SPEECH RECOGNITION SYSTEM

PROJECT REPORT

p-857

Submitted By

KARTHIKEYAN.K
9927K0130

KARTHIKEYAN.V.N
9927K0131

PRAGADEESH KUMAR.M
9927K0152

RAGURAMAN.R
9927K0154

Under The Guidance Of

Mrs D. CHANDRAKALA M.E.,
SENIOR LECTURER

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor Of Engineering
In
Computer Science And Engineering
Of
Bharathiar University, Coimbatore.

DEPARTMENT OF COMPUTER SCIENCE
KUMARAGURU COLLEGE OF TECHNOLOGY
(AFFILIATED TO BHARATHIAR UNIVERSITY)
COIMBATORE-641 006
APRIL - 2003

CERTIFICATE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

Coimbatore – 641006.

PROJECT REPORT 2002-2003

This is to certify that the project report entitled

SPEECH RECOGNITION SYSTEM

is the bonafide work of

KARTHIKEYAN.K *9927K0130*
KARTHIKEYAN.V.N *9927K0131*
PRAGADEESH KUMAR.M *9927K0152*
RAGURAMAN. R *9927KO154*

in partial fulfillment of the requirement for the award of Bachelors Degree in Computer Science and Engineering of Bharathiar University, Coimbatore – 641042.

[illegible] 14/3/'03
Project Guide
(Mrs.D.Chandrakala)

Head of Department
(Prof. S. Thangasamy)

Place: COIMBATORE
Date: 18-03-2003

Submitted for the university exam held on: 18-03-2003

[illegible]
Internal Examiner

[illegible]
External Examiner

DECLARATION

DECLARATION

We

KARTHIKEYAN.K, ***9927K0130***
KARTHIKEYAN.V.N, ***9927K0131***
PRAGADEESH KUMAR.M, ***9927K0152***
RAGURAMAN.R, ***9927K0154***

declare that the project entitled **"SPEECH RECOGNITION (STEM"** is done by us and to the best of our knowledge a similar work has not en submitted earlier to the Bharathiar University or any other institution, for lfillment of the requirement of a course study.

This project report is submitted on the partial fulfillment of the quirements for all awards of degree of Bachelor of Engineering in Computer cience and Engineering of Bharathiar University.

'lace : CBE

)ate : 18·03·2003

KARTHIKEYAN.K

KARTHIKEYAN.V.N

PRAGADEESH KUMAR.M

RAGURAMAN.R

ACKNOWLEDGMENT

ACKNOWLEDGMENT

The exhilaration achieved upon the successful completion of any task ›uld be definitely shared with the people behind the venture. This project is an algam of study and experience of many people without whose help this project ›uld not have taken shape.

At the onset, we take this opportunity to thank the management of our llege for having provided us excellent facilities to work with. We express our ep gratitude to **Dr. K.K. Padmanaban** Bsc(Engg), M.Tech, Ph.D., Principal, ımaraguru college of technology, for ushering us in the path of triumph.

We are greatly indebted to **Prof. Dr. S. Thangaswamy** B.E.(HONS), ı.D., Head of Dept of Computer science and engineering, our guide **rs D.Chandrakala** M.E., Senior Lecturer, **Ms S.Rajini**,B.E,Senior Lecturer, **r. M.N.Gupta** B.E, Lecturer, for achieving us to complete the project by giving nely guidance and constant encouragement throughout our endeavor.

Last but not the least; we thank all our teaching and non-teaching staff ithout whom our project would not have been a success.

Dedicated to my beloved parents, teachers and friends

ABSTRACT

ABSTRACT

Our project aims at developing an automatic speech recognition system. Our ›ctive is to implement Isolated Word Recognition (IWR) using two techniques, ch are:

- Neural networks
- Dynamic Time Warping (DTW)

The crux of the project is speech recognition which is the process by which a nputer identifies spoken words. In our project we recognize isolated words.

The first module is speech recognition using neural networks. In this module ; process of feature extraction is done using Linear Predictive Coding (LPC). ıe working platform is Linux. This system is speaker independent and is an ıline speech recognition process.

The next module is speech recognition using DTW. Here the feature ‹traction is done using Mel-Frequency Cepstrum Coefficients (MFCC). The orking environment is windows and this is an offline process. The system is ɔeaker dependent

Thus from our project we see that we implement most of the concepts under peech recognition like **LPC and MFCC, speaker dependent and independent ystems, online and offline processing, neural network and DTW and two lifferent operating systems.** This is the main advantage of the project and this is where it differs from other projects.

CONTENTS

CONTENTS

INTRODUCTION

INTRODUCTION

PROBLEM STATEMENT:

The purpose of the project is to implement speech recognition system in different operating systems using different techniques. The objective of the project is to identify Isolated words like 'one', 'two', etc, using two techniques:

- Neural networks
- Dynamic time warping(DTW)

The working environments to be used are linux and windows.

LITERATURE SURVEY:

Speech recognition systems are in plenty. There were many previous systems that implemented speech recognition in many applications. The previously developed projects were mostly developed in windows OS. These systems were not much accurate. Some of the techniques were accurate in some respective fields. For example the DTW technique was accurate only in some environments, for other systems neural networks were used. So the proposed system implements both these techniques under different environments. By doing so we would be able to explore the different types of problems that could be encountered while implementing these techniques.

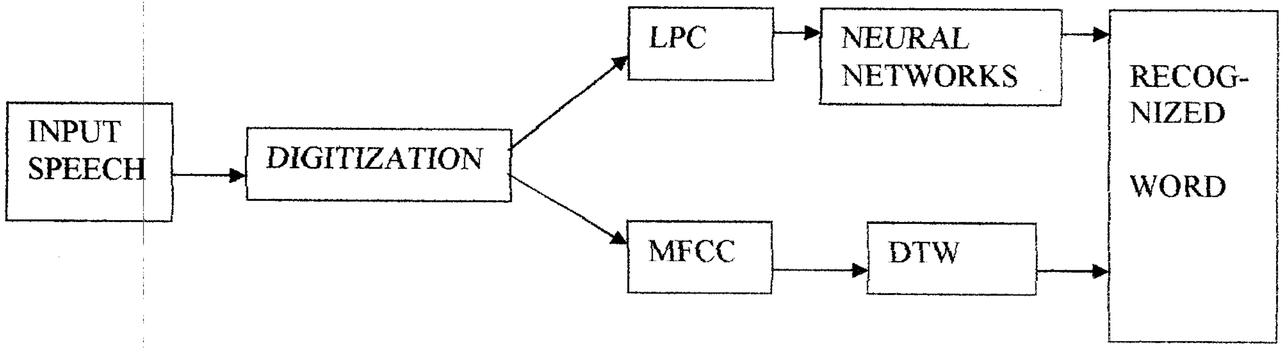
SYSTEM ANALYSIS

SYSTEM ANALYSIS

Proposed system:

Our system aims at developing a speech recognition system. Towards completing the project our first step will be to obtain the input speech from the users and to digitize to input speech signal. The next step is to extract the features which uniquely represent the word using the two types of Cepstral analysis, viz, MFCC and LPC.

Then we store the features and for the search and match of the speech input for recognition we propose to use two techniques which are, Neural networks and DTW. A general overview can be obtained from the picture below.



REQUIREMENTS SPECIFICATION

REQUIREMENTS SPECIFICATION

External Interface Requirements:

- *User interface*: accomplished via keyboard input.
- *Hardware interaction:* The project needs a system with the following hardware configuration: a PC with atleast a pentium processor, 32 MB of RAM and a 640 X 480 screen resolution with a minimum of 256 color, multimedia kit with good quality microphone and speakers. Hared disk space is of 4 MB and above is ideal.
- *Software interaction:* the project requires MATLAB and C language in linux.

Input data requirements:

- The project shall be able to detect voices only when good quality microphones are used in a noise free room.
- The project shall be able to handle data inputs only from six users.

Output data requirements:

The results of the following phases will be displayed as output to the user:

- Pre-emphasis
- End point location
- Windowing
- Auto correlation
- Cepstrum analysis
- Recognized word is displayed

These outputs will be displayed in the MATLAB.

SYSTEM DESIGN

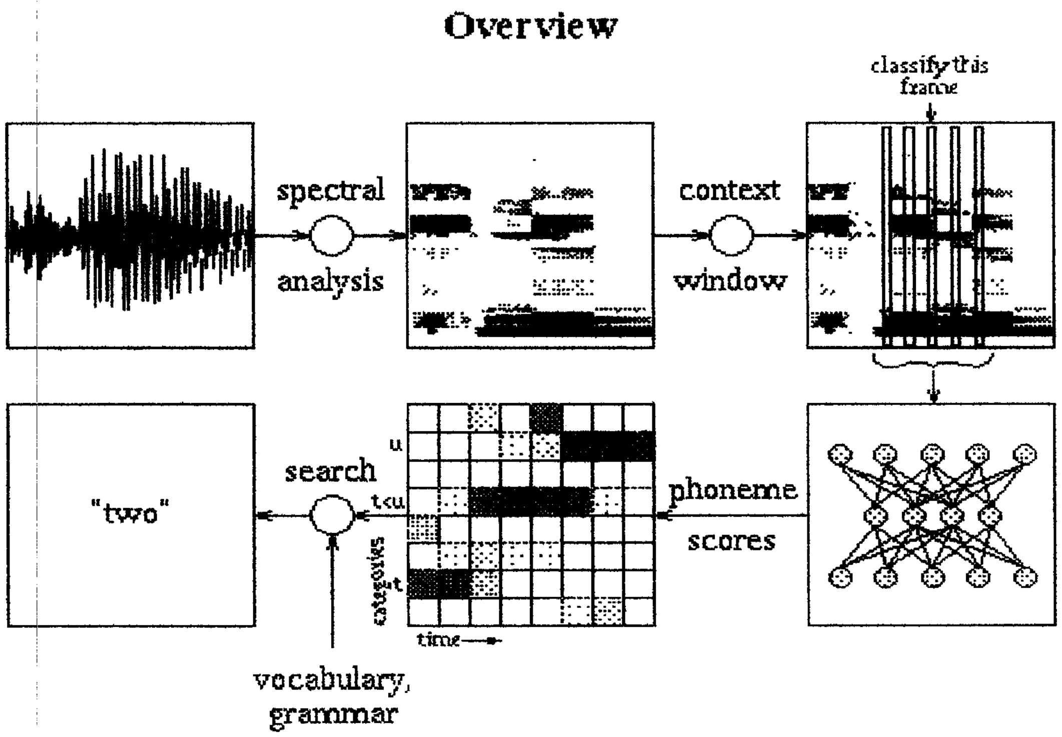
SYSTEM DESIGN

MODULAR DESIGN

The project consists of two modules as already discussed. In this documentation we give a deeper explanation to the modules. This project implements the speech recognition using two techniques which are taken up in the two modules.

