Contents

Abbreviations

Chapter 1 Introduction 1

Chapter 2 System Block Diagram 3

Chapter 3 Hardware Description 5

3.1 Overview of Hardware Configuration 5

3.2 Host 6

3.3 Oculus-F/64 6

3.4 SPIRIT-40 7

Chapter 4 Design Software Skeleton 10

4.1 Cooperation of Multiprocessor 10

4.2 Software Development Environment 11

4.2.1 Oculus-F/64 11

4.2.2 Developing Application Programs for SPIRIT-40 12

4.3 Flowchart 13

Chapter 5 Technical Details 14

5.1 Functions Executed in the Initial State 14

5.1.1 VSR Initialization 14

5.1.2 ASR Initialization 15

5.2 Functions Executed in the Acquisition State 15

5.2.1 Real-time Feature Extraction in ASR 15

5.2.2 Image Sequence Grabbing of Lip motion in VSR 16

5.3 Functions Executed in the Recognition State 17

5.3.1 Feature Extraction in VSR 18

5.3.2 Time-Delay Neural Network (TDNN) 23

5.3.3 Hidden Markov Model (HMM) 27

5.3.4 Transmition of the ASR and VSR Results through Comm Port 35

5.3.5 Sensor Fusion 35

Chapter 6 Results 37

6.1 Experiment Results 37

6.2 Comparison to Other Systems 40

References

Abbreviations

2D-FFT	two dimensional fast fourier transform
AOI	area of interest
ASR	audio speech recognition
AVSR	audio-visual speech recognition
BP	back-propagation
CCD camera	charge-coupled device camera
DMA	direct memory access
DSP	digital signal processor
F/64	Oculus-F/64 DSP board
GSP	graphic signal processor
HMM	Hidden Markov Model
LPC	linear predictive coding
ODX	Oculus Driver
SED	speech endpoint detection
SP-40	SPIRIT-40 DSP board
TDNN	Time-Delay Neural Network
VRAM	Video RAM
VSR	visual speech recognition

CHAPTER
1

Introduction

Recent developments in the design of human computer interface (HCI) are trending toward a new field of research: Multimodal Systems. The purpose of such systems is to increase the accuracy and naturalness of human computer interaction by integrating speech with other sources of information. For example, they may permit people to speak and write in their own native language, to point or gesture while speaking, to view a synthesized human face with synchronized lip movements and emotional expressions while listening to speech, to participate in simulated virtual environments with accompanying speech, and to retrieve and manipulate information stored in rich multimedia formats(e.g., text, graphics, video, speech, hand drawn marks and writing, and so forth).

There are many potential advantages of well-designed multimodal systems. One is the support of robust system performance under adverse conditions. For example, adequate recognition of spoken language could be maintained in a noisy environment with supplementary visual information about corresponding lip movements. The integration of visual and auditory information gathered from the speaker’s facial movements becoming relatively more salient in a noisy environment.

One clear experimental demonstration of how visual cues are integrated with auditory ones during speech recognition is provided by the "McGurk effect" [1]. During this effect, a person observes a videotaped face saying "ga" while listening to "ba" on a soundtrack. The perceptual result is that the auditory and visual information merges, such that the person reports hearing "da." Furthermore, the sound reported can be manipulated by having the person make judgments with eyes shut and open.

Inspired by these empirical results, computationalists have begun attempting to integrate auditory and visual information to improve the accuracy and robustness of speech recognition [2], [3], [4], [5], [6]. Their results support the belief that multimodal systems may display more desirable properties, especially under realistic field conditions, than stand-alone spoken language systems.

Aiming for a multimodal system, a bimodal speech recognition system using acoustic speech signals and lip motions as input sources was first built in this project. Although the present results are limited (speaker dependent, isolated word, small-vocabulary, not real-time), our system is superior to most of the existed bimodal systems. Our system which is an on-line recognition system implemented on PC platform differs from most of the existed bimodal systems which are off-line recognition systems implemented on workstation platforms. Because of on-line recognition, our system is more convenient for testing system performance and closer to the realistic situations. In some off-line systems, hand-segmented steps such as extracting lip region from face or labeling time-stamps on video tape to synchronize the visual signal and acoustic signal are needed, or speaker′s lip region needs to be marked to simplify feature extraction. In our system, however, these hand-segmented steps are automatically processed, and natural image sequence is used as input source for lipreading. Furthermore, the cost of our system is less expensive. Owing to the progress of DSP technology, we can perform so large amount of computations of speech and image data on PC platform, while this is only possible on workstation (or workstations) in those days.

CHAPTER
2
System Block Diagram

Fig. 2.1 shows the Audio-Visual Speech Recognition (AVSR) system consisting of three subsystems: Audio Speech Recognition (ASR), Visual Speech Recognition (VSR), and Sensor Fusion.

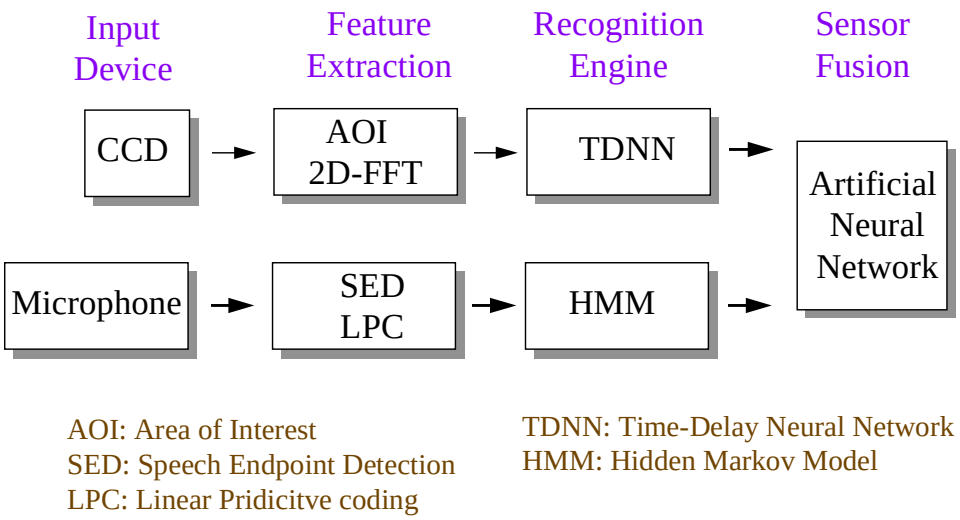


Fig. 2.1

In ASR, acoustic data are gathered by a microphone, then passed to feature extraction unit in which the following processes are executed: speech endpoint detection (SED), signal preprocessing, and LPC computation. The LPC feature is subsequently sent to the recognition engine -- Hidden Markov Model (HMM) to score the audio recognition result. The functional frame of VSR is identical to that of ASR, but processes in each component are different. The differences include: a CCD camera used to grabbing lip-movement image sequence substitutes the microphone in

input device; lip region (area of interest) extracted from face and its 2D-FFT substitute the SED and LPC in feature extraction; Time-Delay Neural Network substitutes HMM in recognition engine. After ASR and VSR have been done, the sensor fusion subsystem combines audio and visual observations to make the final decision. The purpose of sensor fusion is to maximize the performance of the combined system over as wide range of environmental conditions as possible.

The basic timing of system operation can be simply described as follows. In acquisition state, both of microphone and CCD camera gather data continuously. Once voice-end is detected by ASR (feature extraction in ASR is real-time), input devices stopped grabbing data (VSR is informed by ASR through PC) and the system is switched to recognition state. In recognition state, ASR starts to recognize the speech signal using HMM, and VSR starts to extract feature and recognize the lip-image sequence using TDNN. Finally, the sensor fusion subsystem fuses the audio and visual results, and the system is back to acquisition state to wait for the next triggering signal of voice.

C H A P T E R

3

Hardware Description

3.1 Overview of Hardware Configuration

The hardware configuration is shown in Fig. 3.1. The visual channel begins with a TOSHIBA CCD camera with a 16mm lens. It is connected to a Coreco's OCULUS-F/64 DSP board that performs the task of VSR. The audio information is obtained with a National WM333N microphone, routed through a SONITECH's SAIB for preamplification and digitization. A SONITECH's SPIRIT-40 DSP board is used to perform tasks of ASR and sensor fusion. A PC-Pentium/90 is used to be a HOST to control the activities of the SPIRIT-40 and F/64.

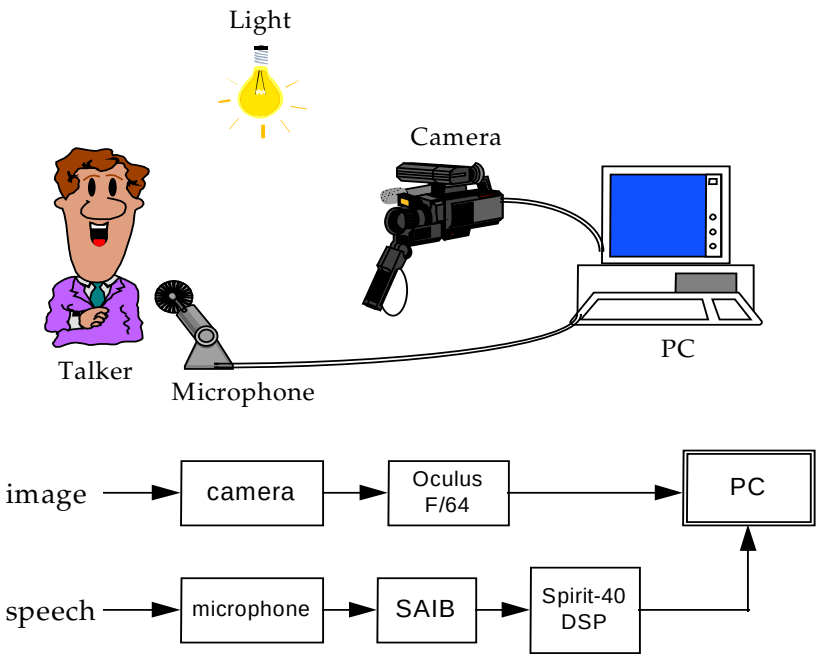


Fig. 3.1

In brief, Oculus-F/64, SPIRIT-40, and PC constitute the kernel of the AVSR system. The DSP boards form the computation center that process most of image and speech data. PC administrates the system operation, and is responsible for user interface. In the rest of this chapter, we shall describe the kernel of the system in more details.

3.2 Host

The Host supervises the operation of the whole system. It first resets the two DSP boards, and performs functions such as controlling the operations of F/64 and synchronizing the start of the SPIRIT-40 program (which is executed on C40 DSP processor). It also takes care of the user interface and data transfer to and from the secondary storage devices.

