Biometric Caller Identification

By

CSUO Usman Saeed

GC Shan Ul Hassan

GC Abeer Ansari

FC Muhammad Ramadan

Submitted to the Faculty of Department of Electrical Engineering, Military College of Signals, National University of Sciences and Technology, Islamabad in partial fulfillment for the requirements of a B.E Degree in Telecom Engineering

May 2015

Abstract

This project is aimed at developing a voice based caller identification system, which will identify the caller from a GSM call on the smart phone using a GSM Module and the VR module interfaced with Arduino UNO.

The incoming call will be received by the GSM module interfaced with Arduino module. The cal is then transferred to the speaker. A mic is installed near the speaker which takes the voice speech as the input to the VR module. The input voice is recorded in the VR module. When the same caller will call again his voice signal is then compared with the previous call data and the result is shown as the positive result. If someone will call the system whose voice is not saved in the data base, the system will tell as an unidentified user.

This project will be useful in identifying the prank calls. Further improvements in the future can be made for accessing individual's bio data with NADRA cooperation. Bank transactions can be made more secure with the help of voice identification. Another advantage of this project is that spontaneous conversation systems which are able to both recognize the spoken material accurately and understand the meaning of the spoken material. Such systems, which are currently beyond the limits of the existing technology, will enable new services such as 'Conversation Summarization', and ultimately even language translation services between any pair of existing languages. Moreover a new generation of voice-enabled service offerings emerging including intelligent agents, customer care wizards, and Call Centre automated attendants, unconstrained dictation capability, and finally unconstrained language translation capability can be employed using this project.

It is hereby certified that the contents and form of the project report entitled “Biometric Caller Identification”, submitted by the syndicate of

1. CSUO Usman Saeed
2. GC Shan Ul Hassan
3. GC Abeer Ansari
4. FC Muhammad Ramadan

have been found satisfactory as per the requirement of the B.E. Degree in Electrical (Telecom) Engineering.

Supervisor:

Lt Col Dr. Tayyab ALi

MCS, NUST

____

DECLARATION

We hereby declare that no content of work presented in this thesis has been submitted in support of another award of qualification or degree either in this institution or anywhere else.

ACKNOWLEDGEMENT

Nothing happens without the will of Allah Almighty. We thank Allah Almighty for giving us knowledge and strength to accomplish this task successfully.

We would like to thank our project supervisor, Lt Col Dr. Tayyab Ali, without his support and encouragement; it would not have been possible to complete this project.

We would also like to thank and our colleagues for helping in developing the project and people who have willingly helped us with their abilities.

Last but not least, we are very thankful to our parents, who bore with us in times of difficulty and hardship. Without their consistent support and encouragement we could not have accomplished our targets successfully.

DEDICATED TO

Almighty Allah,

Faculty for their help

And our parents for their support

Table of Contents

Contents

List of Abbreviations: 14

List of Figures 15

1. Chapter 1 16

1.1 Introduction: 16

1.1.1 Background 16

1.1.2 Problem Statement 16

1.2 Project Description 16

1.3 Prospective Application Area 16

1.4 Scope, Objectives, Deliverables and Specifications: 17

1.4.1 Scope and Objective: 17

1.4.2 Deliverables: 18

1.4.3 Specifications: 18

1.4.3.1 Arduino Kits 18

1.4.3.2 GSM kit 19

1.4.3.3 Voice Recognition Module 20

1.4.3.4 Speakers 21

1.4.3.5 Mic 22

1.4.3.6 LCDs 22

1.4.3.7 Amplifier 23

2. Chapter 2 24

2.1 Literature Review: 24

2.1.1 Overview of existing literature: 24

2.2 Problem Formulation: 25

2.3 Background Study: 26

3. Chapter 3 28

3.1 Detailed Design: 28

3.1.1 Text dependent Speaker Identification System 28

3.1.2 Text-Independent Speaker Identification System: 28

3.2 Design 29

3.3 Algorithm 31

3.3.1 Signal Pre – Processing: 31

3.3.2 Truncation: 31

3.3.3 Frame blocking: 32

3.3.4 Windowing: 32

3.3.5 Feature Extraction: 32

3.3.6 Cepstral Coefficients: 32

3.3.7 Mel-Frequency Wrapping: 33

3.3.8 Modelling: 33

3.3.9 Vector Quantization: 34

4. Chapter 4 35

4.1 Project Simulation and Results 35

4.1.1 Simulation 35

1. Chapter 5 39

5.1 Recommendations for future work 39

5.1.1 Lack of Portability: 39

5.1.2 Low Sensitivity: 39

5.1.3 Simple words instead of sentences: 40

5.1.4 Time Duration: 40

5.1.5 GSM Coverage necessary: 40

5.1.6 Only for Smart Phones: 40

5.1.7 Environment: 40

6. Chapter 6 41

6.1 Conclusion 41

7. Chapter 7 42

7.1 Demonstration Outline 42

8. References 44

9. Appendix A 45

10. Appendix B 47

11. Appendix C 51

12. Appendix D 113

List of Abbreviations:

VR MODULE	VOICE RECOGNITION MODULE
GSM KIT	GLOBAL SYSTEM FOR MOBILE COMMUNICATION MODULE
GMM	GAUSSIAN MIXTURE MODEL
PIN	PERSONAL IDENTIFICATION NUMBER
NADRA	NATIONAL DATABASE AND REGISTRATION AUTHORITY
LCDs	LIQUID CRYSTAL DISPLAY
HI-FI	HIGH FIEDALITY
NUST	NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
DSP	DIGITAL SIGNAL PROCESSING
IEEE	INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS
MFCC	MEL – FREQUENCY CEPTRUM COEFFICIENT
FSK	FREQUENCY SHIFT KEYING
DCT	DISCRETE COSINE TRANSFORM
DTW	DYNAMIC TIME DISTORTION

List of Figures

Figure 1.1: ……………………………………………………………………….......……………….. 8

Figure 1.2: …………………………………………………………….. 9

Figure 1.3: …………………………………………………………………………………….. 10

Figure 1.4: ………………………………………………………………………….……. 11

Figure 1.5: ………………………………………………………………………….…………. 12

Figure 1.6: …………………………………………………………………………………….. 12

Figure 2.1: …………………………………………………………………………………….. 14

Figure 2.2: ……………………………………………….. 14

Figure 3.1: …………………………………………………………………………………….. 19

Figure 3.2: …………………………………………………………………………………….. 19

Figure 3.3: …………………………………………………………………………………….. 20

Figure 3.4: …………………...………………………………………………………………… 21

Figure 4.1: …………………………………………………………………………………….. 25

Figure 4.2: …………………………………………………………………………………….. 25

Figure 4.3: ……………………………………...…... 26

Figure 4.4: …………………………………………………………………………………….. 26

Figure 4.5: …………………………………………………………………………………….. 27

Figure 4.6: …………………………………………………………………………………….. 27

Figure 4.7: ……………………………………………………………... 28

Figure 4.8: …………………………………………………………………………………….. 28

Figure 7.1: …………………………………………………………………………………….. 32

Figure 7.2: …………………………………………………………………………………….. 33

Figure 7.3: …………………………………………………………………………………….. 33

Figure 7.4: …………………………………………………………………………………….. 33

Chapter 1

1.1 Introduction:

1.1.1 Background

Speaker Identification is the process of automatically recognizing who is speaking on the basis of individual information included in speech waves. This technique makes it possible to use the speaker's voice to verify their identity and control access to services such as caller identification, voice dialing, banking by telephone, telephone shopping, database access services, information services, voice mail, security control for confidential information areas and remote access to computers.

1.1.2 Problem Statement

"Preliminary Design and Prototype Development of Biometric Caller Identification of callers for the user using the system."

1.2 Project Description

Previous implementations of this project were on the basis of voice recognition on the microcontrollers. We have implemented the project on Arduino kits. One Arduino kit is attached with the GSM kit. The GSM kit acts as the mobile phone in which we have to identify the callers. As any caller calls to the sim inserted in the GSM kit, the Arduino kit enables us to attend the call. A speaker is attached with the Arduino kit to amplify the voice of the caller. The other Arduino is attached with the Voice Recognition Module v3. The VR module records the incoming call, tests with the sample voice with its previous database and on that is provides the results. The results are displayed on the screen attached with the Arduino kit.

1.3 Prospective Application Area

1. Users have to speak a PIN (Personal Identification Number) in order to gain access to the laboratory door.
2. Users have to speak their credit card number over the telephone line to verify their identity.
3. The project can be used wherever the Biometric Identification of person is required.
4. The presence of an individual is not necessary, only the voice of a person from the mobile can work.
5. On the National level, with the cooperation of NADRA this project can be used to identify the individuals.

1.4 Scope, Objectives, Deliverables and Specifications:

1.4.1 Scope and Objective:

The basic scope of the project is to achieve the following objectives:

1. Understanding and designing of the caller identification
2. Simulation of various blocks in different software
3. Implementation with help of arduino module
4. Interfacing different blocks efficiently
5. Text-Independent Speaker Identification

Following are the goals of the project:

1. Maximizing the security level
2. Using human voice as a biometric tool
3. Eliminating the presence of human for the identification at a curtain spot
4. Innovative product
5. Power efficient
6. Efficient identification under different environmental conditions

1.4.2 Deliverables:

1. Arduino Kits
2. GSM kit
3. Voice Recognition Module
4. Speakers
5. Mic
6. LCDs
7. Amplifier

1.4.3 Specifications:

1.4.3.1 Arduino Kits:

We are using two Arduino kits for this project. The first Arduino kit is interfaced with the GSM kit to attend and listen to the caller. It also amplifies the voice of the caller via speakers provided on its output pins. It also displays the number of the caller in the first LCD interfaced with it. The job of second Arduino is to manage the VR module. The result from the VR module is provided to the second Arduino which displays the result on the second LCD.

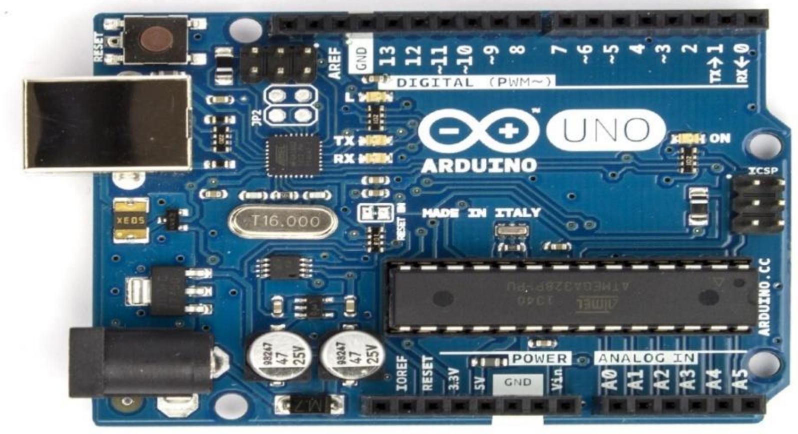


Figure 1.1: The Arduino Uno top view

1.4.3.1.1 Technical Specifications:

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 Ma
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by boot loader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25

1.4.3.2 GSM kit:

As it has been mentioned earlier, we are not using a proper mobile handset for our project. Instead we are using GSM kit compatible with the Arduino kit. The GSM kit provides the same purpose of mobile phone. The GSM is interfaced with the Arduino kit and a sim card is inserted in the GSM kit. When we call the number from our mobile phone, the GSM kit gives the signal to Arduino kit, and hence we can attend the call.

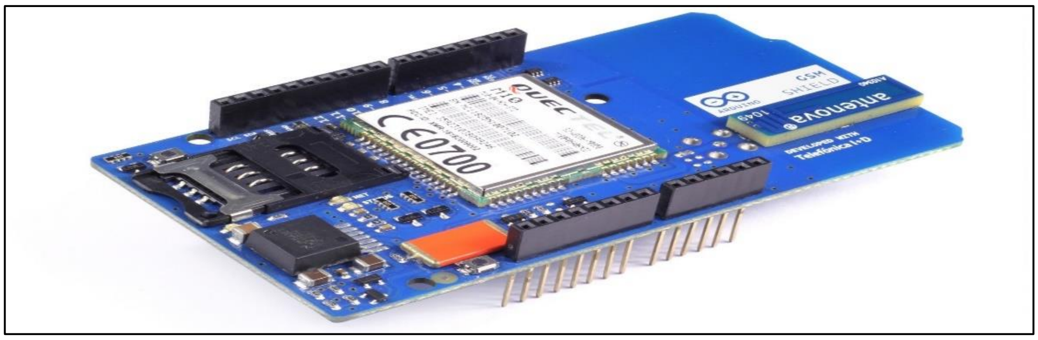


Figure 1.2: The GSM kit top view

1.4.3.2.1 Technical Specifications:

1. Requires an Arduino kit
2. Operating voltage 5v
3. Uses radio modem M10 by Quectel

1.4.3.3 Voice Recognition Module:

The voice recognition module is used to identify the voice of the caller. In this module we have used the Text Independent algorithms to serve our purpose better. First we have to train our VR module to save the voices as the database. Once the voices are saved in the VR module, it can be used as the tool to identify the voices. The VR can have 80 commands in its storage which is the only limitation in our project. The purpose of using this module is that it's the best available module specifically designed for the voice identification. If we implement the same algorithms and techniques on any other module we may not get as good results as we are getting here. A mic is also attached with the VR module to provide it the necessary voice signals as the input.

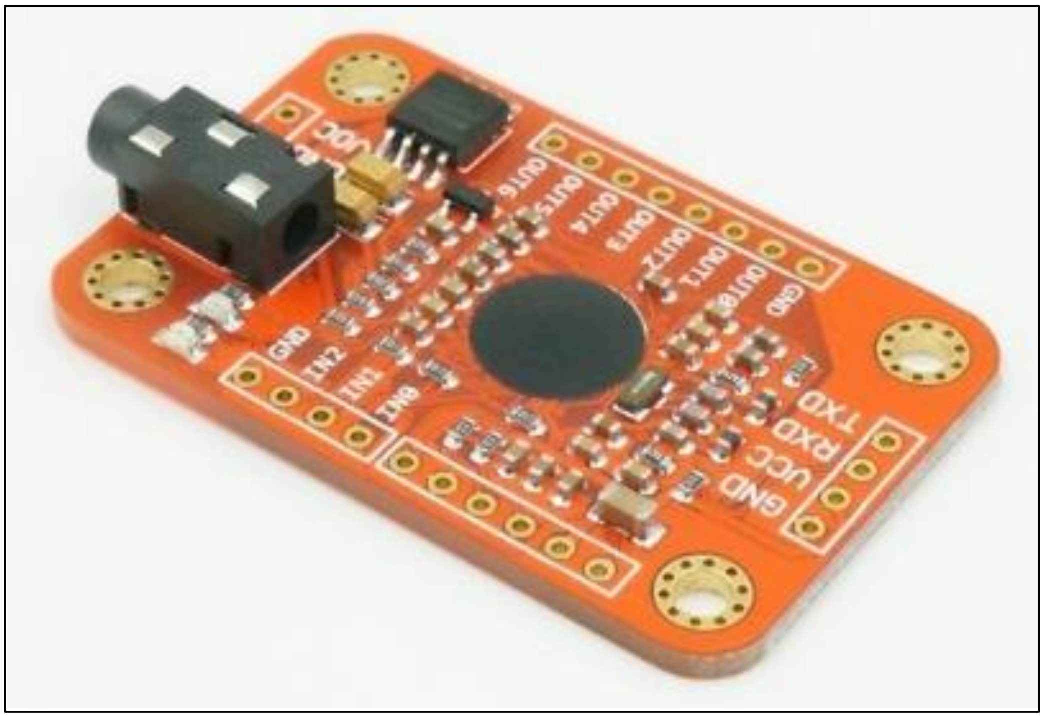


Figure 1.3: The Voice Recognition Module top view

1.4.3.3.1 Technical Specifications:

1. Requires an Arduino kit
2. Voltage: 4.5-5.5V
3. Current: <40mA
4. Analog Interface: 3.5mm mono-channel microphone connector + microphone pin interface
5. Size: 31mm x 50mm
6. Recognition accuracy: 99% (under ideal environment)

1.4.3.4 Speakers:

We are using speakers to listen the voice of the caller via GSM kit. The output of the first Arduino kit gives signals to the amplifier and the amplified signal comes to the speaker. The speaker also serves the purpose of giving input to the mic attached with the VR Module.

Figure 1.4: The Speaker

1.4.3.5 Mic:

A mic is attached with the VR module to give the input to VR. It takes the input for itself from the speaker placed in the system near it to provide a hi-fi sound to the VR module.

Figure 1.5: The Mic

1.4.3.6 LCDs:

Two LCDs are used for the project. One shows the incoming call status. It shows the mobile phone number with which a user is calling. This LCD is attached with the first Arduino kit. The second LCD shows the name of the caller once it is verified. This LCD is attached with the other Arduino kit.