MODULE 1:

A general description of the module is given by the following diagram:



Audio input

The input to the speech recognition system developed is from the microphone. The data is retrieved in the PCM format directly from the sound card buffers. This data is actually a sequence of amplitude values. In order to configure the sound card the parameters like *sampling rate, sample format, data transfer rate, number of channels, buffer size, device handle, number of devices* are to be set.

Cepstral analyzer

The cepstral analyzer implements the pre-emphasizing, normalization, windowing and frame-blocking operations on the input data before obtaining the cepstral coefficients.

Frame Blocking:

The continuous speech signal is blocked into frames of N samples, with adjacent frames being separated by M ($M < N$). The 1st frame consists of the first N samples. The 2nd frame begins M samples after the 1st frame, and overlaps it by $N - M$ samples. Similarly, the 3rd frame begins $2M$ samples after the 1st frame (or M samples after the 2nd frame) and overlaps it by $N - 2M$ 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 ~ 30ms windowing and facilitate the fast radix-2 FFT) and $M = 100$.

Windowing

The next step 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

$$y_1(n)=x_1(n)w(n),\ 0 \le n \le N-1$$

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

$$W(n) = 0.54 - 0.46 \cos(2\pi n/(N-1)),\ \ 0 \le n \le N-1.$$

Fast Fourier Transform (FFT)

FFT converts each frame 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:

$$Xn= \sum_{K=0}^{N-1} X_k\, e^{-2\pi jkn/N} \quad n = 0,1,2.....N-1$$

The resulting sequence *{X*n*}* is interpreted as follow: the zero frequencycorresponds to $n = 0$, positive frequencies $0<f<f_s/2$ correspond to values$1 \le n \le N-1$, while negative frequencies $-f_s/2 < f < 0$ correspond to$N/2+1 \le n \le N-1$.

Linear Prediction Cepstrum Coefficients

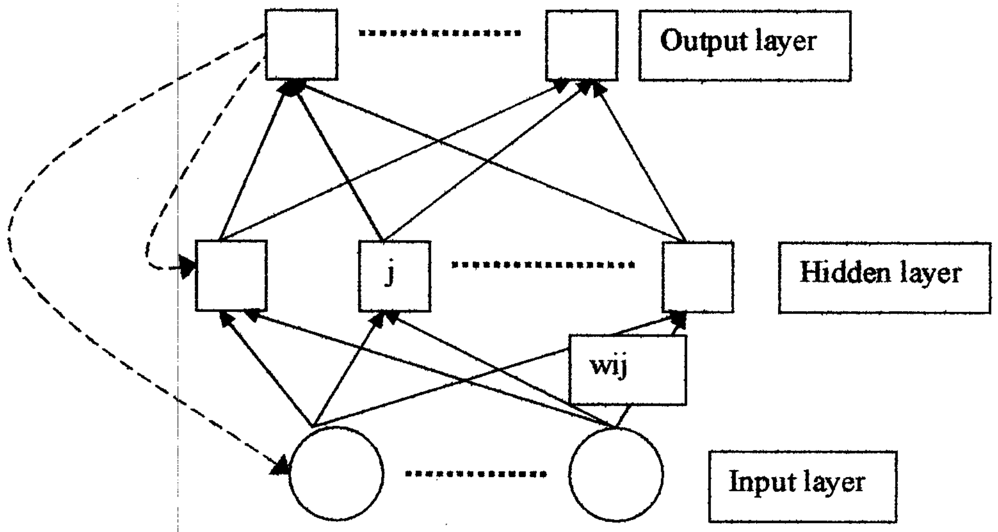
Linear Prediction Cepstrum Coefficients are Linear Prediction Coefficients (LPC) represented in the cepstrum domain. The idea of LPC is based on the speech production model which the characteristic of the vocal tract can be modeled by an all-pole filter. LPC is simply the coefficients of this all-pole filter and is equivalent to the smoothed envelope of the log spectrum of the speech. LPC can be calculated either by the autocorrelation or covariance methods directly from the windowed portion of speech and the LPCC
[3] were acquired from the LPC

$$LPCCi = LPCi + \sum (k-i)/i \ LPCC_{i-k}LPC_k$$

LPCC have been widely used for a few decades and it has been proven that it is more robust and reliable than LPC. However LPCC has also inheriting the disadvantages from LPC. One of the main disadvantages is that LPC approximates speech linearly at all frequencies. This is inconsistent with the perception of human hearing. Also LPC includes the details of the high frequency portion of a speech where containing mostly noise. This inclusion of noise information may affect the system performance.

Basic concepts of neural networks:

A neural network has a parallel distributed architecture with a large number of nodes and connections. Each connection points from one node to another and is associated with a weight.



Construction of neural networks involves the following tasks:

1. Determine the network properties: the network topology (connectivity), the types of connections, the order of connections, and the weight range.
2. Determine the node properties: the activation range and the activation (transfer) function.
3. Determine the system dynamics: the weight initialization scheme, the activation-calculating formula, and the learning rule.

Network Properties:

The topology of a neural network refers to its framework as well as its interconnection scheme. The framework is specified by the no. of layers and the no. of nodes per layer. The types of layer include:

1. The input layer: The nodes in it are called input units, which encode the instance presented to the network for processing. For example, each input unit may be designated by an attribute value possessed by an instance.
2. The hidden layer: The nodes in it are called hidden units, which are not directly observable and hence hidden. They provide nonlinearities for the network.
3. The output layer: The nodes in it are called output units, which encode possible concepts to be assigned to the instance under consideration.

According to the interconnection scheme, a network can be either feed forward or recurrent or its connections either symmetrical or asymmetrical.

1. *Feedforward networks*: All connections point in one direction.
2. *Recurrent networks*: There are feedback connections or loops.

3. *Symmetrical connections:* If there is a connection pointing from node I to node j, then there is also a connection from node j to node I, and the weights associated with the two connections are equal, or notationally, Wij=Wji.
4. *Asymmetrical connections*: If connections are not symmetrical.

Node properties:

The activation levels of input units need not be calculated since they are given. Those of hidden and input units are calculated according to the activation function used. Provided that it is a sigmoid function, the activation level (Oj) of unit j is calculated by

$$\mathbf{Oj = 1/[1+e\text{-}(\Sigma i\ Wij.Xi - \text{Ø}j)]}$$

Where Xi – input from input unit.

Wij - weight on the connection from unit I to unit j.

Øj - threshold on unit j.

In the hard limiting activation function, the output of a neuron is given by

$$\mathbf{Oj = \begin{cases} 1, & \Sigma i\ Wij.Xi > \text{Ø}j \\ 0, & \text{else} \end{cases}}$$

Inference and Learning:

Building an AI system based on the neural network approach will generally involve the following steps:

1. Select a suitable neural network model based on the nature of the problem.
2. Construct a neural network according to the characteristics of the application domain.
3. Train the neural network with the learning procedure of the selected model.
4. Use the trained network for making inference or solving problems.

Single-Layer Perceptron:

A single layer perceptron consists of an input and an output layer. The activation function employed is a hard-limiting function. An output unit will assume the value 1 if the sum of its weighted inputs is greater than its threshold.

Multilayer Perceptrons:

A multilayer perceptron is a feedforward neural network with at least one hidden layer. It can deal with nonlinear classification problems because it can form more decision regions. Each node in the first layer can create a hyperplane. Each node in the second layer can combine hyperplanes to create convex decision regions. Each node in the third layer can combine convex regions to form concave regions.

Supervised and Unsupervised Learning:

The distinction between supervised and unsupervised learning depends on whether the learning algorithm uses pattern-class information. Supervised learning assumes the availability of a teacher or supervisor classifies the training examples into classes, whereas unsupervised learning does not.

Thus, unsupervised learning must identify the pattern-class information as a part of the learning process. The task of unsupervised learning is more abstract and less defined. Unsupervised learning algorithms use unlabeled instances. They blindly or heuristically process them. Unsupervised learning algorithms have less computational complexity and less accuracy than supervised learning algorithms. Unsupervised algorithms can be designed to learn rapidly. This makes unsupervised learning practical in many high-speed, real time and environments, where we may not have enough time and information to apply supervised techniques. Unsupervised learning has also been used for scientific discovery.

Supervised learning algorithms utilize the information on the class membership of each training instance. This information allows supervised learning algorithms to detect pattern misclassifications as a feedback to themselves. Error information contributes to the learning process by rewarding accurate classifications and punishing misclassifications – a process known as credit and blame assignment. It also helps eliminate implausible hypotheses.

Neural Network Learning:

The neural network has also been dubbed the connectionist. It contains a large number of simple neuron like processing elements and a large number of weighted connections between the elements. The weights on the connections encode the knowledge of a network. It uses a highly parallel, distributed control, and can learn to adjust itself automatically.

Backpropagation:

The backpropagation network is probably the most well known and widely used among the current types of neural network systems available. In contrast to earlier work on perceptrons, the backpropagation network is a multilayer feedforward network with a different transfer function in the artificial neuron and a more powerful learning rule. The learning rule is known as **backpropagation**. The training instance set for the network must be presented many times in order for the interconnection weights between the neurons to settle into a state for correct classification of input patterns. While the network can recognize patterns similar to those they have learned, they do not have the ability to recognize new patterns. This is true for all supervised learning networks. In order to recognize new patterns, the network needs to be retrained with these patterns along with previously known patterns. If only new patterns are provided for retraining, then old patterns may be forgotten. In this way, learning is not incremental over time. This is major limitation for supervised networks. Another limitation is that the backpropagation network is prone to local minima.

The backpropagation network learns a mapping from a set of input patterns (e.g., extracted features) to a set of output patterns (e.g., class information). This network can be changed and trained to accomplish a wide variety of mappings. This ability comes from the nodes in the hidden layer or layers of network which learn to respond to features found in the input patterns. The features recognized or extracted by the hidden units (nodes) correspond to the correlation of activity among different input patterns. As the network is trained with different with different examples, the network has the ability to generalize over similar features found in different patterns. The key issue is that the hidden units must be trained to extract a sufficient set of general features applicable to both seen and unseen instances. To achieve this goal, at first, the network must not be over trained. Overtraining the network will make it memorize the individual input output training pairs rather than settling in the mapping for all cases. To prevent this undesired effect, one way is to terminate training once a performance plateau has been reached. Another way is to prune the network, creating a bottleneck between the input and output layers. The bottleneck will force the network to learn in a more general manner.

The backpropagation network is capable of approximating arbitrary mapping given a set of examples. Furthermore, it can learn to estimate posterior probabilities (P (wi|X)) for classification. The sigmoid function guarantees that the outputs are bounded between 0 and 1. In the multiclass case, it is not difficult to train the network so that the outputs sum up to 1. with accurate estimation of posterior probabilities, the network can act as a Bayesian classifier.

