DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COMPUTER BASED SYSTEM CONTROL USING VOICE INPUT

By

Mehmet KARAKAŞ

September, 2010

İZMİR

COMPUTER BASED SYSTEM CONTROL USING VOICE INPUT

A Thesis Submitted to the

Graduate School of Natural and Applied Sciences of Dokuz Eylül University

In Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical Electronics Engineering

by

Mehmet KARAKAŞ

September, 2010

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"COMPUTER BASED SYSTEM CONTROL USING VOICE INPUT"** completed by **MEHMET KARAKAŞ** under supervision of **PROF. DR. MUSTAFA GÜNDÜZALP** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

.....

Prof. Dr. Mustafa GÜNDÜZALP

.....

(Jury Member)

.....

(Jury Member)

Prof.Dr. Mustafa SABUNCU

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor, Prof. Dr. Mustafa GÜNDÜZALP for his continuos support and valuable time in constructing this thesis.

I also thank to my colleagues for their support and help during my study.

And I would like to thank my family for their patience, great support and help.

Mehmet KARAKAŞ

COMPUTER BASED SYSTEM CONTROL USING VOICE INPUT

ABSTRACT

Speech recognition is the perception of human voice by computer or electronic devices. Using speech recognition we can give voice commands to an electronic device. Simple control of a hardware device by voice commands via computer is aimed at this project. Also speaker recognition is performed in the project.

In this thesis, theory of speech processing and recognition is told briefly. As feature extraction method, mel frequency cepstrum coefficients and as comparison method between speeches, dynamic time warping methods are used. For speaker recognition the same methods are also used.

Speech is recorded via microphone into the computer and computer program recognizes the speech command and by serial port, appropriate output is activated on the designed hardware according to the spoken command.

All work is done on computer on MATLAB software. Speech is recorded, processed and given output to the hardware by MATLAB.

Keywords : Speech recognition, speaker recognition, mel frequency cepstrum coefficients, dynamic time warping, MATLAB.

BİLGİSAYAR TABANLI SESLE KONTROL SİSTEMİ

ÖZ

Konuşma tanıma insan sesinin bilgisayar ya da elektronik aletler tarafından algılanmasıdır. Konuşma tanıma yöntemiyle elektronik bir aygıta ses komutları verebilir ve kontrol edebiliriz. Bu projede bilgisayar üzerinden ses komutları ile bir elektronik aygıtı basit olarak kontrol etme amaçlanmıştır. Ayrıca konuşmacı tanıma da bu projede uygulanmıştır.

Bu çalışmada konuşma tanıma teorisi kısaca anlatılmıştır. Ses sinyalinden özellik çıkarma metodu olarak mel frekansı kepstrum katsayıları metodu, ses sinyallerini birbirleriyle karşılaştırmak için ise dinamik zaman kırpma yöntemi kullanılmıştır. Konuşmacı tanıma için de aynı yöntemler uygulanmıştır.

Konuşma, mikrofon vasıtası ile bilgisayara kaydedilir ve bilgisayar, konuşulan sese göre seri port üzerinden, tasarlanan donanım kartındaki uygun çıkışı kontrol eder.

Bilgisayar üzerindeki tüm çalışmalar MATLAB isimli bilgisayar yazılımı üzerinde yapılmıştır. MATLAB üzerinde, hem ses sinyalleri kaydedilir hem de bu ses sinyalleri işlenerek donanım kartı kontrol edilir.

Anahtar kelimeler : Konuşma tanıma, konuşmacı tanıma, mel frekansı kepstrum katsayıları, dinamik zaman kırpma, MATLAB.

CONTENTS

Page

CHAPTER ONE

INTRODUCTION

1.1 Speech and Speech Recognition

Speech is the natural and basic way of communication between humans. Because it is a very easy thing for humans to speak, it is wanted to use to communicate humans to other devices in the environment. To communicate humans with the devices, devices must understand the human speech. Humans have brains to do this job, but devices do not. Because of this, computers or micro-computers should be used on devices.

Speech recognition is the process of understanding of human speech by devices. Basically, speech is somehow acquired by the device by microphone and the analog speech signal is converted to digital signal. Then by using some techniques human speech is recognized by the device. According to human speech the device responses as programmed.

Speech recognition is widely used in many applications today like cell phones or security systems. It is also useful for disabled people to command devices by just their voices. Processing of speech can be done by a computer or by a DSP according to use of interest. Widely use of speech recognition is real time process by DSPs.

Speech recognition is used as speaker verification, speaker recognition or systems independent of speaker and systems recognize isolated words or continuos speech. Speaker verification is speaker dependant and mostly used in security systems. Speaker independant systems are for the use of all people and works with the input of known commands.

1.2 General Idea of Speech Recognition

Human speech presents a formidable pattern classification task for speech recognition system. Numerous speech recognition techniques have been formulated; yet the very best techniques used today have recognition capabilities well below those of a child. This is due to the fact that human speech is highly dynamic and complex. There are generally several types of disciplines present in the human speech. A basic understanding of these disciplines is needed in order to create an effective system. The following provide a brief description of the disciplines that have been applied to speech recognition problems

- **Signal Processing :** This process extracts the important information from the speech signal in a well-organised manner. In signal processing, spectral analysis is used to characterize the time varying properties of the speech signal. Several other types of processing are also needed prior to the spectral analysis stage to make the speech signal more accurate and robust.

- **Acoustics :** The science of understanding the relationship between the physical speech signal and the human vocal tract mechanisms that produce the speech and with which the speech is distinguished.

- **Pattern Recognition :** A set of coding algorithm used to compute data to create prototypical patterns of a data ensemble. It is used to compare a pair of patterns based on the features extracted from the speech signal.

- **Communication and Information Theory :** The procedures for estimating parameters of the statistical models and the methods for recognizing the presence of speech patterns.

- **Linguistics :** This refers to the relationships between sounds, words in a sentence, meaning and logic of spoken words.

- **Physiology :** This refers to the comprehension of the higher-order mechanisms within the human central nervous system. It is responsible for the production and perception of speech within the human beings.

- **Computer Science :** The study of effective algorithms for application in software and hardware. For example, the various methods used in a speech recognition system.

- **Psychology :** The science of understanding the aspects that enables the technology to be used by human beings. (Rabiner and Juang,1993)

1.3 Speech Production

Speech is the acoustic product of voluntary and well-controlled movement of a vocal mechanism of a human. During the generation of speech, air is inhaled into the human lungs by expanding the rib cage and drawing it in via the nasal cavity, velum and trachea. It is then expelled back into the air by contracting the rib cage and increasing the lung pressure. During the expulsion of air, the air travels from the lungs and passes through vocal cords which are the two symmetric pieces of ligaments and muscles located in the larynx on the trachea. Speech is produced by the vibration of the vocal cords. Before the expulsion of air, the larynx is initially closed. When the pressure produced by the expelled air is sufficient, the vocal cords are pushed apart, allowing air to pass through. The vocal cords close upon the decrease in air flow. This relaxation cycle is repeated with generation frequencies in the range of 80Hz – 300Hz. The generation of this frequency depends on the speaker's age, sex, stress and emotions. This succession of the glottis openings and

closure generates quasi-periodic pulses of air after the vocal cords.(Rabiner and Juang,1993)

Figure below shows the schematic view of the human speech apparatus.

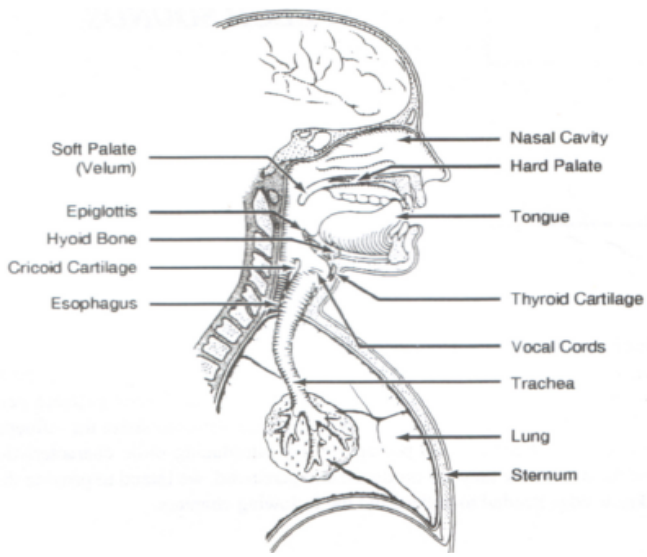

Figure 1.1 How speech occurs in humans.(Rabiner and Juang,1993)

A more machine – style explanation of speech generation in humans is told briefly. “The production (speech generation) process begins when the talker formulates a message (in his mind) that he wants to transmit to the listener via speech. The machine counterpart to the process of message formulation is the creation of printed text expressing the words of the message. The next step in the process is the conversion of the message into a language code. This roughly corresponds to converting the printed text of the message into a set of phoneme sequences corresponding to the sounds that make up the words, along with prosody markers denoting duration of sounds, loudness of sounds, and pitch accent associated with the sounds. Once the language code is chosen, the talker must execute a series of neuromuscular commands to cause the vocal cords to vibrate when appropriate and to shape the vocal tract such that the proper sequence of speech sounds is created and spoken by the talker, thereby producing an acoustic signal as the final output.

The neuromuscular commands must simultaneously control all aspects of articulatory motion including control of the lips, jaw, tongue, and velum (a "trapdoor" controlling the acoustic flow to the nasal mechanism).”(Rabiner and Juang,1993)

1.4 Speech Perception (Recognition)

Once the speech signal is generated and propagated to the listener, the speech perception (or speech-recognition) process begins. First the listener processes the acoustic signal along the basilar membrane in the inner ear, which provides a running spectrum analysis of the incoming signal. A neural transduction process converts the spectral signal at the output of the basilar membrane into activity signals on the auditory nerve, corresponding roughly to a feature extraction process. In a manner that is not well understood, the neural activity along the auditory nerve is convened into a language code at the higher centers of processing within the brain, and finally message comprehension (understanding of meaning) is achieved. A slightly different view of the speech-production/speech-perception process is shown in Figure 1.2. Here we see the steps in the process laid out along a line corresponding to the basic information rate of the signal (or control) at various stages of the process. The discrete symbol information rate in the raw message text is rather low (about 50 bps [bits per second] corresponding to about 8 sounds per second, where each sound is one of about 50 distinct symbols). After the language code conversion, with the inclusion of prosody information, the information rate rises to about 200 bps. Somewhere in the next stage the representation of the information in the signal (or the control) becomes continuous with an equivalent rate of about 2000 bps at the neuromuscular control level, and about 30,000-50,000 bps at the acoustic signal level. A transmission channel is shown in Figure 1.3 , indicating that any of several well-known coding techniques could be used to transmit the acoustic waveform from the talker to the listener. The steps in the speech-perception mechanism can also be interpreted in terms of information rate in the signal or its control and follows the inverse pattern of the production process. Thus the continuous information rate at the basilar membrane is in the range of 30,000-50,000 bps, while at the neural transduction stage it is about 2000 bps. The higher-level processing within the brain

converts the neural signals to a discrete representation, which ultimately is decoded into a low-bit-rate message.(Rabiner and Juang,1993)

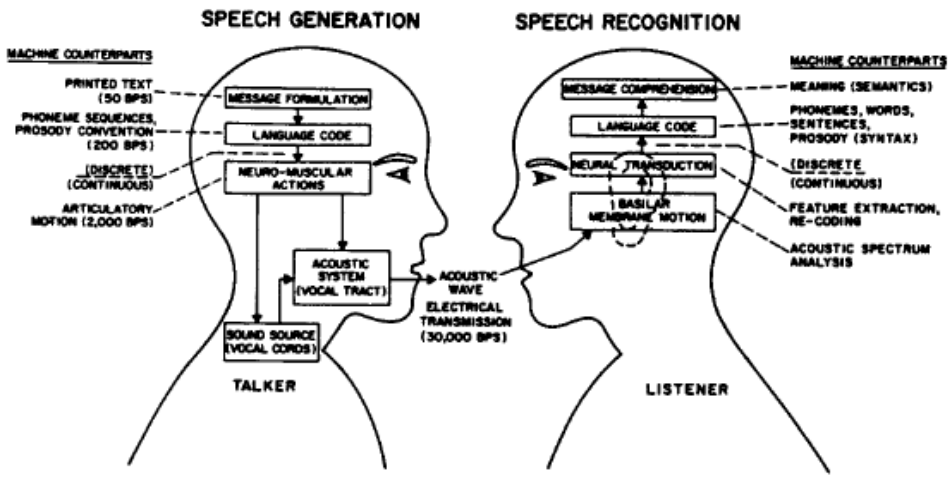

Figure 1.2 Speech production and recognition(Rabiner and Juang,1993)

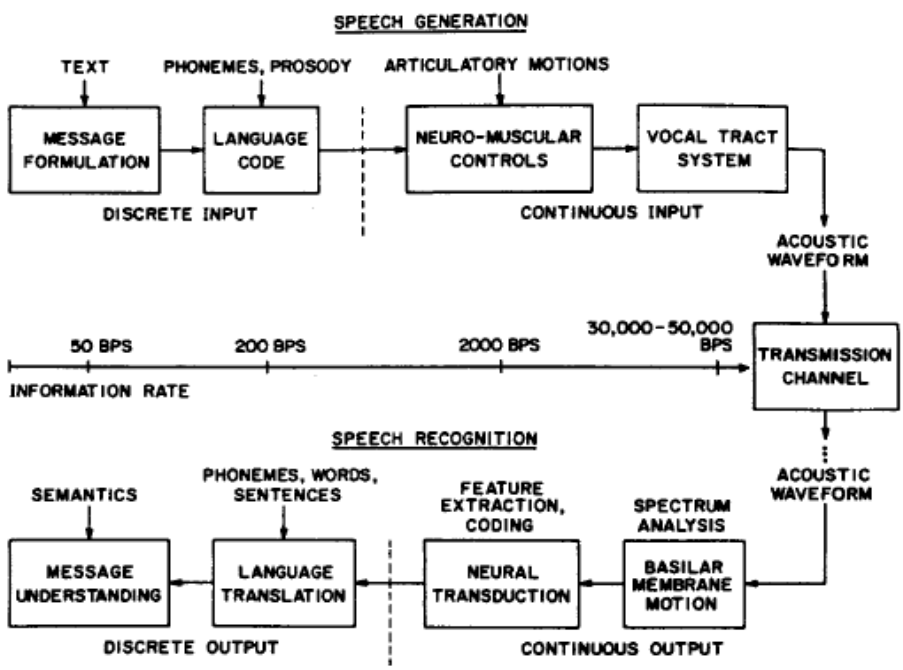

Figure 1.3 Speech generation diagram(Rabiner and Juang,1993)

1.5 Speech Recognition Process

The speech signal is processed several times on the computer side. Firstly, the speech signal is acquired by a microphone to the computer. Because computer cannot process analog signals, the analog speech signal is converted into digital signal after recording. According to the Nyquist Theorem, the minimum sampling rate required is two times the bandwidth of the signal. This minimum sampling frequency is needed for the reconstruction of a band limited waveform without error. Aliasing distortion will occur if the minimum sampling rate is not met.

