MAJOR PROJECT

on

ADVANCED FEATURE EXTRACTION AND ITS IMPLEMENTATION IN SPEECH RECOGNITION SYSTEM

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF DEGREE
Of

MASTER OF ENGINEERING
(Computer Technology and Applications)
Delhi University, Delhi

Submitted by:

RACHNA JAIN
University Roll No 8551

Under the Guidance of:

Dr. S.K SAXENA

Department of Computer Engineering
Delhi College of Engineering, Delhi

DEPARTMENT OF COMPUTER ENGINEERING
DELHI COLLEGE OF ENGINEERING
BAWANA ROAD, DELHI-110042
DELHI UNIVERSITY
2011

Certificate

This is to certify that the major project entitled **"ADVANCED FEATURE EXTRACTION AND ITS IMPLEMENTATION IN SPEECH RECOGNITION SYSTEM"** is the work of **Rachna Jain (**Univ. Roll No. 8551), a student of Delhi College of Engineering. This work was completed under my direct supervision and guidance and forms a part of Master of Engineering (Computer Technology & Applications) course and curriculum.

(Dr. S. K. SAXENA)
Project Guide
Department of Computer Engineering
Delhi College of Engineering
(Now Delhi Technological University)

Acknowledgement

It gives me a great pleasure to express my profound gratitude to my project guide **Dr. S. K. SAXENA,** Senior Faculty, Department of Computer Engineering, Delhi College of Engineering, for his valuable and inspiring guidance throughout the progress of this project.

At the same time, I would like to extend my heart felt thanks to **Dr.(Mrs.) Daya Gupta,** Head of the department, Department of Computer Engineering, Delhi College of Engineering, for keeping the spirits high and clearing the visions to work on the project.

Rachna Jain
Roll No. 8551
(12/CTA/09)

ABSTRACT

The project entitled "ADVANCED FEATURE EXTRACTION AND ITS IMPLEMENTATION IN SPEECH RECOGNITION SYSTEM" deals with the in depth study of Feature Extraction module. It encompasses various feature extraction algorithms which would be implemented and optimized. Different parameters would be extracted both in time domain and frequency domain.

Speech Processing is an upcoming field. A lot of research is conducted in this field. Today people are interested in hand free systems. A password can be hacked but using voice as a password can not be hacked. We want to make the use of mouse obsolete. As we speak 'my computer', it should be double clicked or in general the respective application should be opened.

The project is mainly concerned with the advanced feature extraction and implementation of those features in the speech recognition system. Various feature extraction algorithms have been implemented in the project. They are then compared with each other with respect to their performance. Graphs have been plotted to represent their respective performance in terms of their recognition rates.

The project is designed following a simplistic yet an effective approach so that the user has a hassle free working experience. This project includes text fields for the input word and recognized word and the list of various levels of noise which need to be added.

I have used its application in design of voice automated robot.

Voice based robotic control is an interesting project, mainly used for industrial and surveillance applications. It gives exact concept of controlling a robot by a voice instruction.

Robot is capable of understanding and synthesizing human speech for communication. A voice recognition unit built around a high speed processor that ensures various operations of the system to be performed by voice command. A few of commands recommended for operation are listed as: START, STOP, FORWARD, REVERSE, RIGHT, LEFT, SLOW, FAST,OK, UP, DOWN, CLOCK, ANTICLOCK, CLOSE,OPEN.

After the design of voice automated mobile robot I have designed the gesture automated mobile robot. Robot is capable of understanding and synthesizing gesture for communication.

LIST OF FIGURES

Figure 1: The Human Speech Organ ……………………..……………………..……....3

Figure 2(a): Typical impulse sequence..4

Figure 2(b): Variation of the pitch frequency..4

Figure 3: The Human Speech Production..5

Figure 4(a): Enrollment Phase ....……………………..……........................7

Figure 4(b): Identification Phase...……………………..……………………................7

Figure 5: An example of speech signal..9

Figure 6(a):Block diagram of the LPCC...11

Figure 6(b): Block diagram of RAS..12

Figure 6(c): Block diagram of DAS...13

Figure 6(d): Block diagram of AMFCC 14

Figure 6(e): Block diagram of Higher-lag autocorrelation coefficients relation………15

Figure 7: An example of mel-spacefilterbank...20

Figure 8(a): Block diagram of the LPCC …………………..………........................23

Figure 8(b): Block diagram of OSALPC ……………………...………………........24

Figure 8(c)Block diagram of SMC 25

Figure 9: Conceptual diagram illustrating VQ codebook formation............................32

Figure 10: Steps in HMM..36

Figure 11(a): Results with maximum recognition for Higher-Lag Module for white noise..37

Figure 11(b): Graphical representation of Figure 11(a)..37

Figure 11(c): Results with maximum recognition for Higher-Lag Module for F16 noise.......38

Figure 11(d): Graphical representation of Figure 11(c).......38

Figure 11(e): Results with maximum recognition for Higher-Lag Module for Factory noise.......39

Figure 11(f): Graphical representation of Figure 11(e).......39

Figure 11(g): Results with maximum recognition for Higher-Lag Module for Babble noise.......40

Figure 11(h): Graphical representation of Figure 11(g).......40

Figure 11(i): Results with maximum recognition for Higher-Lag Module for white noise.......41

Figure 11(j): Graphical representation of Figure 11(i).......41

Figure 11(k): Results with maximum recognition for Higher-Lag Module for F16 noise.......42

Figure 11(l): Graphical representation of Figure 11(k).......42

Figure 11(m): Results with maximum recognition for Higher-Lag Module for Factory noise.......43

Figure 11(n): Graphical representation of Figure 11(m).......43

Figure 11(o): Results with maximum recognition for Higher-Lag Module for Babble noise.......44

Figure 11(p): Graphical representation of Figure 11(o).......44

Figure 11(q): Results using Segmental SNR with maximum enhancement for AR-HASE than HASE.......45

Figure 11(r): Graphical representation of Figure 10(q)..45

Figure 12: Block Diagram of the System...47

TABLE OF CONTENTS

Certificate…………………………………………………………….……i

Acknowledgement…………………………………………………..…...ii

Abstract……………………………………………………………..…....iii

Chapter 1. Introduction……………..…………………………………1

1.1 Speech Recognition…………………………………....1
1.2 Speech…………………………………………………2
1.2.1The process of speech production
1.2.2Fundamental of the Human speech Production
1.3 Mechanism of Speech Production………………………..5
1.4 Principle of Speech Recognition…………………………6

Chapter 2. Hardware & Software Requirements…….…………….……8

2.1 Hardware…………………………………………….…...8
2.2 Software………………...………………………….……8

Chapter 3. Speech Feature Extraction…………………….……….…...9

3.1 MFCC…………………………………………...…11
3.2 RAS…………………………………………………...12
3.3 DAS…………………………………………………...13
3.4 AMFCC……………………………………………….....14
3.5 Higher Lag Autocorrelation……………………………..15
3.6 Steps Involved……………………………………...…16

Chapter 4. Speech Feature Extraction(Based on LPC coefficients)…22

4.1 LPCC.…………………..………………………………..22
4.2 OSALPC......…………………………………………..23
4.3 SMC…..24
4.4 Steps Involved……………..……………………………..25

Chapter 5. Feature Matching…………………………………………....31

5.1 Codebook………..... ………….……………....................32
5.1.1 Code Book Generation
5.1.2 Code Book Initialization
5.1.3 Code Book Design
5.2 Vector Quantization…….………......……......................33
5.2.1 VQ Algorithm
5.3Design of Codebook………………………….....................36

5.3.1 Vector Quantization coding Outline

5.4HMM……………………..…………………......................36

Chapter 6. Experimental Result…..………..……………………...…...........37

Chapter 7. Application of Speech Recognition System...……....….........46

7.1 Implementation Framework……………..……...................46

7.2 Working of Robot...…………………..……......................47

7.3 Uses......................…..……………………….....................48

Chapter 8. Conclusion....…...………………………………………..........51

Chapter 9. Future Scope……………………………………………..........52

Chapter 10. References...53

Appendix A….......……………….……………………………………........54

Appendix B……………………………………………………………........56

Appendix C……………………………………………………………........71

Appendix D……………………………………………………………........73

Appendix E……………………………………………………………........75

<u>Chapter 1</u>

INTRODUCTION

1.1 SPEECH RECOGNITION

Speech recognition (also known as **automatic speech recognition** or **computer speech recognition**) converts spoken words to machine-readable input. The term "voice recognition" may also be used to refer to speech recognition, but can more precisely refer to speaker recognition, which attempts to identify the person speaking, as opposed to what is being said. Speech recognition applications include voice dialing (e.g., "Call home"), call routing (e.g., "I would like to make a collect call"), demotic appliance control and content-based spoken audio search (e.g., find a pod cast where particular words were spoken), simple data entry (e.g., entering a credit card number), preparation of structured documents (e.g., a radiology report), speech-to-text processing (e.g., word processors or emails), and in aircraft cockpits (usually termed Direct Voice Input). Speech recognition is the process of converting an acoustic signal, captured by a microphone or a telephone, to a set of words. The recognized words can be the final results, as for applications such as commands & control, data entry, and document preparation. Speech recognition systems can be characterized by many parameters. An isolated-word speech recognition system requires that the speaker pause briefly between words, whereas a continuous speech recognition system does not. Spontaneous, or extemporaneously generated, speech contains influences, and is much more difficult to recognize than speech read from script. Some systems require speaker enrollment---a user must provide samples of his or her speech before using them, whereas other systems are said to be speaker-independent, in that no enrollment is necessary. Some of the other parameters depend on the specific task. Recognition is generally more difficult when vocabularies are large or have many similar-sounding words.

1.2 SPEECH

The purpose of speech is communication. Speech can be represented in terms of its message or information. The information that is communicated through speech is intrinsically of a discrete nature, i.e. it can be represented by a concatenation of element from a finite set of symbols. The symbols from which every sound can be classified are called phonemes. Each language has its own distinctive set of phonemes, typical number between 30 and 50.A human produce speech at an average rate of 10phonemes/sec. If a binary number represent each phoneme, then a six-bit numerical code is more than sufficient to represent all of the phonemes of the English. In speech communication system, the speech signal is transmitted, stored and processed in many ways. In general there are two major concerns in any system:

• Preservation of the message content in the speech signal.

• Representation of the speech signal in a form that is convenient for transmission and storage, or in a form that is flexible.

1.2.1 The process of speech production

Speech signal are composed of a sequence of sounds. The arrangement of these sounds (symbol) is governed by the rules of the language. The study of these rules and their implications in human communication in the domain of linguistics, and classification of the sounds of speech is called phonetics.

1.2.2 Fundamentals of the Human Speech Production

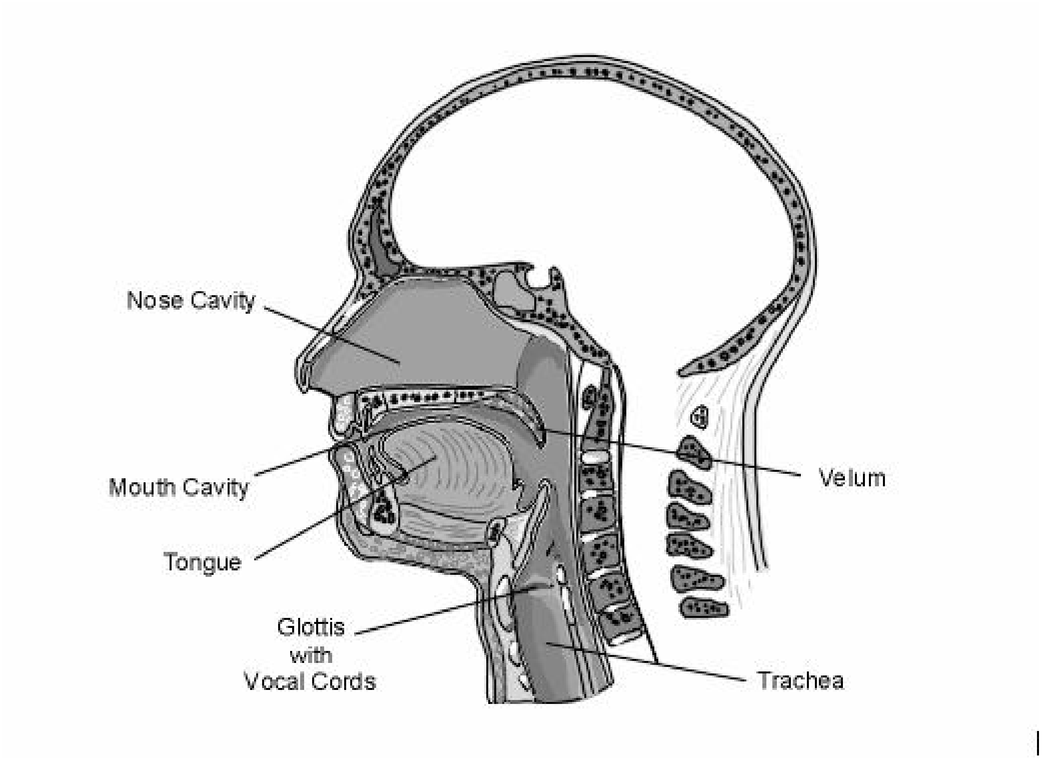


Figure1: The Human Speech Organ

Speech is produced by a cooperation of lungs, glottis (with vocal cords) and articulation tract (mouth and nose cavity). Fig1 shows a cross section of the human speech organ. For the production of voiced sounds, the lungs press air through the epiglottis, the vocal cords vibrate, and they interrupt the air stream and produce a quasi-periodic pressure wave. The pressure impulses are commonly called pitch impulses and the frequency of the pressure signal is the pitch frequency or fundamental frequency.

In Fig2a, a typical impulse sequence (sound pressure function) produced by the vocal cords for a voiced sound is shown. It is the part of the voice signal that defines the speech melody. When we speak with a constant pitch frequency, the speech sounds monotonous but in normal cases a permanent change of the frequency ensues. Fig2b depicted the pitch frequency variation.

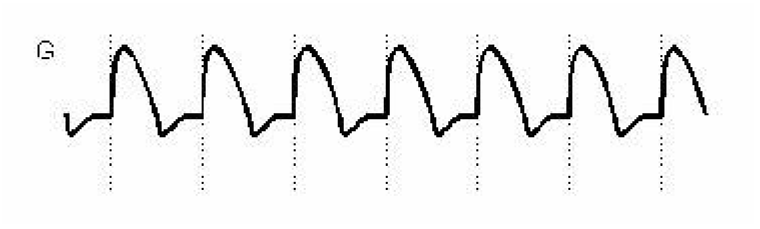


Figure 2a: Typical impulse sequence

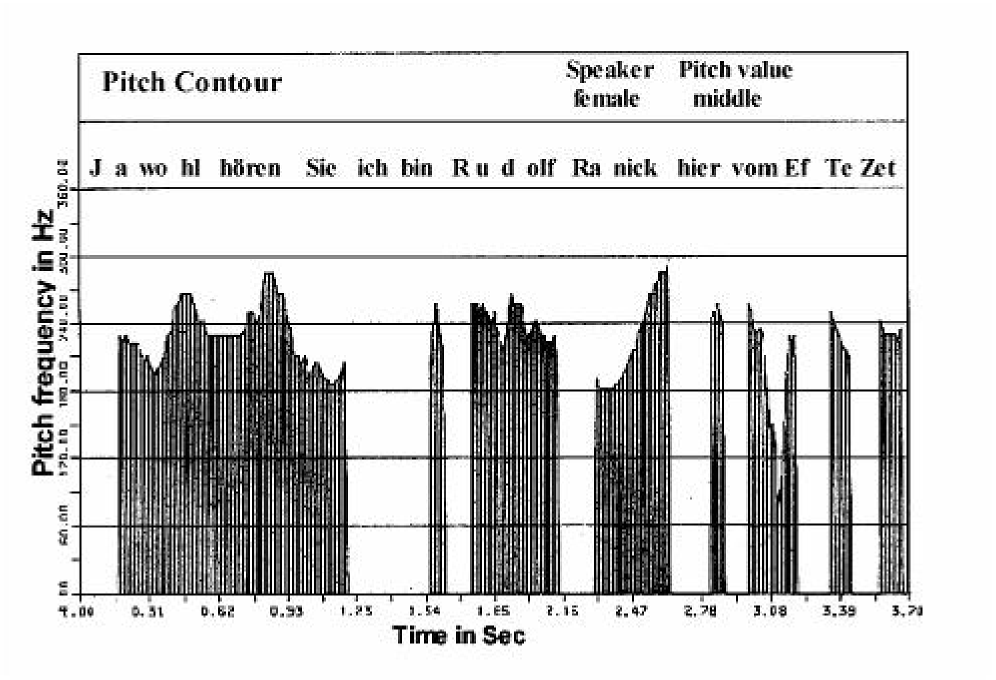


Figure 2b: Variation of the pitch frequency

The pitch impulses stimulate the air in the mouth and for certain sounds (nasals) also stimulate the nasal cavity. When the cavities resonate, they radiate a sound wave which is the speech signal. Both cavities act as resonators with characteristic resonance frequencies, called formant frequencies. Since the mouth cavity can be greatly changed, we are able to pronounce many different sounds. In the case of unvoiced sounds, the excitation of the vocal tract is more noise-like.

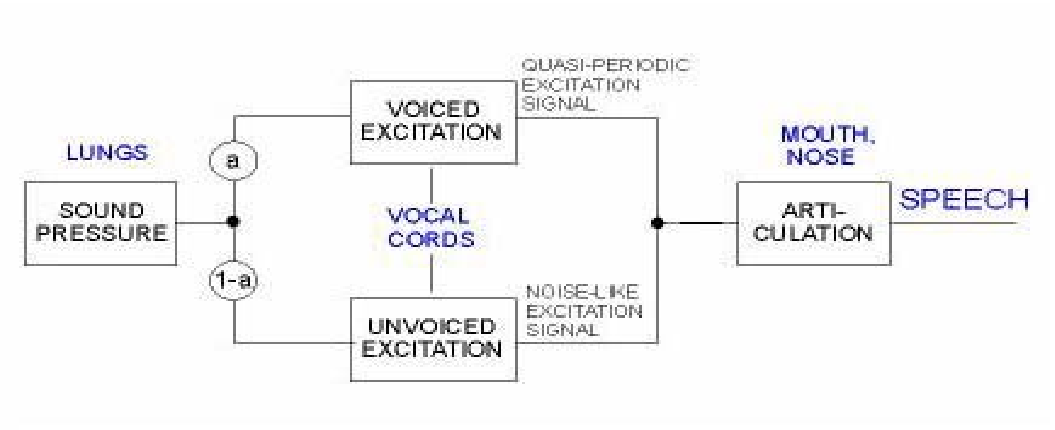


Figure 3: The Human Speech Production

1.3 MECHANISM OF SPEECH PRODUCTION

The human vocal system begins at the opening between the vocal cords, or glottis, and ends at the lips. In the average male, the total length of vocal tract is about 17-cm.the cross section area of the vocal tract is about 20-cm^2. The nasal tract begins at the velum and ends at the nostrils. When the nasal is lowered, the nasal tract is acoustically coupled to the vocal tract to produce the nasal sounds of speech. The sub- glottis system, which is composed of lungs, bronchi, and trachea, serves as a source of energy for the production of speech. Speech is simply the acoustic wave that is radiated from this system when air is expelled from the lungs and the resulting flow of air is perturbed by a constriction somewhere in the vocal tract. Speech sound can be classified into 3 distinct classes according to their mode of excitation. Vocal sounds are produced by forcing air through the glottis with the 1vocal sounds are produced by forcing air through the glottis with the tension of the vocal cords adjusted so that they vibrate in the relaxation oscillation, there by producing quasi-periodic pulses of air which excite the vocal tract. Fricative and unvoiced sounds are generated by forming a constriction in some point in the vocal tract and forcing air through the constriction at a high enough velocity to produce turbulence. Vocal tract, nasal tract is a tube of non-uniform cross-sectional area. As sound propagates down these tubes, the frequency spectrum is shaped by the frequency selectivity of the tube. In the context of speech production, resonance frequencies of the vocal tract tube are called formant frequencies or simply formants. Varying the shape of the vocal tract forms different sounds. The spectrograph produces a two dimensional pattern called a spectrogram in which the vertical dimension corresponds to frequency and the horizontal dimensional to time. The darkness of the pattern is proportional to signal energy.