3.3 Oculus-F/64

The Oculus-F/64 is a high performance image processing board for the ISA 16-bit bus. Targeted at sophisticated image processing applications, Oculus-F/64 incorporates a high speed digital signal processor (TMS320C40) and graphic signal processor (TMS34020), both of which operate in conjunction with a real-time histogram processor and Coreco‘s proprietary “Image Processing Engine” (IP-Engine). The block diagram of Oculus-F/64 is shown in Fig. 3.2.

The Oculus-F/64 has a flexible acquisition section which provides easy interfacing to different video sources. The input video format is programmable and limited only by the amount of memory installed on board and the 40 MHz maximum sampling rate (8-bit A/D). The TMS34020 graphic signal processor (GSP) provides basic image processing functionality and also provides acquisition control (up to 40 MegaSamples /sec). The Oculus-F/64 is also equipped with TMS320C40 DSP. This provides enhanced acquisition sampling rates, a powerful general image processing engine, and control for the Oculus-F/64 bus interface. Real-time IP Engine and histogram processor provide point processing on incoming or feedback images.

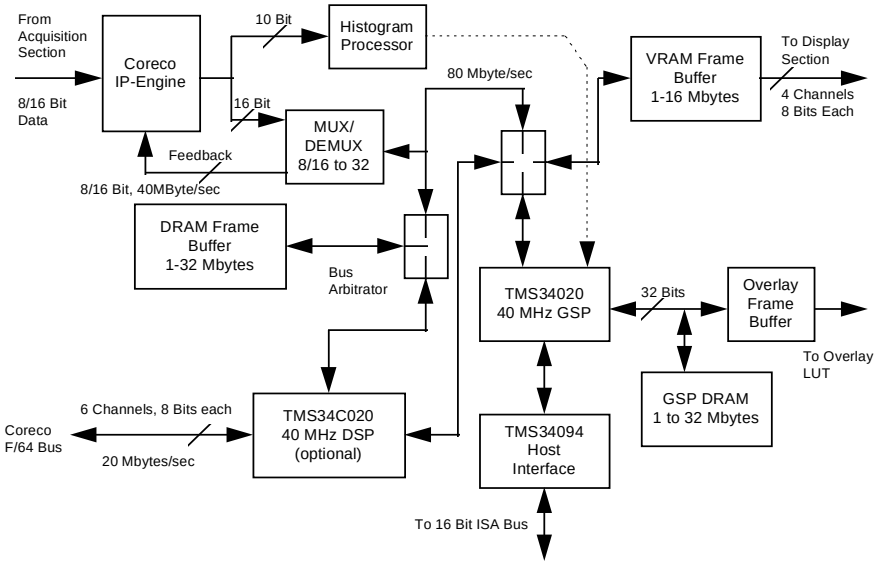


Fig. 3.2

The primary memory modules on the Oculus-F/64 include VRAM frame buffer, DRAM frame buffer, and GSP DRAM. Both VRAM and DRAM frame buffer can be used for image acquisition and processing, but only VRAM can be used as display memory since the contents of VRAM memory can be viewed via the Oculus-F/64 display system. Physical frame buffer arrays can be programmed into smaller logical frame buffers enabling acquisition and processing of multiple images. Apart from being used as frame buffer, the remaining DRAM frame buffer is used to store TMS320C40 DSP program code. The GSP DRAM, however, is simply used to store TMS34020 GSP program code. There are currently 4 Mbyte VRAM, 16 Mbyte DRAM frame buffer, and 1 Mbyet GSP DRAM installed on our Oculus-F/64.

3.4 SPIRIT-40

The SPIRIT-40 is a DSP accelerator card which offers up to 80 MFLOPs of performance for PC AT (ISA Bus) based system. The heart of the SPIRIT-40 is the two Texas Instruments TMS320C40 parallel Digital Signal Processors. Operating at 40 MHz , the C40 features an instruction cycle time of 50ns.

Fig. 3.3 shows the SPIRIT-40 functional block diagram. The host computer (PC) can directly access the SRAM interfaced to the Global Bus of each TMS320C40

processor. The SPIRIT-40 ISA is mapped into the I/O address space of the host computer and occupies eight consecutive 16-bit addresses. The address, data and control information between the host computer and the SPIRIT-40 system is exchanged via the data registers through the SPIRIT-40 Global Bus SRAM banks. The Low Address Registers are used in conjunction with the High Address Registers to address locations within the SPIRIT-40‘s two Global Bus SRAM banks for host I/O transfers. The control/status registers are used to control and monitor SPIRIT-40 operations.

Fig. 3.3

The SPIRIT-40 provides twelve high-speed communication ports (Comm Ports) for inter-processor communication and interfacing with peripherals. Two of the Comm Ports (one from each C40) are used for direct inter-processor communication and are not available for general purpose use. The other ten Comm Ports (four from

each C40) are available for use with internal devices, and are located along the top of the SPIRIT-40.

There are four memory arrays on the SPIRIT-40 (two for each TMS320C40), each consisting of 256Kx32 zero wait state SRAM. This memory resides on the four independent buses of the two C40s.

C H A P T E R

4

Design Software Skeleton

4.1 Cooperation of Multiprocessor

As mentioned in the last section, there are five CPUs in our AVSR system, Pentium, F/64 DSP, F/64 GSP, SPIRIT-40 DSP1, and SPIRIT-40 DSP2. For simplicity, these CPUs are respectively designated as Host, DSP0, GSP, DSP1, and DSP2 in the successive paragraphs. Due to the regularity of the task of GSP, the custom code supported by Coreco company is adopted as the GSP program code in our system. Therefore, we only need to take care of the programming on Host, DSP0, DSP1, and DSP2. Fig. 4.1 shows the diagram of cooperative processors in this system.

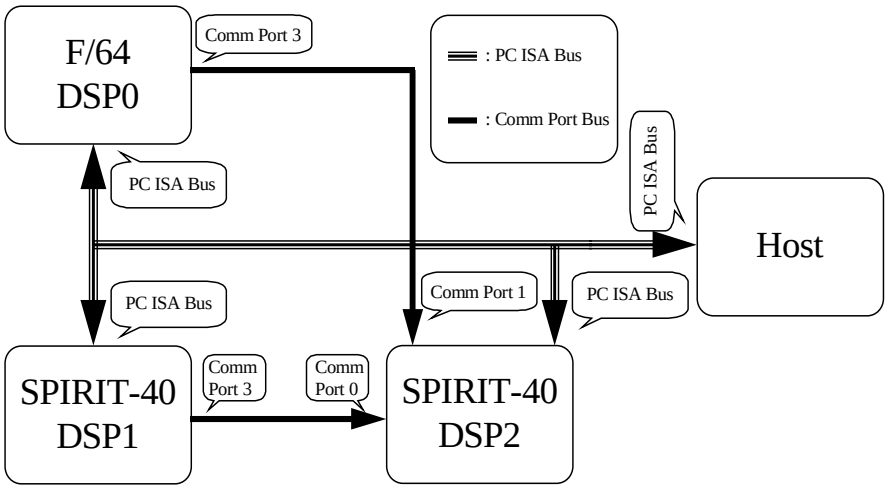


Fig. 4.1

Jobs Assignment

- Host : control the activities on F/64, and SPIRIT-40, and user interface.
- DSP0 : perform the task of VSR system such as image data acquisition, feature extraction, and recognition.

- DSP1 : perform the task of ASR system such as acoustic data acquisition, feature extraction, and recognition.
- DSP2 : perform the task of sensor fusion.

Data Transmission among processors

- Host $\leftrightarrow$ DSP0, DSP1, DSP2 : The interface of the data transmission between Host and one of the three DSPs is ISA bus of the Host.
- DSP0 $\rightarrow$ DSP2: Data from DSP0 is sent to DSP2 through Comm Port 3 of DSP0, and received through Comm Port 1 of DSP2 by DSP2.
- DSP1 $\rightarrow$ DSP2: Data from DSP1 is sent to DSP2 through Comm Port 3 of DSP1, and received through Comm Port 0 of DSP2 by DSP2.

4.2 Software Development Environment

4.2.1 Oculus-F/64

Oculus-F/64 Driver: ODX

The basic software structure is shown in Fig. 4.2. Host program executed on PC controls the activities on F/64 board via Oculus Driver (ODX). ODX is implemented as a Terminate and Stay Resident (TSR) program under DOS. The application communicates with ODX using a message passing interface. ODX is a single thread process. Only one command is processed at a time. Messages sent while a driver is busy are queued and are processed on a FIFO basis.

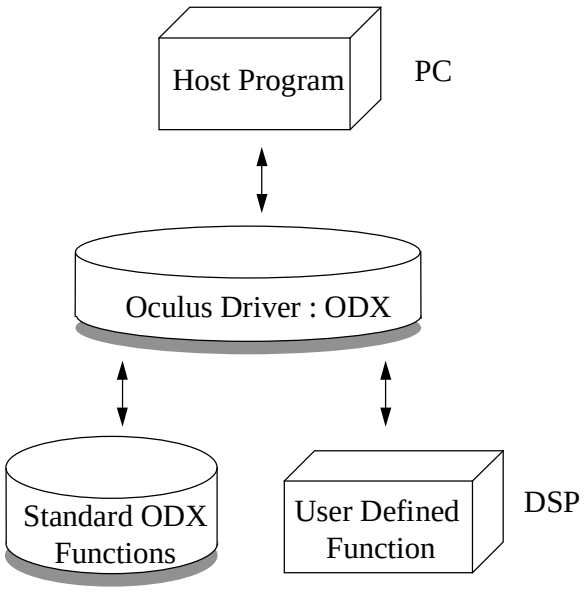


Fig. 4.2

The Oculus driver supports a series of standard ODX functions for application development running on the host computer in either the DOS or Windows environment. Typical functions include acquisition and display control, frame buffer read and write operations. Image processing functions include histograms, contour following, convolutions and morphological operations.

The program developed on DSP0 is simply treated as User Defined Functions which are extensions of standard ODX functions. User Defined Functions are accessed from the host application by using a call to ODX with the function number 47. In such way, the host application can use User Defined Functions just as standard ODX functions.

F/64 Frame Buffer Memory

Frame buffer Memory on the Oculus F/64 is composed of VRAM and DRAM. VRAM memory is used for image display and both DRAM and VRAM memory are used for acquisition and processing.

The total memory available on F/64 board is divided into logical partitions called frame buffers. The frame buffer dimensions are selected in the F/64 configuration program. In most cases multiple logical frame buffers are available and they will all have the same dimensions. The size of the logical frame buffers is the same in VRAM and DRAM.

