Voice authentication for Linux systems
Fang, Hui
ProQuest Dissertations and Theses; 2010; ProQuest
pg. n/a

VOICE AUTHENTICATION FOR LINUX SYSTEMS

BY

HUI FANG

A Thesis Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Southern Connecticut State University
New Haven, Connecticut
January, 2011

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UMI Number: 1487404

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI®
Dissertation Publishing

UMI 1487404
Copyright 2010 by ProQuest LLC.
All rights reserved. This edition of the work is protected against
unauthorized copying under Title 17, United States Code.

ProQuest®

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

VOICE AUTHENTICATION FOR LINUX SYSTEMS

BY

HUI FANG

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Hrvoje Podnar, Department of Computer Science, and it has been approved by the members of the candidate's thesis committee. It was submitted to the School of Graduate Studies and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

[illegible]

Hrvoje Podnar, Ph.D.
Thesis Advisor

[illegible]

Lisa Lancor, Ph.D.
Second Reader

[illegible]

Imad Antonios, Ph.D.
Department Chairperson

[illegible]

Holly Crawford, Ph.D.
Dean, School of Graduate Studies

October 21, 2010
Date

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ABSTRACT

Author:	Hui Fang
Title:	VOICE AUTHENTICATION FOR LINUX SYSTEMS
Thesis Advisor:	Dr. Hrvoje Podnar
Institution:	Southern Connecticut State University
Year:	2011

In the last decade, voice authentication technology has experienced a rapid growth. Due to the unique characteristics of the human voice, voice authentication can be used to verify the identity of individuals. In this thesis, a voice authentication system was developed for Linux systems to provide an alternative to standard authentication methods. An Open Source voice authentication system was developed enabling user access to a PC with their voice. Users enroll in the system using a user-friendly interface application. Based on the enrollment voice samples, a user's voice print reference file is created and stored on the system. The system is then able to authenticate users based on their spoken credentials. The system contains two major components, a Training Module and an Authentication Module seamlessly integrated within the login module of the operating system. The system has been extensively tested with satisfactory results.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ACKNOWLEDGEMENT

First, I would like to thank my advisor: Dr. Hrvoje Podnar, for his help and full support throughout the project development. Without Dr. Podnar, the completion of the work would be nearly impossible. I would also like to thank Dr. Lisa Lancor for her help as a second reader and her valuable comments. Special thanks go to the Computer Science Department and the graduate students for giving me continued support. Last, but not least, I wish to especially thank my parents for unwavering support and understanding during the many hours I dedicated to achieving this milestone in my life and career.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CONTENTS

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF TABLES

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF FIGURES

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 1 INTRODUCTION

This chapter begins by covering key notions and definitions of essential authentication terms. The presentation continues with the overview of authentication based on digital sources of information. A list of current authentication tools is presented together with their advantages. The chapter finishes with the description of the voice authentication problem that this thesis will address.

1.1 Biometric Authentication

Wikipedia defines the authentication process to be: "the act of establishing or confirming something (or someone) as authentic" (Biometric, 2009). Today, authentication is usually applied using electronic means. Multiple forms of electronic authentication techniques have been developed including: user-password pair, biometric based authentication, magnetic stripe cards, smart cards, and Radio Frequency (RF) identification. As technology continues to develop, the future will certainly bring more complex and sophisticated methods of authentication such as computer chip implants, cell phone authentication, or DNA signature comparisons.

Individuals can be described by their features, many of which are unique. Given this uniqueness, techniques have been developed to measure these characteristics. Any combination of measurable physical or behavioral characteristics of a human subject can be collected. Such collection of data is also known as *biometric measures*. The science that measures and analyzes biological data is known as *biometrics*.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The most common biometric technology today is fingerprinting. Fingerprinting is based on the unique patterns of ridges and valleys on an individual's fingertips. At almost 44 percent of the market, fingerprints more than double the next most common technology – face scanning – according to the International Biometric Group (Frommer, D., 2006). In fact there is archaeological evidence that fingerprints have been used at least since 7000 to 6000 BC by the ancient Assyrians and Chinese as a form of identification. For decades, now law enforcement offices have been using fingerprinting technology successfully (Federal Bureau of Investigation (FBI), n.d.). With the fear of identity theft and personal information security threats continuing to rise, the growth of biometric markets will continue to grow globally. The International Biometric Group (IBG), a New York market research firm, estimates the $1.6 billion (USD) global biometric market will grow to $5.3 billion (USD) by 2010. (DigitalWenture, 2010)

Beyond fingerprints and face scans, other technologies used to differentiate people include measuring hand dimensions, scanning irises and retinas, comparing handwriting, recognizing keyboard typing patterns, and analyzing voice prints. All of these techniques differ in speed, stability, reliability, and cost effectiveness. Table 1.1 shows comparison of the characteristics of different biometric technologies (Mayes, 2004).

Table 1.1

Comparison of Biometrics

Characteristics	**Fingerprint**	**Hand**	**Retina**	**Iris**	**Face**	**Signature**	**Voice**
Ease of Use	High	High	Low	Medium	Medium	High	High
Error Incidence	Dryness, dirt, age	Hand injury, age	Glasses	Poor Lighting	Lighting, age, glasses, hair	Changing Signatures	Noise, colds, channel variables
Accuracy	High	High	Very high	Very high	High	High	High
User Acceptance	Medium	Medium	Medium	Medium	Medium	Very High	High
Required Security Level	High	Medium	High	Very high	Medium	Medium	Medium

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Long Term Stability	High	Medium	High	High	Medium	Medium	Medium

In addition to security, the driving force behind biometric verification is convenience. Traditional means of authentication which includes smart cards, magnetic stripe cards, photo ID cards, and physical keys have the disadvantage of getting lost, stolen, duplicated, or misplaced. Passwords can be forgotten, shared, or observed. In contrast to these traditional means of authentication, biometrics authentication has a unique advantage. In trust, biometric credentials are always readily available and cannot be easily lost or stolen. For this reason, Pine Crest Elementary in Sanford, Florida has students use their fingerprints instead of traditional meal tickets (Weber, 2007).

A disadvantage of biometric authentication methods is that the authentication process can sometimes suffer from erroneous original data, computer glitches, and the malfunction of authentication readers. Because of this, biometric systems still need to be improved in terms of accuracy and speed. Biometric systems with the false rejection rate under 1% (together with a reasonably low false acceptance rate) are somewhat rare today. Another disadvantage is that not everyone can use all types of biometric systems. Disabled people that are incapable of using their hands for example, cannot use fingerprint or hand-based systems. Visually impaired people have difficulties using iris or retina scanning based techniques. Thus, given the complexity and level of accuracy needed, some biometrics systems can be very expensive and as such might not be a preferred authentication method.

Today's complex systems are using a combination of features can be used to recognize an individual in order to improve accuracy of biometrical authentication. For example, Next Generation Identification (NGI) System was recently contracted by the Federal Bureau of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Investigation to expand on the current fingerprint-based system and add iris scans, palm prints, and facial images (Jones, 2008).

Voice authentication shares a relatively small part of the overall biometric market. Figure 1.1 shows that voice authentication shared only 4.4 percent of the market in 2007 (Covetek, 2010). However, voice authentication has recently received significant interest and has experienced a growth in popularity. The popularity is due to the fact that individuals find giving voice commands more natural than presenting a hand for fingerprint testing. Voice-based systems can be used effectively in situations where secure authentication is especially important. For example, banks and credit card companies are increasingly turning to voice authentication as a means of decreasing the potential for fraud and identity theft, and at the same time, cutting the costs associated with customer verification (VoiceID, 2004).

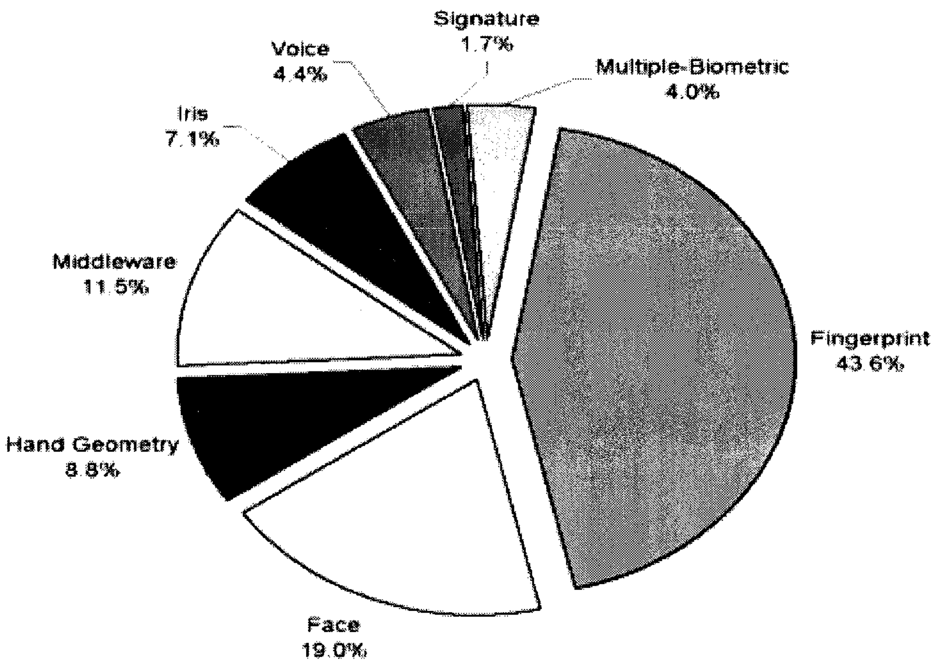


Figure 1.1. Overall Biometrics Market in 2007.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.2 Voice Recognition vs. Voice Authentication vs. Voice Identification

Because this thesis focuses on voice authentication, it is imperative to understand the differences between the definitions of *voice recognition*, *voice identification* and *voice authentication.*

Voice recognition, also known as speech recognition, is a scientific discipline that focuses on recognition of a spoken word. In other words, the main question that voice recognition addresses is the content of the transmitted audio information rather than the originator of such information. Voice recognition needs to resolve difficulties originating from variety of dialogues and languages. In contrast to voice recognition, voice authentication focuses on the individual rather than the content.