After recording and sampling of the speech signal, the digital speech signal must be filtered to avoid noises. Noise can affect the signal and corrupt it. The environment that the microphone placed can be noisy and the microphone and the computer can put additional noise on the signal. By filtering the noise can be removed from the speech signal.

The filtered speech signal is in hand but now it has unwanted and unused blank parts at start and end. These blank parts are unused and cause unnecessary process on the computer. These blank parts should be removed with a start-end point detection and removal algorithm.

In the signal, now there is only speech. On this data suitable feature extraction methods can be used. There are several feature extraction methods to model the speech signal suitable for comparison and matching. Mostly mel frequency cepstrum coefficients (MFCC) and linear predictive coding (LPC) methods are used. Dynamic time warping (DTW) or some other recognition methods are used to recognize and match the speech data.

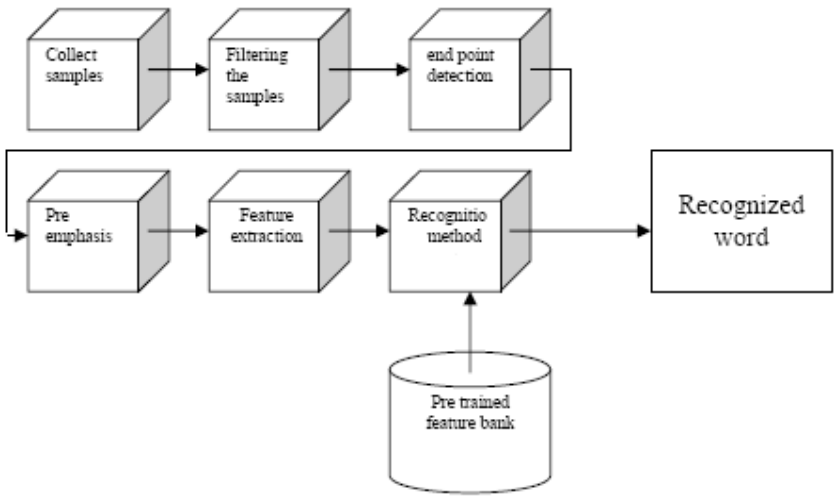

Figure 1.4 Steps of speech recognition

1.6 Speaker Recognition

Speaker recognition is a type of speech recognition. Both techniques use similar methods of speech signal processing. In automatic speech recognition, the speech processing approach tries to extract linguistic information from the speech signal to the exclusion of personal information. However, speaker recognition is focused on the characteristics unique to the individual, disregarding the current word spoken. The uniqueness of an individual's voice is a consequence of both the physical features of the person vocal tract and the person mental ability to control the muscles in the vocal tract. An ideal speaker recognition system would use only physical features to characterize speakers, since these features cannot be easily changed. However, it is obvious that the physical features as vocal tract dimensions of an unknown speaker cannot be simply measured. Thus, numerical values for physical features or parameters would have to be derived from digital signal processing parameters extracted from the speech signal.

Speaker recognition methods can also be divided into text-dependent and text-independent methods. In this project, text-dependant speaker recognition is mentioned. The text-dependent methods are usually based on template matching

techniques in which the time axes of an input speech sample and each reference template or reference model of registered speakers are aligned, and the similarity between them accumulated from the beginning to the end of the utterance is calculated. The structure of text-dependent recognition systems is, therefore, rather simple. Since this method can directly exploit the voice individuality associated with each phoneme or syllable, it generally achieves higher recognition performance than the text-independent method.(Sigmund)

1.7 Outline of Thesis

In chapter one, introduction to speech recognition and some general information about speech and speech recognition is given.

In chapter two, theoretical information is given about speech processing and recognition. Sampling theorem, digital signal processing, mel frequency cepstrum coefficients and dynamic time warping theories that are used in this work are told in this chapter.

In chapter three, software design of the work is told. Functions and code sections which are used in MATLAB are explained in this chapter.

In chapter four, hardware design of the work is told. Serial port and microcontroller details are explained in this chapter.

In chapter five, results obtained during experiment studies of the program and comments and suggestions about future works are told in the chapter.

CHAPTER TWO

THEORIC BACKGROUND

2.1 Conversion of Speech Signal to Digital

Most signals of practical interest, such as speech, biological signals, seismic signals, radar signals, sonar signals. and various communications signals such as audio and video signals, are analog. To process analog signals by digital means, it is first necessary to convert them into digital form. That is, to convert them to a sequence of numbers having finite precision. This procedure is called analog-to-digital conversion, and the corresponding devices are called analog to digital converters (ADCs). Conceptually, we view analog-to-digital conversion as a three-step process. This process is illustrated in Fig. 2.1.

- **Sampling :** This is the conversion of a continuous-time signal into a discrete time signal obtained by taking "samples" of the continuous-time signal at discrete-time instants.
- **Quantization :** This is the conversion of a discrete-time continuous-valued signal into a discrete-time, discrete-valued (digital) signal. The value of each signal sample is represented by a value selected from a finite set of possible values.
- **Coding :** In the coding process, each discrete value x is represented by a b-bit binary sequence. (Proakis & Manolakis, 1996)

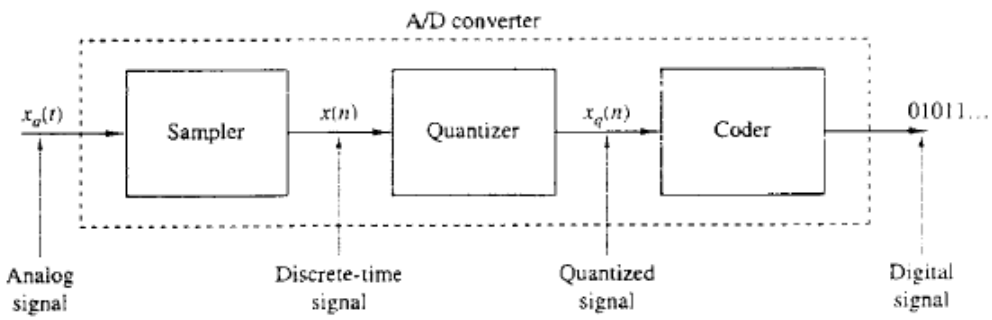

Figure 2.1 Analog to digital conversion(Proakis & Manolakis, 1996)

Here, the analog to digital converter is computer's sound card. The sound card does all the sampling, quantizing and coding processes and when we record our voice via microphone, we see digitized speech signal in MATLAB software.

2.2 Sampling

There are many ways to sample an analog signal. We limit our discussion to periodic or uniform sampling; which is the type of sampling used most often in practice. This is described by the relation

$$x(n) = x_a(nT) \qquad -\infty < n < \infty \qquad (1)$$

where $x(n)$ is the discrete-time signal obtained by "taking samples" of the analog signal $x_a(t)$ every T seconds. The time interval T between successive samples is called the sampling period or sample interval and its reciprocal $1/T = F_s$ is called the sampling rate (samples per second) or the sampling frequency (hertz).

2.2.1 Sampling Theorem

If the highest frequency contained in an analog signal $x_a(t)$ *is* $F_{max} = B$ and the signal is sampled at a rate $F_s > 2F_{max} = 2B$, the sampling rate $F_n = 2B = 2F_{max}$, is called the *Nyquist rate.*(Proakis & Manolakis, 1996) To avoid aliasing, sampling frequency of analog signals must be at least two times of maximum frequency of analog signal.

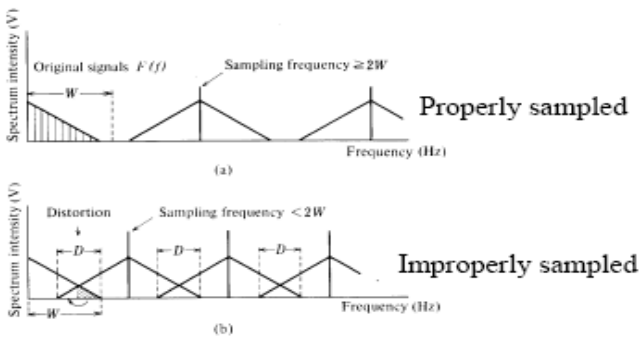

Figure 2.2 Properly and improperly sampled signal

2.3 Filtering

It is definitely needed to apply filtering to the speech signal because of noise. Filtering is obviously the most important process in speech recognition. If filtering operation is inadequate this affects the whole recognition process. With insufficient filtering, the end-point process fails to correctly detect the start and end points of the signal. Thus recognition is not fully satisfied.

Filtering is the most common form of signal processing used in all the applications to remove the frequencies in certain parts and to improve the magnitude, phase, or group delay in some other part(s) of the spectrum of a signal. The vast literature on filters consists of two parts: First, the theory of approximation to derive the transfer function of the filter such that the magnitude, phase, or group delay approximates the given frequency response specifications and procedures to design the filters using the hardware. (Shenoi,2006)

In this project, digital filtering on MATLAB is performed. Because of the noise and signal characteristics FIR bandpass filter is applied to speech signals. Thus, theoric information about FIR filters is mentioned.

2.3.1 FIR Filters

A finite impulse response (FIR) filter is a type of a discrete-time filter. The impulse response of the filter is finite because it settles to zero in a finite number of sample intervals.

The difference equation of the FIR filter type that defines its output is:

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N] \qquad (2)$$

where *x[n]* is input signal, *y[n]* is output signal, b_i are filter coefficients and *N* is the order of the filter.

Designing a filter means calculating the coefficients. To find coefficients from frequency specifications, windowing method is used. In the windowing method, a window function (mostly hamming window) is applied to an ideal IIR filter in the time domain, multiplying the infinite impulse by the window function. This results in the frequency response of the IIR being convolved with the frequency response of the window function thus the imperfections of the FIR filter (compared to the ideal IIR filter) can be understood in terms of the frequency response of the window function.

2.4 Feature Extraction

This stage is often referred as speech processing front end. The primary goal of feature extraction is to simplify recognition by summarizing the vast amount of speech data and obtaining the acoustic properties that define speech individuality. MFCC (Mel Frequency Cepstral Coefficients) is one of the most widely used feature extraction techniques. Since speech signal varies over time, it is more appropriate to analyze the signal in short time intervals where the signal is more stationary. Because of these MFCC method is used as feature extraction technique.

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 a linear frequency spacing below 1000 Hz and a logarithmic spacing above 1000 Hz.

2.5 Mel Frequency Cepstrum Coefficients (MFCC)

The speech signal and all its characteristics can be represented in two different domains, the time and the frequency domain. A speech signal is a slowly time varying signal in the sense that, when examined over a short period of time (between 5 and 100 ms), its characteristics are short-time stationary. This is not the case if we look at a speech signal under a longer time perspective (approximately time $T > 0.5$ s). In this case the signals characteristics are non-stationary, meaning that it changes to reflect the different sounds spoken by the talker.

Psychophysical studies have shown that human perception of the frequency content of sounds, either for pure tones or for speech signals, does not follow a linear scale. This research has led to the idea of defining subjective pitch of pure tones. Thus for each tone with an actual frequency, f, measured in Hz, a subjective pitch is measured on a scale called the "mel" scale. As a reference point, the pitch of a 1 kHz tone, 40 dB above the perceptual hearing threshold, is defined as 1000 mels.

2.6 Mel-frequency cepstrum coefficients processor

A block diagram of the structure of an MFCC processor is given in Figure 2.3. The speech input is typically recorded at a sampling rate of 16000 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 and over, 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, MFFC's are shown to be less susceptible to mentioned variations. The coefficients are calculated as follows:

1- Fourier transform of the signal is taken.
2- By windowing, the power of spectrum is mapped onto mel scale.
3- The logs of the powers at each of the mel frequencies are taken.
4- Discrete cosine transform of the list of mel log powers are taken.
5- The coefficients are the amplitudes of the resulting spectrum.

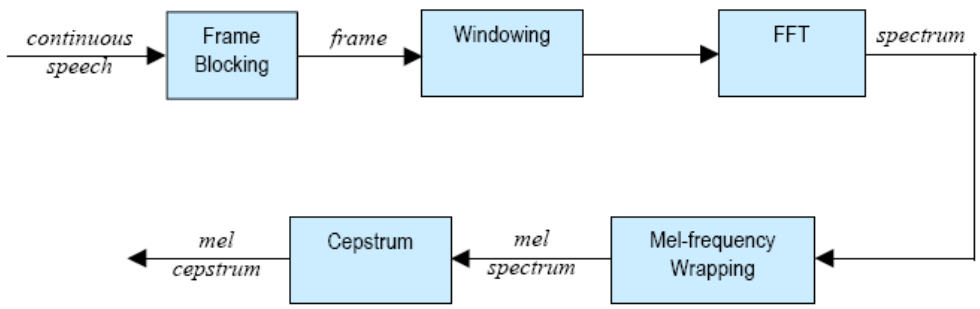

Figure 2.3 Mel frequency cepstrum coefficients processing

2.6.1 Frame Blocking

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 $2M$ samples after the first frame (or M samples after the second 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 ~ 30 msec windowing and facilitate the fast radix-2 FFT) and $M = 100$.

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

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

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

2.6.3 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:

$$X_n = \sum_{k=0}^{N-1} x_k e^{-\frac{2\pi j k n}{N}}$$

$n = 0,1,2,\ldots\ldots,N-1$ (4)

Note that j is 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 $N/2 + 1 \le n \le N - 1$.

The result after this step is often referred to as spectrum or periodogram.

2.6.4 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 a 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:

$$mel(f) = 2595 * log_{10}(1 + f/700) \quad (5)$$

One approach to simulating the subjective spectrum is to use a filter bank, spaced uniformly on the mel scale (see Figure 2.4). 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 $S(\omega)$ 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 12.