1.4 PRINCIPLE OF SPEECH RECOGNITION SYSTEM

Speech recognition can be classified into Enrollment Phase (training phase) and Identification Phase (testing phase). **Enrollment phase** is a process of collection of speech samples and then used to train the model. In the training phase, each registered speaker has to provide samples of their speech for different words so that the system can build or train a reference model. **Identification phase** is the process of testing sample of unknown speech which is compared against word database. During the testing phase, the input speech is matched with stored reference model(s) and recognition decision is made.

The basic structures of **Enrollment phase**:

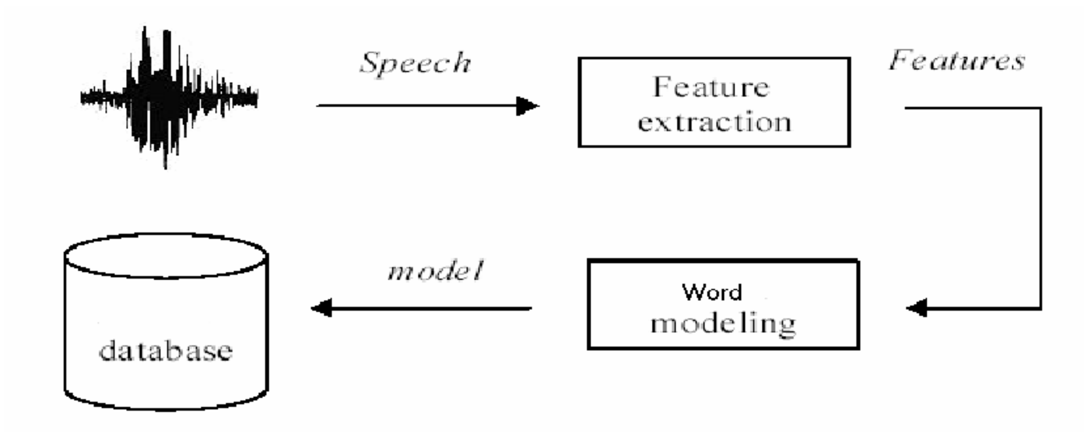


Figure 4(a): Enrollment Phase

The basic structures of **Identification phase**:

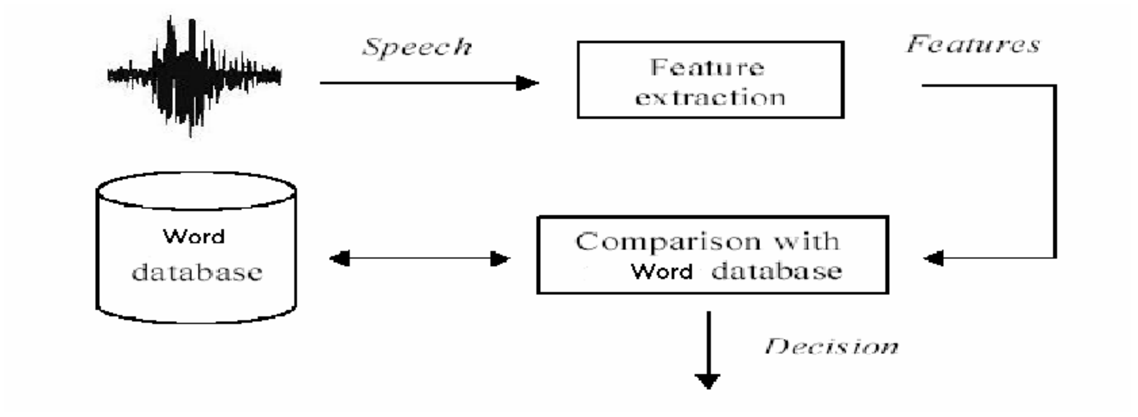


Figure 4(b): Identification Phase

All technologies of speech recognition, enrollment and identification, each has its own advantages and disadvantages and may require different treatments and techniques. The choice of which technology to use is application-specific. All speech recognition systems contain two main modules (as in figure 4) ***feature extraction*** and ***feature matching***.

Chapter 2

HARDWARE AND SOFTWARE REQUIREMENTS

2.1 HARDWARE

- 256 MB RAM (Min.)
- 2.44 GHz Processor
- 10 GB HDD (Min.)

2.2 SOFTWARE

- MATLAB 7.0.1
- MICROSOFT VISUAL STUDIO 2008
- Operating System - Microsoft Window XP

Chapter 3

SPEECH FEATURE EXTRACTION

The purpose of this module is to convert the speech waveform to some type of parametric representation (at a considerably lower information rate) for further analysis and processing. This is often referred as the *signal-processing front end*. The speech signal is a slowly timed varying signal (it is called *quasi-stationary*). An example of speech signal is shown in Figure 5. When examined over a sufficiently short period of time (between 5 and 100 msec), its characteristics are fairly stationary. However, over long periods of time (on the order of 1/5 seconds or more) the signal characteristic change to reflect the different speech sounds being spoken. Therefore, *short-time spectral analysis* is the most common way to characterize the speech signal.

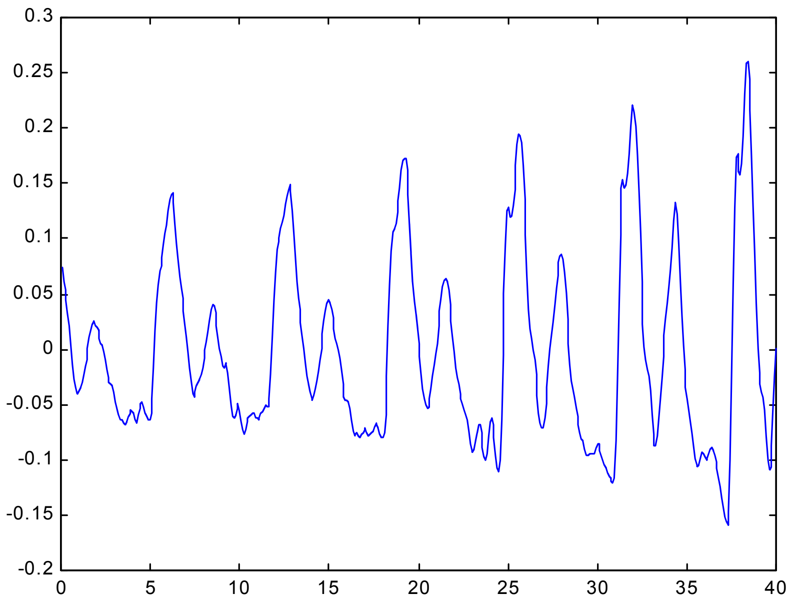


Figure 5: An example of speech signal

Wide range of possibilities exists for parametrically representing the speech signal for the speaker recognition task, such as Linear Prediction Coding (LPC), Mel-Frequency Cepstrum Coefficients (MFCC), and others. MFCC is perhaps the best known and most popular.

In this project, several modules are applied for speech feature extraction. They are:

- MFCC
- RAS
- DAS
- AMFCC
- HIGHER-LAG

MFCC, RAS, DAS, DPS and HIGHER LAG are based on MFCC coefficients.

3.1 MEL-FREQUENCY CEPSTRUM COEFFICIENTS (MFCC)

MFCC's are based on the known variation of the human ear's critical bandwidths with frequency; filters spaced linearly at low frequencies and logarithmically at high frequencies have been used to capture the phonetically important characteristics of speech. This is expressed in the mel-frequency scale, which is linear frequency spacing below 1000 Hz and a logarithmic spacing above 1000 Hz. This sampling frequency was chosen to minimize the effects of aliasing in the analog-to-digital conversion. These sampled signals can capture all frequencies up to 5 kHz, which cover most energy of sounds that are generated by humans. As been discussed previously, the main purpose of the MFCC processor is to mimic the behavior of the human ears. In addition, rather than the speech waveforms themselves, MFCC's are shown to be less susceptible to mentioned variations.

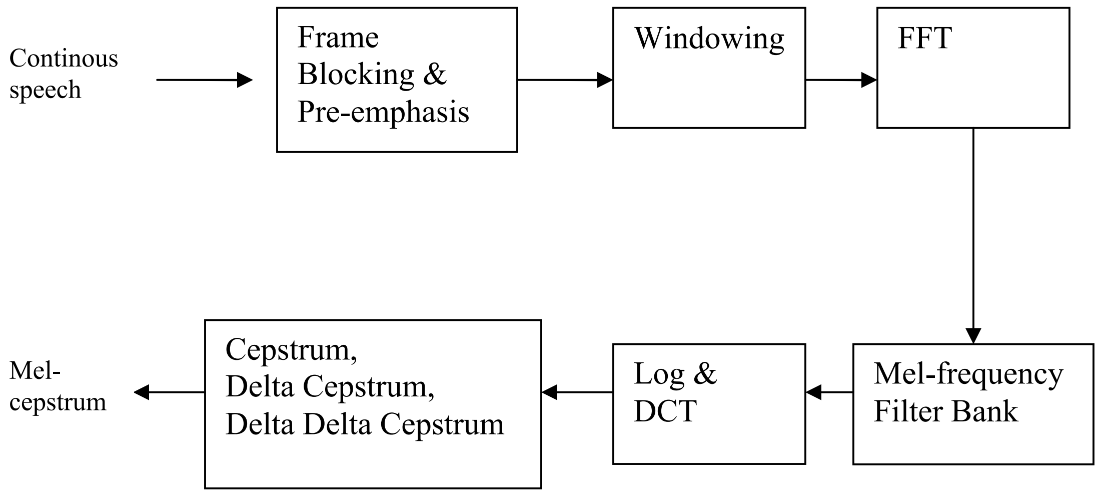

Continous speech
Frame Blocking & Pre-emphasis
Windowing
FFT
Mel-frequency Filter Bank
Log & DCT
Cepstrum, Delta Cepstrum, Delta Delta Cepstrum
Mel-cepstrum

Figure 6(a): Block diagram of the MFCC

3.2 RELATIVE AUTOCORRELATION SEQUENCE (RAS)

In this algorithm, splitting the speech signal into frames and applying a pre-emphasis filter, the autocorrelation sequence of the frame signal is obtained. A FIR filter is then applied to the noisy speech signal autocorrelation sequence. Hamming windowing and short time fourier transform constitute the next stage. A set of cepstral coefficients can then be derived by applying a conventional mel-frequency filter bank to the resultant spectrum and finally passing the logarithm of bin outputs to the DCT block.

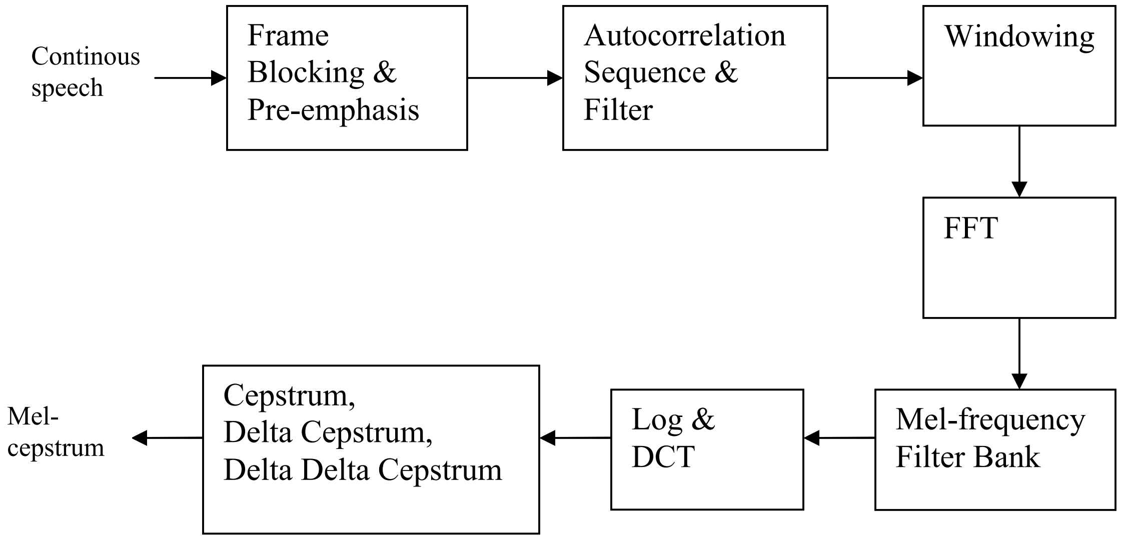

Continous speech
Frame Blocking & Pre-emphasis
Autocorrelation Sequence & Filter
Windowing
FFT
Mel-frequency Filter Bank
Log & DCT
Cepstrum, Delta Cepstrum, Delta Delta Cepstrum
Mel-cepstrum

Figure 6(b): Block diagram of RAS

3.3 DIFFRENTIATED AUTOCORRELATION SEQUENCE (DAS)

This approach combines the advantage of RAS and DPS. In this algorithm, splitting the speech signal into frames and applying a pre-emphasis filter, the autocorrelation sequence of the frame signal is obtained. A FIR filter is then applied to the noisy speech signal autocorrelation sequence. Hamming windowing and short time fourier transform constitute the next stage. Then differential power spectrum of the filtered signal is calculated. By differentiation of the spectrum, we preserve the peaks. Since the spectral peaks convey the most important information in speech signal. A set of cepstral coefficients can then be derived by applying a conventional mel-frequency filter bank to the resultant spectrum and finally passing the logarithm of bin outputs to the DCT block.

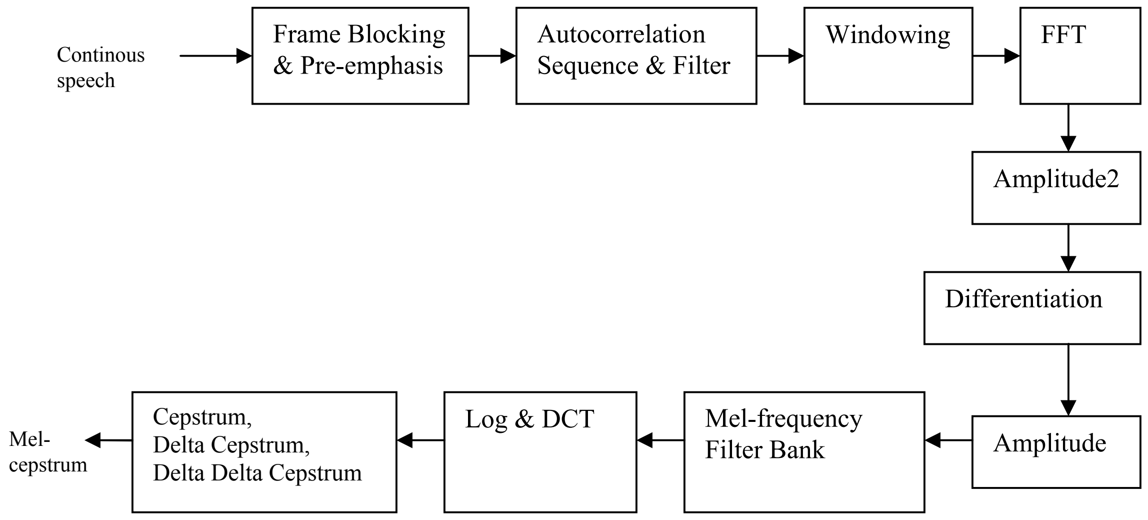

Continous speech
Frame Blocking & Pre-emphasis
Autocorrelation Sequence & Filter
Windowing
FFT
Amplitude2
Differentiation
Amplitude
Mel-frequency Filter Bank
Log & DCT
Cepstrum, Delta Cepstrum, Delta Delta Cepstrum
Mel-cepstrum

Figure 6(c): Block diagram of DAS

3.4 AUTOCORRELATION MEL-FREQUENCY CEPSTRUM COEFFICIENTS (AMFCC)

In this algorithm, splitting the speech signal into frames and applying a pre-emphasis filter. A hamming windowing and autocorrelation constitute the next stage. Then again windowing is done. Then taking short time fast fourier transformation, the absolute value is preserved. A set of cepstral coefficients can then be derived by applying a conventional mel-frequency filter bank to the resultant spectrum and finally passing the logarithm of bin outputs to the DCT block.