Voice identification systems store the voice prints of users to be used at a later time. The user accesses the system and speaks, without supplying any other identifying information. The system then searches through its database for the correct voice print and if a match exists it will authenticate the user. In a voice identification system, one sample is given and the system must search through the entire database and return its best guess. Voice identification focuses on techniques and methods for mapping an individual's voice to the correct person regardless of the content of the spoken word. A voice authentication system, on the other hand, might require a user to provide a particular word or phrase that is expected from the authentication system. Because voice authentication does not work with random words or phrase, this makes the authentication process easier to implement and deliver than is necessary in a voice identification system.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.3 Voice Authentication

Like fingerprints, a person's voice is unique. Compared to other biometrics including finger or hand printing and iris, retina or facial scanning, voice authentication is more natural, less intrusive, and more cost effective to deploy. To collect voice authentication data, one merely needs a microphone. Voice authentication is easy to use and easily accepted by users. It is quite natural to speak as opposed to having one's eye scanned. At this time, voice based authentication methods are the only biometric that allows users to authenticate remotely. Allowing a user to call a phone number and authenticate with one's bank vocally to perform a transaction is much easier than asking the user to go to the bank in person and authenticate via fingerprint. It is quick to enroll in a voice authentication system. The user is asked to speak a certain set of words or phrases, or to speak for a certain length of time.

Characteristics of the human voice are created by the unique shape of the speaker's mouth and throat, and are not affected by such things as a bad cold, extreme emotion, or attempts at disguise. As a result, voice authentication can be used to verify the identity of individuals.

Speech is produced by forcing the air over the vocal chords which generates a certain pitch and tone. The outgoing flow of air is filtered by the mouth, nose and throat. The final result is the vibration of the air. This vibration consist a number of sound waves with different frequencies. A visual representation of such vibration can be seen in Figure 1.2. This sound wave represents a female speaker saying "Good Morning" and "Good Night".

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

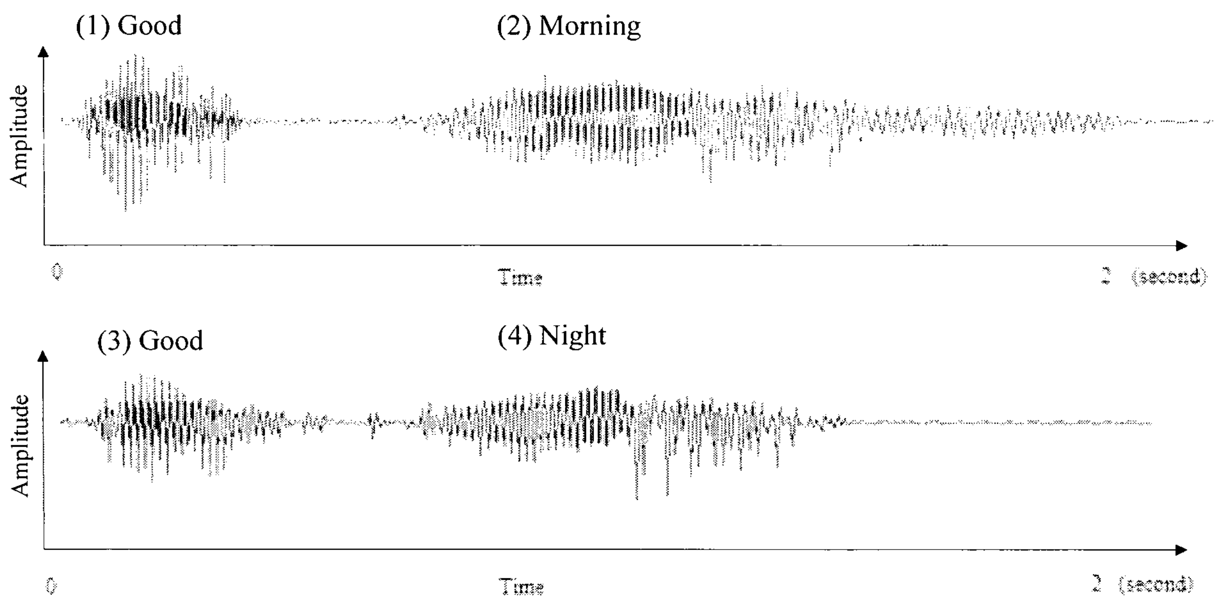


Figure 1.2. Voice Print Sample.

In Figure 1.2, two voice samples – "Good Morning" denoted by (1)-(2) and "Good Night" denoted by (3)-(4) have been taken over a period of two seconds for each. The time varying waveforms of the two phrases show changes in amplitude (the height) over time (the width). The spoken words can be easily identified. For example: the sound wave form of the word "Good" (pattern (1) in Figure 1.2) is visually similar to pattern (3) in both time and amplitude.

Different individuals will generate different voice samples that differ in pitch, rate, stress, duration of pauses, and articulation. All of these properties can be measured and extracted from the voice sample. These unique acoustic features of speech reflect both the anatomy, such as size and shape of the throat and mouth, and learned behavioral patterns such as speaking style and voice pitch. These speech attributes can then be used as a "biometric feature" of one's voice and then stored as a voice print in a database. In a voice authentication portion of an application, the person is asked to speak a required password or phrase and the system compares

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

that spoken utterance with the stored voice print and accepts or rejects the input, and then grants or denies access (Jamison, 2009).

Voice authentication systems have been used in a variety of security-related applications for banks, credit card agencies and law enforcement entities. The United States judicial system has used the technology, on a limited basis, for about 10 years to ensure that prisoners incarcerated in their homes or out on temporary passes are where they are supposed to be (VoiceID, 2004). The U.S. Social Security Administration is allowing employers to file W-2 wage reports online by gaining access using a combination of a personal identification number (PIN) and a voiceprint (Myers, 2004). The Rhode Island Department of Labor and Training implemented a new voice verification system that offers its unemployment insurance customers added protection against fraud and identity theft. This new application ensures that the person requesting payment each week is the same customer who filed an initial unemployment insurance claim with the Department. (State of Rhode Island, 2010)

1.4 Literature Review and Current State-Of-The-Art

Voice authentication products are already in existence. There is considerable amount of voice authentication research and development in industry, national laboratories, and universities. This research has produced multiple commercial products including Nuance Verifier, VoiceVault, and VoiceVerified.

Nuance Verifier (Nuance, 2007) is a voice authentication application package that enables businesses to provide secure access to sensitive information over the telephone. The application acts as a secure front end user interface to call center systems, verifying the identity of callers. Customers who have enrolled in the system can placed a phone call to the system and proceed with business transitions after being authenticated. The authentication process

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

requires a previously stored voiceprint. Nuance Verifier may be utilized for a variety of scenarios such as: customer care (address change, order status), transactions (bill pay applications, claims processing), and productivity (field service, sales force automation). Nuance Verifier includes a voice user interface, standard system configurations, and integrated reporting capabilities that provide access to valuable business and security-related/anti-fraud intelligence. Nuance Verifier uses industry standards such as J2EE and VoiceXML 2.0, the industry standard programming language for voice applications.

Nuance Verified provides interfaces for developers to extend its functionality. One of the applications is the Iraqi Enrollment via Voice Authentication Project (IEVAP) sponsored by the US Department of Defense (DoD) that studies the feasibility of bilingual (English and Jordanian-Arabic) speaker verification and speech recognition technology in support of the Global War on Terrorism security requirements.

VoiceVault (VoiceVault, 2009) is a platform which offers several types of voice authentication network accesses combined with remote password reset functionality. Additional functionalities are included such as payment verification, web voice authentication, voice sign-in and voice tracking. Payment Verification adds 'speak on the dotted line' digital signatures to credit or debit card payments over the phone. For all the voice authentication services, over 100 different calculations and tests are used to compare a previously enrolled voiceprint with a voice print generated at the time of verification. VoiceVault supports personalized 'shared secrets' – where the caller needs to answer her/his own personal security secret question recorded in hers/his own voice.

VoiceVerified by CSIdentity (VoiceVerified, 2008) is a voice authentication solution offers a real-time authentication process which may be used by colleges and universities that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

provides online education content. These institutions can use the VoiceVerified technology to enforce academic integrity of students enrolled in their online courses. In addition to traditional username and password, students are required to speak numbers displayed on the screen to start the online course. The system compares the spoken numbers to the stored voice print and if there is a match, access to the class is granted.

In addition to the large amount of commercial products, a number of freely available and Open Source solutions for voice authentication are also available. One of such solutions is VoiceAuth, written by Lucas Correia Villa Real (Lucas, 2003). VoiceAuth is a voice authentication software module for the Linux Pluggable Authentication Module (PAM) project. VoiceAuth requires users to provide a speech sample of their selected password. If the user and the password are recognized, the authentication is granted. The recognition process includes a frequency analysis phase done by Fourier transformations. VoiceAuth requires pre-installation of the Fourier library and a compatible sound system driver. Unfortunately the development and maintenance of the project were discontinued in 2003.

Another freely available Open Source application is the CMU Sphinx project originally developed at Carnegie Mellon University (Carnegie Mellon University, 2006). Sphinx is a complex speech recognition program written entirely in Java. The application package includes trainers, recognizers, acoustic models, and language models, together with detailed documentation. The main target of the application is recognition of speech. The licensing terms for the Sphinx engines and tools are derived from BSD. There are no restrictions against commercial use or redistribution.

A speech recognition project such as CMU Sphinx is not suitable for direct authentication implementation due to its complexity and scale. Speech recognition is useful but not in the case

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of voice authentication. To our knowledge, easily pluggable Open Source authentication software does not exist except for some attempts such as VoiceAuth that is no longer supported. This creates a product gap for authentication software that can be easily implemented for use on a variety of Linux platforms. Our project addressed this gap by providing Open Source, pluggable authentication software that is user friendly and easily expandable. More details about this voice authentication software are provided in the following chapters.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 2 VALIS SYSTEM DESIGN

This chapter describes the conceptual design and underlying architecture of the Voice Authentication Linux System (VALIS). VALIS consists of two major modules: the Training Module and the Authentications Module. In the Training Module, a reference file for each speaker containing their vocal characteristics using training sentences was created. In the Authentication Module, a sample of a speaker's voice is compared to the associated reference file to grant or deny access to the speaker.

2.1 Training Module

The main purpose of the Training Module is to enable a user to prepare their voice samples for future authentications. Users are required to record their voice sample multiple times. The system then processes the voice samples and extracts the biometric feature that can be used at a later time. In doing so, the system carefully refines the data in order to eliminate unwanted noise. It also attempts to reduce variations of voice samples that naturally occur in human voice.