Figure 1.6: The LCD

1.4.3.7 Amplifier:

The main purpose of amplifier is to amplify the voice signals before transmitting it to the speaker. The amplifier is also attached with the first Arduino kit and get a seperate power supply from the source.

Chapter 2

2.1 Literature Review:

2.1.1 Overview of existing literature:

To identify the caller biometrically, there are two main phases, the Training phase and the Testing phase. The training phase include the training of the system. To train the system, first we have to call and record our voice as the screen shows the message 'Talking'. The speaker is supposed to speaker for some time. Once the voice is processed and recorded successfully stored in the system, we can proceed for the other phase of the system i.e. the testing phase.

In the testing phase we can test and identify the same person. When the same person call the voice is processed and is compared with the data base already stored in the system. If the same speaker call, whose voice has already been stored in the system, the result is shown positive else negative.

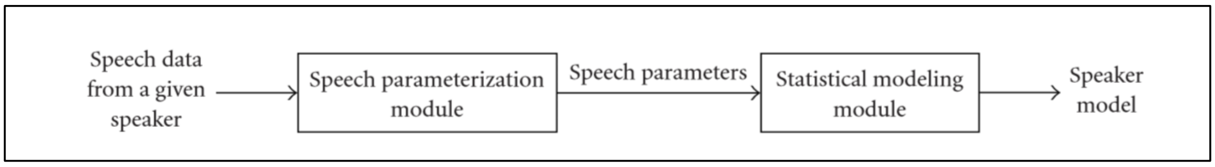

Speech data from a given speaker
Speech parameterization module
Speech parameters
Statistical modeling module
Speaker model

Figure 2.1: The testing phase of the project

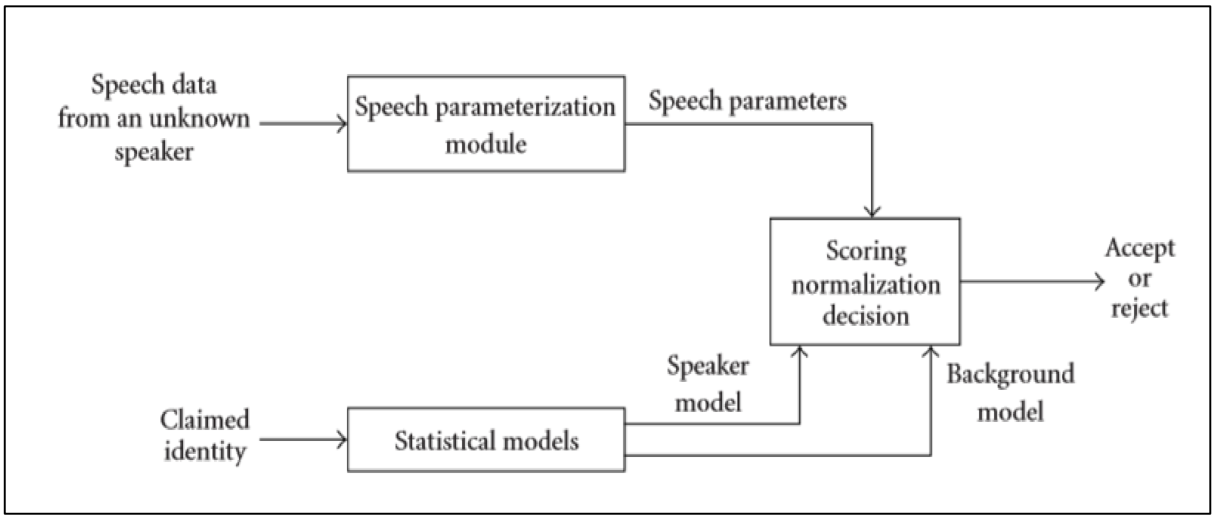


Figure 2.2: The Testing Phase of the project

There are different methods to identify the caller. The two main basic principles are:

1. Text – Dependent Identification system
2. Text – Independent Identification system

As we are using Text – Independent identification system. There are many algorithm and techniques available for the systems implementation. The basic components which are used for the required purpose are:

1. Speech Parameterization
2. Statistical Modeling
3. Normalization
4. Evaluation
5. Extensions of speaker verification

There are many techniques which can be used for the above mentioned processes. The main parts are Speech Parameterization, Statistical Modeling and Normalization because they are used in both the phases of the system.

2.2 Problem Formulation:

Now a days in times of eminent threat of terrorism, the need for rapidly and successfully identifying the caller is very important. Therefore alongside conventional systems of thumb impression, eye retina detection, using human voice as biometric tool for identifying the caller is also considered important. The use of this system will ensure rapid response and make the verification of the caller more efficient. Such a speaker identification system has potential in many security applications.

We can see that most of the existing implementations make use of the Independent Component Analysis techniques. However we are using the project mainly for smart phone users. As the voice recording is sent to the VR module, using different algorithms and methods the recording is verified and the result is sent to Arduino kit. The result is displayed on the screen. The GSM kit over here is used as a mobile phone. It is interfaced with the Arduino kit which enables the GSM kit to attend the call and keep the record of calls. The Arduino kit also displays the caller number on the screen. It also amplifies the voice of the caller using the speaker. The voice from the speaker is received by the mic which transmits the voice signal to Voice Recording Module. The VR module receives the voice as sample and tests it with the previous data stored in it. The VR module compares the characteristics of the voice and if the voice matches with any of its previous data it transfers the result to the second Arduino Kit. The second Arduino kit here is responsible for managing the VR module as well as the LCD display of the result. When the voice is matched the caller identity is displayed on the LCD. The caller name is shown on the LCD which confirms the identity of the caller.

2.3 Background Study:

This project was not done in Military College of Signals, NUST before. However a few projects on the same were done earlier with entirely different techniques. One project of NED Karachi was done but that was all software based. No hardware implementation was done before. The voices are stored on the device where the software is installed and the text dependent mechanism is adapted to trace the caller. However we found ample amount of research papers on this topics from the other foreign universities.

In Chinese University of Hong Kong, some work has been done, submitted by YUAN MENG for the degree of Master of Philosophy in Electronic Engineering. It describes the exploitation of state of art automatic speech recognition for DSP based embedded applications. Implementation of speech recognition algorithms generally involves a lot of floating point arithmetic operations and manipulation of complex functions which are not desirable for low cost DSP hardware which is one reason for its limitation.

IEEE research paper was published under title of 'Suppression of acoustic noise in speech using spectral subtraction'. This paper presents some advanced facilities and services related to the use of everyday telephones. It is based on potential use of 8- bit microcontroller. The decoding circuit has been designed for decoding the FSK modem tone with the limitation that it uses a separate call data circuit. In Nirma University, Ahmedabad India a project has been done using Matlab. In this project UG students have used the MFCC and Vector Quantization Model for the speaker recognition.

The MFCC and Vector Quantization techniques are the most latest and the most efficient techniques used for the caller identification. MFCC technique is used for the Feature Extraction of the speech signal. And Vector Quantization is used for the normalization of the speech signal. It is one the challenging step towards the topic due to the multi-dimensional integration.

Chapter 3

3.1 Detailed Design:

The project consists of different modules that must be implemented individually and then these modules must be interfaced together in order to achieve a complete and integrated unit that identifies the caller.

The project design can be achieved via two different ways.

1. Text – Dependent Identification system
2. Text – Independent Identification system

3.1.1 Text dependent Speaker Identification System:

Text-dependent speaker recognition seeks to associate an unknown speaker with a member from a registered population, given a textual transcription of the phrases uttered by the speaker. Typically, speaker-dependent word or sub-word models are built for each speaker. Given a labeled utterance from an unknown speaker, the system makes its speakers recognition decision based on the likelihood scores of the appropriate speaker-dependent models.

3.1.2 Text-Independent Speaker Identification System:

Its task is to identify the person who speaks regardless of what is saying. At the highest level, all speaker recognition systems contain two main modules feature extraction and feature matching. ‘Feature extraction’ is the process that extracts a small amount of data from the voice signal that can later be used to represent each speaker. ‘Feature matching’ involves the actual procedure to identify the unknown speaker by comparing extracted features from his/her voice input with the ones from a set of known speakers.

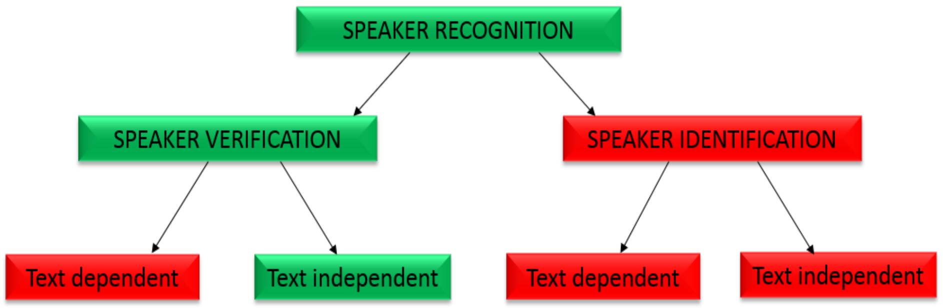


Figure 3.1: The Basic Techniques of Caller Identification System

3.2 Design

We approached the project by first designing the project in the Matlab software. A simulation of this project was implemented and was working successfully. The logics and algorithms we used in this simulations were further used in the hardware implementations.

The basic design of our project is shown in the figure below:

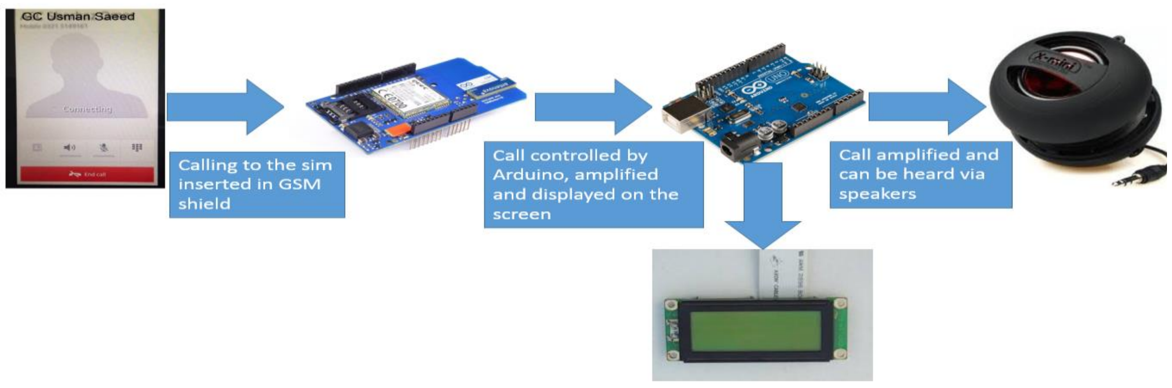

GC Usman Saeed
Calling to the sim inserted in GSM shield
Call controlled by Arduino, amplified and displayed on the screen
Call amplified and can be heard via speakers

Figure 3.2: Block Diagram of the first part of the project. Received call via GSM kit interfaced with Arduino kit, the call info displayed on LCD and the voice of caller amplified on the speaker

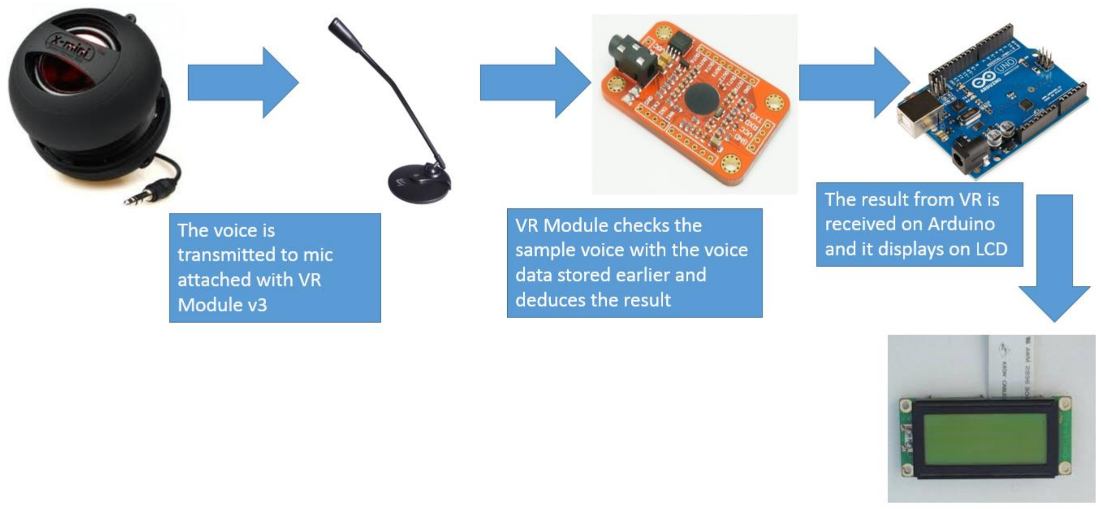

The voice is transmitted to mic attached with VR Module v3
VR Module checks the sample voice with the voice data stored earlier and deduces the result
The result from VR is received on Arduino and it displays on LCD

Figure 3.3: Block Diagram of the second part of the project. Mic receives the voice from speaker and gives input to the VR module, interfaced with Arduino kit. The compared result is shown on LCD via same Arduino kit

In the above mentioned Figure 3.2, the first part of the process is shown. When the caller calls on the system, GSM kit receives the call. GSM kit is interfaced with Arduino kit. As the call comes in, the Arduino kit displays the caller name on the LCD. When we attend the call, the voice signal is amplified on the speaker attached to the same Arduino. In the figure 3.3, the mic receives the audio signals from the speaker. The mic in interfaced with the VR Module which basically processes the voice identification system. The VR module compares the sample voice with the previously present voice data. If the sample voice matches with any previous data it informs the Arduino with the result. The Arduino displays the result with the second LCD.

3.3 Algorithm

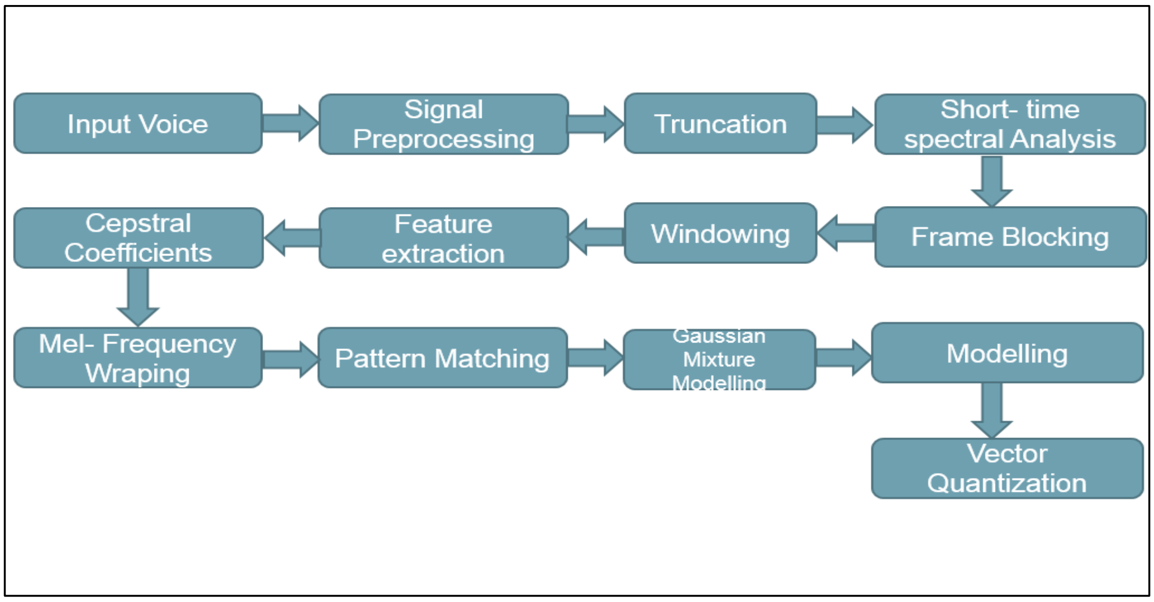


Figure 3.4: Basic algorithm of Voice Recognition System

In the above mentioned figure, the algorithm of our voice recognition system is explained. When the voice of caller is taken as an input to the VR module the many processes takes place. All the process above mentioned in the figure are discussed below.

3.3.1 Signal Pre – Processing:

Before extricating elements of different undertakings before the sign handling must be performed. Discourse flag needs to experience different sign molding strides before being subjected to highlight extraction systems.

3.3.2 Truncation:

Taking default recurrence tests wavread summon is 44100 Hz. At the point when recording a voice clasp, say for 2 seconds, the subsequent number of tests will be around 90,000 and that is a lot to handle. Thus we can flag truncated by determining a certain edge esteem.

3.3.3 Frame blocking:

In this stride is part into consistent discourse sign examples N edges, outlines with neighboring isolating them with tests M is not as much as that of the first edge N. worth comprises of the first N tests. The second casing starts M tests after the first edge, and the obstruction by N - M tests et cetera. This procedure proceeds until it is checking all the discussion of the utilization of one or more casings. We have picked the estimations of M and N to be N = 256 and M = 128 separately.