Note that this filter bank is applied in the frequency domain, therefore it simply amounts to taking those triangle-shape windows in the Figure 2.4 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. (Anonymous)

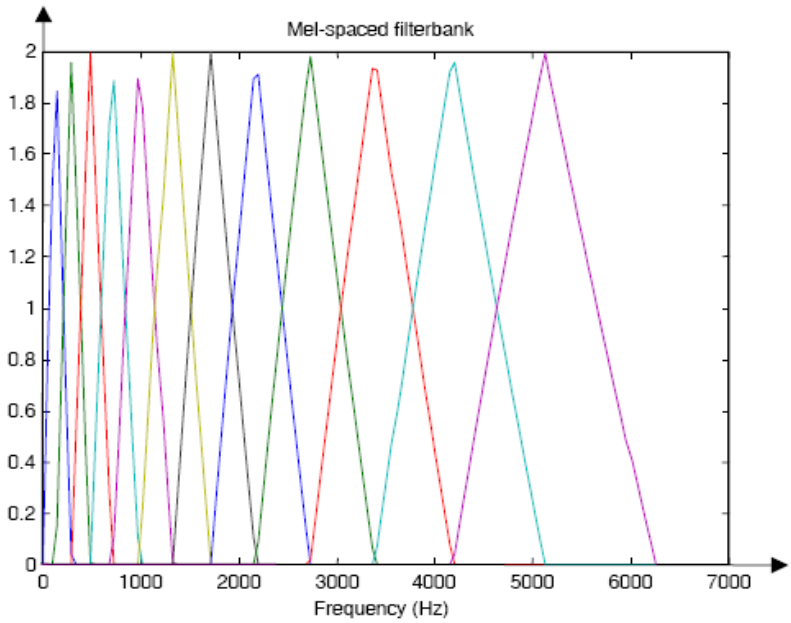

Figure 2.4. An example of Mel-spaced filter bank(Anonymous)

2.6.5 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 S_k, $k = 1,2,\ldots\ldots,K$, we can calculate the MFCC's, c_n as

$$C_n = \sum_{k=1}^{K} (\log S_k) Cos\left[n\left(k - \frac{1}{2} \right) \frac{\pi}{k} \right], \qquad n = 1,2,\ldots\ldots,K \qquad (6)$$

Note that we exclude the first component, ~0*c* from the DCT since it represents the mean value of the input signal which carried little speaker specific information. (Anonymous)

2.7 Dynamic Time Warping (DTW)

Dynamic Time Warping (Sakoe & Chiba 1978) is used to compute a distance between two time series. A time series is a list of samples taken from a signal, ordered by the time that the respective samples were obtained.

A naive approach to calculating a matching distance between two time series could be to resample one of them and then compare the series sample-by-sample. The drawback of this method is that it does not produce intuitive results, as it compares samples that might not correspond well.

Dynamic Time Warping solves this discrepancy between intuition and calculated matching distance by recovering optimal alignments between sample points in the two time series. The alignment is optimal in the sense that it minimizes a cumulative distance measure consisting of "local" distances between aligned samples. The procedure is called 'Time Warping' because it warps the time axes of the two time series in such a way that corresponding samples appear at the same location on a common time axis.

The DTW-distance between two time series (*x*1 . . . *xM)* and (*y*1 . . . *yN)* is *D*(*M,N*), which we calculate in a dynamic programming approach using

$$D(i\ ,\ j) = min\begin{Bmatrix} D(i, j-1) \\ D(i-1, j) \\ D(i-1, j-1) \end{Bmatrix} + d(X_i\ ,\ Y_i) \qquad (6)$$

The particular choice of recurrence equation and "local" distance function $d(\cdot, \cdot)$ varies with the application. Using the given three values $D(i, j-1)$, $D(i-1, j)$ and

$D(i-1, j-1)$ in the calculation of $D(i, j)$ realizes a *local continuity constraint* (Figure 2.5), which ensures smooth time warping.(Toni M. Rath & R. Manmatha)

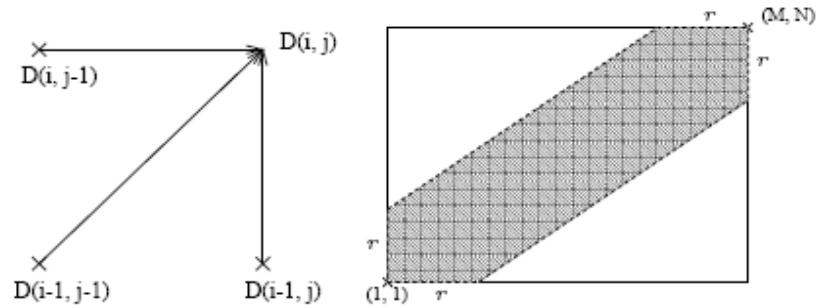

Figure 2.5 Constraints used in the dynamic time warping implementation.
(Toni M. Rath & R. Manmatha)

Since the feature vectors could possibly have multiple elements, a means of calculating the local distance is required. The distance measure between two feature vectors is calculated using the Euclidean distance metric. Therefore the local distance between feature vector x of signal 1 and feature vector y of signal 2 is given by,

$$d(x, y) = \sqrt{\sum_i (X_i - Y_i)^2} \qquad (8)$$

The best matching template is the one for which there is the lowest distance path aligning the input pattern to the template. A simple global distance score for a path is simply the sum of local distances that go to make up the path.

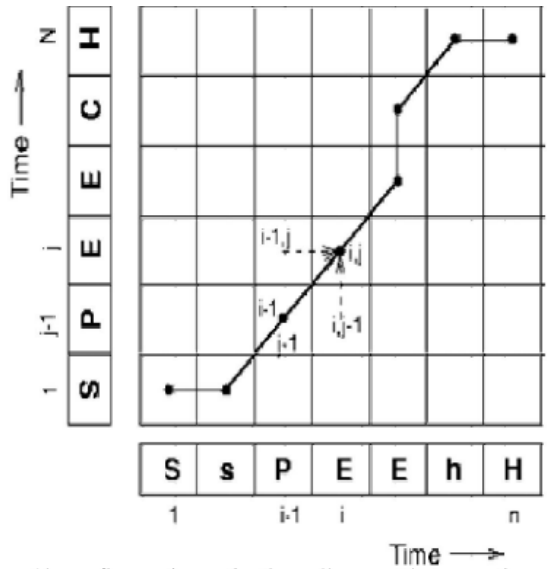

Figure 2.6 Application of DTW (Kale, K. R.)

The minimum distance between “SPEECH” and “SsPEEhH” is shown in the figure 2.6. The program of the algorithm is named dynamic programming.

CHAPTER THREE

SOFTWARE DESIGN

The speech acquisition and recognition is done on computer with the software MATLAB® of The MathWorks Inc.

MATLAB is a software package that lets you do mathematics and computation, analyse data, develop algorithms, do simulation and modeling, and produce graphical displays and graphical user interfaces.

MATLAB is named from MATrix LABoratory compund name. Engineering applications, computations and most simulations are done on MATLAB. MATLAB is a matrix and mathematical based complex software.

3.1 General Algorithm

To realize the system control on computer, a graphical user interface (GUI) is designed. All processes are implemented inside the GUI program code. When the program is started, a main control panel is opened. From this panel, it is chosen whether speaker independent speech recognition or text dependant speaker recognition.

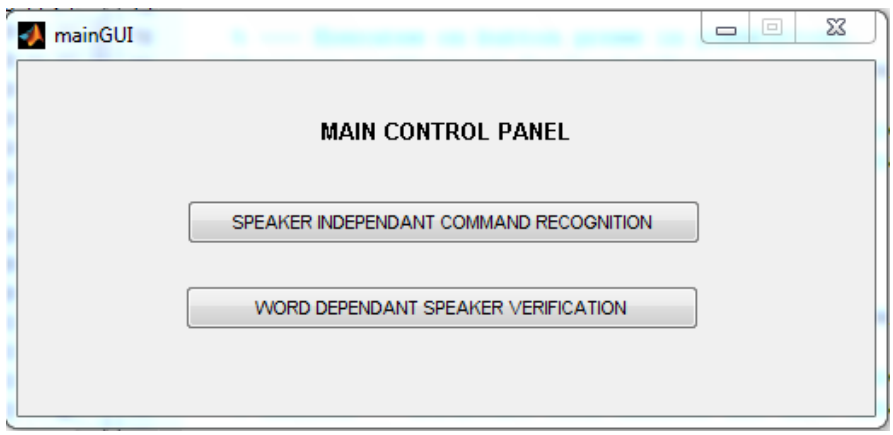

Figure 3.1 - Main control GUI

The speech recognition and system control algorithm works as in the sequence as: First, 4 different words are recorded and named separately. These 4 words are recorded and saved with their given names on the hard disk for further use and comparison. Now, a word is spoken to the microphone from those words to control a device. When a word is spoken, the software compares it with previously recorded 4 words and gives the matching word as the result and the corresponding output becomes active of the control system.

The speaker recognition algorithm also works in the same manner as speech recognition: First, 4 different persons speak a specified word and their voice are recorded and named separately. These 4 voices are recorded and saved with their given names on the hard disk for further use and comparison. Now, a person speaks to the microphone the same word. When a person speaks the word the software compares it with previously recorded 4 persons and gives the matching person as the result.

3.2 Speech Signal Processing

3.2.1 Recording

To record a speech MATLAB's *wavrecord* function is used. We specify the sample number and sample rate as frequency in the *wavrecord* function.

```
fs=16000;
duration=3;
pause;
ses1=wavrecord(duration*fs,fs);
```

As seen in the code section above a sampling frequency of 16000 Hz is specified. The recording duration is 3 seconds. The voice is recorded for 3 seconds into the variable 'ses1'. Now in the variable 'ses1', there is the recording for 3 seconds, 48000 samples.

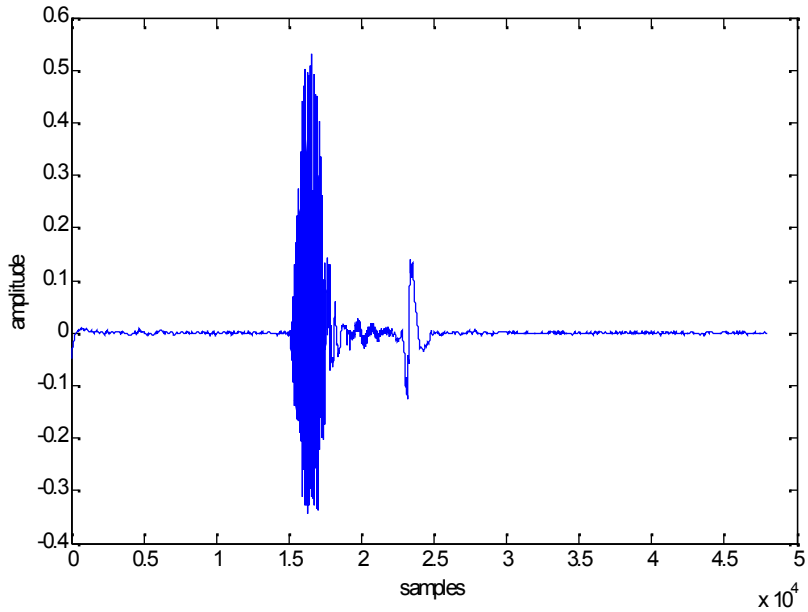

Figure 3.2 Pure recording speech 'aç'

3.2.2 Filtering

After getting the speech into the workspace, the voice signal should be filtered to eliminate noise caused by microphone and the environment. For this purpose a FIR (finite impulse response) bandpass filter is designed via MATLAB's filter design tool.

Before designing the filter, the speech signal is investigated using fast fourier transform. The speech signal is converted from time domain to frequency domain for looking up the noise frequency. It is seen from the frequency domain that the noise is mostly under 1500 Hz. It is also noted that we don't have much speech data above 8000 Hz. Therefore we have designed a FIR bandpass filter with cutoff frequencies 1500 and 8000 Hz.

To design the above specified filter, MATLAB's *fdatool* function is used (Figure 3.3). Filter specifications have been applied to the *fdatool* and it automatically calculates the filter coefficients which are useful for us.

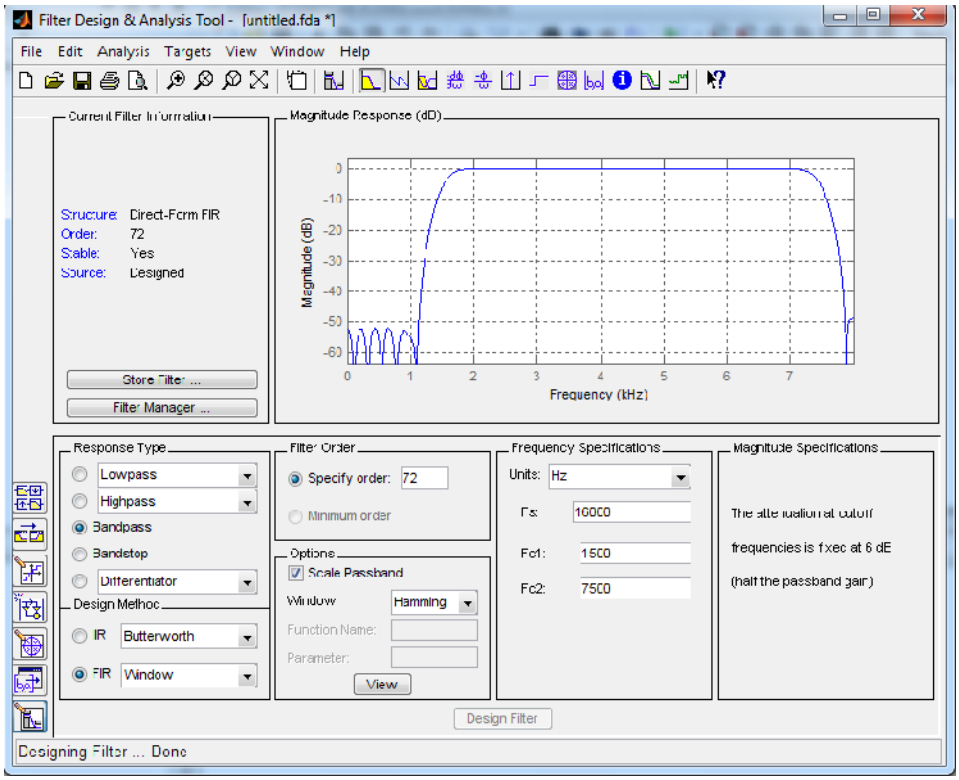

Figure 3.3 MATLAB Filter Design Tool

Order of the filter is specified as 72 because it is wanted sharpness at cutoff frequencies. FIR filter is designed with windowing function and so hamming window is selected as filtering window.