The backpropagation network consists of one input layer, one output layer, and one or more hidden layers. If the input pattern is described by n bits or n values, then there should be n input units to accommodate it. The network is fully connected between and only between adjacent layers.

Backpropagation Algorithm:

- Weight Initialization:

Set all weights and node thresholds to small random numbers. Note that the node threshold is the negative of the weight from the bias unit. (Whose activation level is fixed at 1).

- Calculation of Activation:

1. The activation level of an input unit is determined by the instance presented to the network.
2. The activation level Oj of a hidden and output unit is determined by

$$Oj=F(\Sigma\ Wji.Oi-\Theta j)$$

Where Wji is the weight from an input Oi, Θj is the node threshold and F is a sigmoid function:

$$F(a)=1/(1+e-a)$$

- Weight Training:

1. Start at the output units and work backward to the hidden layers recursively. Adjust weights by

$$Wji(t+1)=Wji(t)+\Delta Wji$$

Where Wji(t) is the weight from unit I to unit j at time t (or *t*th iteration) and ΔWij is the weight adjustment.

1. The weight change is computed by

$$\Delta Wji=\eta\delta jOi$$

Where η is a trial independent learning rate ($0<\eta<1$,e.g., 0.3) and δj is the error gradient at unit j. convergence is sometimes faster by adding a momentum term:

$$Wji(t+1)Wji(t)+ \eta\delta jOi+\alpha[Wji(t)-Wji(t-1)]$$

Where $0<\alpha<1$.

2. The error gradient is given by:

   - For the output units:

     $$\delta j=Oj(1-Oj)(Tj-Oj)$$

     Where Tj is the desired output activation and Oj is the actual output activation at output unit j.

   - For the hidden units:

     $$\delta j=Oj(1-Oj)\Sigma\ \delta k\ Wkj$$

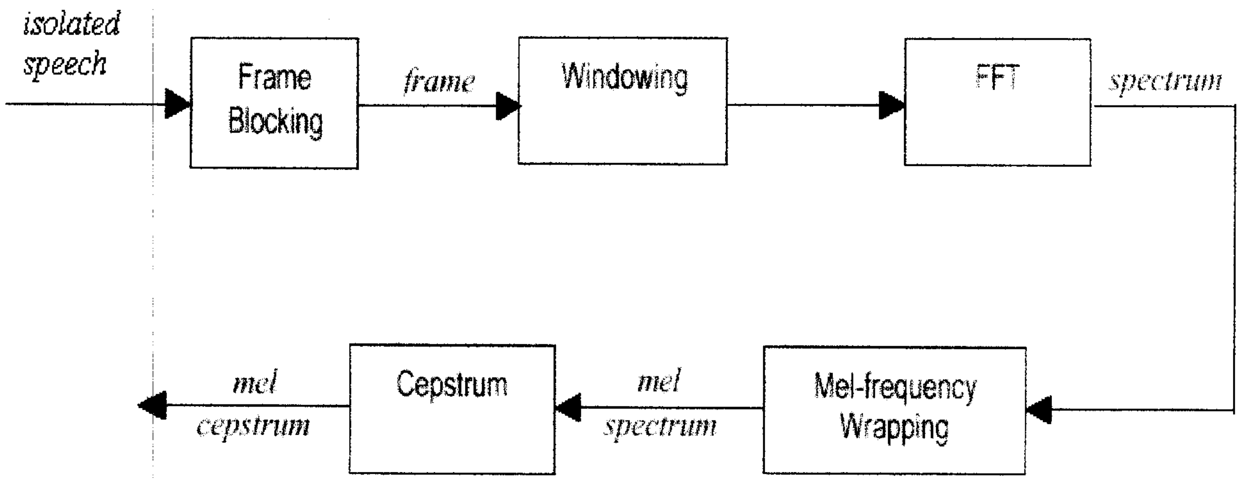
     Where δk is the error gradient at unit k to which a connection points from hidden unit j.

3. Repeat iteration until convergence in terms of the selected error criterion. Iteration includes presenting an instance, calculating activations, and modifying weights.

The name "backpropagation" comes from the fact that the error of hidden units are derived from propagating backward the errors associated with output units since the target values for the hidden units are not given. In the backpropagation network, the activation function chosen is the sigmoid function, which compresses the output value into the range between 0 and 1. The sigmoid function is advantageous in that it can accommodate large signals without saturation while allowing the passing of small signals without excessive attenuation. Also, it is a smooth function so that gradients can be calculated, which are required fir a gradient descent search.

MODULE II:

The first few steps are similar to the first module. They are audio input, pre-emphasizing, normalization, frame blocking and windowing. The next step is the feature extraction for which we follow the MFCC. The general diagram can be represented as follows:



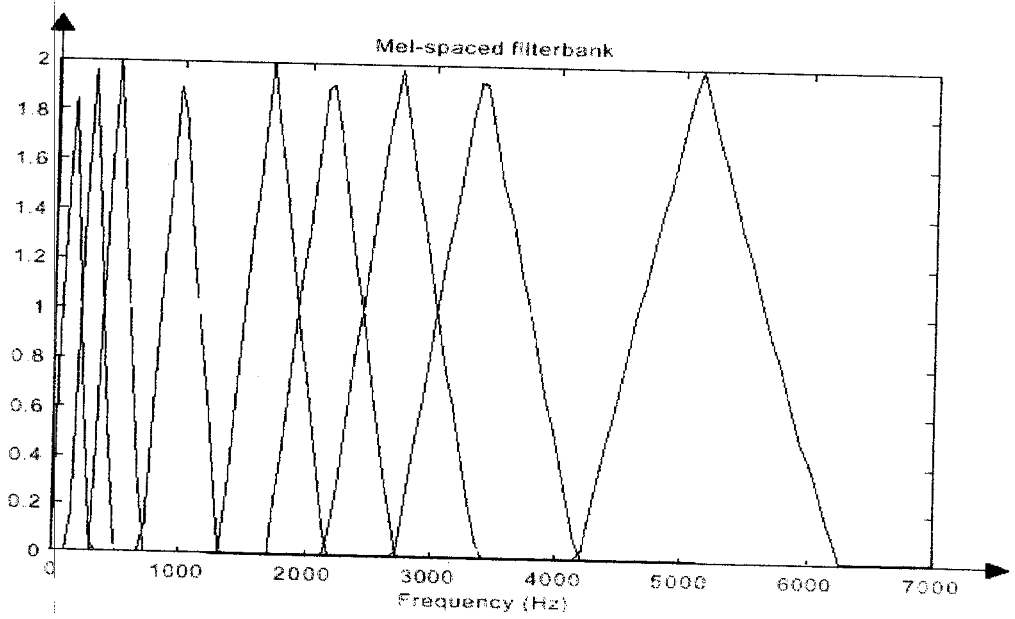
Mel-frequency wrapping:

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 1 KHz and a logarithmic spacing above 1 KHz. 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(1+f/100)

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 band pass frequency response, and the spacing as well as the bandwidth is determined by a constant mel frequency interval. The modified spectrum of S(w) thus consists of the output power of these filters when S(w) 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 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.

mel scale representation



Cepstrum:

Finally the log mel spectrum is converted 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 $S_k, k=1,2,\ldots,K$, we can calculate the MFCC's, , C_n as

$$C_n=\Sigma(\log S_k)\cos[n(k-1/2)(\pi/k)]\ ,\ n=1,2,\ldots k.$$

Note that we exclude the first component, C_o from the DCT since it represents the mean value of the input signal which carried little speaker specific information.

Dynamic Time Warping:

Dynamic programming is an approach to the implicit storage of all possible solutions to problem solutions to a problem requiring the minimization of a global error criterion. It is formulated as a sequential optimization strategy in which the current estimate of the global error function is updated for each possible step. At each step, enough information about the plausible hypotheses are retained so that at the end, when the best global error value is found, the corresponding set of choices that correspond to this value can also be discovered.

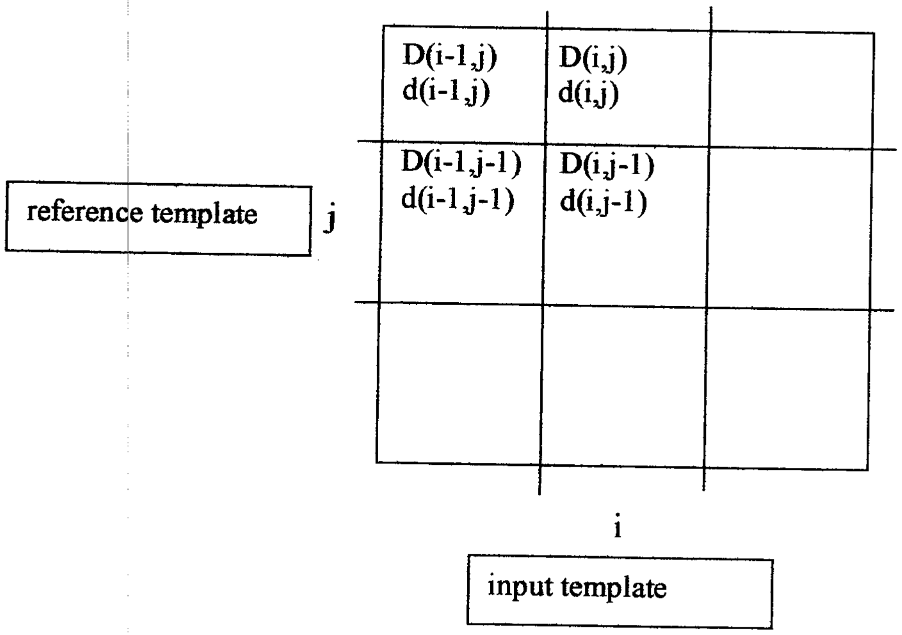
Applied to template matching for speech recognition, this algorithm can be stated fairly simple. Imagine a matrix D in which the rows correspond to frames of a reference template and the columns to an input template. For each matrix element in D we will define a cumulative distortion measure,

$$D(i,j)=d(i,j)\min_{p(i,j)}\{D[p(i,j)]+T[(i,j),p(i,j)]\}$$

Where d is a local distance measure between frames i of the input and frame j of the reference template.

p (i,j) is the set of possible predecessors to i, j; in other words, the coordinates of the possible previous points on the matching trajectory between the two templates.

The T () is a term for the cost associated with any particular transition. Thus, each matrix element is the value of the total error that arises from the best step that could lead to associating those two frames, and since this step is made after a similar optimal decision, the best cumulative distance in the final column (corresponding to the last frame in the input template) will be the distortion corresponding to the best match between reference and input. For isolated word recognition, the reference with the lowest value will be taken as the best match. The basic computational step is illustrated in figure.



Thus, the algorithm consists of the following steps.