Tools used

The Tools used for developing the Oculus-F/64 program include:

- The Coreco ODX Development Toolkit
- The Oculus-F/64 DSP Development Toolkit
- TMS320C40 C Compiler User‘s Guide
- TMS320C4x User‘s Guide

4.2.2 Developing Application Programs for SPIRIT-40

This section describes the steps in developing application programs on the SPIRIT-40 board. It is useful to have the following manuals at hand while writing application programs for the SPIRIT-40:

- TMS320C40 C Compiler User‘s Guide

- TMS320C4x User‘s Guide
- TMS320C40 Assembly Language Tools User‘s Guide

We developed two programs, one for the Host PC and one for the SPIRIT-40 according to the following four steps. Steps 1 and 2 deal with developing the PC-side software. Step 3 and 4 deal with developing the DSP-side (SPIRIT-40) program.

Step 1:

In this step, we develop the PC-side program. The algorithm for the PC-side software is:

- Download the SPIRIT-40 program (COFF file).
- Get the addresses of the required C40 global variables (the symbol table from the executable COFF file).
- Reset the TMS320C40 to start its execution.
- Wait for the SPIRIT-40 initialization to complete by polling a semaphore flag.
- Download the data required by SPIRIT-40 C40.
- Signal the SPIRIT-40 program to start.
- Wait for the desired event happening by polling a event-semaphore flag.
- Process the event if event-semaphore flag is set.
- Wait for the SPIRIT-40 programs to complete, and upload the resulting data from the SPIRIT-40 if required.

Step 2:

In Step 1 we developed the PC-side software source file (xxx.c). We can now create the executable file using the C complier.

Step 3:

In this step, We develop the DSP-side program. We organize it into two modules, one written in C which contains a control routine, and the other written in C40 assembly language which performs fast computation.

Step 4:

The source files for the DSP program are ready now, so we can create the C40 executable file (COFF file) in this step.

4.3 Flowchart

Fig. 4.3 shows the flowchart of software. The color of block indicates the operation state of system when the block is working, where the operation states include: initial state, acquisition state, and recognition state. Moreover, the jobs undertaken by the same processor are located in the same column, and the communications and data transmissions among processors are indicated by transverse connections.

CHAPTER
5

Technical Details

As shown in the software flowchart, there are three operation states in our AVSR system: initial state, acquisition state, and recognition state. At initial state, proper working environments for Oculus-F/64 and SPIRIT-40 are set up. After the system is ready to work, it alternates between acquisition state and recognition state depending on whether voice-end is detected. This chapter is organized according to these three operation states. In the following sections, the theoretic background, algorithms, and implementation details will be discussed.

5.1 Functions Executed in the Initial State

5.1.1 VSR Initialization

The Host performs the following initialization for the VSR:

- Oculus-F/64 firmware setting:

  Call batch file to initialize the firmware on boards, such as FPGA setting, GSP.out download, f64dsp.out (which are previously complied and linked using the TMS320C40 C compiler or assembler by the user).

- Bind to Oculus-F/64 driver (ODX).
- Reset F/64 and select device.
- Download weight data of TDNN to DSP0.

  Dynamically allocate memory on DSP0 for TDNN based on the size of network whose configuration is recorded in a file. By this way, we can test the performance of different structures of TDNN conveniently because we

can easily change the network structure without compiling, linking, and downloading DSP program again.

5.1.2 ASR Initialization

The Host performs the following initialization for the ASR:

- Download the SPIRIT-40 programs onto it‘s C40s:

  Host selects C40 processor and download the executable DSP program to it (which is previously complied and linked using the TMS320C40 C compiler or assembler).

- Reset C40s on the SPIRIT-40.

  To start C40s execution, Host first resets them.

- Initialize the SAIB.

  Host initializes SAIB in order to program A/D configuration consisting chiefly of sampling rate, gain and resolution of A/D.

5.2 Functions Executed in the Acquisition State

5.2.1 Real-time Feature Extraction in ASR

This section describes the processing of real-time feature extraction. Assume A/D sampling rate is f Hz, and resolution of A/D is B-bit. We need memory block A and memory block B for storing the sampling data, and each block size is L*B bits. Then, We can divide time t equally between memory blocks, and get the interval L/f. For convenience of explanation, a new time-axis, denoted as Kw, is defined by

$$Kw=floor(t*f/L),$$

where floor(x) finds the largest integer which is not larger than x.

Fig. 5.1 shows the operation of the real-time feature extraction. At time Kw, only one memory block, A or B, is used for storing the new sampling data from A/D until it is full. It is called **acquisition buffer**, and the other is called **processing buffer** which is used for extracting feature of the data sampled at time Kf=Kw-1. For the next recognition state, we need a N-size buffer to store the feature which is extracted at each time Kf. For storing it in feature buffer, Kf must module feature buffer size N.

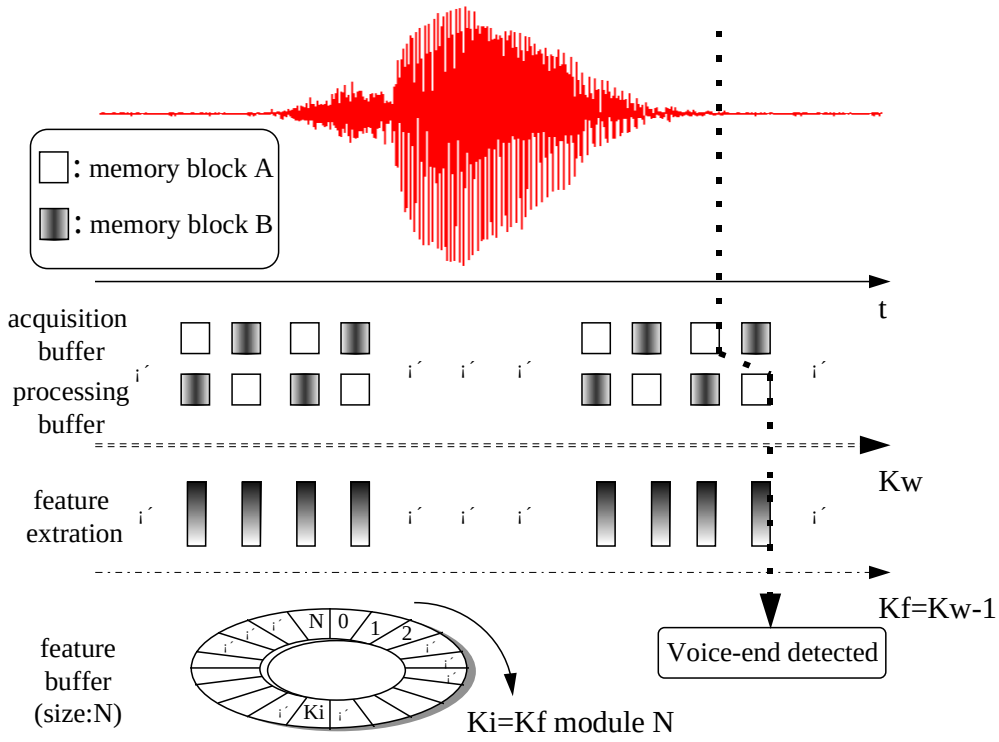


Fig. 5.1

5.2.2 Image Sequence Grabbing of Lip motion in VSR

A good plan of memory management is important for continuously grabbing image sequence in a finite capacity of memory. In the acquisition state, the image data is rapidly accumulated in memory as time going. If there is not a good strategy for memory management, memory will soon be fully occupied.

We adopt a structure of circular queue to solve this problem, which is frequently used in real-time processing. As shown in Fig. 5.2, a physically linear-addressed memory is transformed to a logically circular-addressed memory. Whenever the index of frame buffer reaches the bound, it restarts from zero. In this way, image sequence can be continuously grabbed without worrying about lack of memory.

The determination of sampling rate is also important. There is a trade-off between sampling rate and number of total frame buffers. If sampling rate is too low, information will be dropped too much. On the contrary, if sampling rate is too high, there aren‘t enough frame buffers to store the redundant information. The number of total frame buffers should also be large enough to record the whole image sequence

which starts from mouth-opening and ends at mouth-closed; otherwise, overlapping would occur in the circular-addressed memory.

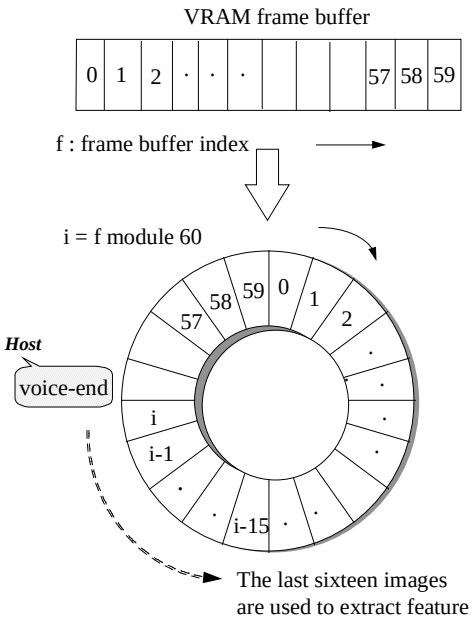


Fig 5.2

In our system, the VRAM is partitioned into 60 frame buffers, each of which can store an image of size 256 by 256. At sampling period of 66 ms, the duration of image sequence that VSR can grab correctly without overlapping is 66*60=3960 ms, which is long enough for a speaker saying an isolated-word.

The continuously grabbing is stopped when a voice-end signal detected by ASR is informed by Host. Then the last sixteen images are passed to feature extraction unit of VSR, and the state of system goes into recognition state.

5.3 Functions Executed in the Recognition State

In the recognition state, the major functions of AVSR, such as HMM, TDNN, and sensor fusion, are executed. Besides, feature execution in VSR is also executed in this state since the AOI extraction and 2D-FFT computation take too much time to be real-time processed in acquisition the state.

5.3.1 Feature Extraction in VSR

The images grabbed by the CCD camera often cover the whole or part of a speaker's face, and the speaker's face may not always locate in the same place every time he (she) speaks. Hence, the area of interest (AOI) around lip needs to be extracted from the face image.

After AOI extraction is accomplished, further feature extraction process is needed since the original images sequence still contain too much redundancy and are not good for classification directly.

Lip Region Extraction From Face Image: AOI

Lip-reading is the method by which a reader comprehends what a speaker is saying, by observing the movements of the speaker's mouth. Many psychological experiments in the area of audio-visual speech recognition have been conducted. The principal results of the research (Dodd and Campbell, 1987; Brooke and Summerfield, 1983) demonstrated that lip movements provide information for the understanding of language.

There are several reports on automatic lip-reading research based on image processing techniques. Numerous feature extraction techniques have been used in automatic lip-reading. Our approach to video preprocessing makes use of several simple computations to reduce the large amount of data to a manageable set of low-level image statistics describing the region of interest around the mouth. These statistics capture such features as the positions of the upper and lower lip.