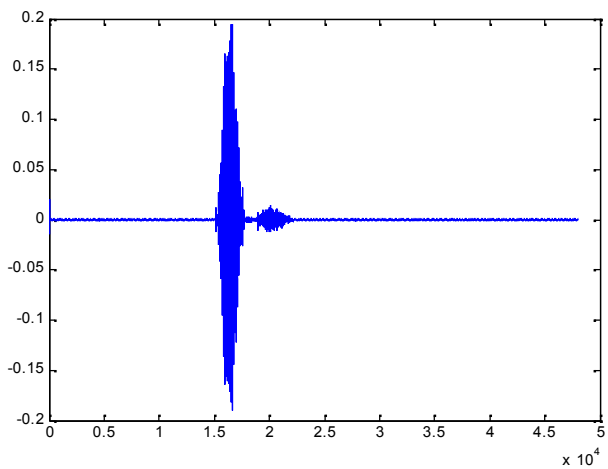

Figure 3.4 Filtered speech ‘aç’

The filter design tool calculates the filter coefficients for use. A function named *sesfiltre* is created. When this fuction is run, it records the speech for 3 seconds and outputs the filtered speech signal. MATLAB's *filter* function is used to implement the filter by using the calculated filter coefficients *'num'*. The code of the function sesfiltre can be found in the appendix section.

3.2.3 End Point Detection

In order to speed up process time we should clear the blanks in the signal start and end sides. To detect start-end points of speech a function in MATLAB called *endpoint* is created. Input to this function is the signal that has speech, and output of function is the only speech signal. The blank parts on the signal before and after the speech are cleared.

Briefly, the end point detection algorithm is based on the energy. A simple logic works in the algorithm as if the energy of the signal is above a threshold it means there is speech data on that section of the signal.

To calculate the energy, first square of the signal is taken and get rid of negative values. Then a kind of windowing is applied such that taking successive 400 values and taking averages of all these 400 values. This is done for eliminating instantaneous peak values occurred in the recording. Getting average values of all successive 400 sample values, the values which are above the threshold are considered as speech data and values below the threshold are eliminated.

Program code of the function can be found in the appendix section.

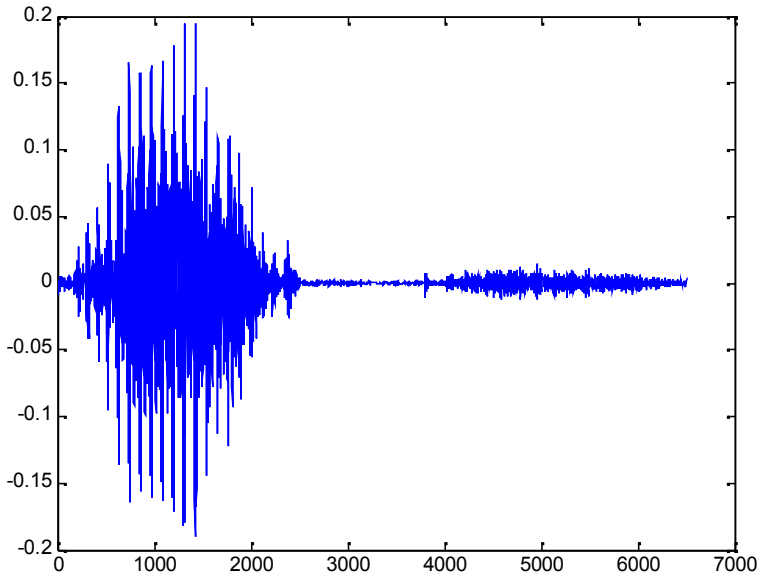

Figure 3.5 Start-end points detected speech 'aç'

3.2.4 MEL Cepstrum Calculation

Now there is only the speech data in hand. Speech features have to be extracted from this data. For this purpose a mel cepstrum calculator function in MATLAB is used. Function named *melcepst* is a function of the toolbox named *Voicebox* (Brookes 1997) which is created for voice and speech applications in MATLAB.

Function *melcepst* accepts speech signal and sampling rate in Hz as input for just simple use.

The speech signal and sampling rate as 16000 Hz with 12 coefficients are entered as input and got the mel cepstrum with 12 coefficients as output. For every sample speeches and command speech a cepstrum coefficients matrix is calculated. By using these matrices sample speeches are compared with the command speech in real time. The code of the function is given in the appendix section.

3.2.5 Distance Calculation

For every sample speech we have to compare the actual recording command speech with these samples. Tough the time durations of the speeches are not equal, dynamic time warping algorithm is used to align the time durations of speeches. For this purpose a function named *dtw* (Dynamic Time Warping) is used in MATLAB. It is a function that is not in MATLAB's toolboxes by default. It can be found in MATLAB's internet site, in file central (Dynamic Time Warping).

Function *dtw* accepts speech signals to be compared with each other as input. The use of the function is as follows:

```
[Dist,D,k,w]=dtw(t,r)
```

Dist is unnormalized distance between t and r
D is the accumulated distance matrix
k is the normalizing factor
w is the optimal path
t is the vector you are testing against
r is the vector you are testing

't' and 'r' are inputs and 'Dist', 'D', 'k', 'w' are outputs. Unnormalized distance is used between matrices and gives a score of minimum distance between two vectors. The code of this function is given in the appendix section.

3.2.6 Comparison and Deciding

Getting all distance scores of individually all samples, the program decides the recognized speech as the minimum distance score of all. It means, the speech which has the minimum distance score with the command speech, is the recognized speech and the program gives its output according to the recognized word.

3.3 Control of Serial Port

When the program decides which word is recognized, it sends data to the hardware which is designed and connected to the serial port. MATLAB has built-in functions to control the serial port easily. According to the recognized word, different data are sent to the hardware. The hardware (microcontroller) understands this data and activates the specified output of the recognized word.

A function called *seri* for controlling serial port is created on MATLAB. The code of the function and its usage is as follows :

```
function seri(a)  %input character
s1 = serial('COM1', 'BaudRate', 9600);%configure se.port
fopen(s1) %open port
fwrite(s1,a) %sends data
fclose(s1) %closes port
```

The function *seri* gets a character as an input and sends this character to the receiver. When configuring the serial port, a com port on the computer which we run our program is set and baud rate is specified in configuration. It is used 9600 bauds as baud rate for the communication which is enough for healthy communication. Other configurations are used as defaults which are 8 data bits, 1 stop bit, none parity, no flow control and LF terminator.

3.4 GUI (Graphical User Interface)

A graphical user interface has been designed for easy use of the program. You can train the system by recording samples, give them names and store, speak the command that you recorded before and see the result in the GUI.

When the program is run, the main control panel is shown on the screen. (Figure 3.6) You choose whether to do speech recognition or speaker recognition from this panel.

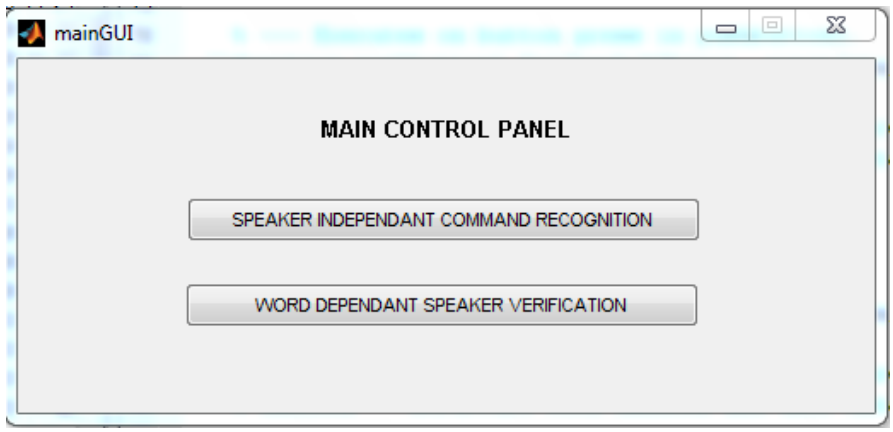

Figure 3.6 Main Control Panel

If ‘Speaker Independent Command Recognition’ button is selected, the GUI corresponding to speech recognition is opened. (Figure 3.7)

If ‘Word Dependant Speaker Verification’ is selected, the GUI corresponding to speaker recognition is opened. (Figure 3.8)

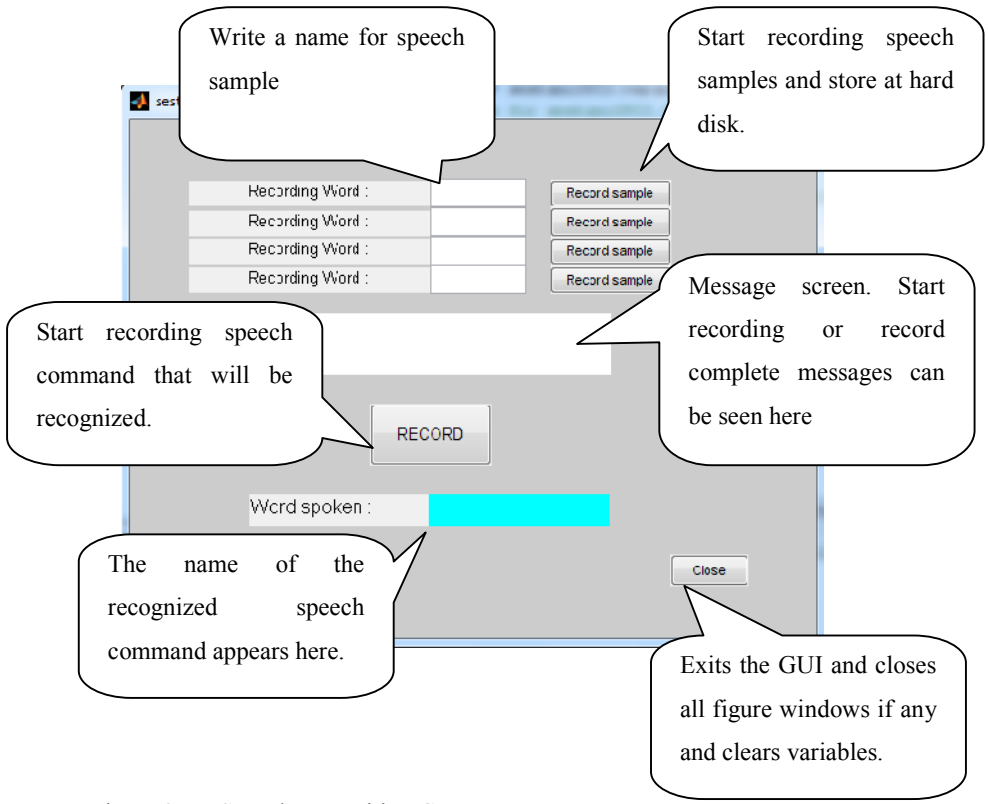

Figure 3.7 – Speech recognition GUI

To record speech samples for the program, first a name for the speech command is written and press ‘Record sample’ button. When this button is pressed 'Record for 3 secs... Press any key to start record...' appears in the message screen. After pressing any key it immediately starts recording and lasts 3 seconds. After 3 seconds when the recording is complete, the name of the recorded speech appears in the message screen that the recording is accomplished and stored in hard disk. This procedure is done for every four samples that will record.

If samples are recorded, then it is ready to use the program. Just press ‘RECORD’ button, speak your command and after 3 seconds, the name of the recognized speech command will be seen in the green area and the corresponding output will be active on the hardware board.

‘Close’ button closes the GUI, closes all opened windows during working of program such as figures and clears the variables from the global workspace.

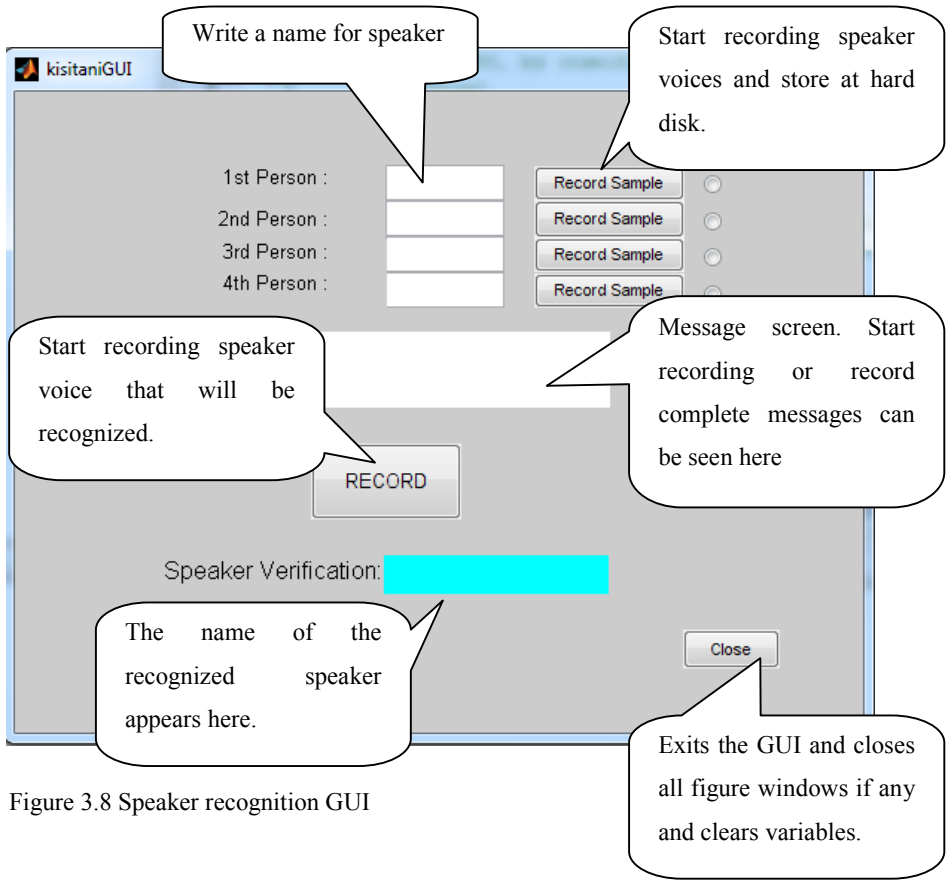

Figure 3.8 Speaker recognition GUI

For speaker recognition, similar processes as speech recognition are done. To record speaker speeches for the program, first a name for the speaker is written and press ‘Record sample’ button. When this button is pressed 'Record for 3 secs... Press any key to start record...' appears in the message screen. After pressing any key it immediately starts recording and lasts 3 seconds. During recording duration, all speakers must speak the same specified word such as’GİRİŞ’. After 3 seconds when the recording is complete, the name of the recorded speech appears in the message screen that the recording is accomplished and stored in hard disk. This procedure is done for every four samples that will record.