3.3.4 Windowing:

The following step is a window for every casing independently to lessen the sign intrusions toward the starting and end of every edge. The idea connected here is to lessen otherworldly bending by utilizing the window sign to decrease to zero toward the starting and end of every casing. On the off chance that we discover that the structure as w (n), $0 \leq n \leq N-1$, where N is the length of the edge, the outcome is a sign from the windows

$$\mathbf{y(n) = x(n)w(n), 0 \leq n \leq N-1.}$$

We have used the Hamming window in our project.

3.3.5 Feature Extraction:

Highlight extraction is the fundamental piece of the task. To focus the voice highlight extraction is utilized. Extraction is important to prepare and in addition the testing stage highlight.

3.3.6 Cepstral Coefficients:

Before sign preparing diminishes the computational unpredictability while taking a shot at discourse signal. We decrease the quantity of tests of the procedure. As opposed to taking a shot at an extensive variety of tests to bind our operations under the sufficiently low along. After the discourse sign molding after the preparade the following step is to concentrate the components of the preparation signal. We have profited from the 2 routes for himself. The primary strategy is to figure Cepstral coefficients of signs utilizing the DCT (discrete cosine change). Cepstral coefficients are figured utilizing the accompanying recipe:

CEPS = DCT (log (ABS (FFT (ywindowed)))

3.3.7 Mel-Frequency Wrapping:

Human view of the recurrence substance of the hints of discourse signs does not take after a straight scale. Per tone and with the genuine recurrence, the pitch is an individual measure on the scale and the purported "mi" scale. Gives a measure of the recurrence mile straight recurrence dispersing of under 1 kHz separating and logarithmic higher than 1 KHz. As a result of the recurrence size of a mile some time recently:

Fmel = (1000 / log (2)) * log (1+ and / 1000)

One way to deal with recreate the range is the utilization of an individual bank, which competitor separated consistently on a scale miles. Bank hopeful has a triangular band pass recurrence reaction. A persistent inclination is resolved separating and data transmission by time interim. We pick K, and the quantity of miles range exchanges to be 20. This channel bank being connected in the field of recurrence adds up to only a Windows application shape a pleasant triangle. .

3.3.8 Modelling:

Utilizing Cepstral and MFCC coefficients, talking segment can speak to an arrangement of highlight vectors. A man express the same word yet in an alternate time have a comparable still distinctive component vector succession. The motivation behind demonstrating voice lies in building a model that can catch these distinctions in an arrangement of extricated elements of a specific head. There are normally two sorts of models that are utilized on an expansive scale as a part of the distinguishing proof of amplifier frameworks:

1. Stochastic models
2. Template models

Stochastic model adventures exploit the likelihood hypothesis by managing discourse creation handle as a procedure of arbitrary outskirt. It expect that the parameters of the hidden stochastic procedure can be evaluated exactly, obvious way. Layout model (non-parametric technique) is attempting to produce a model for discourse generation procedure of a specific client in a non-parametric. It does as such by utilizing numerous vectors extricated from the discourse highlight of the same word grouping by the same individual. The format utilized models to control the work in the early acknowledgment of the speaker in light of the fact that it meets expectations with no suspicion about how they are shaping a component vector. Subsequently format model is naturally more sensible. Nonetheless, late research in stochastic models has uncovered her to be more adaptable, considering the era of the best models of the procedure to recognize the speaker. Cutting edge in highlight coordinating procedures used to recognize the speaker incorporates element time mutilation (DTW), Gaussian blend displaying (GMM), and vector quantization (VQ).

3.3.9 Vector Quantization:

Vector quantization (VQ short) includes taking a vast gathering of tankers procedure normal for a specific client and the creation highlights a littler gathering of highlight vectors that speak to centroids dispersion, any focuses separated in order to decrease the normal separation to each other point. Vector quantization is utilized in light of the fact that it would be excessively unreasonable, making it impossible to speak to every one element vectors in highlight space that we are conceived of discourse speaker preparing meeting. While the VQ calculation does not take eventually to produce centroids, it spares a great deal of time amid the test stage and we simply look a couple highlight vectors rather than limit space normal for a specific client characteristics. So is the monetary trade off we can live with him. A vector amounts vectors k-dimensional maps in space tankers "^{Rk}" to a constrained arrangement of vectors

$$Y = \{Yi: i = 1, 2, \dots N\}.$$

Here k measurement alludes to any component of all exchanges in the element vectors. Furthermore, it called all tankers Yi vector code or a term and an arrangement of all code words called codebook. Also, in this way a specific number of clients, is made books are accessible to every speaker amid the preparation utilizing the VQ stage. Per the expressions "Yi", and there closest neighbor related range called the locale Voronoi, it was characterized by the

$$V_i = \left\{ x \,\varepsilon\, R^k \; : || \, x - y_i \, || \leq || \, x - y_j \, ||, \text{ for all } j \neq i \right\}$$

Chapter 4

4.1 Project Simulation and Results

All the results and simulations related to the Caller identification have been demonstrated below.

4.1.1 Simulation

Basic Techniques implemented in Matlab Simulation Code

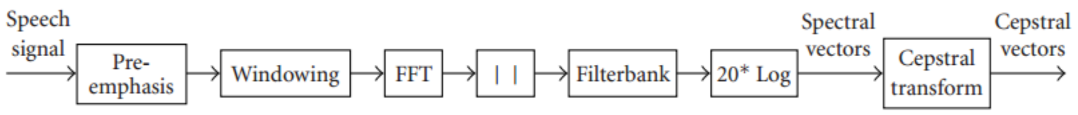


Figure 4.1: Basic Techniques in Matlab code

Main Interface of Simulation

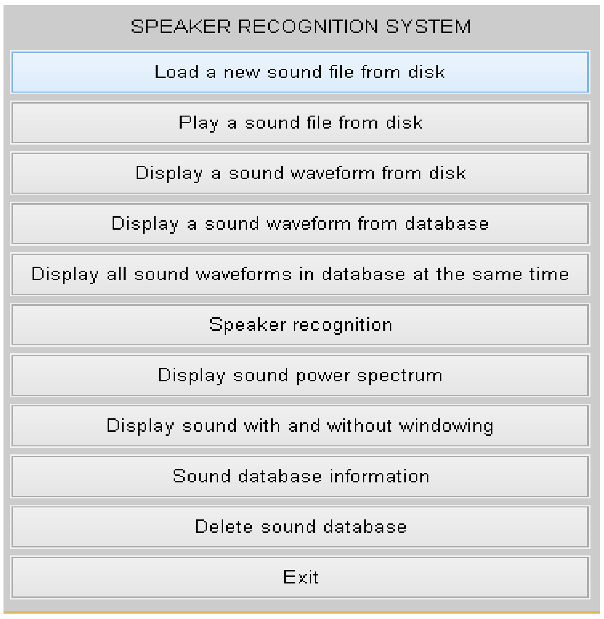
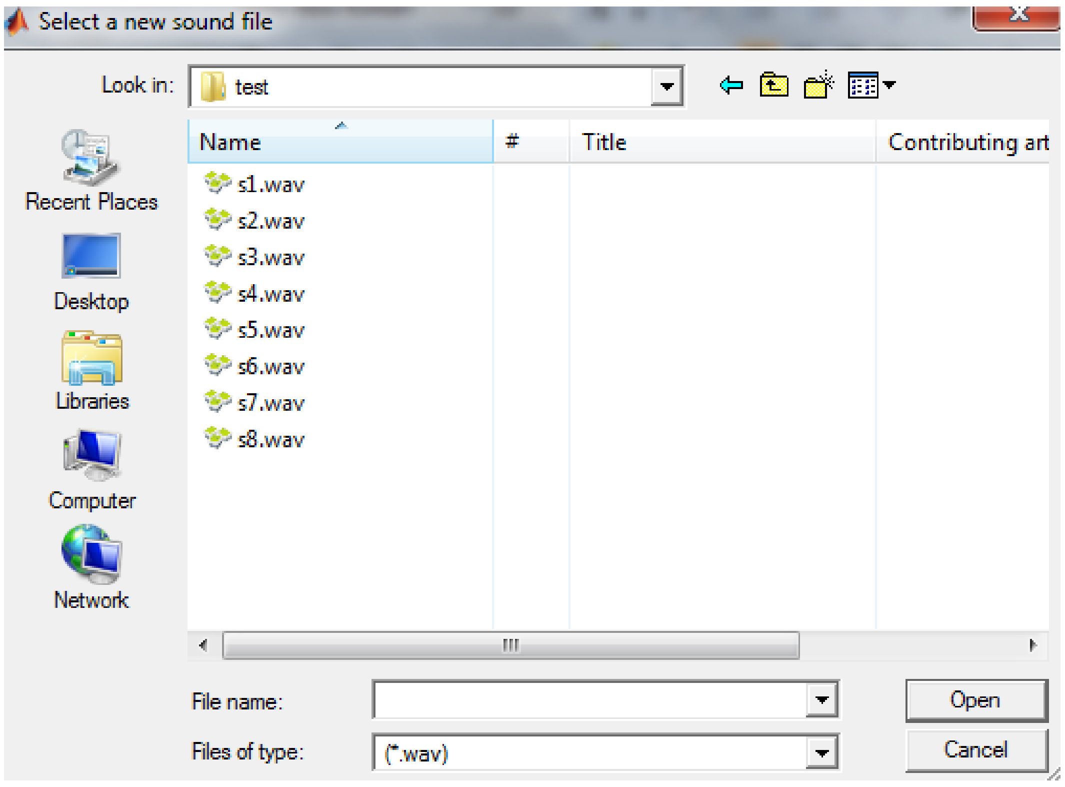

SPEAKER RECOGNITION SYSTEM
Load a new sound file from disk
Play a sound file from disk
Display a sound waveform from disk
Display a sound waveform from database
Display all sound waveforms in database at the same time
Speaker recognition
Display sound power spectrum
Display sound with and without windowing
Sound database information
Delete sound database
Exit

Figure 4.2: Main Interface of the Matlab Code

Sample Voices for the Testing and Training of the Simulation



s1.wav
s2.wav
s3.wav
s4.wav
s5.wav
s6.wav
s7.wav
s8.wav

Files of type: (*.wav)

Figure 4.3: Sample voices for the testing and training of the simulation

Image of Recorded Signal

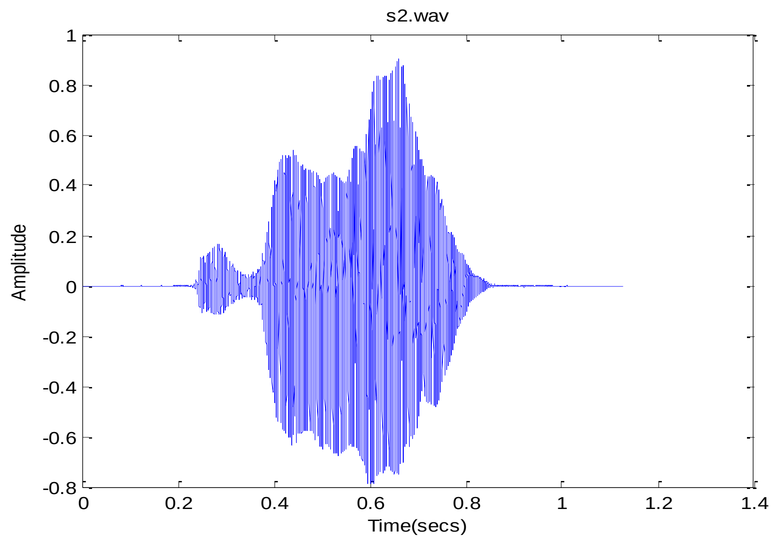


Figure 4.4: Recorded Signal on VR Module

Samples collected from the Recorded Signal

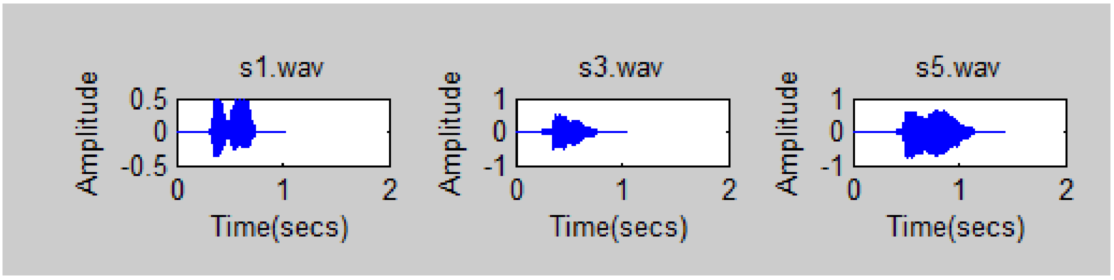
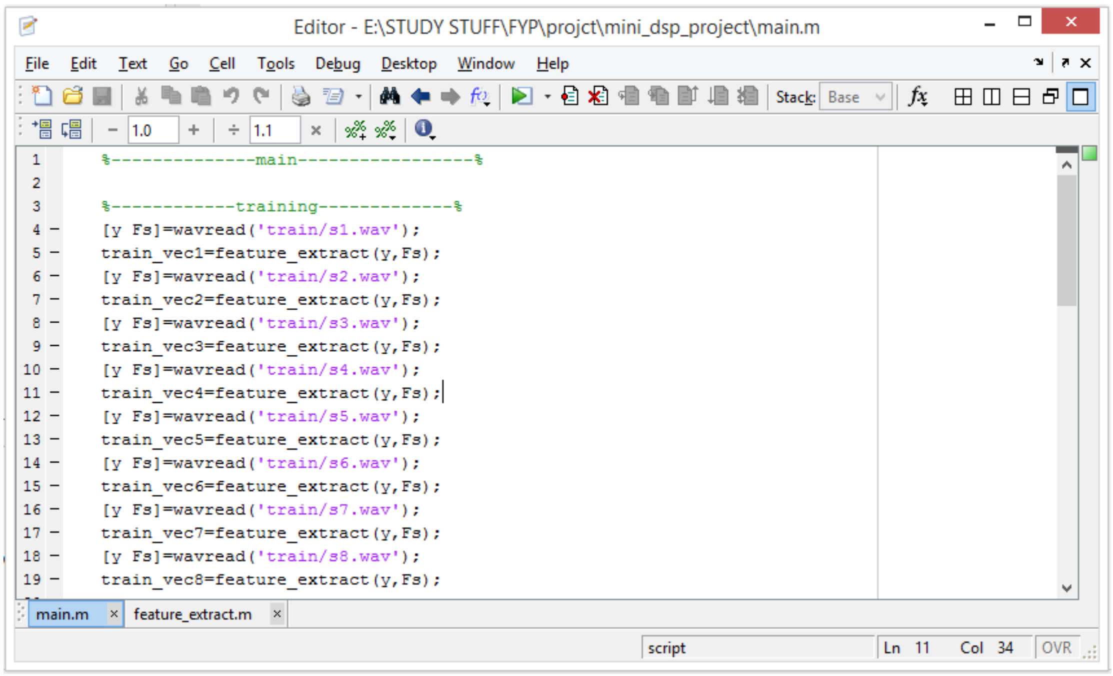


Figure 4.5: Sampled voices from the Recorded signals

.m File: Main file of Matlab Simulation Code

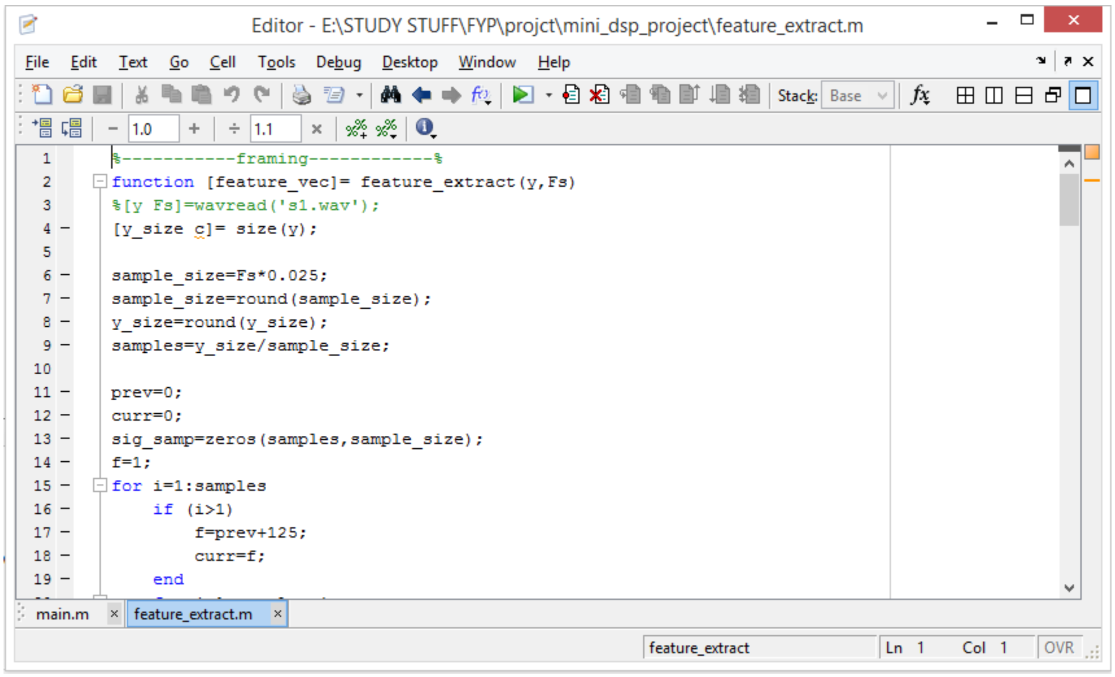


```
%--------------main----------------%

%------------training-------------%
[y Fs]=wavread('train/s1.wav');
train_vec1=feature_extract(y,Fs);
[y Fs]=wavread('train/s2.wav');
train_vec2=feature_extract(y,Fs);
[y Fs]=wavread('train/s3.wav');
train_vec3=feature_extract(y,Fs);
[y Fs]=wavread('train/s4.wav');
train_vec4=feature_extract(y,Fs);
[y Fs]=wavread('train/s5.wav');
train_vec5=feature_extract(y,Fs);
[y Fs]=wavread('train/s6.wav');
train_vec6=feature_extract(y,Fs);
[y Fs]=wavread('train/s7.wav');
train_vec7=feature_extract(y,Fs);
[y Fs]=wavread('train/s8.wav');
train_vec8=feature_extract(y,Fs);
```