Edge detection is by far the most common approach to detecting meaningful discontinuities in gray level. In edge detection an important quantity is the gradient vector. Horizontal gradients are useful to detect the left and right boundaries of shapes. Whereas vertical gradients are useful to detect the upper and lower boundaries of shapes. However, even the boundaries of the facial apparatuses are detected, it is insufficient to determine which area is the mouth. Fortunately, there are two helpful clues to locate the mouth : first, the mouth is often much darker than other area on the frontal face, as a result, there will be significant gradient magnitude around the boundary of the mouth; second, the mouths are always in the shapes of the horizontal bar, and thus the lower and upper boundaries are projected into a narrow

region of the horizontal projection. Based on these two clues, the integral projections and vertical gradient methods were used for finding the mouth.

The vertical and horizontal integral projections are defined as follows. Let $I(x,y)$ be an image. The vertical integral projection of $I(x,y)$ in the $[x_1, x_2]\times[y_1, y_2]$ rectangle is defined as

$$\mathrm{V}(x) = \sum_{y=y_1}^{y_2} I(x,y).$$

The horizontal integral projection is similarly defined as

$$\mathrm{H}(y) = \sum_{x=x_1}^{x_2} I(x,y).$$

The gradient of an image $I(x,y)$ at location (x,y) is the vector

$$\nabla I = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial I}{\partial x} \\ \dfrac{\partial I}{\partial y} \end{bmatrix}.$$

Therefore, the vertical location of human mouth can be detected by vertical gradients rather than horizontal gradients. In order to search the smallest rectangle containing the mouth area, vertical gradients are used to locate the mouth.

In the practical experiment, the images were acquired with a CCD camera at frontal views. The subjects were asked to look into the camera to ensure perfectly frontal images. The illumination was partially controlled; the same powerful light was used. The images were taken randomly during the day time, and the distance of the subject from the camera was fixed approximately. Each video image sequence contained sixteen frames of frontal image at a resolution of 256×256 pixels. Some assumptions were made for simplicity and stability: first, the subject hold his poise during one video sequence; second, the mouth was closed in the first image; third, the target frontal face is of adapted size, i.e., the frontal face occupied the most area of the image. On such assumptions, two phenomena were found by observing the configuration of the mouth as shown in Fig. 5.3: the lower boundary of the mouth is around the lowest positive-to-negative zero crossing of the horizontal integral projections of the vertical gradients; the upper boundary is around the fourth zero crossing from the lower boundary due to human mouth configuration. In other words, observing the configuration of the mouth could explain why the zero crossing

methods were used. First, the region below the lower boundary of the mouth is darker than the boundary itself. It is the reason why the first positive-to-negative zero crossing from the bottom of the image was chosen as the lower boundary of the mouth. Second, the line between two lips is the darkest region over all the mouth and thus it results in the second negative-to-positive zero crossing and the third positive-to-negative zero crossing. Finally, the fourth negative-to-positive zero crossing represented the upper boundary of the mouth.

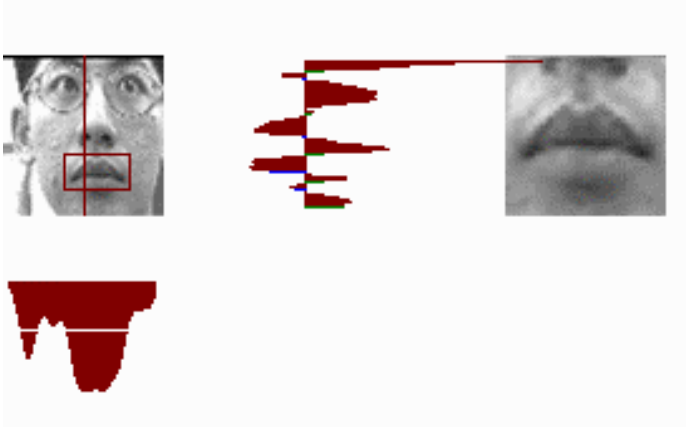
According to the above discussion, a area of interest (AOI) is detected by means of the following operations on the full video image sequence:

- Down sample the image sequence at a resolution of 64×64 pixels.
- Extract the vertical gradients of the first image of the video sequence.
- Calculate the horizontal integral projections of the vertical gradients.
- Smooth the horizontal integral projections of the vertical gradients by low-pass filters.
- Find the zero crossings of the horizontal integral projections of the vertical gradients.
- Number those zero crossings in the bottom-up order. Take the first positive-to-negative zero crossing of the image as the lower boundary of the mouth. Furthermore, take the fourth zero crossing from the lower boundary as the upper boundary.
- Extract the vertical integral projection of the vertical gradient of the area between the upper and lower boundaries of the mouth.
- Find the peak value of the vertical integral projections of the vertical gradients, and get the horizontal position correspondent to the peak value.
- Start from the horizontal position obtained in the previous step to the right side of the vertical integral projections of the vertical gradients. Find the point whose value of the vertical integral projection of the vertical gradient is first below the predefined threshold; the horizontal position of this point is treated as the right side boundary of the mouth.

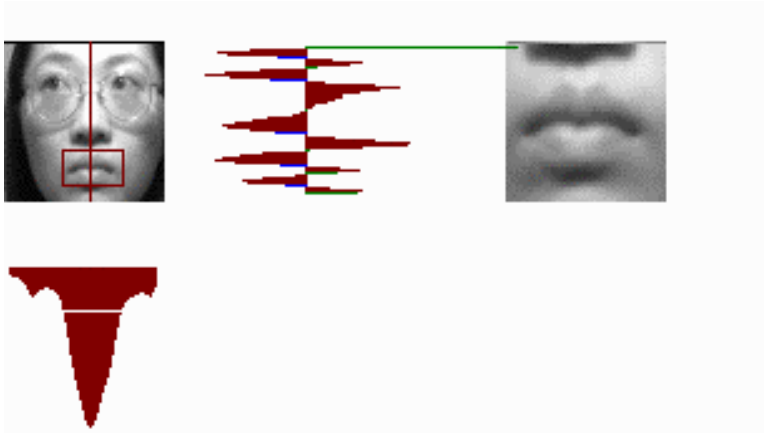
- Apply the same procedure to the vertical integral projection of the vertical gradients. Start from the horizontal position with peak value to the left side of the vertical gradient projections to find the left boundary of the mouth.

Since the purpose of the sixteen images were used as input data of the TDNN, after 2D-FFT transformation, the AOI of these images were normalized to image sizes of 64×64 pixels for the consistent properties of the training data. The normalized images were extracted from the original full video image sequence, i.e., after finding the AOI of the image, the correspondent four corner points of the AOI were found in the original image and then the original image was normalized into the size of 64×64 pixels. The use of the original image rather than the down sampled image is that the former contained more information.

The results of the proposed procedure to find the region of interest (area of the mouth) are shown in Fig. 5.3. Two different sets of image data are shown in Fig 5.3(a) and 5.3(b), respectively. Each image in the left-top corners is the down-sampled original image (64×64). The regions within the dark rectangles in the original images are the regions of interest; the dark vertical lines in the original images correspond to horizontal positions of the peaks of the vertical integral projections of the vertical gradients. The statistical graphs in the bottom and the left of the original images represent the horizontal and vertical integral projections of the vertical gradients, respectively; the white horizontal lines in the graphs of vertical integral projection represent the predefined thresholds. The images in the right-top corners are the normalized AOI of size 64×64 pixels.



(a)



(b)

Fig. 5.3 Illustration of the procedure for finding the area of interest (AOI).
(see the text for details)

2D-FFT

As mentioned previously, a second-stage feature extraction is required to preprocess the image of AOI before the image sent to recognition engine. In vision, however, there does not exist such a well established body of preprocessing techniques as in speech recognition. Basically preprocessing for connectionist vision is split into two different "schools of believes": 1. Apply as much as possible preprocessing in a manner such that just some few characteristic features, such as the vertical or horizontal length of mouth, tongue or teeth appears, are fed into the classifier. 2. Apply only transformations with very little information reduction and let the learning scheme find out the characteristic features.

We follow the second approach, because we believe that conventional image processing techniques may cut out in an early stage important feature, which could be useful for the later global learning scheme.

In our system, The AOI is rescaled to a 64×64 pixel image so that two dimensional FFT results also with 64×64 coefficients. We only consider the log magnitudes of the first 13×13 FFT coefficients and rescale them to [-1.0 1.0]. The reason for using FFT is that this coding is spatial shift invariant. This makes the recognition more stable against inaccurate AOI positions.

5.3.2 Time-Delay Neural Network (TDNN)

Background

Time-delay neural networks are powerful tools for the recognition of spatiotemporal patterns that are often encountered in many tasks of cognitive, such as vision, speech recognition, signal processing and motor control. Proposed by Waibel et al., these networks incorporate time-delay elements along their interconnections, and can be trained by the back-propagation (BP) algorithm. Typically each unit receives a set of signals at the current time, and a time history of each signal channel. Error back-propagation then strengthens the weights along interconnections whose time delays are most relevant to the spatiotemporal pattern to be recognized. TDNNs have been successfully used in many applications, such as speech recognition, optical character recognition, and trajectories recognition.

To recognize temporal patterns, a network must have ability to learn temporal relationships between current data and the past history of events that are important. One of the main differences between biological neurons and artificial neural elements is the coding and transmission scheme for signals which take time to travel from neuron to neuron. The TDNN implements corresponding delays along the interconnections of an artificial neural network. Unlike conventional neural networks, which process static patterns, the TDNN can deal with patterns in the time domain.

Network architecture

The basic unit used in many neural networks computes the weighted sum of its inputs and then passes this sum through a nonlinear function, most commonly used a threshold or sigmoid function. In TDNN, this basic unit is modified by introducing delays Z^{-1} through Z^{-K} as shown in Fig. 5.4. Each TDNN unit applies the sigmoid function to $N*(K+1)$ weighted and summed inputs, where N is the number of features in the vector $\vec{f}(t)$, and K is the number of delayed vectors. Thus the input is actually composed of $K+1$ feature vectors $\{f_j(t),..., f_j(t-k),..., f_j(t-K)\}$, where the features are indexed $1 \le j \le N$.