1. Compute the local distance for each element in column 1 of each distortion matrix (that is, the distance between frame 1 of the input and all the frames of each reference template). Call this the cumulative distortion value for that matrix element.
2. starting with frame 2 of the input, and beginning with the bottom row (frame 1 of the first reference template), compute the local distance and add it to the best cumulative distortion value for all possible predecessors (that is, all possible matches between input and reference that could temporally precede either the current input frame or the current reference frame). Compute this value for each element in the column for each reference template.
3. Continue this operation through each of the other columns.
4. Find the best distortion number in the last column for each reference template and declare it the distortion associated with that reference.
5. Choose the word associated with best of the reference distortions and declare it the winner.

Since, this algorithm applies a dynamic programming approach to the time warp problem; it is often referred to as a dynamic time warp, or DTW.

IMPLEMENTATION

IMPLEMENTATION

Since we have implemented speech recognition in two different operating systems the implementation details differ. Hence, we look into the implementation details module by module.

MODULE I:

In this module we use Linux OS as the platform. We use C for programming purposes. For input audio processing and for signal processing we use the following user-defined functions:

set_snd():

This function is used to set the following values to the sound card:

Sampling frequency	: 8000 Hz
Sampling format	: PCM (pulse coded modulation)
No of bits per sample	: 8-bit
No of channels	: 1(mono)

speech_process():

This function is used to perform the preliminary processing of the input speech functions like pre-emphasizing, end-point detection and noise reduction.

The following functions are used to perform the various neural network processes:

bpnn_create:

This function is used to create a network of structure BPNN

BPNN *bpnn_create(int n_in, int n_hidden, int n_out)

where,

n_in – no of input units

n_hidden – no of hidden units

n_out – no of output units

bpnn_initialize:

This function is used to initialize the created BPNN.

void bpnn_initialize(seed)

where,

seed – random number

bpnn_layerforward:

This function is used to process the network and identify the variation in the network.

void bpnn_layerforward(double *l1,double *l2,double **conn, int n1,int n2)

***bpnn_adjust_weights*:**

This function adjusts the weights according to the error identified in the previous function.

void bpnn_adjust_weights(double *delta, int ndelta, double *ly, int nly, double **w, double **oldw, double eta, double momentum)

bpnn_train():

This function is used to train the neural network.

void bpnn_train(BPNN *net,double eta, double momentum, double *eo,double *eh)

bpnn_read():

This function is used to read the details from the network and identify the word.

BPNN *bpnn_read(char *filename)

MODULE II:

In this module we use the windows98 OS as the platform. For programming we use MATLAB 5.3.

MATLAB has wonderful features like built-in functions for digital signal processing, getting audio input, to perform high-speed numerical computation and visualization. There are also options to display colorful pictures and graphs which help in viewing the energy diagram of various audio inputs. The programming using MATLAB is also easily adaptable for persons who already have proficiency in other programming languages like C, C++, etc.

First we will look into audio specification before we go into implementation details.

Specification for Input Audio data:

Sampling frequency	:	8000 Hz
Sampling format	:	PCM (pulse coded modulation)
No of bits per sample	:	8-bit
No of channels	:	1(mono)

For sound processing we use certain built-in functions in MATLAB which are as follows:

SOUNDSC(Y,...) is the same as SOUND(Y,...) except the data is scaled so that the sound is played as loud as possible without clipping. The mean of the data is removed.

[Y, FS, NBITS]= WAVREAD (FILE) returns the sample rate (FS) in Hertz and the number of bits per sample (NBITS) used to encode the data in the file.

HAMMING (N) returns the N-point symmetric Hamming window in a column vector.

HAMMING (N, SFLAG) generates the N-point Hamming window using SFLAG window sampling. SFLAG may be either 'symmetric' or 'periodic'. By default, a symmetric window is returned.

FFT(X) is the discrete Fourier transform (DFT) of vector X. For matrices, the FFT operation is applied to each column. For N-D arrays, the FFT operation operates on the first non-singleton dimension.

The following user-defined programs were programmed to obtain certain signal processing functions which are as follows:

enframe:

ENFRAME split signal up into (overlapping) frames: one per row.

F = ENFRAME(X,LEN) splits the vector X up into frames. Each frame is of length LEN and occupies one row of the output matrix. The last few frames of X will be ignored if its length is not divisible by LEN. It is an error if X is shorter than LEN.

F = ENFRAME(X,LEN,INC) has frames beginning at increments of INC. The centre of frame I is X((I-1)*INC+(LEN+1)/2) for I=1,2,...
The number of frames is fix((length(X)-LEN+INC)/INC)

endpoint_detect:

This function is used to identify the useful portion of the signal. This is done by calculating the energy levels of the signal.

r1 = endpoint_detect(y)

r1 - useful speech samples.

y - The original speech samples.

mfcc:

This function is used to calculate the mel-frequency cepstrum coefficients for the given sample.

ccep=mfcc(y,M,N,P)

Calculates cepstral coefficients for sequence y, using window length N, window step size M, and order P.

dtwr2:

This function is used to compute the accumulated distortion between two given vectors (acoustic templates) using dynamic time warping technique, to find the number of vectors finding their places in test and reference templates and to allocate the number of vectors of reference and test templates in 'i' &'j' respectively.

acc_globdist=dtwr2(test,ref)

where,

test	– input template
ref	–reference template
acc_glbdist	–accumulative global distance between the input and reference template

euclidean:

This function compares the input and reference template and finds the Euclidean distance between them which helps to find out whether the dtw function can be applied to the set of input and reference template.

y=L2norm(a,b,test,ref)

training session:

This function is used to identify the reference templates which are relative to each other. The templates which vary to a large distance are removed. This function is independent one.

testing session:

This function is used to identify the spoken word. This function calls all the functions which are described above.

SYSTEM TESTING

SYSTEM TESTING

Testing is an activity to verify that a correct system is being built and is performed with the indent of finding fault in the system. Testing is an activity however not restricted to being performed after the development phase is completed. But this is to be carried out in parallel with all stages of system development, starting with requirements specification.

While working with a speech recognition system we need to keep testing the following parts of programming:

Sound cards

Different sound cards and driver versions: Make sure to test on different sound cards and driver versions. The variation in the sound cards, mixer and quality might cause problems with an application. Make sure that the device driver handle specified in the program is correct.

Test for sound card: Identify if the user does not have a sound card, or the user's sound card can't handle IWR, and inform the user.

More than one sound card: The user might have more than one sound card. Make sure they can choose the right one

More than one speech applications: Make sure that there are no more than one speech applications that can access the sound card at the same time.

Usability

Can the users get their microphone working? - does extensive testing to make sure the user can plug in their microphone and use them properly.

Are the settings adjusted? – make sure that the microphone input is enabled in the settings for sound devices. Also in the settings for the microphone volume make sure it is not muted and volume is properly set.

Miscellaneous

Proper machine speed- Some machines require Pentium-speed or better to function. Inform the user if their PC is too slow for speech recognition or text to speech. Some older Pentiums with faulty math coprocessors will have their math coprocessor disabled by Windows NT. If the speech engine relies on floating point, it may be too slow on these machines.

Noiseless environments- make sure to work in noiseless environment for better accuracy and performance

CONCLUSION

CONCLUSION

To summarize our project primarily dealt with providing speech recognition utilities. The human voice is considered the most common form of communication. Using this form of communication to interact with the computer motivated us to undertake this project.

This project is a high-level overview of how speech recognition system works. It is not nearly enough detail to actually write a speech recognizer, but it exposes the basic concepts. Most speech recognition systems work in a similar manner, although not all of them work this way. Recognition accuracy can be affected by regional dialects, quality of the microphone and the ambient noise level during speech session.

On finishing the project we observed that it included most of the topics under speech recognition, like DTW and neural networks, MFCC and LPC, offline and online, different operating systems, etc. thus we were able to summarize speech recognition on the whole. And thus we could have an overview of all speech recognition systems available.

FUTURE ENHANCEMENTS

FUTURE ENHANCEMENTS

There are many factors involved in speech recognition. Although speech recognition technology seems relatively new, computer scientists have been continuously developing it for the past 40 years. They have made great strides in improving the systems and processes, but the futuristic idea of the computer hearing and understanding we are still a long way off. However, there are numerous ongoing projects that deal with topics such as the following:

- Visual cues to help computers decipher speech sounds that are obscured by environmental noise.
- Speech to text translation project for spontaneous speech.
- Building synthetic voices.
- Designing web browsers to be speech enabled.

Due to time constraints we were able to finish only up to the speech recognition process. This could be further used to control electrical and other appliances using the telephone. Also the computer operations can be executed using vocal commands. These further developments require more time and other constraints to be overcome.

ANNEXURE

ANNEXURE

SOURCE CODE LISTING:

main.c

```
#include<stdio.h>
#include <stdlib.h>
#include<time.h>
#include <math.h>
#include<linux/soundcard.h>
#include<fcntl.h>
#include<sys/ioctl.h>
#include"backprop.h"
#define USERS 6
#define ORDER  10
#define COEFF  10
#define sgn(x) ((x<0)?-1:1)
#define ABS(x) ((x<0.0)?x*-1.0:x)
#define P      10000
#define WLEN   300
#define M      256
#define N      1024
#define Q      (P/WLEN)
#define KK  150

int set_snd();
/*** Return random number between 0.0 and 1.0 ***/
double drnd(){  return ((double) random() / (double) BIGRND);
}
bpnn_randomize_weights(double **w,int m,int n)
{
 int i, j;
 for (i = 0; i <= m; i++) {
   for (j = 0; j <= n; j++) {
     w[i][j] = dpn1();
```

```
  } }}
bpnn_zero_weights(double **w,int m,int n)
{
 int i, j;
 for (i = 0; i <= m; i++) {
   for (j = 0; j <= n; j++) {
     w[i][j] = 0.0;
   } }}
void bpnn_initialize(seed)
{
 srandom(seed);
}
/* Call this fun to create a bpn */
BPNN *bpnn_create(int n_in,int n_hidden,int n_out)
{
 int i,j;
 BPNN *newnet;
 newnet = bpnn_internal_create(n_in, n_hidden, n_out);
bpnn_randomize_weights(newnet->input_weights, n_in, n_hidden);
 bpnn_randomize_weights(newnet->hidden_weights, n_hidden, n_out);
 bpnn_zero_weights(newnet->input_prev_weights, n_in, n_hidden);
 bpnn_zero_weights(newnet->hidden_prev_weights, n_hidden, n_out);
 return (newnet);
}
void bpnn_layerforward(double *l1,double *l2,double **conn,int n1,int n2)
{
 double sum;
 int j, k;
 /*** Set up thresholding unit ***/
 /*** For each unit in second layer ***/
   for (j = 0; j <= n2; j++) {
   /*** Compute weighted sum of its inputs ***/
   sum = 0.0;
   for (k = 0; k <= n1; k++) {
     sum += conn[k][j] * l1[k];
   }
   l2[j] = squash(sum);
 }}
void bpnn_output_error(double *delta,double *target,double *output,
                                        int nj,double *err)
```

```
{
 int j;
 double o, t, errsum;
 errsum = 0.0;
  for (j = 0; j <= nj; j++) {
  o = output[j];
  t = target[j];
  delta[j] = o * (1.0 - o) * (t - o);
  errsum += ABS(delta[j]);
 }
 *err = errsum;
}
void bpnn_hidden_error(double *delta_h,double *delta_o,double *hidden,
            double **who,double *err,int nh,int no)
{
 int j, k;
 double h, sum, errsum;
 errsum = 0.0;
  for (j = 0; j <= nh; j++) {
  h = hidden[j];              // hidden layer output
  sum = 0.0;
   for (k = 0; k <= no; k++) {
   sum += delta_o[k] * who[j][k];
  }
  delta_h[j] = h * (1.0 - h) * sum;
  errsum += ABS(delta_h[j]);
 }
 *err = errsum;
}
void bpnn_adjust_weights(double *delta,int ndelta,double *ly,int nly,double
**w, double **oldw,double eta,double momentum)
{
 double new_dw;
 int k, j;
 ly[0] = 1.0;                    // ly  is the input array
 for (j = 0; j <= ndelta; j++) {   // oldw last change of weight
  for (k = 0; k <= nly; k++) {    // delta is error
   new_dw = ((eta * delta[j] * ly[k]) + (momentum * oldw[k][j]));
   w[k][j] += new_dw;
   oldw[k][j] =(eta * delta[j] * ly[k]); //new_dw;
   } }}
/* Call this func to feed forward */
void bpnn_feedforward(BPNN *net)
{
  int in, hid, out;
  in = net->input_n;
  hid = net->hidden_n;
  out = net->output_n;
  /*** Feed forward input activations. ***/
  bpnn_layerforward(net->input_units, net->hidden_units,
     net->input_weights, in, hid);
  bpnn_layerforward(net->hidden_units, net->output_units,
     net->hidden_weights, hid, out);
}
void bpnn_train(BPNN *net,double eta,double momentum,double
*eo,double *eh)
{
  int in, hid, out,i,j;
  double out_err, hid_err;
  in = net->input_n;
  hid = net->hidden_n;
  out = net->output_n;
  /*** Feed forward input activations. ***/
  bpnn_layerforward(net->input_units, net->hidden_units,
     net->input_weights, in, hid);
  bpnn_layerforward(net->hidden_units, net->output_units,
     net->hidden_weights, hid, out);

  /*** Compute error on output and hidden units. ***/
  bpnn_output_error(net->output_delta, net->target, net->output_units,
     out, &out_err);
  bpnn_hidden_error(net->hidden_delta,net->output_delta,
     net->hidden_units,net->hidden_weights, &hid_err,hid,out);
  *eo = out_err;
  *eh = hid_err;
  /*** Adjust input and hidden weights. ***/
  bpnn_adjust_weights(net->output_delta, out, net->hidden_units, hid,
     net->hidden_weights, net->hidden_prev_weights, eta, momentum);
  bpnn_adjust_weights(net->hidden_delta, hid, net->input_units, in,
     net->input_weights, net->input_prev_weights, eta, momentum);
```

```
}
void bpnn_save(BPNN *net,char *filename)
{
 int fd, n1, n2, n3, i, j, memcnt;
 double dvalue, **w;
 char *mem;
 if ((fd = creat(filename,0644)) == -1) {
  printf("BPNN_SAVE: Cannot create '%s'\n", filename);
  return;
 }
 n1 = net->input_n;
 n2 = net->hidden_n;
 n3 = net->output_n;
 printf("Saving %dx%dx%d network to '%s'\n", n1, n2, n3, filename);
 fflush(stdout);
 write(fd, (char *) &n1, sizeof(int));
 write(fd, (char *) &n2, sizeof(int));
 write(fd, (char *) &n3, sizeof(int));
 memcnt = 0;
 w = net->input_weights;
 mem = (char *) malloc ((unsigned) ((n1+1) * (n2+1) * sizeof(double)));
 for (i = 0; i <= n1; i++) {
  for (j = 0; j <= n2; j++) {
   dvalue = w[i][j];
   fastcopy(&mem[memcnt], &dvalue, sizeof(double));
   memcnt += sizeof(double);
  }
 }
 write(fd, mem, (n1+1) * (n2+1) * sizeof(double));
 free(mem);
 memcnt = 0;
 w = net->hidden_weights;
 mem = (char *) malloc ((unsigned) ((n2+1) * (n3+1) * sizeof(double)));
 for (i = 0; i <= n2; i++) {
  for (j = 0; j <= n3; j++) {
   dvalue = w[i][j];
   fastcopy(&mem[memcnt], &dvalue, sizeof(double));
   memcnt += sizeof(double);
  }
 }
```

```
  write(fd, mem, (n2+1) * (n3+1) * sizeof(double));
  free(mem);
  close(fd);
  return;
}
BPNN *bpnn_read(char *filename)
{
  char *mem;
  BPNN *new;
  int fd, n1, n2, n3, i, j, memcnt;
  if ((fd = open(filename, 0, 0644)) == -1) {
    return (NULL);
  }
  printf("\nReading '%s'\n", filename);
  fflush(stdout);
  read(fd, (char *) &n1, sizeof(int));
  read(fd, (char *) &n2, sizeof(int));
  read(fd, (char *) &n3, sizeof(int));
  new = bpnn_internal_create(n1, n2, n3);
  printf("'%s' contains a %dx%dx%d network\n", filename, n1, n2, n3);
  printf("Reading input weights..."); fflush(stdout);
 memcnt = 0;
  mem = (char *) malloc ((unsigned) ((n1+1) * (n2+1) * sizeof(double)));
  read(fd, mem, (n1+1) * (n2+1) * sizeof(double));
  for (i = 0; i <= n1; i++) {
    for (j = 0; j <= n2; j++) {
      fastcopy(&(new->input_weights[i][j]), &mem[memcnt], sizeof(double));
      memcnt += sizeof(double);
    }
  }
  free(mem);
  printf("Done\nReading hidden weights..."); fflush(stdout);
 memcnt = 0;
  mem = (char *) malloc ((unsigned) ((n2+1) * (n3+1) * sizeof(double)));
  read(fd, mem, (n2+1) * (n3+1) * sizeof(double));
  for (i = 0; i <= n2; i++) {
    for (j = 0; j <= n3; j++) {
      fastcopy(&(new->hidden_weights[i][j]), &mem[memcnt],
sizeof(double));
      memcnt += sizeof(double);
    }
   }
   free(mem);
   close(fd);
   printf("Done\n");
   fflush(stdout);
   bpnn_zero_weights(new->input_prev_weights, n1, n2);
   bpnn_zero_weights(new->hidden_prev_weights, n2, n3);
   return (new);
 }
double * speech_process(int audio_fd)   {
unsigned char value[P];
float coeff_res[Q][11];
double x[P+1];
int sample=0,sample1,c,set,Lsup=1,skip=0;
int lc;
double xp[P+1],CP=0.9375,pi=22.0/7.0;
double B=0.0,Clp[M],w[M],Cslp[M],G=0.0;
double E[11],R[11],alfa[11][11];
double K[ORDER+1],ist=0.0,finalalfa[ORDER];
int i,j,k,n,a,p=10,xy=0 ;
double fin[1024],avg[1024],xw[P+1],sum;
int start_in,end_in;
double log_energy[Q],temp,temp2;
int num_zeros[Q];
double threshold,endan[P+1];
double *O;
int left_index, right_index;
int num_samples;
int temp1,IZCT,STZCR,pitch;
double Ts,Td,STE,D,MIN,sum1,A,Y[WLEN],S[WLEN/2];
int temp3=0,cnt=0,flag=0;
/****SETTING & READING THE SOUND CARD *****************/
/*  Using this for loop we are reading ten 1024 samples from the sound card
Reading the 1024 samples from card */
     if (a=read(audio_fd,value,P)==-1) {
          printf("Cant Read\n");
        }
  for (sample=0;sample<P;sample++) {
          x[sample]=((double)value[sample]-128.0);
```

```
                  }
  x[P]=0.0;
/** SOUND PROCESSING OF 1024 SAMPLES  **/
/* To check for Noice */
temp=0.0;
for(i=0;i<P;i++)
 temp+=x[i]*x[i];
temp=log(temp);
if(temp<9.0)          // if log energy is less than some value it is noise
         flag=1;
        temp=0.0;
/* Amplitude Normalisation */
   for(i=0;i<P;i++)  {
     if(temp<ABS(x[i]))
        temp=ABS(x[i]);
        }
   for(i=0;i<P;i++)  {
        x[i]=(x[i]/temp);
         }
/*---------------- voiced unvoiced separation --------------------------*/
        start_in=0;
      end_in=(WLEN-1);
        Ts=pow(10.0,-4.0);
        Td=9.0*pow(10.0,5.0);
        num_samples=0;
        MIN=pow(10,10);
        for(i=0;i<Q;i++)
          for(j=0;j<11;j++)
                coeff_res[i][j]=0.0;
for (set=0; set<Q; set++) {
        temp=0.0;
        temp2=0.0;
   for(j=start_in;j<end_in;j++)    {
        temp+=x[j]*x[j];
        temp2+=x[j];
        }
        temp/=(double)WLEN;
        temp2/=(double)WLEN;
        temp2*=temp2;
          STE=temp-temp2;       // Varience of 4 frames  (STE)
     log_energy[set]=STE;  // Log energy
    if(STE<Ts)          {
     start_in+=WLEN;
     end_in+=WLEN;
     printf("Skipping noise \n");
     continue;
  }
/*----------NUM OF ZERO CROSSING --------------------------------*/
     STZCR=0;
    for(j=start_in;j<end_in-1;j++){
     temp1=sgn(x[j+1])-sgn(x[j]);
       if(ABS(temp1)==2)
              STZCR++;
     }

    STZCR=(8000*STZCR)/WLEN;
    num_zeros[set]=STZCR;          // Zero crossing
    D=((double)STZCR)/STE;
    if((D<MIN)&&(D!=0.0))
     MIN=D;
     Td=KK*MIN;
     if(D<Td)  {
    }
    else {
    printf("Unvoiced");
    start_in+=WLEN;
    end_in+=WLEN;
    continue;
     }

/*-------------------- Processing starts Here ---------------------*/
sum1=0.0;

/* Amplitude Normalisation */
printf("\n MAX is : %lf ",temp);
   for(i=0;i<WLEN;i++)  {
       xw[k*WLEN+i]=((xw[k*WLEN+i]/temp)*(128.0));
      }
/*/// Pitch Detection///*/
   A=0.0;
```

```
    for(j=start_in;j<end_in;j++) {
            if(ABS(x[j])>A)
                    A=ABS(x[j]);
    }
    A*=0.3;
    for(a=0,j=start_in;j<end_in;j++,a++) {
            if(x[j]>0.0) {
                 Y[a]=x[j]-A;
                 if(Y[a]<0.0)
                   Y[a]=0.0;
            }                    // end of if
            else {
            Y[a]=x[j]+A;
               if(Y[a]>0.0)
                   Y[a]=0.0;
            }               // end of else
    }
    temp=0.0;
  for(k=0;k<(WLEN/2);k++)  {
          temp=0.0;
    for(j=0;j<WLEN-k-1;j++)
    temp+=Y[j]*Y[j+k];
    S[k]=temp/(double)(WLEN-k);
  }
  temp=0.0;
 for(k=10;k<(WLEN/2);k++)
         if(S[k]>temp)
         {
                 pitch=k;
                 temp=S[k];
         }
  if((pitch<100)&&(pitch>40))  {
         temp3+=pitch;
         cnt++;
  }
/*------------------ hamming window --------------------------------*/
      for(i=start_in,j=0; i<end_in;j++,i++) {
         B=2*3.14159265*j/(WLEN);
      xp[j]=x[i]*(0.54-0.46*cos(B));
            }
```

```
/*---------------------------------- pre-emphasis --------------------------------*/
x[0]=0.0;
     for( i=1; i<WLEN; i++)  {
          x[i]=xp[i]-CP*xp[i-1];
         }
/*-- array-initialisation --------*/
     for (sample=0; sample<=p; sample++) {
         E[sample]=0.0;
         R[sample]=0.0;
         K[sample]=0.0;
   }
  for (sample=0; sample<=p; sample++)  {
          for (sample1=0; sample1<=p; sample1++)
                 alfa[sample1][sample]=0.0;
          }

/*------------- auto correlation -*/
sum1=0.0;
for(i=0; i<=p; i++) {
        R[i]=0.0;
        for(j=1; j<=(WLEN-i); j++)
         R[i]+=x[j]*x[j+i];          //       R[0-10]
          sum1=sum1+ABS(R[i]);
        }
sum1/=10.0;
/*---------------------------------- LPC evaluation --------------------------*/
  alfa[0][0]=0.0;
  E[0]=R[0];       //Fixed
  for(i=1;i<=p;i++)
        {
     ist=0.0;
     for(j=1;j<=(i-1);j++)
         ist+=alfa[j][i-1]*R[i-j];
         K[i]=(R[i]-ist)/E[i-1];
         alfa[i][i]=K[i];
   for(j=1;j<=(i-1);j++)
     alfa[j][i]=alfa[j][i-1]-(K[i]*alfa[i-j][i-1]);
          E[i]=( 1- K[i]*K[i])* E[i-1];
}
 for(j=1;j<=p;j++)  {
  finalalfa[j]=alfa[j][p];  //        finalalfa[1-10]
 }

/*------------- Cepstral analysis ---------------------*/
 for(n=0;n<M;n++)
       Clp[n]=0.0;
 for(n=1;n<=p;n++)
   {
    for(i=1;i<n;i++)
      G+=(i/n)*Clp[i]*finalalfa[n-1];
      Clp[n]=finalalfa[n]+G;              //Clp[1-10]
   }
/*--------- Liftering Process ---------------------*/
  for(n=1;n<=p;n++)  {
   w[n]=1+((WLEN/2)*sin(n*pi/WLEN));
   Cslp[n]=Clp[n]*w[n];          // Cslp[1-10]
  }
  for(i=1;i<=p;i++)
   coeff_res[set][i]=Cslp[i];
   start_in+=WLEN;
   end_in+=WLEN;

}                        // end of processing of 10 - 1024 samples
/**** WRITTING DATA TO A FILE***/
   xy=0;
for(c=0;c<10;c++)
        {
        temp=0.0;
        temp2=0.0;
for(set=0;set<Q;set++) {
      if(coeff_res[set][c]!=0.0)
                temp+=1.0;
        temp2+=coeff_res[set][c];
        }
  if(temp!=0.0)
        avg[xy]=temp2/temp;
  else
          avg[xy]=0.0;
        xy++;
  }
```

```
O=(double*)malloc(10*sizeof(double));
for(c=0;c<10;c++) {
 O[c]=(double)avg[c];
}
if(cnt!=0) {
   O[0]=((double)temp3)/((double)cnt*100.0);      // pitch period average
}
if(flag==1)
   O[0]=0.0;
 return(O);
   }
/*  setting the snd card  */
int set_snd()   {
int speed=8000,format=AFMT_U8;
int audio_fd;
/* Setting the Sampling rate(11025) & Format (8 bit) of
              the Sound Card  and opening it for recording    */
printf("test\n");
if((audio_fd=open("/dev/dsp",O_RDWR,0))==-1)     {
   perror("/dev/dsp");
   exit(1);
   }
  if(ioctl(audio_fd,SNDCTL_DSP_SPEED,&speed)==-1)  {
   perror("SNDCTL_DSP_SPEED");
   exit(1);
  }
  if(speed==8000)  {
   printf("Now Sampling rate is : %d \n",speed);
  }
 if(ioctl(audio_fd,SNDCTL_DSP_SETFMT,&format)==-1)  {
   perror("SNDCTL_DSP_SETFMT");
   exit(1);
  }
 return audio_fd;
}
/*&&&&&&&&& MAIN &&&&&&&&&*/
main()
{
/*** DECALRATION & INITIALISATION ***/
int i,j,k,l,m,audio_fd,ID;
```

```
double eta=0.2,momentum=0.6,eo,eh;
char ch='a',ch1;
BPNN *bpn;
double *Input,*target,temp,temp1=0.0;
FILE *fperr,*fp;
 double *noise,avg=0.0;
Input=(double *)malloc(10*sizeof(double));
target=(double *)malloc(3*sizeof(double));
audio_fd=set_snd();
/* Here we are going to fetch noise  */
   bpnn_initialize();
   printf("\nSelect one \n");
   printf("\n 1 -Training the net \n");
   printf("\n 2 -Identification \n");
   printf("\n else QUIT\n");
   ch=getchar();
/*##### TRAINING ####*/
getchar();
   if(ch=='1')
  { printf("\n enter 0 to start training  1  to continue ....");
   scanf("%d",&k);
   getchar();
   if(k==0) {
     bpn=bpnn_create(10,16,USERS);
     bpnn_save(bpn,"output.txt");
     bpnn_free(bpn);
   }
   fp=fopen("data.dat","a");
   fperr=fopen("error.txt","a");
     while(1)
  {
   printf("type q to stop entering data \n");
   ch=getchar();
   getchar();
    if(ch=='q')
      break;
   printf("enter id of the person\n");
scanf("%d",&i);
   getchar();
   printf("type enter to start talking\n");
  getchar();
  close(audio_fd);
 audio_fd=set_snd();
  Input=speech_process(audio_fd);
  fprintf(fp,"\n%d:",i);
  for(i=0;i<10;i++) {
        fprintf(fp,"%lf ",Input[i]);
        printf("\n %lf",Input[i]);
        }  };
  fclose(fp);
  printf("\n Type enter to start Training \n");
  fp=fopen("data.dat","r");
  if(fp==NULL) {
  printf("\n Can't open File data.dat");
  exit(0);
}
  for(l=0;l<30000;l++)  {
  if(fscanf(fp,"\n%d:",&i)==EOF)
  {
  rewind(fp);
  fscanf(fp,"\n%d:",&i);
  }
  printf("\n Traininig for user: %d",i);
  bpn=bpnn_read("output.txt");
      for(j=0;j<USERS;j++)
      bpn->target[j]=0.0;
      bpn->target[i-1]=1.0;
      for(j=0;j<10;j++)
      {
      fscanf(fp,"%lf ",&temp);
      Input[j]=temp;
      }
  for(i=0;i<10;i++) {
   bpn->input_units[i]=Input[i];
   }
   bpnn_train(bpn,eta,momentum,&eo,&eh);
  fprintf(fperr," hidden err :%lf  output err :%lf \n",eh,eo);
  for(i=0;i<USERS;i++)
   printf("\noutput %d : %lf",i+1,bpn->output_units[i]);    // for debug
   bpnn_save(bpn,"output.txt");
```

```
     bpnn_free(bpn);
    }
fclose(fperr);
fclose(fp);
  }
/*#####IDENTIFICATION ####*/
  else if(ch=='2') {
     printf("\n Type anything to start speaking for identification \n");
     ch1=getchar();
     Input=speech_process(audio_fd);
 if(Input[0]==0.0)  {
            printf("\n Sorry It's Noise \n");
            exit(0);
    }
    bpn=bpnn_read("output.txt");
    for(i=0;i<10;i++) {
    bpn->input_units[i]=Input[i];
     printf("\n%lf",bpn->input_units[i]);
     }
     bpnn_feedforward(bpn);
    for(i=0;i<USERS;i++)     {
      printf("\n %d : %lf",i,bpn->output_units[i]);
     if(temp1<bpn->output_units[i]) {
       temp1=bpn->output_units[i];
         ID=i+1;
         } }
   printf("\n \n You Are Identified as : %d \n ",ID);
  }
free((double *) Input);
free((double *) target);
return 0;}
```

