs. When they are installed onto a mother board, two of the channels are hardwired to form an array. The remaining two channels are connected to the cross bar switch. Being software programmable, the cross bar switch allows a user to configure its connections. This mechanism provides simple means to configure the architecture of a multi-transputer system.

4.1.3 Transtech TMB16 mother board

The TMB16 is a TRAM mother board designed to plug into a PC ISA bus. The board has 10 TRAM slots and an IMS C004 link switch to allow networks of TRAMs to be set up under software control. Figure 4.3 shows a mother board which can accommodate 10 TRAM modules and is designed to interface to a PC through the ISA expansion slot. Note that the slots are not all oriented the same way and that the ordering of slots is not continuous. This allows better utilization of the mother board when plugging in TRAMs of different sizes. The mother board is specially wired so that if it is populated with Size 1 TRAMs then the transputers are all connected in a pipeline. This is achieved by connecting link 2 of one transputer to link 1 of the next transputer. Figure 4.4 shows the hardwired connection.

The control architecture (shown in Figure 4.5) is that the host computer controls only one transputer (the root or master transputer). This configuration is for TDS users where the transputer development system runs on the master processor. All other processors in the network are controlled from the master processor's

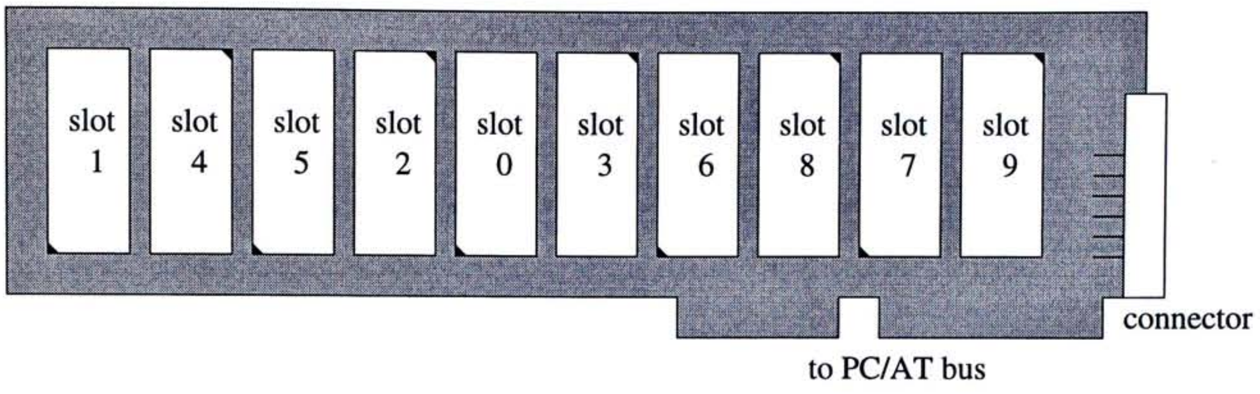


Figure 4.3: TRAM layout of an interfacing mother board.

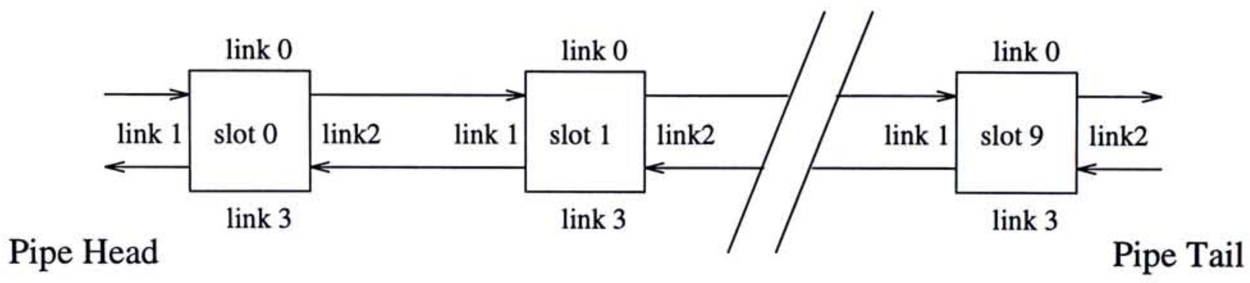


Figure 4.4: TMB16 mother board default transputer pipeline.

subnetwork. This enables TDS to boot/debug user programs.

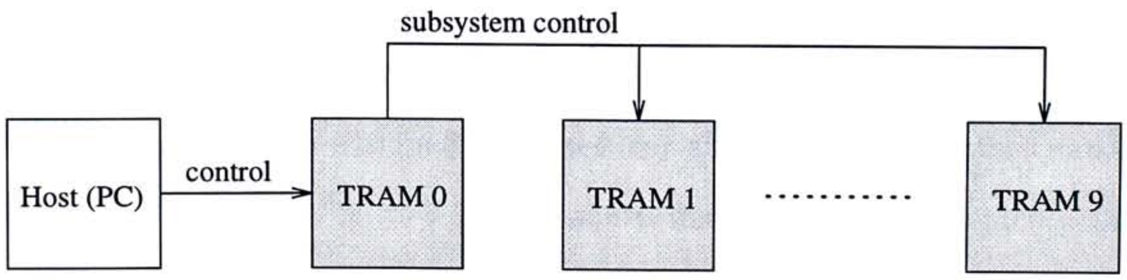


Figure 4.5: The control architecture for TDS.

When TRAMs which sizes are larger than one are used, they do not use all of the sites underneath them. The only active site is the one below pin 1 of the TRAM. This means that the pipeline is broken at the unused slots underneath the TRAM. To bridge these breaks, a special pipe jumper can be used. Figure 4.6 shows a pipe jumper.

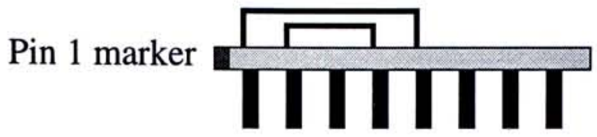


Figure 4.6: Pipe jumper for TMB16 mother board.

TRAM can exist in other forms such as an Analogue to Digital (A/D) converter. For example, the adt108 A/D TRAM is 100kHz sampling Analogue to Digital Converter Module built to the INMOS TRAM format. It is primarily designed to provide a flexible, low cost method of capturing real-time analog signals. The converted data output is via the standard INMOS link adapter to any one of the four links found in TRAM systems. The adt108 is a Size 4 TRAM conforming to the INMOS electrical and mechanical format for TRAM modules.

4.1.4 Farming technique

A farming technique is a multi-processor structure that consists of a controller processor and a pool of interconnected processors as a processor farm. The controller schedules and assigns tasks among the processors in the processor farm. Each farming processor performs its assigned task, returns the results of the task, and waits for new work as shown in Figure 4.7. The controller keeps track of which processors are busy or idling and repeats the assignment until the master process is completed.

Farming technique for multi-processing is well suited for transputer implementation. The performance of farming technique can be evaluated by considering the time each processor dedicates for communication and the time actually used in computation. An efficient system should keep the communication time to a minimum so that an increase in the number of processors in the network increases linearly the global processing speed. Also the computing time of the tasks should not be too short, otherwise the communication load could be comparable with the processing

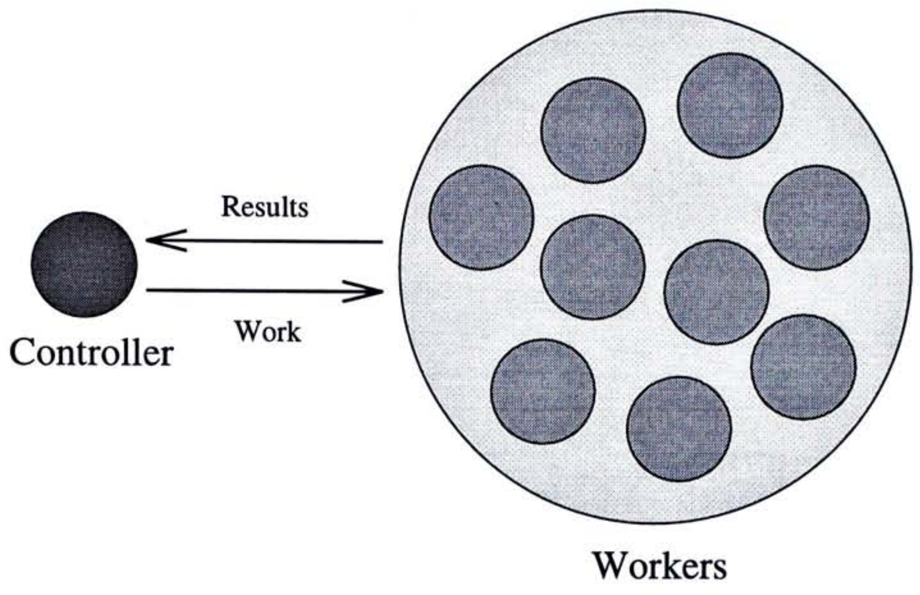


Figure 4.7: A processor farm consists of workers and a controller which assigns tasks to the workers.

load and the network efficiency would degenerate. Three possibles topologies of farm model are shown in Figure 4.8. Basically, the performance of them are similar. The most important factor is the algorithm itself. For maximum efficiency in any farm model, every processor should do the same amount of work (i.e. the work load should be evenly balanced among the processors). Our major advantage of farming is that the size of the processor farm can be expanded easily.

Concerning with efficiency, any specific problem on a parallel machine requires that the problem is divided up into tasks is to be solved. Each task can be done by a processor. If the size of each task is very small, the problem is said to be fine-grained. If each task is large, the problem is said to be large-grained. The granularity (introduced in Section 1.3.3) of a problem is a measure of the size of the tasks which a problem is divided into. If a problem is inherently large-gained and the difficulty of each subtask is very different from that of other subtasks, it will be difficult to balance the computing load using a processor farm. However, processing in speech recognition can be divided up into increasingly smaller pieces and the granularity of the tasks can be arbitrarily decided. Such problems can be handled well with processor farms. Since the transputer is a single-chip microprocessor with

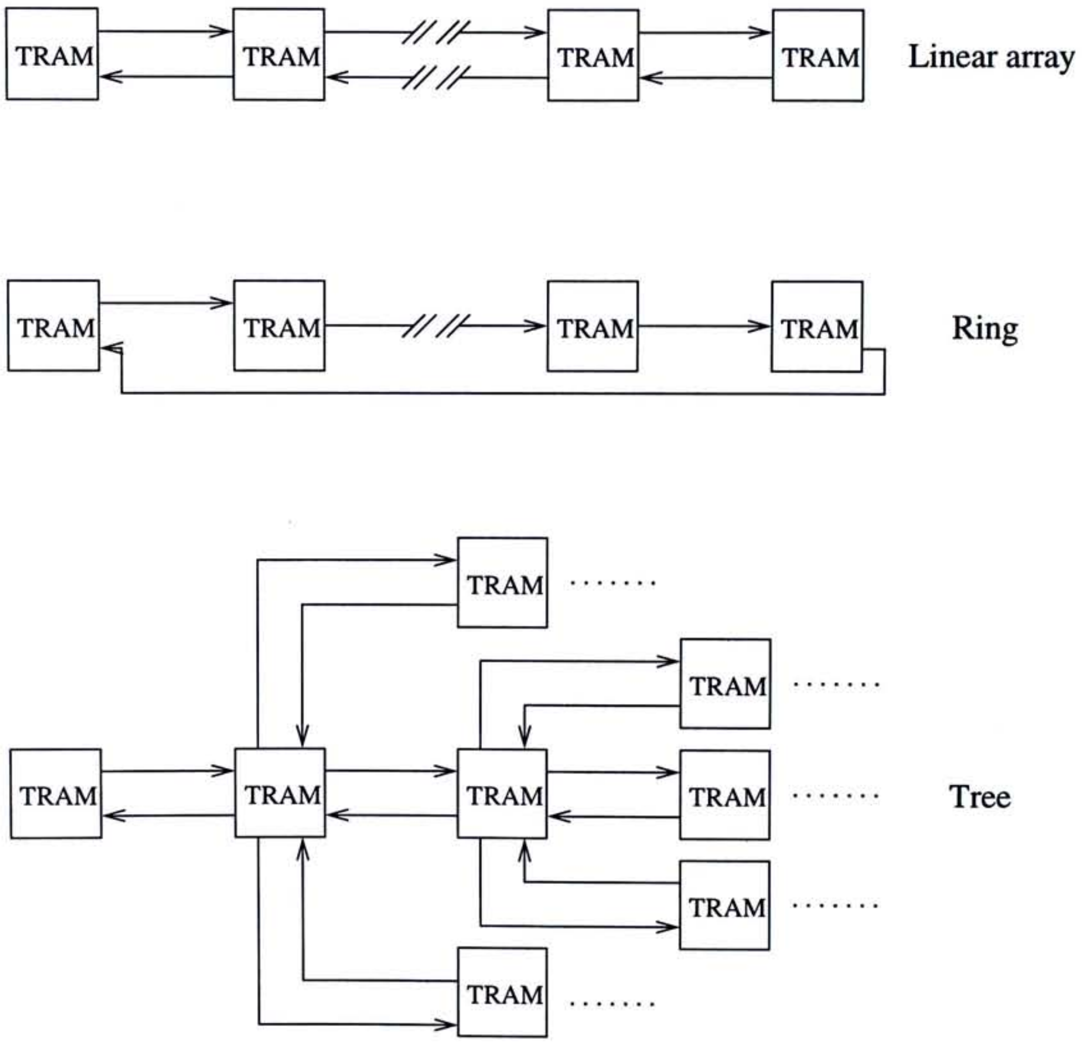


Figure 4.8: Farm models (linear array, ring and tree).

significant communications capability, it qualifies as a medium-grained computer. Thus a processor farm composed of transputers should be generally be given a collection of medium-grained tasks [22]. Since speech recognition involves complicated computation, it will be suitable to employ farming technique with transputers.

Transputer configuration in the system

A PC based five-transputer system configured as processor farm was used as a development platform for our work. The block diagram of the system is shown in Figure 4.9. The 80486 (33 MHz) host PC had 8 Mbytes of memory, a 10 slot TRAM

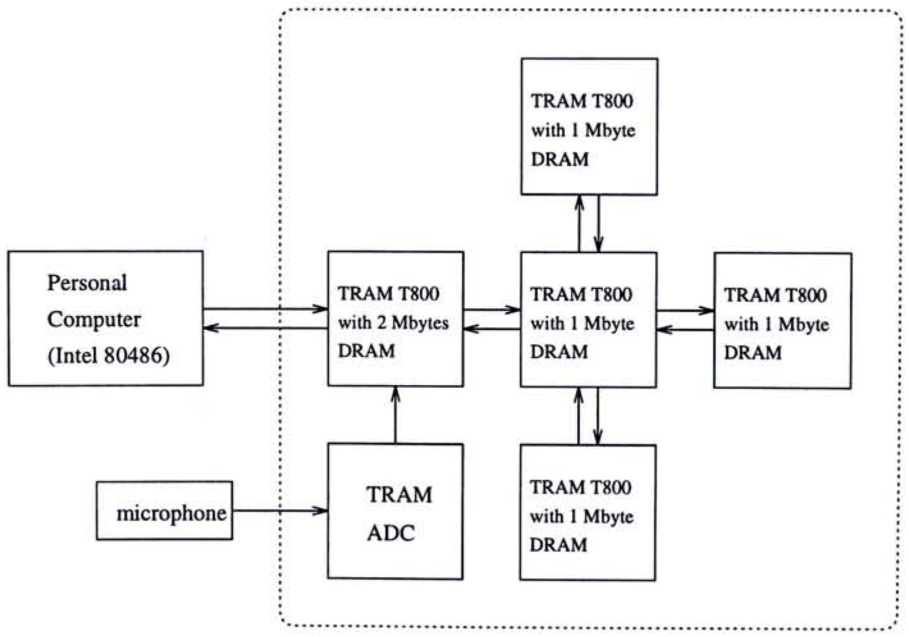

TRAM T800 with 1 Mbyte DRAM
Personal Computer (Intel 80486)
TRAM T800 with 2 Mbytes DRAM
TRAM T800 with 1 Mbyte DRAM
TRAM T800 with 1 Mbyte DRAM
microphone
TRAM ADC
TRAM T800 with 1 Mbyte DRAM

Figure 4.9: Development system block diagram.

mother board installed on ISA expansion slot and ran under the MSDOS operating system.

The host PC provided the user interface and hard disk storage for program development. The software development platform was based on the INMOS ANSI C toolset [54] software cross-development system. ANSI C language is used to specify the components of a system in terms of communicating processes. The design can be directly expressed in the parallel constructs of the language.

4.2 Farming technique on extracting spectral amplitude feature

To illustrate the potential of this development engine, spectral analysis (by taking FFT) of a speech segment is performed on each and every short-time frame. There are 30 about frames for a typical isolated syllable, therefore if FFT is needed to be done on each frame, it means a lot of duplication of process with different input data. Hence, it is well suited to employ farming technique for parallel processing in performing this task. The implementation is described as follows,

1. For FFT calculation on 30 frames of speech waveform with a five transputer system, each transputer will take 6 frames to share the work. Three processes are required to complete the task. One for master transputer, one for junction transputer and one for slave transputers. The process in master transputer distributes data (part of digitized speech signal) and parameters to junction transputer. The junction transputer which receives the data and parameters retains its share of work, then distributes the rest of the work (data and parameters) to the 3 slave transputers. All transputers including master, junction and slave will start the FFT calculation as soon as they receive the data. The junction transputer waits for results from the slave transputers before sending them to the master transputer together with its own result. On receiving the results from the junction transputer, the master transputer combines with its own to form the complete spectral analysis result. The source codes of the processes are written in parallel C language which is standard C language with additional function for concurrency such as channel communication. Figure 4.10 shows the activities between trasnputers in the network. For communication between transputers, a path is defined as a channel between two transputers at the programming level [55]. They are point-to-point unidirectional connections and the transfer of data is one way in order to provide maximum

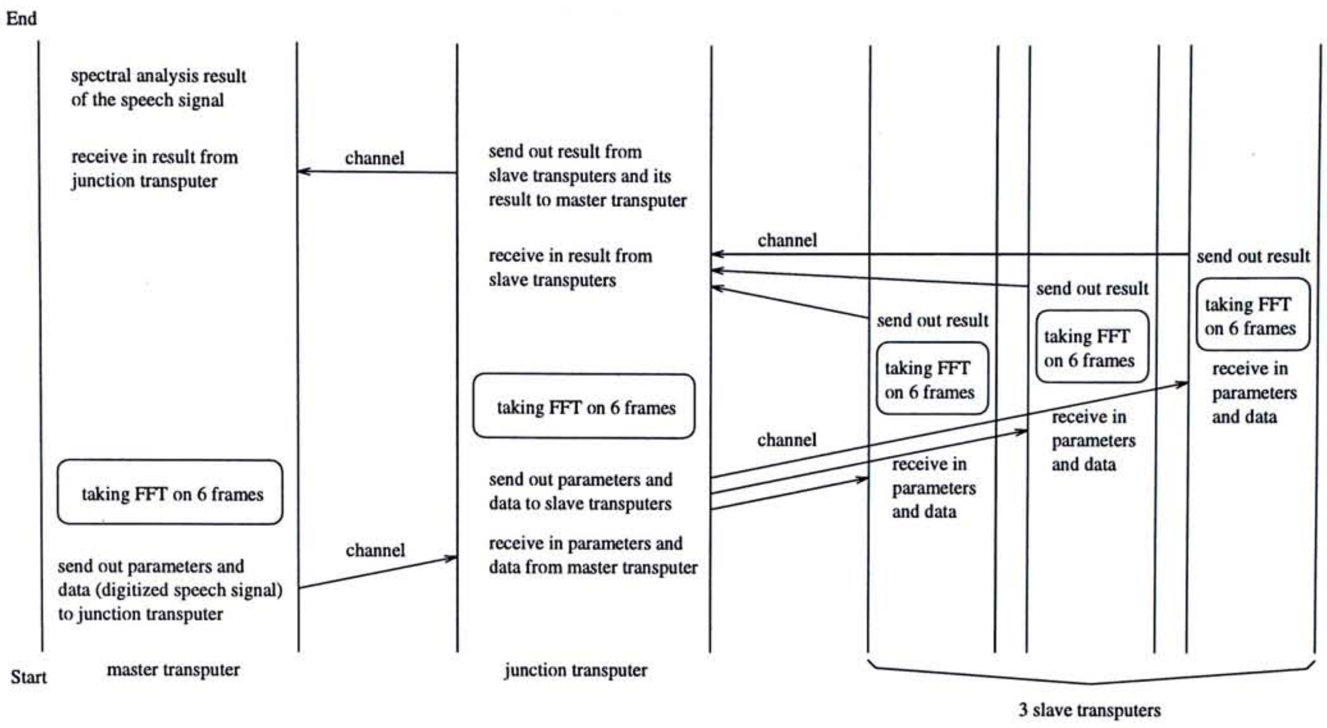

End
spectral analysis result of the speech signal
receive in result from junction transputer
channel
send out result from slave transputers and its result to master transputer
receive in result from slave transputers
channel
send out result
taking FFT on 6 frames
send out result
taking FFT on 6 frames
send out result
taking FFT on 6 frames
receive in parameters and data
taking FFT on 6 frames
receive in parameters and data
channel
receive in parameters and data
taking FFT on 6 frames
send out parameters and data to slave transputers
channel
receive in parameters and data from master transputer
send out parameters and data (digitized speech signal) to junction transputer
Start
master transputer
junction transputer
3 slave transputers

Figure 4.10: Activities among the transputer network for computing FFT on 30 frames of speech signal.

speed with minimal wiring. Therefore, processes which exchange data with each other must form a pair of channels.

2. For transputers on TMB16 mother board, the connections between their link 0 and link 3 are connected by the programmable link switch (IMS C004). In order to build the five-transputer network (its link connection is shown in Figure 4.11), TRAMs s3 and s4 are needed to be connected to TRAM s1 through the softwired links. A text file (with extension ".wir") is edited as:

```
y
s1 0 t s4 0.
s1 3 t s3 3.
```

Then a program of network configuration software with the command "ncs" is run to read this file and make the connection.

3. Before the program is compiled, a configuration file (fft.cfs) is necessary to be defined as:

```
T800 (memory = 2M) root;
```

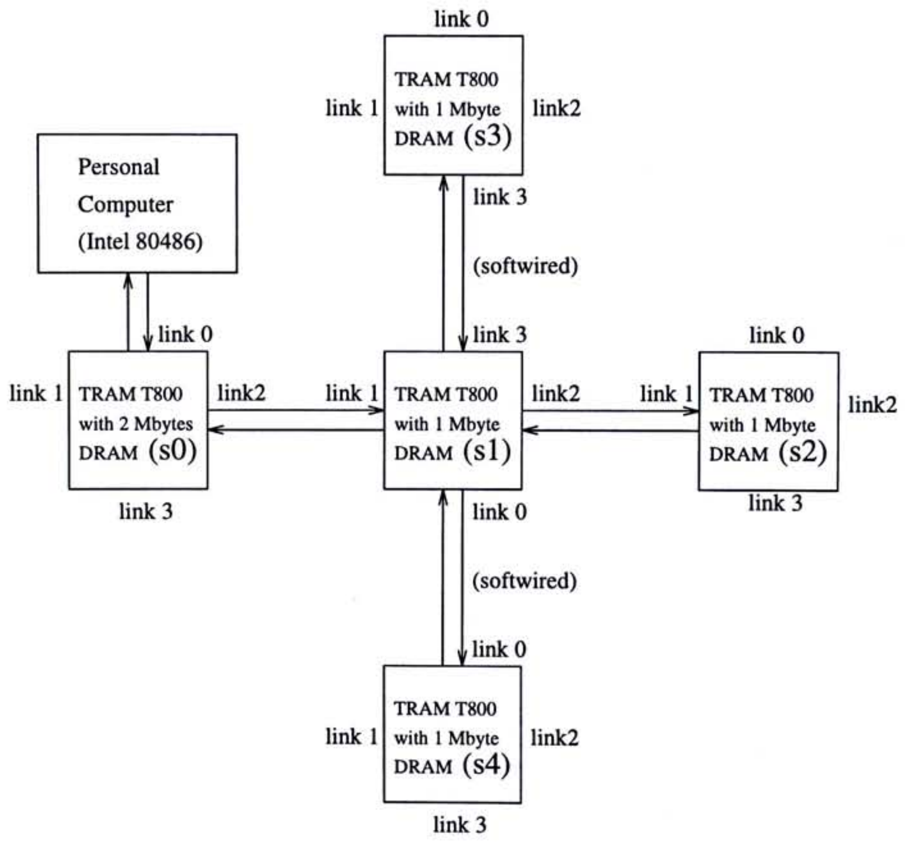

link 0
link 1
TRAM T800 with 1 Mbyte DRAM (s3)
link2
link 3
(softwired)
Personal Computer (Intel 80486)
link 0
link 3
link 1
TRAM T800 with 2 Mbytes DRAM (s0)
link2
link 3
link 1
TRAM T800 with 1 Mbyte DRAM (s1)
link2
link 0
link 1
TRAM T800 with 1 Mbyte DRAM (s2)
link2
link 3
link 0
(softwired)
link 0
link 1
TRAM T800 with 1 Mbyte DRAM (s4)
link2
link 3

Figure 4.11: Taking FFT for a speech signal on the five-transputer network with farming technique.

```
T800 (memory = 1M) s[4];

connect root.link[0] to host;
connect root.link[2] to s[0].link[1];
connect s[0].link[0] to s[1].link[0];
connect s[0].link[3] to s[3].link[3];
connect s[0].link[2] to s[2].link[1];

input from_server;
output to_server;

process(stacksize=400k, heapsize=900k,
        interface(input HostInput, output HostOutput,
                input in, output out,
                input adt_in, output adt_out))master;

process(stacksize=40k, heapsize=200k,
                interface(input up_in, output up_out,
                input down_in[3], int InputSize = 3,
                output down_out[3], int OutputSize = 3))junction;

process(stacksize=40k, heapsize=200k,
                interface(input in, output out))slave[3];

connect master.HostInput to from_server;
connect master.HostOutput to to_server;
connect master.in to junction.up_out;
connect master.out to junction.up_in;

rep i=0 for 3{
        connect junction.down_in[i] to slave[i].out;
        connect junction.down_out[i] to slave[i].in;
}

use "m_fft.lku" for master;
use "j_fft.lku" for junction;
rep i=0 for 3{
        use "s_fft.lku" for slave[i];
}

place master on root;
place junction on s[0];
rep i=0 for 3{
        place slave[i] on s[i+1];
}

place from_server on host;
place to_server on host;
place master.in on root.link[2];
place master.out on root.link[2];

place junction.down_in[0] on s[0].link[0];
place junction.down_out[0] on s[0].link[0];
place junction.down_in[1] on s[0].link[2];

place junction.down_in[2] on s[0].link[3];
place junction.down_out[2] on s[0].link[3];
```

From the configuration file, the source code "m_fft.c" is defined as master, "j_fft.c" is defined as junction and "s_fft.c" is defined as slave.