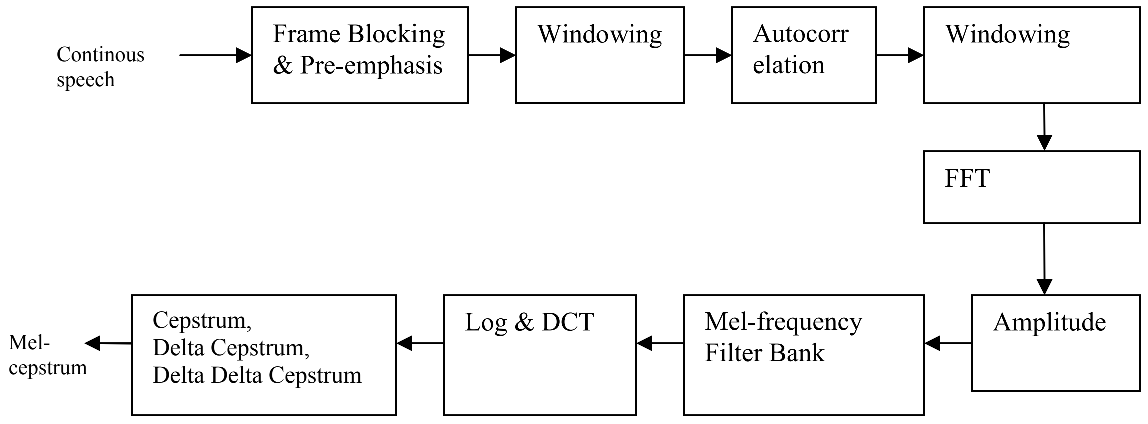

Continous speech
Frame Blocking & Pre-emphasis
Windowing
Autocorr elation
Windowing
FFT
Amplitude
Mel-frequency Filter Bank
Log & DCT
Cepstrum, Delta Cepstrum, Delta Delta Cepstrum
Mel-cepstrum

Figure 6(d): Block diagram of AMFCC

3.5 HIGHER-LAG AUTOCORRELATION COEFFICIENTS

In this algorithm, splitting the speech signal into frames and applying a pre-emphasis filter. Then autocorrelation is applied because the noise autocorrelation sequence is more significant in lower lags. Therefore, noise robust spectral estimation is possible on the higher lag autocorrelation coefficients. Hence eliminating the lower lags of the noisy speech signal autocorrelation should lead to removal of the main noise components. A hamming windowing constitute the next stage. Then taking short time fast fourier transformation, the absolute value is preserved. Then differentiation of the square of absolute is taken to preserve the spectral peaks. A set of cepstral coefficients can then be derived by applying a conventional mel-frequency filter bank to the resultant spectrum and finally passing the logarithm of bin outputs to the DCT block.

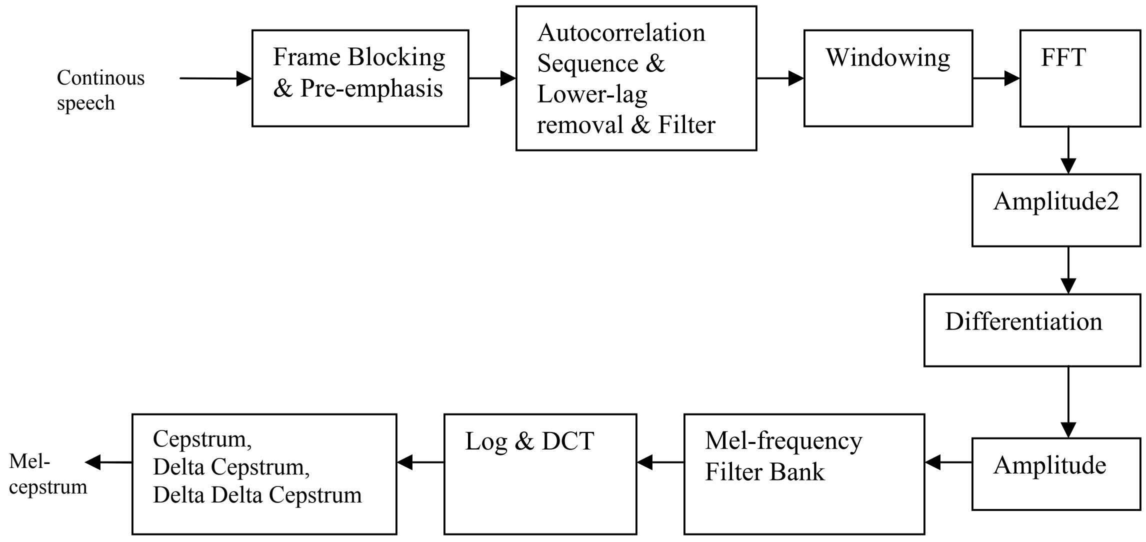

Continous speech
Frame Blocking & Pre-emphasis
Autocorrelation Sequence & Lower-lag removal & Filter
Windowing
FFT
Amplitude2
Differentiation
Amplitude
Mel-frequency Filter Bank
Log & DCT
Cepstrum, Delta Cepstrum, Delta Delta Cepstrum
Mel-cepstrum

Figure 6(e): Block diagram of Higher-lag autocorrelation coefficients

3.6 STEPS INVOLVED

3.6.1 Pre-emphasis

Pre-emphasis s(n) (input signal) using a pre-emphasis filter $H(z) = 1 - a_{pre}z^{-1}$, where a_{pre} is the pre-emphasis coefficient. The value of a_{pre} is given as an experimental parameter.

$X(n) = s(n) - a_{pre}\, s(n-1)$, $n = 0,1,\ldots,N-1$.Frame Blocking

3.6.2 Framing

In this step the continuous speech signal is blocked into frames of *N* samples, with adjacent frames being separated by *M* (*M* < *N*).
The first frame consists of the first *N* samples. The second frame begins *M* samples after the first frame, and overlaps it by *N* - *M* samples.
Similarly, the third frame begins 2*M* samples after the first frame (or *M* samples after the second frame) and overlaps it by *N* - 2*M* samples.
This process continues until all the speech is accounted for within one or more frames. Typical values for *N* and *M* are *N* = 256 (which is equivalent to ~ 30 msec windowing and facilitate the fast radix-2 FFT) and *M* = 100.

3.6.3 Windowing

The next step in the processing is to window each individual frame so as to minimize the signal discontinuities at the beginning and end of each frame. The concept here is to minimize the spectral distortion by using the window to taper the signal to zero at the beginning and end of each frame. If we define the window as $w(n)$, $0 \le n \le N-1$, where *N* is the number of samples in each frame, then the result of windowing is the signal

$$y_l(n) = x_l(n)w(n), \quad 0 \le n \le N-1$$

Typically the *Hamming window* is used, which has the form:

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right), \quad 0 \le n \le N-1$$

3.6.4 Autocorrelation Sequence:

$$ryy(m,k) = \frac{1}{N-K} \sum_{i=0}^{N-1-K} y(m,i)*y(m,i+k)$$

$$0 \le m \le M-1, \quad 0 \le n \le N-1$$

Pole preservation property: Original signal is excited by an impulse train and a white noise, then, the poles of the autocorrelation sequence would be the same as the poles of the original signal.

For many typical noise types, noise autocorrelation sequence is more significant in lower lags. Therefore, Noise-Robust Spectral estimation is possible on the higher lag autocorrelation coefficient.

3.6.5 Filter

High pass filtering of autocorrelation sequence done to reduce noise effect.

3.6.6 Fast Fourier Transform (FFT)

The next processing step is the Fast Fourier Transform, which converts each frame of *N* samples from the time domain into the frequency domain. The FFT is a fast algorithm to implement the Discrete Fourier Transform (DFT) which is defined on the set of N samples $\{x_n\}$, as follow:

$$X_n = \sum_{k=0}^{N-1} x_k e^{-2\pi jkn/N}, \qquad n = 0,1,2,..., N-1$$

Note that we use j here to denote the imaginary unit, i.e. $j = \sqrt{-1}$. In general X_n's are complex numbers. The resulting sequence $\{X_n\}$ is interpreted as follow: the zero frequency corresponds to $n = 0$, positive frequencies $0 < f < F_s/2$ correspond to values $1 \le n \le N/2-1$, while negative frequencies $-F_s/2 < f < 0$ correspond to $N/2+1 \le n \le N-1$. Here, F_s denote the sampling frequency. The result after this step is often referred to as *spectrum* or *periodogram.*

3.6.7 Discrete Fourier Transform (DFT)

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi kn/N}$$

where k=0,1,2……………(N-1).

The *inverse Discrete Fourier Transform* (IDFT) of the sequence x(n)

$$X(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k)\, e^{j2\pi kn/N}$$

where n =0,1,2……………(N-1).

The DFT transform pair is denoted by

$$\{x(n)\} \longleftrightarrow \{X(k)\}$$

Frequency resolution of the DFT

The frequency resolution of the N-point DFT

$f_r = f_s / N$

- The DFT can resolve exactly only the frequencies falling exactly at: k fs/N. There is spectral leakage for components falling between the DFT bins.
- Typically we use an FFT that is as large as we can afford.
- Zero-padding is often using to provide more resolution in the frequency components.
- Zero padding is often combined with tapered windows

The DFT and the FFT Complexity

The N-point DFT requires N ^2 multiplications and N ^2 –1 additions to compute the Discrete frequency spectrum. The complexity of the DFT is reduced using the FFT to N/2 logN (Base 2) multiplications and N logN (Base2) additions. For example If N=4096 the DFT requires 16,777,216 multiplications while the FFT requires 49,152 multiplications.

3.6.8 Mel-frequency wrapping

As mentioned above, psychophysical studies have shown that human perception of the frequency contents of sounds for speech signals does not follow a linear scale. Thus for each tone with an actual frequency, *f*, measured in Hz, a subjective pitch is measured on a scale called the 'mel' scale. The *mel-frequency* scale is linear frequency spacing below 1000 Hz and a logarithmic spacing above 1000 Hz. As a reference point, the pitch of a 1 kHz tone, 40 dB above the perceptual hearing threshold, is defined as 1000 mels. Therefore we can use the following approximate formula to compute the mels for a given frequency *f* in Hz:

$$mel(f) = 2595 * \log_{10}(1 + f / 700)$$

One approach to simulating the subjective spectrum is to use a filter bank, spaced uniformly on the mel scale (see Figure 4). That filter bank has a triangular bandpass frequency response, and the spacing as well as the bandwidth is determined by a constant mel frequency interval. The modified spectrum of $S(\omega)$ thus consists of the output power of these filters when $S(\omega)$ is the input. The number of mel spectrum coefficients, K, is typically chosen as 20.

Note that this filter bank is applied in the frequency domain; therefore it simply amounts to taking those triangle-shape windows in the Figure 4 on the spectrum. A useful way of thinking about this mel-wrapping filter bank is to view each filter as a histogram bin (where bins have overlap) in the frequency domain.

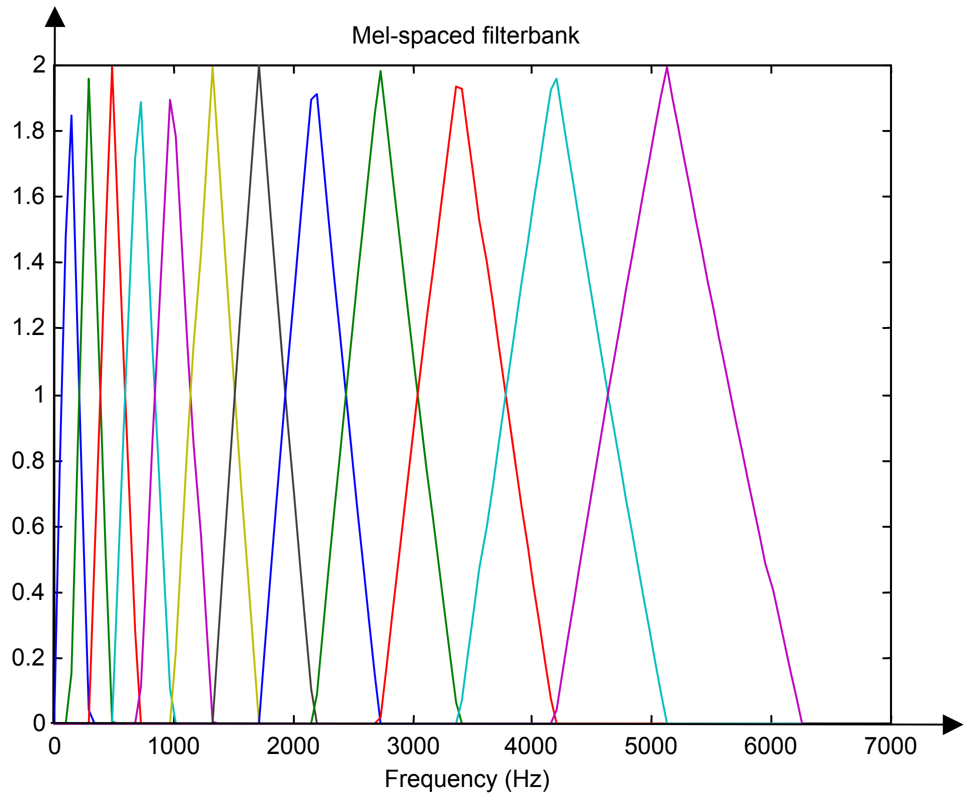


Figure 7: An example of mel-spaced filterbank

Another reason to use a non-linear frequency analysis is to get around the frequency/time resolution tradeoff. Using a narrow bandwidth at low frequencies enables harmonics to be resolved but gives poor onset information. Using a larger bandwidth at higher frequencies allows for high temporal resolution of bursts, etc.

3.6.9 Cepstrum

In this final step, we convert the log mel spectrum back to time. The result is called the mel frequency cepstrum coefficients (MFCC). The cepstral representation of the speech spectrum provides a good representation of the local spectral properties of the signal for the given frame analysis. Because the mel spectrum coefficients (and so their logarithm) are real numbers, we can convert them to the time domain using the Discrete Cosine Transform (DCT). Therefore if we denote those mel power spectrum coefficients that are the result of the last step are $\tilde{S}_k$, $k = 1,2,...,K$, we can calculate the MFCC's, $\tilde{c}_n$, as

$$\tilde{c}_n = \sum_{k=1}^{K} (\log \tilde{S}_k) \cos\left[n\left(k - \frac{1}{2} \right) \frac{\pi}{K} \right], \qquad n = 1,2,...,K$$

Note that we exclude the first component, $\tilde{c}_0$, from the DCT since it represents the mean value of the input signal which carried little speaker specific information.

Chapter 4

SPEECH FEATURE EXTRACTION (BASED ON LPC COEFFICIENTS)

Wide range of possibilities exists for parametrically representing the speech signal for the speaker recognition task, such as Linear Prediction Coding (LPC), Mel-Frequency Cepstrum Coefficients (MFCC), and others. MFCC is perhaps the best known and most popular. In this project, several modules are applied for speech feature extraction:

- LPCC
- OSALPC
- SMC

LPCC, OSALPC and SMC are based on LPC coefficients [1, 2, 3].

4.1 LINEAR PREDICTION CEPSTRAL COEFFICIENTS (LPCC)

LPC features are generated in accordance with the vocal cord or human vocal tract. LPCC algorithm [1, 2, 3] is based on LPC features. They model the speech as a linear but time varying system. Speech samples from previous time points are combined linearly to predict the current value. LPCC performs better in case of loud signals. In this algorithm, splitting the speech signal into frames and applying a pre-emphasis filter, then autocorrelation is applied followed by windowing to remove discontinuities. A set of cepstral coefficients can then be derived by applying Levinson Durbin Recursion Algorithm. Then the final result is the LPCC extracted features.

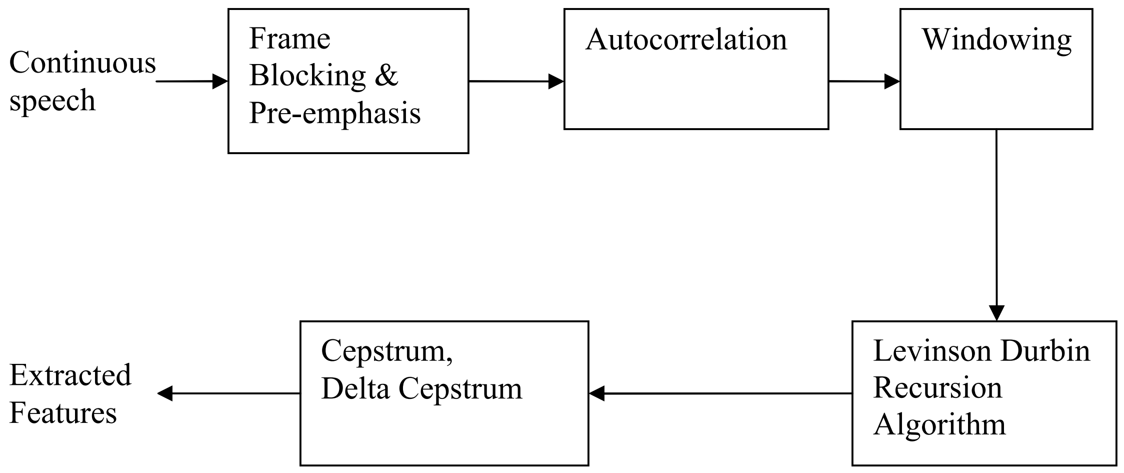


Figure 8(a): Block diagram of the LPCC

4.2 ONE-SIDED AUTOCORRELATION LINEAR PREDICTION CEPSTRALS (OSALPC)

LPC features are generated in accordance with the vocal cord or human vocal tract. OSALPC algorithm [1, 2] is based on LPC features. OSALPC performs better in case of loud as well as noisy signals. This technique performs only a partial deconvolution of the speech signal. In spite of tha, OSALPC shows better speech recognition performance than conventional LPCC in severe conditions of additive noise. In this algorithm, splitting the speech signal into frames and applying a pre-emphasis filter, then autocorrelation is applied followed by windowing to remove discontinuities. Then, autocorrelation of order 12 is applied. A set of cepstral coefficients can then be derived by applying Levinson Durbin Recursion Algorithm. Then the final result is the OSALPC extracted features.

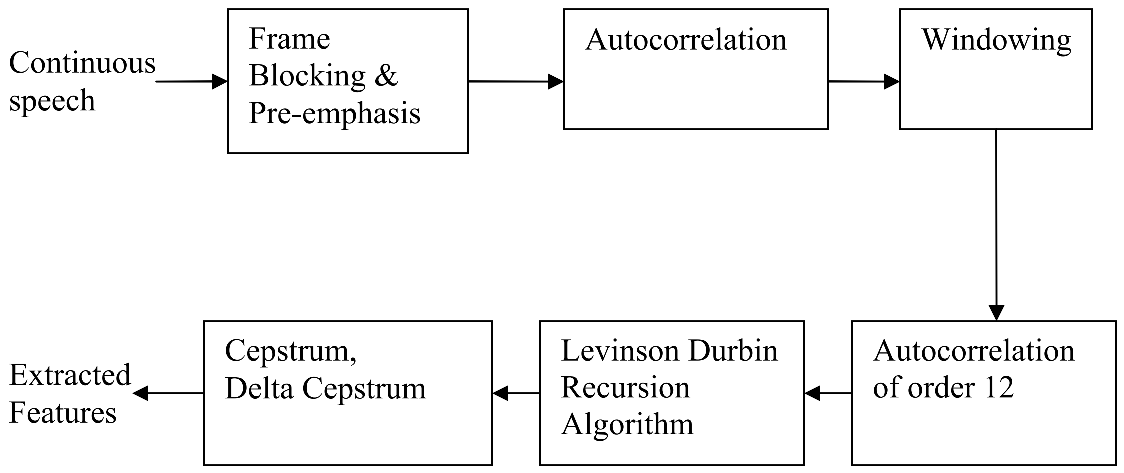

Continuous speech
Frame Blocking & Pre-emphasis
Autocorrelation
Windowing
Autocorrelation of order 12
Levinson Durbin Recursion Algorithm
Cepstrum, Delta Cepstrum
Extracted Features

Figure 8(b): Block diagram of OSALPC

4.3 SHORT-TIME MODIFIED COHERENCE (SMC)

LPC features are generated in accordance with the vocal cord or human vocal tract. SMC algorithm [1, 2] is based on LPC features. SMC performs better in case of loud signals. They are more robust to additive white noise hence performs better than conventional LPCC in severe conditions of additive white noise. In this algorithm, splitting the speech signal into frames and applying a pre-emphasis filter, then autocorrelation is applied followed by windowing to remove discontinuities. Then FFT is applied to convert time domain to frequency domain followed by computing the absolute and the IFFT to convert back to time domain. A set of cepstral coefficients can then be derived by applying Levinson Durbin Recursion Algorithm. Then the final result is the SMC extracted features.