Figure 4.6: .m File of Matlab Simulation Code

.m File: Feature Extraction file of Matlab Simulation Code:


Editor - E:\STUDY STUFF\FYP\projct\mini_dsp_project\feature_extract.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
main.m
feature_extract.m
feature_extract
Ln 1 Col 1 OVR

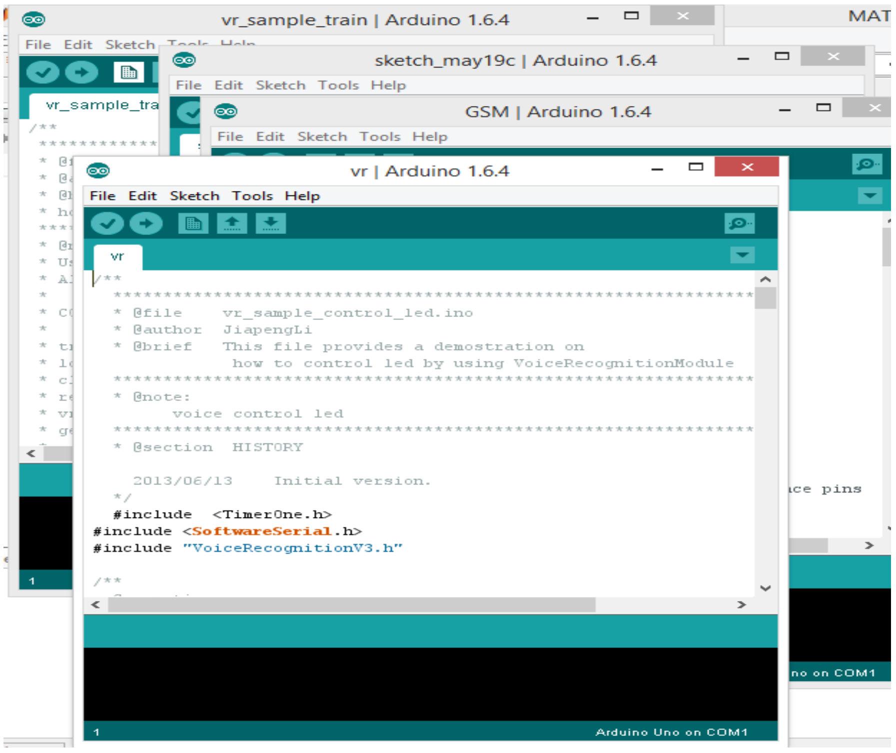
```
%-----------framing-----------%
function [feature_vec]= feature_extract(y,Fs)
%[y Fs]=wavread('s1.wav');
[y_size c]= size(y);

sample_size=Fs*0.025;
sample_size=round(sample_size);
y_size=round(y_size);
samples=y_size/sample_size;

prev=0;
curr=0;
sig_samp=zeros(samples,sample_size);
f=1;
for i=1:samples
    if (i>1)
        f=prev+125;
        curr=f;
    end
```

Figure 4.7: .m file of Feature Extraction in Matlab Simulation Code

Code of Arduino UNO on Arduino Software


vr_sample_train | Arduino 1.6.4
sketch_may19c | Arduino 1.6.4
GSM | Arduino 1.6.4
vr | Arduino 1.6.4
File Edit Sketch Tools Help
vr
Arduino Uno on COM1

```
/**
  ******************************************************************
  * @file    vr_sample_control_led.ino
  * @author  JiapengLi
  * @brief   This file provides a demostration on
              how to control led by using VoiceRecognitionModule
  ******************************************************************
  * @note:
       voice control led
  ******************************************************************
  * @section  HISTORY

    2013/06/13    Initial version.
  */
  #include  <TimerOne.h>
#include <SoftwareSerial.h>
#include "VoiceRecognitionV3.h"

/**
```

Figure 4.8: Code of Arduino UNO Hardware on Arduino software

Chapter 5

5.1 Recommendations for future work

This project has not been performed in Military College of Signals before. It was the first of its own kind. Therefore many difficulties were faced during the completion of this project. However in the near future this project can be further be improved. The main idea of this project was to develop a prototype of a biometric caller identification system, which can be carried by the user anywhere he wants. As told earlier, due to very rare practical development on this project, we were not able to make a complete portable system. The following points will show the limitations of the project and which can be further improved in the near future.

5.1.1 Lack of Portability:

The hardware implementation of this project does not allow the user for its free and handy use. For further improvements the following suggestions are:

1. Interfacing of the Arduino modules with the Bluetooth headphones; this will allow the user to identify the caller wherever he is using the phone.
2. Integration of the current model of the hardware with Zigbee module can help to build an Identification hub. This will allow user to use the system for multipurpose.
3. With the cooperation of NADRA; wherever the need of biometric identification is required, voice can be used and the presence of the human body is not required.

5.1.2 Low Sensitivity:

The VR module is very low sensitive in case of training phase. Once we are training the module, we have to make sure that the module must collect the same voice twice to store in its data base. We don’t have much liberty while training the module to change our voice in a slightest way. Similarly in case of testing we have to speak in a very similar manner so that the system may detects us. Further work and research can overcome this problem.

5.1.3 Simple words instead of sentences:

One of the basic limitation is our project is we cannot train or test the module in a simple way as we speak in our daily lives. We have faced this problem in the final stages of our project when all the hardware was interfaced. To record our voices, we have to speak simple words instead of a complete sentence.

5.1.4 Time Duration:

The time duration of recording is also very less. Most of the time one misses the recording window available and user does not get the required results.

5.1.5 GSM Coverage necessary:

GSM coverage is necessary for the project to run. As the primary function of our project is to identify the caller, we cannot use the system where the GSM coverage is not available. However with further modifications using the ZigBee module we can switch it to the internet as well.

5.1.6 Only for Smart Phones:

Currently our project is restricted to smart phones only. As the smart phones now contain the feature of noise cancelation we get better quality of voice from them.

5.1.7 Environment:

The environment and surroundings of the caller matters a lot in our project. Our project is based on human voice using a mobile phone. When a caller calls from a noisy place it get difficult for the VR module to identify the voice of the caller because of which we may get the required results.

Chapter 6

6.1 Conclusion

The biometric caller identification is a unique project as for the first time we have tried to use human voice as a biometric tool for identification. In the current scenario of research and development, big companies like Google, Apple and Microsoft are working on the topic of voice. However despite this project was a success as we were getting result accurate up to 90%. A few more improvements on the hardware implementation can improve its efficiency and can further be used on the commercial scale for the identification purposes.

Following objectives are being achieved:

1. Understanding and designing of the biometric caller identification system
2. Use of MFCC and VQ techniques in the voice identification
3. Development of an efficient caller identification system which can work under a little harsh environment.
4. The system can be used, in its current form, on a commercial scale. For that purpose memory of the VR module is needed to be enhanced.

The limitations of the project have been discussed in Chapter 5. These limitations can be overcome by using additional module with the Arduino kits. These redundancies are not permanent and the system is providing its core goal i.e. identification of the caller.

The objective of portability could be achieved this time. For that purpose, the fabrication of mobile headphones with the VR module is required and is another vast topic to research on.

Chapter 7

7.1 Demonstration Outline

All the result and simulation related to the Biometric Caller Identification have been demonstrated in the report. These are the basic outline of demonstration:

1. GSM kit with First Arduino kit
2. Speaker interfaced with First Arduino kit
3. First LCD interfaced with First Arduino kit
4. VR module interfaced with Second Arduino kit
5. Mic integrated with the VR module
6. Second LCD interfaced with Second Arduino kit
7. First LCD showing the status of system being ready
8. First LCD showing the status of call received and attended
9. Second LCD showing the identification of Caller
   9.1 If the caller is identified the name of the caller is displayed
   9.2 If the caller is unidentified, the LCS shows the status As 'UNID CALLER'

The final Hardware: Outside View

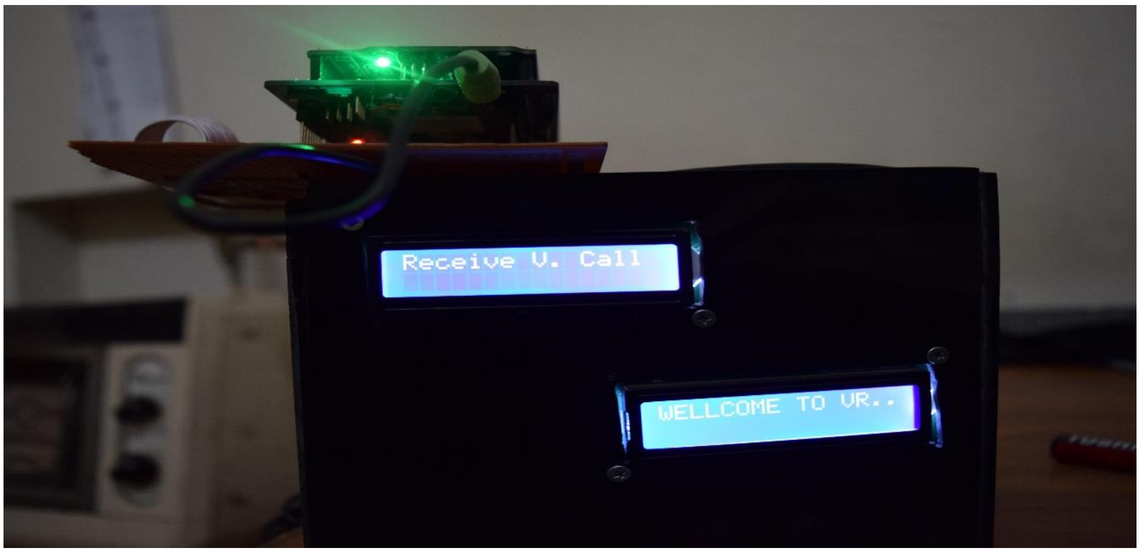


Figure 7.1: The Hardware of Biometric Caller Identification

The final Hardware: Inside View

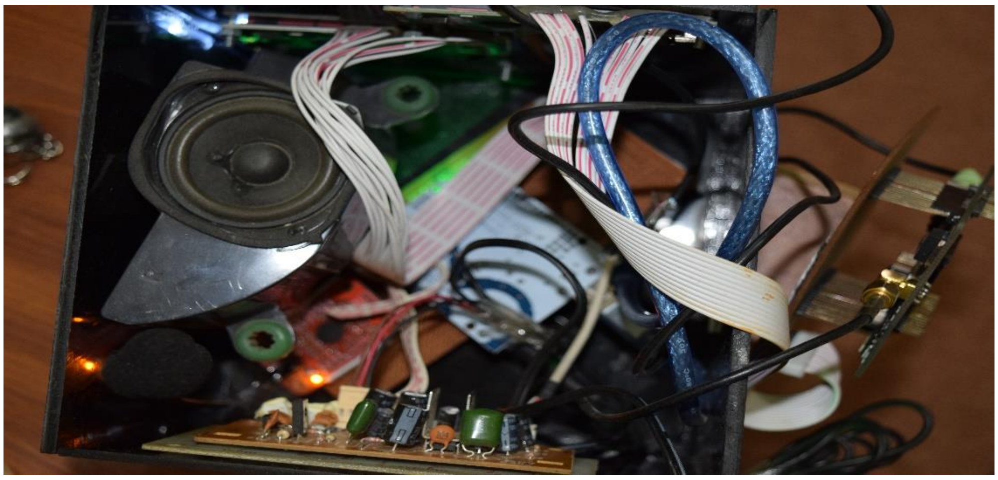


Figure 7.2: The Hardware, Inside View

When a known Caller Calls: Name Displayed

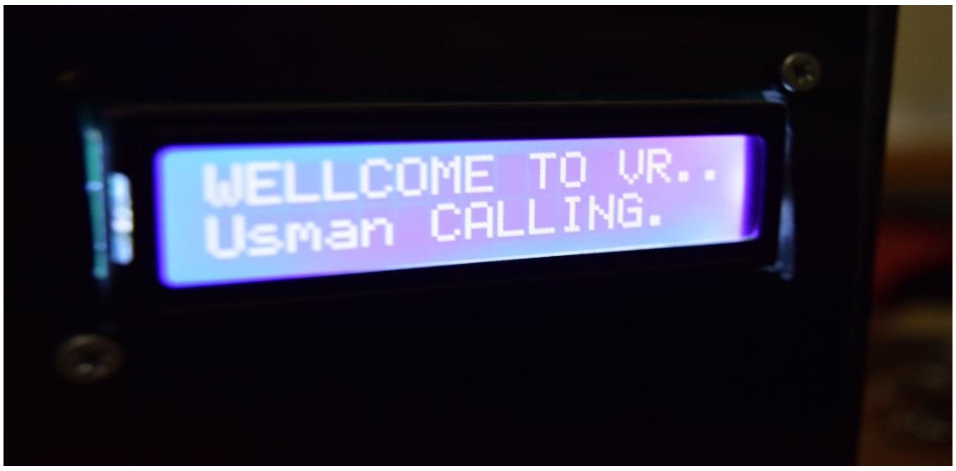


Figure 7.3: Name Display on the LCD for the known Caller

When an Unknown Caller Calls: UNID Displayed

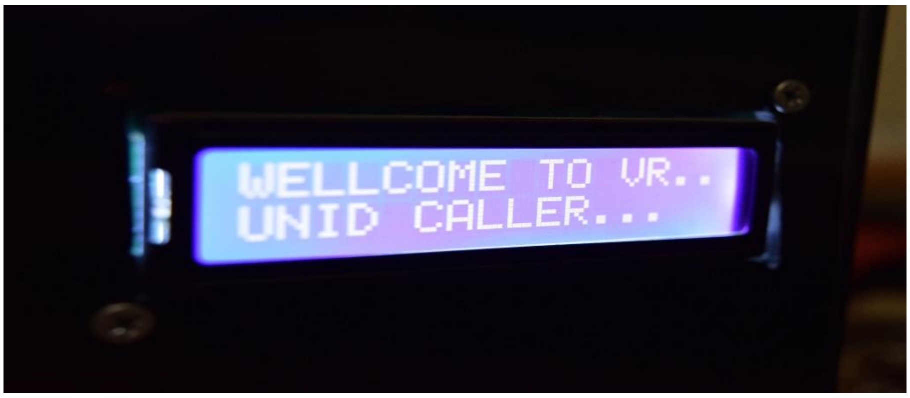


Figure 7.4: UNID displayed when the unknown Caller Calls

References

Many institutions and companies working on the satellite projects have done this type of work. Some references of these works are as follows:

[1] ‘Speech Recognition on DSP’ by YUAN Meng in Chinese University of Hong Kong as the Thesis for the Maters in Electronic Engineering in 2004.

[2] ‘Voice Recognition Software’ made in NED Karachi. The software was made by the undergraduate students of NED in 2011.

[3] ‘Suppression of acoustic noise in speech using spectral subtraction’ by Boll, S.F., 1979. IEEE Transactions on Acoustics, Speech, and Signal Processing ASSP 27 (2), 113–120.1

[4] ‘Speaker Recognition Using MFCC and Vector Quantization Model’ by Darshan Mandalia, Pravin Gareta for the Bachelor of Technology in Electronics & Communication Engineering in Nirma University, Ahmedabad

[5] ’ Decoding speech in the presence of other sources. Speech Communication’ by Barker, J.P., Cooke, M.P., Ellis, D.P.W., 2005.