If samples are recorded, then it is ready to use the program. Just press ‘RECORD’ button, speak your word and after 3 seconds, the name of the recognized speaker will be seen in the green area and the corresponding output will be active on the hardware board.

‘Close’ button closes the GUI, closes all opened windows during working of program such as figures and clears the variables from the global workspace.

CHAPTER FOUR

HARDWARE DESIGN

Commands by speech are given to the computer and the computer compares and recognizes the speech as our command to the device connected to the computer. We simulate the device connected to the computer as LEDs controlled by serial port of the computer.

A PCB is designed and made which consists of serial port, power input plug, pic microcontroller and leds controlled by pic. This board basically gets data from computer via serial port and the microcontroller decides which led to be on according to the recognized speech on the computer.

4.1 Serial Communication

In telecommunication and computer science, serial communication is the process of sending data one bit at one time, sequentially, over a communication channel or computer bus. This is in contrast to parallel communication, where several bits are sent together, on a link with several parallel channels. Serial communication is used for all long-haul communication and most computer networks, where the cost of cable and synchronization difficulties make parallel communication impractical. (Serial Communication)

Serial communication has many protocols but the most common and used by so many years in computers is RS232.

4.1.1 RS232 Serial Protocol

In telecommunications, RS-232 (Recommended Standard 232) is a standard for serial binary data signals connecting between a *DTE* (Data Terminal Equipment) and a *DCE* (Data Circuit-terminating Equipment). It is commonly used in computer serial ports. (RS-232)

Electronic data communications between elements will generally fall into two broad categories: single-ended and differential. RS232 (single-ended) was introduced in 1962, and despite rumors for its early demise, has remained widely used through the industry. The specification allows for data transmission from one transmitter to one receiver at relatively slow data rates (up to 20K bits/second) and short distances (up to 50Ft. at the maximum data rate).

Independent channels are established for two-way (full-duplex) communications. The RS232 signals are represented by voltage levels with respect to a system common (power / logic ground). The "idle" state (MARK) has the signal level negative with respect to common, and the "active" state (SPACE) has the signal level positive with respect to common. RS232 has numerous handshaking lines (primarily used with modems), and also specifies a communications protocol. In general if you are not connected to a modem the handshaking lines can present a lot of problems if not disabled in software or accounted for in the hardware (loop-back or pulled-up). RTS (Request to send) does have some utility in certain applications. (RS-485)

Voltages of -3v to -15v with respect to signal ground are considered logic '1' (the marking condition), whereas voltages of +3v to +15v are considered logic '0' (the spacing condition). The range of voltages between -3v and +3v is considered a transition region for which a signal state is not assigned. (Tech RS232)

The pins and descriptions of signals on a DB9 RS232 serial interface are as follows:

Pin No.	Name	Dir	Notes/Description
1	DCD	IN	Data Carrier Detect. Raised by DCE when modem synchronized.
2	RD	IN	Receive Data (a.k.a RxD, Rx). Arriving data from DCE.
3	TD	OUT	Transmit Data (a.k.a TxD, Tx). Sending data from DTE.
4	DTR	OUT	Data Terminal Ready. Raised by DTE when powered on. In auto-answer mode raised only when RI arrives from DCE.
5	SGND	-	Ground
6	DSR	IN	Data Set Ready. Raised by DCE to indicate ready.
7	RTS	OUT	Request To Send. Raised by DTE when it wishes to send. Expects CTS from DCE.
8	CTS	IN	Clear To Send. Raised by DCE in response to RTS from DTE.
9	RI	IN	Ring Indicator. Set when incoming ring detected - used for auto-answer application. DTE raised DTR to answer.

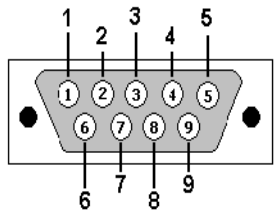

Figure 4.1 RS232 Serial interface (Tech RS232)

Usually, not all pins and signals are used. Only receive, transmit and ground pins are required for proper and basic serial communication. In that way, only pins 2, 3 and 5 (receive, transmit and ground respectively) are used in communication on purpose at the project.

4.1.2 USART(The Universal Synchronous Asynchronous Receiver Transmitter)

The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the two serial I/O modules. (USART is also known as a Serial Communications Interface or SCI). The USART can be configured as a full duplex asynchronous system that can communicate with peripheral devices such as CRT terminals and personal computers, or it can be configured as a half duplex synchronous system that can communicate with peripheral devices such as A/D or D/A integrated circuits, serial EEPROMs etc.

The USART can be configured in the following modes:

- Asynchronous (full duplex)
- Synchronous - Master (half duplex)
- Synchronous - Slave (half duplex)

The USART module also has a multi-processor communication capability using 9-bit address detection.(Microchip (1999))

Asynchronous mode communication is applied on the communication between software – MATLAB and hardware – PIC. The Universal Asynchronous Receiver/Transmitter (UART) controller is the key component of the serial communications subsystem of a computer. The UART takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Serial transmission of digital information (bits) through a single wire or other medium is much more cost effective than parallel transmission through multiple wires. A UART is used to convert the transmitted information between its sequential and parallel form at each end of the link. Each UART contains a shift register which is the fundamental method of conversion between serial and parallel forms.

The UART usually does not directly generate or receive the external signals used between different items of equipment. Typically, separate interface devices are used to convert the logic level signals of the UART to and from the external signaling levels.

External signals may be of many different forms. Examples of standards for voltage signaling are RS-232, RS-422 and RS-485 from the EIA. Historically, the presence or absence of current (in current loops) was used in telegraph circuits. Some signaling schemes do not use electrical wires. Examples of such are optical fiber, IrDA (infrared), and (wireless) Bluetooth in its Serial Port Profile (SPP). Some signaling schemes use modulation of a carrier signal (with or without wires).

Examples are modulation of audio signals with phone line modems, RF modulation with data radios, and the DC-LIN for power line communication.

Communication may be "full duplex" (both send and receive at the same time) or "half duplex" (devices take turns transmitting and receiving).

As of 2008, UARTs are commonly used with RS-232 for embedded systems communications. It is useful to communicate between microcontrollers and also with PCs. Many chips provide UART functionality in silicon, and low-cost chips exist to convert logic level signals (such as TTL voltages) to RS-232 level signals (for example, Maxim's MAX232).(USART)

4.2 Microcontroller PIC 16F877

PIC 16F877 is a midrange microcontroller with 40 pins. It can work at up to 20Mhz clock speed. PIC 16F877 is preferred in the project because of its easy programming and support of serial communication with USART(Universal Synchronous Asynchronous Receiver Transmitter) on hardware. The corresponding PIC C source code of the PIC can be found in the appendix section.

4.2.1 Core Features

• High performance RISC CPU
• Only 35 single word instructions to learn
• All single cycle instructions except for program branches which are two cycle
• Operating speed: DC - 20 MHz clock input DC - 200 ns instruction cycle
• Up to 8K x 14 words of FLASH Program Memory,
  Up to 368 x 8 bytes of Data Memory (RAM)
  Up to 256 x 8 bytes of EEPROM Data Memory
• Interrupt capability (up to 14 sources)
• Eight level deep hardware stack
• Power-on Reset (POR)

• Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
• Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
• Programmable code protection
• Power saving SLEEP mode
• Selectable oscillator options
• Low power, high speed CMOS FLASH/EEPROM technology
• Fully static design
• In-Circuit Serial Programming (ICSP) via two pins
• Single 5V In-Circuit Serial Programming capability
• In-Circuit Debugging via two pins
• Processor read/write access to program memory
• Wide operating voltage range: 2.0V to 5.5V
• High Sink/Source Current: 25 mA ranges
• Low-power consumption:
- < 0.6 mA typical @ 3V, 4 MHz
- 20 μA typical @ 3V, 32 kHz
- < 1 μA typical standby current (Microchip (1999))

4.2.2 Peripheral Features

• Timer0: 8-bit timer/counter with 8-bit prescaler
• Timer1: 16-bit timer/counter with prescaler, can be incremented during SLEEP via external crystal/clock
• Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
• Two Capture, Compare, PWM modules
- Capture is 16-bit, max. resolution is 12.5 ns
- Compare is 16-bit, max. resolution is 200 ns
- PWM max. resolution is 10-bit
• 10-bit multi-channel Analog-to-Digital converter
• Synchronous Serial Port (SSP) with SPI (Master mode) and I2C (Master/Slave)
• Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection

• Parallel Slave Port (PSP) 8-bits wide, with external RD, WR and CS controls

• Brown-out detection circuitry for Brown-out Reset (BOR) (Microchip (1999))

4.2.3 Pinning Diagram

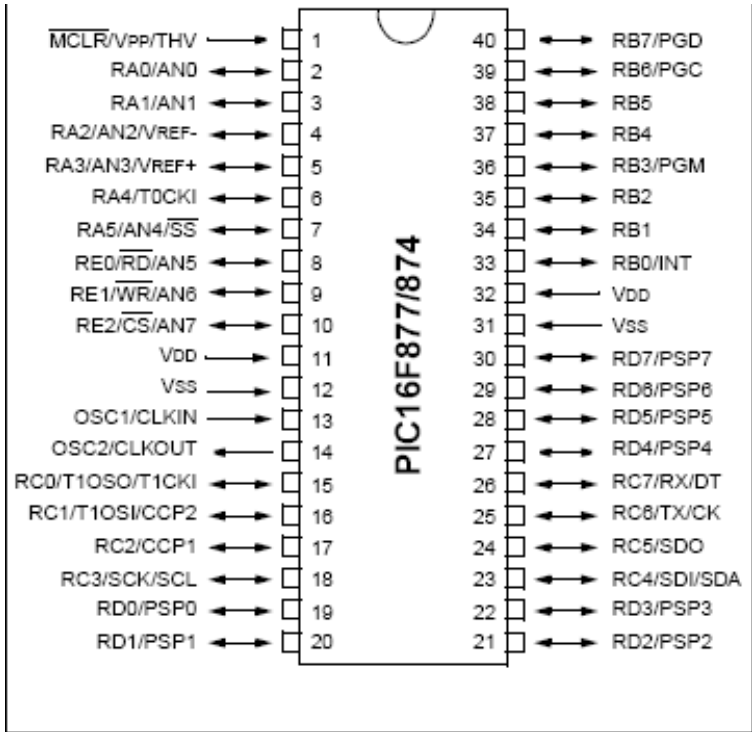

Figure 4.2 PIC 16F877 Pinning Diagram(Microchip (1999))

4.3 MAX 232 Integrated Circuit

The MAX232 is an integrated circuit that converts signals from an RS-232 serial port to signals suitable for use in TTL compatible digital logic circuits. The MAX232 is a dual driver/receiver and typically converts the RX, TX, CTS and RTS signals.

The receivers reduce RS-232 inputs (which may be as high as ± 25 V), to 39tandard 5 V TTL levels. These receivers have a typical threshold of 1.3 V, and a typical hysteresis of 0.5 V.(MAX232)

Main features of the MAX232 IC which I have used on the hardware are:

- Meets or Exceeds TIA/EIA-232-F and ITU Recommendation V.28
- Operates From a Single 5-V Power Supply With 1.0-_F Charge-Pump Capacitors
- Operates Up To 120 kbit/s
- Two Drivers and Two Receivers
- ±30-V Input Levels
- Low Supply Current . . . 8 mA Typical
- ESD Protection Exceeds JESD 22 – 2000-V Human-Body Model (A114-A)
- Upgrade With Improved ESD (15-kV HBM) and 0.1-_F Charge-Pump Capacitors is Available With the MAX202
- Applications – TIA/EIA-232-F, Battery-Powered Systems, Terminals, Modems, and Computers (Texas Instruments (2004))

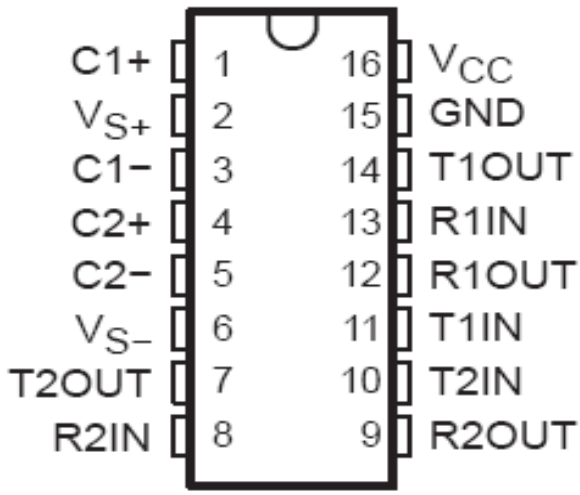

Figure 4.3 MAX 232 IC Pinning Diagram

(Texas Instruments (2004))

4.4 Output Circuit Board

This board is used to get the output data from the computer via serial port. According to the recognized word, the corresponding output becomes active on the board. The circuit diagram schematics of the board can be found in Figure 4.4. In the schematics, power supply connections of the microcontroller are not shown as these are assumed to be connected.