The system is designed to enable a modular architecture. The main system components are shown in Figure 2.1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

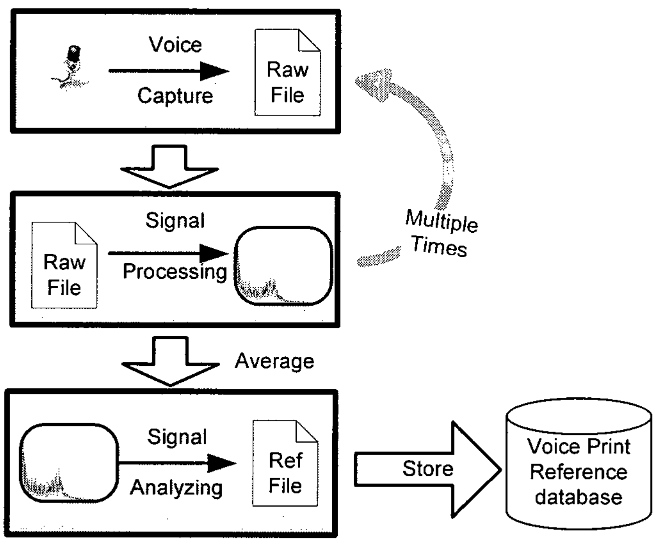


Figure 2.1. Training Module.

During the initial step, the user is required to provide a voice sample. The system provides user friendly interface for voice capture. The user's voice is recorded using a microphone and the sample is converted into a digital format raw file. To start the voice authentication process, the user is asked to select a short word to be used for the capture and authentication. Once the voice sample is captured, the digital voice representation gets displayed on the user's screen. At this point, the user has the freedom to discard the voice sample if it is not satisfactory. A user needs to collect multiple good quality voice samples in order to proceed to next phase. The voice sample is visually observed to make sure that is of good quality: reasonably loud but not too loud, isolated phase, signal is dominant over noise.

The user is asked to provide multiple good quality sound samples. The system calculates the average voice image to be used for further processing. The voice sample can be updated at a later time if the authentication process is erroneous. The errors are usually detected by the authentication process and are resolved by the trainer.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Ambient noise can distort the quality of voice samples requiring the repetition of data collection. Therefore, before a voice sample is processed, this noise needs to be removed from reduced in the original voice data. Such noise includes any unwanted sound which may be created by either analog or digital electronics. For example, background noise can be created by keyboard clicks, electric fans, or any human equipment caused by electrical appliances present in the same room where the sample is taken. In fact, most appliances generate electrical and audible interference with a sound capture system. Another source of noise, also known as "white noise", is created by moving electrons. This noise can be heard as acoustic noise during a playback of a voiceless signal sample. Visually, it manifests as "snow" on a television or video image. Noise also includes background noise that is created by other people. Such noise is simply defined as an unwanted by-product of other activities. (Noise, 2010)

Removing noise in VALIS is a two-step process. In the first step, the program assumes that the first 500 ms of recording is a voiceless sample containing the ambient background noise that remains constant throughout the duration of the recording. In other words, the first 500 ms of the voice sample contains only noise. The program uses this noise sample to filter the noise out from the remaining sample. This process is done by subtracting the background noise portion from the rest of recording.

In the second step, a digital audio filter is applied to the voice sample. A digital audio filter is a mathematical function that is used to separate data from unwanted signals such as interference, noise or unneeded signals. In VALIS, raw sound signals are processed by a digital audio filter that reduces signal outside of certain frequency ranges. A number of filters exist that utilize this approach. One example is a bandpass filter. A bandpass filter passes one frequency band and attenuates frequencies above and below that band. Figure 2.2 depicts a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

bandpass filter where "Passband" is the distance between low frequency (f1) and high frequency (f2).

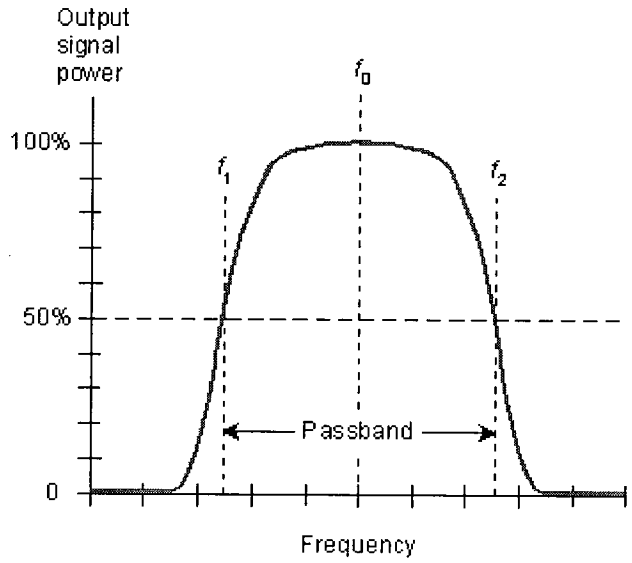


Figure 2.2. Bandpass Filter.

Other types of filters commonly used as highpass filters and lowpass filters. A highpass filter passes high frequencies and attenuates low frequencies while a lowpass filter passes low frequencies and attenuates high frequencies. A bandpass filter was chosen for VALIS since the most identifiable human voice peaks are produced in the range from 80Hz to 300Hz. Frequencies higher than 300Hz or lower than 80Hz are considered as noise in VALIS. While a Banpass filter is used with VALIS, the modular architecture of the software allows for easy filter additions or modifications. Additional filters are considered in the future work chapter.

Once the noise has been filtered, biometric measures can then be extracted from the filtered data. A number of mathematical algorithms can be applied including Fourier Frequency analysis, together with mathematical operations including signal filtering and averaging. The Training Module deconstructs each sound data by transforming it into sound frequencies. Given a fixed phrase, every person typically produces a sound sample using a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

slightly different set of basic pitches. The Training Module isolates the pitches represented by frequencies from the filtered sound data using the Fourier transform which converts the signal information to a magnitude and phase component for each frequency. Often the Fourier transform is converted to the power spectrum, which is the magnitude of each frequency component squared (Fourier Transform, 2010).

Figure 2.3 displays the frequency spectrum graph view of the phrase "Hello" by the same speaker. The phrase was recorded three times within the same physical environment and each recording is shown in Figure 2.3 a, b and c. The graphs in Figure 2.3 represent the frequency band of the three signals approximately between 0Hz to 2000Hz. Each graph shows changes in amplitude (the height) over the frequencies (the width) of the three samples. A greater amplitude means the frequency has a stronger presence in the original signal. Because human subjects usually produce a signal with the same frequencies based on their vocal chords, the highest amplitude peaks can be used as a unique property of the recorded signal.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