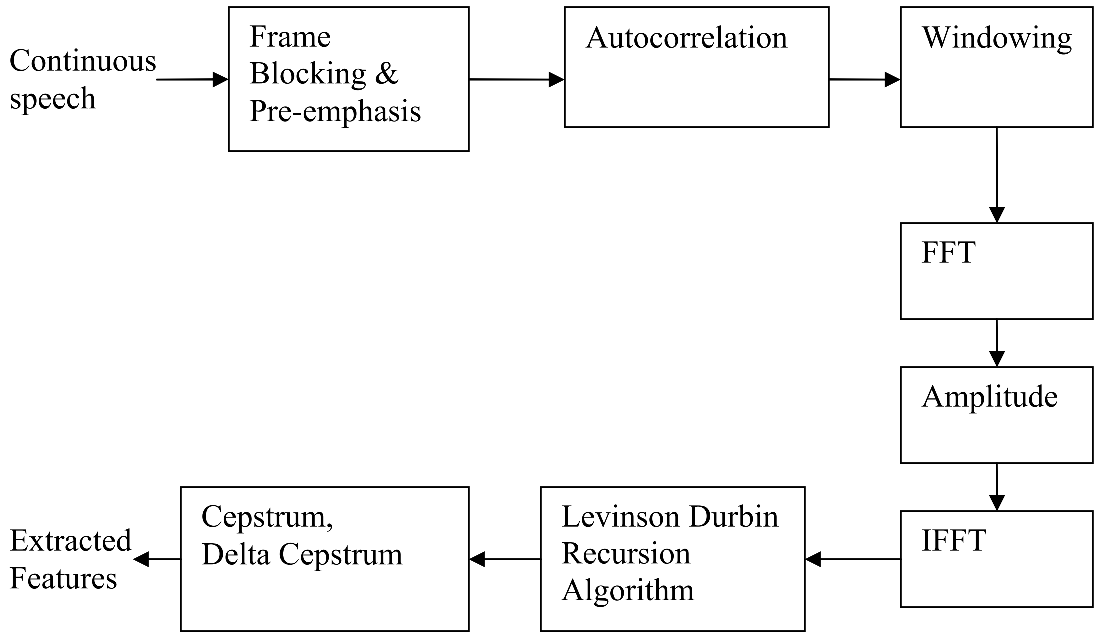


Figure 8(c): Block diagram of SMC

4.4 STEPS INVOLVED

4.4.1 Pre-emphasis

Pre-emphasis s(n) (input signal) using a pre-emphasis filter $H(z) = 1 - a_{pre}(z\text{-}1)$, where a_{pre} is the pre-emphasis coefficient [1, 2]. The value of a_{pre} is given as an experimental parameter.

$$X(n) = s(n) - a_{pre}\, s(n-1), \qquad n = 0,1,\ldots..,N-1 \qquad (1)$$

4.4.2 Framing

In this step the continuous speech signal is blocked into frames of *N* samples, with adjacent frames being separated by *M* (*M* < *N*).
The first frame consists of the first *N* samples. The second frame begins *M* samples after the first frame, and overlaps it by *N* - *M* samples.

Similarly, the third frame begins 2*M* samples after the first frame (or *M* samples after the second frame) and overlaps it by *N* - 2*M* samples.

This process continues until all the speech is accounted for within one or more frames. Typical values for *N* and *M* are *N* = 256 (which is equivalent to ~ 30 msec windowing and facilitate the fast radix-2 FFT) and *M* = 100 [1, 2].

4.4.3 Windowing

The next step in the processing is to window [1, 2] each individual frame so as to minimize the signal discontinuities at the beginning and end of each frame.

The concept here is to minimize the spectral distortion by using the window to taper the signal to zero at the beginning and end of each frame. If we define the window as $w(n)$, $0 \le n \le N-1$, where *N* is the number of samples in each frame, then the result of windowing is the signal.

$$y_l(n) = x_l(n)w(n), \quad 0 \le n \le N-1 \qquad (2)$$

Typically the *Hamming window* is used, which has the form:

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right), \quad 0 \le n \le N-1 \qquad (3)$$

4.4.4 Autocorrelation Sequence:

Pole preservation property [1]: Original signal is excited by an impulse train and a white noise, then, the poles of the autocorrelation sequence would be the same as the poles of the original signal.

For many typical noise types, noise autocorrelation sequence is more significant in lower lags. Therefore, Noise-Robust Spectral estimation is possible on the higher lag autocorrelation coefficient.

$$r_{yy}(m,k) = \frac{1}{N-K}\sum_{i=0}^{N-1-K} y(m,i) * y(m,i+k), \quad 0 \le \mathrm{m} \le \mathrm{M\text{-}1},\ 0 \le \mathrm{n} \le \mathrm{N\text{-}1} \qquad (4)$$

4.4.5 Fast Fourier Transform (FFT)

The next processing step is the Fast Fourier Transform, which converts each frame of N samples from the time domain into the frequency domain. The FFT is a fast algorithm to implement the Discrete Fourier Transform (DFT) which is defined on the set of N samples $\{x_n\}$, as follow:

$$X_n = \sum_{k=0}^{N-1} x_k e^{-2\pi jkn/N}, \qquad n = 0,1,2,..., N-1 \tag{5}$$

Note that we use j here to denote the imaginary unit, i.e. $j = \sqrt{-1}$. In general X_n's are complex numbers. The resulting sequence $\{X_n\}$ is interpreted as follow: the zero frequency corresponds to $n = 0$, positive frequencies $0 < f < F_s/2$ correspond to values $1 \le n \le N/2-1$, while negative frequencies $-F_s/2 < f < 0$ correspond to $N/2+1 \le n \le N-1$. Here, F_s denote the sampling frequency. The result after this step is often referred to as *spectrum* or *periodogram*.

The FFT Complexity

The N-point DFT requires N ^2 multiplications and N ^2 –1 additions to compute the Discrete frequency spectrum. The complexity of the DFT is reduced using the FFT to N/2 logN (Base 2) multiplications and N logN (Base2) additions. For example If N=4096 the DFT requires 16,777,216 multiplications while the FFT requires 49,152 multiplications.

4.4.6 Levinson Durbin Recursion Algorithm

Levinson Durbin Recursion Algorithm [1] is the core of the LPCC, OSALPC and SMC modules. This is an algorithm for finding an all-pole IIR filter with a prescribe deterministic autocorrelation sequence. It has applications in filter design, coding and spectral estimation. This algorithm mainly consists of four steps:

4.4.6.1 LPC Analysis

It converts each frame of p+1 autocorrelations into an "LPC parameter set" in which the set might be the LPC coefficients, PARCOR coefficients, the cepstral coefficients or any desired transformation. The formal method for converting from autocorrelation coefficients to an LPC parameter set is known as Durban's method.

$$E^{(0)} = r(0) \tag{6}$$

$$k_i = \left\{ r(i) - \sum_{j=1}^{L-1} \alpha_j^{i-1} r(|i-j|) \right\} \Big/ E^{(i-1)}, \qquad 1 \le i \le p \tag{7}$$

$$\alpha_i^{(i)} = k_i \tag{8}$$

$$\alpha_j^{i} = \alpha_j^{(i-1)} - k_i \alpha_{i-j}^{(i-1)} \tag{9}$$

$$E^{(i)} = \left(1 - k_i^2\right) E^{(i-1)} \tag{10}$$

where,

a_m= LPC coefficients,

k_m=PARCOR coefficients

g_m=log area ratio coefficients

4.4.6.2 LPC Parameter Conversion to Cepstral Coefficients

A very important LPC parameter set, which can be derived directly from the LPC coefficient set, is the LPC cepstral coefficients, c(m). The recursion used is

$$c_0 = \ln \sigma^2 \tag{11}$$

$$c_m = a_m + \sum_{k=1}^{m-1} \left(\frac{k}{m}\right) c_k a_{m-k}, \qquad 1 \le m \le p \tag{12}$$

$$c_m = \sum_{k=1}^{m-1} \left(\frac{k}{m}\right) c_k a_{m-k}, \qquad m > p \tag{13}$$

where, σ^2 is the gain term in LPC model.

4.4.6.3 Parameter Weighing

Because of the sensitivity of the low-order cepstral coefficients to overall spectral slope and the sensitivity of the higher-order cepstral coefficients to noise, it has become a standard technique to weight the cepstral coefficients by a tapered window so as to minimize these sensitivities.

$$w_m = 1 + \frac{Q}{2}\sin\left(\frac{\Pi m}{Q}\right), \qquad 1 \le m \le Q \tag{14}$$

This weighting function truncates the computation and de-emphasizes c_m around m=1 and m=Q.

4.4.6.4 Temporal Cepstral Derivative

An improved representation of the speech spectrum can be obtained by extending the analysis to include information about the temporal cepstral derivative (both first and second derivatives have been investigated and found to improve the performance of speech recognition system).

$$\frac{\partial c_m(t)}{\partial t} = \Delta c_m(t) \approx \mu \sum_{k=-K}^{K} k c_m(t+k) \tag{15}$$

where, μ is an appropriate normalization constant and (2k+1) is the number of frames over which computation is performed.

4.4.7 Cepstrum

The cepstral representation of the speech spectrum provides a good representation of the local spectral properties of the signal for the given frame analysis.

4.4.8 Segmental SNR

Segmental SNR [8] is used for comparing the performance of HASE and AR-HASE modules. In this process, the speech signal is divided into large number of segments and SNR is calculated for each segment. Then the average SNR is computed for the whole input speech, called the Segmental SNR for that speech signal.
For each segment, SNR is calculated as,

$$SS(k) = \log\left(1 + \frac{SUM[xa(i)]^2}{0.01 + SUM[xa(i) - xb(i)]^2}\right) \tag{16}$$

Then the final average Segmental SNR is calculated as,

$$SSNR = 10 * \log\left(10^{[SUMSS(k)/N]-1}\right) \tag{17}$$

Chapter 5

FEATURE MATCHING

The objects of interest are generically called *patterns* and in our case are sequences of acoustic vectors that are extracted from an input speech using the techniques described in the previous section. The classes here refer to individual speakers. Since the classification procedure in our case is applied on extracted features, it can be also referred to as *feature matching*. Furthermore, if there exists some set of patterns that the individual classes of which are already known, then one has a problem in *supervised pattern recognition*. This is exactly our case since during the training session, we label each input speech with the ID of the speaker (S1 to S8). These patterns comprise the *training set* and are used to derive a classification algorithm. The remaining patterns are then used to test the classification algorithm; these patterns are collectively referred to as the *test set*. If the correct classes of the individual patterns in the test set are also known, then one can evaluate the performance of the algorithm. The state-of-the-art in feature matching techniques used in speech recognition includes Dynamic Time Warping (DTW), Hidden Markov Modeling (HMM), and Vector Quantization (VQ). In this project, the VQ approach will be used, due to ease of implementation and high accuracy. VQ is a process of mapping vectors from a large vector space to a finite number of regions in that space. Each region is called a *cluster* and can be represented by its center called a *codeword*. The collection of all codewords is called a *codebook*. Figure below shows a conceptual diagram to illustrate this recognition process. In the figure, only two speakers and two dimensions of the acoustic space are shown. The circles refer to the acoustic vectors from the speaker 1 while the triangles are from the speaker 2. In the training phase, a speech-specific VQ codebook is generated for each known speaker by clustering his/her training acoustic vectors. The result codewords (centroids) are shown in Figure 8 by black circles and black triangles for speaker 1 and 2, respectively. The distance from a vector to the closest codeword of a codebook is called a VQ-distortion. In the recognition phase, an input utterance of an unknown voice is “vector-quantized” using each trained codebook and the *total VQ distortion* is computed. The speech corresponding to the VQ codebook with smallest total distortion is identified.

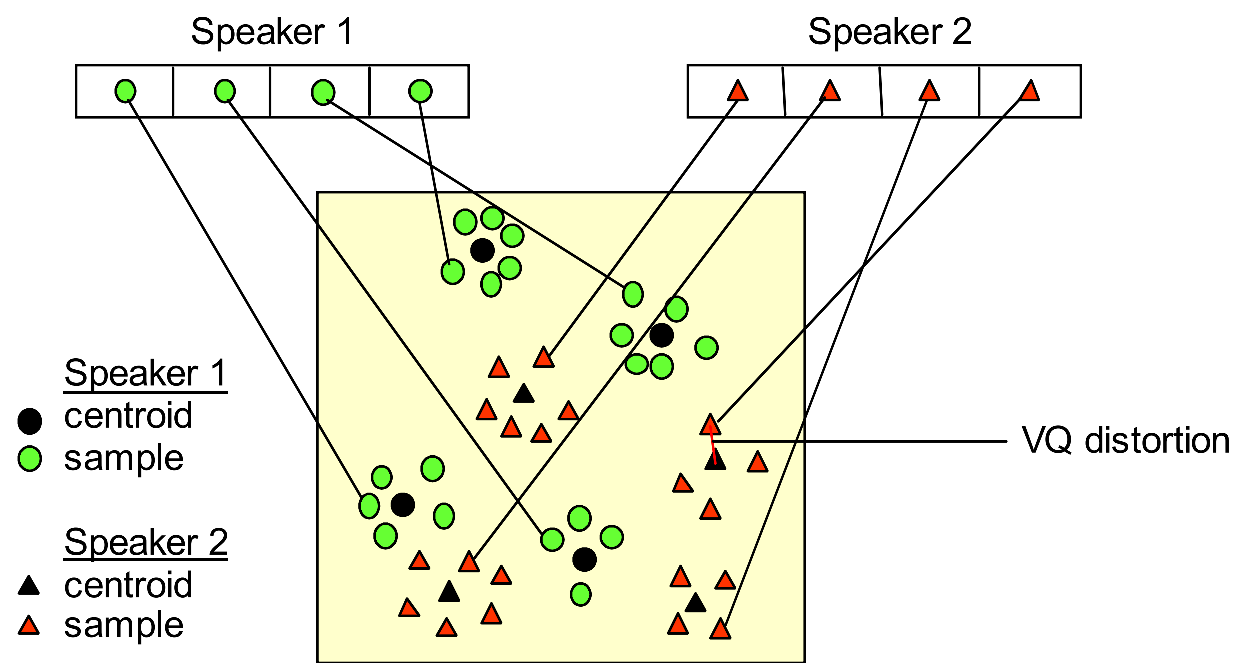


Figure 9: Conceptual diagram illustrating VQ codebook formation.

5.1 CODEBOOK

5.1.1 Codebook Generation

- Generate codebook from a Training set
- Training Set: Set of vectors derived from image vectors
- Code vectors should minimize distortion

5.1.2 Codebook Initialization

Three basic schemes:

- Random
- Perturb and Split (Bottoming)
- Pair wise Nearest Neighbor (PNN) clustering (PNN

5.1.3 Codebook Design

- Basic objective: Minimize search time for code vector
- Full (Exhaustive) Search: very expensive
- Design emphasis: Organization of codebook

5.2 VECTOR QUANTIZATION

Vector quantization (VQ) is a data compression technique, with several successful applications in speech and image coding or speech recognition. Vector quantization is a classical quantization technique from signal processing which allows the modeling of probability density functions by the distribution of prototype vectors. It was originally used for data compression. It works by dividing a large set of points (vectors) into groups having approximately the same number of points closest to them. Each group is represented by its centroid point, as in k-means and some other clustering algorithms. The density matching property of vector quantization is powerful, especially for identifying the density of large and high-dimensioned data. Since data points are represented by the index of their closest centroid, commonly occurring data have low error, and rare data high error. This is why VQ is suitable for lossy data compression. It can also be used for lossy data correction and density estimation. A simple training algorithm for vector quantization is:

Pick a sample point at random

1. Move the nearest quantization vector centroid towards this sample point, by a small fraction of the distance
2. Repeat

A more sophisticated algorithm reduces the bias in the density matching estimation, and ensures that all points are used, by including an extra sensitivity parameter:

1. Increase each centroid's sensitivity by a small amount
2. Pick a sample point at random
3. Find the quantization vector centroid with the smallest <distance-sensitivity>
    1. Move the chosen centroid toward the sample point by a small fraction of the distance
    2. Set the chosen centroid's sensitivity to zero
4. Repeat

There are two categories of approaches in speech verification. In the first one verification decisions are based on speech selected by the speaker and not previously known by the verification system. Within the second approach, the verification system is trained on a particular utterance and the same utterance is letter spoken by the speaker who claims that identity verification using vector quantization (VQ). A typical approach to speech verification is DTW (Dynamic Time Warping) which consists of selecting parameters that can be derived from the speech waveform and then representing each speaker by a time series of these parameters (reference template) obtained from a particular utterance. The parameters are chosen to reflect speaker specific organic differences in the structure of vocal apparatus or to reflect specific learned differences in the use of vocal apparatus. After obtaining the set of references for each speaker to be verified, an unknown speaker claims an identity and speaks an appropriate utterance. This utterance is analyzed and a time series of parameters is obtained. The unknown speaker's parameters are then time aligned to the reference stored for the speaker whose identity was claimed and the decision to accept or reject is based on a measure of similarity between this two time series of parameters.

The parameters employed by this technique, might be: the pitch, the short time energy, the short time spectrum and coefficients. In addition to the template matching method described above, statistical methods are sometimes used. These methods require large amounts of training data to estimate the probability densities of the parameters chosen to represent the speaker.

Use dependency between N consecutive samples to break-up an N dimensional space in cells in a more efficient way than with scalar quantization

• Signal to be quantized is considered as a series of vectors x, containing N samples.

• y the ith vector-of-quantized amplitudes.

5.2.1 VQ algorithm

A codebook may be small in the beginning and may be gradually expanded to the final size. One method is to split an existing cluster in two smaller clusters and assign a codebook entry to each. The following steps describe this method of designing the codebook: create an initial cluster consisting of the entire training set; this initial codebook contains a single centroid for the entire set; split this cluster in two sub clusters, getting a codebook of twice the size; repeat this cluster-splitting process until the codebook reaches the desired size, ideally each cluster should be divided by a hyper plan normal to the direction of maximum distortion.