4. With source codes and configuration file, the designed program can be compiled. First, each source code is compiled individually by "icc" which is an ANSI standard C compiler with additional support for concurrency to form an object file in a standard intermediate code format which then linked (by "ilink") to generate a linked unit. Linked units can be used in configuration descriptions to map software onto specific arrangements of transputers. Finally, "icconf" and "icollect" are used to generate a bootable file (with extension ".btl") which can be directly loaded onto the transputer network by using the

host file server tool "iserver" which both loads the program and starts up the runtime environment that supports interaction with the host. The configurer "icconf" generates configuration information for transputer networks from the configuration file (fft.cfs). The code collector tool "icollect" takes the data file generated by "icconf" and generates a single file that can be loaded and run on the transputer network.

The batch file (make.bat) for the compiling procedure is:

```
icc m_fft.c /ta
icc j_fft.c /ta
icc s_fft.c /ta
ilink m_fft.tco /f startup.lnk /ta
ilink j_fft.tco /f startrd.lnk /ta
ilink s_fft.tco /f startrd.lnk /ta
icconf fft.cfs
icollect fft.cfb
```

By making use of the transputer, it can be seen that the computation required in extracting feature parameters can be sub-divided into small tasks and performed in parallel. This enables the viability of implementing the speech recognition system in real time.

4.3 Feature extraction for LPC

In order to obtain an accurate result for performance evaluation. Hidden Markov Model Toolkit (HTK) was used to perform the pre-processing role, such as feature extraction and generation of a codebook. Since the toolkit was used, the system at this moment would not be implemented in real time. The software toolkit was hosted on "SUN" workstation. Details of using HTK can be referred to the reference manual [56].

In the toolkit, there are totally 11 options for selecting the basic kind of speech feature. They are summarized as:

1. sampled waveform
2. linear prediction filter coefficients
3. linear prediction reflection coefficients
4. LPC cepstral coefficients
5. LPC cepstra plus delta coefficients
6. LPC reflection coef in 16 bit integer format
7. mel-frequency cepstral coefficients
8. log mel-filter bank channel outputs
9. linear mel-filter bank channel outputs
10. user defined sample kind
11. vector quantized data

Moreover energy, delta or acceleration coefficients can be appended optionally to its feature vector.

In our case, feature of LPC cepstral coefficients were used for speech signals. The feature vector size was 10. An energy of a frame were appended to its feature vector to make the size being 11. In addition, delta features of them were further appended to the original feature vector. As a result, the feature vector contained 22 elements. It was shown in Equation (4.1).

$$\text{frame feature} = (C_1, C_2, C_3, ..., C_{10}, E, \delta C_1, \delta C_2, ..., \delta C_{10}, \delta E) \tag{4.1}$$

"HCopy" and "HQuant" Tools

- HCopy

The function of "HCopy" is to convert a raw data (waveform) file to a designated output file. The designated output in our case is LPC cepstral coefficients with energy and their delta information as in Equation (4.1). The function is invoked by typing the command line:

```
HCopy -C config_filename input_filename output_filename
```

The configure file, config_filename:

```
SOURCEKIND = WAVEFORM
SOURCERATE = 1000        # (*100ns) ie 10.0 kHz
TARGETKIND = LPCEPSTRA_E_D
LPCORDER = 10
NUMCEPS = 10
TARGETRATE = 100000      # ie 10 msecs
WINDOWSIZE = 260000      # ie 26 msecs
ZMEANSOURCE = T
USEHAMMING = T
PREEMCOEF = 0.97
SAVEWITHCRC = F
```

Parameters involved in the overall process is illustrated in Figure 4.12 which shows the sampled waveform being converted into a sequence of parameter features. In general, HTK regards both waveform files and parameter files as being just sample sequences, the only difference being that in the former case the samples are 2-byte integers and in latter they are multi-component vectors. The sample rate of the input waveform will normally be determined from the input file itself. However, it can be set explicitly using the configuration parameter SOURCERATE. The period between each parameter vector determines the output sample rate and it is set using the configuration parameter TARGETRATE. The segment of waveform used to determine each parameter vector is usually referred to as a window and its size is set by the configuration parameter WINDOWSIZE. Notice that the window size and frame rate are independent. Normally, the window size will be larger than the frame rate so that successive windows overlap as illustrated in Figure 4.12.

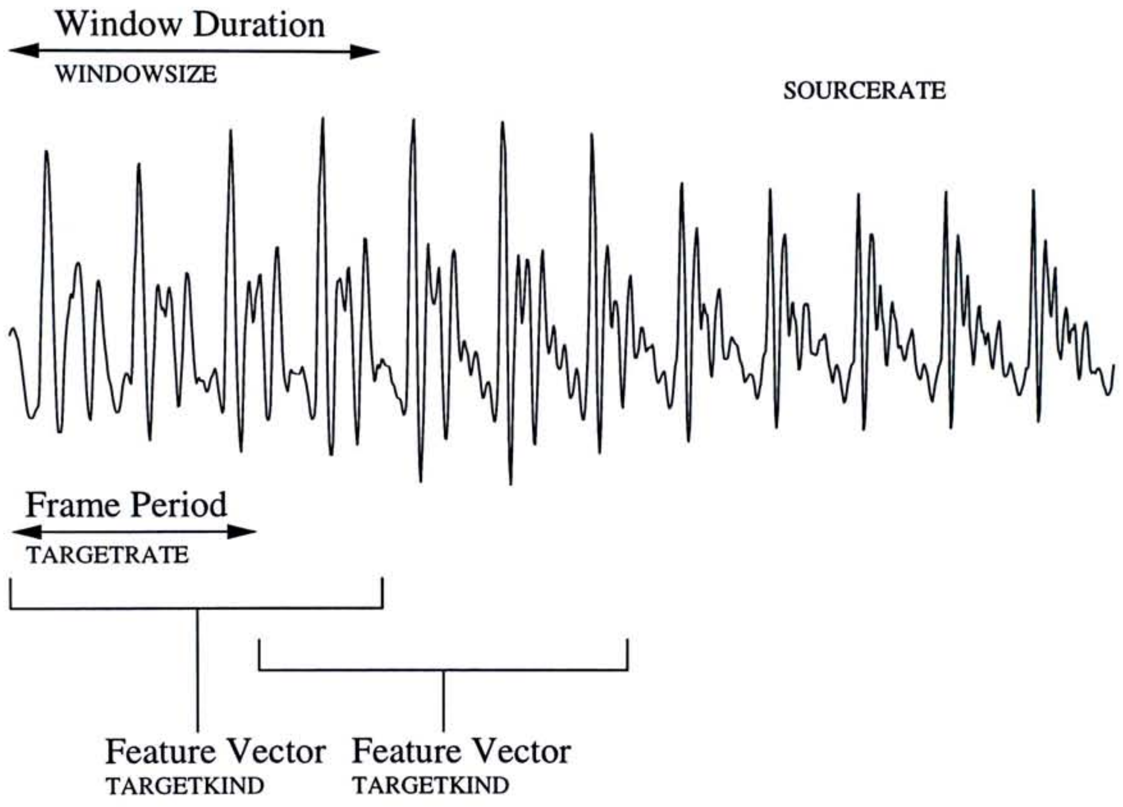


Figure 4.12: Parameters for speech encoding process by HTK.

- HQuant

  The function of "HQuant" is to creat a codebook from the speech samples into a parameterized form. Codebook construction consists of finding clusters in the training data, assigning an unique reference vector (the cluster centroid) to each, and storing the resultant reference vectors in a codebook. The function is invoked by typing the command line:

  ```
  HQuant -n S N codebook_filename -S train_files
  ```

  where S=stream, N=codebook size and train_files=script file.

4.4 DTW based recognition

4.4.1 Feature extraction

In template matching, one popular method to represent the characteristic of speech signal is to compute and record its spectral amplitude in frequency domain. The spectral amplitudes of the speech signal in each small segment are mapped to 12 frequency bands forming feature vectors. Figure 4.13 shows the block diagram of a speech signal which the feature vectors are computed after end-point detection to be used as reference templates to be recognized. In this feature extraction process,

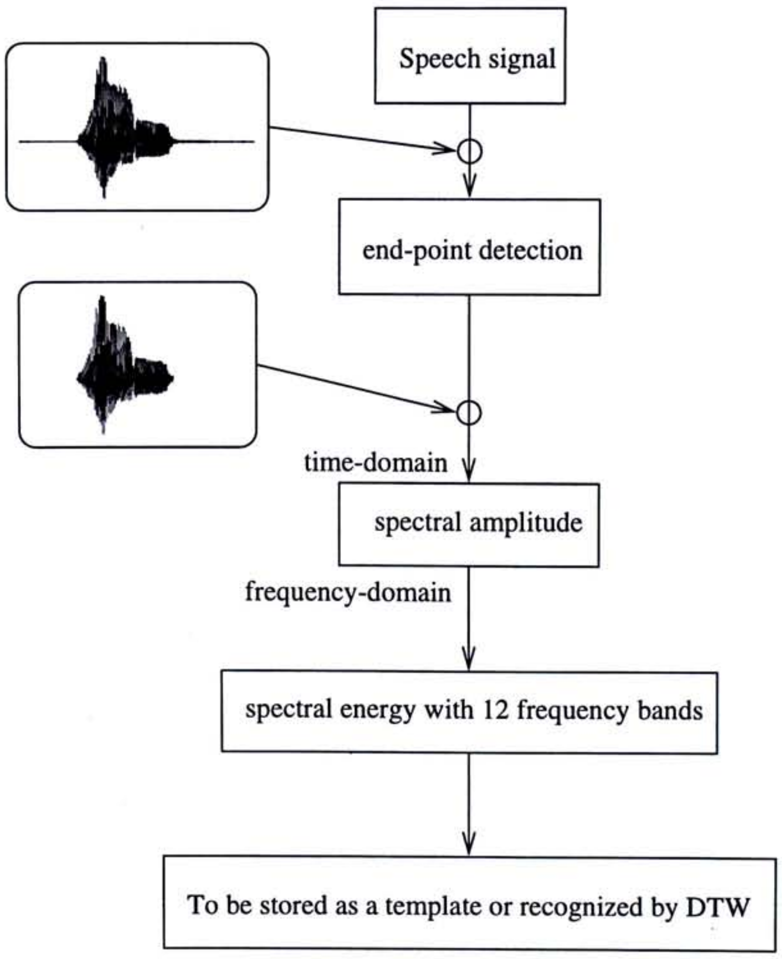

Speech signal
end-point detection
time-domain
spectral amplitude
frequency-domain
spectral energy with 12 frequency bands
To be stored as a template or recognized by DTW

Figure 4.13: Block diagram of a speech signal from which features are extracted after end-point detection and stored as template or recognized by template matching with DTW.

the main algorithm that is used for computing the signal features in the frequency domain is the FFT algorithm. It is possible to split the FFT calculation into parallel components to be run by different processors. However, we found that it was simpler and more effective to create multiple replications of the FFT process to run on individual processor. The programming example of it is shown in Section 4.2. Each transputer takes a portion of the useful signal for processing. Figure 4.14 shows the organization of the five-processor farm for operation on concurrent feature extraction. One of the transputers is used as controller which distributes the task data to different workers (extracting spectral information on specified frames). While workers do their work, the controller is also doing the same task on different frames. After finishing the process, results will be passed back to the controller.

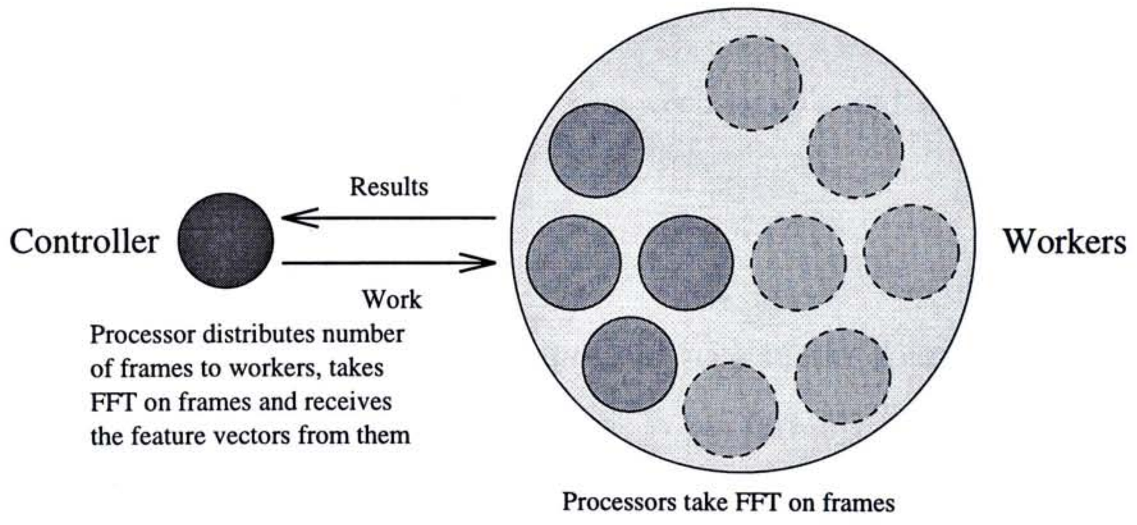

Results
Controller
Work
Workers
Processor distributes number of frames to workers, takes FFT on frames and receives the feature vectors from them
Processors take FFT on frames

Figure 4.14: Five transputers are used as a processor farm for feature extraction on a speech signal.

4.4.2 Training and matching

After the features of speech signals are normalized and extracted, they are stored as reference templates for recognition. Figure 4.15 shows the block diagram of 10 words stored in the transputer network. The feature vectors of each word are stored in local memories among transputers. Since five transputers are used, each transputer shares the loading and takes care of 2 words in the vocabulary. The stored templates

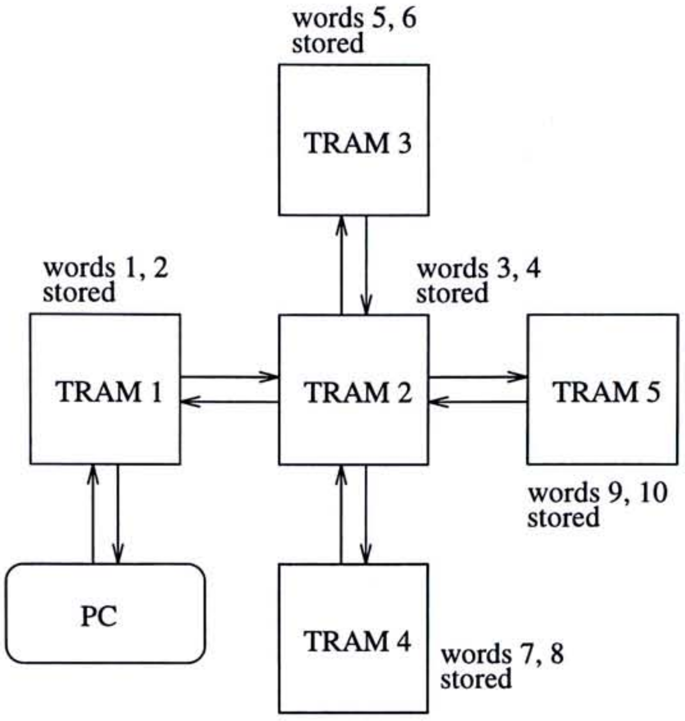


Figure 4.15: Features are extracted from 10 speech signals and stored in the transputer network for template matching with DTW.

are used for matching by using DTW.

For many parallel algorithms, an important part of the solution is to balance the computational load among the processors. When farming technique is employed to perform the job, each processor is distributed with similar work load. With increasing vocabulary size, the matching process simply requires the addition of more transputers to the matching network. The software can easily be changed to adopt adding more transputers at the configuration level. In the five-transputer network with 10 words vocabulary, errors are calculated between a tested speech signal and prestored speech signals. The prestored speech signal with the minimum error should be chosen as the word that is matched to the tested word. Before matching, feature vectors of the tested word are distributed to each transputer. In matching, each transputer calculates the errors of 2 words in the vocabulary and chooses the minimum one. For junction TRAM, besides calculating its minimum error it receives a error from each slave TRAM. Therefore, it is necessary to choose the minimum one again from its minimum error and that of slave TRAMs. Finally, the master TRAM can get the minimum error of all words in the vocabulary. It is illustrated in Figure 4.16.

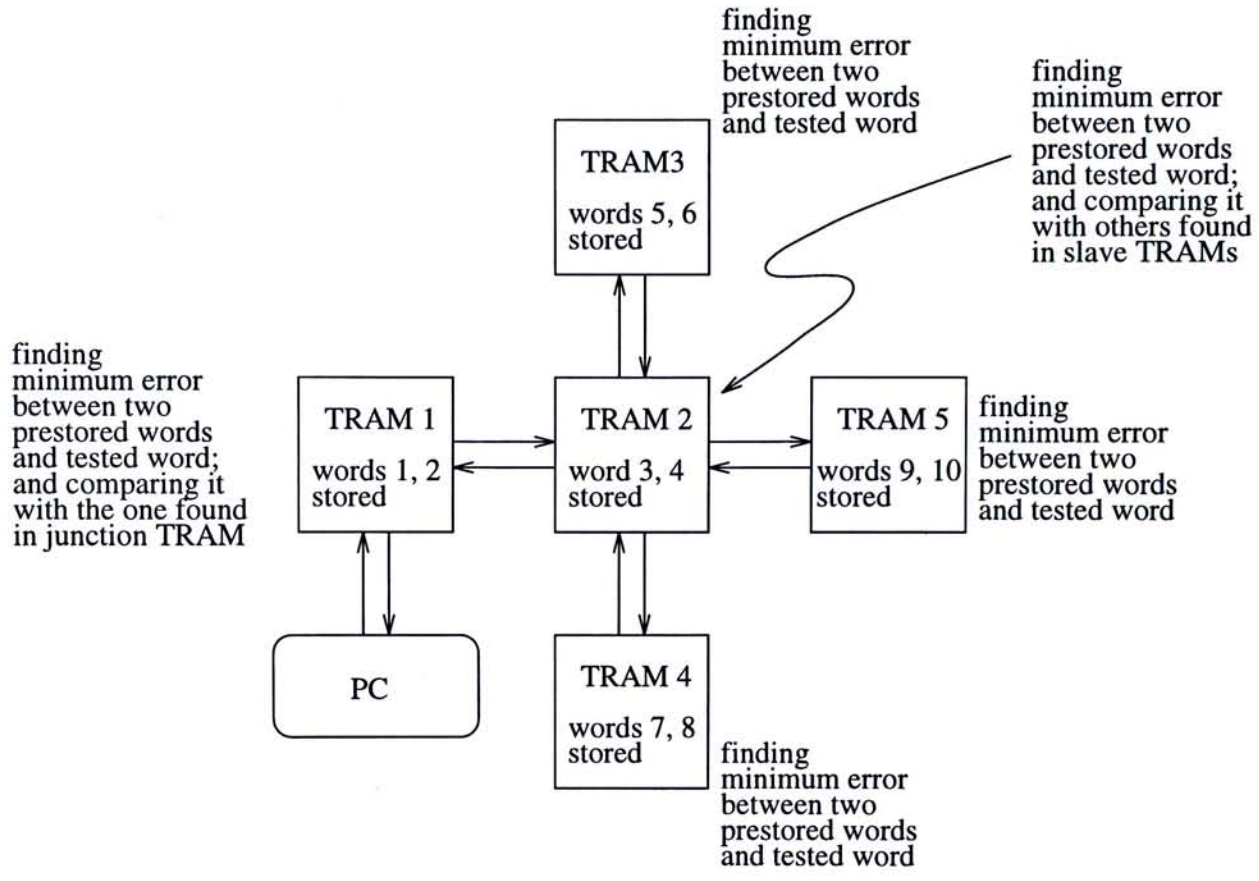

finding minimum error between two prestored words and tested word
finding minimum error between two prestored words and tested word; and comparing it with others found in slave TRAMs
TRAM3
words 5, 6 stored
finding minimum error between two prestored words and tested word; and comparing it with the one found in junction TRAM
TRAM 1
words 1, 2 stored
TRAM 2
word 3, 4 stored
TRAM 5
words 9, 10 stored
finding minimum error between two prestored words and tested word
PC
TRAM 4
words 7, 8 stored
finding minimum error between two prestored words and tested word

Figure 4.16: Activities of a five transputers network in recognizing a speech sample with a 10-word vocabulary by template matching with DTW.

4.5 HMM based recognition

4.5.1 Feature extraction

In the statistical matching method of discrete HMM, VQ is necessary to limit many possible feature vectors to a fixed number used in modeling. The codebook with size 32 could be obtained by HTK Toolkit as well as feature vectors.

4.5.2 Model training and matching

When training model parameters, insufficient training data can affect the performance of the recognizer. For discrete observation an HMM, the re-estimation of $\overline{b}_j(k)$, Equation (3.20) of Section 3.2.4, requires a count of the expected number of times in state j and observing symbol v_k simultaneously. If the training samples are not large enough and there is no occurrence of this event, $b_j(k)$ become zero and will stay zero after re-estimation. As a result, $\mathrm{P}(O|\lambda)$ would be zero for any observation sequence that includes ($o_t{=}v_k$ and $q_t{=}j$). By increasing the size of the training observation set, the problem can be solved.

In re-estimation of model parameters, $\overline{a}_{ij}$ and $\overline{b}_j(k)$ are attempted to be evaluated concurrently to adapt the parallel processing. Referring to their updating Equations (3.19) and (3.20), the computation effort of $\overline{b}_j(k)$ relies mainly on $\overline{a}_{ij}$. It is because that once $\gamma_t(i,j)$ is calculated for $\overline{a}_{ij}$, it can be saved to calculate $\gamma_t(j)$ for $\overline{b}_j(k)$. Therefore, the main computation load goes to the re-estimation of a_{ij}. Comparing with $\overline{a}_{ij}$, the re-estimation time of $\overline{b}_j(k)$ is much shorter. It is not effective to re-estimate them concurrently by using more than one processor. Before training, the features of speech signals have to be extracted, and quantized to a coded sequence in a codebook. After training, model parameters in the vocabulary are obtained and stored for matching purpose. For recognition, feature vectors with quantization of the tested word are distributed to each transputer for matching. The algorithm is shown in Figure 4.17. The larger probability with the word in the vocabulary will be treated as the matched one to the tested speech signal.