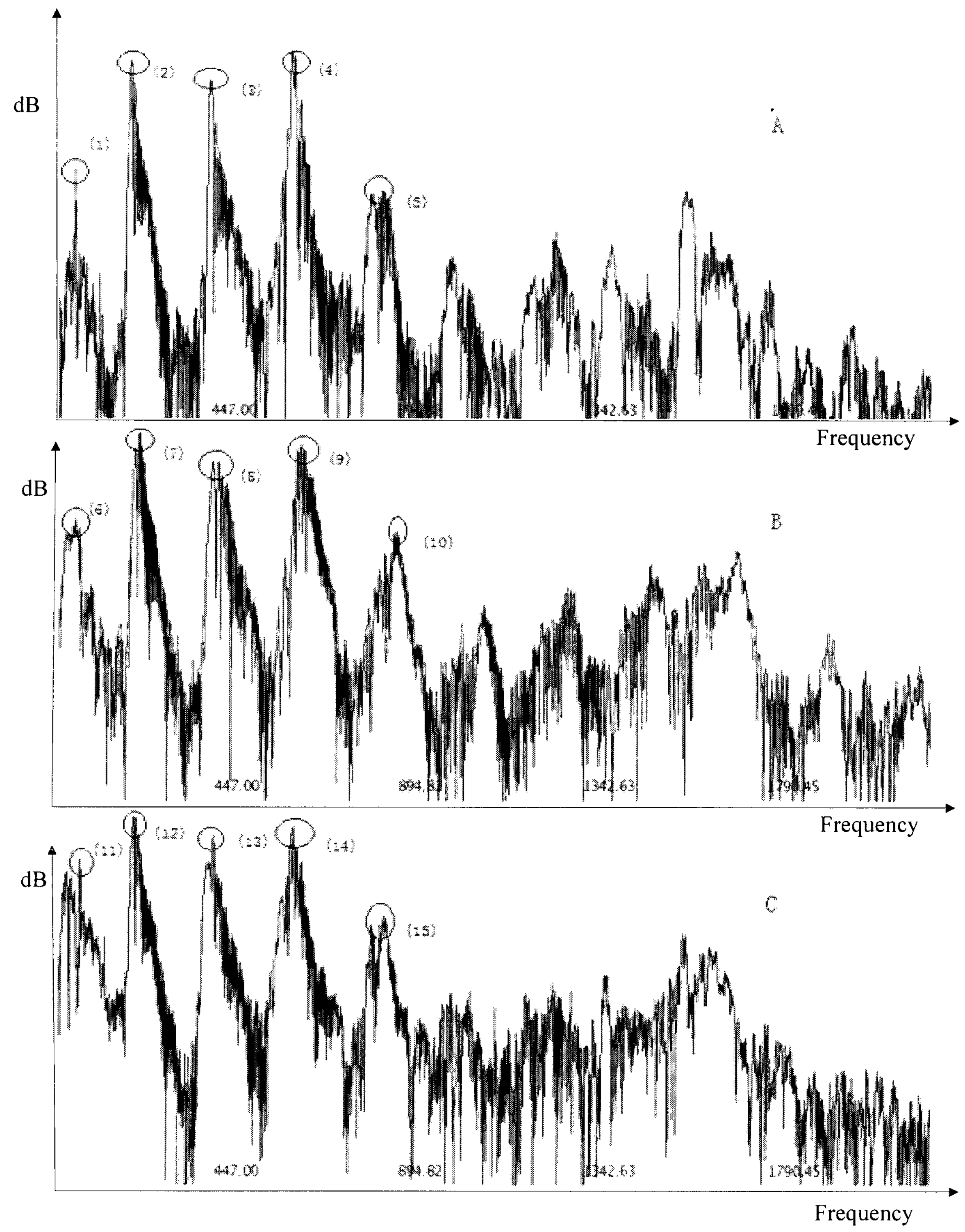


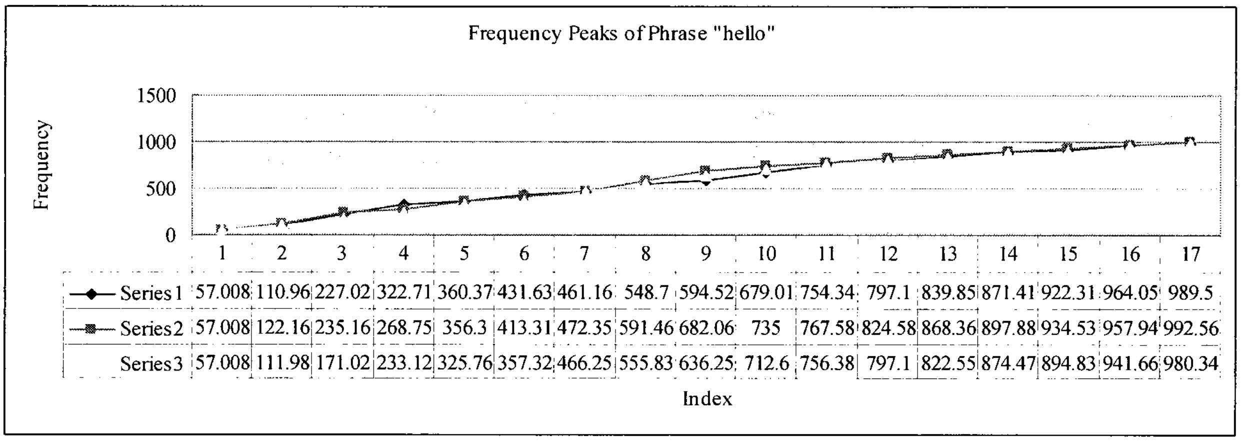
Figure 2.3. Frequency Spectrum of Phrase "Hello".

In Figure 2.3, it is easy to visually notice that all the amplitude peaks over the three graphs are similar to one another. In other words, the frequencies that achieved the highest

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

amplitude are comparable. For comparison purposes, the peaks in each graph are labeled. From Figure 2.3, points (1), (2), (3), (4), (5) in graph A correspond to points (6), (7), (8), (9), (10) in graph B, and points (11), (12), (13), (14), (15) in graph C.

Figure 2.4 displays a plot of 3 frequency peaks of the phrase "Hello" by the same speaker. The plot represents the frequency band between 0Hz to 1500Hz on the y axis and 17 indexes on the x axis. The data table of the plot is shown on the bottom of the graph. The diagram shows the peak points from the same speaker are nearly the same graphically and numerically.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
◆ Series1	57.008	110.96	227.02	322.71	360.37	431.63	461.16	548.7	594.52	679.01	754.34	797.1	839.85	871.41	922.31	964.05	989.5
■ Series2	57.008	122.16	235.16	268.75	356.3	413.31	472.35	591.46	682.06	735	767.58	824.58	868.36	897.88	934.53	957.94	992.56
Series3	57.008	111.98	171.02	233.12	325.76	357.32	466.25	555.83	636.25	712.6	756.38	797.1	822.55	874.47	894.83	941.66	980.34

Figure 2.4. Peaks of Phrase "Hello".

By examining the frequency detail for the specific word and comparing them, an analysis can be conducted to determine if they are the same frequency points and were made by the same person. There are several techniques to extract frequencies. Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) are commonly used algorithms for computing the frequency spectrum of the sampled data. There are many distinct FFT algorithms involving a wide range of mathematics, from simple complex-number arithmetic to group theory and number theory.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

VALIS examines the array of transformed frequency domains and finds all the possible peaks. A peak is an inverted "V" shape point which represents maximum energy in the frequency spectrum. A point is considered a peak if it is the highest point within a pre-determined frequency range window. In order to find the major peaks, the minor peaks are removed by using a smoothing function. "The simplest form of smoothing is the 'moving averages' method, which simply replaces each data value with the average of the neighboring values." (Peng, 2009) This technique creates a smoothed array of data containing an average of points from a moving window.

The program saves parameters including FFT, frequency peaks and duration of the recording from the data analysis of the sound sample in a voice print reference file. The reference file is a vector containing information of the biometric features of the sound signal. For each user, the program creates one reference file and stores in it a directory referenced by its login username. Once the reference file is created, an individual may be voice authenticated.

2.2 Authentication Module

The Authentication Module is primarily used to enable users to log into a Linux system if given their correct credentials. The credentials may be either voice or keyboard entry. The keyboard entry option was added in case the user has vocal problems. If keyboard entry was selected, the user will be prompted to enter his/her username and password as usual. If the voice entry method is chosen, a recorded voice prompt will be played. The process of voice authentication requires a number of steps.

First, the system plays a pre-recorded message that requests the users to speak the same phrase that they used during the training phase. At this point, the user is required to speak after the voice prompt has completed. A microphone connected to a sound card captures the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

person's voice which is then stored in a temporary file. The physical characteristics of the vocal tract and its harmonic and resonant frequencies are extracted and compared to the stored voice reference. The user may be asked to repeat the pass phrase to account for voice or environmental variances. If the voice is a match to the reference file, the access is granted. If the voice does not match any of the existing users' reference files, the system will return to the log in screen. At this point, the user has an option to retry the authentication process.

A conceptual view of the authentication phase is presented in Figure 2.5.

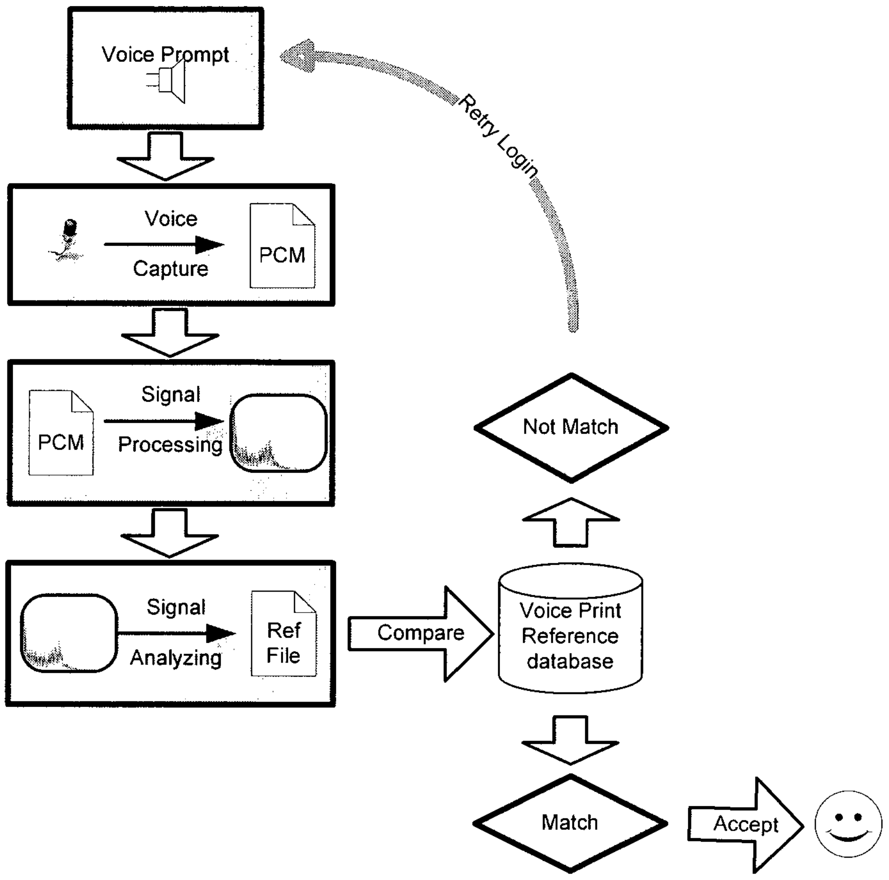


Figure 2.5. Authentication Module.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

All the modules from Figure 2.5 are embedded within an Authentications Module that is provided by the Linux kernel. The details about the embedding process will be provided in Chapter 3.

To authenticate a user successfully, the user is required to have an account created on the computer and have been enrolled in the system via the Training Module. The speaker is asked to speak into the microphone using exactly the enrolled password phrase. In order to ensure good voice sample quality, users are suggested to train when their voice is in normal condition. It is a good practice to avoid taking voice samples while having a cold or seasonal allergy. The voice sample should be taken speaking in a normal voice, preferably in a quiet and undisturbed environment.

The system takes the recording and processes the signal the same way it process the voice sample in the training module. The biometric features extracted from the voice sample are then compared to the voiceprint reference file stored on the local system. This is done by using pattern matching methods which includes a T-test and correlation.

T-test is a one of the most commonly used techniques for estimating the difference between sample means. The T-test assesses whether the means of two groups of sample data are statistically different from each other. There are two main versions of the T-test. The one-sample tests a hypothesis by comparing a sample mean with a hypothesized true mean. The two-sample uses a similar technique to compare two sample means against one another (T-test, 2010). The two sample T-test is used in the Authentication Module of VALIS to compare voice frequency peaks between the voice sample and the stored reference files.

A correlation is a measure which describes the degree of relationship between two variables. It shows whether and how strongly pairs of variables are statistically related

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(Correlation, 2010f). VALIS then uses correlation to compare FFTs and peaks stored in arrays, from the voice sample and reference files.

The system searches through the entire database and returns with a suitable match. The person that has the closest match is selected, or “nobody” will be returned if no reference is matched.

In the next chapter, we provide details on the methods and techniques used in VALIS.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 3 METHODOLOGY

In this chapter, the methods and techniques that are used in this project are discussed. VALIS is carefully designed to utilize only readily available Open Source software including Linux Fedora Core 5, KDE desktop application, Linux PAM, FFTW, ALSA driver and the C++ language. This chapter details the resolution of complexities involved in the system design utilizing the aforementioned components.

3.1 Development Platform

The main purpose of VALIS is to authenticate users who log into a Linux based computer system by voice. To accomplish this, the system is developed based on the Fedora Core 5 operating system environment. Fedora Linux can be installed on a wide range of hardware, including mobile devices. It is Open Source, stable, and flexible.