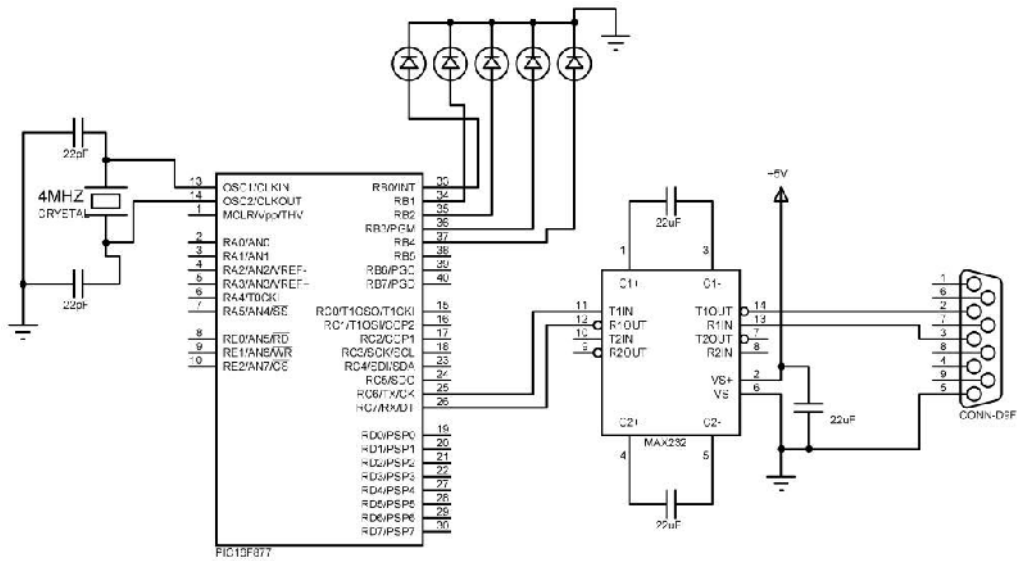

Figure 4.4 Circuit board

CHAPTER FIVE

EXPERIMENTAL WORK

As mentioned in previous chapters, all the work is done on a computer with MATLAB. In this chapter, brief instructions on how to use the control software will be mentioned.

When the program is started, first a main control GUI appears (Figure 5.1).

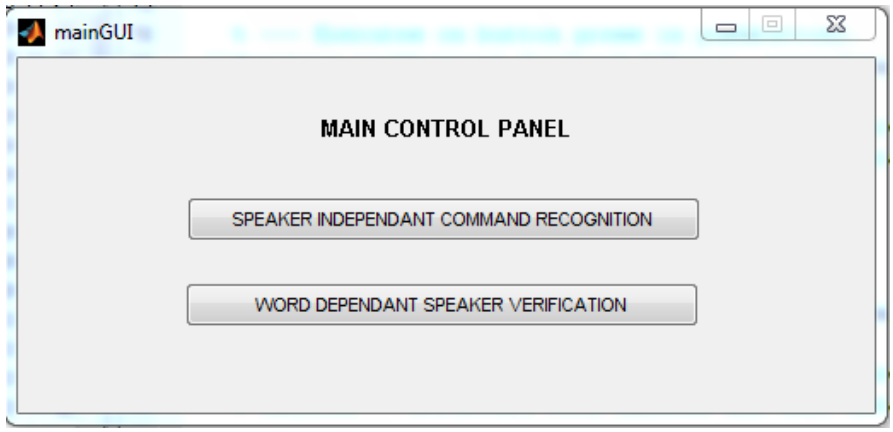

Figure 5.1 Main control GUI

From this control panel speaker independent command recognition or word dependant speaker verification is chosen and clicked on the appropriate button.

5.1 Speaker Independent Command Recognition

If speaker independent command recognition button is selected, a new control panel GUI appears (Figure 5.2). If there is no sample commands recorded before, the system must be trained and sample commands must be recorded.

For training, 4 different words are spoken and recorded. A name is written for each word, ‘record sample’ button is pressed, a message indicating recording duration appears and your speech is saved after the duration. (Figure 5.2)

Figure 5.2 Speaker independent command recognition control panel

For experimental work, 4 different words; 'aç', 'kapat', 'başla' and 'dur' are spoken and recorded. These 4 words are associated with their filenames (Figure 5.2) and saved on computers hard disk.

After speaking and recording of sample words, training process is complete. Sample words are spoken by only me but commands are spoken by me and my sister to test speaker dependency and independency. To speak speech commands 'record' button is pressed and command is spoken during 3 seconds of duration. End of 3 seconds, the system records the speech and makes comparison between spoken command and samples. Within a few seconds the result appears in the green area on the GUI. If the spoken command is one of the sample words, the filename of the recognized word appears on the 'word spoken' area (Figure 5.3). If spoken command is not one of the samples, a message indicating that the spoken word is not

recognized appears on the green area on the GUI. At the moment the system makes the comparison and gives the result, the corresponding output becomes active and a green led of the corresponding word illuminates (Figure 5.?).

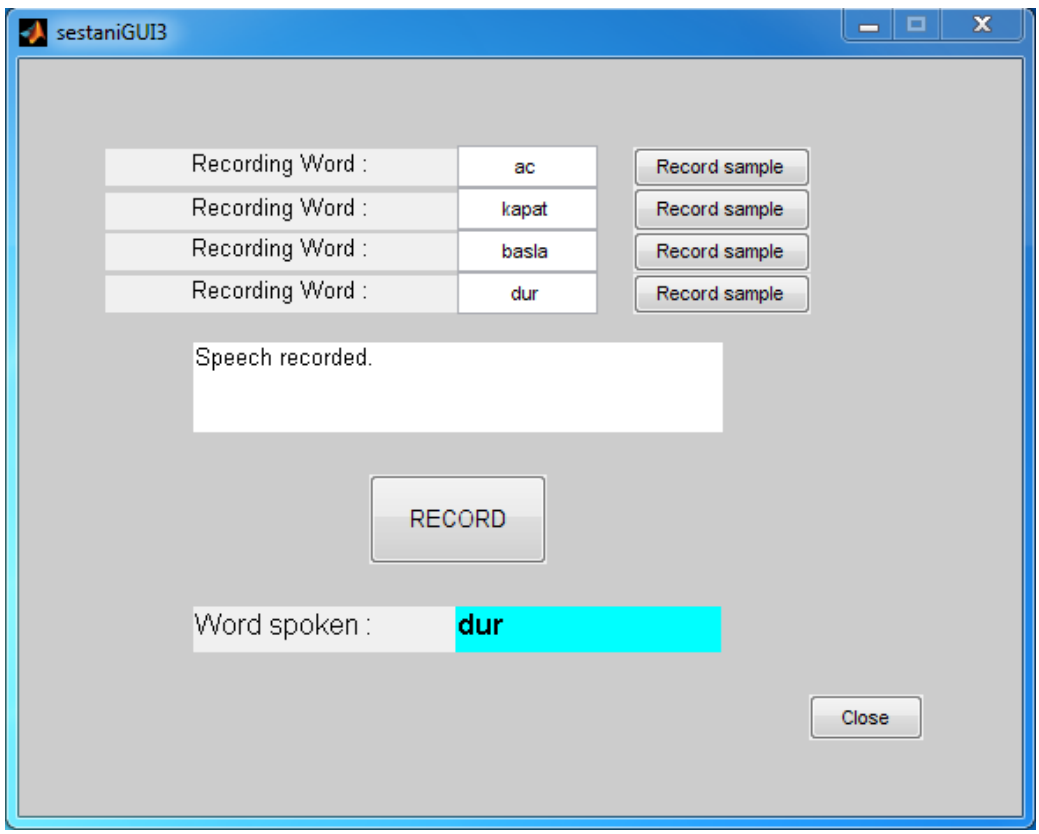

Figure 5.3 Command ‘dur’ is recognized as a result.

5.2 Word Dependant Speaker Verification

If word dependant speaker verification button is pressed, a new control panel GUI of speaker verification appears (Figure 5.4). If a database of sample speakers not created, four different speakers must record a certain word separately to create samples.

For training, 4 different speakers speak and record a certain word such as ‘giriş’. Speakers name is written for each speaker, ‘record sample’ button is pressed, a message indicating recording duration appears and your speech is saved after the duration. (Figure 5.4)

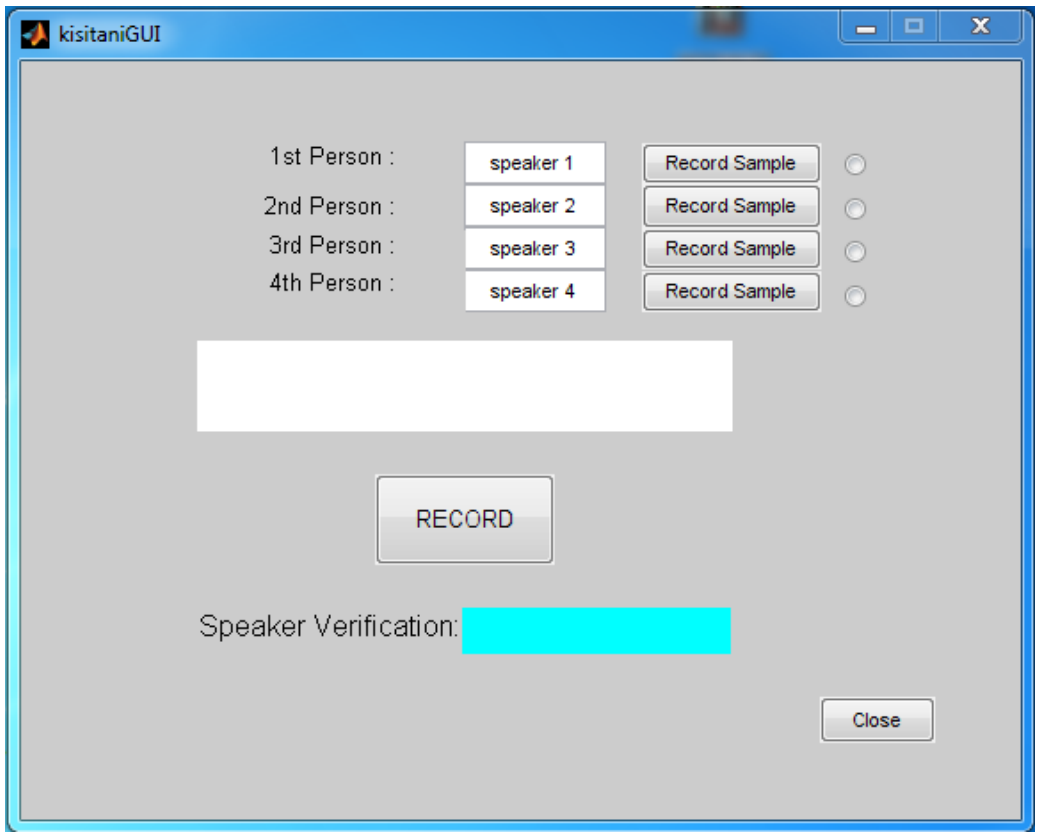

Figure 5.4 Word dependant speaker verification control panel

For experimental work, 4 different speakers voice; ‘speaker 1’, ‘speaker 2’, ‘speaker 3’ and ‘speaker 4’ are recorded. These 4 speakers are associated with their filenames (Figure 5.5) and saved on computers hard disk.

After speaking and recording of samples, training process is complete. Sample voices are spoken by my family members. To speak verification word ‘record’ button is pressed and word is spoken during 3 seconds of duration. End of 3 seconds, the system records the speech and makes comparison between spoken word and samples. Within a few seconds the result appears in the green area on the GUI. If the spoken word belongs to one of the sample speakers, the filename of the recognized speaker appears on the ‘speaker verification’ area (Figure 5.5). If spoken speaker is not one of the samples, a message indicating that the spoken speaker is not recognized appears on the green area on the GUI. At the moment the system makes the

comparison and gives the result, the corresponding output becomes active and a green led of the corresponding word illuminates (Figure 5.6).

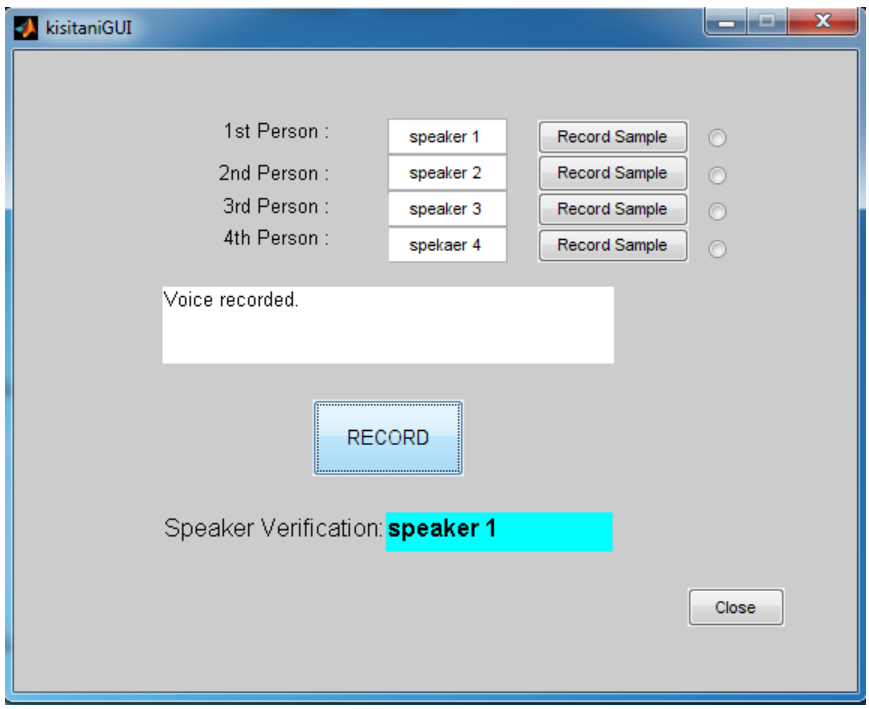

Figure 5.5 ‘speaker 1’ is verified.

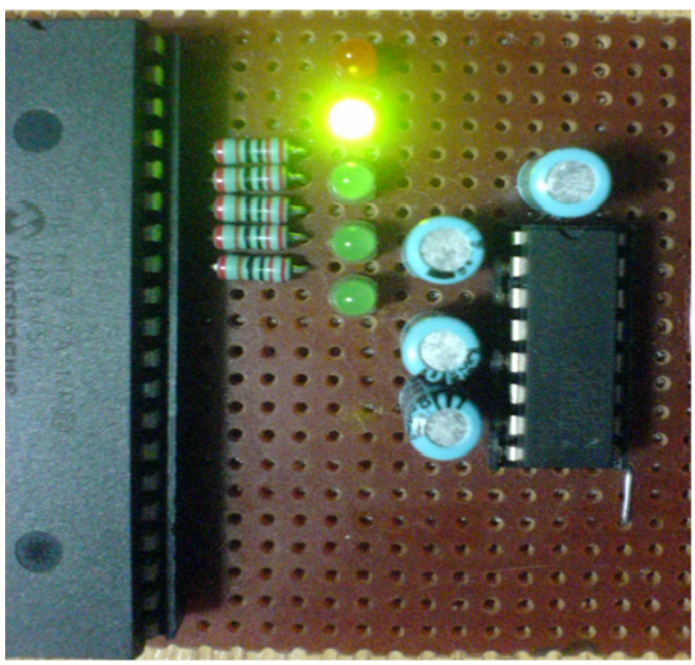

Figure 5.6 Green led illuminates on output circuit board

CHAPTER SIX

CONCLUSION

6.1 Results

The speech recognition part of the project is designed as a speaker independent system so that the system is tested with same speaker also with diffent speakers. The system is designed for 4 words for now. Though it is planned to be small databased system, four words is enough for beginning. It can easily be upgraded to more vocabulary databased.