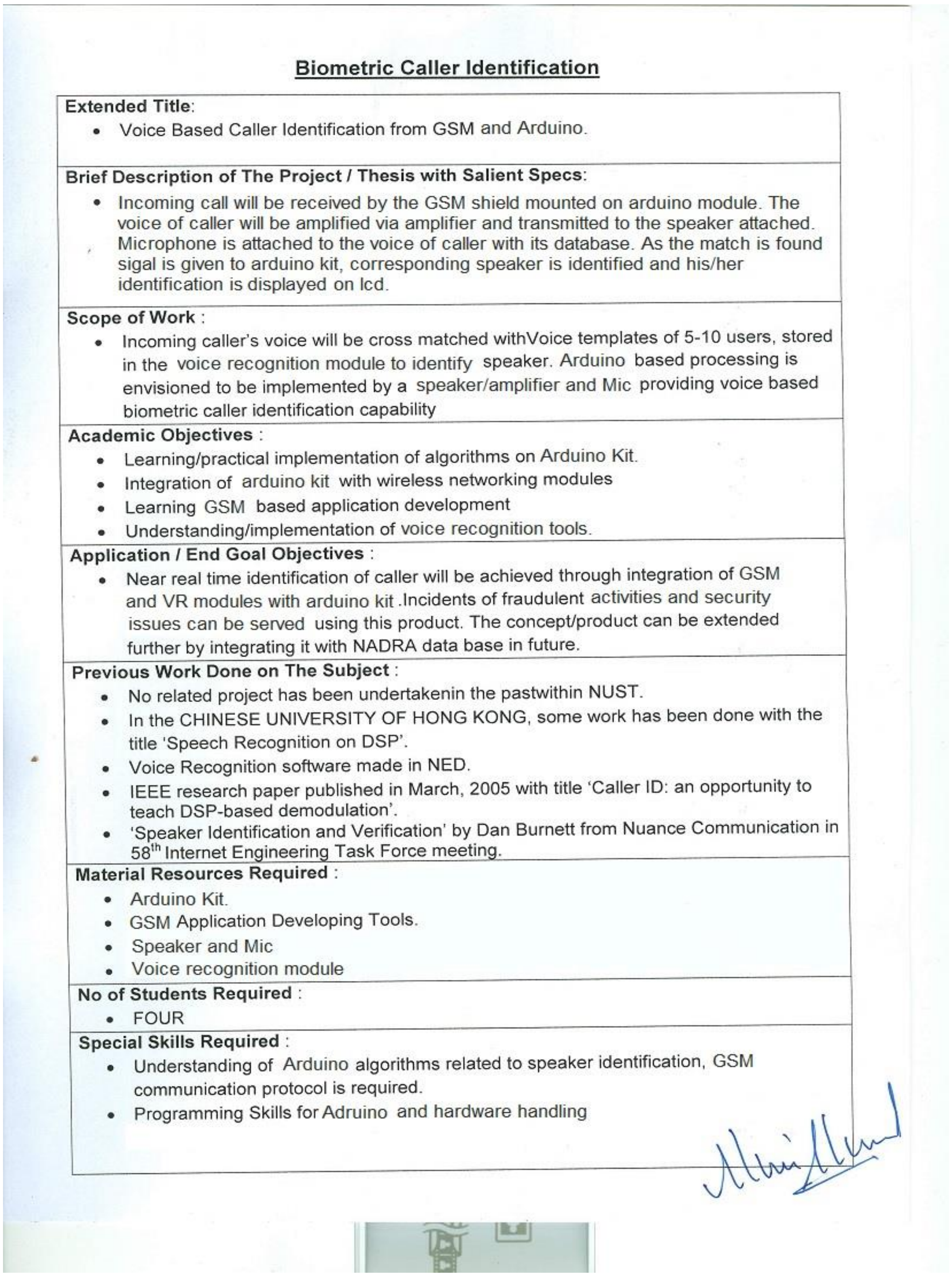
[6] ‘Auditory Scene Analysis’ by Bregman, A.S., 1990. Published by the MIT Press.

[7] ‘A neural oscillator sound separator for missing data speech recognition’ by Brown, G.J., Barker, J., Wang, D.L., 2001 was presented in the Proceedings of the International Joint Conference on Neural Networks ’01, pp. 2907–2912.

[8] ‘Robust automatic speech recognition with missing and unreliable acoustic data’ by Cooke, M.P., Green, P., Josifovski, L., Vizinho, A., 2001.

[9] ‘Speaker recognition using universal background model on YOHO database’ by Alexandre Majetniak, Aalborg University, 2011.

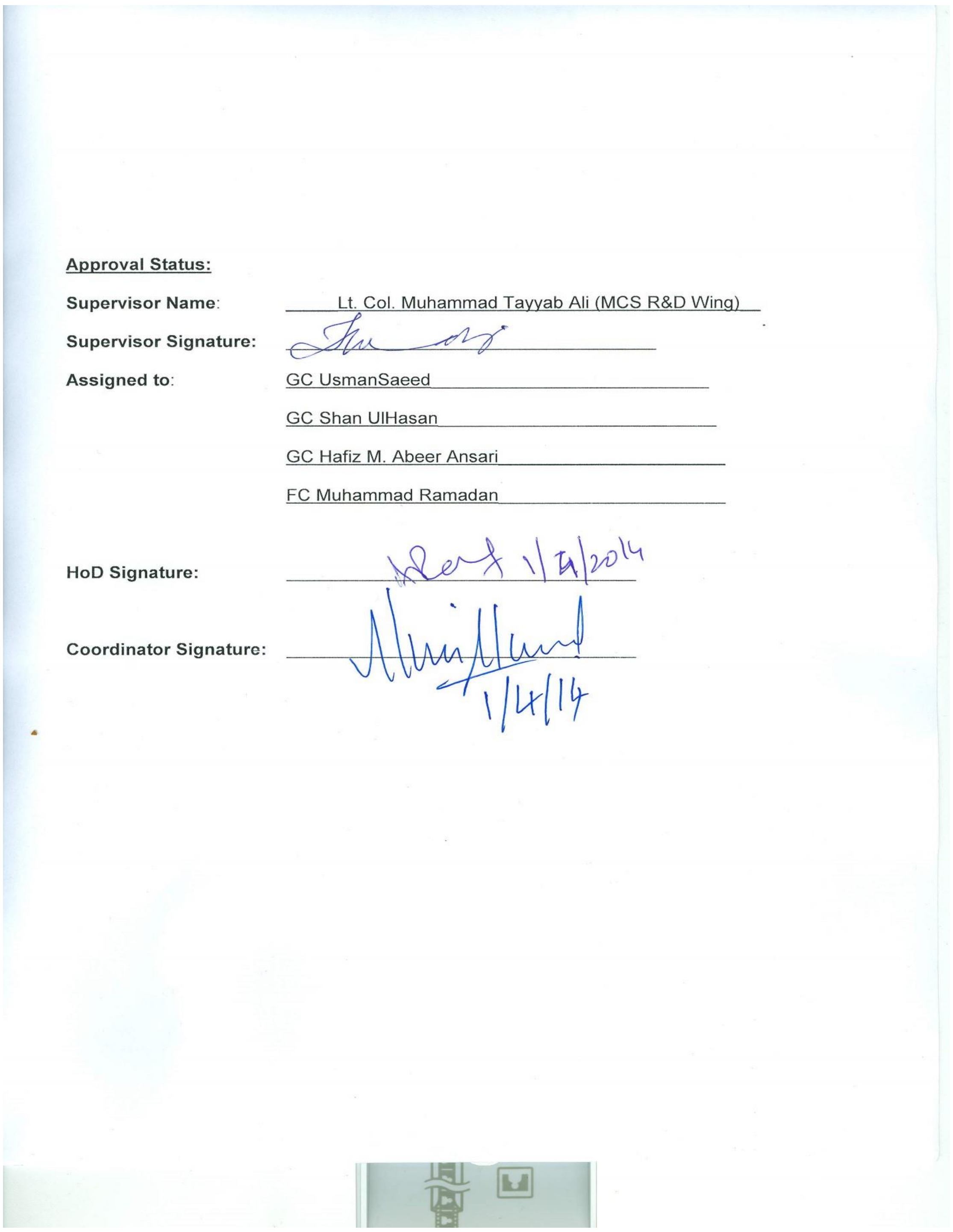
Appendix A



Biometric Caller Identification

Extended Title: • Voice Based Caller Identification from GSM and Arduino.
Brief Description of The Project / Thesis with Salient Specs: • Incoming call will be received by the GSM shield mounted on arduino module. The voice of caller will be amplified via amplifier and transmitted to the speaker attached. Microphone is attached to the voice of caller with its database. As the match is found sigal is given to arduino kit, corresponding speaker is identified and his/her identification is displayed on lcd.
Scope of Work : • Incoming caller's voice will be cross matched withVoice templates of 5-10 users, stored in the voice recognition module to identify speaker. Arduino based processing is envisioned to be implemented by a speaker/amplifier and Mic providing voice based biometric caller identification capability
Academic Objectives : • Learning/practical implementation of algorithms on Arduino Kit. • Integration of arduino kit with wireless networking modules • Learning GSM based application development • Understanding/implementation of voice recognition tools.
Application / End Goal Objectives : • Near real time identification of caller will be achieved through integration of GSM and VR modules with arduino kit .Incidents of fraudulent activities and security issues can be served using this product. The concept/product can be extended further by integrating it with NADRA data base in future.
Previous Work Done on The Subject : • No related project has been undertakenin the pastwithin NUST. • In the CHINESE UNIVERSITY OF HONG KONG, some work has been done with the title 'Speech Recognition on DSP'. • Voice Recognition software made in NED. • IEEE research paper published in March, 2005 with title 'Caller ID: an opportunity to teach DSP-based demodulation'. • 'Speaker Identification and Verification' by Dan Burnett from Nuance Communication in 58th Internet Engineering Task Force meeting.
Material Resources Required : • Arduino Kit. • GSM Application Developing Tools. • Speaker and Mic • Voice recognition module
No of Students Required : • FOUR
Special Skills Required : • Understanding of Arduino algorithms related to speaker identification, GSM communication protocol is required. • Programming Skills for Adruino and hardware handling

[illegible]



Approval Status:

Supervisor Name:	Lt. Col. Muhammad Tayyab Ali (MCS R&D Wing)
Supervisor Signature:	[illegible]
Assigned to:	GC UsmanSaeed
	GC Shan UlHasan
	GC Hafiz M. Abeer Ansari
	FC Muhammad Ramadan
HoD Signature:	[illegible] 1/4/2014
Coordinator Signature:	[illegible] 1/4/14

Appendix B

Code used for the project

Matlab Code for Simulation

```
%--------------main-----------------%

%------------training-------------%
[y Fs]=wavread('train/s1.wav');
train_vec1=feature_extract(y,Fs);
[y Fs]=wavread('train/s2.wav');
train_vec2=feature_extract(y,Fs);
[y Fs]=wavread('train/s3.wav');
train_vec3=feature_extract(y,Fs);
[y Fs]=wavread('train/s4.wav');
train_vec4=feature_extract(y,Fs);
[y Fs]=wavread('train/s5.wav');
train_vec5=feature_extract(y,Fs);
[y Fs]=wavread('train/s6.wav');
train_vec6=feature_extract(y,Fs);
[y Fs]=wavread('train/s7.wav');
train_vec7=feature_extract(y,Fs);
[y Fs]=wavread('train/s8.wav');
train_vec8=feature_extract(y,Fs);

%-----------------testing------------%
[y Fs]=wavread('test/s4.wav');
test_vec1=feature_extract(y,Fs);

%----------Euclidean distance------------%
[r c]=size(test_vec1);
EU=sqrt(sum(sum(test_vec1-train_vec8).^2));
EU1=sqrt(sum(sum(test_vec1-train_vec7).^2));
EU2=sqrt(sum(sum(test_vec1-train_vec6).^2));
EU3=sqrt(sum(sum(test_vec1-train_vec5).^2));

EU4=sqrt(sum(sum(test_vec1-train_vec4).^2));

EU5=sqrt(sum(sum(test_vec1-train_vec3).^2));

EU6=sqrt(sum(sum(test_vec1-train_vec2).^2));

EU7=sqrt(sum(sum(test_vec1-train_vec1).^2));
%8,3,1
if EU<EU1 && EU<EU2 && EU<EU3 && EU<EU4 && EU<EU5 && EU<EU6 && EU<EU7
    disp('matached with speaker no .8');

elseif EU1<EU && EU1<EU2 && EU1<EU3 && EU1<EU4 && EU1<EU5 && EU1<EU6 && EU1<EU7
        disp('matched with speaker no . 7');

elseif EU2<EU1 && EU2<EU && EU2<EU3 && EU2<EU4 && EU2<EU5 && EU2<EU6 && EU2<EU7
        disp('matched with speaker no . 6');
elseif EU3<EU1 && EU3<EU2 && EU3<EU && EU3<EU4 && EU3<EU5 && EU3<EU6 && EU3<EU7
        disp('matched with speaker no . 5');
elseif EU4<EU1 && EU4<EU && EU4<EU3 && EU4<EU2 && EU4<EU5 && EU4<EU6 && EU4>EU7
        disp('matched with speaker no . 4');
elseif EU5<EU1 && EU5<EU2 && EU5<EU && EU5<EU4 && EU5<EU3 && EU5<EU6 && EU5<EU7
        disp('matched with speaker no . 3');
elseif EU6<EU1 && EU6<EU && EU6<EU3 && EU6<EU4 && EU6<EU5 && EU6<EU2 && EU6<EU7
        disp('matched with speaker no . 2');
elseif EU7<EU1 && EU7<EU2 && EU7<EU && EU7>EU4 && EU7<EU5 && EU7<EU6 && EU7<EU3
        disp('matched with speaker no . 1');
```

```
end



%-----------framing------------%
function [feature_vec]= feature_extract(y,Fs)
%[y Fs]=wavread('s1.wav');
[y_size c]= size(y);

sample_size=Fs*0.025;
sample_size=round(sample_size);
y_size=round(y_size);
samples=y_size/sample_size;

prev=0;
curr=0;
sig_samp=zeros(samples,sample_size);
f=1;
for i=1:samples
    if (i>1)
        f=prev+125;
        curr=f;
    end
    for j=1:sample_size

        sig_samp(i,j)=y(f,1);

        f=f+1;
    end
    if (i>=2)
        prev=curr;
    end
end

z=zeros(313,1);
for k=1:313
    z(k,1)=sig_samp(1,k);

end
%{
zj=zeros(626,1);
for kj=1:626
    zj(kj,1)=y(kj,1);
end
figure();
plot(z);
figure();
plot(zj);
%}
%-----------------periodigram---------%

sig=zeros(sample_size,1);

%sig_four=abs(fft2(sig_samp,512));
sig_four=zeros(samples,512);

for i1=1:samples

    for j1=1:313

        sig(j1,1)=sig_samp(i1,j1);
     end
    x=sig;
    xf=fft(x,512);
    for j2=1:512
        sig_four(i1,j2)=xf(j2,1);
    end

end

sig_four1=abs(sig_four).^2;
```

```
sig_four1=sig_four1./sample_size;

fj=zeros(313,1);
for kj=1:313
    fj(kj,1)=sig_four1(30,kj);
end
%----------------- freq points------------------%

fs=Fs/2;
fs_h=fs;
fs_l=300;
pointf=(fs_h-fs_l)/11;
o_freq_points=zeros(12,1);
o_freq_points(1,1)=fs_l;
o_freq_points(12,1)=fs_h;
b=fs_l;
for zj=2:11
    b=b+pointf;
    o_freq_points(zj,1)=floor(b);
end
%-----------------mel freq points------------------%
mel_h=floor(1125*log(1+(fs_h/700)));
mel_l=floor(1125*log(1+(fs_l/700)));

tri=(mel_h-mel_l)/11;
freq_points=zeros(12,1);
freq_points(1,1)=mel_l;
freq_points(12,1)=mel_h;
point=mel_l;
%-----------------mel freq points------------------%
for ik=2:11
    point=point+tri;
    freq_points(ik,1)=floor(point);
end
fil_point=zeros(12,1);
%-------------freq points converting to size of the array------------------------%
for jk=1:12

    fil_point(jk,1)=floor((513*(o_freq_points(jk,1)))/Fs);
end
%-------------generation of filter banks----------%
filter_1=[triang(fil_point(2,1)).',zeros(1,478)];
filter_2=[zeros(1,fil_point(1,1)),triang(fil_point(3,1)-
fil_point(1,1)).',zeros(1,456)];
filter_3=[zeros(1,fil_point(2,1)),triang(fil_point(4,1)-
fil_point(2,1)).',zeros(1,434)];
filter_4=[zeros(1,fil_point(3,1)),triang(fil_point(5,1)-
fil_point(3,1)).',zeros(1,411)];
filter_5=[zeros(1,fil_point(4,1)),triang(fil_point(6,1)-
fil_point(4,1)).',zeros(1,389)];
filter_6=[zeros(1,fil_point(5,1)),triang(fil_point(7,1)-
fil_point(5,1)).',zeros(1,367)];
filter_7=[zeros(1,fil_point(6,1)),triang(fil_point(8,1)-
fil_point(6,1)).',zeros(1,345)];
filter_8=[zeros(1,fil_point(7,1)),triang(fil_point(9,1)-
fil_point(7,1)).',zeros(1,323)];
filter_9=[zeros(1,fil_point(8,1)),triang(fil_point(10,1)-
fil_point(8,1)).',zeros(1,300)];
filter_10=[zeros(1,fil_point(9,1)),triang(fil_point(11,1)-
fil_point(9,1)).',zeros(1,278)];
filter_11=[zeros(1,fil_point(10,1)),triang(fil_point(12,1)-
fil_point(10,1)).',zeros(1,256)];
%------------applying filter banks and extracting values-------%
filter_vec=zeros(11,512);
filter_vec(1,:)=filter_1;
filter_vec(2,:)=filter_2;
filter_vec(3,:)=filter_3;
filter_vec(4,:)=filter_4;
filter_vec(5,:)=filter_5;
filter_vec(6,:)=filter_6;
filter_vec(7,:)=filter_7;
filter_vec(8,:)=filter_8;
filter_vec(9,:)=filter_9;
filter_vec(10,:)=filter_10;
filter_vec(11,:)=filter_11;
feature_vec=zeros(40,11);
```

```
for sk=1:40

    for dk=1:11
    l=sum(sig_four1(sk,:).*filter_vec(dk,:));
    feature_vec(sk,dk)=log(l);

    end

end
    feature_vec=dct(feature_vec);
end
```