Multiple desktop environments are available, one of them being KDE. KDE is an Open Source desktop environment developed for Linux systems. The KDE project provides desktop functions and developer tools for writing stand-alone applications targeted for Linux systems (KDE, 2010). In this project, KDE was used to develop the graphical user interface of the Training Module. Figure 3.1 shows the interface for a typical KDE desktop.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

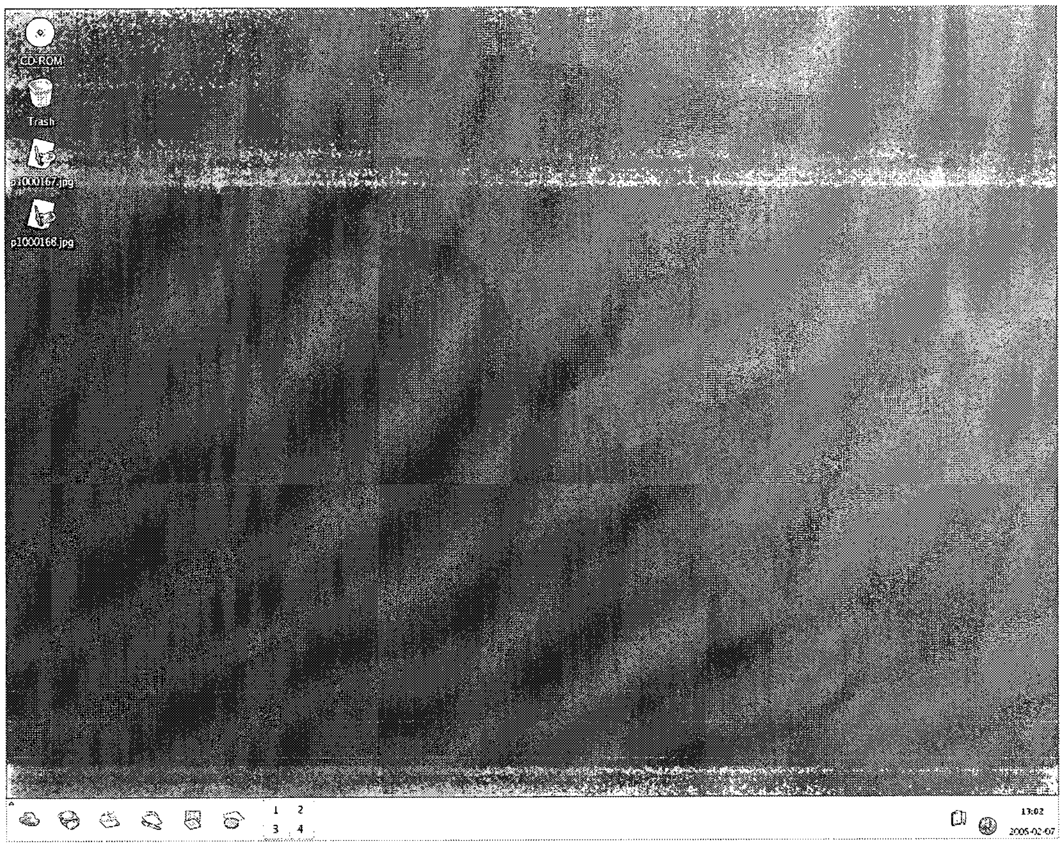


Figure 3.1. Typical KDE Desktop.

Linux Pluggable Authentication Module (PAM) is an Open Source Linux module that provides a multitude of flexible mechanism for authenticating users. Local system administrators can code their own authentication procedures and embed them into PAM so that the Linux kernel will call the corresponding authentication procedure at the moment of login. The customized authentication procedure can be targeted for different types of login such as a console user login, specific service login or a desktop environment user login. VALIS includes an authentication procedure that performs an authentication phase which collects the user's voice print, extracts the biometric data, and compares it against the database with the training data.

VALIS is developed with the C++ programming language which provides easy interfaces to the Linux authentication modules. Freely available Open Source signal processing software written in C++ was also used in this program.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2 Data Capture

During both the enrollment phase and the authentication phase of the system, a user's voice must be recorded and processed. A voiceprint reference file is created for comparison with pre-recorded samples taken for user identification. Voice samples are recorded by requesting communication with the sound card hardware. In order to do so, the application needs to communicate with an appropriate sound card driver.

The Open Source community provides a number of sound card device drivers for Linux. The most popular drivers are: Advanced Linux Sound Architecture (ALSA), Open Sound Systems (OSS), Enlightened Sound Daemond (EsounD), and analog Real-time Synthesizer (aRts). Among all those mentioned, ALSA is a driver of choice because it is already embedded in the Linux kernel. It provides control over the low-level audio functions suitable for VALIS. The driver comes with an API library 'libasound' that can be used in a user's application. Originally, older sound control functions needed to be programmed at the kernel level using specific sound device file handles with IOCtl calls. For backwards compatibility, ALSA provides kernel modules that emulate older sound drivers, so most existing sound applications continue to run unchanged. Sound can be recorded in a raw or compressed format. In this project, the sound is recorded in the raw data format.

The Pulse Code Modulation (PCM) interface is one of the commonly used interfaces for digital audio applications. ALSA supports the PCM interface for managing digital audio capture and playback. A sound card stores recorded samples in a hardware buffer. When the hardware buffer is full, it generates an interrupt. The kernel sound driver uses direct memory access (DMA) to transfer samples to an application buffer in memory. ALSA splits the buffer up into a series of periods and transfers the data in units of a period. A period stores frames,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

each of which contains the samples captured at one point in time. The Pseudocode for the program that uses the PCM interface is shown below (Tranter, 2004) in Figure 3.2.

```
open interface for capture or playback
set hardware parameters
(access mode, data format, channels, rate, etc.)
while there is data to be processed:
      read PCM data (capture)
      or write PCM data (playback)
close interface
```

Figure 3.2. PCM Pseudocode.

Once the recorded sample is “transferred” into digital data, the data can be processed for further analysis.

3.3 Signal Processing

As discussed previously, one of the biometric features of a voice sample is frequency based on the uniqueness of vocal chords. VALIS uses the Fourier transform to convert the data from a waveform view to a frequency view. Fourier analysis is performed by taking the whole recording, and re-evaluating it mathematically to determine the energy present at various frequencies. Both the time domain and frequency domain data sets represent exactly the same information, they just present the information in a different way. The time domain waveform shows the amplitude or sound pressure on a microphone vs. time, whereas the frequency view is represented by energy vs. frequency. Figure 3.3 shows a time domain graph and a frequency domain graph.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

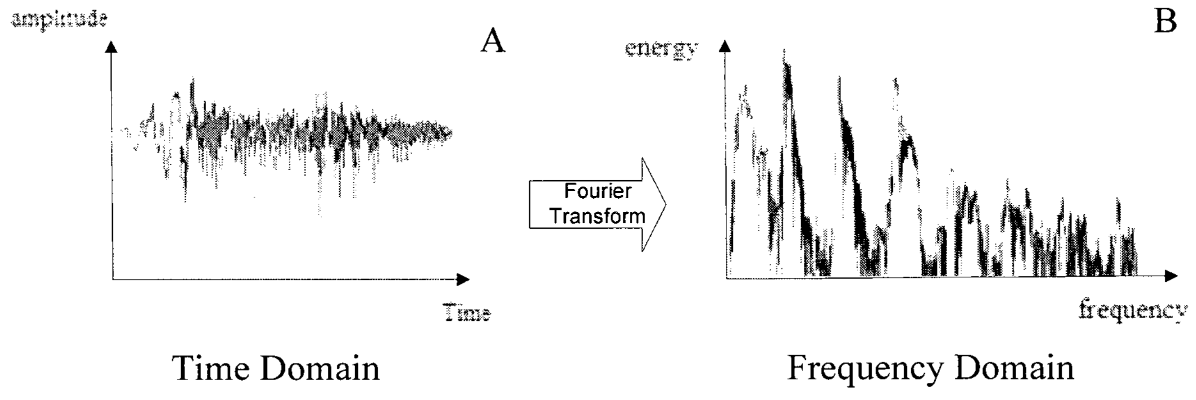


Figure 3.3. Time Domain vs. Frequency Domain.

In Figure 3.3 (B), the x-axis represents the frequency and the y-axis represents the amplitude spectrum. The amplitude spectrum shows which frequencies are contained in the original signal. The resulting transformation is in the form of a distribution of signal amplitude values as a function of frequency. In the frequency domain, this is the square of the FFT's magnitude.

VALIS uses the Fast Fourier Transform in the West (FFTW) to perform the frequency analysis. FFTW is a subroutine library for computing the Discrete Fourier Transform (DFT) in one or more dimensions of an arbitrary input discrete signal. FFTW provides both real and complex Fourier Transformations as well as the discrete cosine/sine transforms (DST/DCT). Since it is written in C, FFTW's package is very portable. The same program will perform well on most architecture without any modification. With each new version of the FFTW package, more efficient transformations are provided (FFTW, n.d). VALIS includes the FFTW library which is called to perform the FFT transform function.

Once the signal is brought into the frequency domain, it is further processed through the use of a smoothing algorithm which utilizes a moving window average to highlight the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

individual frequency peaks. After smoothing, the individual peaks are selected and stored into the appropriate user's reference file.

3.4 Signal Analysis

The voice is a difficult signal to analyze because it varies greatly over time and is complex in nature. Voice authentication systems usually expect a specific phase to be spoken. The user is asked to say a specific word, phrase or sentence. The voice is captured and stored in the form of a voice print. This voice print is then stored in a database as a reference file. To compare the biometric feature of the voice sample to the voice print, VALIS uses T-test and correlation.

A T-test is used to compare two small sets of data collected independently of one another. The T-test can be performed knowing just the mean, standard deviation, and number of data points from each sample. The formula for the T-test is shown in Figure 3.4 where $\bar{x}$ is the sample mean, *n* is the size of the sample and *s* is the sample standard deviation of the difference of two means. The denominator represents the experimental error in the measurement of the two sets of combined data.

$$t = \frac{|\bar{x}_1 - \bar{x}_2|}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

Figure 3.4. T-test.