MODULE II:

End point detection:

```
function acc_globdist=dtwr2(test,ref)
 [row1,col1]=size(ref);
[row2,col2]=size(test);
flag=(col1==col2)|(row1==row2);
if flag==0
  error('The lengths of test and reference vectors do not deserve the hope to
get matched');
end
if flag==1
     [ord,ind]=min([abs(row1-row2) abs(col1-col2)]);
  if ind==1
    i=col1;
    j=col2;
  else
    i=row1;
    j=row2;
  end
end
%
%TO CHECK VALIDITY OF COMPARISION
elap=i-j;
tresh=0.5;
if (abs(elap)>(tresh*(min(i,j))))
  acc_globdist = 60;
  break
end
%TO COMPUTE THE DISTORTION OF PRIMAL MATRIX
ELEMENTS

for c1=1:i
  cumdist(c1,1)=euclidean(1,c1,test,ref);
end
for c2=2:j
  cumdist(1,c2)= euclidean(c2,1,test,ref);
end
%TO TRACE OUT THE DISTORTION MATRIX
```

```
pv=1;ph=1;
for c3=2:j
  for c4=2:i
    step1=1;
    while step1<=3
       if step1==1
         idiff=1;jdiff=0;
         pred(step1)=cumdist(c4-idiff,c3-jdiff)+ph;
       end
       if step1==2
         idiff=1;jdiff=1;
         pred(step1)=cumdist(c4-idiff,c3-jdiff)+(2*euclidean(c4-idiff,c3-
jdiff,ref,test));
       end
       if step1==3
         idiff=0;jdiff=1;
         pred(step1)=cumdist(c4-idiff,c3-jdiff)+pv;
       end
       step1=step1+1;
     end
    [globdist,index]=min(pred);
    cumdist(c4,c3)=globdist;
 end
end
% TO COMPUTE THE GLOBAL DISTORTION BETWEEN THE
TEMPLATES
%
c5=1;
while (c5<=abs(elap)+1)
   if sign(elap)<0
     elapdist(c5)=cumdist(i,(j-c5+1));
   end
   if sign(elap)>0
     elapdist(c5)=cumdist((i-c5+1),j);
   end
   if sign(elap)==0
     elapdist = cumdist(i,j);
   end
   c5=c5+1;
end
```

```
acc=sum(elapdist);
acc_globdist=(acc/max(size(elapdist)));
```