The program is designed and worked on a laptop computer Intel Centrino 2 Core Duo P8400 (2.26 GHz) processor with 3GB memory on a Windows 7 operating system. The MATLAB version which have been worked is 7.6. All the codes are written and implemented on this configured laptop computer.

The tests are done by a standard microphone with no specific properties. The testing environment was a default living room with default noise and not specially designed for recording purposes.

Just for speech recognition test, these four different words, ‘AÇ’, ‘KAPAT’, ‘BAŞLA’ and ‘DUR’ are recorded and sampled.

Table 6.1 Results for speech recognition

	AÇ	KAPAT	BAŞLA	DUR
Speaker Dependant	%87	%90	%90	%87
Speaker Independant	%80	%84	%84	%80

The tests are performed by me and my sister. The first test was with same speaker, same word meaning speaker dependant. 30 successive tries for each word are performed. With the word ‘AÇ’ and ‘DUR’ approximately 87% recognition success

is achieved. With other words, ‘KAPAT’ and ‘BAŞLA’ approximately 90% recognition success is achieved. The recognition rates are calculated based on accuracy method. It is the successful recognition number per total number of tryings. (Table 6.1)

The second test was with different speakers to test speaker independency. The samples are spoken by me and the commands by my sister. 30 successive tries for each word are performed. With the word ‘AÇ’ and ‘DUR’ approximately 80% recognition success is achieved. With other words, ‘KAPAT’ and ‘BAŞLA’ approximately 84% recognition success is achieved. Also in the second test the recognition rates are calculated based on accuracy method. It is the successful recognition number per total number of tryings. (Table 6.1)

For speaker recognition, 4 different persons spoke the same word ‘GİRİŞ’ as a sample. Then each speaker tested the system for 30 times with the same word. For speaker 1 and speaker 3 approximately 93% recognition success is achieved. For speakers 2 and 4, approximately 90% recognition success is achieved. Again, the recognition rates are calculated based on accuracy method. (Table 6.2)

Table 6.2 Speaker recognition results

	Speaker 1	Speaker 2	Spekaer 3	Speaker 4
For the word ‘GİRİŞ’	%93	%90	%93	%90

6.2 Conclusion

The system control using voice input, is designed to be used at home or any other place to assist people in their daily life. For this purpose, it is designed for quick and simple use. Unlike other works on this subject, it is not needed to record many samples. Just one sample is enough for the system to train and work. This gives the advantage of simple, mobile and quick use.

The tests are performed in a standard living room with no especially designed for recording purposes. It was somehow quiet in the room during tests but the environmental noise is a big factor on recognition success. It should be very quiet during recordings to achieve good results. In some investigated works on speech recognition, speech recordings are done in a noise-free room so recognition rates of these works are better than this work.

Another big factor is the microphone. Since it is used here a standard PC microphone, microphone sourced noises cause big problems. Microphones which are especially designed for recording purposes with low noise levels, should be used to achieve better results.

Also the working platform, computer is a factor on the results. This work is performed on a laptop computer with on-board sound card. In some cases noise glitches caused by the computer hard disk could be seen on the recorded speeches plots. But with a separate sound card with good filtering on mic-in jacks, will definitely have a positive effect.

As predicted, speaker dependant recognition has better results than speaker independent. The main reason of this is, the speech, the pronunciation differs from person to person. The methods that we use for feature extraction, MFCC, use spectral features of speech so the speech is affected by the vocal tracts or respiratory. These effects are reflected to recognition rates. Because of this, spekar dependant recognition gives better results. This also proofs that speaker recognition works better than speech recognititon.

To achieve the best result, we have to speak the command exactly the in the same way and pronunciation with the recorded sample. Since this is difficult for different persons, it is also difficult for the same speaker.

One of the biggest problems in this work was noises and glitches. When there is noise in the environment or even in your computer, it becomes very hard to successfully detect start-end points of speech. Some noises are not filtered after filtering and they affect the start – end point detection. Usually noises are thought to be speech data by the software. And these noises cause wrong recognition results.

Just to work on signal processing and speech recognition we are using a very small vocabulary database. Since we are using small database, the methods and techniques that we have used are suitable for our application. In case of large vocabulary databases, these methods also work but I believe it will be reasonable to use another methods such as Hidden Markov Models.

To achieve better results, may be better filtering and end – point detection algorithms can be used to eliminate hard noises. Because the methods are useful for speech recognition the main problem is to get rid of noises to make the end – point detection and feature extraction work healthy.

REFERENCES

Anonymous, (n.d), *A project report on speaker recogniton implemented in matlab*

Brookes, M. (1997), *Voicebox,* retrieved October 2007, from http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html

Kale K. R., (2006), *Dynamic Time Warping*, retrieved April 2008, from http://www.cnel.ufl.edu/~kkale/dtw.html.

Microchip (1999) , *PIC16F87x Datasheet,* Microchip Inc.

Proakis, J. & Manolakis, D. (1996*). Digital Signal Processing Principles, Algorithms and applications(3rd edition)*. New Jersey : Prentice-Hall Inc.

Rabiner, L. & Juang, B. (1993). *Fundamentals of Speech Recognition,* New Jersey : Prentice-Hall Inc.

Rath, M. T., & Manmatha, R. (2003). Word Image Matching Using Dynamic Time Warping. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03) - Volume 2*, 2003

Sakoe, H. & Chiba, S. (1978) Dynamic programming algorithm optimization for spoken word recognition, *IEEE(1978)*

Sheonei B. A. (2006), *Introduction to Digital Signal Processing and Filter Design,* John Wiley & Sons Inc. New Jersey.

Sigmund M. (2008). Automatic Speaker Recognition by Speech Signal. In A. Zemliak, (Ed.), *Frontiers in Robotics, Automation and Control* (41-54). Croatia: InTech Education and Publishing.

Texas Instruments (2004), *MAX232 Datasheet,* Texas Instruments Incorporated

The Mathworks Inc. (2009), *Dynamic Time Warping,* retrieved February 2009, from http://www.mathworks.com/matlabcentral/fileexchange/6516

The RE Smith (2009), *RS-485*, retrieved May 2009, from http://www.rs485.com/rs485spec.html

Wikimedia Inc. (2009), *MAX232*, retrieved May 2009, from http://en.wikipedia.org/wiki/MAX232

Wikimedia Inc. (2009), *RS-232*, retrieved May 2009, from http://en.wikipedia.org/wiki/RS-232

Wikimedia Inc. (2009), *Serial Communication*, retrieved May 2009, from http://en.wikipedia.org/wiki/Serial_communication

Wikimedia Inc. (2009), *USART*, retrieved May 2009, from http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter

Zytrax Inc. (2009), *Tech RS232*, retrieved May 2009, from http://www.zytrax.com/tech/layer_1/cables/tech_rs232.htm