5.3 DESIGN OF CODEBOOK

5.3.1 Vector Quantization coding Outline

- Divide data (signal) into non-overlapping vectors
- Each vector contains ‘n’ elements (pixels/samples))
- For each speech vector:
  - ➢ Find closest vector in codebook
  - ➢ Get its index in codebook
  - ➢ Encode indices

    -Expression similar to scalar quantizer design

    -We do not know p(x)

    -Cannot be optimized analytically

- Instead of p(x), a set of representative examples or trainings vectors is used to design the codebook in an iterative optimization procedure.

5.4 HIDDEN MARKOV MODEL (HMM)

A **Hidden Markov Model (HMM**) is a statistical model in which the system being modeled is assumed to be a Markov process with unknown parameters, and the challenge is to determine the hidden parameters from the observable parameters. The extracted model parameters can then be used to perform further analysis, for example for patter recognition applications. An HMM can be considered as the simplest dynamic Bayesia network. In a regular Markov model, the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a *hidden* Markov model, the state is not directly visible, but variables influenced by the state are visible. Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states. Hidden Markov models are especially known for their application in temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges and bioinformatics.

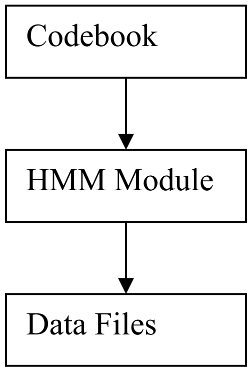


Figure 10: Steps in HMM

Chapter 6

EXPERIMENTAL RESULTS

Module\Noise	0dB	10dB	15dB	20dB	40dB
MFCC	24.1379	37.9310	58.6207	68.9655	86.2069
AMFCC	25.8621	55.1724	72.4138	77.5862	89.6552
RAS	37.9310	72.4138	87.9310	91.3793	91.3793
DAS	36.2069	75.8621	91.3793	93.1034	96.5517
HLAG	46.5517	87.9310	96.5517	98.2759	100.0000

Figure 11(a): Results with maximum recognition for Higher-Lag Module for white noise

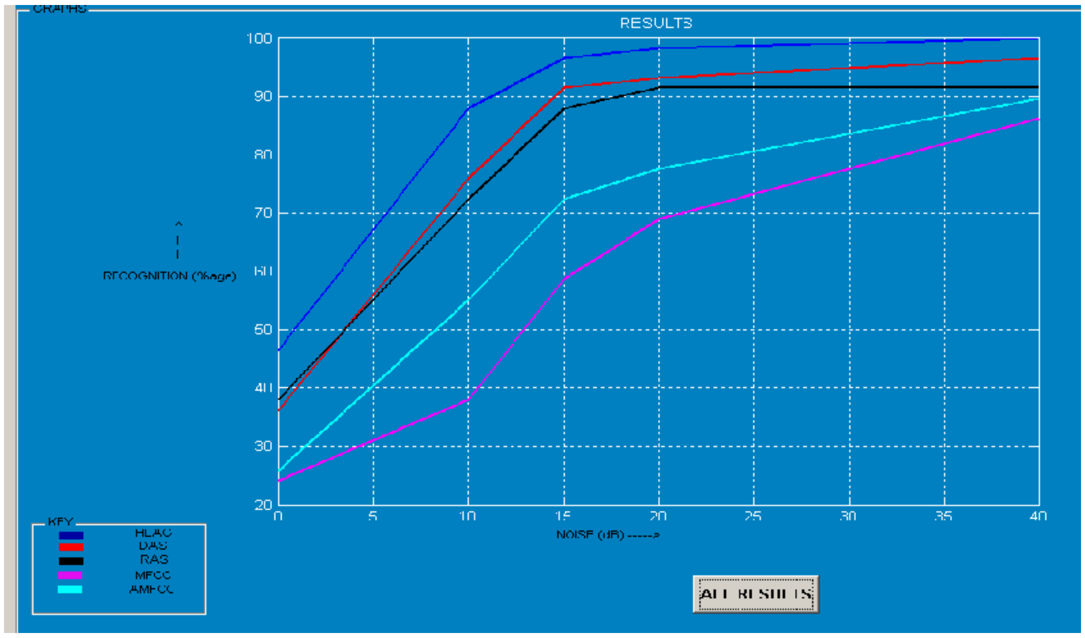


Figure 11(b): Graphical representation of Figure 11(a)

Module\Noise	0dB	10dB	15dB	20dB	40dB
MFCC	25.8621	43.1034	60.3448	77.5862	86.2069
AMFCC	32.7586	62.0690	77.5862	81.0345	87.9310
RAS	20.6897	50.0000	77.5862	94.8276	93.1034
DAS	29.3103	74.1379	84.4828	91.3793	96.5517
HLAG	29.3103	84.4828	94.8276	94.8276	100.0000

Figure 11(c): Results with maximum recognition for Higher-Lag Module for F16 noise

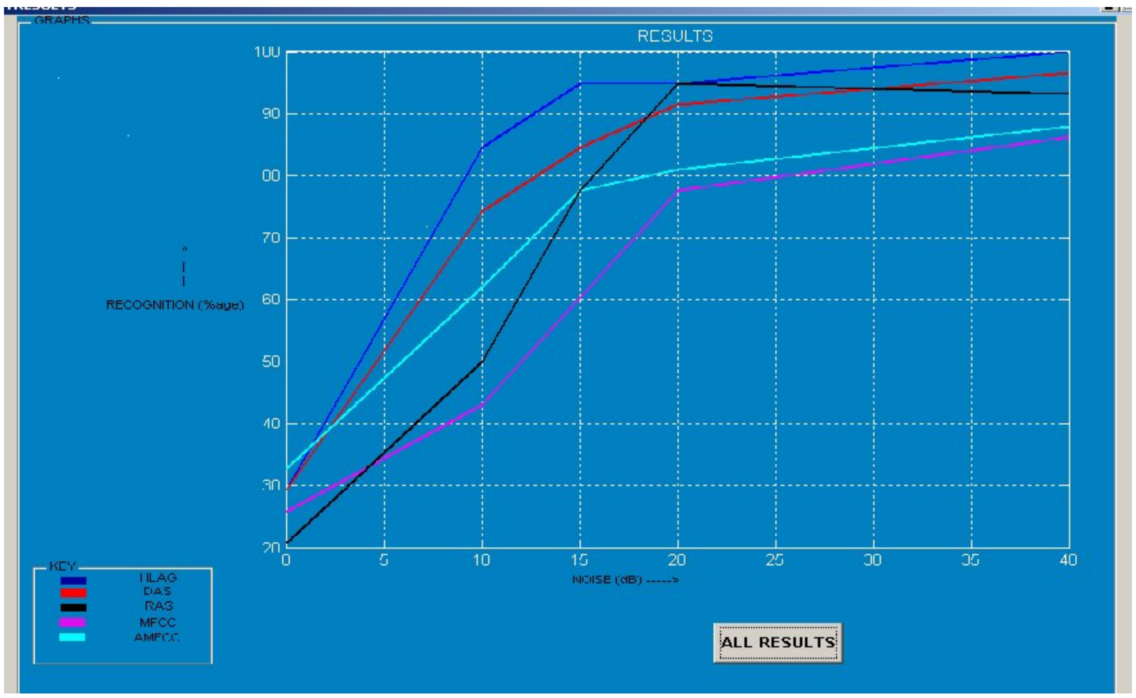


Figure 11(d): Graphical representation of Figure 11(c)

Module\Noise	0dB	10dB	15dB	20dB	40dB
MFCC	25.8621	44.8276	58.6207	75.8621	86.2069
AMFCC	25.8621	62.0690	82.7586	82.7586	87.9310
RAS	27.5862	50.0000	68.9655	87.9310	93.1034
DAS	27.5862	77.5862	91.3793	96.5517	98.2759
HLAG	27.5862	79.3103	91.3793	96.5517	100.0000

Figure 11(e): Results with maximum recognition for Higher-Lag Module for Factory noise

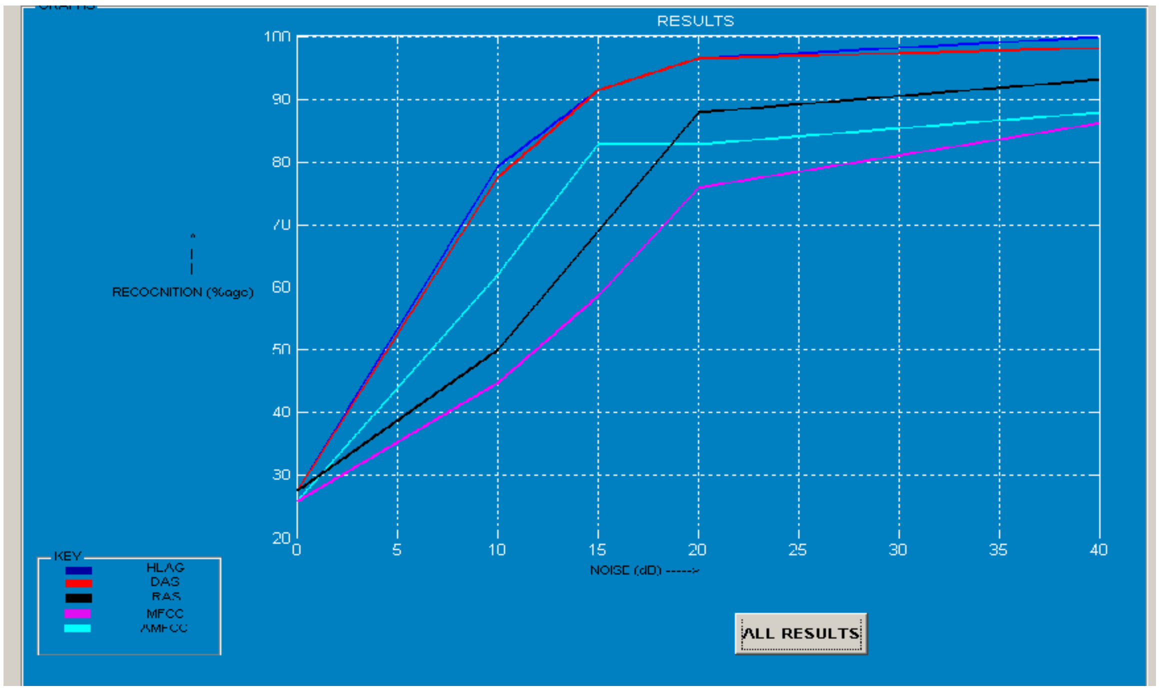


Figure 11(f): Graphical representation of Figure 11(e)

Module\Noise	**0dB**	**10dB**	**15dB**	**20dB**	**40dB**
MFCC	27.5862	44.8276	63.7931	79.3103	87.9310
AMFCC	25.8621	51.7241	68.9655	84.4828	87.9310
RAS	24.1379	44.8276	67.2414	86.2069	93.1034
DAS	36.2069	68.9655	89.6552	96.5517	98.2759
HLAG	29.3103	70.6897	93.1034	94.8276	100

Figure 11(g): Results with maximum recognition for Higher-Lag Module for Babble noise

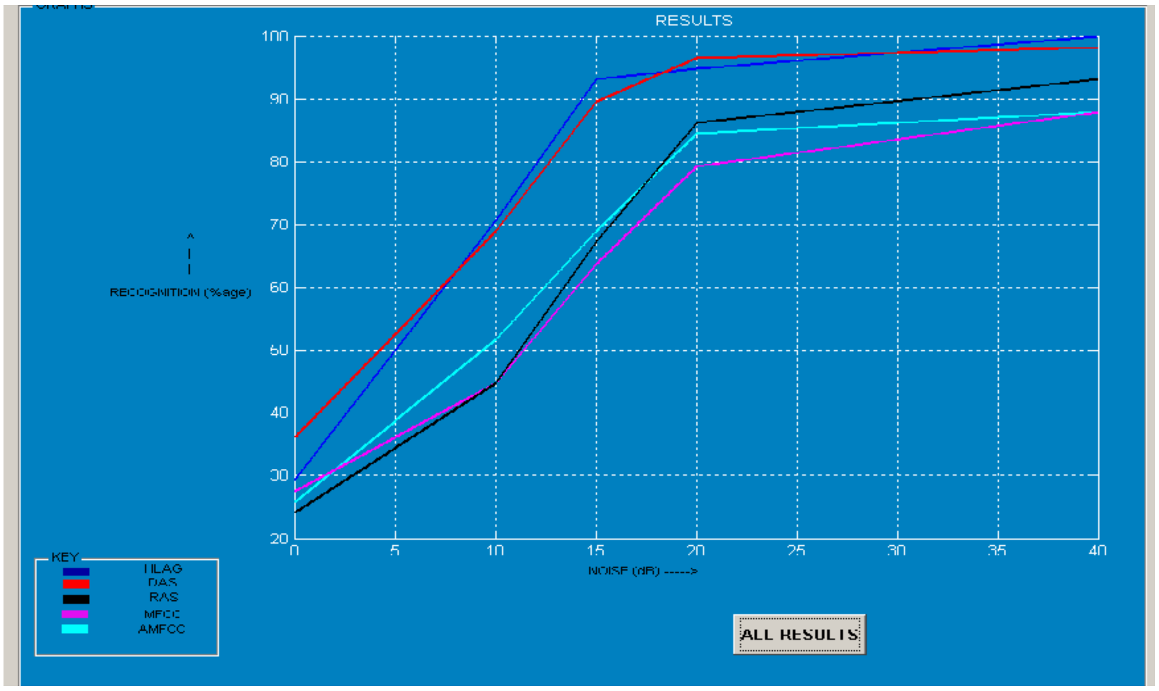


Figure 11(h): Graphical representation of Figure 11(g)

Figure 11(i): Results with maximum recognition for Higher-Lag Module for white noise

Module\Noise	0dB	10dB	15dB	20dB	40dB
LPCC	24.1379	31.0345	60.3448	77.5862	96.5517
OSALPC	32.7586	27.5862	55.1724	89.6552	98.2759
SMC	24.1379	25.8621	50.0000	86.2069	93.1034
HLAG	38.2414	60.6897	84.1379	91.3793	96.5517

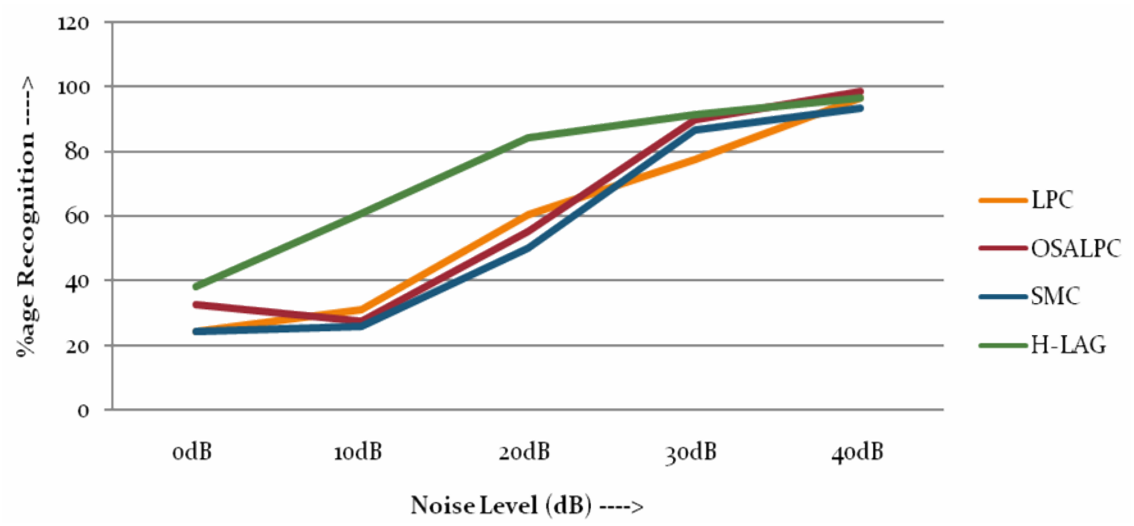


Figure 11(j): Graphical representation of Figure 11(i)

Module\Noise	0dB	10dB	15dB	20dB	40dB
LPCC	25.8621	32.7586	74.1379	91.3793	96.5517
OSALPC	25.8621	31.0345	77.5862	91.3793	98.2759
SMC	24.1379	48.2759	75.8621	91.3793	93.1034
HLAG	35.8621	64.4828	71.7241	96.5517	96.5517

Figure 11(k): Results with maximum recognition for Higher-Lag Module for F16 noise

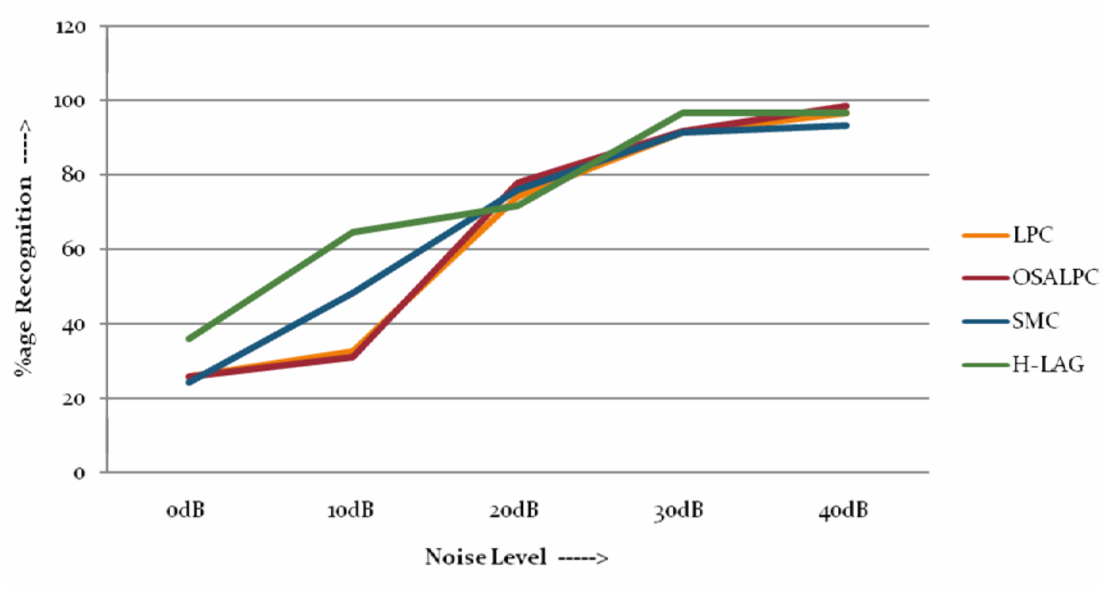


Figure 11(l): Graphical representation of Figure 11(k)