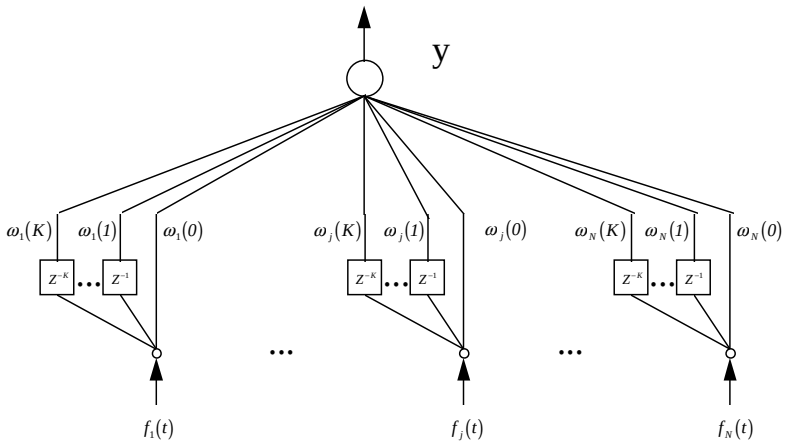


Fig. 5.4

Fig. 5.5 shows the architecture of the TDNN with four layers. Each frame of the TDNN input is a column of 16 features computed at discrete-time intervals. The first hidden layer consists of eight hidden units organized as a column. Each unit is fully connected to the 16 inputs of each feature vector. The outputs of the first hidden layer units are dependent upon the feature vector at time t and the two preceding vectors (for these cells, $N = 16$ and $K = 2$). Thus the units in the first hidden layer do not generate outputs until three successive frames of input features are available. If input frames are computed at 10 ms intervals, the units of the first hidden layer generate their first output after 30 ms, and generate subsequent outputs at 10 ms intervals. This is important because the units of the TDNN integrate activity from adjacent time-delayed vectors, which allows each vector to be weighted separately at several "snapshots" in time. This results in a temporal integration which is not linear; adjacent feature vectors are treated as additional inputs occurring at the same time.

The second hidden layer in Fig. 5.5 is composed of a column of three units. Each of these units is connected to eight units of the first hidden layer. Like the units of the first hidden layer that integrate adjacent input frames before generating an output, the units of the second hidden layer do not generate an output until they process the activity of the units of the first hidden layer over five time intervals.

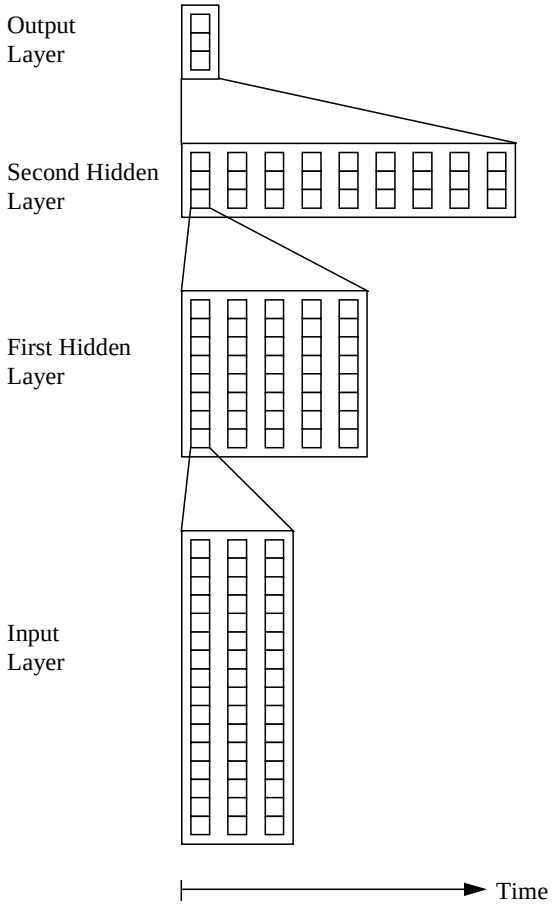


Fig. 5.5

The TDNN output is determined by three units which integrate the activity of the units of the second hidden layer over nine frames. Each output layer unit is connected to only one of the second hidden layer units. This implies that the units of the second hidden layer are class-specific. For each output unit, all weights to the second hidden layer have identical values. Thus the response at the second hidden layer is integrated over time with equal weighting.

TDNN training

The TDNN is trained using the BP algorithm. However, units within the TDNN compute activities at different time intervals, with units in the input layer active at the input-frame rate and units in the output activated only after a complete feature vectors has been processed. Thus a unique implementation of the BP training algorithm has been derived.[7].

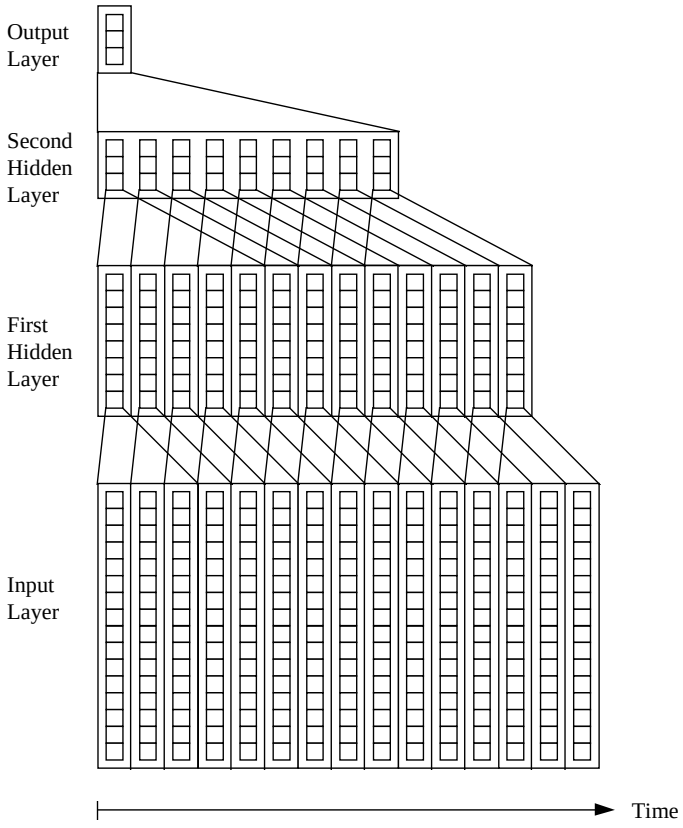


Fig. 5.6

Training is accomplished using copies of the TDNN, with one copy generated for each shift in time (Fig. 5.6). For a pattern containing 15 frames, 13 different copies of the TDNN are created. These consist of 13 copies of the input layer, 9 copies of the first hidden layer, and only one copy of the second hidden and output layers. Network copies enable the entire history of the network activity to be present at once, thereby allowing the BP algorithm to compute the error for each network as if they represented separate instances in time. The average of the change in weights over all copies is used to update the weights. When training is completed, each TDNN copy will have identical weight matrices, and a single TDNN may be used for recognition.

Implementation issues

The TDNN in our AVSR was trained on workstation. After training the learned weights of TDNN were sent to PC for recognition. When AVSR starts running, the

weights are loaded to F/64 DSP board from disk. Hence, the TDNN running on DSP only has recognition ability, and has no learning ability.

To let the TDNN be easily transferred between different platforms and let the learning tasks be managed efficiently, we designed a scheme for the TDNN training. Before training is performed, a configuration file of TDNN was edited, in which the network structure, training strategy and source of training patterns were recorded. The training program would read this file at first to set the training conditions. Therefore, the program needn't be compiled and linked again whenever it does a new learning task with different TDNN configuration. In this manner, we can try different configurations of TDNN and analyze the effects caused by them more conveniently. As for the recognition program running on DSP, the same property is still held since the configuration file is also read by the program at initial state.

5.3.3 Hidden Markov Model (HMM)

General Principles

Speech recognition systems generally assume that the speech signal is a realization of some message encoded as a sequence of one or more symbols (see Fig. 5.7). To effect the reverse operation of recognizing the underlying symbol sequence given a spoken utterance, the continuous speech waveform is first converted to a sequence of equally spaced discrete parameter vectors. This sequence of parameter vectors are assumed to form an exact representation of the speech waveform on the basis that for the duration covered by a single vector (typically 10 ms or so), the speech waveform can be regarded as being stationary. Although this is not strictly true, it is a reasonable approximation. Typical parametric representations in common use are smoothed spectral or linear prediction coefficients plus various other representations derived from these.

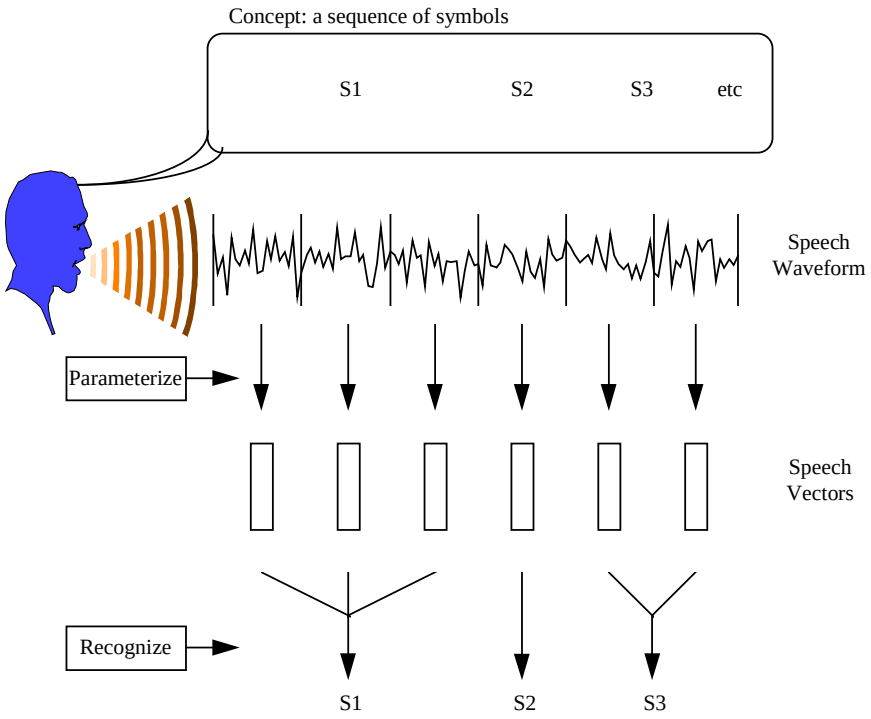


Fig. 5.7

The role of the recognizer is to effect a mapping between sequences of speech vectors and the wanted underlying symbol sequences. Two problems make this very difficult. First, the mapping from symbols to speech is not one-to-one since different underlying symbols can give rise to similar speech sounds. Furthermore, there are large variations in the realized speech waveform due to speaker variability, mood, environment, etc. Second, the boundaries between symbols cannot be identified explicitly from the speech waveform. Hence, it is not possible to treat the speech waveform as a sequence of concatenated static patterns.

The second problem of not knowing the word boundary locations can be avoided by restricting the task to isolated word recognition. As shown in Fig. 5.8, this implies that the speech waveform corresponds to a single underlying symbol (e.g., word) chosen from a fixed vocabulary. Despite the fact that this simpler problem is somewhat artificial, it nevertheless has a wide range of practical applications. Furthermore, it serves as a good basis for introducing the basis idea of HMM-based recognition before dealing with the more complex continuous speech case. Hence, isolated word recognition using HMMs will be dealt with first in this research.