Appendix C

Arduino Code for the Hardware Implementation

```
// Include the GSM library
#include <GSM.h>

// PIN Number
#define PINNUMBER ""

// initialize the library instance
GSM gsmAccess;
GSMVoiceCall vcs;

// Array to hold the number for the incoming call
char numtel[20];
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 7, 6, 5, 4);



void setup()

{
 lcd.begin(16, 2);
  // initialize serial communications and wait for port to open:
  Serial.begin(9600);
  lcd.clear();
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  Serial.println("Receive Voice Call");
lcd.print("Receive V. Call");
  // connection state
  boolean notConnected = true;

  // Start GSM kit
  // If your SIM has PIN, pass it as a parameter of begin() in quotes
  while (notConnected)
  {
    if (gsmAccess.begin(PINNUMBER) == GSM_READY)
      notConnected = false;
    else
    {
      Serial.println("Not connected");
      lcd.print("Not connected");
      delay(1000);
    }
  }

  // This makes sure the modem correctly reports incoming events
  vcs.hangCall();

  Serial.println("Waiting for a call");
   lcd.setCursor(0, 1);
  lcd.print("Waiting for call");
```

```
}
void usman(){
 lcd.clear();
 while (!Serial) {
  ; // wait for serial port to connect. Needed for Leonardo only
 }

 Serial.println("Receive Voice Call");
lcd.print("Receive V. Call");
 // connection state
 boolean notConnected = true;

 // Start GSM kit
 // If your SIM has PIN, pass it as a parameter of begin() in quotes
 while (notConnected)
 {
  if (gsmAccess.begin(PINNUMBER) == GSM_READY)
   notConnected = false;
  else
  {
   Serial.println("Not connected");
   lcd.print("Not connected");
   delay(1000);
  }
 }

 // This makes sure the modem correctly reports incoming events
 vcs.hangCall();

 Serial.println("Waiting for a call");
```

```
   lcd.setCursor(0, 1);
  lcd.print("Waiting for call");
}
int i=2;
void loop()
{
  // Check the status of the voice call
  switch (vcs.getvoiceCallStatus())
  {
    case IDLE_CALL: // Nothing is happening

      break;

    case RECEIVINGCALL: // Yes! Someone is calling us

      Serial.println("RECEIVING CALL");
    // lcd.print("RECEIVING CALL");

      // Retrieve the calling number
      vcs.retrieveCallingNumber(numtel, 20);

      // Print the calling number
      lcd.clear();
      Serial.print("Number:");
      Serial.println(numtel);
       lcd.print("No.:");
      lcd.setCursor(5, 0);
          lcd.print(numtel);
      // Answer the call, establish the call
      vcs.answerCall();
    break;

   case TALKING:  // In this case the call would be established

    Serial.println("TALKING. Press enter to hang up.");
  lcd.setCursor(0, 1);
   lcd.print("TALKING. ");

    // while (Serial.read() != '\n')
      delay(100000);
    vcs.hangCall();
    Serial.println("Hanging up and waiting for the next call.");
    usman();
    break;
  }

  delay(1000);


}

/**
  ******************************************************************************
  * @file    vr_sample_control_led.ino
  * @author  JiapengLi
  * @brief   This file provides a demostration on
              how to control led by using VoiceRecognitionModule
  ******************************************************************************
  * @note:
      voice control led
```

```
 ******************************************************************************
 * @section  HISTORY

   2013/06/13    Initial version.
 */
 #include  <TimerOne.h>
#include <SoftwareSerial.h>
#include "VoiceRecognitionV3.h"

/**
  Connection
  Arduino    VoiceRecognitionModule
   2   ------->     TX
   3   ------->     RX
*/
VR myVR(2,3);    // 2:RX 3:TX, you can choose your favourite pins.

uint8_t records[7]; // save record
uint8_t buf[64];

int led = 13;

#define onRecord    (0)
#define on1Record    (1)
#define on2Record    (2)
#define on3Record    (3)
#define on4Record    (4)
#define on5Record    (5)
#define on6Record    (6)
#define on7Record    (7)
```

```
/**
  @brief   Print signature, if the character is invisible,
           print hexible value instead.
  @param   buf     --> command length
           len     --> number of parameters
*/
void printSignature(uint8_t *buf, int len)
{
  int i;
  for(i=0; i<len; i++){
    if(buf[i]>0x19 && buf[i]<0x7F){
      Serial.write(buf[i]);
    }
    else{
      Serial.print("[");
      Serial.print(buf[i], HEX);
      Serial.print("]");
    }
  }
}

/**
  @brief   Print signature, if the character is invisible,
           print hexible value instead.
  @param   buf  -->  VR module return value when voice is recognized.
             buf[0]  -->  Group mode(FF: None Group, 0x8n: User, 0x0n:System
             buf[1]  -->  number of record which is recognized.
             buf[2]  -->  Recognizer index(position) value of the recognized record.
		buf[3]  -->  Signature length
		buf[4]~buf[n] --> Signature
*/
void printVR(uint8_t *buf)
{
  Serial.println("VR Index\tGroup\tRecordNum\tSignature");

  Serial.print(buf[2], DEC);
  Serial.print("\t\t");

  if(buf[0] == 0xFF){
    Serial.print("NONE");
  }
  else if(buf[0]&0x80){
    Serial.print("UG ");
    Serial.print(buf[0]&(~0x80), DEC);
  }
  else{
    Serial.print("SG ");
    Serial.print(buf[0], DEC);
  }
  Serial.print("\t");

  Serial.print(buf[1], DEC);
  Serial.print("\t\t");
  if(buf[3]>0){
    printSignature(buf+4, buf[3]);
  }
  else{
    Serial.print("NONE");
  }
  Serial.println("\r\n");
}
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 7, 6);
int freqStep = 100;
void setup()
{
  /** initialize */
  myVR.begin(4800);
  lcd.begin(16, 2);
  /** initialize */
 // lcd.clear();
  lcd.print("WELLCOME TO VR..");

  Serial.begin(115200);
  Serial.println("Elechouse Voice Recognition V3 Module\r\nControl LED sample");

  pinMode(led, OUTPUT);

  if(myVR.clear() == 0){
    Serial.println("Recognizer cleared.");
  }else{
    Serial.println("Not find VoiceRecognitionModule.");
    Serial.println("Please check connection and restart Arduino.");
    while(1);
  }
```

```
if(myVR.load((uint8_t)onRecord) >= 0){
  Serial.println("onRecord loaded");
//  lcd.print("Usman CALL");
}
 if(myVR.load((uint8_t)on1Record) >= 0){
  Serial.println("on1Record loaded");
//  lcd.print("Usman CALL");
}
 if(myVR.load((uint8_t)on2Record) >= 0){
  Serial.println("on2Record loaded");
//  lcd.print("Usman CALL");
}
 if(myVR.load((uint8_t)on3Record) >= 0){
  Serial.println("on3Record loaded");
//  lcd.print("Usman CALL");
}
 if(myVR.load((uint8_t)on4Record) >= 0){
  Serial.println("on4Record loaded");
//  lcd.print("Usman CALL");
}
 if(myVR.load((uint8_t)on5Record) >= 0){
  Serial.println("on5Record loaded");
//  lcd.print("Usman CALL");
}
 if(myVR.load((uint8_t)on6Record) >= 0){
  Serial.println("on6Record loaded");
//  lcd.print("Usman CALL");
}
 if(myVR.load((uint8_t)on7Record) >= 0){
  Serial.println("on7Record loaded");
  //  lcd.print("Usman CALL");
  }


}
void dim_check() {
  delay(3000);
   lcd.setCursor(0, 1);
     lcd.print("    ..........  ");
}


void loop()
{


  int ret;
  ret = myVR.recognize(buf, 50);
  if(ret>0){
    switch(buf[1]){


      case onRecord:
        /** turn on LED */
        digitalWrite(led, HIGH);
     lcd.setCursor(0, 1);
     lcd.print("Usman CALLING");


        break;


         case on1Record:
        /** turn on LED */
        digitalWrite(led, HIGH);
     lcd.setCursor(0, 1);
lcd.print("Usman CALLING");

   break;

   case on2Record:
   /** turn on LED */
   digitalWrite(led, HIGH);
lcd.setCursor(0, 1);
lcd.print("Usman CALLING");

   break;

    case on3Record:
   /** turn on LED */
   digitalWrite(led, HIGH);
lcd.setCursor(0, 1);
lcd.print("Usman CALLING");

   break;



 case on4Record:
   /** turn off LED*/
   digitalWrite(led, LOW);
  lcd.setCursor(0, 1);
  lcd.print("Abeer CALLING");
  // lcd.print("ABEER CALLING");
   break;
   case on5Record:
   /** turn off LED*/
```

```
      digitalWrite(led, LOW);
     lcd.setCursor(0, 1);
     lcd.print("Abeer CALLING");
     // lcd.print("ABEER CALLING");
      break;
      case on6Record:
      /** turn off LED*/
      digitalWrite(led, LOW);
     lcd.setCursor(0, 1);
     lcd.print("Abeer CALLING");
     // lcd.print("ABEER CALLING");
      break;
      case on7Record:
      /** turn off LED*/
      digitalWrite(led, LOW);
     lcd.setCursor(0, 1);
     lcd.print("Abeer CALLING");
     // lcd.print("ABEER CALLING");
      break;
    default:
      Serial.println("Record function undefined");
      break;
  }
  /** voice recognized */
  printVR(buf);


  dim_check();
//  if (led==HIGH) {lcd.setCursor(0, 1); lcd.print("Usman CALLING");}
// else if (led==LOW) {lcd.setCursor(0, 1); lcd.print("Abeer CALLING");}
 // else {lcd.setCursor(0, 1); lcd.print("                ");}
 }
}

/**
 ******************************************************************************
 * @file    vr_sample_control_led.ino
 * @author  JiapengLi
 * @brief   This file provides a demostration on
            how to control led by using VoiceRecognitionModule
 ******************************************************************************
 * @note:
      voice control led
 ******************************************************************************
 * @section  HISTORY

   2013/06/13    Initial version.
 */
 #include  <TimerOne.h>
#include <SoftwareSerial.h>
#include "VoiceRecognitionV3.h"

/**
  Connection
  Arduino    VoiceRecognitionModule
   2   ------->     TX
   3   ------->     RX
*/
VR myVR(2,3);    // 2:RX 3:TX, you can choose your favourite pins.

uint8_t records[7]; // save record
```

```
uint8_t buf[64];

int led = 13;

#define onRecord    (0)
#define on1Record    (1)
#define on2Record    (2)
#define on3Record    (3)
#define on4Record    (4)
#define on5Record    (5)
#define on6Record    (6)
#define on7Record    (7)


/**
  @brief   Print signature, if the character is invisible,
           print hexible value instead.
  @param   buf     --> command length
           len     --> number of parameters
*/
void printSignature(uint8_t *buf, int len)
{
  int i;
  for(i=0; i<len; i++){
    if(buf[i]>0x19 && buf[i]<0x7F){
      Serial.write(buf[i]);
    }
    else{
      Serial.print("[");
      Serial.print(buf[i], HEX);
      Serial.print("]");
    }
  }
}


/**
  @brief   Print signature, if the character is invisible,
           print hexible value instead.
  @param   buf  -->  VR module return value when voice is recognized.
             buf[0]  -->  Group mode(FF: None Group, 0x8n: User, 0x0n:System
             buf[1]  -->  number of record which is recognized.
             buf[2]  -->  Recognizer index(position) value of the recognized record.
             buf[3]  -->  Signature length
             buf[4]~buf[n] --> Signature
*/
void printVR(uint8_t *buf)
{
  Serial.println("VR Index\tGroup\tRecordNum\tSignature");

  Serial.print(buf[2], DEC);
  Serial.print("\t\t");

  if(buf[0] == 0xFF){
    Serial.print("NONE");
  }
  else if(buf[0]&0x80){
    Serial.print("UG ");
    Serial.print(buf[0]&(~0x80), DEC);
  }
  else{
    Serial.print("SG ");
    Serial.print(buf[0], DEC);
  }
  Serial.print("\t");

  Serial.print(buf[1], DEC);
  Serial.print("\t\t");
  if(buf[3]>0){
    printSignature(buf+4, buf[3]);
  }
  else{
    Serial.print("NONE");
  }
  Serial.println("\r\n");
}
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(A1, A0, 5, 4, 7, 6);
int freqStep = 100;



const int in1 = 8; //32     // the number of the input pin
const int in2 = 9; // 33
const int in3 = 10; //34
const int in4 = 11; // 35
int buttonState1 = 0;         // variable for reading the pushbutton status
int buttonState2 = 0;
int buttonState3 = 0;
int buttonState4 = 0;
```

```
void setup()
{
  pinMode(in1, INPUT);
 pinMode(in2, INPUT);
 pinMode(in3, INPUT);
 pinMode(in4, INPUT);

 /** initialize */
 myVR.begin(4800);
 lcd.begin(16, 2);
 /** initialize */
// lcd.clear();
 lcd.print("WELLCOME TO VR..");

 Serial.begin(115200);
 Serial.println("Elechouse Voice Recognition V3 Module\r\nControl LED sample");

 pinMode(led, OUTPUT);

 if(myVR.clear() == 0){
   Serial.println("Recognizer cleared.");
 }else{
   Serial.println("Not find VoiceRecognitionModule.");
   Serial.println("Please check connection and restart Arduino.");
   while(1);
 }

 if(myVR.load((uint8_t)onRecord) >= 0){
   Serial.println("onRecord loaded");
 //  lcd.print("Usman CALL");
 }
  if(myVR.load((uint8_t)on1Record) >= 0){
   Serial.println("on1Record loaded");
 //  lcd.print("Usman CALL");
 }
  if(myVR.load((uint8_t)on2Record) >= 0){
   Serial.println("on2Record loaded");
 //  lcd.print("Usman CALL");
 }
  if(myVR.load((uint8_t)on3Record) >= 0){
   Serial.println("on3Record loaded");
 //  lcd.print("Usman CALL");
 }
  if(myVR.load((uint8_t)on4Record) >= 0){
   Serial.println("on4Record loaded");
 //  lcd.print("Usman CALL");
 }
  if(myVR.load((uint8_t)on5Record) >= 0){
   Serial.println("on5Record loaded");
 //  lcd.print("Usman CALL");
 }
  if(myVR.load((uint8_t)on6Record) >= 0){
   Serial.println("on6Record loaded");
 //  lcd.print("Usman CALL");
 }
  if(myVR.load((uint8_t)on7Record) >= 0){
   Serial.println("on7Record loaded");
 //  lcd.print("Usman CALL");
 }
```

```
}
void dim_check() {
  delay(3000);
   lcd.setCursor(0, 1);
     lcd.print("    ..........  ");
}

void loop()
{

  buttonState1 = digitalRead(in1);//b
  buttonState2 = digitalRead(in2);//d
  buttonState3 = digitalRead(in3);//a
  buttonState4 = digitalRead(in4);//c

  if( buttonState3==1)
  {
    lcd.setCursor(0, 1);
     lcd.print("Usman CALLING");
    dim_check();
  }
  else if ( buttonState1==1)
  {
    lcd.setCursor(0, 1);
     lcd.print("Abeer CALLING");
     dim_check();
  }
   else if ( buttonState4==1)
  {
    lcd.setCursor(0, 1);
   lcd.print("Shan CALLING");
   dim_check();
}
else if ( buttonState2==1)
{
  lcd.setCursor(0, 1);
   lcd.print("NO CALLING");
   dim_check();
}



int ret;
ret = myVR.recognize(buf, 50);
if(ret>0){
  switch(buf[1]){

    case onRecord:
      /** turn on LED */
      digitalWrite(led, HIGH);
   lcd.setCursor(0, 1);
   lcd.print("Usman CALLING");

      break;

       case on1Record:
      /** turn on LED */
      digitalWrite(led, HIGH);
   lcd.setCursor(0, 1);
   lcd.print("Usman CALLING");
   break;


   case on2Record:

   /** turn on LED */

   digitalWrite(led, HIGH);

lcd.setCursor(0, 1);

lcd.print("Usman CALLING");


   break;


    case on3Record:

   /** turn on LED */

   digitalWrite(led, HIGH);

lcd.setCursor(0, 1);

lcd.print("Usman CALLING");


   break;




 case on4Record:

   /** turn off LED*/

   digitalWrite(led, LOW);

  lcd.setCursor(0, 1);

  lcd.print("Abeer CALLING");

  // lcd.print("ABEER CALLING");

   break;

   case on5Record:

   /** turn off LED*/

   digitalWrite(led, LOW);

  lcd.setCursor(0, 1);
```

```
      lcd.print("Abeer CALLING");
      // lcd.print("ABEER CALLING");
       break;
       case on6Record:
       /** turn off LED*/
       digitalWrite(led, LOW);
      lcd.setCursor(0, 1);
      lcd.print("Abeer CALLING");
      // lcd.print("ABEER CALLING");
       break;
       case on7Record:
       /** turn off LED*/
       digitalWrite(led, LOW);
      lcd.setCursor(0, 1);
      lcd.print("Abeer CALLING");
      // lcd.print("ABEER CALLING");
       break;
     default:
       Serial.println("Record function undefined");
       break;
    }
    /** voice recognized */
    printVR(buf);

    dim_check();
  //  if (led==HIGH) {lcd.setCursor(0, 1); lcd.print("Usman CALLING");}
  // else if (led==LOW) {lcd.setCursor(0, 1); lcd.print("Abeer CALLING");}
   // else {lcd.setCursor(0, 1); lcd.print("                ");}
  }
}
```

```
/**
 ******************************************************************************
 * @file    vr_sample_train.ino
 * @author  JiapengLi
 * @brief   This file provides a demostration on
 * how to train VoiceRecognitionModule to record your voice
 ******************************************************************************
 * @note:
 * Use serial command to control VoiceRecognitionModule. '
 * All commands are case insensitive. Default serial baud rate 115200.
 *
 * COMMAND       FORMAT                    EXAMPLE                Comment
 *
 * train         train (r0) (r1)...        train 0 2 45           Train records
 * load          load (r0) (r1) ...        load 0 51 2 3          Load records
 * clear         clear                     clear                  remove all records in  Recognizer
 * record        record / record (r0) (r1)...  record / record 0 79      Check record train status
 * vr            vr                        vr                     Check recognizer status
 * getsig        getsig (r)                getsig 0               Get signature of record (r)
 * sigtrain      sigtrain (r) (sig)        sigtrain 0 ZERO        Train one record(r) with
signature(sig)
 * settings      settings                  settings               Check current system settings
 ******************************************************************************
 * @section  HISTORY
 *
 * 2013/06/13    Initial version.
 */
#include <SoftwareSerial.h>
#include "VoiceRecognitionV3.h"
```

```
/**
 * Connection
 * Arduino    VoiceRecognitionModule
 * 2   ------->     TX
 * 3   ------->     RX
 */
VR myVR(2,3);    // 2:RX 3:TX, you can choose your favourite pins.

/***************************************************************************/
/** declare print functions */
void printSeperator();
void printSignature(uint8_t *buf, int len);
void printVR(uint8_t *buf);
void printLoad(uint8_t *buf, uint8_t len);
void printTrain(uint8_t *buf, uint8_t len);
void printCheckRecognizer(uint8_t *buf);
void printUserGroup(uint8_t *buf, int len);
void printCheckRecord(uint8_t *buf, int num);
void printCheckRecordAll(uint8_t *buf, int num);
void printSigTrain(uint8_t *buf, uint8_t len);
void printSystemSettings(uint8_t *buf, int len);
void printHelp(void);

/***************************************************************************/
// command analyze part
#define CMD_BUF_LEN      64+1
#define CMD_NUM     10
typedef int (*cmd_function_t)(int, int);
uint8_t cmd[CMD_BUF_LEN];
```

```
uint8_t cmd_cnt;

uint8_t *paraAddr;

int receiveCMD();

int checkCMD(int len);

int checkParaNum(int len);

int findPara(int len, int paraNum, uint8_t **addr);

int compareCMD(uint8_t *para1 , uint8_t *para2, int len);


int cmdTrain(int len, int paraNum);

int cmdLoad(int len, int paraNum);

int cmdTest(int len, int paraNum);

int cmdVR(int len, int paraNum);

int cmdClear(int len, int paraNum);

int cmdRecord(int len, int paraNum);

int cmdSigTrain(int len, int paraNum);

int cmdGetSig(int len, int paraNum);

int cmdSettings(int len, int paraNum);

int cmdHelp(int len, int paraNum);

/** cmdList, cmdLen, cmdFunction has correspondence */

const char cmdList[CMD_NUM][10] = {  // command list table

  {

    "train"  }

  ,

  {

    "load"  }

  ,

  {

    "clear"  }

  ,

  {
```

```
  "vr"  }
 ,
 {
  "record"  }
 ,
 {
  "sigtrain"  }
 ,
 {
  "getsig"  }
 ,
 {
  "Settings"  }
 ,
 {
  "test"  }
 ,
 {
  "help"  }
 ,
};
const char cmdLen[CMD_NUM]= {    // command length
 5,  //  {"train"},
 4,  //  {"load"},
 5,  //  {"clear"},
 2,  //  {"vr"},
 6,  //  {"record"},
 8,  //  {"sigtrain"},
 6,  //  {"getsig"},
 8,  //  {"Settings"},
  4,  //  {"test"},

  4,  //  {"help"}

};

cmd_function_t cmdFunction[CMD_NUM]={      // command handle fuction(function pointer
table)

  cmdTrain,

  cmdLoad,

  cmdClear,

  cmdVR,

  cmdRecord,

  cmdSigTrain,

  cmdGetSig,

  cmdSettings,

  cmdTest,

  cmdHelp,

};


/***************************************************************************/

/** temprory data */

uint8_t buf[255];

uint8_t records[7]; // save record


void setup(void)

{

  myVR.begin(9600);


  /** initialize */

  Serial.begin(115200);

  Serial.println(F("Elechouse Voice Recognition V3 Module \"train\" sample."));
  printSeperator();
  Serial.println(F("Usage:"));
  printSeperator();
  printHelp();
  printSeperator();
  cmd_cnt = 0;
}

void loop(void)
{
  int len, paraNum, paraLen, i;

  /** receive Serial command */
  len = receiveCMD();
  if(len>0){
    /** check if the received command is valid */
    if(!checkCMD(len)){

      /** check parameter number of the received command  */
      paraNum = checkParaNum(len);

      /** display the receved command back */
      Serial.write(cmd, len);

      /** find the first parameter */
      paraLen = findPara(len, 1, &paraAddr);

      /** compare the received command with command in the list */
      for(i=0; i<CMD_NUM; i++){
        /** compare command length */
      if(paraLen == cmdLen[i]){
        /** compare command content */
        if( compareCMD(paraAddr, (uint8_t *)cmdList[i], paraLen) == 0 ){
          /** call command function */
          if( cmdFunction[i](len, paraNum) != 0){
            printSeperator();
            Serial.println(F("Command Format Error!"));
            printSeperator();
          }
          break;
        }
      }
    }

    /** command is not supported*/
    if(i == CMD_NUM){
      printSeperator();
      Serial.println(F("Unkonwn command"));
      printSeperator();
    }
  }
  else{
    /** received command is invalid */
    printSeperator();
    Serial.println(F("Command format error"));
    printSeperator();
  }
}

/** try to receive recognize result */
```

```
  int ret;
  ret = myVR.recognize(buf, 50);
  if(ret>0){
    /** voice recognized, print result */
    printVR(buf);
  }
}

/**
 * @brief   receive command from Serial.
 * @param   NONE.
 * @retval  command length, if no command receive return -1.
 */
int receiveCMD()
{
  int ret;
  int len;
  unsigned long start_millis;
  start_millis = millis();
  while(1){
    ret = Serial.read();
    if(ret>0){
      start_millis = millis();
      cmd[cmd_cnt] = ret;
      if(cmd[cmd_cnt] == '\n'){
        len = cmd_cnt+1;
        cmd_cnt = 0;
        return len;
      }
      cmd_cnt++;
      if(cmd_cnt == CMD_BUF_LEN){
        cmd_cnt = 0;
        return -1;
      }
    }

    if(millis() - start_millis > 100){
      cmd_cnt = 0;
      return -1;
    }
  }
}

/**
 * @brief   compare two commands, case insensitive.
 * @param   para1  -->  command buffer 1
 * para2  -->  command buffer 2
 * len    -->  buffer length
 * @retval  0  --> equal
 * -1  --> unequal
 */
int compareCMD(uint8_t *para1 , uint8_t *para2, int len)
{
  int i;
  uint8_t res;
  for(i=0; i<len; i++){
    res = para2[i] - para1[i];
    if(res != 0 && res != 0x20){
      res = para1[i] - para2[i];
      if(res != 0 && res != 0x20){
        return -1;
      }
    }
  }
  return 0;
}

/**
 * @brief   Check command format.
 * @param   len  -->  command length
 * @retval  0  -->  command is valid
 * -1  -->  command is invalid
 */
int checkCMD(int len)
{
  int i;
  for(i=0; i<len; i++){
    if(cmd[i] > 0x1F && cmd[i] < 0x7F){

    }
    else if(cmd[i] == '\t' || cmd[i] == ' ' || cmd[i] == '\r' || cmd[i] == '\n'){

    }
    else{
      return -1;
    }
  }
  return 0;
}
```

```
/**
 * @brief   Check the number of parameters in the command
 * @param   len  -->  command length
 * @retval  number of parameters
 */
int checkParaNum(int len)
{
  int cnt=0, i;
  for(i=0; i<len; ){
    if(cmd[i]!='\t' && cmd[i]!=' ' && cmd[i] != '\r' && cmd[i] != '\n'){
      cnt++;
      while(cmd[i] != '\t' && cmd[i] != ' ' && cmd[i] != '\r' && cmd[i] != '\n'){
        i++;
      }
    }
    i++;
  }
  return cnt;
}

/**
 * @brief   Find the specified parameter.
 * @param   len       -->  command length
 * paraIndex -->  parameter index
 * addr      -->  return value. position of the parameter
 * @retval  length of specified parameter
 */
int findPara(int len, int paraIndex, uint8_t **addr)
{
  int cnt=0, i, paraLen;
  uint8_t dt;
  for(i=0; i<len; ){
    dt = cmd[i];
    if(dt!='\t' && dt!=' '){
      cnt++;
      if(paraIndex == cnt){
        *addr = cmd+i;
        paraLen = 0;
        while(cmd[i] != '\t' && cmd[i] != ' ' && cmd[i] != '\r' && cmd[i] != '\n'){
          i++;
          paraLen++;
        }
        return paraLen;
      }
      else{
        while(cmd[i] != '\t' && cmd[i] != ' ' && cmd[i] != '\r' && cmd[i] != '\n'){
          i++;
        }
      }
    }
    else{
      i++;
    }
  }
  return -1;
}

int cmdHelp(int len, int paraNum)
{
  if(paraNum != 1){
    return -1;
  }
  printSeperator();
  printHelp();
  printSeperator();
  return 0;
}

/**
 * @brief   Handle "train" command
 * @param   len     --> command length
 * paraNum --> number of parameters
 * @retval  0 --> success
 * -1 --> Command format error
 */
int cmdTrain(int len, int paraNum)
{
  int i, ret;
  if(paraNum < 2 || paraNum > 8 ){
    return -1;
  }

  for(i=2; i<=paraNum; i++){
    findPara(len, i, &paraAddr);
    records[i-2] = atoi((char *)paraAddr);
    if(records[i-2] == 0 && *paraAddr != '0'){
      return -1;
    }
  }
  printSeperator();
  ret = myVR.train(records, paraNum-1, buf);
  //  ret = myVR.train(records, paraNum-1);
  if(ret >= 0){
    printTrain(buf, ret);
  }
  else if(ret == -1){
    Serial.println(F("Train failed."));
  }
  else if(ret == -2){
    Serial.println(F("Train Timeout."));
  }
  printSeperator();
  return 0;
}

/**
 * @brief   Handle "load" command
 * @param   len     --> command length
 * paraNum --> number of parameters
 * @retval  0 --> success
 * -1 --> Command format error
 */
int cmdLoad(int len, int paraNum)
{
  int i, ret;
  if(paraNum < 2 || paraNum > 8 ){
    return -1;
  }

  for(i=2; i<=paraNum; i++){
    findPara(len, i, &paraAddr);
    records[i-2] = atoi((char *)paraAddr);
    if(records[i-2] == 0 && *paraAddr != '0'){
      return -1;
    }
  }
  //  myVR.writehex(records, paraNum-1);
  ret = myVR.load(records, paraNum-1, buf);
  printSeperator();
  if(ret >= 0){
    printLoad(buf, ret);
  }
  else{
    Serial.println(F("Load failed or timeout."));
  }
  printSeperator();
  return 0;
}

/**
 * @brief   Handle "clear" command
 * @param   len     --> command length
 * paraNum --> number of parameters
 * @retval  0 --> success
 * -1 --> Command format error
 */
int cmdClear(int len, int paraNum)
{
  if(paraNum != 1){
    return -1;
  }
  if(myVR.clear() == 0){
    printSeperator();
    Serial.println(F("Recognizer cleared."));
    printSeperator();
  }
  else{
    printSeperator();
    Serial.println(F("Clear recognizer failed or timeout."));
    printSeperator();
  }
  return 0;
}

/**
 * @brief   Handle "vr" command
 * @param   len     --> command length
 * paraNum --> number of parameters
 * @retval  0 --> success
 * -1 --> Command format error
 */
int cmdVR(int len, int paraNum)
{
  int ret;
  if(paraNum != 1){
    return -1;
  }
  ret = myVR.checkRecognizer(buf);
  if(ret<=0){
    printSeperator();
    Serial.println(F("Check recognizer failed or timeout."));
    printSeperator();
    return 0;
  }
  printSeperator();
  printCheckRecognizer(buf);
  printSeperator();
  return 0;
}

/**
 * @brief   Handle "record" command
 * @param   len     --> command length
 * paraNum --> number of parameters
 * @retval  0 --> success
 * -1 --> Command format error
 */
int cmdRecord(int len, int paraNum)
{
  int ret;
  if(paraNum == 1){
    ret = myVR.checkRecord(buf);
    printSeperator();
    if(ret>=0){
      printCheckRecordAll(buf, ret);
    }
    else{
      Serial.println(F("Check record failed or timeout."));
    }
    printSeperator();
  }
  else if(paraNum < 9){
    for(int i=2; i<=paraNum; i++){
      findPara(len, i, &paraAddr);
      records[i-2] = atoi((char *)paraAddr);
      if(records[i-2] == 0 && *paraAddr != '0'){
        return -1;
      }
    }

    ret = myVR.checkRecord(buf, records, paraNum-1);    // auto clean duplicate records
    printSeperator();
    if(ret>=0){
      printCheckRecord(buf, ret);
    }
    else{
      Serial.println(F("Check record failed or timeout."));
    }
    printSeperator();
  }
  else{
    return -1;
  }
  return 0;
}

/**
 * @brief   Handle "sigtrain" command
 * @param   len     --> command length
 * paraNum --> number of parameters
 * @retval  0 --> success
 * -1 --> Command format error
 */
int cmdSigTrain(int len, int paraNum)
{
  int ret, sig_len;
  uint8_t *lastAddr;
  if(paraNum < 2){
    return -1;
  }

  findPara(len, 2, &paraAddr);
  records[0] = atoi((char *)paraAddr);
  if(records[0] == 0 && *paraAddr != '0'){
    return -1;
  }

  findPara(len, 3, &paraAddr);
  sig_len = findPara(len, paraNum, &lastAddr);
  sig_len +=( (unsigned int)lastAddr - (unsigned int)paraAddr );

  printSeperator();
  ret = myVR.trainWithSignature(records[0], paraAddr, sig_len, buf);
  //  ret = myVR.trainWithSignature(records, paraNum-1);
  if(ret >= 0){
    printSigTrain(buf, ret);
  }
  else{
    Serial.println(F("Train with signature failed or timeout."));
  }
  printSeperator();


  return 0;
}


/**
 * @brief   Handle "getsig" command
 * @param   len     --> command length
 * paraNum --> number of parameters
 * @retval  0 --> success
 * -1 --> Command format error
 */
int cmdGetSig(int len, int paraNum)
{
  int ret;
  if(paraNum != 2){
    return -1;
  }


  findPara(len, 2, &paraAddr);
  records[0] = atoi((char *)paraAddr);
  if(records[0] == 0 && *paraAddr != '0'){
    return -1;
  }


  ret = myVR.checkSignature(records[0], buf);


  printSeperator();
  if(ret == 0){
    Serial.println(F("Signature isn't set."));
  }
  else if(ret > 0){
    Serial.print(F("Signature:"));