Module\Noise	0dB	10dB	15dB	20dB	40dB
LPCC	23.1034	52.0690	75.8621	86.2069	96.5517
OSALPC	27.5862	53.4483	91.3793	94.8276	98.2759
SMC	24.1379	51.7241	81.0345	87.9310	93.1034
HLAG	35.8621	71.0345	84.8276	96.5517	96.5517

Figure 11(m): Results with maximum recognition for Higher-Lag Module for Factory noise

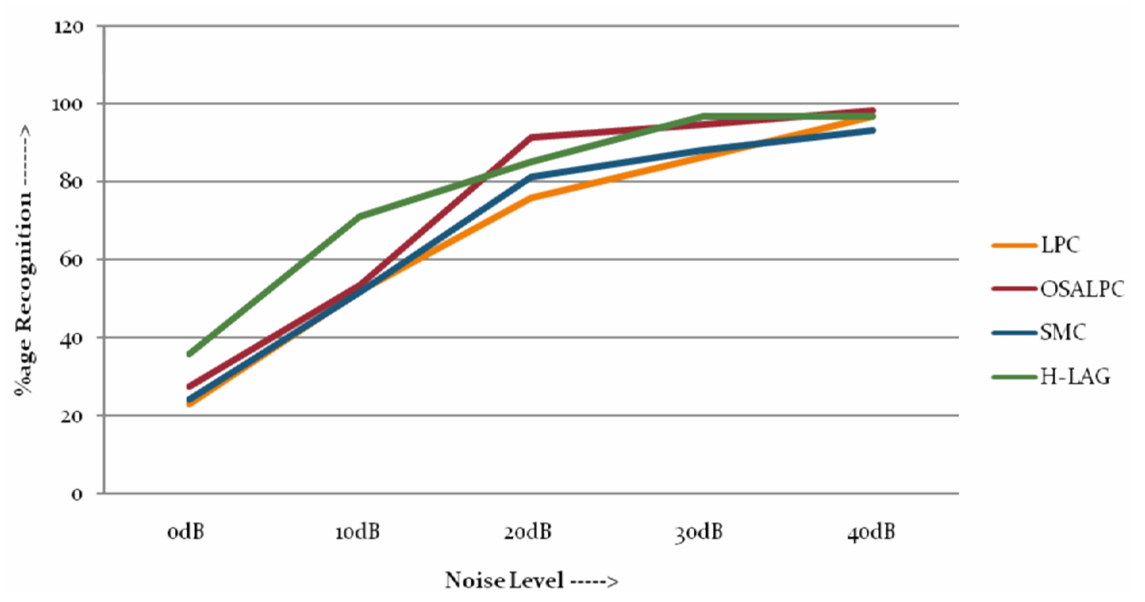


Figure 11(n): Graphical representation of Figure 11(m)

Module\Noise	**0dB**	**10dB**	**15dB**	**20dB**	**40dB**
LPCC	33.1034	75.8621	91.3793	94.8276	96.5517
OSALPC	36.2069	77.5862	87.9310	94.8276	98.2759
SMC	32.4138	76.2069	89.6552	91.3793	93.1034
HLAG	36.2069	77.3448	87.9310	96.5517	96.5517

Figure 11(o): Results with maximum recognition for Higher-Lag Module for Babble noise

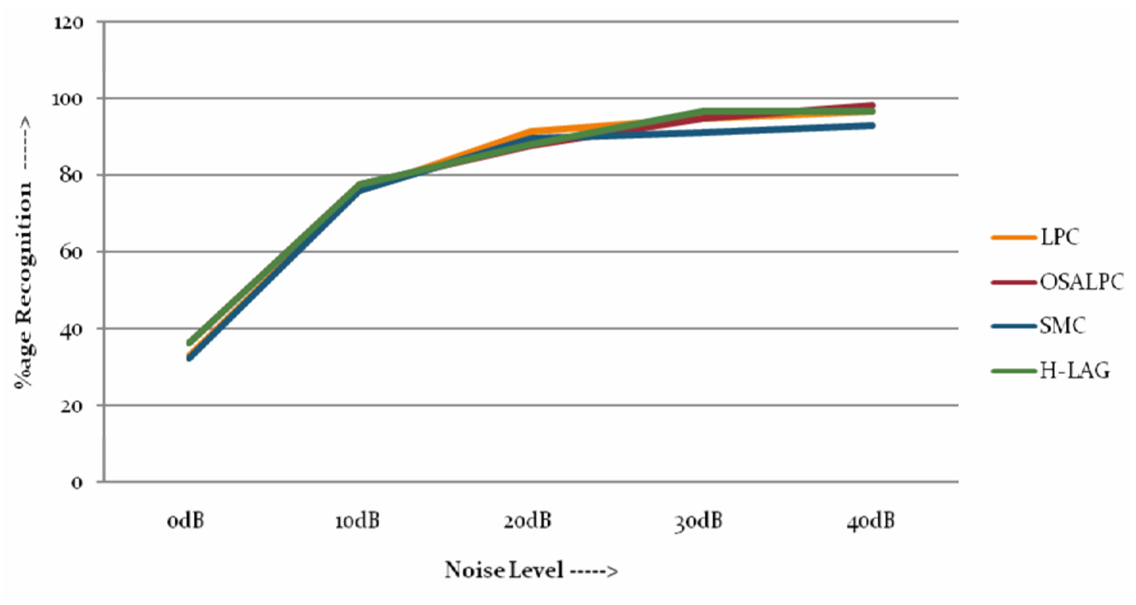


Figure 11(p): Graphical representation of Figure 11(o)

NOISE/ MODULE	0dB	10Db	20dB	30Db	40dB
HASE (Segmental SNR in dB)	-34.5231	-22.9957	-21.3760	-14.2681	-16.6824
AR-HASE (Segmental SNR in dB)	-0.9359	1.0797	4.6661	6.3311	5.8434

Figure 11(q): Results using Segmental SNR with maximum enhancement for AR-HASE than HASE

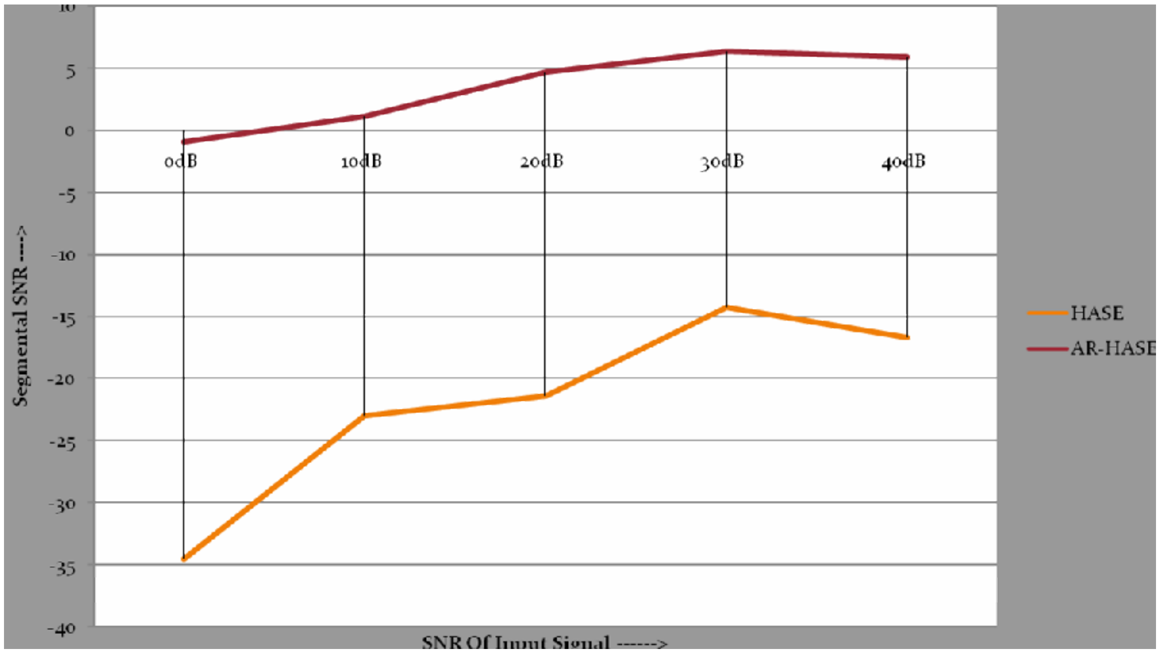


Figure 11(r): Graphical representation of Figure 10(q)

Chapter 7

APPLICATION OF SPEECH RECOGNITION SYSTEM

Speech Processing is an upcoming field. A lot of research is conducted in this field. Today people are interested in hand free systems. A password can be hacked but using voice as a password can not be hacked. We want to make the use of mouse obsolete. As we speak 'my computer', it should be double clicked or in general the respective application should be opened.
Extracted features in speech recognition system can also be used for speech Speech-to-Text conversion. For further use, speech recognition system with various enhancements can be used for Automatic Translation, Vehicle Navigation System, Mobile Telephony, Hands-free computing, Robotics etc.

I have used its application in design of voice automated robot.

Voice based robotic control is an interesting project, mainly used for industrial and surveillance applications. It gives exact concept of controlling a robot by a voice instruction.

Robot is capable of understanding and synthesizing human speech for communication. A voice recognition unit built around a high speed processor that ensures various operations of the system to be performed by voice command. A few of commands recommended for operation are listed as: START, STOP, FORWARD, REVERSE, RIGHT, LEFT, SLOW, FAST,OK, UP, DOWN, CLOCK, ANTICLOCK, CLOSE,OPEN.

7.1 Implementation Framework

The speech recognition system is programmed in the manner that the system has to be trained in spoken language (or vocal utterances) by the user so that circuit is to recognize the voice of a specific user. This board allows us to experiment with many facets of speech recognition technology. It has 8 bit data which interfaced with any microcontroller.

The 89S51 microcontroller contains four ports of each eight pins. In this project one port is dedicated for speech recognition. Relays are interfaced through ULN driver circuit to control the electrical appliances. A simple yet powerful program is written in assembly language and burned into the microcontroller to record and accept voice instructions and to control the devices. . This project uses regulated 5V, 1A power supply.

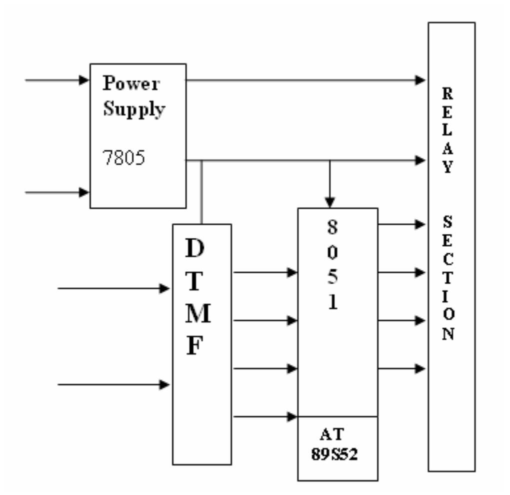


Figure 12 Block diagram of the system

7.2 Working of the Robot

The robot works as follows.

1. The modules are assembled and power supply is given to the robot.

2. Different Words are spoken by user with the help of microphone, words are recognized by speech recognition module . Once the word is accepted, the corresponding tone is generated against each word.

3. This tone is sent to DTMF Decoder(IC 8870) for decoding.

4. The decoded signals from this module are sent to 8051 module (AT89S52) for interpretation. The microcontroller takes the decision on this basis and passes signals to the relay circuit to make the robot turn accordingly.

5. The robot turn based on the signals received from the 8051 module and thus, the robot moves in the desired direction.

7.3 USES

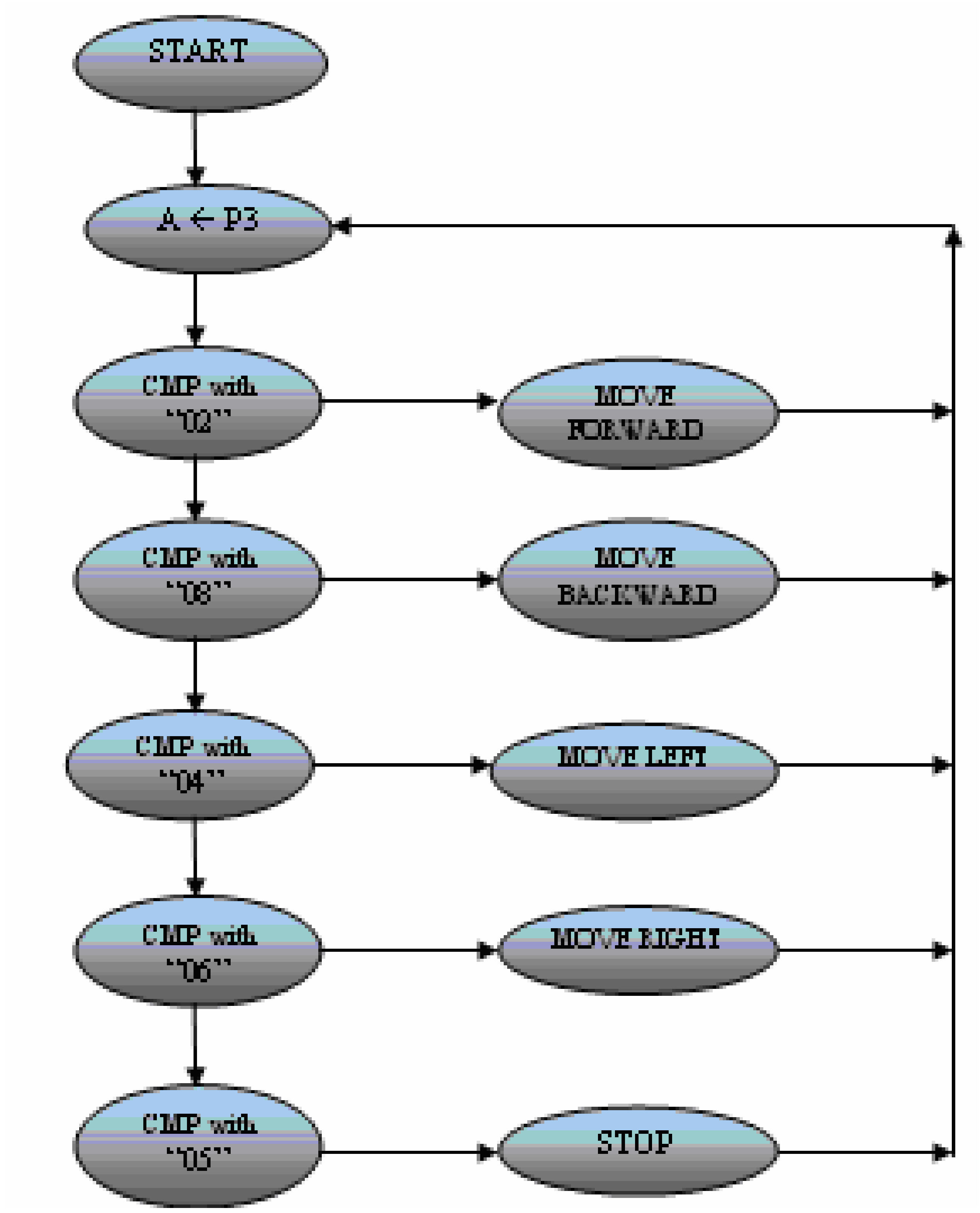
The following software was used for the development of the project:

1. It helps physically disabled persons by carrying some objects from one place to another place using the arm structure in the robot.
2. It guides the blind persons to reach a particular Destination by using the voice feature.
3. It is used to guide visitors in an organization by providing information about the facilities available.
4. Because of the presence of the Real-time Clock (DS1307), time-based control of the robot is possible. For example, it is used in hospitals to inform patients to take the tablets at the right time.
5. It is used in hazardous places.
6. The photo electric sensor in the robot will sense the obstacles and it will make decisions according to the obstacles it encounters.

DESIGN OF GESTURE AUTOMATED MOBILE ROBOT

After the design of voice automated mobile robot I have designed the robot which will move according to gesture .when I will move the object in any direction through webcam robot will move in that direction.

FLOW CHART


START
A ← P3
CMP with "02"
MOVE FORWARD
CMP with "08"
MOVE BACKWARD
CMP with "04"
MOVE LEFT
CMP with "06"
MOVE RIGHT
CMP with "05"
STOP

Chapter 8

CONCLUSION

In this system implementation of advanced feature extraction for speech recognition system. Various feature extraction algorithms are implemented and are then compared with each other with respect to their performance. Graphs are plotted at different noise levels to represent their respective performance in terms of their recognition rates. The user can play the file and on generating the result, the recognized word is displayed on the screen. Wide range of possibilities exists for parametrically representing the speech signal for the speech recognition task, such as Linear Prediction Coding (LPC), Mel-Frequency Cepstrum Coefficients (MFCC), and others. And after removal of lower lags from the input speech signal the Higher Lag Autocorrelation Coefficients method is perhaps then the best known and most effective.

The robot is controlled through connected speech input. The language input allows a user to interact with the robot which is familiar to most of the people. The advantages of speech activated robots are hands-free and fast data input operations. In future, it is expected that speech recognition systems will be used as man-machine interface for robots in rehabilitation, entertainment etc. In view of this, aforementioned system is a source of learning process for a mobile robot which takes speech input as commands and performs some navigation task through a distinct man-machine interaction with the application of the learning. The speech recognition system is trained in such a way that it recognizes defined commands and the designed robot navigates based on the instruction through the Speech Commands. The medium of interaction between humans and computers is on the processing of speech (words uttered by the person). . The complete system consists of three sub-systems, the speech recognition system, a central controller and the robot .we have studied the various factors such as noise which interferes speech recognition and distance factor. The results prove that proposed robot is capable of understanding the meaning of speech commands. After the design of voice automated mobile robot it is also possible to control a robot through gesture recognition. The result also proves that robot is capable of understanding the meaning of gesture commands.

Chapter 9

FUTURE SCOPE

After recognizing the speech i.e. the word spoken by the user, the user can also be recognized, by using the advance features that are extracted in this project. These advanced features of the input speech can further be implemented using various algorithms for speaker recognition.

Speech Processing is an upcoming field. A lot of research is conducted in this field. Today people are interested in hand free systems. A password can be hacked but using voice as a password can not be hacked. We want to make the use of mouse obsolete. As we speak 'my computer', it should be double clicked or in general the respective application should be opened.

Extracted features in speech recognition system can also be used for speech Speech-to-Text conversion. For further use, speech recognition system with various enhancements can be used for Automatic Translation, Vehicle Navigation System, Mobile Telephony, Hands-free computing etc.

Chapter 10

REFERENCES

[1] L.R. Rabiner and B.H. Juang, Fundamentals of Speech Recognition, Prentice-Hall,Englewood Cliffs, N.J., 1993.

[2] L.R Rabiner and R.W. Schafer, Digital Processing of Speech Signals, Prentice-Hall,Englewood Cliffs, N.J., 1978.

[3] Y. Linde, A. Buzo & R. Gray, "An algorithm for vector quantizer design", IEEETransactions on Communications, Vol. 28, 1980.