The formula presented in Figure 3.4 calculates the T-test. The formula in Figure 3.5 returns the degrees of freedom to be compared with the T-test where *n* is the size of the sample and *s* is the sample standard deviation of the difference of two means.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\text{d.f.} = \frac{(s_1^2/n_1 + s_2^2/n_2)^2}{(s_1^2/n_1)^2/(n_1 - 1) + (s_2^2/n_2)^2/(n_2 - 1)}.$$

Figure 3.5. Degrees of Freedom.

Degrees of freedom represents the number of independent pieces of information available to estimate another piece of information. The first 10 degrees of freedom are shown in the probability table in Table 3.1. This table is based on the t distribution originally formulated by W.S. Gossett. To use the table provided, find the critical value that corresponds to the number of degrees of freedom that were calculated. If *t* exceeds the tabled value, the means are significantly different at the probability level that is listed. Gossett's tables report the lowest probability value for which t exceeds the critical value.

Table 3.1

T-test Probability Table

Degrees of Freedom	**Probability, p**			
	0.1	**0.05**	**0.01**	**0.001**
1	6.31	12.71	63.66	636.62
2	2.92	4.30	9.93	31.60
3	2.35	3.18	5.84	12.92
4	2.13	2.78	4.60	8.61
5	2.02	2.57	4.03	6.87
6	1.94	2.45	3.71	5.96
7	1.89	2.37	3.50	5.41
8	1.86	2.31	3.36	5.04
9	1.83	2.26	3.25	4.78
10	1.81	2.23	3.17	4.59

The columns in table 3.1 specify the probability of the conclusion being correct. For example, if the calculated t value exceeds the tabulated value for $p = 0.01$, then there is a 99% chance of the means being significantly different. By convention, a difference between means at the 95% level is "significant", a difference at 99% level is "highly significant" and a difference

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

at 99.9% level is "very highly significant". It should be noted that T-test statistics cannot actually prove or disprove a decision but rather can signify that there is a possibility of the reference file matching the voice print. Therefore to determine if a voice sample and a reference file belong to the same speaker, further analysis needs to be done.

An additional test is correlation which is used to compare similarities between two voice sample sets. Figure 3.6 contains the formula for Correlation where *x* represents the first sample set and *y* represents the second one to be compared to the first one. In figure 3.6, *n* denotes the number of samples. The formula returns *r* which is the correlation coefficient between sample set x and sample set y.

$$r = \frac{\sum_{i=1}^{n}\left((x_i - \bar{x})(y_i - \bar{y})\right)}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Figure 3.6. Correlation.

The correlation coefficient always takes a value between -1 and 1, with 1 or -1 indicating perfect correlation. A coefficient of -1 indicates strong negative correlation, while +1 suggests strong positive correlation. A positive correlation indicates a positive association between the variables (increasing values in one variable correspond to increasing values in the other variable), while a negative correlation indicates a negative association between the variables (increasing values is one variable correspond to decreasing values in the other variable). A coefficient of zero means there is no correlation between the two sets of variables. A correlation greater than 0.8 is generally described as strong, whereas a correlation between 0 and 0.5 is generally described as weak.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

VALIS uses the correlation coefficient to determine the strength of the association between two sets of voice sample data. In addition, correlation is used to compare the similarities between the strongest frequencies. Strong correlation between frequencies highlights the voice sample has strong probability of being from the same individual. Once correlation coefficients are calculated, they are stored into a table, and according to the comparison algorithm, the system will pick the user with the strongest correlation result or return "NOBODY" if all users' data yield weak correlations.

In the next chapter, the implementation of VALIS will be discussed.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 4 IMPLEMENTATION

VALIS is an application intended to authenticate users by their voice, as an alternative to users typing their passwords. In this chapter, the implementation of VALIS is discussed. There are two modules that are used to implement VALIS: the Training Module and the Authentication Module.

4.1 Training Module

The Training Module enrolls a person into the Voice authentication System. The user is asked to speak a preselected word. The response is captured, processed and stored in the form of a voice print. This voice print is then stored into a database as a reference file. This results in a user having an authentication profile in the system which is equivalent to a username and password. The full process is presented in Figure 4.1. At the beginning of the process, a voice sample is recorded and transformed into digital data (Read Data). The data is then normalized (Windowing) and filtered for noise reduction (Filter). The processed data is transformed into a frequency domain (FFT) using FFT and saved (Save FFT). After multiple voice samples are collected, the FFTs are averaged (Average Data) and normalized (Chop FFT). In order to find the strongest peaks, the FFT is processed through a smoothing filter (Smooth FFT). The frequency peaks (Find Peak) and the duration of the sound data (Find Duration) are saved into a voiceprint reference file as well as the FFT.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

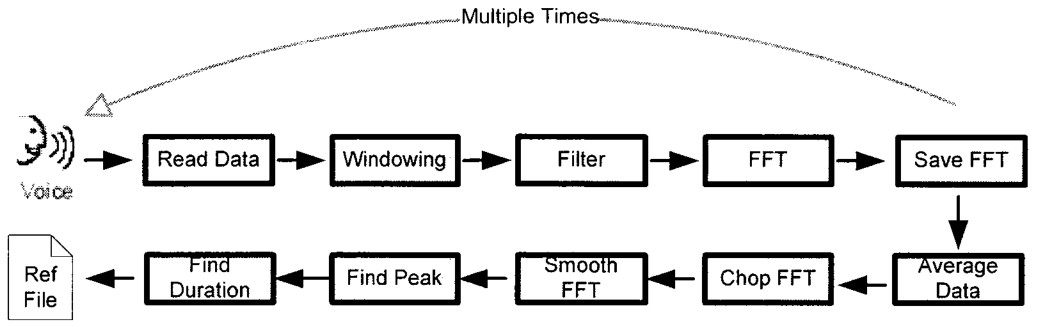


Figure 4.1. Diagram Representation of Training Module.

These functional blocks shown in Figure 4.1 can be grouped into three sections: Graphical user interface, sound recording, and data processing.

4.1.1 Graphical User Interface

The Training Module is designed to sample the user's voice for enrollment purposes. A friendly graphical user interface was designed using a code template written for the Linux window environment. The template is part of the KDE window development tutorial written by Antonio Larrosa Jimenez at 2002 (KDE Tutorial, 2002) and integrated into this project. KDE provides users with the ability to develop their own window applications and because of its popularity and ease-of-use, VALIS extends the functionality of KDE.

The Training Module is implemented with a customized KDE window. Figure 4.2 shows the main window of VALIS titled "Training". When the Training Module is called, the main window opens with a menu bar and toolbar.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

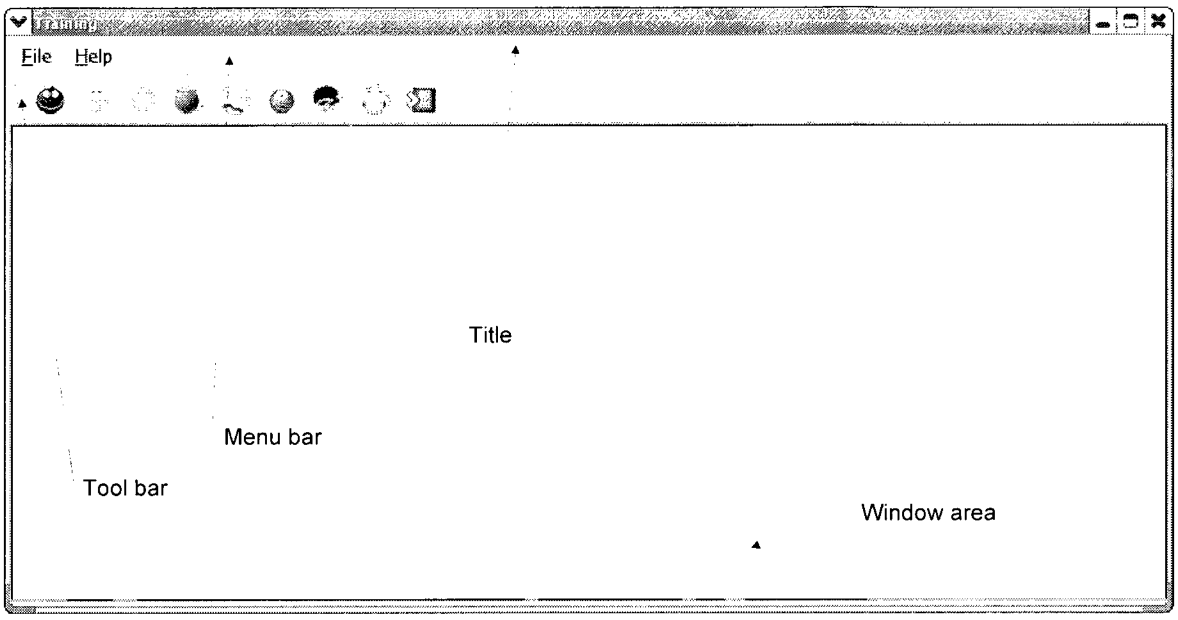
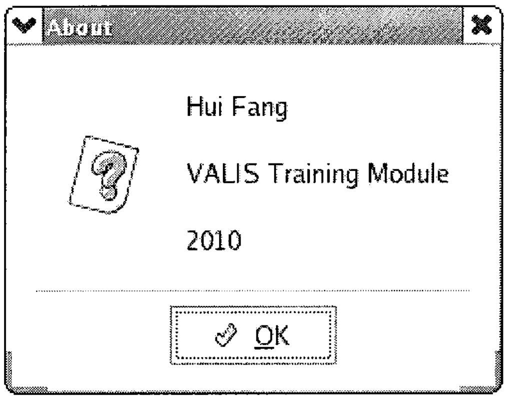


Figure 4.2. Training Module.

The menu bar has a simple list of menu items related to file operations. The menu bar has a File menu and a Help menu. On the File menu item, there is a "Quit" option which enables users to quit the program. On the Help menu item, besides KDE application help information, there is an "About" option which displays the author and program information shown in Figure 4.3.



Hui Fang

VALIS Training Module

2010

Figure 4.3. About.