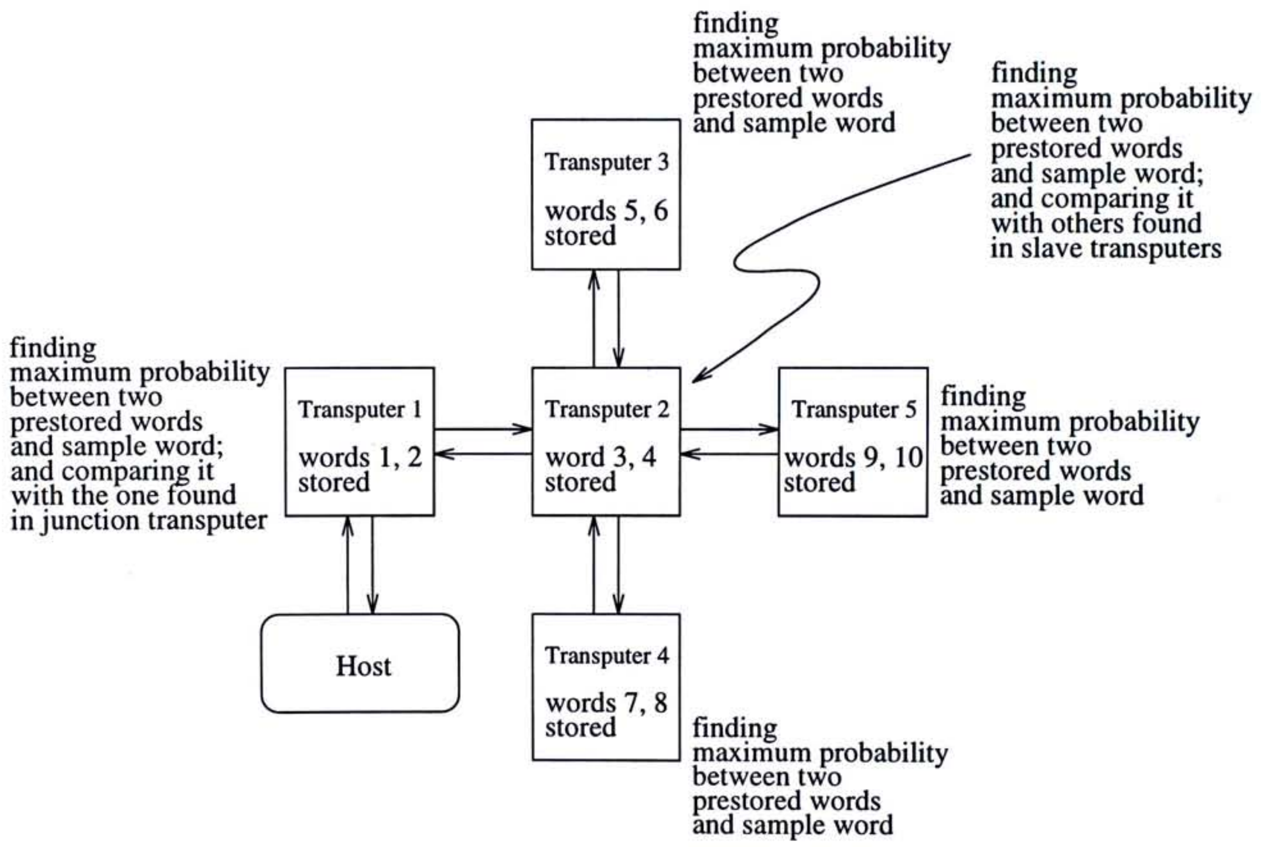

finding maximum probability between two prestored words and sample word
finding maximum probability between two prestored words and sample word; and comparing it with others found in slave transputers
Transputer 3
words 5, 6 stored
finding maximum probability between two prestored words and sample word; and comparing it with the one found in junction transputer
Transputer 1
words 1, 2 stored
Transputer 2
word 3, 4 stored
Transputer 5
words 9, 10 stored
finding maximum probability between two prestored words and sample word
Host
Transputer 4
words 7, 8 stored
finding maximum probability between two prestored words and sample word

Figure 4.17: Activities of a five transputers network in recognizing a speech sample with a 10-word vocabulary by HMM method.

4.6 2dHMM based recognition

4.6.1 Feature extraction

In using 2dHMM for modeling speech signal, two distinctive feature sets can be used to characterize the speech waveform. Due to the natural characteristic of a speech signal, the extracted spectral information is good enough to represent it. On the other hand, the two feature sets can be the same for the model λ' and model λ'' in 2dHMM. The test done for modeling 2dHMM with the English vocabulary is using the two same feature sets of cepstral coefficients.

Being a tonal language, the pronunciation of a Chinese character is characterized by its constituent phonemes and distinctive tone. For Cantonese, two different feature sets, namely tone and cepstral coefficient, are used. Since the two feature sets from a given speech signal are independent to each other, they can be extracted concurrently after the end points of the signal are detected as shown in Figure 4.18. Figures 4.19 and 4.20 show spectral amplitudes and tone features of two utterances pictorially. It can be seen that waveform of the word "tsin1" is similar to that of word "tsin2". However, the tone features of them are different. It is similar to HMM in Section 4.5.1, VQ is applied to limit feature vectors to a finite number. Furthermore, the pitch values of the tone profile are quantized linearly. The extraction of these two different feature sets can be obtained in parallel by using farming technique.

4.6.2 Training

Again, learning algorithm could be implemented in parallel in order to make training more efficient. For 2dHMM training, the re-estimation of a', a'', b' and b'' can be implemented parallelly. On a single processor only, training of one iteration proceeds sequentially as shown at the top of Figure 4.21. The total time is the time required

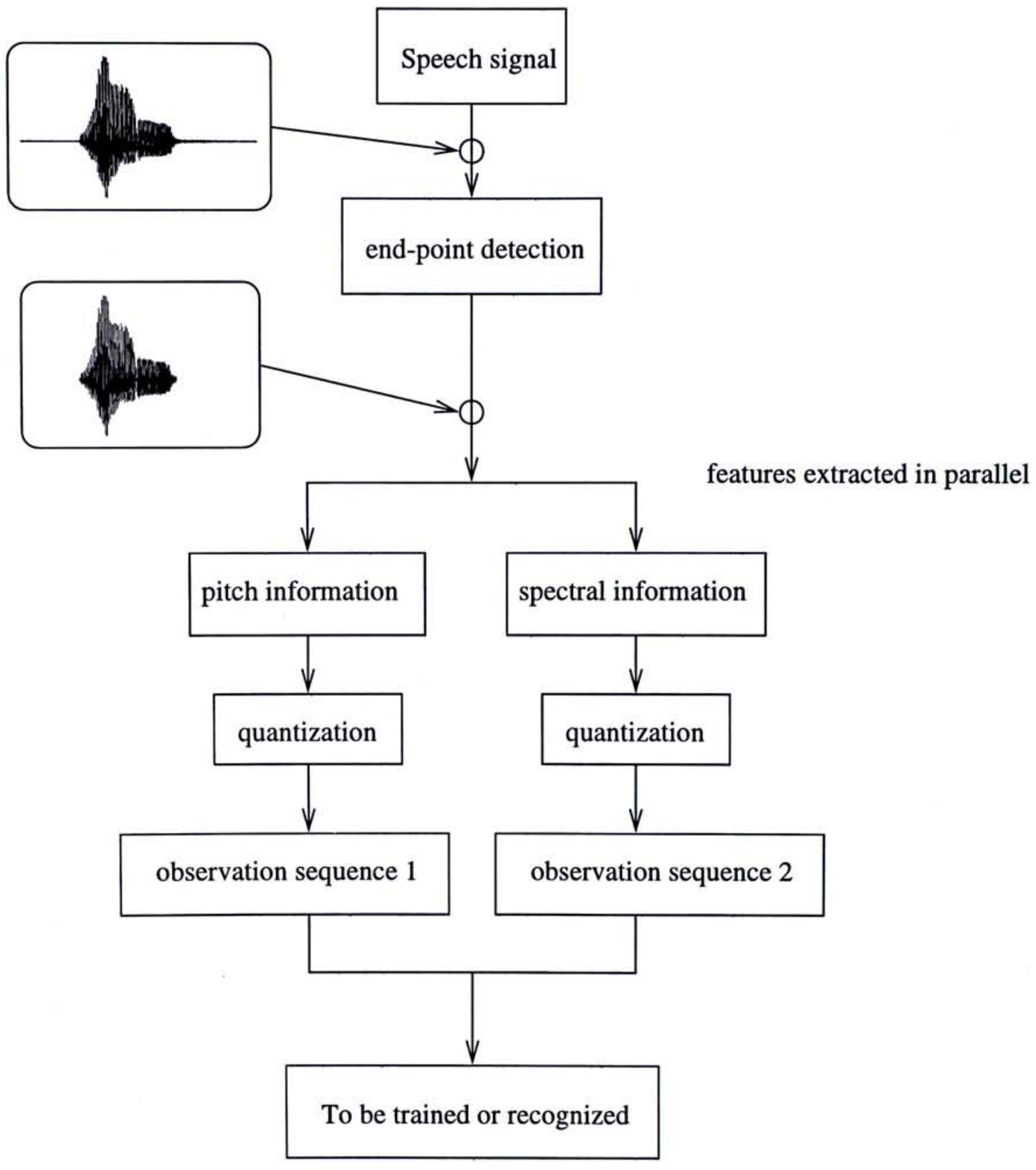


Figure 4.18: Block diagram of a speech signal which two distinctive features are extracted parallelly after end-point detection for 2dHMM method.

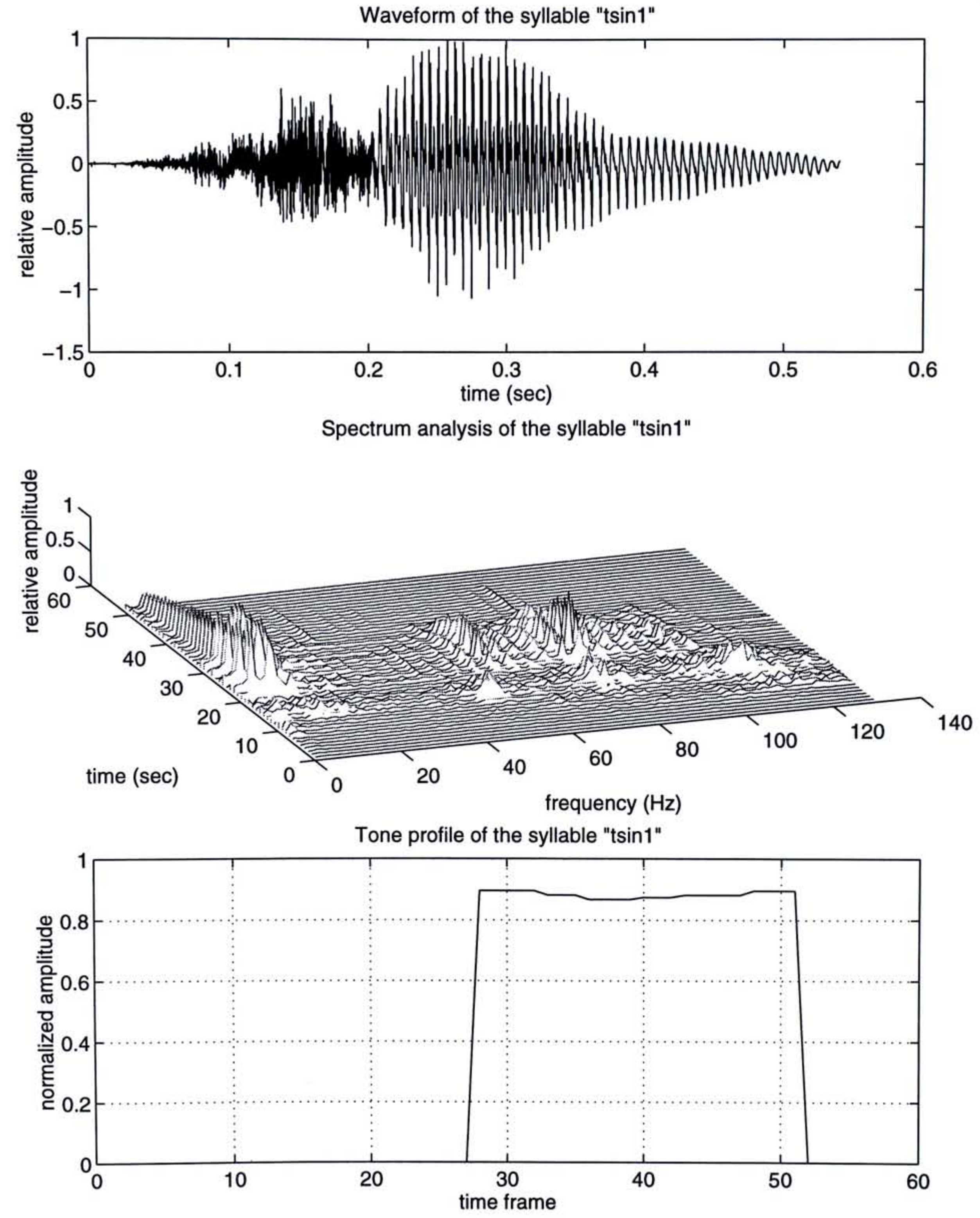


Figure 4.19: Voice signal of a Cantonese syllable "tsin1" and its features that represent it.

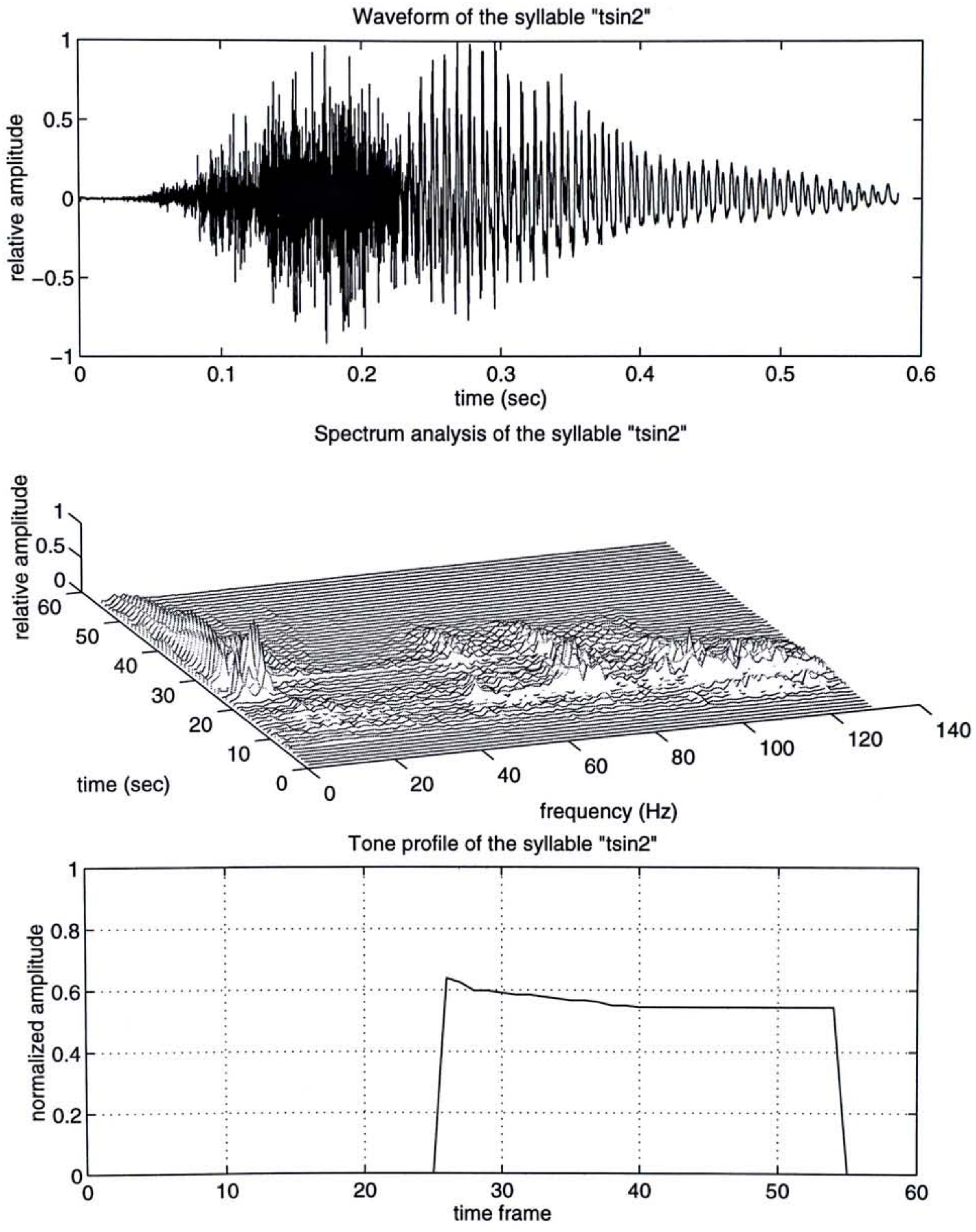


Figure 4.20: Voice signal of a Cantonese syllable "tsin2" and its features that represent it.

to compute forward and backward parameters, and to re-estimate the new model parameters. Several of these steps may be done concurrently, as shown at the bottom of Figure 4.21. Here the forward and backward computations are pipelined with the

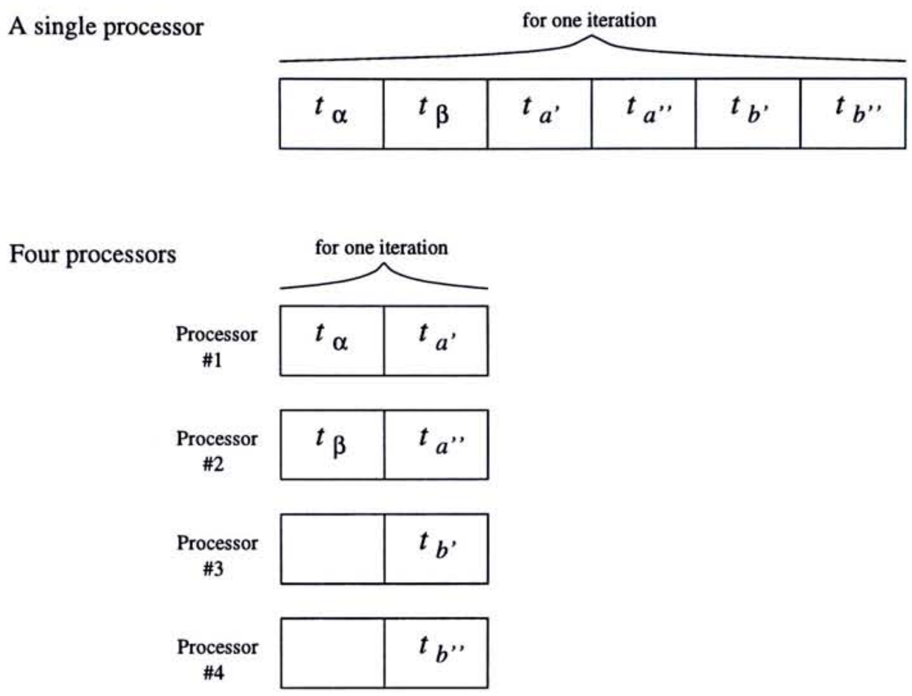


Figure 4.21: The difference of training time between single and multiple processors.

re-estimate steps. The re-estimation of a', a'', b' and b'' are computed concurrently. Indeed, forward and backward can also be computed concurrently. The re-estimated model parameters may be obtained from the forward and backward parameters. Equations (3.27), (3.28), (3.29) and (3.30) show the transition probabilities and the output probabilities which are obtained from forward and backward parameters.

4.6.3 Recognition

Again, similar to the matching processes of HMM, computational load are distributed among the processors by farming technique. The prestored values after training are model parameters of the 2dHMM in the vocabulary. They are distributed among workers to share the work load. For a 10-syllable vocabulary and a five-transputer network, each worker is assigned to be responsible for two words. Recognition begins with quantized feature vectors and quantized tone features of the sample speech signal being distributed to each transputer for computation of $P(O|\lambda)$. The transputers evaluate probabilities for their local stored words and choose the maximum probability. For junction transputer, besides calculating its local maximum probability, it receives probabilities of other words from the slave transputers and compares them. Therefore, the highest matching probability of a word among junction transputer and slave transputers is selected and passed to the master transputer. Finally, the master transputer picks the highest probability value and the word associated with it is the recognized word.

4.7 Training convergence in HMM and 2dHMM

The re-estimation algorithm based on ML is converged to local maximum. 10 English words from "zero" to "nine" were used for training of the HMM and 2dHMM. Convergence results with 30 iterations are plotted in Figures 4.22 to 4.26.

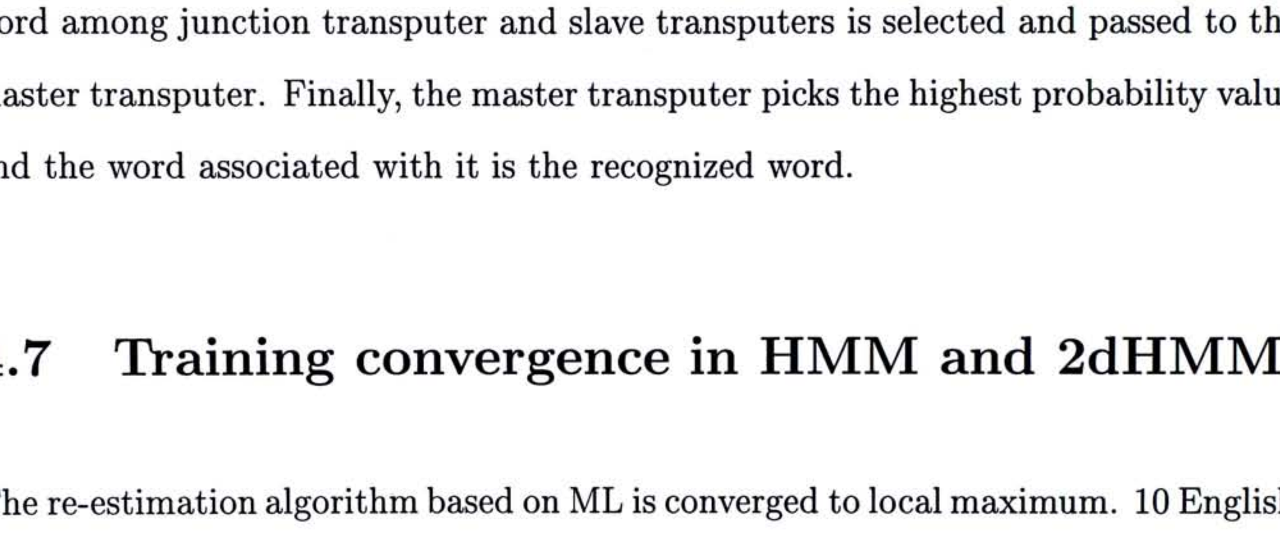


Figure 4.22: Probability convergence of words "one" (left) and "two" (right) during training.

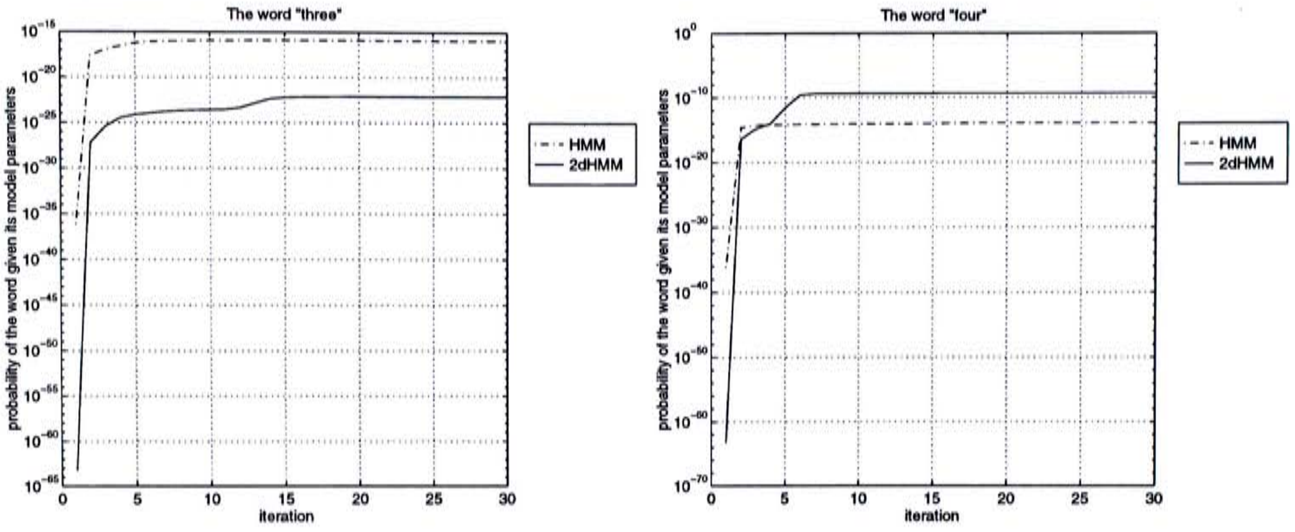


Figure 4.23: Probability convergence of words "three" (left) and "four" (right) during training.