[4] Satya Dharanipragada, Umit S.Yatanel & Bhaskar D.Rao, "Robust FeatureExtraction For Continous Speech Recognition using the MVDR Spectrum Estimation Method, IEEE Trans. Audio, Speech & Language Processing, Vol.15, No.1, pp.224-234, Jan 2007.

[5] Benjamin J. Shannon, Kuldip K. Paliwal, "Feature Extraction from Higher Lag Autocorrelation Coefficients for Robust Speech Recognition", ELSEVIER Trans. Speech & Audio Processing, Jan 2003.

[6] Chulhee Lee, Domghoon Hyun, Euisum Choi, Jimwook Go & Chungyong Lee, "Optimizing Feature Extraction For Speech Recognition", IEEE Trans. Speech & Audio Processing, Vol.11, No.1, pp.80-87, Jan 2003.

[7] Jingdong Chen, Kuldip K. Paliwal & Satoshi Nakamura, "Cepstrum Derived From Diffrentiated Power Spectrum for Robust Speech Recognition", Jan 2003.

[8] www.ieeexplore.ieee.org

Appendix A

FUNCTIONS USED

Modules:

1. pre.m : This function is used for p re-emphasis purpose.
2. frm.m : This function is used for blocking of continuous speech signal into frames of N samples, with adjacent frames being separated by M ($M < N$).
3. ham.m : This function is used for windowing purpose.
4. auto_corr.m : This function is used to take autocorrelation sequence.
5. fft_amp.m : This function is used to take fast fourier transformation.
6. filtering.m : This function is used for filtering purpose.
7. melbank.m : This function is used to apply malbank filters.
8. mfcc.m : This function is used to take mfcc.

Training:

1. create_code.m : This function is used to generate codebook.
2. HMMmodule.m : this function is used to apply HMM on the generated codebook.

Testing:

1. Testallmodules.m : This function is used for testing.
2. Testmodule.m : This function is used for testing.

Interfaces:

1. spee.m : This function is used to show interface.
2. TESTRESULTS.m : This function is used to show results.

TRAINING DATABASE

- Number of input words = 14
- Number of utterances for each word = 15
- Sampling frequency = 16 KHz
- Number of bits required for quantization = 16 bits
- Frame Size = 16 ms

TESTING DATABASE

- Number of words = 14
- Number of utterances for each word = 5

Appendix B

CODING

Input Speech

```
[s,fs]= wavread(FILE);
%n is the no of samples in each frames
%m is the diffrence b/w the start of 2 frames
n=256;
m=156;
l1=length(s);
```

Pre-emphasis

```
for i=2:l1
   sig(i)=s(i)*(1-(0.97*s(i-1)));
end

l=length(sig)
nb = floor((l-n)/m)+1;
```

Framing

```
x(1:n,1:nb)=0;    %framing the signal with overlap

for j=1:nb
   x(:,j)=s(((n-m)*(j-1))+1:((n-m)*(j-1))+256);
end
```

Autocorrelation

```
sum=0;
for j=1:nb
   for k=1:n
      for i=1:n-k
         z=a(i,j)*a(i+k,j);
         sum=sum+z;
         z=0;
      end
      auco(k,j)=sum;
      sum=0;
```

```
    end
end
```

Lower Lag Removal

```
for i=1:nb
    for k=23:n
        lagre(k-22,i)=auco(k,i);
    end
end

for j=1:nb
   for k=235:n
      lagre(k,j)=0;
   end
end
```

Filter

```
for j=3:nb-2
    for k=1:n
     sum=auco(k,j+1)+auco(k,j+2)-auco(k,j-1)-auco(k,j-2) ;
     h(k,j-2)=sum/10;
     sum=0;
    end
end
for j=floor(nb-4):nb
    for i=1:n
        h(i,j)=0;
    end
end
```

Hamming

```
k=0.54;

for j = 1:nb
        for i = 0:n-1
           o=0.46*cos((i*2*pi)/255);
           p=k-o;
           a(i+1,j)=p*x(i+1,j);
        end
end
```

FFT

```
for j = 1:nb
   for i=1:n
      sfft(i,j)= fft(h(i,j));
   end
end
```

Mel-frequency bank & mfcc

```
%p number of filters in filterbank
%n length of fft
%fs sample rate in Hz

f0=700/fs;
fn2=floor(n/2);
lr=log(1+0.5/f0)/(p+1);

%convert to fft bin numbers with 0 for DC term

b= n*(f0*(exp([0 1 p p+1]*lr)-1));
b1=floor(b(1))+1;
b2=ceil(b(2));
b3=floor(b(3));
b4=min(fn2,ceil(b(4)))-1;
pf=(log(1+(b1:b4)/n/f0)/lr);
fp=floor(pf);
pm=pf-fp;
r=[fp(b2:b4) 1+fp(1:b3)];
c=[b2:b4 1:b3]+1;
v=2*[1-pm(b2:b4) pm(1:b3)];
m1=sparse(r,c,v,p,1+fn2);


for j=1:nb
   n2=1+floor(n/2);
   ms=m1*v(1:n2,j).^2;
   if ms==0
      v1(:,j)=0;
   else
      v1(:,j)=dct(log(ms));
   end
   %converting back to time domain
```

```
End
```

Codebook Creation

```
%accessing the folder containing the necesssary modules
addpath('C:\MATLAB701\work\modules');

Z = 4;   %no. of words(folders) in the database(\train directory)
B = 128;      %number of codebook entries (clusters)
start = 1;

for i = 1 : Z
   WORD = strcat('C:\MATLAB701\work\modules\train\',int2str(i+4),'\')
   %folder(word) no. i

   % Retrieve the filenames from the database
   FileNames = dir(WORD);
   FileNames = char(FileNames.name);
   [rowFN, colFN] = size(FileNames);
   FileNames = [FileNames(3:end, :)];
   for index = 1:rowFN-2
         %load the files from the database
      FILE = strcat(WORD,FileNames(index, :));
      COEF = HLAGmodule(FILE);
      clear FILE;
      if(start == 1) %then data not init yet
         OBS = COEF;
         start = 2;
      else
         OBS = [OBS  COEF];
      end
   end
end
   OBS = OBS';
   [idx CB] = kmeans(OBS,B,'maxiter',150,'EmptyAction','drop');
   %open file for writing
   DATA = strcat('C:\MATLAB701\work\modules\test\','codebookHLAG1','.mat');
   save(DATA,'CB');
```

HMM

```
%uses the modules stored in the HMM directory.
%making the directory accesible to MATLAB
addpath('C:\MATLAB701\work\modules\studentsHLAG\HMM');
```

```
B = 128;       %number of codebook entries
HMM = 4;      %the number of hmm's (depends on no. of words)
L = 7;        %number of states per hmm
count = zeros(1,B); %to count how many times each codeword is observed
skp = 0;
warning off all;

%read codebook in from file
C = load('C:\MATLAB701\work\modules\test\codebookHLAG1.mat');
CB = C.CB;

min = 1e-4;
%will use fixed number of states (7) for each digit's HMM
%first initialize the observational probabilities all to a min value
%format is observe(model,observation,state)
observe = min*ones(HMM,B,L);
%the follwing fixed transition table will be used for all hmms
trans = zeros(L);
trans_lg = zeros(L);    %pre-computed logs of transitions
%hmm is left to right
edges = 2;
for i = 1:L
   for j = 1:i-1                              %no back transitions
      trans(i,j) = 0;
      trans_lg(i,j) = -10e50;
   end

   if(i+(edges-1) <= L)
      for j = i:i+(edges-1)
         trans(i,j) = 1/edges;
         trans_lg(i,j) = log10(1/edges);
      end
   else                                       %allowed transitions
      for j = i:L
         trans(i,j) = 1/(L-i+1);
         trans_lg(i,j) = log10(1/(L-i+1));
      end
   end

   for k = j+1:L
      trans(i,k) = 0;
      trans_lg(i,k) = -10e50;                 %too far to transition
   end
end
%the initial probs
```

```
%ntial = 1/L*ones(1,L);
ntial = [.8 .18  .01 .01 .01 .01 .01];

for i = 1:HMM
   trans_hat(i,:,:) = trans;
   init_hat(i,:) = ntial;
end

%initial estimate of the obs probs
for i = 0 : HMM-1
   WORD = strcat('C:\MATLAB701\work\modules\train\',int2str(i+5),'\') %folder(word)
no. i+1
   % Retrieve the filenames from the database
   FileNames = dir(WORD);
   FileNames = char(FileNames.name);
   FileNames = [FileNames(3, :)];  %considering only 1 utterance
   % load the files from the database
   FILE = strcat(WORD,FileNames);
   cep = HLAGmodule(FILE);
   clear FILE;
   cep = transpose(cep);
   length = size(cep);
   seg = fix(length(1) / L);
   rem = mod(length(1),L);
   for j = 0:L-1    %state j
      for k = 1:seg    %frame j*seg+k
         index = vq(CB,cep(j*seg+k,:));
         count(index) = count(index) + 1;
      end
      if (j == L-1)   %add any extra segments at end to last state
         if(rem ~= 0)
            for k = 1:rem
               index = vq(CB,cep(j*seg+k,:));
               count(index) = count(index) + 1;
            end
         end
      end
      for k = 1:B    %computing the probs
         if (count(k) ~= 0)
            if(j ~= L-1)
               observe(i+1,k,j+1) = count(k) / seg;
            else
               observe(i+1,k,j+1) = count(k) / (seg + rem);
            end
         end
      end
```

```
        count = zeros(1,B); %make count zero again for next state
    end
end


for i = 0:HMM-1
    WORD = strcat('C:\MATLAB701\work\modules\train\',int2str(i+5),'\') %folder(word)
no. i
    % Retrieve the filenames from the database
    FileNames = dir(WORD);
    FileNames = char(FileNames.name);
    [rowFN, colFN] = size(FileNames);

    FileNames = [FileNames(3:end, :)];  %taking all utterances
    for index = 1:rowFN-2
        % load the files from the database
        FILE = strcat(WORD,FileNames(index, :));
        cep = HLAGmodule(FILE);
        clear FILE;
        cep = transpose(cep);
        length = size(cep); nf = length(1);
        for nf_i = 1:nf
            O(nf_i) = vq(CB,cep(nf_i,:));
        end   %creating the index sequence
        clear temp;
        for x = 1:L
            for y = 1:L
                temp(x,y)= trans_hat(i+1,x,y);
            end
        end
        [seq,p_o] = viterbi(observe(i+1,:,:),log10(temp),ntial,O,nf,L);
        %make initial re-estimates based on optimal state seq
        %number of vectors with codebook index in state
        b_count = zeros(B,L);
        %number of vectors in state
        count = zeros(L);
        for seq_i = 1:nf
            b_count(O(seq_i),seq(seq_i)) = b_count(O(seq_i),seq(seq_i))+1;
            count(seq(seq_i)) = count(seq(seq_i)) + 1;
        end
        %find available probs
        for seq_i = 1:B
            for seq_j = 1:L
                if(b_count(seq_i,seq_j))
                observe(i+1,seq_i,seq_j)=b_count(seq_i,seq_j)/count(seq_j);
                end
            end
        end
        %do forward backward
        [alpha,pp,ct] = fwd(observe(i+1,:,:),trans_hat(i+1,:,:),...
            ntial,O,nf,L);
        beta = bck(observe(i+1,:,:),trans_hat(i+1,:,:),O,nf,L,ct);
        %now repeat for each observational prob
        p_o_t = p_o;
        for ob = 1:B
            for n = 1:L
                if(p_o == 0)
                    display('here')
                    p_o = 10e-20;
                end
                %new re-estimated model
                observe_hat(i+1,ob,n) = em(alpha,beta,observe(i+1,:,:),...
                    trans_hat(i+1,:,:),O,seq,n,ob);
            end
        end
        %compare new model vs previous model if better keep else discard
        [seq,prb] = viterbi(observe_hat(i+1,:,:),log10(temp),ntial,O,nf,L);
        %since digit being trained on and the model is known, then a
        %higher prob is desired
        (1/nf)*(prb-p_o_t);
        %pause
        if(prb > p_o_t)
            observe(i+1,:,:) = observe_hat(i+1,:,:);
            trans_hat(i+1,:,:) = aem(alpha,beta,observe(i+1,:,:),...
                trans_hat(i+1,:,:),O);
            init_hat(i+1,:) = iem(alpha,beta,observe(i+1,:,:),...
                trans_hat(i+1,:,:),O);
        end
    end
end

%open file for writing
    DATA = strcat('C:\MATLAB701\work\modules\test\','hmmHLAG1','.mat');
    save(DATA,'observe');
%open file for writing
    DATA = strcat('C:\MATLAB701\work\modules\test\','transHLAG1','.mat');
    save(DATA,'trans_hat');
%open file for writing
    DATA = strcat('C:\MATLAB701\work\modules\test\','initHLAG1','.mat');
    save(DATA,'init_hat');
```

Testing Module

```
addpath('C:\MATLAB701\work\modules');
addpath('C:\MATLAB701\work\modules\studentsHLAG\HMM');

B = 128;       %number of codebook entries
     %the number of hmm's
L = 7;        %number of states per hmm
min = 1e-4;
warning off all;

%-----------------------------test for HLagmodule--------------------------


if df==0
HMM = 4;
%read codebook in from file
C = load('C:\MATLAB701\work\modules\test\codebookHLAG.mat');
CB = C.CB;

%read other data files
C = load('C:\MATLAB701\work\modules\test\hmmHLAG.mat');
observe = C.observe;

C = load('C:\MATLAB701\work\modules\test\transHLAG.mat');
trans_hat = C.trans_hat;

C = load('C:\MATLAB701\work\modules\test\initHLAG.mat');
init_hat = C.init_hat;

elseif df==4
 HMM = 4;
   %read codebook in from file
C = load('C:\MATLAB701\work\modules\test\codebookHLAG1.mat');
CB = C.CB;

%read other data files
C = load('C:\MATLAB701\work\modules\test\hmmHLAG1.mat');
observe = C.observe;

C = load('C:\MATLAB701\work\modules\test\transHLAG1.mat');
trans_hat = C.trans_hat;

C = load('C:\MATLAB701\work\modules\test\initHLAG1.mat');
init_hat = C.init_hat;
```

```
elseif df==8
 HMM = 4;
   %read codebook in from file
C = load('C:\MATLAB701\work\modules\test\codebookHLAG2.mat');
CB = C.CB;

%read other data files
C = load('C:\MATLAB701\work\modules\test\hmmHLAG2.mat');
observe = C.observe;
C = load('C:\MATLAB701\work\modules\test\transHLAG2.mat');
trans_hat = C.trans_hat;

C = load('C:\MATLAB701\work\modules\test\initHLAG2.mat');
init_hat = C.init_hat;

elseif df==12
HMM = 2;
%read codebook in from file
C = load('C:\MATLAB701\work\modules\test\codebookHLAG3.mat');
CB = C.CB;

%read other data files
C = load('C:\MATLAB701\work\modules\test\hmmHLAG3.mat');
observe = C.observe;

C = load('C:\MATLAB701\work\modules\test\transHLAG3.mat');
trans_hat = C.trans_hat;

C = load('C:\MATLAB701\work\modules\test\initHLAG3.mat');
init_hat = C.init_hat;

end

%the follwing fixed transition table will be used for all hmms
%ntial = 1/L*ones(1,L);
ntial = [.8 .18 .01 .01 .01 .01 .01];
wrong_count = zeros(1,HMM);
right_count = zeros(1,HMM);
conf = zeros(HMM,HMM);
for i = 0:HMM-1
   WORD = strcat('C:\testfiles\'); %folder(word) no. i+1
   %WORD = strcat('C:\MATLAB701\work\modules\train\',int2str(i+1),'\')
%folder(word) no. i+1
   % Retrieve the filenames from the database
   FileNames = dir(WORD);
   FileNames = char(FileNames.name);
```

```
    [rowFN, colFN] = size(FileNames);
    FileNames = [FileNames(3:end, :)];  %taking all utterances
    for index = 1:rowFN-2
        % load the files from the database
        FILE = strcat(WORD,FileNames(index, :));
        cep = HLAGmodule(FILE);
        cep = transpose(cep);
        length = size(cep);
        nf = length(1);
        for nf_i = 1:nf
            O(nf_i) = vq(CB,cep(nf_i,:));
        end
        for hm = 1:HMM
            clear temp;
            for x = 1:L
                for y = 1:L
                    temp(x,y)= trans_hat(hm,x,y);
                end
            end
            [seq,lhood(hm)] = viterbi(observe(hm,:,:),log10(temp),...
                ntial,O,nf,L);
        end
        lhood;
        [mx,indx] = max(lhood);
        conf(i+1,indx) = conf(i+1,indx)+1;
        if(indx == i+1)
            right_count(i+1) = right_count(i+1) + 1;
        else
            wrong_count(i+1) = wrong_count(i+1) + 1;
        end
    end
end
clear WORD;
percent_err = sum(wrong_count) / sum(wrong_count+right_count) * 100;
success = sum(right_count) / sum(wrong_count+right_count) * 100;
%conf = conf/index * 100;
wrong_count;
right_count;

[b,c]=max(right_count);
%per=success;
```