APPENDICES

Appendix A. Source codes of MATLAB GUI and functions

```
function varargout = sestaniGUI3(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@sestaniGUI3_OpeningFcn, ...
                   'gui_OutputFcn',
@sestaniGUI3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...

                   'gui_Callback',   []);
if nargin & isstr(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
% End initialization code - DO NOT EDIT


% --- Executes just before sestaniGUI3 is made visible.
function sestaniGUI3_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
% varargin   command line arguments to sestaniGUI3 (see
VARARGIN)

% Choose default command line output for sestaniGUI3
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
% UIWAIT makes sestaniGUI3 wait for user response (see
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the
command line.
function varargout = sestaniGUI3_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
% fs=16000;
% duration=3;
set(handles.text5,'String','Kayit 3sn surecektir. Kayda
baslamak icin bir tusa basin.');
pause;
% set(handles.text5,'String','Ses kaydediliyor...');

ses1f=sesfiltre();
set(handles.text5,'String','Sesiniz alindi.');

ses1e=endpoint(ses1f);
ses11=melcepst(ses1e,16000);
guidata(hObject,handles)
figure,plot(ses1e)

guidata(hObject,handles)

globalLoad('ornek1.mat');
globalLoad('ornek2.mat');
```

```
globalLoad('ornek3.mat');
globalLoad('ornek4.mat');

distance(1)=dtw(ses11,ornek11)
distance(2)=dtw(ses11,ornek22)
distance(3)=dtw(ses11,ornek33)
distance(4)=dtw(ses11,ornek44)

min_dist=min(distance)
while min_dist <= 500
    switch min_dist
        case distance(1)
        set(handles.text7,'String',isim1);
        seriport('1');
        case distance(2)
        set(handles.text7,'String',isim2);
        seriport('2');
        case distance(3)
        set(handles.text7,'String',isim3);
        seriport('3');
        case distance(4)
        set(handles.text7,'String',isim4);
        seriport('4');
    end
end
if min_dist > 500
       set(handles.text7,'String','Kelime taninamadi');
       seriport('e');
end

clear all

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1
as text
%        str2double(get(hObject,'String')) returns
contents of edit1 as a double

% isim1=get(hObject,'String');
%
% handles.isim1 = isim1;
% guidata(hObject,handles)
```

```
% --- Executes during object creation, after setting all
properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2
as text
%        str2double(get(hObject,'String')) returns
contents of edit2 as a double


% --- Executes during object creation, after setting all
properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3
as text
%        str2double(get(hObject,'String')) returns
contents of edit3 as a double


% --- Executes during object creation, after setting all
properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4
as text
%        str2double(get(hObject,'String')) returns
contents of edit4 as a double
```

```
% --- Executes during object creation, after setting all
properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
%
% fs=16000;
% duration=3;
set(handles.text5,'String','Kayit 3sn surecektir. Kayda
baslamak icin bir tusa basin.');
pause;
ornek1f=sesfiltre();
set(handles.text5,'String','Sesiniz alindi.');
ornek1e=endpoint(ornek1f);
ornek11=melcepst(ornek1e,16000);
guidata(hObject,handles)

isim1=get(handles.edit1,'String');
set(handles.text5,'String',isim1)
guidata(hObject,handles)
figure,plot(ornek1e)

save ('ornek1.mat', 'ornek11','isim1','ornek1e');

% --- Executes on button press in pushbutton4.
```

```
function pushbutton4_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% fs=16000;
% duration=3;
set(handles.text5,'String','Kayit 3sn surecektir. Kayda
baslamak icin bir tusa basin.');
pause;
ornek2f=sesfiltre();
set(handles.text5,'String','Sesiniz alindi.');
ornek2e=endpoint(ornek2f);
ornek22=melcepst(ornek2e,16000);
guidata(hObject,handles)

isim2=get(handles.edit2,'String');
set(handles.text5,'String',isim2)
guidata(hObject,handles)
figure,plot(ornek2e)

save ('ornek2.mat', 'ornek22','isim2','ornek2e');

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% fs=16000;
% duration=3;
set(handles.text5,'String','Kayit 3sn surecektir. Kayda
baslamak icin bir tusa basin.');
pause;
ornek3f=sesfiltre();
ornek3e=endpoint(ornek3f);
ornek33=melcepst(ornek3e,16000);
guidata(hObject,handles)

isim3=get(handles.edit3,'String');
set(handles.text5,'String',isim3)
guidata(hObject,handles)
```

```
figure,plot(ornek3e)

save ('ornek3.mat', 'ornek33','isim3','ornek3e');

% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% fs=16000;
% duration=3;
set(handles.text5,'String','Kayit 3sn surecektir. Kayda
baslamak icin bir tusa basin.');
pause;
ornek4f=sesfiltre();
ornek4e=endpoint(ornek4f);
ornek44=melcepst(ornek4e,16000);
guidata(hObject,handles)

isim4=get(handles.edit4,'String');
set(handles.text5,'String',isim4)
guidata(hObject,handles)
figure,plot(ornek4e)

save ('ornek4.mat', 'ornek44','isim4','ornek4e');


% --- Executes during object creation, after setting all
properties.
function text5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text5 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called




% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton7 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)


clear all
close all
close
```

```
function varargout = mainGUI(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @mainGUI_OpeningFcn,
...
                   'gui_OutputFcn',  @mainGUI_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
% End initialization code - DO NOT EDIT


% --- Executes just before mainGUI is made visible.
function mainGUI_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
% varargin   command line arguments to mainGUI (see
VARARGIN)

% Choose default command line output for mainGUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes mainGUI wait for user response (see
UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the
command line.
function varargout = mainGUI_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

sestaniGUI3

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

kisitaniGUI
```

```
function ses = endpoint(y)

%end point detection%
leny=length(y);
y2=y.^2;
for i=0:(leny/400)-1
    for n=(i*400)+1:(i+1)*400
        y2buff(n)=y2(n);
        y2avg((i+1))=mean(y2buff(n));
    end
end
s=0;
y2avg=10^7.*(y2avg);
y2avg=(y2avg)';
for k=1:(leny/400)
    if y2avg(k) > 750
        s=s+1;
        x(s)=k;
    end
end
start=400*min(x)-5;
endpoint=400*max(x)+5;
ses=y(start:endpoint);
```

```
function c=melcepst(s,fs,w,nc,p,n,inc,fl,fh)
%MELCEPST Calculate the mel cepstrum of a signal
C=(S,FS,W,NC,P,N,INC,FL,FH)
%
%
% Simple use: c=melcepst(s,fs)  % calculate mel cepstrum
with 12 coefs, 256 sample frames
%                 c=melcepst(s,fs,'e0dD') % include log
energy, 0th cepstral coef, delta and delta-delta coefs
%
% Inputs:
%     s  speech signal
%     fs  sample rate in Hz (default 11025)
%     nc  number of cepstral coefficients excluding 0'th
coefficient (default 12)
%     n   length of frame (default power of 2 <30 ms))
%     p   number of filters in filterbank (default
floor(3*log(fs)) )
%     inc frame increment (default n/2)
%     fl  low end of the lowest filter as a fraction of
fs (default = 0)
%     fh  high end of highest filter as a fraction of fs
(default = 0.5)
%
%       w   any sensible combination of the following:
%
%               'R'  rectangular window in time domain
%               'N' Hanning window in time domain
%               'M' Hamming window in time domain
(default)
%
%             't'  triangular shaped filters in mel
domain (default)
%             'n'  hanning shaped filters in mel domain
%             'm'  hamming shaped filters in mel domain
%
%               'p' filters act in the power domain
%               'a' filters act in the absolute magnitude
domain (default)
%
%              '0'  include 0'th order cepstral
coefficient
%               'e'  include log energy
%               'd' include delta coefficients (dc/dt)
%               'D' include delta-delta coefficients
(d^2c/dt^2)
%
```

```
%             'z'  highest and lowest filters taper down
to zero (default)
%             'y'  lowest filter remains at 1 down to 0
frequency and
%                  highest filter remains at 1 up to
nyquist freqency
%
%              If 'ty' or 'ny' is specified, the total
power in the fft is preserved.
%
% Outputs:  c     mel cepstrum output: one frame per row
%


%      Copyright (C) Mike Brookes 1997
%
%      Last modified Thu Jun 15 09:14:48 2000
%
%   VOICEBOX is a MATLAB toolbox for speech processing.
Home page is at
%
http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.htm
l
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
%   This program is free software; you can redistribute
it and/or modify
%   it under the terms of the GNU General Public License
as published by
%   the Free Software Foundation; either version 2 of the
License, or
%   (at your option) any later version.
%
%   This program is distributed in the hope that it will
be useful,
%   but WITHOUT ANY WARRANTY; without even the implied
warranty of
%   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the
%   GNU General Public License for more details.
%
%   You can obtain a copy of the GNU General Public
License from
%   ftp://prep.ai.mit.edu/pub/gnu/COPYING-2.0 or by
writing to
%   Free Software Foundation, Inc.,675 Mass Ave,
Cambridge, MA 02139, USA.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%

if nargin<2 fs=11025; end
if nargin<3 w='M'; end
if nargin<4 nc=12; end
if nargin<5 p=floor(3*log(fs)); end
if nargin<6 n=pow2(floor(log2(0.03*fs))); end
if nargin<9
   fh=0.5;
   if nargin<8
     fl=0;
     if nargin<7
        inc=floor(n/2);
     end
  end
end

if any(w=='R')
   z=enframe(s,n,inc);
elseif any (w=='N')
   z=enframe(s,hanning(n),inc);
else
   z=enframe(s,hamming(n),inc);
end
f=rfft(z.');
[m,a,b]=melbankm(p,n,fs,fl,fh,w);
pw=f(a:b,:).*conj(f(a:b,:));
pth=max(pw(:))*1E-6;
if any(w=='p')
   y=log(max(m*pw,pth));
else
   ath=sqrt(pth);
   y=log(max(m*abs(f(a:b,:)),ath));
end
c=rdct(y).';
nf=size(c,1);
nc=nc+1;
if p>nc
   c(:,nc+1:end)=[];
elseif p<nc
   c=[c zeros(nf,nc-p)];
end
if ~any(w=='0')
   c(:,1)=[];
end
if any(w=='e')
   c=[log(sum(pw)).' c];
```

```
end

% calculate derivative

if any(w=='D')
  vf=(4:-1:-4)/60;
  af=(1:-1:-1)/2;
  ww=ones(5,1);
  cx=[c(ww,:); c; c(nf*ww,:)];
  vx=reshape(filter(vf,1,cx(:)),nf+10,nc);
  vx(1:8,:)=[];
  ax=reshape(filter(af,1,vx(:)),nf+2,nc);
  ax(1:2,:)=[];
  vx([1 nf+2],:)=[];
  if any(w=='d')
     c=[c vx ax];
  else
     c=[c ax];
  end
elseif any(w=='d')
  vf=(4:-1:-4)/60;
  ww=ones(4,1);
  cx=[c(ww,:); c; c(nf*ww,:)];
  vx=reshape(filter(vf,1,cx(:)),nf+8,nc);
  vx(1:8,:)=[];
  c=[c vx];
end

if nargout<1
   [nf,nc]=size(c);
   t=((0:nf-1)*inc+(n-1)/2)/fs;
   ci=(1:nc)-any(w=='0')-any(w=='e');
   imh = imagesc(t,ci,c.');
   axis('xy');
   xlabel('Time (s)');
   ylabel('Mel-cepstrum coefficient');
    map = (0:63)'/63;
    colormap([map map map]);
    colorbar;
end
```

```
function [Dist,D,k,w]=dtw(t,r)
%Dynamic Time Warping Algorithm
%Dist is unnormalized distance between t and r
%D is the accumulated distance matrix
%k is the normalizing factor
%w is the optimal path
%t is the vector you are testing against
%r is the vector you are testing
[rows,N]=size(t);
[rows,M]=size(r);
for n=1:N
    for m=1:M
        d(n,m)=(t(n)-r(m))^2;
end
end

D=zeros(size(d));
D(1,1)=d(1,1);

for n=2:N
    D(n,1)=d(n,1)+D(n-1,1);
end
for m=2:M
    D(1,m)=d(1,m)+D(1,m-1);
end
for n=2:N
    for m=2:M
        D(n,m)=d(n,m)+min([D(n-1,m),D(n-1,m-1),D(n,m-
1)]);
    end
end

Dist=D(N,M);
n=N;
m=M;
k=1;
w=[];
w(1,:)=[N,M];
while ((n+m)~=2)
    if (n-1)==0
        m=m-1;
    elseif (m-1)==0
        n=n-1;
    else
      [values,number]=min([D(n-1,m),D(n,m-1),D(n-1,m-
1)]);
      switch number
      case 1
```

```
        n=n-1;
      case 2
        m=m-1;
      case 3
        n=n-1;
        m=m-1;
      end
  end
    k=k+1;
    w=cat(1,w,[n,m]);
end
```

```
function s1f=sesfiltre()
fs=16000;
duration=3;

s1=wavrecord(duration*fs,fs);
num=[-1.09041473512197e-06,-5.28181936798424e-
06,1.26555570153634e-05,1.66243870615639e-
05,3.80967700078374e-19,-4.96468473801319e-05,-
0.000116517088205569,0.000150009951137571,6.7166252190554
3e-05,0.000384647554951928,-
0.000689279783802523,3.10190143067951e-05,-
0.000679898824926323,0.00156033298828913,-
0.000206141676181893,0.000474797797630931,-
0.00190726952727015,0.000190072085399157,0.00031402882333
1747,0.000759783821947938,-1.10673941770952e-
18,0.000656999716411328,0.000549560963727619,0.0005566499
13091216,-0.00897545257841688,0.00355400804739326,-
0.00245869904788399,0.0299186222226689,-
0.0212696939878349,0.00161661697648135,-
0.0615679810021347,0.0612042054015895,0.0201023444391269,
0.0916577611264803,-0.167223442578380,-
0.229878051006195,0.562500086021632,-0.229878051006195,-
0.167223442578380,0.0916577611264803,0.0201023444391269,0
.0612042054015895,-
0.0615679810021347,0.00161661697648135,-
0.0212696939878349,0.0299186222226689,-
0.00245869904788399,0.00355400804739326,-
0.00897545257841688,0.000556649913091216,0.00054956096372
7619,0.000656999716411328,-1.10673941770952e-
18,0.000759783821947938,0.000314028823331747,0.0001900720
85399157,-0.00190726952727015,0.000474797797630931,-
0.000206141676181893,0.00156033298828913,-
```

```
0.000679898824926323,3.10190143067951e-05,-
0.000689279783802523,0.000384647554951928,6.7166252190554
3e-05,0.000150009951137571,-0.000116517088205569,-
4.96468473801319e-05,3.80967700078374e-
19,1.66243870615639e-05,1.26555570153634e-05,-
5.28181936798424e-06,-1.09041473512197e-06;];

s1f=filter(num,1,s1);
```

```
function varargout = kisitaniGUI(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@kisitaniGUI_OpeningFcn, ...
                   'gui_OutputFcn',
@kisitaniGUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(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
% End initialization code - DO NOT EDIT


% --- Executes just before kisitaniGUI is made visible.
function kisitaniGUI_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
% varargin   command line arguments to kisitaniGUI (see
VARARGIN)

% Choose default command line output for kisitaniGUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes kisitaniGUI wait for user response (see
UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the
command line.
function varargout = kisitaniGUI_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
fs=16000;
duration=3;
set(handles.text5,'String','Record for 3 secs... Press
any key to start record...');
pause;
ses1=wavrecord(duration*fs,fs);
[b,a]=butter(32,0.2,'high');
ses1f=filter(b,a,ses1);
ses1e=myVAD(ses1f);
ses1e=endpoint(ses1e);
ses11=melcepst(ses1e,16000);
guidata(hObject,handles)
figure,plot(ses1e)

set(handles.text5,'String','Sesiniz alýndý.');
guidata(hObject,handles)

globalLoad('ornek1.mat');
globalLoad('ornek2.mat');
globalLoad('ornek3.mat');
globalLoad('ornek4.mat');

distance(1)=myDTW(ses11,ornek11)
distance(2)=myDTW(ses11,ornek22)
```

```
distance(3)=myDTW(ses11,ornek33)
distance(4)=myDTW(ses11,ornek44)

min_dist=min(distance)

switch min_dist
    case distance(1)
        set(handles.text7,'String',isim1);
        seriport('1');
     case distance(2)
        set(handles.text7,'String',isim2);
        seriport('2');
     case distance(3)
        set(handles.text7,'String',isim3);
        seriport('3');
     case distance(4)
        set(handles.text7,'String',isim4);
        seriport('4');
end

clear all

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1
as text
%        str2double(get(hObject,'String')) returns
contents of edit1 as a double

% isim1=get(hObject,'String');
%
% handles.isim1 = isim1;
% guidata(hObject,handles)



% --- Executes during object creation, after setting all
properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
```

```
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2
as text
%        str2double(get(hObject,'String')) returns
contents of edit2 as a double


% --- Executes during object creation, after setting all
properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3
as text
%        str2double(get(hObject,'String')) returns
contents of edit3 as a double


% --- Executes during object creation, after setting all
properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4
as text
%        str2double(get(hObject,'String')) returns
contents of edit4 as a double


% --- Executes during object creation, after setting all
properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

fs=16000;
duration=3;
set(handles.text5,'String','Record for 3 secs... Press
any key to start record...');
pause;
ornek1=wavrecord(duration*fs,fs);
[b,a]=butter(32,0.2,'high');
ornek1f=filter(b,a,ornek1);
ornek1e=myVAD(ornek1f);
ornek1e=endpoint(ornek1e);
ornek11=melcepst(ornek1e,16000);
guidata(hObject,handles)

isim1=get(handles.edit1,'String');
set(handles.text5,'String',isim1)
guidata(hObject,handles)
figure,plot(ornek1e)

save ('ornek1.mat', 'ornek11','isim1','ornek1e');

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton4 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

fs=16000;
duration=3;
set(handles.text5,'String','Record for 3 secs... Press
any key to start record...');
pause;
ornek2=wavrecord(duration*fs,fs);
[b,a]=butter(32,0.2,'high');
ornek2f=filter(b,a,ornek2);
ornek2e=myVAD(ornek2f);
ornek2e=endpoint(ornek2e);
ornek22=melcepst(ornek2e,16000);
guidata(hObject,handles)

isim2=get(handles.edit2,'String');
set(handles.text5,'String',isim2)
guidata(hObject,handles)
figure,plot(ornek2e)

save ('ornek2.mat', 'ornek22','isim2','ornek2e');

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

fs=16000;
duration=3;
set(handles.text5,'String','Record for 3 secs... Press
any key to start record...');
pause;
ornek3=wavrecord(duration*fs,fs);
[b,a]=butter(32,0.2,'high');
ornek3f=filter(b,a,ornek3);
ornek3e=myVAD(ornek3f);
ornek3e=endpoint(ornek3e);
ornek33=melcepst(ornek3e,16000);
guidata(hObject,handles)

isim3=get(handles.edit3,'String');
```

```
set(handles.text5,'String',isim3)
guidata(hObject,handles)
figure,plot(ornek3e)

save ('ornek3.mat', 'ornek33','isim3','ornek3e');

% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

fs=16000;
duration=3;
set(handles.text5,'String','Record for 3 secs... Press
any key to start record...');
pause;
ornek4=wavrecord(duration*fs,fs);
[b,a]=butter(32,0.2,'high');
ornek4f=filter(b,a,ornek4);
ornek4e=myVAD(ornek4f);
ornek4e=endpoint(ornek4e);
ornek44=melcepst(ornek4e,16000);
guidata(hObject,handles)

isim4=get(handles.edit4,'String');
set(handles.text5,'String',isim4)
guidata(hObject,handles)
figure,plot(ornek4e)

save ('ornek4.mat', 'ornek44','isim4','ornek4e');


% --- Executes during object creation, after setting all
properties.
function text5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text5 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called
```

```
% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)


clear all
close all
close
```

Appendix B. Source codes of PIC C

```
#include <pic.h>

void hata_sil(void)

{

unsigned char d;

if(OERR) {

  TXEN=0;

  TXEN=1;

  CREN=0;

  CREN=1; }

if(FERR) {

  d=RCREG;

  TXEN=0;

  TXEN=1; }

}

USART_gonder(unsigned char c)

{

while(!TXIF) {

  hata_sil(); }

TXREG = c ;

}
```

```
unsigned char USART_al(void)
{
while(!RCIF) {
  hata_sil(); }
return RCREG;
}
main(void) {
unsigned char karakter;
SPBRG=25;
BRGH=1;
SYNC=0;
SPEN=1;
CREN=1;
TX9=0;
RX9=0;
TXEN=0;
TXEN=1;
TRISB=0x00;
PORTB=0x00;
for(;;)
{
  karakter=USART_al();
```

```
  switch(karakter) {
    case '0':  {PORTB=0x01; break;}
    case '1':  {PORTB=0x02; break;}
    case '2':  {PORTB=0x04; break;}
    case '3':  {PORTB=0x08; break;}
    default :  {PORTB=0x10; break;}
}

//karakter++;

  USART_gonder(karakter);
}
}
```