Dtw process:

```
function acc_globdist=dtwr2(test,ref)
 [row1,col1]=size(ref);
[row2,col2]=size(test);
flag=(col1==col2)|(row1==row2);
if flag==0
  error('The lengths of test and reference vectors do not deserve the hope to
get matched');
end
if flag==1
    [ord,ind]=min([abs(row1-row2) abs(col1-col2)]);
  if ind==1
    i=col1;
    j=col2;
  else
    i=row1;
    j=row2;
  end
end
elap=i-j;
tresh=0.5;
if (abs(elap)>(tresh*(min(i,j))))
  acc_globdist = 60;
  break
end
%TO COMPUTE THE DISTORTION OF PRIMAL MATRIX
ELEMENTS
for c1=1:i
  cumdist(c1,1)=euclidean(1,c1,test,ref);
end
for c2=2:j
  cumdist(1,c2)= euclidean(c2,1,test,ref);
end
%TO TRACE OUT THE DISTORTION MATRIX
pv=1;ph=1;
```

```
for c3=2:j
  for c4=2:i
    step1=1;
    while step1<=3
       if step1==1
         idiff=1;jdiff=0;
         pred(step1)=cumdist(c4-idiff,c3-jdiff)+ph;
       end
       if step1==2
         idiff=1;jdiff=1;
         pred(step1)=cumdist(c4-idiff,c3-jdiff)+(2*euclidean(c4-idiff,c3-
jdiff,ref,test));
       end
       if step1==3
         idiff=0;jdiff=1;
         pred(step1)=cumdist(c4-idiff,c3-jdiff)+pv;
       end
       step1=step1+1;
     end
    [globdist,index]=min(pred);
    cumdist(c4,c3)=globdist;
 end
end
% TO COMPUTE THE GLOBAL DISTORTION BETWEEN THE
TEMPLATES
%
c5=1;
while (c5<=abs(elap)+1)
   if sign(elap)<0
     elapdist(c5)=cumdist(i,(j-c5+1));
   end
   if sign(elap)>0
     elapdist(c5)=cumdist((i-c5+1),j);
   end
   if sign(elap)==0
     elapdist = cumdist(i,j);
   end
   c5=c5+1;
end
acc=sum(elapdist);
```

```
acc_globdist=(acc/max(size(elapdist)));
```

mfcc:

```
function ccep=mfcc(y,M,N,P);
subplot(2,3,1);
plot(y);
title('Input Speech Signal');
xlabel('time-->');
ylabel('Amplitude-->');
%normalization
y=y/max(abs(y));
subplot(2,3,2);
plot(y);
title('Normalised Speech Signal');
xlabel('Time');
ylabel('Amplitude');


NYQ=N/2;
% Triangular Filter Defs
for i=1:10
fstart(i)=(2*i)-1;
end;
fcent=fstart+2;
fstop=fstart+4;
fstart(11)=23;
fstart(12)=27;
fstart(13)=31;
fstart(14)=35;
fstart(15)=40;
fstart(16)=46;
fstart(17)=55;
fstart(18)=61;
fstart(19)=70;
fstart(20)=81;
fcent(11)=27;
fcent(12)=31;
fcent(13)=35;
fcent(14)=40;
fcent(15)=46;

fcent(16)=55;
fcent(17)=61;
fcent(18)=70;
fcent(19)=81;
fcent(20)=93;
fstop(11)=31;
fstop(12)=35;
fstop(13)=40;
fstop(14)=46;
fstop(15)=55;
fstop(16)=61;
fstop(17)=70;
fstop(18)=81;
fstop(19)=93;
fstop(20)=108;
seqlen=size(y,1);
m=0;
startpt=1;
endpt=N;
m=1;
while endpt<=seqlen
        winseq=hamming(N).*y(startpt:endpt,1);
  magspec=abs(fft(winseq));
  plot(magspec)
        for i=1:20 % Calc triangle filter outputs
                for j=fstart(i):fcent(i)
       filtmag(j)=(j-fstart(i))/(fcent(i)-fstart(i));
        end;
                for j=fcent(i)+1:fstop(i)
                        filtmag(j)=1-(j-fcent(i))/(fstop(i)-fcent(i));
    end;
    Y(i)=sum(magspec(fstart(i):fstop(i)).*filtmag(fstart(i):fstop(i))');
  end;
 subplot(2,3,2)
  plot(filtmag);

  title('Triangular filter bank');
  xlabel('Frequency');
  ylabel('Normalised Amplitude');
        Y=log(Y.^2);
        for i=1:P
                coefwin=cos((pi/20)*i*(linspace(1,20,20)-0.5))';
                ccep(i,m)=sum(coefwin.*Y');
  end;
        m=m+1;
        startpt=1+(m-1)*M;
  endpt=startpt+N-1;
  startpt
endpt

end;

subplot(2,3,2);
plot(winseq);
title('After Windowing');
xlabel('Time');
ylabel('Normalised Amplitude');
subplot(2,3,3);
plot(magspec);
title('After FFT');
xlabel('Frequency');
ylabel('Normalised Amplitude');
```

Training session:

```
%PROGRAM FOR TRAINING SECTION
name1 = input('Enter the file name : ');
N = input('Enter the  starting number of reference templates to be processed:
');
N1 = input('Enter the  last number of reference templates to be processed : ');
clear ave;clear match;
for k = N:N1
        k
  for l=N:N1
  l
        [y,fs,n] = wavread(strcat(name1,int2str(k),'.wav'));
        c = endpoint_detect(y); %end point detection and returns valid speech
samples
  test = mfcc(c,100,256,10); %returns mel frequency cepstral coefficients
  [y1,fs,n] = wavread(strcat(name1,int2str(l),'.wav'));
        c1= endpoint_detect(y1);
  test1 = mfcc(c1,100,256,10);
  match(l,k)= dtwr2(test,test1);
  end
end
for d =1:(abs(N-N1)+1)
  ave(d) = comp_mean(match(d,:));
end
ave
match
```

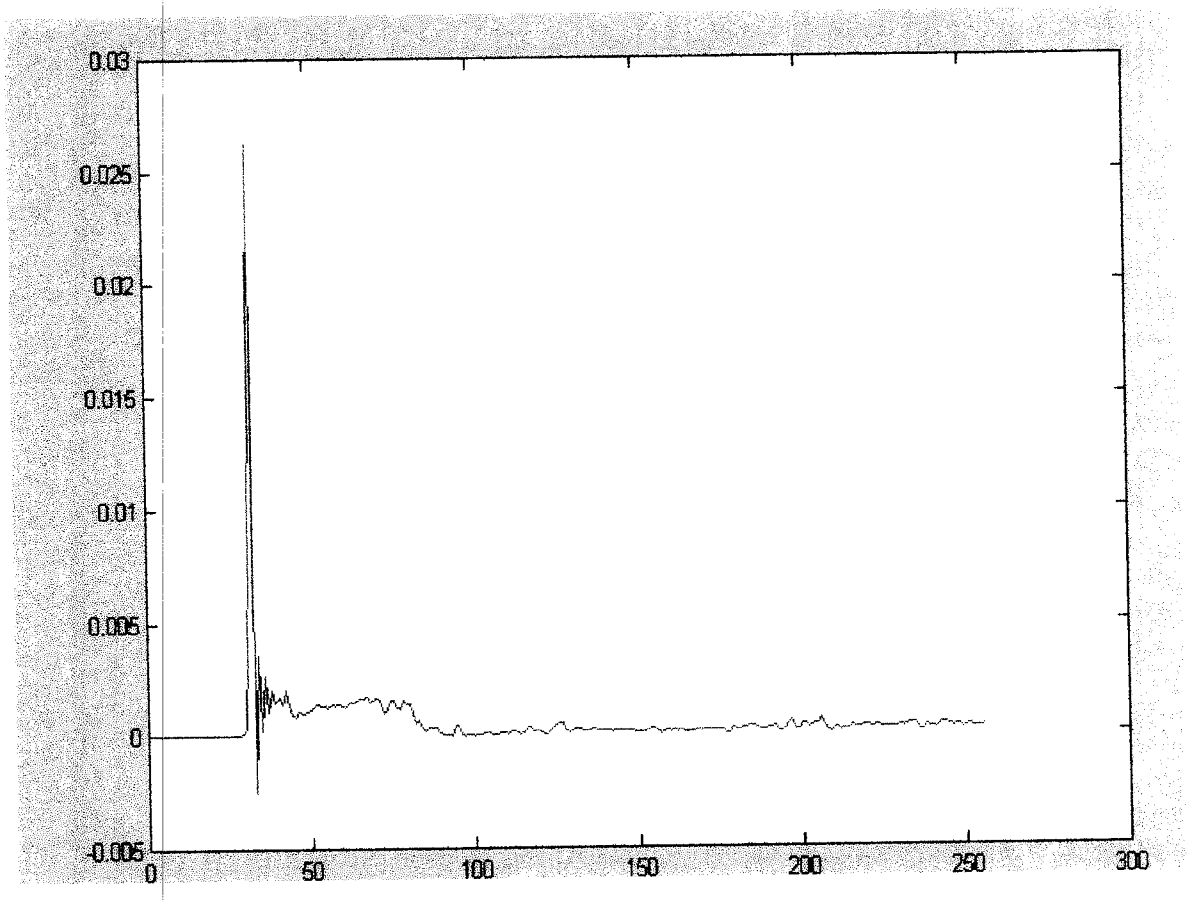
Testing session:

```
 p=0;
while p==0
N = input('Enter the file name : ');
[y,fs,n] = wavread(N);
c = endpoint_detect(y);
test = mfcc(c,100,256,10);
%
%COMPARING SECTION
clc
disp('Comparing with templates for the word "SIX"')
for l=1:10
  index=l
  clear y1 test1 c1;
  [y1,fs,n] = wavread(strcat('si',int2str(l),'.wav'));
  c1= endpoint_detect(y1);
  test1 = mfcc(c1,100,256,10);
  match(1,l)= dtwr2(test,test1);
end
clc
disp('Comparing with templates for the word "FOUR"')
for l=1:10
  index=l
  clear y1 test1 c1;
  [y1,fs,n] = wavread(strcat('fou',int2str(l),'.wav'));
        c1=endpoint_detect(y1);
  test1 = mfcc(c1,100,256,10);
  match(2,l)= dtwr2(test,test1);
end
clc

disp('Comparing with templates for the word "EIGHT"')
for l=1:10
  index=l
  clear y1 test1 c1;
  [y1,fs,n] = wavread(strcat('eight',int2str(l),'.wav'));
  c1= endpoint_detect(y1);
  test1 = mfcc(c1,100,256,10);
  match(3,l)= dtwr2(test,test1);
end
clc
disp('Comparing with templates for the word "NINE"')
for l=1:10
  index=l
  clear y1 test1 c1;
  [y1,fs,n] = wavread(strcat('nine',int2str(l),'.wav'));
  c1= endpoint_detect(y1);
  test1 = mfcc(c1,100,256,10);
  match(4,l)= dtwr2(test,test1);
end

%
%AVERAGE CALCULATION
for k =1:4
  ave(k) = comp_mean(match(k,:));
end
flag=0;
for k1=1:length(ave)
  if ave(k1) >= 30
        flag = flag + 1;
        end
end
if flag<length(ave)
 [min,opt] = min(ave);
```