Interface

```
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @spee_OpeningFcn, ...
                   'gui_OutputFcn',  @spee_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

function spee_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);

function varargout = spee_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

function seter(b)
global a;
a=b;
function pushbutton1_Callback(hObject, eventdata, handles)
global user_entry;

f1=strfind(user_entry,'132.wav')
f2=strfind(user_entry,'133.wav')
f3=strfind(user_entry,'134.wav')
f4=strfind(user_entry,'135.wav')
g1=strfind(user_entry,'136.wav')
g2=strfind(user_entry,'137.wav')
g3=strfind(user_entry,'140.wav')
g4=strfind(user_entry,'141.wav')
h1=strfind(user_entry,'170.wav')
h2=strfind(user_entry,'171.wav')
h3=strfind(user_entry,'172.wav')
h4=strfind(user_entry,'173.wav')
i1=strfind(user_entry,'174.wav')
i2=strfind(user_entry,'175.wav')
```

```
if f1~=0
        st=TESTallmodules(0);
elseif f2~=0
        st=TESTallmodules(0);
elseif f3~=0
        st=TESTallmodules(0);
elseif f4~=0
        st=TESTallmodules(0);
elseif g1~=0
        st=TESTallmodules(4);
elseif g2~=0
        st=TESTallmodules(4);
elseif g3~=0
        st=TESTallmodules(4);
elseif g4~=0
        st=TESTallmodules(4);
elseif h1~=0
        st=TESTallmodules(8);
elseif h2~=0
        st=TESTallmodules(8);
elseif h3~=0
        st=TESTallmodules(8);
elseif h4~=0
        st=TESTallmodules(8);
elseif i1~=0
        st=TESTallmodules(12);
elseif i2~=0
        st=TESTallmodules(12);
end
clear user_entry;

sprintf('NAME OF CITY IS %s',st);

h = uicontrol('Style', 'text', 'String', st,...
    'Position', [590 210 300 50],'HorizontalAlignment',...
    'center','Fontsize',18,'fontweight','bold','foregroundcolor',[0 0 0]);

function pushbutton2_Callback(hObject, eventdata, handles)
play();

function edit1_Callback(hObject, eventdata, handles)

global user_entry;
user_entry = get(hObject,'string');
savefile(user_entry);
```

```
h = uicontrol('Style', 'text', 'String', user_entry,...
   'Position', [200 210 300 20],'HorizontalAlignment',...
   'center','Fontsize',10,'fontweight','bold');
h = uicontrol('Style', 'text', 'String', '',...
   'Position', [200 170 300 20],'HorizontalAlignment',...
   'center','Fontsize',10,'fontweight','bold');

function edit1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function pushbutton26_Callback(hObject, eventdata, handles)
WhiteNoise('C:\testfiles\test.wav',0);
h = uicontrol('Style', 'text', 'String', '0dB Noise Added',...
   'Position', [200 170 300 20],'HorizontalAlignment',...
   'center','Fontsize',10,'fontweight','bold');

function pushbutton27_Callback(hObject, eventdata, handles)
WhiteNoise('C:\testfiles\test.wav',5);
h = uicontrol('Style', 'text', 'String', '5dB Noise Added',...
   'Position', [200 170 300 20],'HorizontalAlignment',...
   'center','Fontsize',10,'fontweight','bold');

function pushbutton28_Callback(hObject, eventdata, handles)
 WhiteNoise('C:\testfiles\test.wav',10);
h = uicontrol('Style', 'text', 'String', '10dB Noise Added',...
   'Position', [200 170 300 20],'HorizontalAlignment',...
   'center','Fontsize',10,'fontweight','bold');

function pushbutton29_Callback(hObject, eventdata, handles)
 WhiteNoise('C:\testfiles\test.wav',15);
h = uicontrol('Style', 'text', 'String', '15dB Noise Added',...
   'Position', [200 170 300 20],'HorizontalAlignment',...
   'center','Fontsize',10,'fontweight','bold');

function pushbutton30_Callback(hObject, eventdata, handles)
 WhiteNoise('C:\testfiles\test.wav',20);
h = uicontrol('Style', 'text', 'String', '20dB Noise Added',...
   'Position', [200 170 300 20],'HorizontalAlignment',...
   'center','Fontsize',10,'fontweight','bold');

function pushbutton31_Callback(hObject, eventdata, handles)
 WhiteNoise('C:\testfiles\test.wav',25);
```

```
h = uicontrol('Style', 'text', 'String', '25dB Noise Added',...
    'Position', [200 170 300 20],'HorizontalAlignment',...
    'center','Fontsize',10,'fontweight','bold');

function pushbutton32_Callback(hObject, eventdata, handles)
 WhiteNoise('C:\testfiles\test.wav',30);
h = uicontrol('Style', 'text', 'String', '30dB Noise Added',...
    'Position', [200 170 300 20],'HorizontalAlignment',...
    'center','Fontsize',10,'fontweight','bold');

function pushbutton33_Callback(hObject, eventdata, handles)
 WhiteNoise('C:\testfiles\test.wav',40);
h = uicontrol('Style', 'text', 'String', '40dB Noise Added',...
    'Position', [200 170 300 20],'HorizontalAlignment',...
    'center','Fontsize',10,'fontweight','bold');
```

Appendix C

Assembly Language Program Code for robot

```
                FWD EQU P0.0
                RWD EQU P0.1

                LFT EQU P0.2
                RGT EQU P0.3

               INP EQU P1

          ;-------------------------------

              ORG 0000H

         MAIN:
          MOV A,INP

          CJNE A,#1111 0010B,NXT1        ; 2
            AJMP FWDF

     NXT1:   CJNE A,#1111 1000B,NXT2        ; 8
            AJMP BWDF

     NXT2:   CJNE A,#1111 0100B,NXT3        ; 4
            AJMP LFTF

    NXT3:   CJNE A,#1111 0110B,NXT4        ; 6
             AJMP RGTF

   NXT4:   CJNE A,#1111 0101B,MAIN        ; 5
            AJMP STP
         ;------------------------------
    STP:    MOV P0,#1111 1111B
            ACALL DELAY
             AJMP MAIN

        ;-------------------------------

          FWDF:
         MOV P0,#1111 1111B
            ACALL DELAY
         MOV P0,#1111 1110B
           AJMP MAIN
```

```
 ;-------------------------------

 BWDF:
   MOV P0,#1111 1111B
   ACALL DELAY
  MOV P0,#1111 1101B
   AJMP MAIN
 ;-------------------------------

  LFTF:
   MOV P0,#1111 1111B
     ACALL DELAY
  MOV P0,#1111 1011B
    AJMP MAIN
  ;-------------------------------

   RGTF:
  MOV P0,#1111 1111B
   ACALL DELAY
 MOV P0,#1111 0111B
    AJMP MAIN
 ;-----------------------------

 DELAY:
      MOV R7,#05H
RPT3:  MOV R6,#255
RPT2:  MOV R5,#255
RPT1:  DJNZ R5,RPT1
    DJNZ R6,RPT2
      DJNZ R7,RPT3
      RET
```

Appendix D

Code For Voice Automated Mobile Robot

```
using Microsoft.Win32;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using SpeechLib;
using System.IO;
using System.Threading;
using System.Diagnostics;

namespace speechtotext
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      public void listener_Reco(int StreamNumber, object StreamPosition,
SpeechRecognitionType RecognitionType, ISpeechRecoResult Result)
      {

         string heard = Result.PhraseInfo.GetText(0, -1, true);

         textBox1.Text = heard;
         FileInfo path = new FileInfo(Application.ExecutablePath);
         string new_path = path.Directory.FullName + "\\" + heard + ".wav";
         path = new FileInfo(new_path);
         if (path.Exists)
         {
            Process.Start(path.FullName, "wmplayer.exe");
         }
      }



      private void button1_Click(object sender, EventArgs e)
```

```
        {
          // Speech Recognition Object

          SpSharedRecoContext listener;


          // Grammar object

          ISpeechRecoGrammar grammar;

          listener = new SpeechLib.SpSharedRecoContext();

          listener.Recognition += new
_ISpeechRecoContextEvents_RecognitionEventHandler(listener_Reco);

          grammar = listener.CreateGrammar(0);

          grammar.DictationLoad("", SpeechLoadOption.SLOStatic);
          grammar.DictationSetState(SpeechRuleState.SGDSActive);

        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }


    }
}
```

Appendix E

Code For Gesture Automated Mobile Robot

```
namespace motion
{
   using System;
   using System.Drawing;

   using AForge.Imaging;
   using AForge.Imaging.Filters;
   using System.Collections;
   using System.Runtime.InteropServices;
   using System.Reflection;
   using Tiger.Video.VFW;
   using System.Windows.Forms;
   using System.Diagnostics;
   using System.IO;
   /// <summary>
   /// MotionDetector1
   /// </summary>
   public class MotionDetector1 : IMotionDetector
   {
      private IFilter grayscaleFilter = new GrayscaleBT709();
      private Difference differenceFilter = new Difference();
      private IFilter thresholdFilter = new Threshold(15, 255);
      private IFilter erosionFilter = new Erosion();
      private Merge mergeFilter = new Merge();

      private IFilter extrachChannel = new ExtractChannel(RGB.R);
      private ReplaceChannel replaceChannel = new ReplaceChannel(RGB.R);

      private FiltersSequence processingFilter = new FiltersSequence();

      private MainForm _mForm;
      private ArrayList points = new ArrayList();

      private int blankFrameCount = 0; // Blank frame count

      private Bitmap leftArrow, rightArrow, upArrow, downArrow, d1aArrow,
d1bArrow, d2aArrow, d2bArrow, question;
      private Bitmap lastGesture;

      private bool drawGesture = false;
```

```
public MainForm mForm
{
   get { return _mForm; }
   set { _mForm = value; }
}

//// Get a handle to an application window.
//[DllImport("USER32.DLL")]
//private static extern IntPtr FindWindow(string lpClassName,
//    string lpWindowName);

//// Activate an application window.
//[DllImport("USER32.DLL")]
//private static extern bool SetForegroundWindow(IntPtr hWnd);


// Constructor
public MotionDetector1()
{
   processingFilter.Add(differenceFilter);
   processingFilter.Add(thresholdFilter);
   processingFilter.Add(erosionFilter);

   #region Gifs

   Assembly assembly = this.GetType().Assembly;

   leftArrow = new Bitmap(assembly.GetManifestResourceStream(
             string.Format("motion.Resources.left.gif")));


   rightArrow = new Bitmap(assembly.GetManifestResourceStream(
   string.Format("motion.Resources.right.gif")));

   upArrow = new Bitmap(assembly.GetManifestResourceStream(
   string.Format("motion.Resources.up.gif")));

   downArrow = new Bitmap(assembly.GetManifestResourceStream(
   string.Format("motion.Resources.down.gif")));

   d1aArrow = new Bitmap(assembly.GetManifestResourceStream(
   string.Format("motion.Resources.d1a.gif")));

   d1bArrow = new Bitmap(assembly.GetManifestResourceStream(
   string.Format("motion.Resources.d1b.gif")));
```

```
    d2aArrow = new Bitmap(assembly.GetManifestResourceStream(
    string.Format("motion.Resources.d2a.gif")));

    d2bArrow = new Bitmap(assembly.GetManifestResourceStream(
    string.Format("motion.Resources.d2b.gif")));

    question = new Bitmap(assembly.GetManifestResourceStream(
    string.Format("motion.Resources.question.gif")));

    #endregion
}

// Reset detector to initial state
public void Reset()
{
    //if (backgroundFrame != null)
    //{
    //    backgroundFrame.Dispose();
    //    backgroundFrame = null;
    //}
}

// For making beeping sounds
private class Beeper
{
    //[DllImport("Kernel32.dll")]
    //public static extern bool Beep(UInt32 frequency, UInt32 duration);

    public static void RecognizedBeep()
    {
        //Beep(2000, 50);
        //System.Threading.Thread.Sleep(50);
        //Beep(3000, 50);
    }

    public static void UnrecognizedBeep()
    {
        //Beep(256, 200);
    }
}

// Process new frame
public void ProcessFrame(ref Bitmap image)
{
    Utility.UnsafeBitmap uBitmap = new Utility.UnsafeBitmap(image);
```

```
bool brightnessFound = false;

float brightest = 0;
int xPos = 0, yPos = 0;

uBitmap.LockBitmap();

for (int y = 0; y < uBitmap.Bitmap.Height; y += 5)
{
   for (int x = 0; x < uBitmap.Bitmap.Width; x += 5)
   {
      byte red, green, blue;
      red = uBitmap.GetPixel(x, y).red;
      green = uBitmap.GetPixel(x, y).green;
      blue = uBitmap.GetPixel(x, y).blue;

      float brightness = (299 * red + 587 * green + 114 * blue) / 1000;

      if (brightness > _mForm.threshold)
      {
         if (brightness > brightest)
         {
            brightest = brightness;
            xPos = x;
            yPos = y;
            brightnessFound = true;
         }
      } // (brightness > _mForm.threshold)
   } // x loop
} // y loop


if (brightnessFound == true)
   points.Add(new Point(xPos, yPos));
else
   blankFrameCount++;

if (blankFrameCount < 5 && drawGesture == true)
{
   Graphics dc = Graphics.FromImage((System.Drawing.Image)image);
   dc.DrawImage((System.Drawing.Image)lastGesture, 0, image.Height - 68);
   dc.Dispose();
}
else if (blankFrameCount > 5 && drawGesture == true)
{
   drawGesture = false;
```

```
}


if (blankFrameCount > 5)
{
   if (points.Count >= 2)
   {
      Point[] pnts = (Point[])points.ToArray(typeof(Point));
      string gesture = RecognizeMovement(pnts);
      Graphics dc;
      drawGesture = true;
      FileInfo fi = new FileInfo(Application.ExecutablePath);
      string path = fi.Directory.FullName;
      #region Gestures
      switch (gesture)
      {
         case "LEFT":
            path = Path.Combine(path, "Left.wav");
            fi = new FileInfo(path);
            if (fi.Exists)
            {
               Process.Start(path,"wmplayer.exe");
            }
            dc = Graphics.FromImage((System.Drawing.Image)image);
            dc.DrawImage((System.Drawing.Image)leftArrow, 0,
            image.Height - 68);
            dc.Dispose();
            Beeper.RecognizedBeep();
            lastGesture = leftArrow;
            break;

         case "RIGHT":
            path = Path.Combine(path, "Right.wav");
            fi = new FileInfo(path);
            if (fi.Exists)
            {
               Process.Start(path,"wmplayer.exe");
            }
            dc = Graphics.FromImage((System.Drawing.Image)image);
            dc.DrawImage((System.Drawing.Image)rightArrow, 0,
            image.Height - 68);
            dc.Dispose();
            Beeper.RecognizedBeep();
            lastGesture = rightArrow;
            break;
```

```
case "UP":
   path = Path.Combine(path, "Up.wav");
   fi = new FileInfo(path);
   if (fi.Exists)
   {
      Process.Start(path,"wmplayer.exe");
   }
   dc = Graphics.FromImage((System.Drawing.Image)image);
   dc.DrawImage((System.Drawing.Image)upArrow, 0,
   image.Height - 68);
   dc.Dispose();
   Beeper.RecognizedBeep();
   lastGesture = upArrow;
   break;

case "DOWN":
   path = Path.Combine(path, "Down.wav");
   fi = new FileInfo(path);
   if (fi.Exists)
   {
      Process.Start(path, "wmplayer.exe");
   }
   dc = Graphics.FromImage((System.Drawing.Image)image);
   dc.DrawImage((System.Drawing.Image)downArrow, 0,
   image.Height - 68);
   dc.Dispose();
   Beeper.RecognizedBeep();
   lastGesture = downArrow;
   break;

case "DIAGONAL1A":
   path = Path.Combine(path, "Stop.wav");
   fi = new FileInfo(path);
   if (fi.Exists)
   {
      Process.Start(path, "wmplayer.exe");
   }
   dc = Graphics.FromImage((System.Drawing.Image)image);
   dc.DrawImage((System.Drawing.Image)d1aArrow, 0,
   image.Height - 68);
   dc.Dispose();
   Beeper.RecognizedBeep();
   lastGesture = d1aArrow;
   break;

case "DIAGONAL1B":
    path = Path.Combine(path, "Start.wav");
    fi = new FileInfo(path);
    if (fi.Exists)
    {
       Process.Start(path, "wmplayer.exe");
    }
    dc = Graphics.FromImage((System.Drawing.Image)image);
    dc.DrawImage((System.Drawing.Image)d1bArrow, 0,
    image.Height - 68);
    dc.Dispose();
    Beeper.RecognizedBeep();
    lastGesture = d1bArrow;
    break;

case "DIAGONAL2A":
    //Console.WriteLine("DIAGONAL - 2A");

    dc = Graphics.FromImage((System.Drawing.Image)image);
    dc.DrawImage((System.Drawing.Image)d2aArrow, 0,
    image.Height - 68);
    dc.Dispose();
    Beeper.RecognizedBeep();
    lastGesture = d2aArrow;
    break;

case ("DIAGONAL2B"):
    //Console.WriteLine("DIAGONAL - 2B");

    dc = Graphics.FromImage((System.Drawing.Image)image);
    dc.DrawImage((System.Drawing.Image)d2bArrow, 0,
    image.Height - 68);
    dc.Dispose();
    Beeper.RecognizedBeep();
    lastGesture = d2bArrow;
    break;

case ("?"):
    //Console.WriteLine("?");

    dc = Graphics.FromImage((System.Drawing.Image)image);
    dc.DrawImage((System.Drawing.Image)question, 0,
     image.Height -68);
    dc.Dispose();
    Beeper.UnrecognizedBeep();
    lastGesture = question;
    break;
```

```
                }
                #endregion

                //if ((gesture != "?") && (_mForm.controlWMP == true))
                //    ControlMediaPlayer(gesture);

            }

            points.Clear();

            blankFrameCount = 0;
        }

        uBitmap.UnlockBitmap();
        uBitmap.Dispose();
    }


    // Code for controlling Windows Media Player
    private void ControlMediaPlayer(string gesture)
    {
        //IntPtr mediaPlayerHandle = FindWindow("WMPlayerApp", "Windows Media
Player");

        //// Verify that WMP is a running process.
        //if (mediaPlayerHandle == IntPtr.Zero)
        //{
        //    System.Windows.Forms.MessageBox.Show("WMP is not running.");
        //    return;
        //}

        //switch (gesture)
        //{
        //    case "LEFT":
        //        SetForegroundWindow(mediaPlayerHandle);
        //        SendKeys.SendWait("^b");
        //        break;

        //    case "RIGHT":
        //        SetForegroundWindow(mediaPlayerHandle);
        //        SendKeys.SendWait("^f");
        //        break;

        //    case "UP":
        //        SetForegroundWindow(mediaPlayerHandle);
        //        SendKeys.SendWait("^s");
    //        break;

    //    case "DOWN":
    //        SetForegroundWindow(mediaPlayerHandle);
    //        SendKeys.SendWait("^p");
    //        break;
    //}
}


// Laser movement recognition code
private string RecognizeMovement(Point[] pnts)
{
    int x1, y1, x2, y2;
    int length = pnts.Length;

    x1 = pnts[0].X;
    y1 = pnts[0].Y;

    x2 = pnts[length - 1].X;
    y2 = pnts[length - 1].Y;

    int dx = Math.Abs(x2 - x1);
    int dy = Math.Abs(y2 - y1);
    bool diagonal = false;

    if (dx > dy)
    {
        if ((dy == 0) || ((dx / dy) >= 3))
        {
            if ((x2 - x1) > 0)
                return ("RIGHT");
            else
                return ("LEFT");
        }
        else
            diagonal = true;
    }
    else if (dy > dx)
    {
        if ((dx == 0) || ((dy / dx) >= 3))
        {
            if ((y2 - y1) > 0)
                return ("DOWN");
            else
                return ("UP");
                }
                else
                    diagonal = true;

            }
            else
                diagonal = true;


            if (diagonal == true)
            {
                // Recognize diagonal type
                if ((x2 > x1) && (y2 > y1))
                {
                    return ("DIAGONAL1A");
                }
                else if ((x2 < x1) && (y2 < y1))
                {
                    return ("DIAGONAL1B");
                }
                else if ((x2 < x1) && (y2 > y1))
                {
                    return ("DIAGONAL2A");
                }
                else if ((x2 > x1) && (y2 < y1))
                {
                    return ("DIAGONAL2B");
                }
            }

            // If nothing else returned a value...
            return "?";
        }


    }
}
```