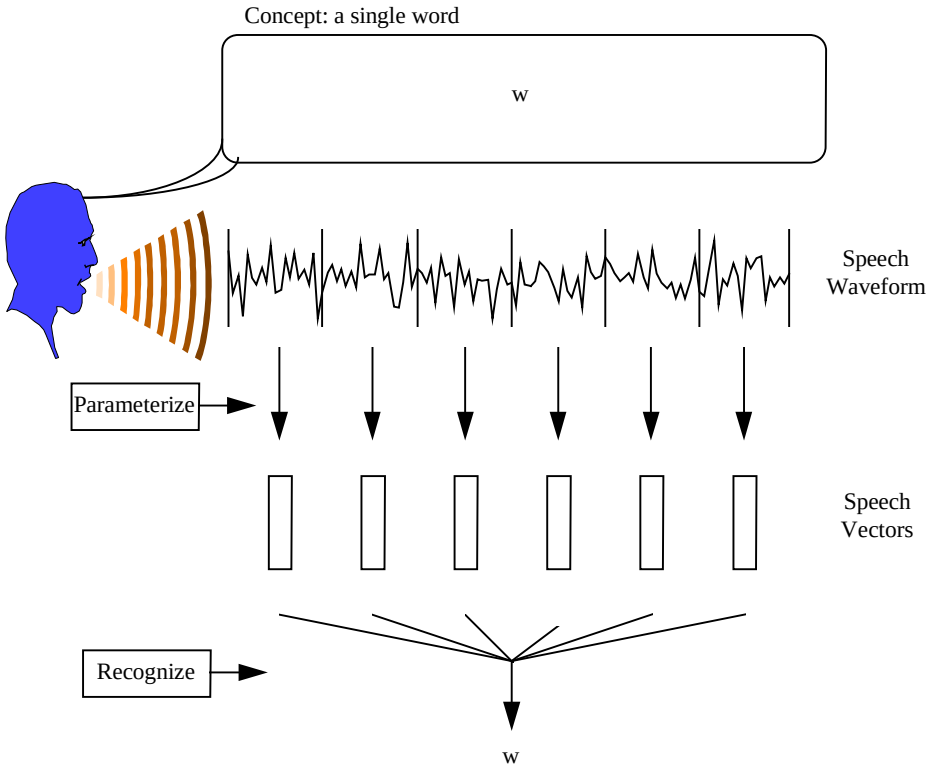


Fig. 5.8

Isolated Word Recognition

Let each spoken word be represented by a sequence of speech vectors or observations O, defined as

$$O = o_1, o_2, \ldots, o_t, \ldots, o_T, \tag{1}$$

where o_t is the speech vector observed at time t. The isolated word recognition problem can then be regarded as that of computing

$$\arg \max_i \{ P(w_i \mid O) \}, \tag{2}$$

where w_i is the i'th vocabulary word. This probability is not computable directly but using Bayes' Rule gives

$$P(w_i \mid O) = P(w_i \mid O)\, P(w_i)/P(O). \qquad (3)$$

Thus, for a given set of prior probabilities $P(w_i)$, the most likely spoken word depends only on the likelihood $P(w_i \mid O)$. Given the dimensionality of the observation sequence O, the direct estimation of the joint conditional probability $P(o_1, o_2, \ldots \mid w_i)$ from examples of spoken words is not practicable. However, if a parametric model of word production such as a Markov model is assumed, the estimation from data is possible since the problem of estimating the class conditional observation densities $P(O \mid w_i)$ is replaced by the much simpler problem of estimating the Markov model parameters.

In HMM based speech recognition, it is assumed that the sequence of observed speech vectors corresponding to each word is generated by a Markov model as shown in Fig. 5.9. A Markov model is a finite state machine which changes state once every time and at each time t that a state j is entered, a speech vector o_t is generated with probability density $b_j(o_t)$. Furthermore, the transition from state i to state j is also probabilistic and is governed by the discrete probability a_{ij}. Fig. 5.9 shows an example of this process where the six state model moves through the state sequence X=1,2,2,3,4,4,5,6 in order to generate the sequence o_1 to o_6.

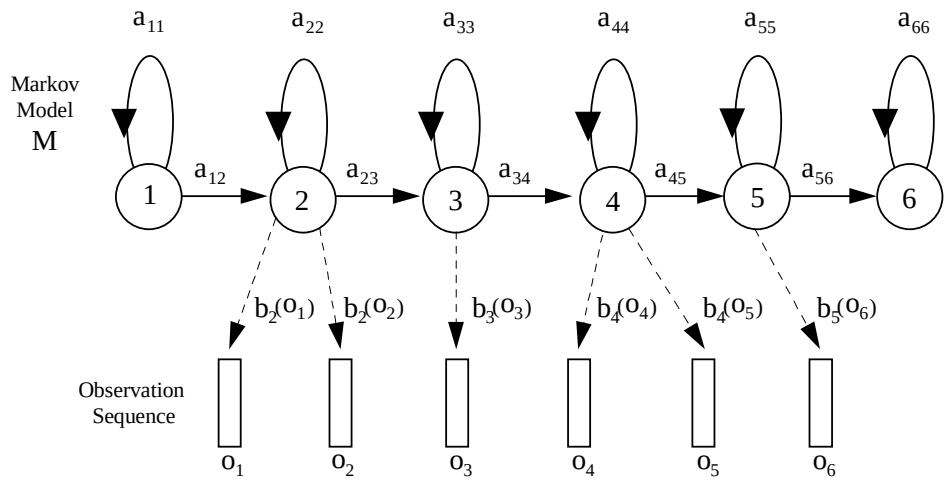


Fig. 5.9

The probability that O is generated given the model M moving through the state sequence X is calculated simply as the product of the transition probabilities and the output probabilities. So for the state sequence X in Fig. 5.9, we have

$$P(O,X|M) = a_{12}\, b_2(o_1)\, a_{22}\, b_2(o_2)\, a_{23}\, b_3(o_3). \tag{4}$$

However, in practice, only the observation sequence O is known and the underlying state sequence X is hidden. This is why it is called a ***Hidden Markov Model***. Given that X is unknown, the require likelihood is computed by summing over all possible state sequences X = **x**(1), **x**(2), **x**(3),..., **x**(T), that is

$$P(O|M) = \sum_{X} a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(o_t)\, a_{x(t)x(t+1)} \tag{5}$$

where **x**(0) is constrained to be the model entry state and **x**(T+1) is constrained to be the model exit state.

As an alternative to Eq. (5), the likelihood can be approximated by only considering the most likely state sequence, that is

$$P'(O|M) = \max_{X} \{ \prod_{t=1}^{T} b_{x(t)}(o_t) a_{x(t)x(t+1)} \}. \quad (6)$$

Although the direct computation of Eqs. (5) and (6) is not tractable, simple recursive procedures exist which allow both quantities to be calculated very efficiently. Before going any further, however, note that if Eq. (2) is computable then the recognition problem is solved. Given a set of models M_i corresponding to words w_i, Eq. (2) is solved by using Eq. (3) under the assumption that

$$P(O|w_i) = P(O|M_i). \quad (7)$$

All this, of course, assumes that the parameters $\{a_{ij}\}$ and $\{b_j(o_t)\}$ are known for each model M_i. Herein lie the elegance and power of the HMM framework. Given a set of training examples corresponding to a particular model, the parameters of that model can be determined automatically by a robust and efficient re-estimation procedure. Thus, provided that a sufficient number of representative examples of each word can be collected then a HMM can be constructed which implicitly models all of the many sources of variability inherent in real speech. Fig. 5.10 summarizes the use of HMMs for isolated word recognition. First, a HMM is trained for each vocabulary word using a number of examples of that word. In the figure, we consider only these words: "one", "two", "three". Second, to recognize some unknown word, the likelihood of each model generating that word is calculated and the most likely model identifies the word.

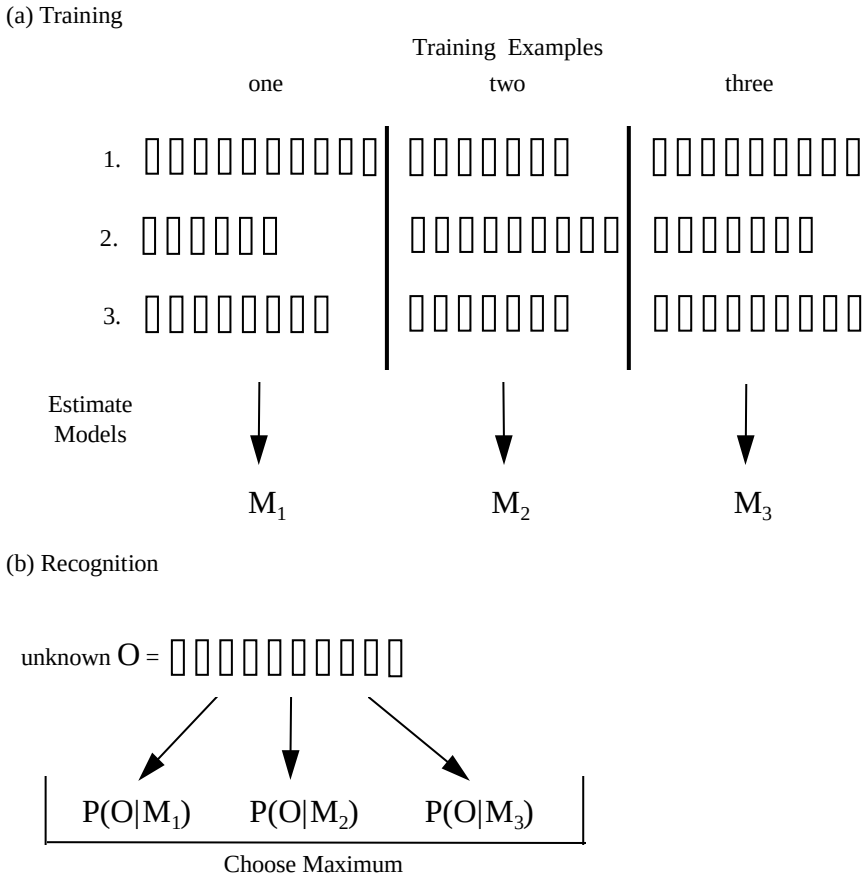


Fig. 5.10

Output Probability Specification

Before the problem of parameter estimation can be discussed in more detail, the form of the output distribution $\{b_j(o_t)\}$ needs to be made explicit. In common with most other continuous density HMM systems, ASR represents output distributions by Gaussian Mixture Densities. In HMM of the ASR subsystem, the observation vector at time t is only one stream. The formula for computing $b_j(o_t)$ is then

$$b_j(o_t) = \mathrm{N}(o;\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} e^{-\frac{1}{2}(o-\mu)'\Sigma^{-1}(o-\mu)} \qquad (8)$$

where $\mathrm{N}(\cdot;\mu,\Sigma)$ is multivariate Gaussian with mean vector μ and covariance matrix Σ, and n is the dimensionality of o.