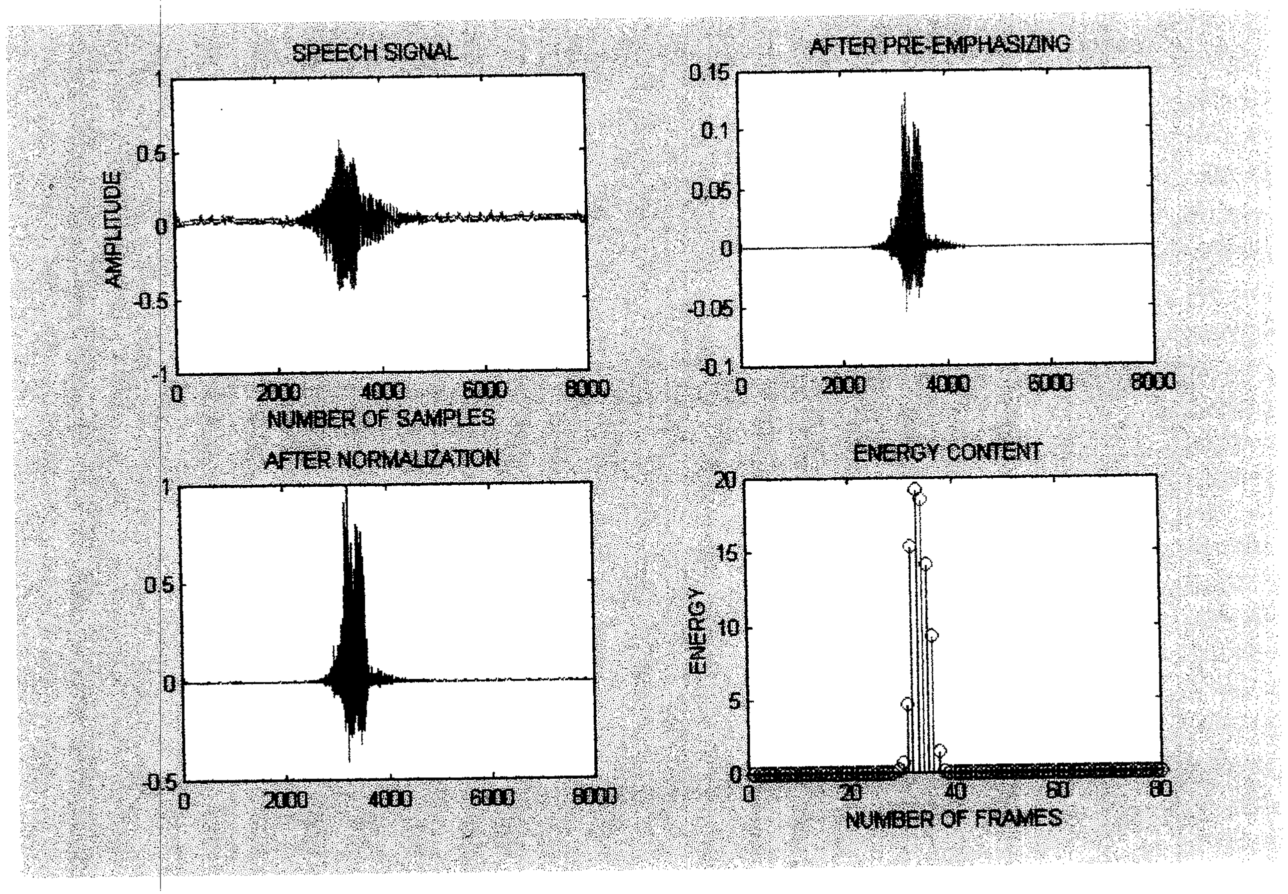
```
%
%OUTPUT SECTION
clc
if opt==1
  disp('You have utterred the number 6');
  disp('Thank You');
  [tri1,fs,n]=wavread('tria2.wav');
  soundsc(tri1);
end
if opt==2
  disp('You have utterred the number 4');
  disp('Thank You');
  [tri2,fs,n]=wavread('tria1.wav');
  soundsc(tri2);
end
if opt==3
  disp('You have utterred the number 8');
  disp('Thank You');
  [tri,fs,n]=wavread('tria.wav');
  soundsc(tri);
end
if opt==4
  disp('You have utterred the number 9');
  disp('Thank You');
  [tri,fs,n]=wavread('tria9.wav');
  soundsc(tri);
end
break
else
  disp('Sorry!');
  disp('      ');
  disp('Your voice is not audible. Please try again');
end
end
```

SCREEN SHOTS:

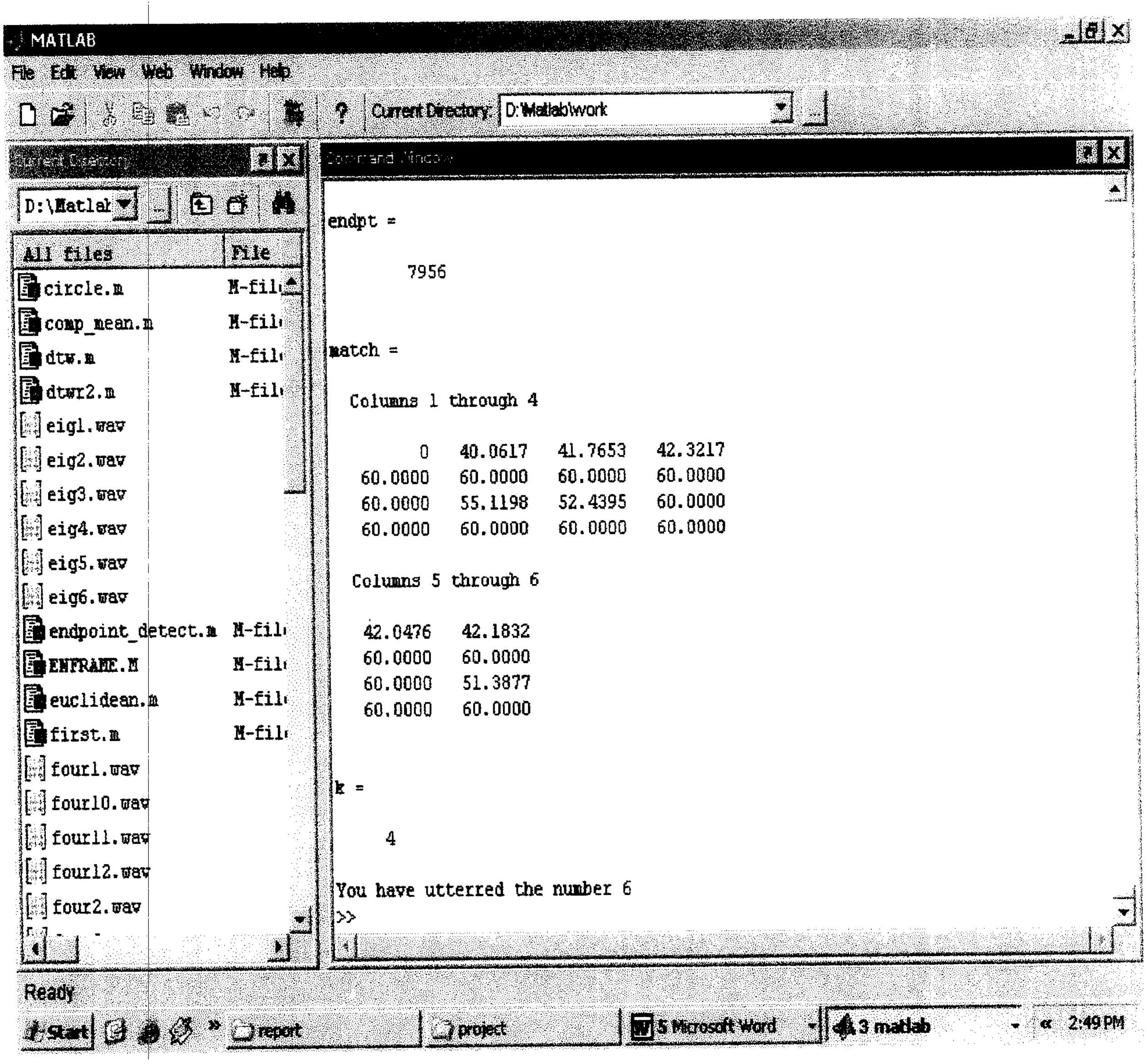
FRAMING



SPEECH PROCESSING:



OUTPUT SCREEN:(DTW)



```
endpt =

      7956


match =

  Columns 1 through 4

         0   40.0617   41.7653   42.3217
   60.0000   60.0000   60.0000   60.0000
   60.0000   55.1198   52.4395   60.0000
   60.0000   60.0000   60.0000   60.0000

  Columns 5 through 6

   42.0476   42.1832
   60.0000   60.0000
   60.0000   51.3877
   60.0000   60.0000


k =

     4

You have utterred the number 6
>>
```

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Neural networks in computer intelligence, Limin Fu, McGraw-Hill International Editions, 1994.
2. Getting started with MATLAB 5, Rudra Pratap, Oxford University Press, 1999.
3. Using Speech Recognition, Judith.A.Markowitz, PTR Publishing House, 1995.
4. Digital Signal Processing, S.Salivahanan, A.Vallavaraj, C. Gnanapriya, Tata-McGraw-Hill Publications, 2000.
5. Digital Signal Processing Using MATLAB, Vinay.K.Ingle, John.G. Proakias, Thomson Publications, 1998.

Web Sites References:

1. http://lcavwww.epfl.ch, 15-DEC-2002
2. http://www.accurate-automation.com, 22-DEC-2002
3. http://www.fon.hum.uva.nl, 8-JAN-2003