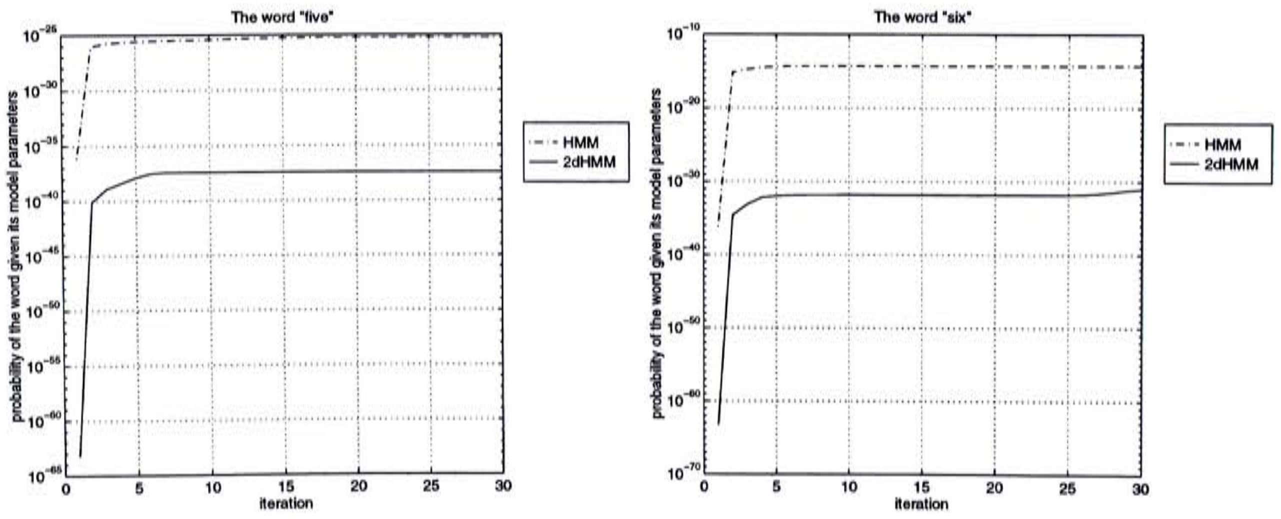


Figure 4.24: Probability convergence of words "five" (left) and "six" (right) during training.

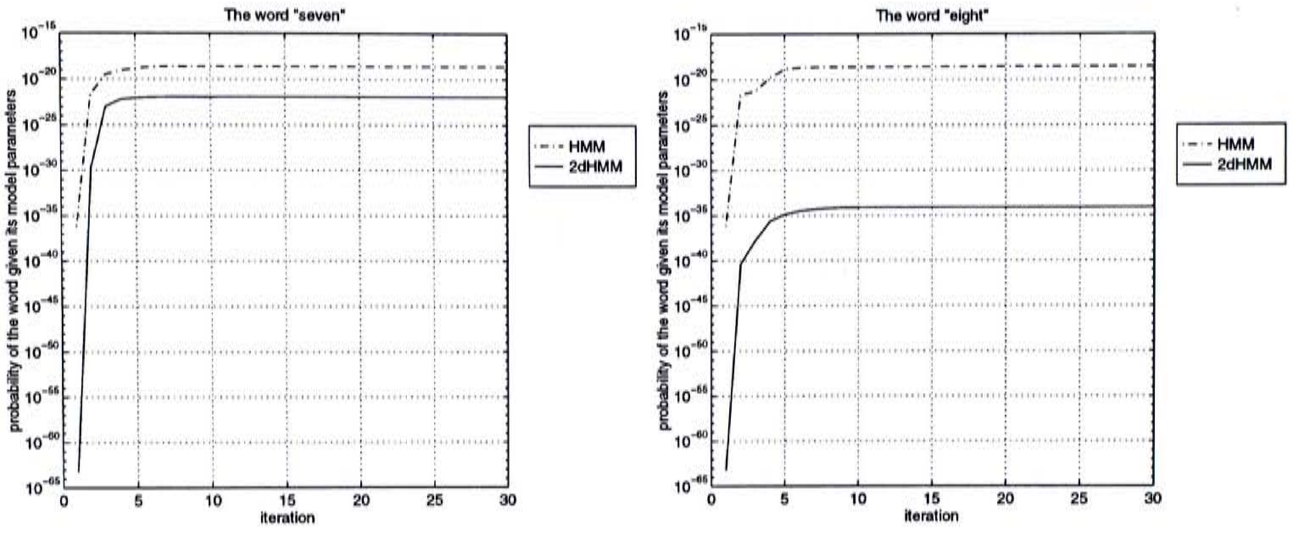


Figure 4.25: Probability convergence of words "seven" (left) and "eight" (right) during training.

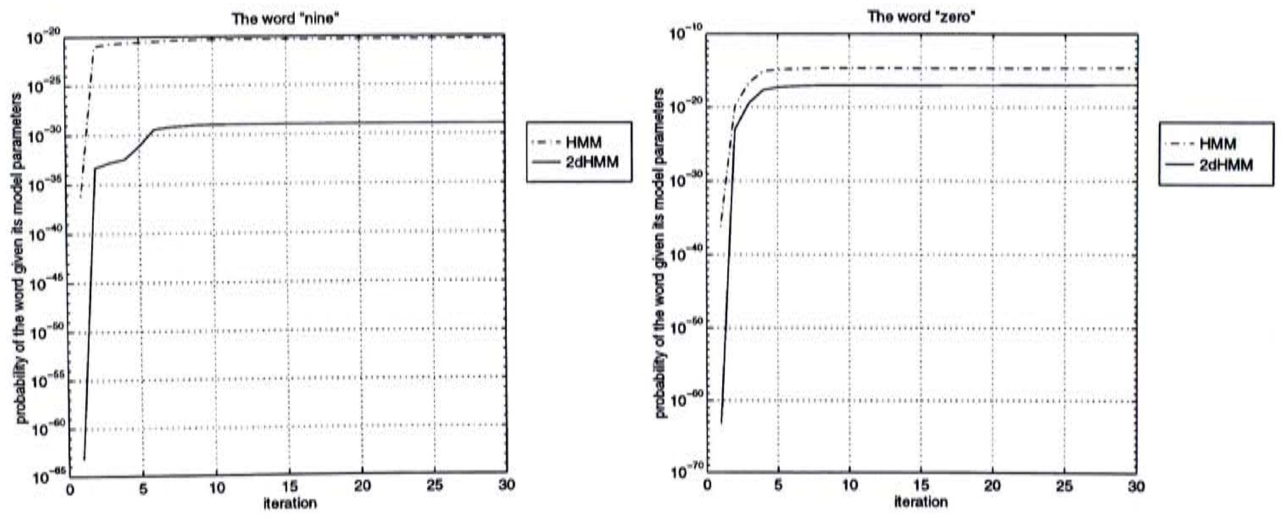


Figure 4.26: Probability convergence of words "nine" (left) and "zero" (right) during training.

From these figures, satisfactory results for both 2dHMM and HMM could be obtained after 10 iterations. The algorithm was able to converge to a stable point. Their convergence rates for training the same word are similar. The algorithm is only guaranteed to produce fixed-point solutions. Although in practice the lack of global optimality does not seem to cause serious problems in recognition performance [57]. The Baum-Welch reestimation algorithm as described in Sections 3.2.4 and 3.3.1 is a convenient, straightforwardly implementable solution to the ML estimation problem.

4.8 Discussions

Three speech recognition methods (DTW, HMM and 2dHMM) are described. The configurations of transputer networks for them are similar. For parts of feature extraction and matching, the computing load can be shared with the farming technique. Due to the more complicated algorithm of 2dHMM, it can be implemented concurrently.

Chapter 5

Experimental Results

In the Chapter, various experiments were designed to evaluate the performance of different speech recognizers, especially for the 2dHMM based recognizer. Both the recognition accuracy and speed of processing were the major concern. We first contrasted the performance between template matching method with DTW and statistical approachs (HMM and 2dHMM). The vocabulary consisted of 40 English words. The comparison was described in detail in Section 5.1. For the comparison between HMM and 2dHMM, a 20-English-word vocabulary was used and the experimental detail will be described in Section 5.2.1. In Section 5.2.2, we describe several experiments that have been designed to test the performance of the 2dHMM based recognizer for Chinese language, specifically for Cantonese. A small vocabulary of 10 Cantonese syllables were built to test the recognizer. Furthermore, 80 Cantonese syllables were collected as a relative large vocabulary to do another test that was designed to evaluate the performance of 2dHMM. It will be described in Section 5.3. To contrast the computing time involved in these recognition algorithms, the effective algorithm and multi-transputer based system were designed to set up the test. Experimental details could be found in Sections 5.4 and 5.5.

5.1 Comparison of DTW, HMM and 2dHMM

In the test, three isolated word speech recognizers using DTW, HMM and 2dHMM were implemented on a transputer based multi-processor system. A 40-English-word vocabulary was installed in each of the recognizers. Word samples were selected from an isolated word database published by Texas Instruments (TI 46-Word Speaker-Dependent Isolated Word Corpus). Each word is represented by a numeral and the vocabulary is shown in Table 5.1. Totally, 360 samples were used (9 samples for

Table 5.1: Word codes for 40 English words in the vocabulary.

Code	Word	Code	Word	Code	Word	Code	Word
1	ONE	11	A	21	K	31	U
2	TWO	12	B	22	L	32	V
3	THREE	13	C	23	M	33	W
4	FOUR	14	D	24	N	34	X
5	FIVE	15	E	25	O	35	Y
6	SIX	16	F	26	P	36	Z
7	SEVEN	17	G	27	Q	37	ENTER
8	EIGHT	18	H	28	R	38	ERASE
9	NINE	19	I	29	S	39	GO
10	ZERO	20	J	30	T	40	HELP

each word in the vocabulary). The feature parameter used is the spectral amplitude with 12 elements (as shown in Equation 2.9). For modeling 2dHMM, the zero crossing rate is used as an additional temporal feature for the auxiliary model which provides state sequence to weight the main model. For the recognizer that was using the template matching with DTW method, the reference templates were one of the spectral amplitudes from 9 samples of each word. The HMM based recognizer was trained with 9 samples of each word. The accuracy of each recognizer was tested by feeding the same 9 samples of each word in the 40-English-word vocabulary for recognition. The results are shown in Table 5.2 for the template matching using DTW recognizer, Table 5.3 for the HMM and Table 5.4 for the 2dHMM. We attempt to compare the recognition performance between template matching and statistical models. Accuracy results were 75%, 89% and 89% for DTW, HMM and 2dHMM respectively. It can be observed that the recognition accuracy for both HMM and 2dHMM based recognizers were better than template matching method.

Table 5.2: Isolated word recognition by template matching and DTW.

Word	Recognized as																																								Accuracy (%)
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	
1	5																																		4						55.6
2		9																																							100
3			9																																						100
4				9																																					100
5					9																																				100
6						9																																			100
7							9																																		100
8								1			6										2																				11.1
9									7																										2						77.8
10										9																															100
11											6										3																				66.7
12												6		2																		1									66.7
13													9																												100
14												2		4			1															2									44.4
15															3											4				2											33.3
16																6								1													1			1	66.7
17																	6			2										1											66.7
18																		9																							100
19					2														6																1						66.7
20																				7	2																				77.8
21																					9																				100
22																						9																			100
23																							4	5																	44.4
24																							3	6																	66.7
25																						3			4														2		44.4
26															1						1					2				5											22.3
27																											9														100
28																			1									8													88.9
29																													5					4							55.6
30																										1				8											88.9
31																											4				5										55.6
32			1											2																		6									66.7
33							2																										7								77.8
34																					1													8							88.9
35																																			9						100
36																																5				4					44.4
37																																					9				100
38																																		1				8			88.9
39										1												1																	7		77.8
40																						4																		5	55.6
																																									75% (overall)

Table 5.3: Isolated word recognition using HMM.

Word	Recognized as																																							Accuracy (%)
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	
1	9																																								100
2		9																																							100
3			9																																						100
4				9																																					100
5					9																																				100
6						8	1																																			88.9
7							9																																			100
8								3			2							2			2																						33.3
9									9																																	100
10										9																																100
11											7										2																						77.8
12												7		2																														77.8
13													9																													100
14												1		8																														88.9
15															8															1												88.9
16																8																				1						88.9
17												1					6																2										66.7
18																		9																								100
19																			9																							100
20																				9																						100
21											1										8																						88.9
22																						9																				100
23																							7	1							1										77.8
24																							2	7																		77.8
25				1																					8																	88.9
26															1		1									6				1										66.7
27																							3	1			5															55.6
28																												9														100
29																													9													100
30																														9												100
31																							2								7											77.8
32												1		2																			1	5										55.6
33																																	9									100
34																																		9								100
35																																			9							100
36																																				9						100
37																																					9					100
38																																						9				100
39										2																													7			77.8
40																																			1					8	88.9
																																									89% (overall)

Table 5.4: Isolated word recognition using 2dHMM.

Word	Recognized as																																								Accuracy (%)
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	
1	7									1																														1	77.8
2		7					1																				1														77.8
3			8																											1											88.9
4				9																																					100
5					9																																				100
6						9																																			100
7							9																																		100
8								2			3							2			1																	1			22.2
9									9																																100
10										9																															100
11											8									1																					88.9
12												7		2																											77.8
13													9																												100
14												1		7																		1									77.8
15															7											2															77.8
16																8																					1				88.9
17			1										1				7																								77.8
18																		9																							100
19					1														8																						88.9
20		2																		7	1																				77.8
21																					9																				100
22																						9																			100
23																							7	1							1										77.8
24																							1	8																	88.9
25																							1		8																88.9
26															1											8															88.9
27																											9														100
28																												9													100
29																													8									1			88.9
30													2																	7											77.8
31																															9										100
32																															1	8									88.9
33																																	9								100
34																																		9							100
35										1																									8						88.9
36																																				9					100
37																																					9				100
38																													2									7			77.8
39										2																							1						6		66.7
40																						1																		8	88.9
																																									89% (overall)

5.2 Comparison between HMM and 2dHMM

5.2.1 Recognition test on 20 English words

In this subsection, experiments for evaluating the performance in terms of both accuracy and unrecognized rates between 2dHMM and HMM under different sets of training samples are described. Word samples were selected from the TI 46-Word Speaker-Dependent Isolated Word Corpus. Training samples used for a word model in the experiments were 10, 20, 30, 40, 50, 60, 70 and 80 in experiments I to VIII respectively. Different combinations of 2-dimensional features for 2dHMM had been tried such as zero crossing rate, spectral amplitude, energy, LPC coefficients and cepstral coefficients, it was found that the spectral property of a speech signal was the best feature set to be used. From the result for accuracy performance in Section 5.1, the 2dHMM based recognizer with zero crossing rate as the other feature parameter for the auxiliary model which provides state sequence to weight the main model did not provide significant improvement in performance over that of HMM. Moreover, spectral analysis are the most important information to represent a speech signal. Therefore, two feature sets for two models (auxiliary and main) in 2dHMM were chosen to be the same. In other words, the 2dHMM based recognizer used one more observation sequence (feature set) than HMM. The observation sequence used for the auxiliary model in 2dHMM are the same for HMM based recognizer and the main model in 2dHMM. Cepstral coefficients, energy, delta cepstral coefficients and delta energy were used as the feature set for both HMM and 2dHMM. For recognition, 2203 samples (roughly 112 samples for each word) were fed into both the HMM and 2dHMM based recognizers in all experiments. The recognition accuracy and failure rate under different sets of training samples were summaried in Tables 5.5 and 5.6.

From the experimental results, it is found that the performance of accuracy for both HMM and 2dHMM are similar at each set of training. This is shown clearly

Table 5.5: Recognition accuracy under different numbers of training samples.

Number of samples used in training a word model	HMM	2dHMM
	Accuracy rate (%)	
10	29.10	25.74
20	59.96	61.28
30	72.54	72.31
40	80.75	81.39
50	87.38	87.88
60	89.38	89.92
70	90.51	91.15
80	91.19	91.74

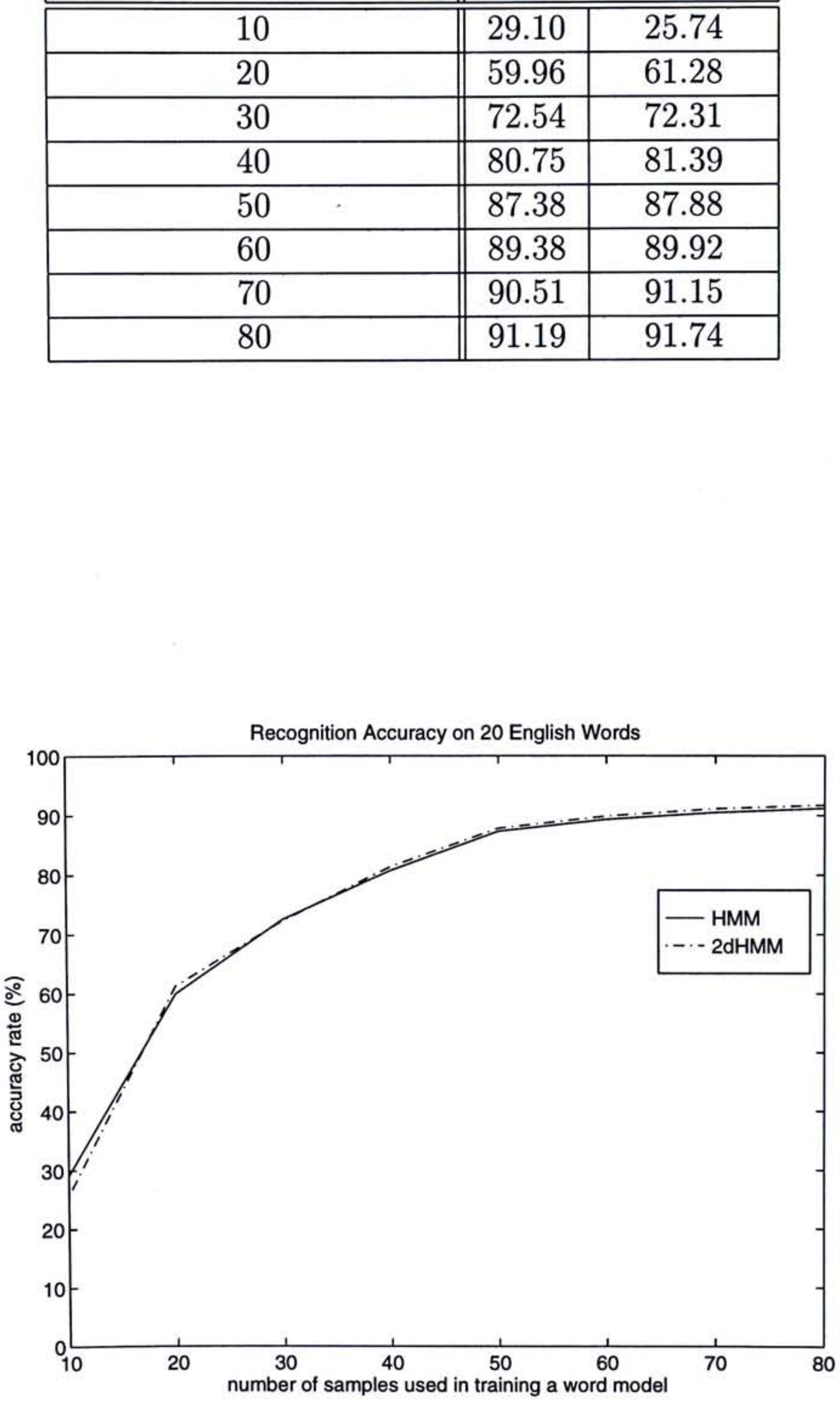


Figure 5.1: Recognition accuracies against different numbers of training samples.

Table 5.6: Unrecognized word under different numbers of training samples.

Number of samples used in training a word model	HMM	2dHMM
	Number of unrecognized words	
10	1294	1339
20	427	184
30	163	89
40	87	47
50	43	25
60	28	15
70	16	9
80	13	9

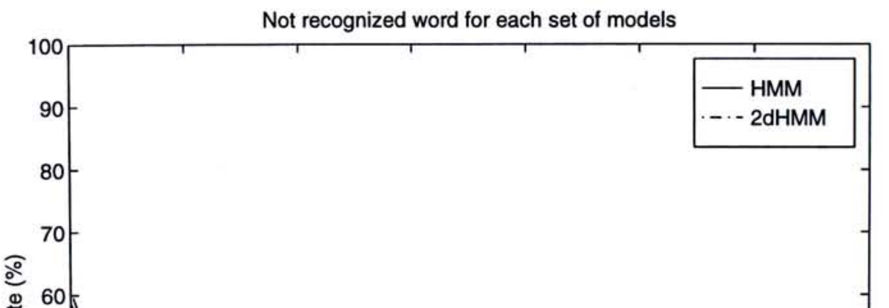


Figure 5.2: The graph of unrecognized word against different numbers of training samples.

in Figure 5.1. It is known that the model parameters of 2dHMM is more than that of HMM. In theory, more training samples are necessary in modeling 2dHMM than HMM since more parameters are needed to be trained in 2dHMM. It is expected that the overall unrecognition rate in 2dHMM should be higher than that of HMM. Referring to Figure 5.2, the main difference of unrecognition rate between HMM and 2dHMM existed when training samples were around 20. However, the accuracy rate between them was similar under different numbers of training samples.

For under training, the performance of 2dHMM should be worst than HMM. It is due to the fact that the model cannot be well trained using too few training samples. However, this situation can be improved by performing inter-training within those training samples. Extra (inter) training can be allowed and performed in 2dHMM, since more than one type of feature set (observation sequence) are used. One feature set can be obtained from a speech sample whilest the other feature set can be obtained from another sample with the same pronunciation. Moreover, the technique of DTW is necessary for two different speech samples with the same pronunciation to be processed in order to normalize their length. From Section 5.2.1, the accuracy rate for HMM is 29.10% with 10 training samples per word model. For 2dHMM, the accuracy rate is 25.74%. Other experiments were done by inter-training of 2-dimensional among 10 training samples. The results were summaried in Table 5.7. From the experiment, the accuracy result was improved from 25.74% to 33.41%, i.e. 29.8% of improvement.

Table 5.7: Inter-training performed in 2dHMM.

number of training samples	accuracy rate (%)
10 (original)	25.74
10 (original) + 70 (inter-training)	33.41

Conclusions

From the test it was observed that there were not much difference between 2dHMM and HMM when only spectral information with energy were used as feature parameters. However in the situation of under training, recognition accuracy could be improved by performing inter-training in the 2dHMM based recognizer. It was concluded that the other distinctive feature set was necessary to be used for the auxiliary model in 2dHMM to achieve a better recognition result. It is anticipated that tone can be treated as a temporal feature set to represent speech signals in tonal language like Cantonese. Being a tonal language, Cantonese is characterized by its constituent phonemes and distinctive tone. Both phonemes and tone are essential elements for Cantonese to be considered as features in modeling HMMs and 2dHMMs. Therefore, Cantonese is chosen to build a vocabulary for further investigation of the performance of an 2dHMM based recognizer. The experimental results were shown in Section 5.2.2.

5.2.2 Recognition test on 10 Cantonese syllables

Experiments were carried out to study the recognition performance when using a Cantonese vocabulary. Being a tonal language, the pronunciation of a Chinese character is characterized by its phonemes and distinctive tone. Tone is the variation of pitch which could be used as a kind of temporal feature for speech signal. Therefore, both phonemes and tone are essential to be considered as features in a Cantonese speech recognition system. This language, commonly used in Hong Kong and southeastern China, has nine lexical tones. Tones 1 to 6 are classified as non-entering tones and tones 7 to 9 as entering tones. The phonology divides each syllable into an Initial and a Final instead of smaller phonetic units [58, 59].

10 Cantonese syllables (shown in Table 5.8) were chosen to build the vocabulary.

In the vocabulary, each Cantonese syllable is similar to the other. Therefore, there are 5 pairs of Cantonese utterances. Each pair has the same phonemes with different tone. As shown in the Table, the tone of a syllable is represented by the last digit in its phonetic symbol [1]. Again, the feature set of spectral information was

Table 5.8: 10 Cantonese syllables in the vocabulary.

Syllable	Initial	Final	Tone	Chinese character (example)
ts^hin1	ts^h	in	1	千
ts^hin2	ts^h	in	2	前
kɐu3	k	ɐu	3	九
mai4	m	ai	4	買
man4	m	an	4	晚
kɐu6	k	ɐu	6	舊
mai6	m	ai	6	賣
man6	m	an	6	萬
jɐt7	j	ɐt	7	一
jɐt9	j	ɐt	9	日

used. Owing to the limited speech samples, experiments were repeated by choosing different training samples from all samples, while the remaining samples were used for recognition. The only difference between them was the choice of training samples. In the test, only two male speakers' samples were used. Totally 19 sets of speech samples (11 sets from speaker 1 and 8 sets from speaker 2) were employed for training and recognizing purposes. Each set of samples had 10 syllables in the vocabulary. In each experiment, 15 sets of samples were used to train a syllable model and the remaining 4 sets of samples were to be recognized. Different combinations of them were used to construct 4 experiments (A-15, B-15, C-15 and D-15). Names of all sample sets were shown in Table 5.9. M1 = male speaker 1, M2 = male speaker 2. For M1, there were 11 samples (from S1 to S11). For M2, there were 8 samples (from S1 to S8). There were three tests corresponding to each experiment.

[1]The symbols used for indicating pronunciation are those of the International Phonetic Association

Table 5.9: Sample listing for two male speakers.

	Name of different sample set
Speaker 1 (M1)	M1_S1, M1_S2, M1_S3, M1_S4, M1_S5, M1_S6, M1_S7, M1_S8, M1_S9, M1_S10, M1_S11
Speaker 2 (M2)	M2_S1, M2_S2, M2_S3, M2_S4, M2_S5, M2_S6, M2_S7, M2_S8

For Test 1 (T1) : the feature set of spectral information, (as in Equation 4.1)

$$\text{frame feature} = (C_1, C_2, C_3, ..., C_{10}, E, \delta C_1, \delta C_2, ..., \delta C_{10}, \delta E)$$

was used to model HMM.