Viterbi Algorithm

A formal technique for finding the single best state sequence is called the Viterbi algorithm. To find the single best state sequence, X= {**x**(1), **x**(2), **x**(3),..., **x**(T)}={ $q_1\, q_2 \ldots q_T$ },for the given observation sequence, O={ $o_1, o_2, \ldots, o_T$ }, we need to define the quantity

$$\delta_t(i) = \max_{q_1 q_2 \cdots q_{t-1}} P\,[q_1\, q_2 \ldots q_{t-1}, q_t = i, o_1, o_2, \ldots, o_t | \lambda\,]. \qquad (9)$$

That is, $\delta_t(i)$, is the best score (highest probability) along a single path at time t, which accounts for the first t observations and ends in state i. By induction we have

$$\delta_{t+1}(j) = [\max_i \delta_t(i)\, a_{ij}\,] \cdot \; b_j(o_{t+1}). \qquad (10)$$

To actually retrieve the state sequence, we need to keep track of argument that maximized Eq. (10), for each t and j. We do this via array $\psi_t(i)$. The complete procedure for finding the best state sequence can now be stated as follows:

1. Initialization

$$\delta_1(i) = \pi_i\, b_i(o_1), \qquad 1 \le i \le N \qquad (11.a)$$
$$\psi_1(i) = 0 \qquad (11.b)$$

2. Recursion

$$\delta_t(j) = \max_{1 \le i \le N} [\,\delta_{t-1}(i)\, a_{ij}\,] b_j(o_t) \qquad 2 \le t \le T \qquad (12.a)$$
$$\psi_t(j) = \arg\max_{1 \le i \le N} [\,\delta_{t-1}(i)\, a_{ij}\,] \qquad 1 \le j \le N \qquad (12.b)$$

3. Termination

$$P^* = \max_{1 \le i \le N} [\,\delta_T(i)\,] \qquad (13.a)$$

$$q_T^* = \arg \max_{1 \le i \le N} [\delta_T(i)] \qquad (13.b)$$

4. Path (state sequence) backtracking

$$q_t^* = \psi_{t+1}\left(q_{t+1}^*\right), \qquad t=T-1, T-2,...,1. \qquad (14)$$

Note that the direct computation of likelihoods leads to being underflow, so log likelihoods are used instead.

Viterbi Approach Re-Estimation

To determine the parameters of a HMM, it is first necessary to make a rough guess at what they might be. Once this is done, more accurate parameters can be found by applying the so-called Viterbi approach re-estimation formulae. The essential problem is to estimate the means and variances of a HMM in which each state output distribution is single mixture Gaussian, that is

$$b_j(o_t) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}(o-\mu)' \Sigma^{-1} (o-\mu)} . \qquad (15)$$

If a Viterbi approach is used, the mean vectors and covariance matrices for the observation densities are reestimated by simple averaging. This is easily described when there is only one mixture component per state. In this case, the recognition experiment is performed on the observation sequence. Each observation vector is then assigned to the state that produced it on the optimal path by examining the backtracking information. If the observation o_t is produced by state j, let us write "$o_t \sim i$". Suppose that Y observation vectors are so assigned. Then we have

$$\mu_j = \frac{1}{N_j} \sum_{\substack{t=1 \\ o_t \sim j}}^{T} o_t \qquad (16)$$

and

$$\Sigma j = \frac{1}{N_j} \sum_{\substack{t=1 \\ o_t \sim j}}^{T} \left(o_t - \mu_j\right)\left(o_t - \mu_j\right)' . \qquad (17)$$

Recognition

The previous section has described the basic ideas underlying HMM parameter re-estimation using Viterbi Algorithm. It has been recognized as the efficient recursive algorithm for computing the maximum likelihood (P(O|M)) of the model (M). Hence this algorithm could be used for recognition.

5.3.4 Transmition of the ASR and VSR Results through Comm Port

After the results of HMM and TDNN are present, ASR on DSP1 and VSR on DSP0 will initialize DMA channels to send the results to sensor fusion subsystem on DSP2 through comm ports. The sensor fusion subsystem then fuses the audio recognition result and video recognition result to make the final decision.

5.3.5 Sensor Fusion

The means by which the disparate streams of information (from the eyes and from the ears) is integrated is far from understood. This is unfortunate since such an understanding would certainly be helpful to this research.

Massaro and Cohen [8] and Massaro [9, 10] have studied sensor integration in bimodal (auditory plus visual) speech perception. They observed that experimental subjects found extremely difficult to base phonemic discrimination on one mode of perception alone, when both types of stimulus were present. They also found that integration occurred before the binary decision was made. In another experiment, they found that the information was not integrated in a “compromising” manner, but rather an “enhancing” manner. That is, strong support for a hypothesis from both perceptual modes results in a much higher degree of confidence than does support from either mode by itself.

In this project, a two-layer, feedforward neural network shown in Fig. 5.11 is trained to perform sensor fusion. Based on Massaro‘s observation that integration occurs before binary decision, a fully-connected network was designed.

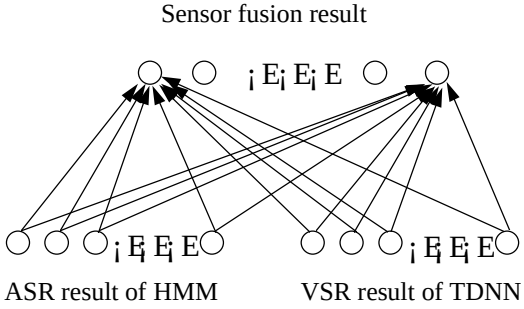


Fig. 5.11

CHAPTER
6
Results

In this project, a on-line bimodal speech recognition system using acoustic data and lip motion as input sources was built. The specification of our AVSR system is speaker-dependent, isolated-word, and small vocabulary. There are two sets of Chinese vocabulary currently used for recognition. One consists of digits 1 to 6, and the other consists of 19 highly-confused words for audio speech recognition which we chose purposely.

6.1 Experiment Results

Fig. 6.1 and Fig. 6.2 show several user-interface screens grabbed during on-line recognition process of our AVSR system. The upper-left window shows the speech waveform and the result of ASR. The lower-left window shows the corresponding image sequence of lip motion, and the result of VSR is shown in the lower-right window. The final result of AVSR is shown in the upper-right window.

Fig. 6.1(a) and Fig. 6.1(b) show the recognition result of two Chinese digits. Fig. 6.2(a) and Fig. 6.2(b) show the recognition results of two Chinese confusing words.

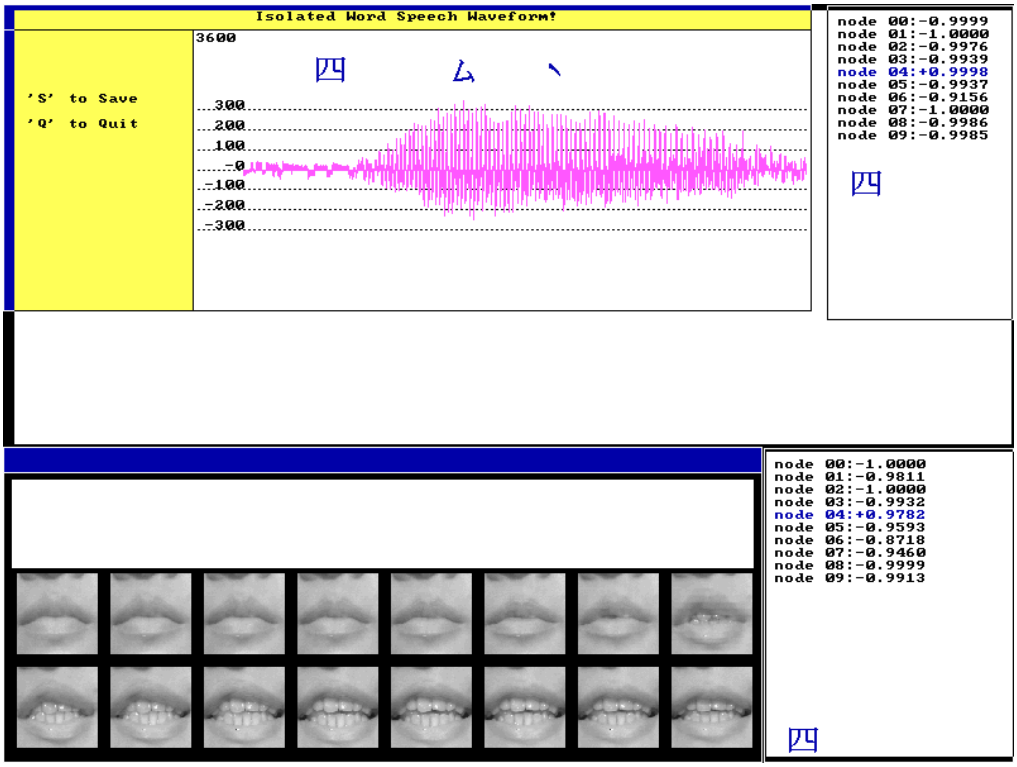


Fig. 6.1(a)

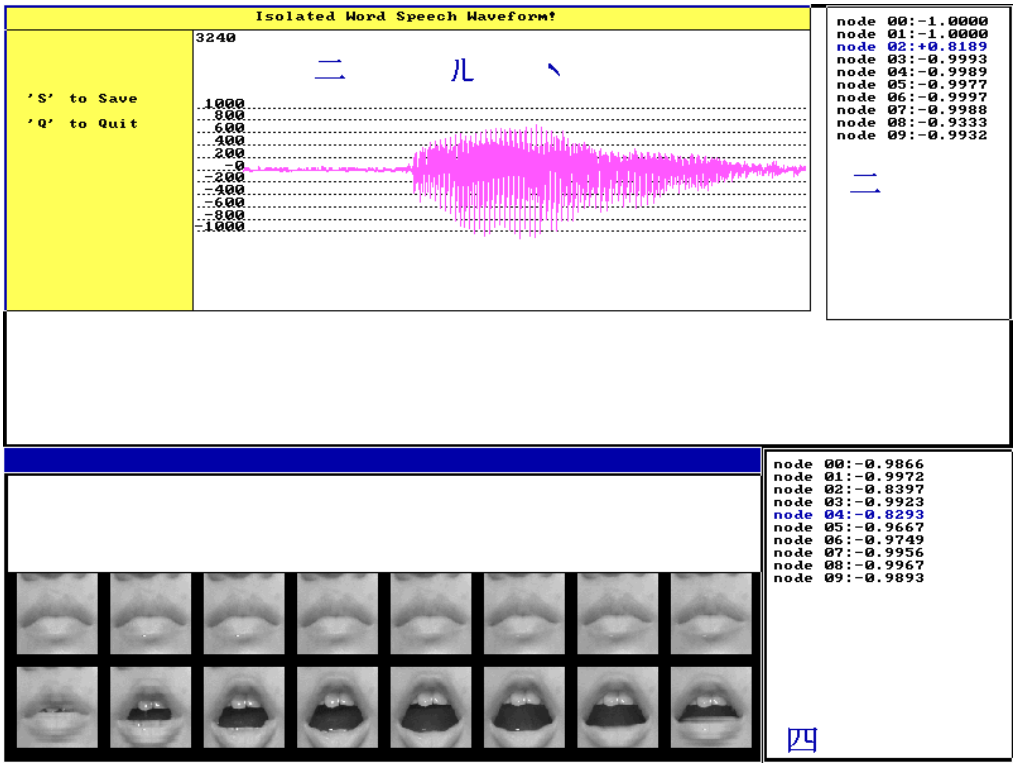


Fig 6.1(b)