Majority of interactive capabilities are handled by the toolbar and are provided to the user with a one-click capability. Due to the modularity of the implementation, adding additional

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

functionality to the toolbar can be easily performed. To create a toolbar in KDE is done by utilizing the KToolBar class. All toolbar items have a standard look and functionality. To create a new toolbar, one needs to provide the following code:

```
ToolBar *toolbar=new KToolBar(this);
```

Adding individual items to the toolbar can be done by adding a line of code as shown in Figure 4.4. Before creating a toolbar item that is clickable, its functionality must be specified. In the example code in Figure 4.4, clicking on the new icon will call the function "recordMe()". The icon look is specified by the external graphical image file: "/root/kde/p7/pics/record.gif". The additional parameters include: tooltip text and mouse click activation signal. The complete code can be found in the Appendix.

```
toolbar->insertButton(UserIcon("/root/kde/p7/pics/record.gif"), 0,
SIGNAL(clicked(int)),this,SLOT(recordMe()),TRUE, i18n("RecordVoice"));
```

Figure 4.4. Code Example of a Single ToolBar Item.

The toolbar has the total of nine icons shown as Figure 4.5.

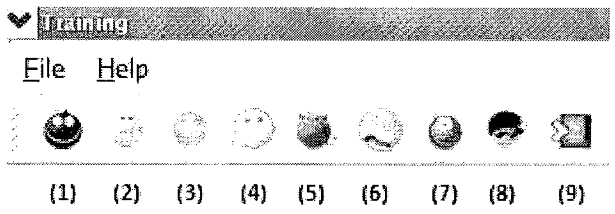


Figure 4.5. Toolbar.

A description of each toolbar item is given below:

(1) Record Voice: When this button is clicked, PCM opens the sound card device and records the voice sample for two seconds. The voice sample will be saved into a raw data file as bytes. The raw data is converted into digital format utilizing an array and drawn as a wave form in the window area described in Figure 4.2.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(2) Play Voice: When this button is clicked, the program will play the recorded raw data sound file through the sound card.

(3) Read Data: When this button is clicked, the program will read the raw data file and draw the wave form in the window area. This function is used to process pre-recorded voice samples saved on the system instead of voice samples coming directly from the microphone.

(4) Save Data: When this button is clicked, the system saves a copy of the FFT data in the memory in a binary (.bin) file along with the login username for further processing.

(5) Bandpass Filter: When this button is clicked, the program calls the "filterMe()" function in the Training module (p7) and applies a bandpass filter to the voice sample loaded in memory and displays the filtered sample data as a wave form in the window area.

(6) Smooth Filter: When this button is clicked, the program calls the function "SmoothMe()" and applies a smoothing filter to the FFT data and displays the smoothed FFT graph in the window area.

(7) FFT: When this button is clicked, the program transforms the sample data from the time domain into the frequency domain and displays a FFT graph in the window area.

(8) FindMe: When this button is clicked, the program reads the binary FFT file previously saved by item (4) and loads the information into memory. The program then averages the FFTs and applies the smoothing filter to the averaged FFT. The processing continues by finding frequency peaks. The newly processed data is then stored into a reference file.

(9) Quit: When this button is clicked, the Training Module quits.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Multiple toolbar items require drawing graphs in the Window Area. To do so, the `Qt` drawing library is used. `Qt` is a cross-platform Open Source application development framework used for the development of programs that require drawing capabilities on Linux systems. `Qt` is distributed under the terms of the GNU Lesser General Public License.

In order to utilize `Qt` capabilities in an application, several header files must be included; namely `<qPainter.h>` and `<qConvas.h>`. The next step would be to define a painter structure that is used for drawing:

```
QPainter *pp;
```

`QPainter` can draw everything from simple lines to complex shapes like pies and chords. The typical steps to draw are: (1) Construct a painter, (2) Set a pen, brush, etc., (3) Draw, and (4) Destroy the painter. A simple drawing example using qPainter is shown in Figure 4.6. `pp->begin (tt)` constructs a painter. `pp->setBrush(white)` sets the background to white color. `pp->drawRect(0,0,ww,wh)` draws a rectangle from coordinate 0,0 to window width (`ww`) and window height (`wh`) which was previously set to 800 and 400 respectively. `pp->setPen(green)` sets the pen to green. The `pp->drawLine` loop draws green lines between the `soundD` array points according to window size by calculations. And finally, `pp->end()` destroys the painter at the end.

```
pp->begin(tt);
pp->setBrush(white);
pp->drawRect(0,0,ww,wh);
pp->setPen(green);

for(i=1; i<ww;i++){
  pp->drawLine(i, (int)( wh*0.5-wh*0.5*(soundD[nsnd*(i-1)]/30000.00)),
i+1, (int)( wh*0.5 - wh*0.5*(soundD[nsnd*i] / 30000.00)) );
}
pp->end();
```

Figure 4.6. DrawData (Code).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Another use of `qPainter` is in drawing the frequency domain. The `DrawFFT()` function uses this same method to draw the frequency graph after the FFT is generated. Complete code for `DrawFFT()` can be found in the Appendix.

The Training Module also provides a zooming capability. Once an area is selected, it may be enlarged. When a graph is enlarged, the value corresponding to the selected point will display on the bottom of the graph to help describe the selected point. Figure 4.7 shows an example of the zooming capability in VALIS.

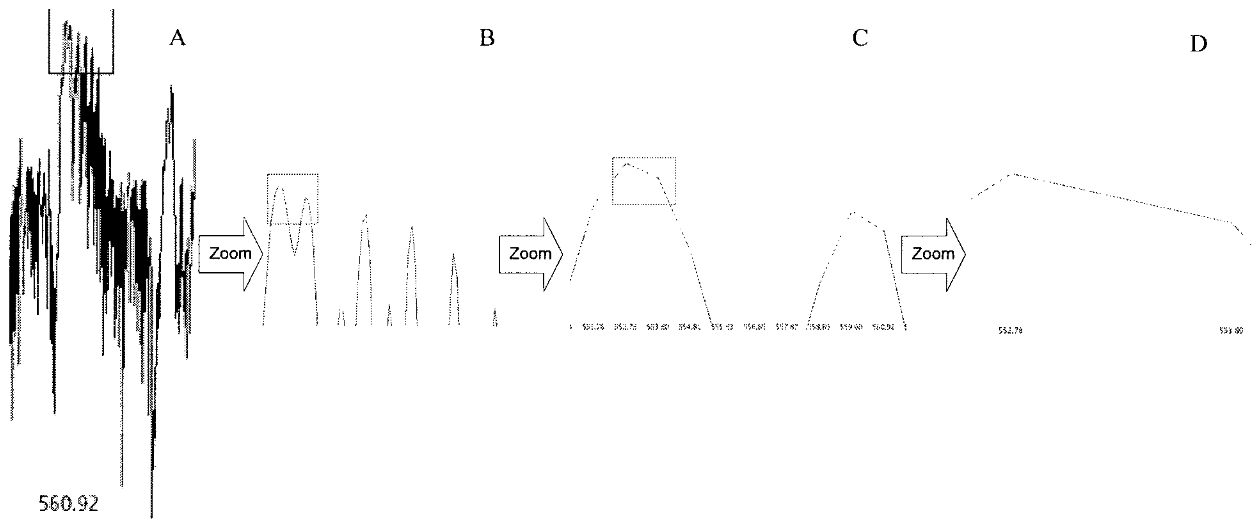


Figure 4.7. Zoom.

In this figure, picture A shows a rectangular area that has been selected to be enlarged. Picture B shows the enlarged area after the first execution of the zoom, and a subsequent area of selection for zooming. Picture C and D shows the more zooming execution is performed, the more detail of the graph is shown. This enables the user to visually observe the detail of the signal. Zooming is performed by using a mathematical linear formula for scaling as shown in Figure 4.8.

```
scaleA = wh/(TNTMIN - TNTMAX);
scaleB = - scaleA * TNTMAX;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
scaleC = ww/(Xfft[fftsize-1] - Xfft[0]);
scaleD = - scaleC * Xfft[0];
```

Figure 4.8. Scale (Code).

In Figure 4.8, `scaleA` and `scale B` are set for window area zooming purposes. `Scale C` and `D` are set for graph zooming purposes. By using the current mouse coordinates to select a new area (`x1,y1,x2,y2`), the program calculates the zooming ratio by using the code in Figure 4.9:

```
X1 = (cx1 - scaleD) / scaleC;
X2 = (cx2 - scaleD) / scaleC;
TNTMAX = (cy1 - scaleB) / scaleA;
TNTMIN = (cy2 - scaleB) / scaleA;
...
pp->drawLine(scaleC*Xfft[i-1]+scaleD,scaleA * Yfft[i-1] + scaleB ,
scaleC*Xfft[i  ]+scaleD,scaleA * Yfft[i  ] + scaleB);
```

Figure 4.9. DrawLine (Code).

`pp->drawLine` draws data in new coordinates calculated by `scales A, scales B, scales C, scales D.`

4.1.2 Sound recording functions

The `RecordData()` function is used to record a user's voice through the sound card. Once the data is captured, it is displayed as a graph on the screen. Figure 4.10 shows how a voice sample will display on the window area. From experience a time period of two seconds was used for recording voice sample. For test purposes, the voice was sampled using a single mono channel. During development, several words were tested including vowels such as "a", "e", and "o". "Hello" turned out to be a good word for the analysis. "Hello" can fit exactly into a two second time window and has enough pitches and vowels to be analyzed. More importantly, everyone says "Hello" in a consistent way.

Figure 4.10 is a screenshot of a good sound capture of the word "Hello". The sound wave fits in the middle of the two second window area. The beginning of the sound wave has

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

enough space for noise reduction purposes. The complete signal is captured within two seconds without being streamed. More importantly, it has sufficient amplitude and a low level of noise.