For Test 2 (T2) : the feature set of spectral and tone information,

$$\text{frame feature} = (C_1, C_2, C_3, ..., C_{10}, E, \delta C_1, \delta C_2, ..., \delta C_{10}, \delta E, pitch)$$

was used to model HMM.

For Test 3 (T3) : The feature set of spectral information for the main model in 2dHMM and pitch information for the auxiliary model in 2dHMM,

$$\text{frame feature (main HMM)} = (C_1, C_2, C_3, ..., C_{10}, E, \delta C_1, \delta C_2, ..., \delta C_{10}, \delta E)$$

$$\text{frame feature (auxiliary HMM)} = pitch$$

were used to model 2dHMM.

- Experiment A-15

Samples used in training : M1_S1, M1_S2, M1_S3, M1_S4, M1_S5, M1_S6, M1_S7, M1_S8, M1_S9, M2_S1, M2_S2, M2_S3, M2_S4, M2_S5, M2_S6

Samples to be recognized : M1_S10, M1_S11, M2_S7, M2_S8

T1 : The result was shown in Table 5.10.

Table 5.10: Recognition result (A-15, T1) : 3-state HMM with spectral and energy information as a set of feature.

Syllable	Recognized as										Not recognized	Accuracy (%)
	ts^{h}in1	ts^{h}in2	kɐu3	mai4	man4	kɐu6	mai6	man6	jɐt7	jɐt9		
ts^{h}in1	2	2										50
ts^{h}in2	2	2										50
kɐu3			3			1						75
mai4							3				1	0
man4					3						1	75
kɐu6			3								1	0
mai6							4					100
man6					3						1	0
jɐt7									2	2		50
jɐt9										4		100
												50.00 (overall)

T2 : The result was shown in Table 5.11.

Table 5.11: Recognition result (A-15, T2) : 3-state HMM with spectral, energy and pitch information as a set of feature.

Syllable	Recognized as										Not recognized	Accuracy (%)
	ts^{h}in1	ts^{h}in2	kɐu3	mai4	man4	kɐu6	mai6	man6	jɐt7	jɐt9		
ts^{h}in1	2	2										50
ts^{h}in2	1	3										75
kɐu3			2			1					1	50
mai4				2			2					50
man4					2		1	1				50
kɐu6			1			2					1	50
mai6				1			3					75
man6					2		1	1				25
jɐt7									1	3		25
jɐt9							1			3		75
												52.50 (overall)

T3 : The result was shown in Table 5.12.

Table 5.12: Recognition result (A-15, T3) : 3×3-state 2dHMM with spectral and energy information as a set of feature and pitch information as the other set of feature.

	Recognized as										Not	Accuracy
Syllable	ts^{h}in1	ts^{h}in2	kɐu3	mai4	man4	kɐu6	mai6	man6	jɐt7	jɐt9	recognized	(%)
ts^{h}in1	1										3	25
ts^{h}in2		3									1	75
kɐu3			4									100
mai4				3							1	75
man4					2						2	50
kɐu6			2			1					1	25
mai6							4					100
man6								3			1	75
jɐt7									3		1	75
jɐt9										4		100
												70.00 (overall)

- Experiment B-15

Samples used in training : M1_S1, M1_S2, M1_S3, M1_S4, M1_S5, M1_S6, M1_S7, M1_S10, M1_S11, M2_S1, M2_S2, M2_S3, M2_S4, M2_S7, M2_S8

Samples to be recognized : M1_S8, M1_S9, M2_S5, M2_S6

T1 : The result was shown in Table 5.13.

Table 5.13: Recognition result (B-15, T1) : 3-state HMM with spectral and energy information as a set of feature.

	Recognized as										Not	Accuracy
Syllable	ts^{h}in1	ts^{h}in2	kɐu3	mai4	man4	kɐu6	mai6	man6	jɐt7	jɐt9	recognized	(%)
ts^{h}in1	3	1										75
ts^{h}in2	2	2										50
kɐu3			4									100
mai4				4								100
man4					3						1	75
kɐu6			2			2						50
mai6				3			1					25
man6					2			1			1	25
jɐt7									2	2		50
jɐt9									1	3		75
												62.50 (overall)

T2 : The result was shown in Table 5.14.

Table 5.14: Recognition result (B-15, T2) : 3-state HMM with spectral, energy and pitch information as a set of feature.

	Recognized as										Not	Accuracy
Syllable	ts^{h}in1	ts^{h}in2	kɐu3	mai4	man4	kɐu6	mai6	man6	jɐt7	jɐt9	recognized	(%)
ts^{h}in1	1	3										25
ts^{h}in2	1	3										75
kɐu3			3			1						75
mai4				2			2					50
man4				1	2				1			50
kɐu6			2			2						50
mai6				2			2					50
man6					1		1	2				50
jɐt7									2	2		50
jɐt9										4		100
												57.50 (overall)

T3 : The result was shown in Table 5.15.

Table 5.15: Recognition result (B-15, T3) : 3×3-state 2dHMM with spectral and energy information as a set of feature and pitch information as the other set of feature.

	Recognized as										Not	Accuracy
Syllable	ts^{h}in1	ts^{h}in2	kɐu3	mai4	man4	kɐu6	mai6	man6	jɐt7	jɐt9	recognized	(%)
ts^{h}in1	4											100
ts^{h}in2		4										100
kɐu3			4									100
mai4				4								100
man4					3						1	75
kɐu6						4						100
mai6				1			1				2	25
man6					3						1	0
jɐt7									4			100
jɐt9										3	1	75
												77.50 (overall)

- Experiment C-15

Samples used in training : M1_S1, M1_S2, M1_S3, M1_S4, M1_S5, M1_S8, M1_S9, M1_S10, M1_S11, M2_S1, M2_S2, M2_S5, M2_S6, M2_S7, M2_S8

Samples to be recognized : M1_S6, M1_S7, M2_S3, M2_S4

T1 : The result was shown in Table 5.16.

Table 5.16: Recognition result (C-15, T1) : 3-state HMM with spectral and energy information as a set of feature.

Syllable	Recognized as										Not recognized	Accuracy (%)
	ts^{h}in1	ts^{h}in2	kɐu3	mai4	man4	kɐu6	mai6	man6	jɐt7	jɐt9		
ts^{h}in1	4											100
ts^{h}in2	2	2										50
kɐu3			3								1	75
mai4				4								100
man4					3			1				75
kɐu6			2			1					1	25
mai6				2			2					50
man6					2			2				50
jɐt7									3	1		75
jɐt9									1	2	1	50
												65.00 (overall)

T2 : The result was shown in Table 5.17.

Table 5.17: Recognition result (C-15, T2) : 3-state HMM with spectral, energy and pitch information as a set of feature.

Syllable	Recognized as										Not recognized	Accuracy (%)
	ts^{h}in1	ts^{h}in2	kɐu3	mai4	man4	kɐu6	mai6	man6	jɐt7	jɐt9		
ts^{h}in1	2	2										50
ts^{h}in2	2	2										50
kɐu3			2			1				1		50
mai4				1			3					25
man4					1		1	2				25
kɐu6			1			2	1					50
mai6							4					100
man6					1		1	2				50
jɐt7			1			1			1	1		25
jɐt9									1	3		75
												50.00 (overall)

T3 : The result was shown in Table 5.18.

Table 5.18: Recognition result (C-15, T3) : 3×3-state 2dHMM with spectral and energy information as a set of feature and pitch information as the other set of feature.

	Recognized as										Not	Accuracy
Syllable	ts^{h}in1	ts^{h}in2	kɐu3	mai4	man4	kɐu6	mai6	man6	jɐt7	jɐt9	recognized	(%)
ts^{h}in1	4											100
ts^{h}in2		2									2	50
kɐu3			3								1	25
mai4				4								100
man4					3			1				75
kɐu6						3					1	75
mai6							4					100
man6					2			1			1	25
jɐt7									2		2	50
jɐt9										3	1	75
												67.50 (93.10) (overall)

- Experiment D-15

Samples used in training : M1_S1, M1_S2, M1_S3, M1_S6, M1_S7, M1_S8, M1_S9, M1_S10, M1_S11, M2_S3, M2_S4, M2_S5, M2_S6, M2_S7, M2_S8

Samples to be recognized : M1_S4, M1_S5, M2_S1, M2_S2

T1 : The result was shown in Table 5.19.

Table 5.19: Recognition result (D-15, T1) : 3-state HMM with spectral and energy information as a set of feature.

Syllable	Recognized as										Not recognized	Accuracy (%)
	ts^hin1	ts^hin2	kɐu3	mai4	man4	kɐu6	mai6	man6	jɐt7	jɐt9		
ts^hin1	3	1										75
ts^hin2	2	1									1	25
kɐu3						2					2	0
mai4				4								100
man4					3			1				75
kɐu6			2			1					1	25
mai6				2			2					50
man6					1			2			1	50
jɐt7									1	3		25
jɐt9									1	3		75
												50.00 (overall)

T2 : The result was shown in Table 5.20.

Table 5.20: Recognition result (D-15, T2) : 3-state HMM with spectral, energy and pitch information as a set of feature.

Syllable	Recognized as										Not recognized	Accuracy (%)
	ts^hin1	ts^hin2	kɐu3	mai4	man4	kɐu6	mai6	man6	jɐt7	jɐt9		
ts^hin1	2	1									1	50
ts^hin2	1	2									1	50
kɐu3			3							1		75
mai4							4					0
man4								4				0
kɐu6			1			3						75
mai6				1			3					75
man6					1		1	1			1	25
jɐt7									3	1		75
jɐt9										4		100
												52.50 (overall)

T3 : The result was shown in Table 5.21.

Table 5.21: Recognition result (D-15, T3) : 3×3-state 2dHMM with spectral and energy information as a set of feature and pitch information as the other set of feature.

Syllable	Recognized as										Not recognized	Accuracy (%)
	ts^{h}in1	ts^{h}in2	kɐu3	mai4	man4	kɐu6	mai6	man6	jɐt7	jɐt9		
ts^{h}in1	3										1	75
ts^{h}in2		2									2	50
kɐu3			2								2	50
mai4				4								100
man4					4							100
kɐu6			2								2	0
mai6				2			1				1	25
man6					1		1	2				50
jɐt7									4			100
jɐt9										1	3	25
												57.5 (overall)

In Experiments (A-15, B-15, C-15 and D-15), 10 Cantonese syllables were chosen to evaluate the accuracy performance between 2dHMM and HMM. Each Cantonese syllable was similar to another one in the vocabulary. The average accuracy for HMM was 51.25% while for 2dHMM was 65.63%. The unrecognized rates were 7.5% and 24.38% for HMM and 2dHMM respectively. From the results, it is apparent that the training samples were not enough. The models were not well trained since the unrecognized rate for 2dHMM was 24.38%. For HMM with pitch feature merged into feature vectors, the accuracy rate was improved to 53.13% (from 51.25%).

The other test was designed for 2dHMM, 4 extra samples from other 3 male speaks were involved in training a model. The overall accuracy result was 68.13% with unrecognized rate 20%. With more training samples, the performance of 2dHMM was a bit better than before, even the extra training samples were not obtained from the two male speakers. It proved that improvement was achieved with more training samples.

Conclusions

Accuracy rates in all experiments was not impressive, it is due to the limited collection of speech samples. However, our experiments have shown the important point that the performance of 2dHMM with two distinctive feature sets used in Cantonese vocabulary was better than HMM even these feature sets are used together as one for HMM. In the next experiment, the evaluation of performance would be concentrated on 2dHMM recognizer only. The relative large vocabulary of 80 Cantonese syllables was built to test the recognizer. Syllables in the speech corpus was designed to cover 9 lexical tones and most Initials and Finals in Cantonese (which includes 20 Initials and 36 Finals). The result was analyzed in Section 5.3.

5.3 Recognition test on 80 Cantonese syllables

In this test, 80 Cantonese syllables were selected to build the database. The experiment was designed to assess the accuracy performance of 2dHMM. The listing of these syllables is shown in Table 5.22. Again, the two distinctive feature sets used for 2dHMMs were

$$\text{frame feature (main HMM)} = (C_1, C_2, C_3, ..., C_{10}, E, \delta C_1, \delta C_2, ..., \delta C_{10}, \delta E),$$

and

$$\text{frame feature (auxiliary HMM)} = pitch.$$

The pre-processing state was done by the HTK Toolkit. Each syllable was trained by using 16 samples.

For recognition, 320 samples (4 samples for each syllable) were fed into the 2dHMM recognizer to evaluate the performance. The result was summarized in Table 5.23.

Referring to Figure 5.23, every syllable in the vocabulary has the chance to be

Table 5.22: 80 Cantonese syllables in the vocabulary.

Syllable	Syllable index	Chinese character (e.g.)	Syllable	Syllable index	Chinese character (e.g.)
pak8	1	百	jɐu6	41	又
pat8	2	八	jyn4	42	軟
pɔŋ6	3	磅	jyt9	43	月
pɐk7	4	北	k^{h}ei2	44	期
pun5	5	半	k^{h}ɐn4	45	近
tai6	6	大	lɛŋ5	46	靚
tim3	7	點	lœŋ4	47	兩
tin6	8	電	lʊk9	48	六
tɔ1	9	多	mai4	49	買
tɐi1	10	低	mai6	50	賣
tʊŋ1	11	東	man4	51	晚
tsɔ3	12	左	man6	52	萬
tsʊŋ1	13	中	mou4	53	母
fai5	14	快	mɐn1	54	蚊
fɔŋ1	15	方	mɐn2	55	文
fɐn1	16	分	nam2	56	南
fu6	17	父	nei4	57	你
kei3	18	幾	nin2	58	年
kɔŋ3	19	講	nϕy4	59	女
kɔ5	20	個	ŋɔ4	60	我
kou1	21	高	0m4	61	五
kɐm1	22	金	p^{h}ɛŋ2	62	平
kɐu3	23	九	sam1	63	三
kɐu6	24	舊	sei5	64	四
k^{w}an1	25	關	sɪŋ1	65	星
k^{w}ɐi5	26	季	sin1	66	先
haŋ2	27	行	siu3	67	小
ha6	28	下	sœŋ6	68	上
hœŋ1	29	香	sɐi1	69	西
hϕy5	30	去	sɐi5	70	細
hɔi1	31	開	sɐp9	71	十
hɔk9	32	學	syt8	72	說
hou3	33	好	t^{h}a1	73	他
hɐi6	34	系	ts^{h}in1	74	千
hɐu6	35	後	ts^{h}in2	75	前
jɛ6	36	夜	ts^{h}ϕn1	76	春
ji6	37	二	ts^{h}ϕt7	77	七
jɐp9	38	入	ts^{h}ɐt7	78	出
jɐt7	39	一	ts^{h}ɐu1	79	秋
jɐt9	40	日	wai6	80	壞

Table 5.23: Recognition accuracy of 80 Cantonese syllables.

Syllable	A-20		B-20		C-20		D-20		E-20		F-20		total incorrect rec'd
	wrong rec'd	not rec'd	wrong rec'd	not rec'd	wrong rec'd	not rec'd	wrong rec'd	not rec'd	wrong rec'd	not rec'd	wrong rec'd	not rec'd	
1			1		2		2		3		2		10
2	1				2		3		1		3		10
3	2				1		2		1		1	1	8
4	1		1				1		2		2		7
5	1						1				2	2	6
6	3	1	2					1	2			1	10
7			2	1	1								4
8	1		2				1		2		2		8
9							1		2		3		6
10	1			1	1			1			3		7
11		1			2			1				2	6
12					1	1					1	3	6
13		1									1		2
14							1			1	2	2	6
15					1						1		2
16			1		1	1		1	1		1		6
17							1		3		1	1	6
18			1		4				1			1	7
19	1			3						1	2		7
20	2						1		1		1		5
21	1					1	2		1		1		6
22			1		1		1				1		4
23										1		2	3
24							2		1	1	2		6
25				1		1		1				3	6
26	1	1	2						2	1	2		9
27			1					2	1				4
28	1		1		1		1		4		3		11
29										2	2		4
30	1	2								1	1	2	7
31		1		1						1	2	2	7
32			1		1		2		2		2		8
33			1					1			2		4
34	1	1							1		1	1	5
35			1				1						2
36			2		2		1				2	1	8
37	2		4		2		1		1		1		11
38			1				2		1			1	5
39								2			1		3
40	3		2		4		2		1	1	2	1	16
41		2						2		1		3	8
42	1						1				1		3
43					1		4		1		1		7
44			1	1	1		2	1	2				8
45		1	1			1				1			4
46	1					1				1	4		7
47		1		1				1			1	1	5
48				1	1	1	1		2		1		7
49		1						1				3	5
50		1				2	3		3	1	2	1	13
51	1	1		3		2					1		8
52	3		1		1		3		1	1	3	1	14
53	1	2	1		2		1				1		8
54	1		1		1				3		1		7
55	1						2		1		2		6
56	1	1					1		2	1	1		7
57	2						1			1	1		5
58		1	1					1				2	5
59	1			1	1					1		1	5
60	1				1					1	2		5
61	2		1				1				2		6
62			1					1			1		3
63	1				2	1		1			1	1	7
64	2		1							1	3		7
65								1				1	2
66	2		3		1			1	3				10
67		1		2				1		1		2	7
68	1							1	1			2	5
69		1		1		1			1		2		6
70	1			2			1				1	2	7
71							2			2	1		5
72	1						1		1		1		4
73					1		1		1		4		7
74	1				1		1		1		4		8
75		1					1	1		1		2	6
76	1				1		1		1		1		5
77	1		1				2				1		5
78	1		1		1				1		2		6
79		1		1		1		1				2	6
80								1				2	3
total incorrect rec'd	51	23	41	20	43	14	59	25	59	24	99	52	
	(74)		(61)		(57)		(84)		(83)		(151)		

recognized correctly in either set of experiments (A-20, B-20, ... , F-20). The total number of incorrect recognition for all 80 Cantonese syllables in the vocabulary varied from 2 to 16. On the other hand, the total number of incorrect recognition for different sets of experiments (A-20, B-20, C-20, ... F-20) varied from 61 to 151. The variation between them is quite substantial. The overall recognition rate, $= \frac{1920-510}{1920} \times 100\% = 73.44\%$, is not high. This is because training samples are not enough. However the result is satisfactory by using 16 samples to train 2dHMMs. Recall from Section 5.2.2 that the maximum accuracy rate is 65.63% for the 10-Cantonese-syllable vocabulary (5 pairs of utterances with the same syllable but different tone) under 15 training samples.

In Table 5.23, some unrecognized results are analyzed. The overall maximum unrecognized number is the syllable "40" (日). It is found that there is over 30% of mis-recognition between the syllable "40" and the syllable "38" (入). It is because they have the same tone and Initial. Again, over 30% of mis-recognition between the syllable "52" (萬) and the syllable "51" (晚). It is because they have the same Initial and Final. Some syllables with less the chance of confusion are shown in Table 5.24. If these 9 Cantonese syllables are used to build a vocabulary, the recognition rate would be much high.

Table 5.24: Syllables with less chance to be unrecognized.

Syllable index	example
13	中
15	方
23	九
35	後
39	一
42	軟
62	平
65	星
80	壞

To analyze the incorrect recognition corresponding to their 9 lexical tones, the results are shown in Figure 5.3. There is not too much different in their relative percentage of wrong recognition for 9 tones. It is concluded that the pitch detection algorithm used is suitable to extract tone features for Cantonese syllables, since incorrect rates are distributed all over 9 lexical tones.

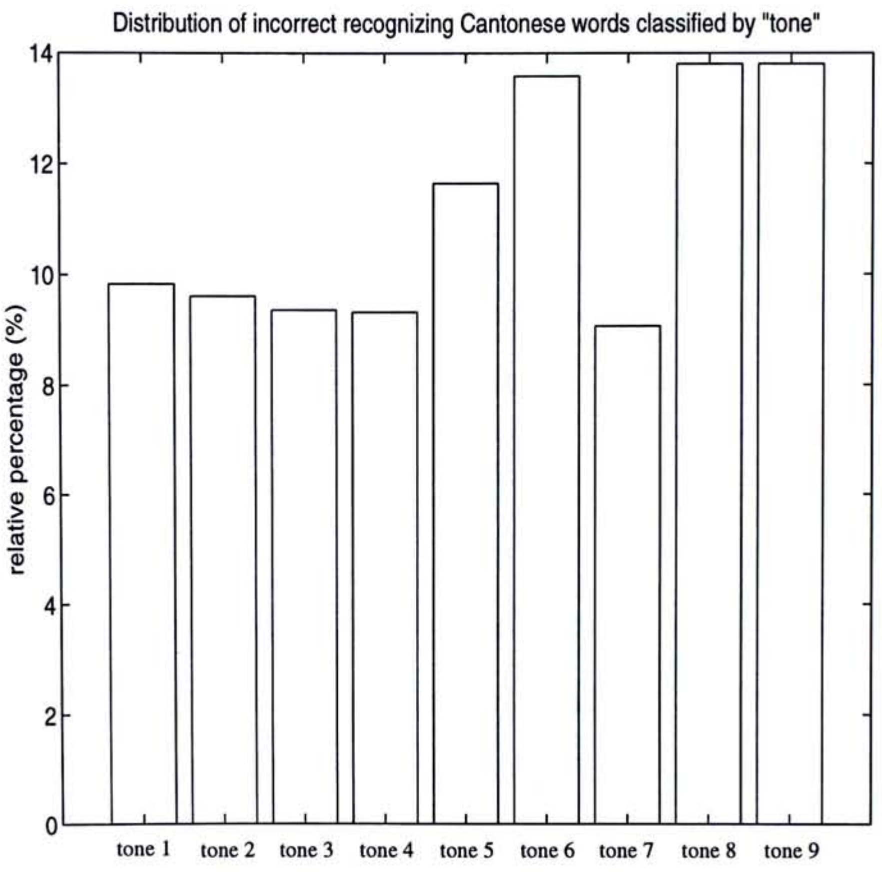


Figure 5.3: Distribution of incorrect recognizing Cantonese syllables corresponding to their 9 lexical tones.

5.4 Speed matching

The system recognition (matching) time for each of the recognition methods, template matching with DTW, HMM and 2dHMM are tabulated in Table 5.25. Results are based on a five-transputer system with a ten-syllable vocabulary. The state size of HMM is 3 and for 2dHMM is 3 in λ' and 3 in λ''. It can be seen that the

Table 5.25: Comparison of matching speed for DTW, HMM and 2dHMM.

Recognition method	Processing time (sec.) in					
	FFT features	VQ after FFT	ZCR features	quantization after ZCR	matching	overall
DTW	1.182	–	–	–	0.5468	1.7288
HMM	1.182	0.1	–	–	0.1041	1.3861
2dHMM	1.182	0.1	0.4	0.01	0.8591	2.5511

recognition speed of using 2dHMM is the slowest. This may be explained from Table 5.25 which shows that extra computation was required for extracting the zero crossing rate (ZCR) and quantizing the values for the 2dHMM method. Further, the 2dHMM topology which consists of two statistically related HMMs required extra computation for probabilities calculation. Recognition by template matching with DTW required a considerable amount of computation time for matching. This shows the computation intensiveness of the DTW algorithm. Therefore, the most efficient recognition method was HMM. Moreover, the pre-stored data as reference templates for HMM and 2dHMM are less than that of DTW.

5.5 Computational performance

The performance of the transputer system is commonly evaluated by the two factors described in Section 1.3.2. They are speedup and efficiency.

$$\text{Speedup} = \frac{\text{time required by a sequential task}}{\text{time required by parallel algorithm doing the same task}}$$

$$\text{Efficiency} = \frac{\text{speedup}}{\text{number of processor used}}$$

5.5.1 Training performance

In template matching, the feature set extracted can be used directly as a reference template. Therefore, there is no training processing for template matching type recognition. This is not the case for statistical models such as HMM and 2dHMM when they are used to model speech samples. Training processing is required to initialize the word models based on features extracted from speech samples. In Section 4.5.2, we mentioned that the re-estimation of model parameters, $\overline{a}_{ij}$ and $\overline{b}_j(k)$ of HMM can not be implemented effectively by using more than one processor. Referring to their updating Equations (3.19) and (3.20), the computation effort of $\overline{b}_j(k)$ relies mainly on $\overline{a}_{ij}$. It is because that once $\gamma_t(i,j)$ is calculated for $\overline{a}_{ij}$ it can be saved to calculate $\gamma_t(j)$ for $\overline{b}_j(k)$. Therefore, the mainly computation is for the re-estimation of a_{ij}. Comparing to $\overline{a}_{ij}$, the re-estimation time of $\overline{b}_j(k)$ is much shorter.