    printSignature(buf, ret);
    Serial.println();
  }
  else{
    Serial.println(F("Get sig error or timeout."));
  }
  printSeperator();

  return 0;
}

/**
 * @brief   Handle "test" command
 * @param   len     --> command length
 * paraNum --> number of parameters
 * @retval  0 --> success
 * -1 --> Command format error
 */
int cmdTest(int len, int paraNum)
{
  printSeperator();
  Serial.println(F("TEST is not supported."));
  printSeperator();
  return 0;
}

int cmdSettings(int len, int paraNum)
```

```
{
  int ret;
  if(paraNum != 1){
    return -1;
  }
  ret = myVR.checkSystemSettings(buf);
  if( ret > 0){
    printSeperator();
    printSystemSettings(buf, ret);
    printSeperator();
  }
  else{
    printSeperator();
    Serial.println(F("Check system settings error or timeout"));
    printSeperator();
  }
  return 0;
}

/*****************************************************************************/
/**
 * @brief   Print signature, if the character is invisible,
 * print hexible value instead.
 * @param   buf     --> command length
 * len     --> number of parameters
 */
void printSignature(uint8_t *buf, int len)
{
  int i;
  for(i=0; i<len; i++){
    if(buf[i]>0x19 && buf[i]<0x7F){
      Serial.write(buf[i]);
    }
    else{
      Serial.print(F("["));
      Serial.print(buf[i], HEX);
      Serial.print(F("]"));
    }
  }
}

/**
 * @brief   Print signature, if the character is invisible,
 * print hexible value instead.
 * @param   buf  -->  VR module return value when voice is recognized.
 * buf[0]  -->  Group mode(FF: None Group, 0x8n: User, 0x0n:System
 * buf[1]  -->  number of record which is recognized.
 * buf[2]  -->  Recognizer index(position) value of the recognized record.
 * buf[3]  -->  Signature length
 * buf[4]~buf[n] --> Signature
 */
void printVR(uint8_t *buf)
{
  Serial.println(F("VR Index\tGroup\tRecordNum\tSignature"));

  Serial.print(buf[2], DEC);
  Serial.print(F("\t\t"));

  if(buf[0] == 0xFF){
    Serial.print(F("NONE"));
  }
  else if(buf[0]&0x80){
    Serial.print(F("UG "));
    Serial.print(buf[0]&(~0x80), DEC);
  }
  else{
    Serial.print(F("SG "));
    Serial.print(buf[0], DEC);
  }
  Serial.print(F("\t"));

  Serial.print(buf[1], DEC);
  Serial.print(F("\t\t"));
  if(buf[3]>0){
    printSignature(buf+4, buf[3]);
  }
  else{
    Serial.print(F("NONE"));
  }
  Serial.println(F("\r\n"));
}

/**
 * @brief   Print seperator. Print 80 '-'.
 */
void printSeperator()
{
  for(int i=0; i<80; i++){
    Serial.write('-');
  }
  Serial.println();
}


/**
 * @brief   Print recoginizer status.
 * @param   buf  -->  VR module return value when voice is recognized.
 * buf[0]     -->  Number of valid voice records in recognizer
 * buf[i+1]   -->  Record number.(0xFF: Not loaded(Nongroup mode), or not set (Group mode))
(i= 0, 1, ... 6)
 * buf[8]     -->  Number of all voice records in recognizer
 * buf[9]     -->  Valid records position indicate.
 * buf[10]    -->  Group mode indicate(FF: None Group, 0x8n: User, 0x0n:System)
 */
void printCheckRecognizer(uint8_t *buf)
{
  Serial.print(F("All voice records in recognizer: "));
  Serial.println(buf[8], DEC);
  Serial.print(F("Valid voice records in recognizer: "));
  Serial.println(buf[0], DEC);
  if(buf[10] == 0xFF){
    Serial.println(F("VR is not in group mode."));
  }
  else if(buf[10]&0x80){
    Serial.print(F("VR is in user group mode:"));
    Serial.println(buf[10]&0x7F, DEC);
  }
  else{
    Serial.print(F("VR is in system group mode:"));
    Serial.println(buf[10], DEC);
  }
  Serial.println(F("VR Index\tRecord\t\tComment"));
  for(int i=0; i<7; i++){
    Serial.print(i, DEC);
    Serial.print(F("\t\t"));
    if(buf[i+1] == 0xFF){
      if(buf[10] == 0xFF){
        Serial.print(F("Unloaded\tNONE"));
      }
      else{
        Serial.print(F("Not Set\t\tNONE"));
      }
    }
    else{
      Serial.print(buf[i+1], DEC);
      Serial.print(F("\t\t"));
      if(buf[9]&(1<<i)){
        Serial.print(F("Valid"));
      }
      else{
        Serial.print(F("Untrained"));
      }
    }

    Serial.println();
  }
}

/**
 * @brief   Print record train status.
 * @param   buf  -->  Check record command return value
 * buf[0]     -->  Number of checked records
 * buf[2i+1]  -->  Record number.
 * buf[2i+2]  -->  Record train status. (00: untrained, 01: trained, FF: record value out of range)
 * (i = 0 ~ buf[0]-1 )
 * num  -->  Number of trained records
 */
void printCheckRecord(uint8_t *buf, int num)
{
  Serial.print(F("Check "));
  Serial.print(buf[0], DEC);
  Serial.println(F(" records."));

  Serial.print(num, DEC);
  if(num>1){
    Serial.println(F(" records trained."));
  }
  else{
    Serial.println(F(" record trained."));
  }

  for(int i=0; i<buf[0]*2; i += 2){
    Serial.print(buf[i+1], DEC);
    Serial.print(F("\t-->\t"));
    switch(buf[i+2]){
    case 0x01:
      Serial.print(F("Trained"));
      break;
    case 0x00:
      Serial.print(F("Untrained"));
      break;
    case 0xFF:
      Serial.print(F("Record value out of range"));
      break;
    default:
      Serial.print(F("Unknown Stauts"));
      break;
    }
    Serial.println();
  }
}

/**
 * @brief   Print record train status.
 * @param   buf  -->  Check record command return value
 * buf[0]     -->  Number of checked records
 * buf[2i+1]  -->  Record number.
 * buf[2i+2]  -->  Record train status. (00: untrained, 01: trained, FF: record value out of range)
 * (i = 0 ~ buf[0]-1 )
 * num  -->  Number of trained records
 */
void printCheckRecordAll(uint8_t *buf, int num)
{
  Serial.print(F("Check 255"));
  Serial.println(F(" records."));


  Serial.print(num, DEC);
  if(num>1){
    Serial.println(F(" records trained."));
  }
  else{
    Serial.println(F(" record trained."));
  }
  myVR.writehex(buf, 255);
  for(int i=0; i<255; i++){
    if(buf[i] == 0xF0){
      continue;
    }
    Serial.print(i, DEC);
    Serial.print(F("\t-->\t"));
    switch(buf[i]){
    case 0x01:
      Serial.print(F("Trained"));
      break;
    case 0x00:
      Serial.print(F("Untrained"));
      break;
    case 0xFF:
      Serial.print(F("Record value out of range"));
      break;
    default:
      Serial.print(F("Unknown Stauts"));
      break;
    }
    Serial.println();
  }
}


/**
 * @brief   Print check user group result.
 * @param   buf  -->  Check record command return value
 * buf[8i]    -->  group number.
 * buf[8i+1]  -->  group position 0 status.
 * buf[8i+2]  -->  group position 1 status.
 * ...               ...
 * buf[8i+6]  -->  group position 5 status.
 * buf[8i+7]  -->  group position 6 status.
 * (i = 0 ~ len)
 * len  -->  number of checked groups
 */
void printUserGroup(uint8_t *buf, int len)
{
  int i, j;
  Serial.println(F("Check User Group:"));
  for(i=0; i<len; i++){
    Serial.print(F("Group:"));
    Serial.println(buf[8*i]);
    for(j=0; j<7; j++){
      if(buf[8*i+1+j] == 0xFF){
        Serial.print(F("NONE\t"));
      }
      else{
        Serial.print(buf[8*i+1+j], DEC);
        Serial.print(F("\t"));
      }
    }
    Serial.println();
  }
}

/**
```

```
 * @brief   Print "load" command return value.
 * @param   buf  -->  "load" command return value
 * buf[0]    -->  number of records which are load successfully.
 * buf[2i+1]  -->  record number
 * buf[2i+2]  -->  record load status.
 * 00 --> Loaded
 * FC --> Record already in recognizer
 * FD --> Recognizer full
 * FE --> Record untrained
 * FF --> Value out of range"
 * (i = 0 ~ (len-1)/2 )
 * len  -->  length of buf
 */
void printLoad(uint8_t *buf, uint8_t len)
{
  if(len == 0){
    Serial.println(F("Load Successfully."));
    return;
  }
  else{
    Serial.print(F("Load success: "));
    Serial.println(buf[0], DEC);
  }
  for(int i=0; i<len-1; i += 2){
    Serial.print(F("Record "));
    Serial.print(buf[i+1], DEC);
    Serial.print(F("\t"));
    switch(buf[i+2]){
    case 0:
      Serial.println(F("Loaded"));
      break;
    case 0xFC:
      Serial.println(F("Record already in recognizer"));
      break;
    case 0xFD:
      Serial.println(F("Recognizer full"));
      break;
    case 0xFE:
      Serial.println(F("Record untrained"));
      break;
    case 0xFF:
      Serial.println(F("Value out of range"));
      break;
    default:
      Serial.println(F("Unknown status"));
      break;
    }
  }
}

/**
 * @brief   Print "train" command return value.
 * @param   buf  -->  "train" command return value
 * buf[0]    -->  number of records which are trained successfully.
 * buf[2i+1]  -->  record number
 * buf[2i+2]  -->  record train status.
 * 00 --> Trained
 * FE --> Train Time Out
 * FF --> Value out of range"
 * (i = 0 ~ len-1 )
 * len  -->  length of buf
 */
void printTrain(uint8_t *buf, uint8_t len)
{
  if(len == 0){
    Serial.println(F("Train Finish."));
    return;
  }
  else{
    Serial.print(F("Train success: "));
    Serial.println(buf[0], DEC);
  }
  for(int i=0; i<len-1; i += 2){
    Serial.print(F("Record "));
    Serial.print(buf[i+1], DEC);
    Serial.print(F("\t"));
    switch(buf[i+2]){
    case 0:
      Serial.println(F("Trained"));
      break;
    case 0xFE:
      Serial.println(F("Train Time Out"));
      break;
    case 0xFF:
      Serial.println(F("Value out of range"));
      break;
    default:
      Serial.print(F("Unknown status "));
      Serial.println(buf[i+2], HEX);
      break;
    }
  }
}

/**
 * @brief   Print "sigtrain" command return value.
 * @param   buf  -->  "sigtrain" command return value
 * buf[0]  -->  number of records which are trained successfully.
 * buf[1]  -->  record number
 * buf[2]  -->  record train status.
 * 00 --> Trained
 * F0 --> Trained, signature truncate
 * FE --> Train Time Out
 * FF --> Value out of range"
 * buf[3] ~ buf[len-1] --> Signature.
 * len  -->  length of buf
 */
void printSigTrain(uint8_t *buf, uint8_t len)
{
  if(len == 0){
    Serial.println(F("Train With Signature Finish."));
    return;
  }
  else{
    Serial.print(F("Success: "));
    Serial.println(buf[0], DEC);
  }
  Serial.print(F("Record "));
  Serial.print(buf[1], DEC);
  Serial.print(F("\t"));
  switch(buf[2]){
  case 0:
    Serial.println(F("Trained"));
    break;
  case 0xF0:
    Serial.println(F("Trained, signature truncate"));
    break;
  case 0xFE:
    Serial.println(F("Train Time Out"));
    break;
  case 0xFF:
    Serial.println(F("Value out of range"));
    break;
  default:
    Serial.print(F("Unknown status "));
    Serial.println(buf[2], HEX);
    break;
  }
  Serial.print(F("SIG: "));
  Serial.write(buf+3, len-3);
  Serial.println();
}

/**
 * @brief   Print "settings" command return value.
 * @param   buf  -->  "settings" command return value
 * buf[0]  -->  number of records which are trained successfully.
 * buf[1]  -->  record number
 * buf[2]  -->  record train status.
 * 00 --> Trained
 * F0 --> Trained, signature truncate
 * FE --> Train Time Out
 * FF --> Value out of range"
 * buf[3] ~ buf[len-1] --> Signature.
 * len  -->  length of buf
 */

const unsigned int io_pw_tab[16]={
  10,  15,  20,  25,  30,  35,  40,  45,
  50,  75,  100, 200, 300, 400, 500, 1000
};

void printSystemSettings(uint8_t *buf, int len)
{

  switch(buf[0]){
  case 0:
  case 3:
    Serial.println(F("Baud rate: 9600"));
    break;
  case 1:
    Serial.println(F("Baud rate: 2400"));
    break;
  case 2:
    Serial.println(F("Baud rate: 4800"));
    break;
  case 4:
    Serial.println(F("Baud rate: 19200"));
    break;
  case 5:
```

```
  Serial.println(F("Baud rate: 38400"));

  break;

default:

  Serial.println(F("Baud rate: UNKONOWN"));

  break;

}


switch(buf[1]){

case 0:

case 0xFF:

  Serial.println(F("Outpu IO Mode: Pulse"));

  break;

case 1:

  Serial.println(F("Outpu IO Mode: Toggle"));

  break;

case 2:

  Serial.println(F("Outpu IO Mode: Clear(When recognized) "));

  break;

case 3:

  Serial.println(F("Outpu IO Mode: Set(When recognized)"));

  break;

default:

  Serial.println(F("Output IO Mode: UNKONOWN"));

  break;

}


if(buf[2] > 15){

  Serial.println(F("Pulse width: UNKONOWN"));

}

else{
    Serial.print(F("Pulse Width: "));
    Serial.print(io_pw_tab[buf[2]], DEC);
    Serial.println(F("ms"));
  }

  if(buf[3] == 0 || buf[3] == 0xFF){
    Serial.println(F("Auto Load: disable"));
  }
  else{
    Serial.println(F("Auto Load: enable"));
  }

  switch(buf[4]){
  case 0:
  case 0xFF:
    Serial.println(F("Group control by external IO: disabled"));
    break;
  case 1:
    Serial.println(F("Group control by external IO: system group selected"));
    break;
  case 2:
    Serial.println(F("Group control by external IO: user group selected"));
    break;
  default:
    Serial.println(F("Group control by external IO: UNKNOWN STATUS"));
    break;
  }
}

void printHelp(void)
```

```
{
  Serial.println(F("COMMAND       FORMAT                    EXAMPLE
Comment"));
  printSeperator();
  // Serial.println(F("----------------------------------------------------------------------------------------------
------------"));
  Serial.println(F("train         train (r0) (r1)...          train 0 2 45            Train records"));
  Serial.println(F("load          load (r0) (r1) ...          load 0 51 2 3           Load records"));
  Serial.println(F("clear         clear                       clear                   remove all records in
Recognizer"));
  Serial.println(F("record        record / record (r0) (r1)...  record / record 0 79      Check record
train status"));
  Serial.println(F("vr            vr                          vr                      Check recognizer status"));
  Serial.println(F("getsig        getsig (r)                  getsig 0                Get signature of record
(r)"));
  Serial.println(F("sigtrain      sigtrain (r) (sig)          sigtrain 0 ZERO         Train one record(r)
with signature(sig)"));
  Serial.println(F("settings      settings                    settings                Check current system
settings"));
  Serial.println(F("help          help                        help                    print this message"));
}
```

Appendix D

Timeline

MONTHS	**TARGET**	**STATUS**
JUNE 2014	PROJECT APPROVED	DONE
JULY 2014	LITERATURE STUDYING	DONE
AUGUST 2014	LITERATURE STUDYING	DONE
SEPTEMBER 2014	LEARNING OF SOUND PROCESSING TECHNIQUES	DONE
OCTOBER 2014	SIMULATION DESIGNING (IMPLEMENING ALGORITHMS)	DONE
NOVEMBER 2014	SIMULATION DESIGNING (IMPLEMENING ALGORITHMS)	DONE
DECEMBER 2014	SIMULATION DESIGNING (IMPLEMENING ALGORITHMS)	DONE

MONTHS	**TARGET**	**STATUS**
JAN 2015	INTERFACING THE GSM KIT AND VR MODULE WITH ARDUINO KIT	DONE
FEB 2015	IMPLEMENTING THE CODE ON HARDWARE	DONE
MARCH 2015	IMPLEMENTING THE CODE ON HARDWARE	DONE
APRIL 2015	IMPLEMENTING THE CODE ON HARDWARE	DONE
MAY 2015	FINAL TESTING	DONE