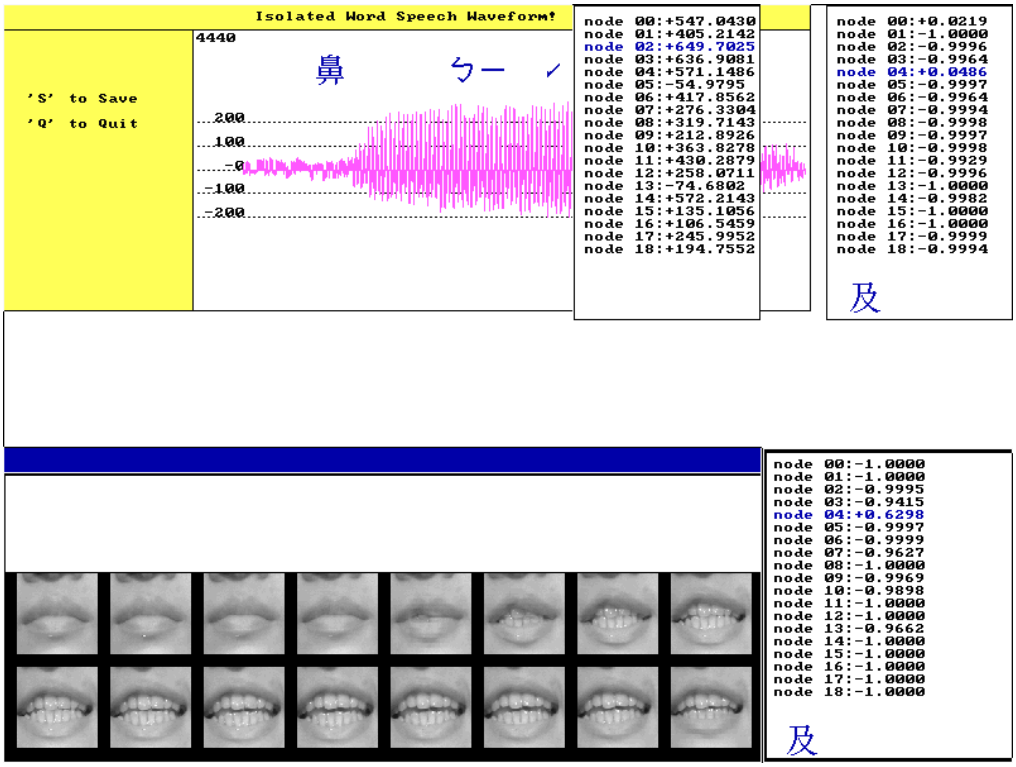


Fig 6.2(a)

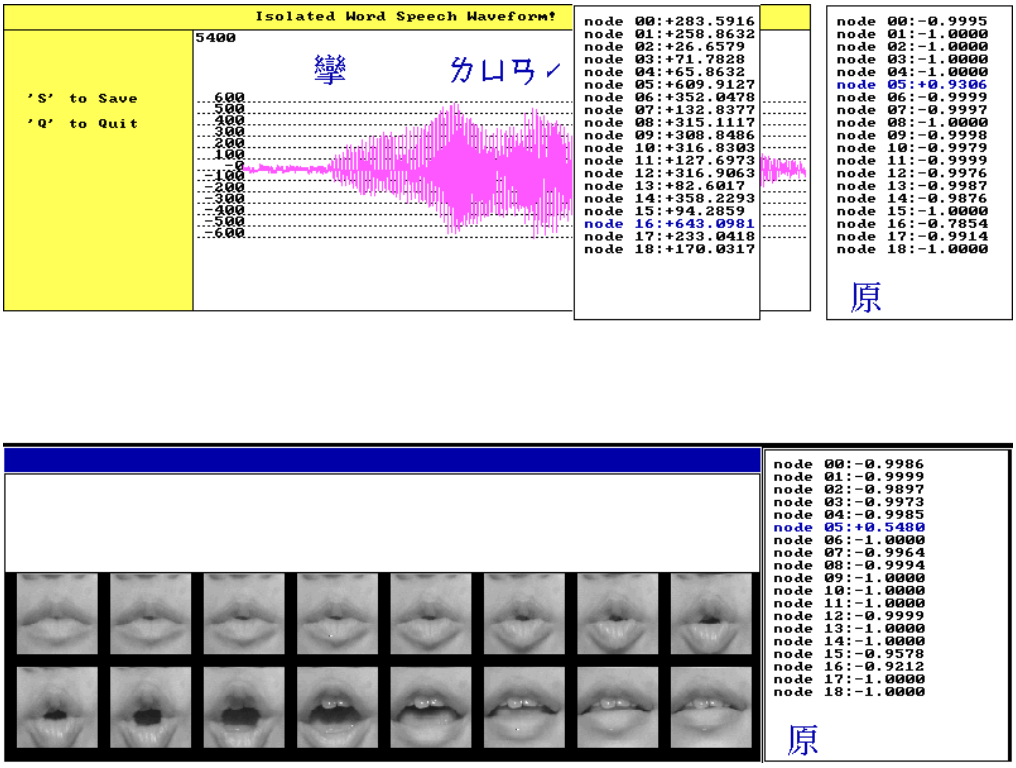


Fig. 6.2(b)

6.2 Comparison to Other Systems

The research on bimodal speech recognition nowaday is still quite limited. Due to different specification, recording environments, and languages, the performance of different systems vary very much. The performance we refer here includes not only recognition rate, but also the degree of how much the experiment condition approaches to the realistic condition. Some researchers are attentive to recognition rate, and relatively less care about practicality, such as some limitation of recognition environment mentioned in Chapter 1. However, we believe that the research goal should be applicable to real life, therefore we try to find a balance between recognition rate and limitations of recognition environment.

AUTHOR	DATA	METHOD	SPEAKER DEPENDENT	VOCABU-LARY	VOCABU-LARY SIZE	RECOGNITION %		ENVIRONMENT
Petajan 1988	Acoustical Optical	Dynamic Time Warping	Yes	Letters, Numbers	36	79 %		Head-geared Camera and Lighting
Pentland 1989	Optical	Optical Flow	No	Digits	10	70 %		Off-line Processing Insensitive to Light
Yuhas 1989	Acoustical Optical	Neural Networks	Yes	Vowel Phonemes	9	80-90 %		Off-line Processing
Stork 1992	Acoustical Optical	Neural Networks (Time Delay)	No	Consonant Phonemes	10	51 %		Off-line Processing Markers on Face Needed
Bregler 1993	Acoustical Optical	Neural Networks	Yes	Connected Letters	26	47 %		Off-line Processing AOI Extraction by Hand-segment
Goldschen 1993	Optical	HMMs	Yes	Continuous Sentences	150 TIMIT Sentences (732 Words)	25 % of Complete Sentences		Off-line Processing Use Petajan's Database
Lin 1996	Acoustical Optical	HMM TDNN	Yes	Isolated Chinese Digits	6	80~90% (On-line)	95~99% (Off-line)	On-line Processing Fixed-lighting
Lin 1996	Acoustical Optical	HMM TDNN	Yes	Isolated Confusing Chinese Words	19	50~60% (On-line)	80~90% (Off-line)	On-line Processing Fixed-lighting

Table 6.1

Table 6.1 shows the comparison results of some bimodal speech recognition systems. It is noted that there is not a well-established database or criterion to judge the system performance. Consequently, the system performance may not be

determined only by the recognition rate. The on-line and off-line recognition rate of our system are shown in the last two columns of Table 6.1. Due to the unified recording conditions, the off-line recognition rate is generally higher than on-line recognition rate.

References

[1] H. McGurk and J. MacDonald, “Hearing lips and seeing voices,” *Nature*, vol. 264, pp. 746-748, 1976.

[2] E. D. Patajan, “Automatic lipreading to enhance speech recognition,” in *Proc. IEEE Commun. Soc. Global Telecommun. Conf.* (Atlanta, GA), Nov. 1984, pp. 26-29.

[3] E. D. Petajan, B. Bischoff, and D. Bodoff, “An improved automatic lipreading system to enhance speech recognition,”, in *Proc. ACM SIGCHI-88*, 1988, pp. 19-25.

[4] A. Pentland and K. Mase, “Lip reading: Automatic visual recognition of spoken words,” in *Proc. Image Understanding and Machine Vision, Optical Soc. Amer.*, June 12-14, 1989.

[5] B. P. Yuhas, Jr. et al., “Neural network models of sensory integration for improved vowel recognition,” *Proc. IEEE*, vol. 78, no. 10, pp. 1658-1668, 1988.

[6] B. P. Yuhas, Jr. Mr. H. Goldstein, and T. J. Sejnowski, "Integration of acoustic and visual speech signals using neural networks," *IEEE Commun. Mag.*, Nov. 1989.

[7] A. Waibel, T. Hanazawa, G. E. Hinton, K. Shikano and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37 (3), March 1989, pp. 328-339.

[8] D. W. Massaro and M. M. Cohen, "Evaluation and integration of visual and auditory information in speech perception." *J. Experimental Psychs. Human Perception and performance.* 9:753-771, 1983.

[9] D. W. Massaro, "Speech perception by ear and eye." in B. Dodd and R. Campbell. eds., *Hearing by Eye: the psychology of Lip-reading*. pp. 53-83. Lawrence Erlbaum Associates, London, 1987.

[10] D. W. Massaro. *Speech Perception by Ear and Eye: a Paradigm for Psychological Inquiry*. Lawrence Erllbaum Associates. Hillsdale. NJ. 1987.