For the 2dHMM, as shown in Figure 4.21 in Section 4.6.2, clearly the re-estimation of model parameters must start after $\alpha()$ and $\beta()$ has been calculated. Therefore, the training algorithm can be decomposed into two steps. The first step is to calculate $\alpha()$ and $\beta()$. The next step is the re-estimation of model parameters (a', a'', b' and b'') from $\alpha()$ and $\beta()$. The first step may be decomposed into two concurrent processes and the second step may also be decomposed into four concurrent processes as shown in Figure 5.4. The efficiency of the decomposition may be estimated

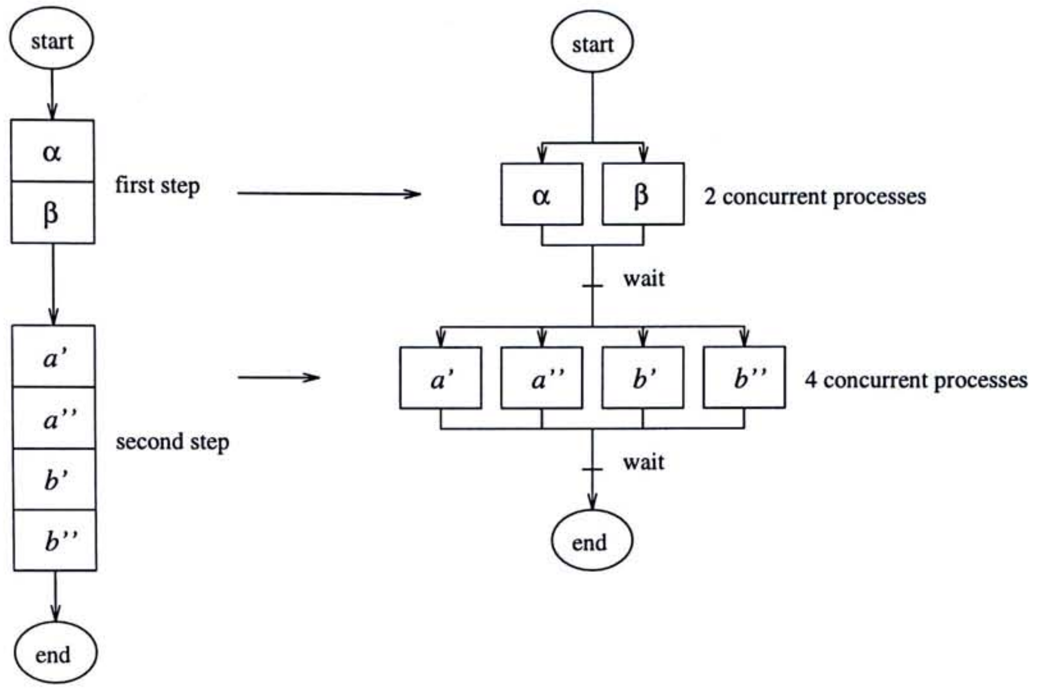


Figure 5.4: The training algorithm consists of 2 steps, each can be decomposed into a number of concurrent processes.

by using the diagram in Figure 5.5. Assuming the processing time for each sub block is t, hence, for a one transputer the processing time for one training iteration will be $6t$. For two transputers, as the processes may be executed concurrently, the processing time is $3t$. It can be seen that minimum processing time and best utilization of a processor can be achieved when the transputer number is four. Further increasing the number of transputer does not improve the speed nor the efficiency. The estimations based on Figure 5.5 were confirmed by testing the transputer system with different number of transputer and the measured efficiencies are shown in Table 5.26. Therefore, the overall speedup in re-estimation should be approximately equal to three by using four transputers.

5.5.2 Recognition performance

Recognition speed is an important factor for successful real time speech recognition application. Speed is less important for training, as this can be done off-line in most cases. All of the three recognition methods, template matching with DTW, HMM

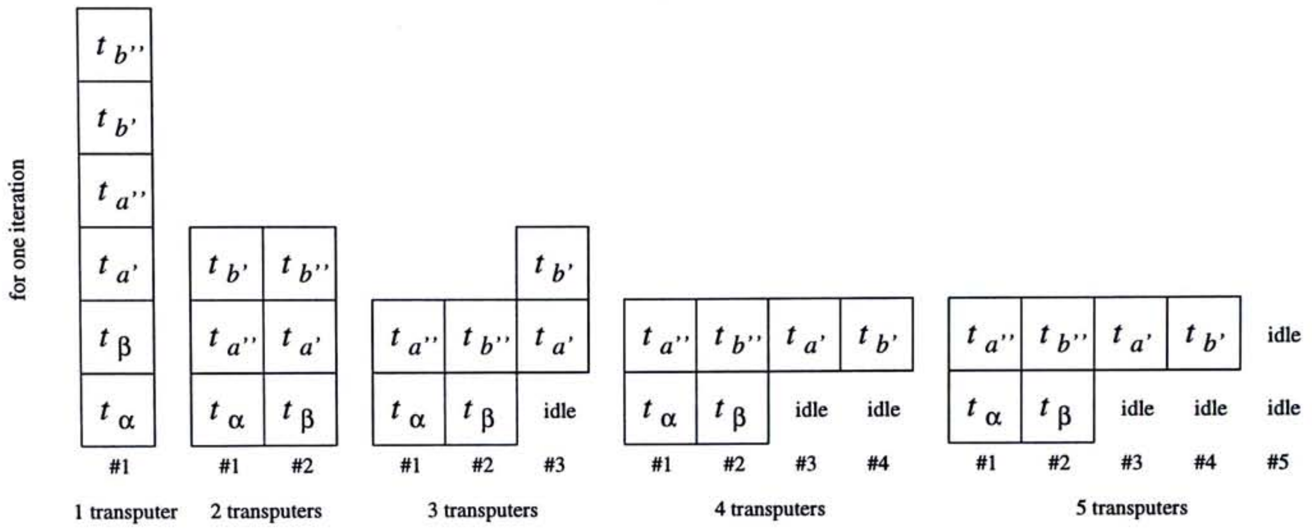


Figure 5.5: Activities of transputer in a one to five-transputer system for the processing of one iteration of re-estimation of model parameters for 2dHMM method.

Table 5.26: Training performance in re-estimating model parameters of 2dHMM.

	Speedup	Efficiency (%)
1 transputer	1.000	100
2 transputers	1.803	90.15
3 transputers	2.228	74.27
4 transputers	3.012	75.30
5 transputers	3.012	60.24

and 2dHMM are well suited to be implemented by farming technique.

The recognition procedure can be broadly divided into a two-step sequence, feature extraction, and matching. In the feature extraction step, since the features are extracted frame by frame by using the farming technique, the controller distributes the work evenly among the worker transputers. Figure 5.6 shows the architecture of the transputer network connected for farming. Transputer 'A' directly commu-

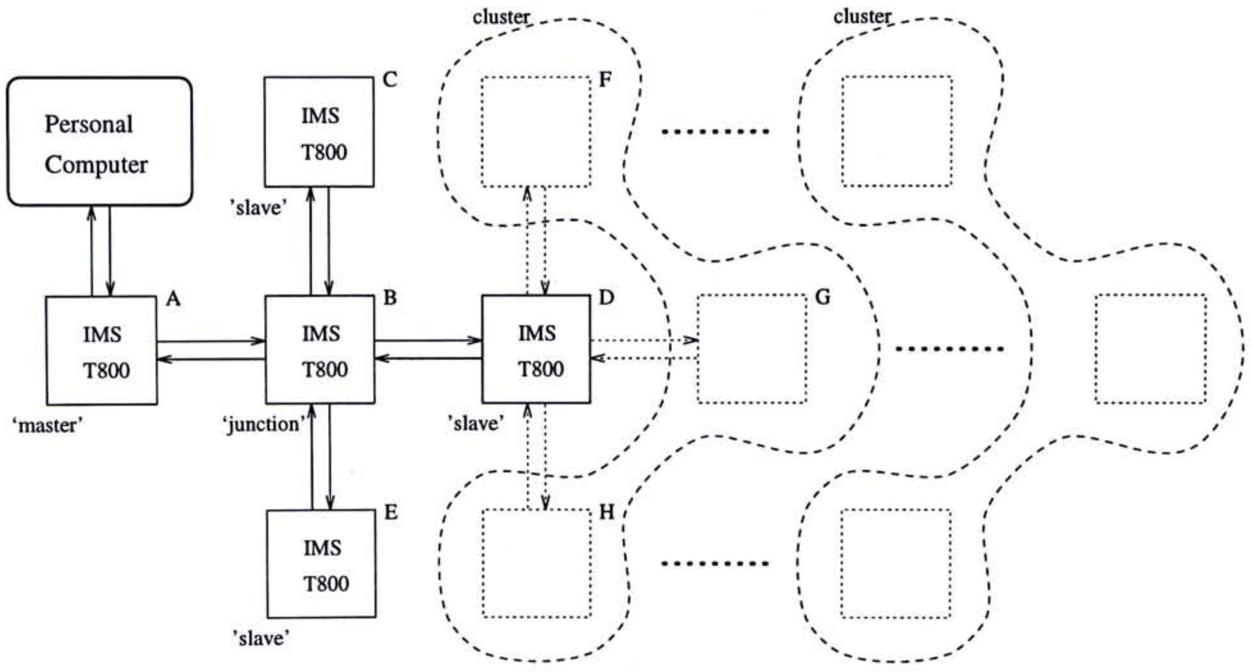


Figure 5.6: 5-transputer network can be expanded by adding more transputers.

nicates with the host computer which provides the user interface such as keyboard, monitor, etc. The two free links in the transputer 'A' may be used to interface to analog-to-digital transputer module. We call this transputer 'master'. The transputer 'B' is called 'junction'. The remain transputers (such as 'C', 'D' and 'E') are called 'slave'. The architecture may be expanded by adding cluster of 3 transputers. For example, transputer 'F', 'G', 'H' as a cluster may be added to the network without any modification to the program code. However, the network configuration definition will has to be adjusted to reflect the new topology. In addition, transputer 'D' will become a 'junction' and this will also has to be updated to allow the 'junction' program code to be downloaded to this transputer.

In the recognition step, this applies to all three recognition methods, speech samples are matched against all the word models or templates and probabilities or matching scores are ranked for the determination of the best match. Since independent matching calculation has to be done on all the works in the vocabulary, they can be implemented concurrently. By using farming technique, the word models or templates are distributed among the worker transputers and when a speech sample is given for recognition. The sample is copied to all worker transputers for matching. Each worker transputer would calculate independently the matching probabilities or matching scores and return them to the master transputer for the determination of the best match. This farming arrangement allows easy expansion of the size of the worker transputers, hence increasing the recognition speed without modification of the program code. Figure 5.7 depicts the 2dHMM matching step (it is similar for both HMM and template matching case). For 2dHMM based recognition, with

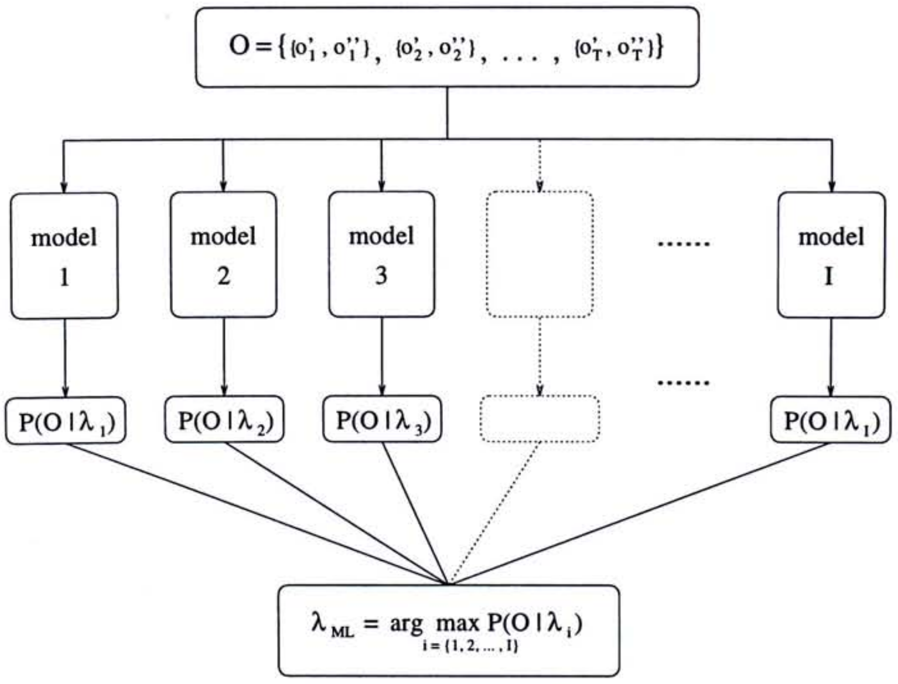

$O = \{\{o'_1, o''_1\}, \{o'_2, o''_2\}, \ldots, \{o'_T, o''_T\}\}$
model 1
model 2
model 3
......
model I
$P(O|\lambda_1)$
$P(O|\lambda_2)$
$P(O|\lambda_3)$
......
$P(O|\lambda_I)$
$\lambda_{ML} = \arg \max_{i=\{1,2,\ldots,I\}} P(O|\lambda_i)$

Figure 5.7: Matching processing for 2dHMM.

a 10 word vocabulary, the computational performance of the matching processing for one transputer to five transputers are shown in Table 5.27. The speedups and efficiencies are further plotted in Figure 5.8 and Figure 5.9. From these Figures, the

Table 5.27: Recognition performance in matching of 2dHMM.

Number of transputer(s)	Recognition time (sec)	Speedup	Efficiency (%)
1	4.2486	1.000	100
2	2.3771	1.787	89.37
3	1.6957	2.506	83.52
4	1.2679	3.351	83.77
5	0.8616	4.931	98.62

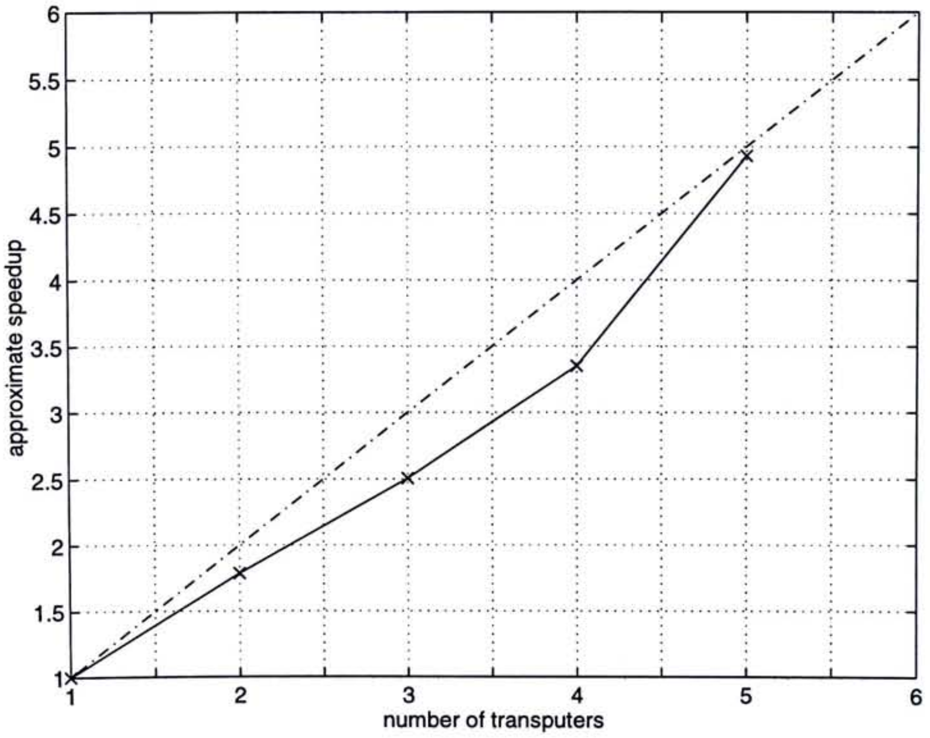


Figure 5.8: 2dHMM recognition speedup.

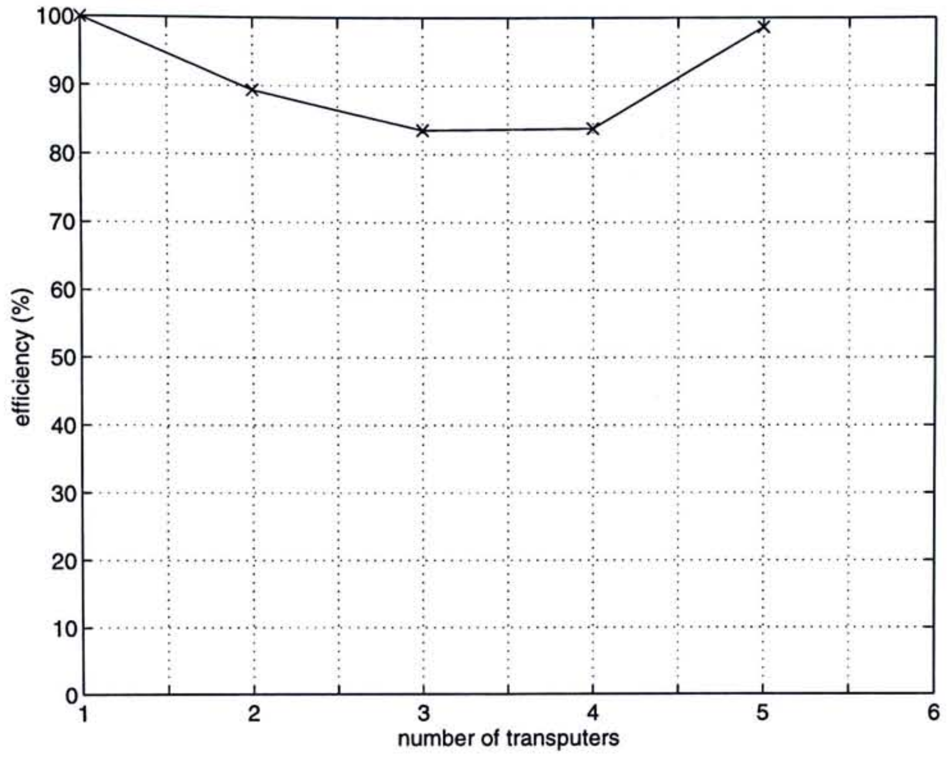


Figure 5.9: 2dHMM recognition efficiency.

highest efficiency of 98.62% is obtained when 5 transputers are used in the system. It is because 5 transputers can share the same number of word templates from the 10-word vocabulary.

However, speedup will not be increased linearly if too many transputers to be processed for a small vocabulary. It is due to the problem of communication overhead. Different number of transputer should be designed to suit for a particular vocabulary size to obtain the optimum performance. It is easy to modify the system since the transputer is scalable and the farming model may be expanded without modification on program codes for performance improvement.

Chapter 6

Discussions and Conclusions

Automatic speech recognition involves many practical problem such as end-point detection of a speech signal, and variations arise from different speakers or even from the same speaker. Many different approaches have been attempted to deal with the problem of speech recognition. Both recognition accuracy and speed are much concerned. In our studies, template matching and statistical models were investigated. The method of template matching with Dynamic Time Warping (DTW) can solve the problem of mis-alignment between two speech signals of the same word. For statistical modeling of Hidden Markov Model (HMM), the advantage is that it can be modeled by multiple observations in order to make the model more useful for practical applications. A modified model of HMM which takes into account of two distinctive feature sets was developed for accuracy improvement. Prototype systems were constructed for evaluation.

Comparing the results (in Section 5.1) obtained by template matching with DTW for the vocabulary of 40 English words where samples were extracted from TI-46 speech database, the recognizing methods of HMM and 2-dimensional HMM (2dHMM) performed better than template matching with DTW, even a small codebook size for the two discrete models was 32. One of the major differences between these two approaches is the vector quantization process which is necessary for the

HMM and 2dHMM based recognizers. It is expected that vector quantization error affected the recognition results. As the size of the codebook increases, the quantization error deceases but requiring more computation for quantization. Therefore, it is a trade off between codebook size and computing time.

Having proved HMM and 2dHMM were better methods than template matching with DTW for recognition, further tests were devised to focus the comparison between HMM and 2dHMM. The results showed that HMM and 2dHMM performed similarly with spectral, energy and their delta change information as a feature set, but the performance of them was quite dependent on the number of training.

Further tests were conducted by using tonal language, 10 Cantonese syllables were used to build the vocabulary. By the use of spectral and pitch information for Cantonese language as the two distinctive sets of features that the 2dHMM models on, the result on accuracy was improved significantly. In order to completely test the recognizer, a 80-Cantonese-syllable vocabulary which cover 9 lexical tones and most Initials and Finals in Cantonese was used.

The ultimate aim for an speech recognition system is to operate in real time with fast response. The complexity of the DTW, HMM and 2dHMM algorithms draws on heavy computing power to meet this requirement. One solution is to employ multi-processing. To address this issue, a transputer base multi-processing farming model was developed for the implementation of the algorithm. This model mimics the co-ordination of a controller and a group of workers. By decomposing the algorithm into communication processes and worker processes, the tests with the transputer system showed that the arrangement was highly efficient and yielded a linearly speedup with additional transputers (up to 5 transputers) for the 10 words vocabulary, particularly in the recognition process. As the training of speech vocabulary for recognition is mostly done off-line, computational efficiency for the training process is secondary.

Not only the multi-processing model is well suited for DTW, HMM and 2dHMM based recognition processing, another advantage is that the physical transputer network may be expanded in a cluster of 3 transputers. The advantage is without any modification of the program code. The only adjustment required is the network configuration file which reflects the new network configuration and the transputers' identities for program download. Although this implies that the transputer network may be expanded indefinitely for large vocabulary application, the communication overhead between the controller and the worker at the end of the array will become too large. It means the performance will be degraded if too many processors. Therefore, there should be an optimum number of processors for a particular recognition system. However, the algorithm can be designed to adapt different situations. One solution to this shortcoming is by arranging the sub-vocabulary words at each transputer in a fashion such that frequent occurring words are near the controller and less frequent words are near the end of worker array. By choosing a suitable threshold value, the controller may be made to watch for a matching result that exceeds the threshold for a first guess instead of sorting through the complete set of matching results.

In conclusion, the use of statistical modeling of HMM in isolated word speech recognition was found to perform better than template matching with DTW. The HMM was improved by the development of 2dHMM which models speech signals on two distinctive sets of features. Based on transputer multi-processing, a farming model was development for the implementation of different recognition algorithms. The model may be expanded without modification on program codes for performance improvement when it is necessary. It is expected the 2dHMM together with the farming model provides a platform for further exploration on feature type variations or even a Multi-dimensional HMM (MdHMM) which models on M sets of features.

Bibliography

[1] J. A. Howard. Flight testing of the afti/f-16 voice interactive avionics system. In *Proceedings of Military Speech Tech 1987*, pages 76 – 82. Arlingtion, Va., Media Dimensions, 1987.

[2] F. A. Rosenhoover, J. S. Eckel, F. A. Gorg, and S. W. Rabeler. Afti/f-16 voice interactive avionics evaluation. In *Proceedings of the National Aerospace and Electronics Conference (NAECON'87)*. IEEE, 1987.

[3] J. T. Williamson. Flight test results of the afti/f-16 voice interactive avionics program. In *Proceedings of the American Voice I/O Society (AVIOS) 87 Voice I/O Systems Applications Conference*, pages 335 – 345. Alexandria, Va., 1987.

[4] M. Mazo, F. J. Rodriguez, J. L. Lazaro, J. Urena, J. C. Garcia, E. Santiso, and P. A. Revenga. Electronic control of a wheelchair guided by voice commands. *Control Engineering Practice*, 3(5):665 – 674, May 1995.

[5] S. Dobler, D. Geller, R. Haeb-Umbach, P. Meyer, H. Ney, and H.W. Ruehl. Design and use of speech recognition algorithms for a mobile radio telephone. *Speech Communication*, 12:221 – 229, 1993.

[6] Joseph W. Picone. Signal modeling techniques in speech recognition. *Proceedings of the IEEE*, 81(9):1215 – 1247, September 1993.

[7] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1), February 1978.

[8] C. M. Myers, L. R. Rabiner, and A. E. Rosenberg. Performance tradeoffs in dynamic time warping algorithm for isolated word recognition. *IEEE Transactions on Acoustic, Speech and Signal Processing*, 28(6), December 1980.

[9] L. R. Rabiner. Note on some factors affecting performance of dynamic time warping algorithms for isolated word recognition. *The Bell System Technical Journal*, 1(3), March 1982.

[10] N. G. Stainhaouer and G. Carayannis. New Parallel Implementations for DTW Algorithms. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(4):705 – 711, April 1990.

[11] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4 – 16, January 1986.

[12] X. D. Huang, Y. Ariki, and M. A. Jack. *Hidden Markov Models for Speech Recognition.* Edinburgh University Press, 1990.

[13] B. H. Juang and L. R. Rabiner. Hidden Markov Models for speech recognition. *Technometrics*, 33:251 – 272, August 1991.

[14] M. J. Creaney and R. N. Gorgui-Naguib. A scaly artificial neural network architecture for isolated word recognition using the British Telecommunications English alphabet database. In *1993 IEEE International Conference on Neural Networks*, volume 3, pages 1644 –1649. New York: IEEE, 1993.

[15] J. C. Principe and J. W. Tracey. Isolated-word speech recognition using the focused gamma neural network. *Journal of Artificial Neural Networks*, 1(4):481 – 499, 1994.

[16] B. S. Atal. Speech technology in 2001: New research directions. In *Voice communication between humans and machines*, pages 467 – 481. National Academy Press, 1994.

[17] Pietro Laface and Renato De Mori, editors. *Speech Recognition and Understanding.* Springer-Verlag, 1990.

[18] R. Nakatsu and Y. Suzuki. What does voice-processing technology support today? In *Voice Communication Between Humans and Machines*, pages 390 – 421. National Academy Press, 1994.

[19] D. M. Weber. Parallel implementation of time delay neural networks for phoneme recognition. *1993 IEEE International Conference on Neural Networks*, 3:1583 – 1587, 1993.

[20] Jianmin Li and Ditang Fang. Parallel distributed binary mapping models for speech recognition. *1994 IEEE International Conference on Acoustics, Speech and Signal Processing*, 1:I/405 – 408, 1994.

[21] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing.* McGraw-Hill, 1985.

[22] R. S. Cok. *Parallel Programs for the Transputer.* Prentice Hall, 1991.

[23] G. R. Desrochers. *Principles of Parallel and Multi-processing.* McGraw-Hill, 1988.

[24] L. R. Rabiner and B. H. Juang. *Fundamentals of Speech Recognition.* Prentice Hall International Editions, 1993.

[25] L. R. Rabiner and M. R. Sambur. An algorithm for determining the endpoints of isolated utterances. *The Bell System Technical Journal*, 54(2):297 – 315, February 1975.

[26] E. O. Brigham. *The Fast Fourier Transform and Its Applications.* Prentice Hall: Englewood Cliffs, N. J., 1988.

[27] G. Hopper and R. Adhami. An FFT-based speech recognition system. *Journal of the Franklin Institute*, 329(3):555 – 562, 1992.

[28] N. Levinson. The weiner rms error criterion in filter design and prediction. *Journal of Mathematical Physics*, 25:261 – 278, 1947.

[29] J. Durbin. The fitting of time series models. *Review of the Institute for International Statistics*, 28:233 – 243, 1960.

[30] Y. K. Lau and C. K. Chan. Speech recognition based on zero crossing rate and energy. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 33(1):320 – 323, 1985.

[31] Y. K. Lau. Digital Chinese Speech Recognition System: final report on the research project. Technical report, Dept. of Electronic Engineering, Hong Kong Polytechnic, 1986.

[32] L. R. Rabiner. On the use of autocorrelation analysis for pitch detection. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):24 – 33, February 1977.

[33] T. Lee. *Automatic recognition of isolated Cantonese syllables using Neural Networks*. PhD thesis, Department of Electronic Engineering, The Chinese University of Hong Kong, May 1996.

[34] M. M. Sondhi. New methods of pitch extraction. *IEEE Transactions on Audio and Electroacoustics*, 16(2):262 – 266, June 1968.

[35] Y. H. Cheng. An efficient tone classifier for speech recognition of cantonese. Master's thesis, Department of Electronic Engineering, The Chinese University of Hong Kong, 1991.

[36] L. R. Bahl, P. F. Brown, P. V. de Souza, and R. L. Mercer. Speech recognition with continuous parameter Hidden Markov Models. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 40 – 43. New York: IEEE, 1988.

[37] A. B. Poritz and A. G. Richter. Hidden Markov Models in isolated word recognition. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 705 – 708. New York: IEEE, 1986.

[38] L. R. Rabiner, B. H. Juang, S. E. Levinson, and M. M. Sondhi. Recognition of isolated digits using Hidden Markov Models with continuous mixture densities. *AT&T Technical Journal*, 64:1211 – 1222, 1986.

[39] X. Huang and M. A. Jack. Unified techniques for vector quantization and Hidden Markov Models using semicontinuous models. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 639 – 642. New York: IEEE, 1989.

[40] S. E. Levinson. Continuously variable duration Hidden Markov Models for automatic speech recognition. *Computer Speech and Language*, 1:29 – 45, 1986.

[41] L. R. Rabiner and B. Gold. *Theory and Application of Digital Signal Processing.* Prentice Hall, 1975.

[42] T. Parsons. *Voice and Speech Processing.* McGraw-Hill, 1987.

[43] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28:84 – 95, January 1980.

[44] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 37:1554 – 1563, 1966.

[45] L. E. Baum and J. A. Egon. An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to a model for ecology. *Bulletin of the American Meteorological Society*, 73:360 – 363, 1967.

[46] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41:164 – 171, 1970.

[47] G. D. Forney. The viterbi algorithm. In *Proceedings of the IEEE*, volume 61, pages 268 – 278, March 1973.

[48] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transcations on Information Theory*, 13:260 – 269, April 1967.

[49] S. E. Levinson, L.R. Rabiner, and M.M. Sondhi. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *The Bell System Technical Journal*, 62(4):1035 – 1074, April 1983.

[50] F. Brugnara, R. De Mori, D. Giuliani, and M. Omologo. A parallel HMM approach to speech recognition. In *Proceedings of the Eurospeech*, pages 1103 – 1106, 1991.

[51] L. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257 – 286, 1989.

[52] J. Ffynlo Craine and Graham R. Martin. *Micro-computers in Engineering and Science.* Addison-Wesley Publishing Company, 1985.

[53] I. Grabam and T. King. *The Transputer Handbook.* Prentice Hall, 1990.

[54] INMOS Limited. *ANSI C Toolset User Manual*, August 1990.

[55] Jeremy Hinton and Alan Pinder. *Transputer Hardware and System Design.* Prentice Hall International, 1993.

[56] ENTROPIC. *HTK - Hidden Markov Model Toolkit.*

[57] D. B. Paul. Training of HMM recognizers by simulated annealing. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 13 – 16. New York: IEEE, 1985.

[58] S. Matthews and V. Yip. *Cantonese: A Comprehensive Grammar.* Routledge Press, London, 1994.

[59] Y. R. Chao. *A Grammar of Spoken Chinese, translated by Ting Pang Hsin.* Chinese University Press, 1980.

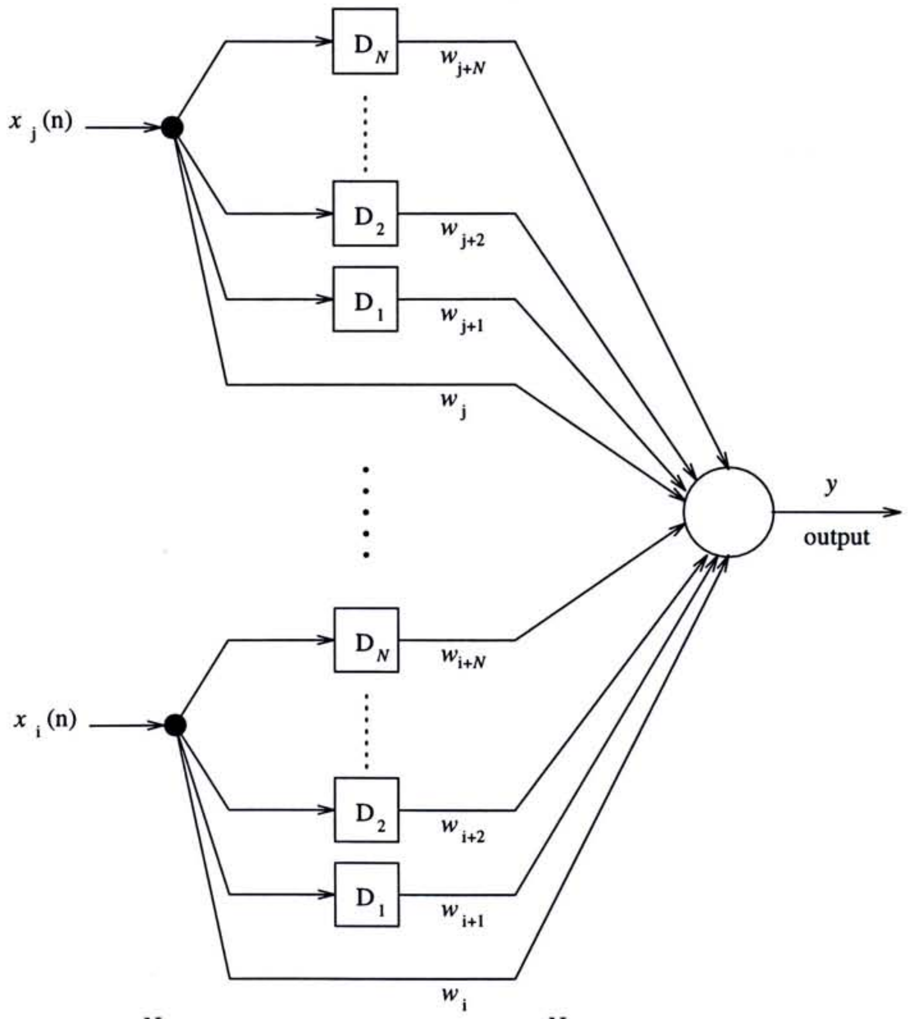
[60] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. Phoneme recognition using Time Delay Neural Networks. *IEEE Transactions on Acoustics, Speech and Signal Processing,* 37:328 – 339, 1989.

[61] INMOS Limited. *The transputer instruction set – a compiler writers' guide,* 1987.

Appendix A

An ANN Model for Speech Recognition

Since conventional ANNs (like multilayer perceptrons) are used to deal with static patterns, it is not suitable for the use of speech signal which is inherently dynamic in nature. Hence, some modifications are necessary to conventional ANNs. The simple one of these modifications is to use the Time Delay Neural Network (TDNN) computational element (by *A. Waibel et al.* [60]) which is shown in Figure A.1. This structure extends the input to each computational element to include N frames of a speech signal [24].



$$y = f(\sum_{p=0}^{N} w_{i+p}x_i(n-p) + \ \dots\ + \sum_{q=0}^{N} w_{j+q}x_j(n-q) - \phi)$$

Figure A.1: Time Delay Neural Network (TDNN) computational element.

Appendix B

A Speech Signal Represented in Fequency Domain (Spectrogram)

A way of characterizing the speech signal and representing the information associated with the sounds is via a spectral representation. Figures B.1, B.2, B.3, B.4 and B.5 show speech signals of English words and their spectrograms. The grey scale shows the amplitude of spectral energy. The points of high spectral energy (more darkness) in the spectrogram corresponds to the estimated formant frequencies.

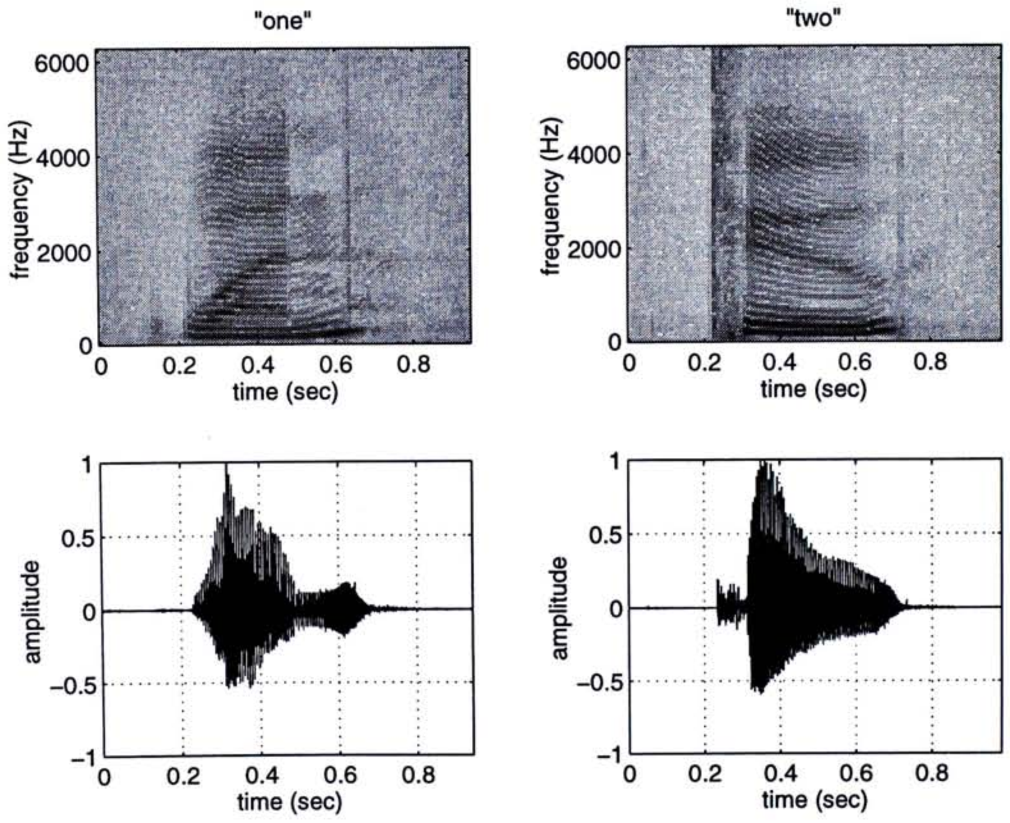


Figure B.1: Spectrograms of English words "one" (left) and "two" (right). Their corresponding time-domain signals are plotted below spectrograms.

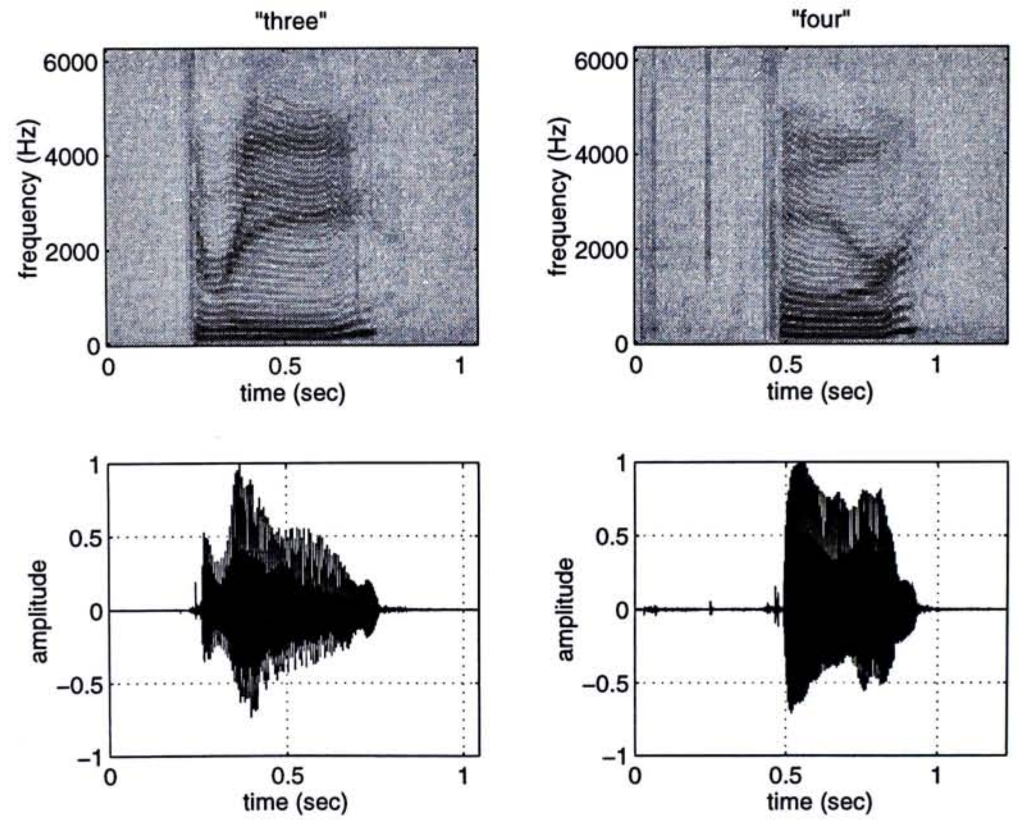


Figure B.2: Spectrograms of English words "three" (left) and "four" (right). Their corresponding time-domain signals are plotted below spectrograms.

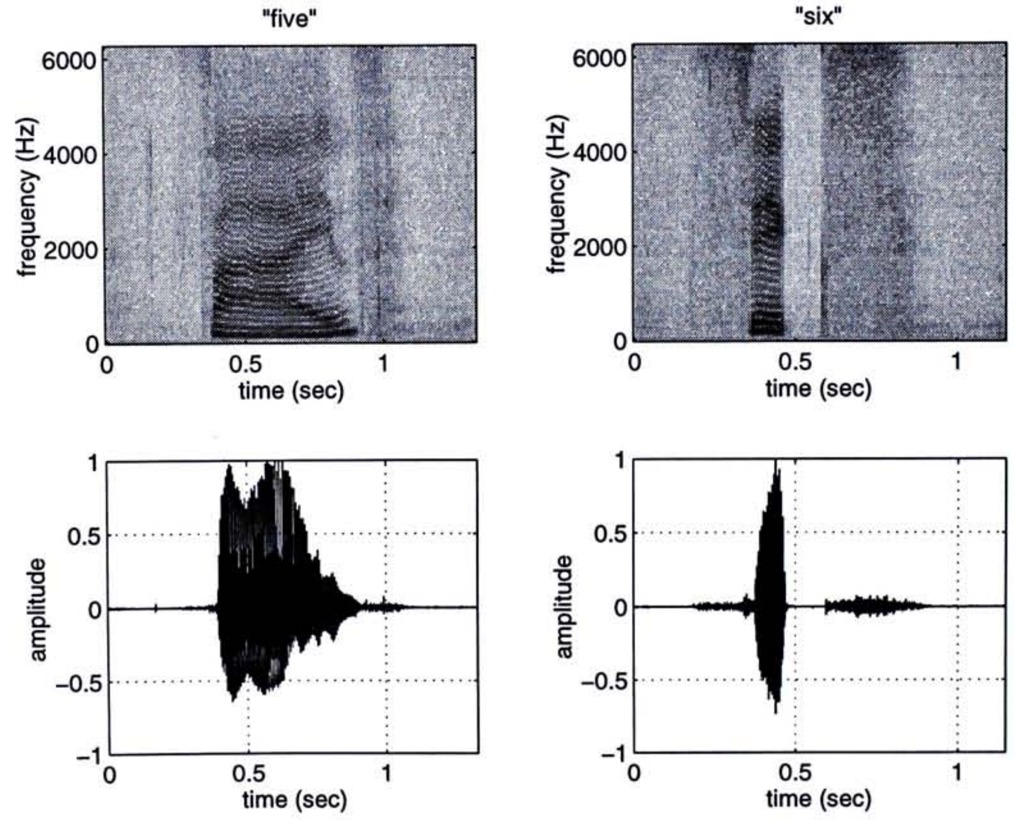


Figure B.3: Spectrograms of English words "five" (left) and "six" (right). Their corresponding time-domain signals are plotted below spectrograms.

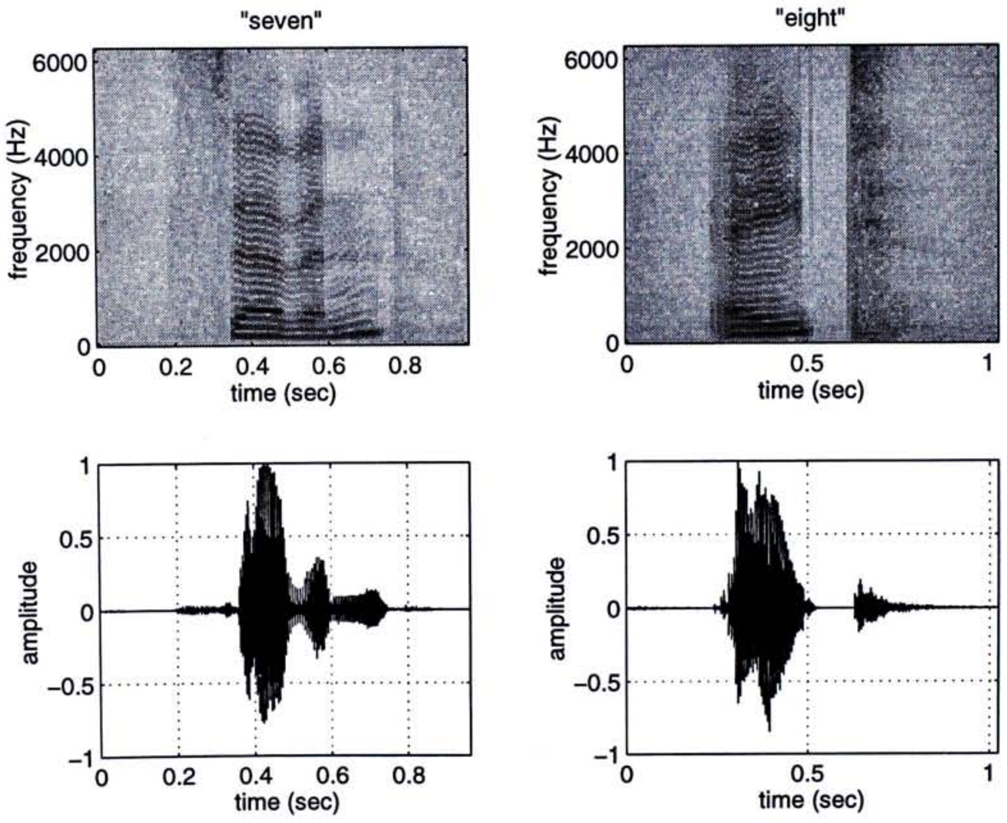


Figure B.4: Spectrograms of English words "seven" (left) and "eight" (right). Their corresponding time-domain signals are plotted below spectrograms.

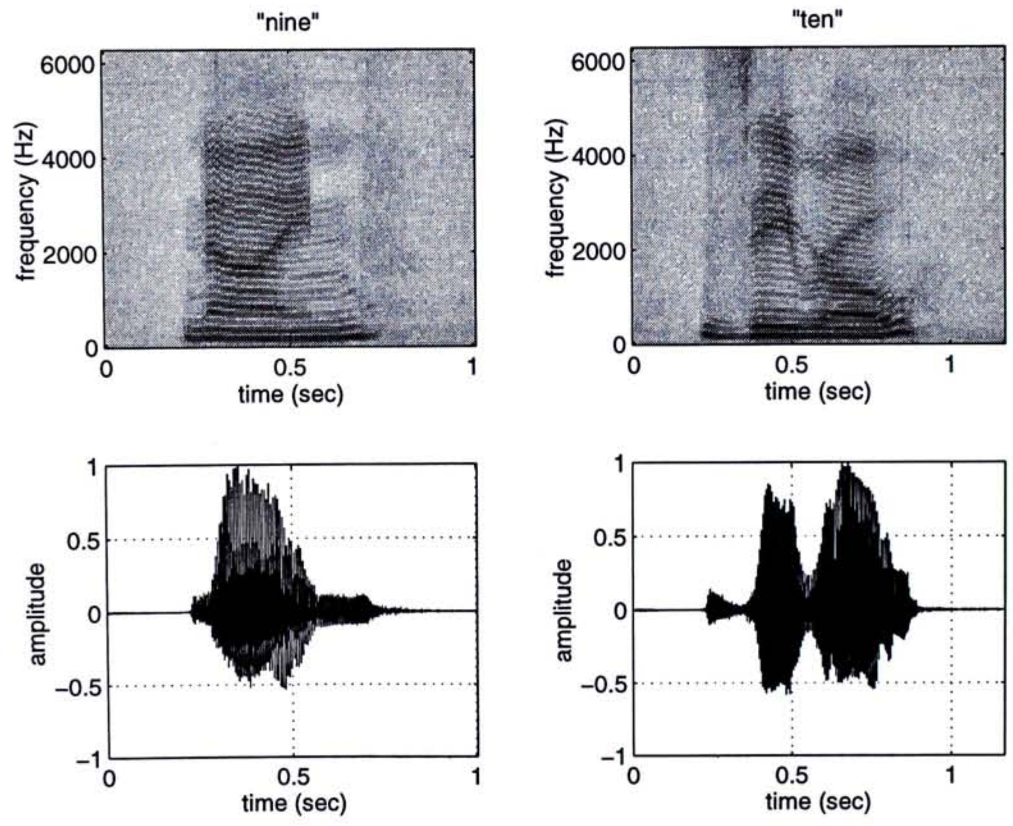


Figure B.5: Spectrograms of English words "nine" (left) and "ten" (right). Their corresponding time-domain signals are plotted below spectrograms.

Appendix C

Dynamic Programming

From the Equation (3.7) in Section 3.1, it can be solved by the technique of dynamic programming without exhaustively searching all possibilities of the warping function c_k. Only the numerator is considered since the denominator is a normalizing factor. Let $g(i_k, j_k)$ be the cumulative function that storing the cumulative distance from the beginning point $(1,1)$ to the point (i_k, j_k) along the optimum path.

$$g(c_k) = g(i_k, j_k) = \min_{c_1, c_2, \ldots, c_k} (\sum_{p=1}^{k} d(c_p) W(p)) \tag{C.1}$$

where $d(c_k)$ is distance function ready for a valid path from c_k to c_{k+1}. For initialization, $g(1,1)$ is equal to $d(1,1)$ which is the beginning point for all possible paths. For using Type II Constraint (local path constraints) [8], three possible paths can reach the point (i, j). The cumulative distance is

$$g(i,j) = \min \left\{ \begin{array}{l} g(i-1, j) \\ g(i-1, j-1) \\ g(i, j-1) \end{array} \right\} \tag{C.2}$$

Hence the total normalized distance is

$$\hat{D} = \frac{g(N, M)}{N} \tag{C.3}$$

Comparing with others, the one with smaller value of $\hat{D}$ is most likely to be the reference template.

Appendix D

Markov Process

Let S be a finite or countable set and $X_0, X_1, X_2, ...$ be a sequence of random variables whose ranges are contained in S. The sequence is a Markov process if

$$\mathrm{P}[X_{n+1} = j | X_0 = i_0, ..., X_n = i_n] = \mathrm{P}[X_{n+1} = j | X_n = i_n] = \mathrm{P}_{ij} \tag{D.1}$$

Equation (D.1) may be interpreted as stating that, for a Markov process, the conditional distribution of any future state X_{n+1} given the past states and depends only on the present state. The value P_{ij} represents the probability that the process transits from state i to state j. Since probabilities are in the range [0, 1] and the process must make a transition into some states, we have that

$$\mathrm{P}_{ij}, \quad N \geq i, j \geq 0; \qquad \sum_{j=0}^{N} \mathrm{P}_{ij} = 1, \quad i = 0, 1, ..., N$$

where N is the total number of state in the process.

Appendix E

Maximum Likelihood (ML)

Maximum likelihood (ML) is a method to solve the estimation problem of model parameters λ. For λ, $\mathrm{P}(O|\lambda)$ is maximized for the given training sequence O. For unobservable data which are a sequence of hidden state $\{q_t\}_{t=1}^{T}$, the Q-function of models λ and $\overline{\lambda}$ can be defined as

$$Q(\lambda, \overline{\lambda}) = \frac{1}{\mathrm{P}(O|\lambda)} \sum_q \mathrm{P}(O, q|\lambda) \log(O, q|\overline{\lambda}) \tag{E.1}$$

Q is a function of $\overline{\lambda}$ in the maximization procedure and $\mathrm{P}(O|\lambda)$ is considered as a constant. With this auxiliary function,

$$Q(\lambda, \overline{\lambda}) \geq Q(\lambda, \lambda) \Rightarrow \mathrm{P}(O|\overline{\lambda}) \geq \mathrm{P}(O|\lambda) \tag{E.2}$$

Considering,

$$\frac{\mathrm{P}(O|\overline{\lambda})}{\mathrm{P}(O|\lambda)} = \sum_q \frac{\mathrm{P}(O, q|\overline{\lambda})}{\mathrm{P}(O|\lambda)} = \sum_q \frac{\mathrm{P}(O, q|\lambda)}{\mathrm{P}(O|\lambda)} \frac{\mathrm{P}(O, q|\overline{\lambda})}{\mathrm{P}(O, q|\lambda)}$$

and taking log on both sides,

$$\begin{aligned} \log \frac{\mathrm{P}(O|\overline{\lambda})}{\mathrm{P}(O|\lambda)} &= \sum_q \log \frac{\mathrm{P}(O,q|\lambda)}{\mathrm{P}(O|\lambda)} \frac{\mathrm{P}(O,q|\overline{\lambda})}{\mathrm{P}(O,q|\lambda)} \\ &\leq \sum_q \frac{\mathrm{P}(O,q|\lambda)}{\mathrm{P}(O|\lambda)} \log \frac{\mathrm{P}(O,q|\overline{\lambda})}{\mathrm{P}(O,q|\lambda)} \\ &= Q(\lambda, \overline{\lambda}) - Q(\lambda, \lambda) \end{aligned}$$

The reestimation formulas of Equations (3.19) and (3.20) in Section 3.2.4 can be derived directly by maximizing the auxiliary function $Q(\lambda, \overline{\lambda})$. Since

$$\log \mathrm{P}(O, q|\overline{\lambda}) = \log \overline{\pi}_{q_1} + \sum_{t=1}^{T-1} \log \overline{a}_{q_t, q_{t+1}} + \sum_{t=1}^{T} \log \overline{b}_{q_t}(o_t)$$

therefore

$$Q(\lambda, \overline{\lambda}) = Q_{\pi}(\lambda, \pi) + \sum_{i=1}^{N} Q_{a_i}(\lambda, a_i) + \sum_{i=1}^{N} Q_{b_i}(\lambda, b_i)$$

where

$$\begin{aligned} \pi &= [\pi_1, \pi_2, ..., \pi_N], \\ a_i &= [a_{i1}, a_{i2}, ..., a_{iN}], \\ b_i &= [b_i(1), b_i(2), ..., b_i(K)] \end{aligned}$$

and

$$\begin{aligned} Q_{\pi}(\lambda, \pi) &= \sum_{i=1}^{N} \mathrm{P}(O, q_1 = i|\lambda) \log \pi_i, \\ Q_{a_i}(\lambda, a_i) &= \sum_{j=1}^{N} \sum_{t=1}^{T-1} \mathrm{P}(O, q_t = i, q_{t+1} = j|\lambda) \log a_{ij}, \\ Q_{b_i}(\lambda, b_i) &= \sum_{t=1}^{T} \mathrm{P}(O, q_t = i|\lambda) \log b_i(o_t) \end{aligned}$$

Therefore these individual terms are to be maximized to achieve the maximization of $Q(\lambda, \overline{\lambda})$ over $\overline{\lambda}$. The individual auxiliary functions $(Q_{\pi}, Q_{a_i}, Q_{b_i})$ are all in the form

$$\sum_{j=1}^{N} w_j \log y_j$$

and it is a function of $\{y_j\}_{j=1}^{N}$ which is subject to the stochastic constraints. Therefore, each function attains a global maximum at the single point

$$y_j = \frac{w_j}{\sum_{i=1}^{N} w_i}, \qquad j = 1, 2, ..., N$$

Hence,

$$\begin{aligned} \overline{\pi}_i &= \frac{\mathrm{P}(O,q_1=i|\lambda)}{\sum_{i=1}^{N} \mathrm{P}(O,q_1=i|\lambda)} = \frac{\mathrm{P}(O,q_1=i|\lambda)}{\mathrm{P}(O|\lambda)} \\ \overline{a}_{ij} &= \frac{\sum_{t=1}^{T-1} \mathrm{P}(O,q_t=i,q_{t+1}=j|\lambda)}{\sum_{j=1}^{N}\sum_{t=1}^{T-1} \mathrm{P}(O,q_t=i,q_{t+1}=j|\lambda)} = \frac{\sum_{t=1}^{T-1} \mathrm{P}(O,q_t=i,q_{t+1}=j|\lambda)}{\sum_{t=1}^{T-1} \mathrm{P}(O,q_t=i|\lambda)} \\ \overline{b}_i(k) &= \frac{\sum_{t=1}^{T} \mathrm{P}(O,q_t=i|\lambda)\delta(o_t,v_k)}{\sum_{t=1}^{N} \mathrm{P}(O,q_t=i|\lambda)} \end{aligned}$$

where

$$\delta(o_t, v_k) = \begin{cases} 1 & \text{if } o_t = v_k \\ 0 & \text{otherwise} \end{cases}$$

By using forward-backward algorithm,

$$\begin{aligned}
\overline{a}_{ij} &= \frac{\sum\limits_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum\limits_{t-1}^{T} \alpha_t(i) \beta_t(i)} \\
\overline{b}_i(k) &= \frac{\sum\limits_{t=1}^{T} \alpha_t(i) \beta_t(i) \delta(o_t, v_k)}{\sum\limits_{t-1}^{T} \alpha_t(i) \beta_t(i)}
\end{aligned}$$

Appendix F

Multiple Training

For increasing the accuracy in modeling of HMM or 2dHMM, multiple training is necessary from multiple observation sequences [24]. For a single observation sequence to train a model (i.e., for reestimation of model parameters), the transient nature of the states within a model allows only a small number of observations for any state (until a transition is made to a successor state). Therefore, using multiple observation sequences can make reliable estimates of all model parameters. For a set of K observation sequences as

$$O = (O,^{(1)}, O^{(2)}, ..., O^{(K)})$$

where $O^{(k)} = (o_1^k, o_2^k, ..., o_T^k)$ is the k^{th} observation sequence. Each observation sequence is independent of every other observation sequence. The parameters of the model is adjusted to maximize

$$\mathrm{P}(O|\lambda) = \prod_{k=1}^{K} \mathrm{P}(O^{(k)}|\lambda) = \prod_{k=1}^{K} \mathrm{P}_k \tag{F.1}$$

Since the reestimation formulas are based on frequencies of occurrence of various events, the reestimation formulas for multiple observation sequences are modified by adding together the individual frequencies of occurrence for each sequence. The modification of the reestimation procedures for HMM and 2dHMM are in Sections F.1 and F.2.

F.1 HMM

The modified reestimation formulas for $\overline{a}_{ij}$ and $\overline{b}_j(l)$ are

$$\overline{a}_{ij} = \frac{\sum\limits_{k=1}^{K} \frac{1}{\mathrm{P}_k} \sum\limits_{t=1}^{T_k-1} \alpha_t(i) a_{ij} b_j(o_{t+1}^k) \beta_{t+1}^k(j)}{\sum\limits_{k=1}^{K} \frac{1}{\mathrm{P}_k} \sum\limits_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)} \tag{F.2}$$

and

$$\overline{b}_j(l) = \frac{\sum\limits_{k=1}^{K} \frac{1}{\mathrm{P}_k} \sum\limits_{t \in o_t = v_l} \alpha_t^k(i) \beta_t^k(i)}{\sum\limits_{k=1}^{K} \frac{1}{\mathrm{P}_k} \sum\limits_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)} \tag{F.3}$$

and π is not reestimated since $\pi_1 = 1$, $\pi_i = 0$ for $i \neq 1$.

F.2 2dHMM

The modified reestimation formulas for $\overline{a}'_{i'j'}$, $\overline{a}''_{i''j''}$, $\overline{b}'_{j'}(y')$ and $\overline{b}''_{i'',j'',j'}(y'')$ are

$$\overline{a}'_{i',j'} = \frac{\sum\limits_{k=1}^{K} \frac{1}{\mathrm{P}_k} \sum\limits_{t=1}^{T_k-1} \sum\limits_{i'',j''} \alpha_t(i',i'') a'_{i',j'} b'_{j'}(o'^{(k)}_{t+1}) a''_{i'',j'',j'} b''_{j'',j'}(o''^{(k)}_{t+1}) \beta_{t+1}(j',j'')}{\sum\limits_{k=1}^{K} \frac{1}{\mathrm{P}_k} \sum\limits_{t=1}^{T_k-1} \sum\limits_{j',i'',j''} \alpha_t(i',i'') a'_{i',j'} b'_{j'}(o'^{(k)}_{t+1}) a''_{i'',j'',j'} b''_{j'',j'}(o''^{(k)}_{t+1}) \beta_{t+1}(j',j'')} \tag{F.4}$$

and the estimation of the probability $a''_{i'',j'',i'}$ is,

$$\overline{a}''_{i'',j'',i'} = \frac{\sum\limits_{k=1}^{K} \frac{1}{\mathrm{P}_k} \sum\limits_{t=1}^{T_k-1} \sum\limits_{i'} \alpha_t(i',i'') a'_{i',j'} b'_{j'}(o'^{(k)}_{t+1}) a''_{i'',j'',j'} b''_{j'',j'}(o''^{(k)}_{t+1}) \beta_{t+1}(j',j'')}{\sum\limits_{k=1}^{K} \frac{1}{\mathrm{P}_k} \sum\limits_{t=1}^{T_k-1} \sum\limits_{i',j''} \alpha_t(i',i'') a'_{i',j'} b'_{j'}(o'^{(k)}_{t+1}) a''_{i'',j'',j'} b''_{j'',j'}(o''^{(k)}_{t+1}) \beta_{t+1}(j',j'')} \tag{F.5}$$

In the discrete case, the reestimation formula for the model λ' is,

$$\overline{b}'_{j'}(y') = \frac{\sum\limits_{k=1}^{K} \frac{1}{\mathrm{P}_k} \sum\limits_{t \in o'_t = v'_l} \sum\limits_{i',i'',j''} \alpha_t(i',i'') a'_{i',j'} b'_{j'}(o'^{(k)}_{t+1}) a''_{i'',j'',j'} b''_{j'',j'}(o''^{(k)}_{t+1}) \beta_{t+1}(j',j'')}{\sum\limits_{k=1}^{K} \frac{1}{\mathrm{P}_k} \sum\limits_{t=1}^{T_k-1} \sum\limits_{i',i'',j''} \alpha_t(i',i'') a'_{i',j'} b'_{j'}(o'^{(k)}_{t+1}) a''_{i'',j'',j'} b''_{j'',j'}(o''^{(k)}_{t+1}) \beta_{t+1}(j',j'')} \tag{F.6}$$

where

$$\delta(m,n) = \begin{cases} 1 & \text{if } m = n \\ 0 & \text{otherwise} \end{cases}$$

The output probabilities for λ'' are estimated as,

$$\bar{b}''_{i'',j'',j'}(y'') = \frac{\sum_{k=1}^{K} \frac{1}{\mathrm{P}_k} \sum_{t \in o''_t = v''_l} \sum_{i',i''} \alpha_{t-1}(i',i'') a'_{i',j'} b'_{j'}(o'^{(k)}_{t+1}) a''_{i'',j'',j'} b''_{j'',j'}(o''^{(k)}_{t+1}) \beta_{t+1}(j',j'')}{\sum_{k=1}^{K} \frac{1}{\mathrm{P}_k} \sum_{t=1}^{T_k-1} \sum_{i',i''} \alpha_t(i',i'') a'_{i',j'} b'_{j'}(o'^{(k)}_{t+1}) a''_{i'',j'',j'} b''_{j'',j'}(o''^{(k)}_{t+1}) \beta_{t+1}(j',j'')} \tag{F.7}$$

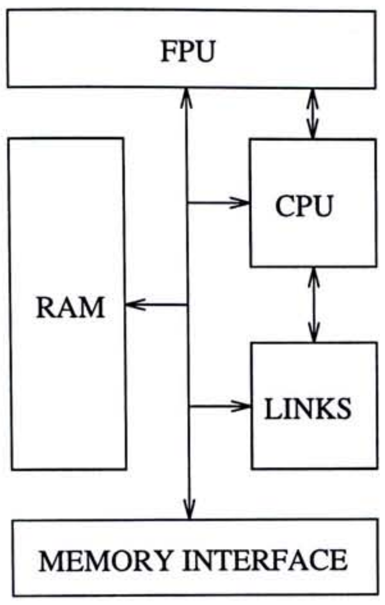
Appendix G

IMS T800 Transputer

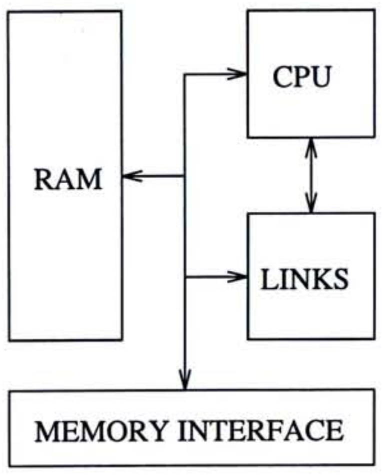
G.1 IMS T800 architecture

The IMS T800, with its on-chip floating point unit, is only 20% larger in area than the IMS T414. The small size and high performance come from a design with takes careful note of silicon economics. This contrasts starkly with conventional co-processors, where the floating point unit typically occupies more area than a complete micro-processor, and requires a second chip (or in the case of the Weitek 1167 floating point processor for the Intel 80386, second, third and fourth chips).

The architecture of the IMS T800 is similar to that of the IMS T414. However, in addition to the memory, links, central processing unit (CPU), and external memory interface, there is a micro-coded floating point unit (FPU) which operates concurrently with and under the control of the CPU. The block diagrams below indicate the way in which the major blocks of the IMS T800 and IMS T414 are interconnected. The CPU of the IMS T800, just like that of the IMS T414, contains three registers (A, B and C) used for integer and address arithmetic, which form a hardware stack. Loading a value into the stack pushes B into C, and A into B, before loading A. Storing a value from A pops B into A and C into B. In addition there is an O register which is used in the formation of instruction operands. Similarly, the FPU includes a three register floating-point evaluation stack, containing the AF, BF and CF registers. When values are loaded onto, or stored from the stack the AF, BF and CF registers push and pop in the same way as the A, B and C registers.



IMS T800



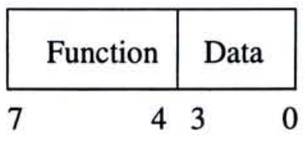
IMS T414

The addresses of floating point values are formed on the CPU stack, and values are transferred between the addressed memory locations and the FPU stack under the control of the CPU. As the CPU stack is used only to hold the addresses of floating point values, the wordlength of the CPU is independent of that of the FPU. Consequently, it would be possible to use the same FPU together with, for example, a 16 bit CPU such as that used on the IMS T212 transputer.

The IMS T800, like the IMS T414, operates at two priority levels. The FPU register stack is duplicated so that when the IMS T800 switches from low to high priority none of the state in the floating point unit is written to memory. This results in a worst-case interrupt response of only 2.5 μs (-30), or 3.7 μs (-20). Furthermore, the duplication of the register stack enables floating point arithmetic to be used in an interrupt routine without any performance penalty.

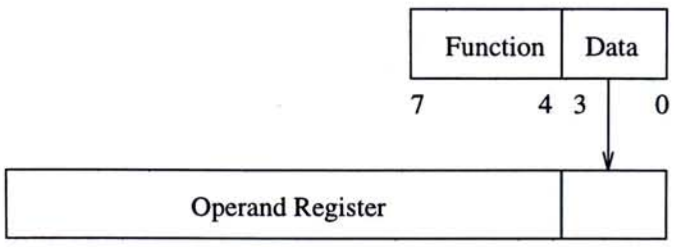
G.2 Instruction encoding

All transputers share the same basic instruction set. It contains a small number of instructions, all with the same format, chosen to give a compact representation of the operations most frequently occurring in programs. Each instruction consists of a single byte divided into two four bit parts. The four most significant bits are



a function code, and the four least significant bits are a data value. The sixteen functions include loads, stores, jumps and calls and enable the most common instructions to be represented in a single byte. As this encoding permits only 4 bits of operand per instruction two of the function codes (*prefix and negative prefix*) are used to allow the data part of any instruction to be extended in length. Another of the sixteen functions (*operate*) treats its data portion as an operation on values held in the processor registers. This allows up to 16 such operations to be encoded in a single byte instruction.

All instructions are executed by loading the four data bits into the least significant four bits of the O register, which is then used as the instruction's operand. All instructions except the prefix instructions end by clearing the O register, ready for the next instruction. The *prefix* instruction loads its four data bits into the O



register, and then shifts the O register up four places. The *negative prefix* instruction is similar, except that it complements the operand register before shifting it up. Consequently operands can be extended to any length up to the length of the operand register by a sequence of prefix instructions.

The prefix functions can be used to extend the operand of an *operate* instruction just like any other. The instruction representation therefore provides for an indefinite number of operations. The encoding of operations is chosen so that the most common operations, such as *add* and *greater than*, are represented without a *prefix* instruction.

The IMS T800 has additional instructions which load into, operate on, and store from, the floating point register stack. It also contains new instructions which support color graphics, pattern recognition and the implementation of error correcting codes. These instructions have been added whilst retaining the existing IMS T414 instruction set. This has been possible because of the extensible instruction encoding used in transputers.

G.3 Floating point instructions

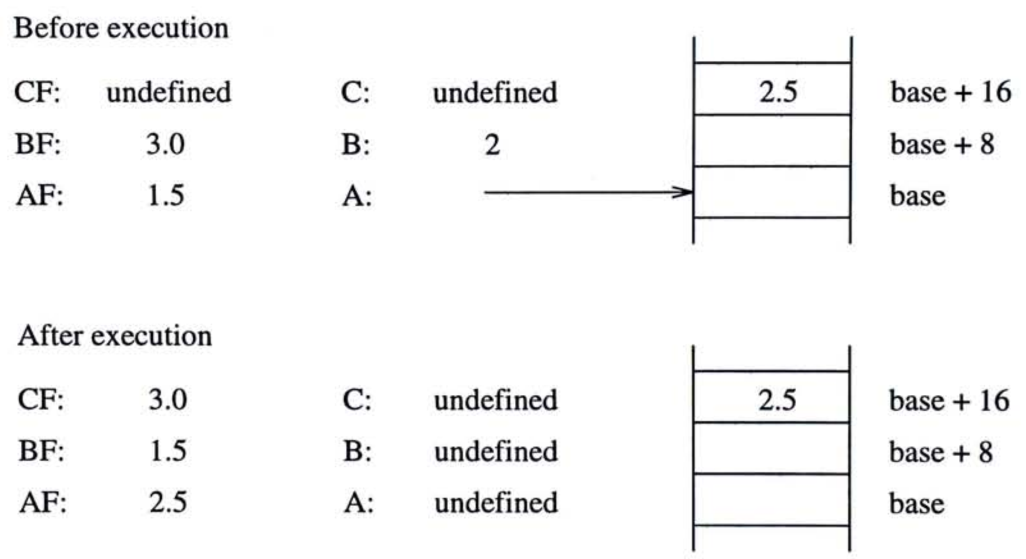
The core of the floating point instruction set was established fairly early in the design of the IMS T800. This core includes simple load, store and arithmetic instructions. Examination of statistics derived from Fortran programs suggested that the addition of some more complex instructions would improve performance and code density. Proposed changes to the instruction set were assessed by examining their effect on a number of numerical programs. For each proposed instruction set, a compiler was constructed, the programs compiled with it, and the resulting code then run on a simulator. The resulting instruction set is now described.

In the IMS T800 operands are transferred between the transputer's memory and the floating point evaluation stack by means of floating point load and store instructions. There are two groups of such instructions, one for single length numbers, one for double length. In the description of the load and store instructions which follow only the double length instructions are described. However, there are single length instructions which correspond with each of the double length instructions.

The address of a floating point operand is computed on the CPU's stack and the operand is then loaded, from the addressed memory location, onto the FPU's stack. Two new addressing operations have been added to the CPU to improve access to double-word (64-bit real and integer) values. The first of these, *word subscript double*, is used to index double-word values. The second of these, *duplicate*, is used when the CPU has to manipulate the addresses of both the more significant and less significant words of a double word object.

Operands in the floating point stack are tagged with their length. The operand's tag will be set when the operand is loaded or is computed. The tags allow the number of instructions needed for floating point operations to be reduced; there is no need, for example, to have both *floating add single* and *floating add double* instructions; a single *floating add* will suffice.

There are two instructions to load double length floating point numbers into the floating point evaluation stack from the transputer's memory. These are *floating load non-local double* and *floating load indexed double.* The *floating load non-local double* instruction loads the value pointed to by the A register of the CPU's stack. The *floating load indexed double* instruction has the same effect as the instruction *word subscript double* followed by *floating load non-local double.* The value in the B register is used as a double-word offset from the base pointer in the A register and the selected double length value is loaded into the AF register. The diagram below shows the effect of executing a *floating load indexed double* instruction. The



effect of the *floating load indexed* instructions can be achieved by a sequence of just two instructions. However, their presence does decrease code size; the *floating load indexed double* instruction is encoded in only two bytes, whereas the equivalent instruction sequence would require four bytes. This appears to be a worthwhile optimization as this instruction sequence would be compiled for every array access.

However, there are just two floating store instructions, *floating store non-local single* and *floating store non-local double.* These both store the value in the AF register into the location pointed to by the A register. There are no *floating store indexed* instructions. This may be surprising given that the *floating load indexed* instructions exist; however, in any program there are less store operations than load operations and, therefore, there is less to be gained by optimizing store (write to memory) operations than optimizing load (read from memory) operations.

The common floating point operations of addition, subtraction, multiplication and division are provided by single instructions. These instructions operate on values in the AF and BF registers, storing the result of the operation into the AF register and popping the CF register into the BF register. Similarly, the floating point comparison operations, *floating point greater than* and *floating point equality,* compare values stored in the AF and BF registers; however, they load the result of the comparison into the A register of the CPU.

G.4 Optimizing use of the stack

The depth of the register stacks in the CPU and FPU is carefully chosen. Floating point expressions commonly have embedded address calculations, as the operands of floating point operators are often elements of one dimensional or two dimensional arrays. The CPU stack is deep enough to allow most integer calculations and address calculations to be performed within it. Similarly, the depth of the FPU stack allows most floating point expressions to be evaluated within it, employing the CPU stack to form addresses for the operands.

No hardware is used to deal with stack overflow. A compiler can easily examine expressions and introduce temporary variables in memory to avoid stack overflow. The number of such temporary variables can be minimized by careful choice of the evaluation order; an algorithm to perform this optimization is given in [61]. The algorithm, already used to optimize the use of the integer stack of the IMS T414, is also used for the main CPU of the IMS T800.

G.5 Concurrent operation of FPU and CPU

In the IMS T800 the FPU operates concurrently with the CPU. This means that it is possible to perform an address calculation in the CPU whilst the FPU performs a floating point calculation. This can lead to significant performance improvements in real applications which access arrays heavily. This aspect of the IMS T800's performance was carefully assessed, partly through examination of the 'Livermore Loops'. These are a collection of small kernels designed to represent the types of calculation performed on super-computers. They are of interest because they contain constructs which occur in real programs which are not represented in such programs as the Whetstone benchmark. In particular, they contain accesses to two and three-dimensional arrays, operations where the concurrency within the IMS T800 is used to good effect. In some cases the compiler is able to choose the order of performing address calculations so as to maximize overlapping; this involves a modification of the algorithm mentioned earlier.