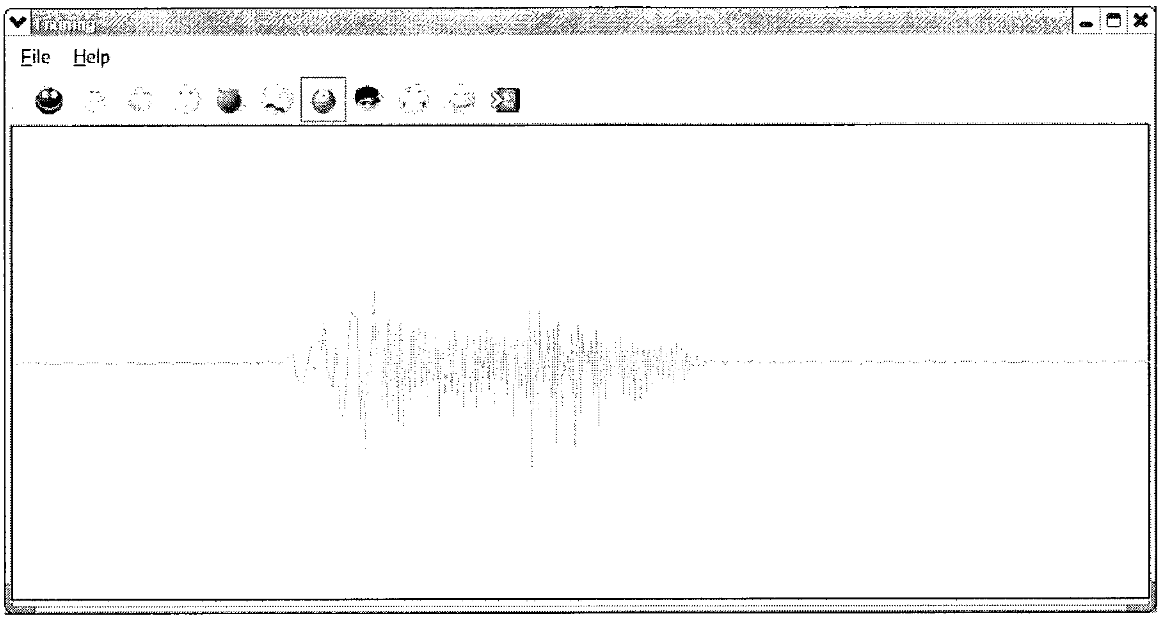


Figure 4.10. “Hello” Sound Wave Graph.

The `RecordData()` function opens the default PCM device, sets some parameters and then displays the values of most of the hardware parameters. To open the PCM device, a handle for the PCM device needs to be defined:

```
snd_pcm_t *handle;
```

The direction of the PCM stream can be specified which will determine the playback or capture of a sound sample. Figure 4.11 shows a piece of code taken from the ALSA project (Nagorni, 2002). In Figure 4.11, when the PCM stream opens, the mode as `SND_PCM_STREAM_CAPTURE` is specified. Before PCM data can be written to the soundcard, the access type, sample format, sample rate, number of channels, number of periods and period size need to be specified. In Figure 4.11, 44100 Hz is the standard sampling frequency for the recording. Frames is set to 32, and for mono channel, one frame has a length of two bytes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
snd_pcm_t *handle;
snd_pcm_hw_params_t *params;
snd_pcm_uframes_t frames;
rc = snd_pcm_open(&handle, "default", SND_PCM_STREAM_CAPTURE, 0);
snd_pcm_hw_params_alloca(&params);
snd_pcm_hw_params_any(handle, params);
snd_pcm_hw_params_set_access(handle, params,
SND_PCM_ACCESS_RW_INTERLEAVED);
snd_pcm_hw_params_set_format(handle, params, SND_PCM_FORMAT_S16_LE);
snd_pcm_hw_params_set_channels(handle, params, 1);
SRATE = 44100;
snd_pcm_hw_params_set_rate_near(handle, params, &SRATE, &dir);
frames = 32;
snd_pcm_hw_params_set_period_size_near(handle, params, &frames, &dir);
rc = snd_pcm_hw_params(handle, params);
snd_pcm_hw_params_get_period_size(params, &frames, &dir);
size = frames * 2;
buffer = (char *) malloc(size);
snd_pcm_hw_params_get_period_time(params, &SRATE, &dir);

loops = 2000000 / SRATE; //set to 2 seconds
```

Figure 4.11. PCM Capture Data (Code).

To convert the signal from a continuous wave to a series of digital measurements, a process called *sampling* is used. Sampling involves measuring the amplitude of the signal at regular intervals, several times per second. In the main processing loop, the samples are read from the sound hardware using the `snd_pcm_readi` command and written to standard output using `snd_pcm_write`. This function records approximately two seconds of data and saves it to a file. A separate function to play back the sound recording is used if necessary. The playback stream can be defined by the `SND_PCM_STREAM_PLAYBACK` command. Using the same sample rate, frame size and buffer size, the recorded sound sample can be played back for verification purposes.

Once verified, the recorded sound wave must to be imported into the program for future processing. This process requires a number of binary transformations including byte shifts and masking. The final result is an array of sound bytes. After the data is converted into a byte array, the signal processing can start.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.1.3 Signal Processing Functions

Once the sound sample is recorded and converted into digital data, the signal processing can be started. First, a test function is used to ensure that a signal is present. This testing finds the strongest point amplitude in the signal. If the resulting amplitude is smaller than a pre-described threshold, the program assumes no signal is present. At this point, a better sound sample must to be recorded.

If a signal is present, the duration of the sound sample must be calculated. This is again is based on a pre-described amplitude threshold for duration. The signal is assumed to be present if the signal level is above the threshold. If so, the duration of the signal is recorded. The threshold value is calculated as 50% of the strongest available signal. The program takes the difference between the first point and the last point of the threshold as the duration of the spoken word.

The program deliberately drops the first 500ms of the signal. This is done to remove unpredictable hardware related interference such as the initial start of the sound card or the click of the mouse to start recording.

The second 500 ms of data is used for noise reduction purposes. It is assumed that the beginning of the recording will contain no signal and only noise. This 500 ms of data can be subtracted from the rest of the sound data. Given the random nature of noise, it is expected that the noise subtraction will cancel the noise in the signal. This approach is well documented. (Kawamura, Fujii, Itoh, & Fukui, 2001)

The program applies a bandpass filter to the remaining sound data. Figure 4.12 consists of code where myb is defined as a bandpass filter and applied to sound data soundD[]. The filter is defined in a separate file called myQuad.c. myQuad.c is a simple implementation of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Biquad filters developed by Tom St Denis (Denis, 2001). This code was selected due to the availability of filters incorporated within it. The Biquad filter is defined below where dbGain is the gain of the filter, freq is the center frequency, srate is the sampling rate and bandwidth is the bandwidth in octaves.

```
extern biquad *BiQuad_new(int type, smp_type dbGain,
smp_type freq, smp_type srate, smp_type bandwidth);
```

In Figure 4.12, `myb` is called by the Training Module through inclusion of `myQuad.c` in the header file. `BPF` is defined as a bandpass Filter in `myQuad.c`.

```
biquad *myb;
myb = BiQuad_new(BPF, 1, 261.626, 44100, 3);
for (int a=0; a<cnt; a++)
     soundD[a] = BiQuad(soundD[a], myb);
```

Figure 4.12. Bandpass Filter (Code).

Once the data is filtered, the program transforms the data from the time domain into the frequency domain. Fourier analysis is currently one of the best techniques for performing detailed frequency analysis of time based signals. FFTW is a free C subroutine library for computing the DFT in one or more dimensions (FFTW, n.d.). To use FFTW, the `<fftw3.h>` file needs to be included in the program header. When compiling FFTW, it must link with the `fftw3` library, `-lfftw3 -lm` on the project.

First, the input and output arrays need to be allocated. `fftw_malloc` behaves like `malloc` except that it properly aligns the array when SIMD (Single Instruction Multiple Data) instructions are available. In Figure 4.13, `COMPRESS` is used to skip some points. The code copies the original data `soundd[]` into `in[]`, one point at a time.

```
startI = 0;
stopI= cnt -1;

fftmany  = stopI/COMPRESS-(startI -1)/COMPRESS;

in  = (double *) fftw_malloc( fftmany * sizeof(double));
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
out = (double *) fftw_malloc( fftmany * sizeof(double));
fftsize = fftmany/2+1;
cons = 0.5 * SRATE /(double)(fftsize) / (double)(COMPRESS);

i=0;
j=0;
while(i < fftmany){
        j++;
        if(j%COMPRESS == 0 && (j>= startI) && (j <= stopI) ){
        in[i++]=soundD[j-1];
        }
}
```
Figure 4.13. FFTW (Code).

A *plan* is an object that contains all the data that FFTW needs to compute the FFT. An `fftw_plan` contains all information necessary to compute the transform, including the pointers to the input and output arrays. `FFTW_R2HC` computes a real-input DFT with output into a "halfcomplex" format. `FFTW_MEASURE` tells FFTW to find an optimized plan by actually computing several FFTs and measuring their execution time (FFTW, n.d.). In Figure 4.14, the transform returns a plan from an input array (`in`) with `fftmany` number of members to an output array (`out`) in half complex format (`FFTW_R2HC`) and measures the time (`FFTW_MEASURE`). `fftw_destroy_plan` deallocates the `plan` and all its associated data to release the memory.

```
plan = fftw_plan_r2r_1d(fftmany,in,out,FFTW_R2HC,FFTW_MEASURE);
     ...
fftw_destroy_plan(plan);
```
Figure 4.14. Plan (Code).

FFTW implements *wisdom* to save plans to disk and restore them. At its simplest, wisdom provides a way of saving plans to disk so that they can be reused in other program runs. (FFTW-Tutorial, 2003) A wisdom file can be generated or imported. In VALIS, wisdom is imported from a wisdom file called `triwis.txt`. This is done by defining the wisdom file at the beginning of the program using:

```
#define WISDOM "triwis.txt"
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

`fftw_import_wisdom_from_file` in Figure 4.15 will load the wisdom file, which is stored in the same directory with the Training Module. `fftw_export_wisdom_to_file` will save the wisdom file when the calculation is finished.

```
/* get wisdom */
if(wisdomptr = fopen(WISDOM,"r")){
  fftw_import_wisdom_from_file(wisdomptr);
}
              ...

 /* save wisdom */
wisdomptr = fopen(WISDOM,"w");
fftw_export_wisdom_to_file(wisdomptr);
fclose(wisdomptr);
```

Figure 4.15. Wisdom (Code).

After the data is transformed from the time domain into the frequency domain using FFTW, a smoothing function is used to average the frequency graph in order to define the major peaks. The smoothing techniques shift the position of points making up a frequency plot, in order to remove small deviations and capture only the most significant trends. Figure 4.16 shows the code of the smoothing function. In Figure 4.17, a variable `tmp` is declared to sum the neighboring three points and average them.

```
if(size>=3){
      tmp[0] = sample[0];
      for(i=1;i<size-1;i++){
            tmp[1] = sample[i];
            sample[i] = (tmp[0] + tmp[1] + sample[i+1]) / 3.0;
            tmp[0] = tmp[1]; }
}
```

Figure 4.16. Smoothing (Code).

Figure 4.17 shows graphically what the data looks like before and after the smoothing filter.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

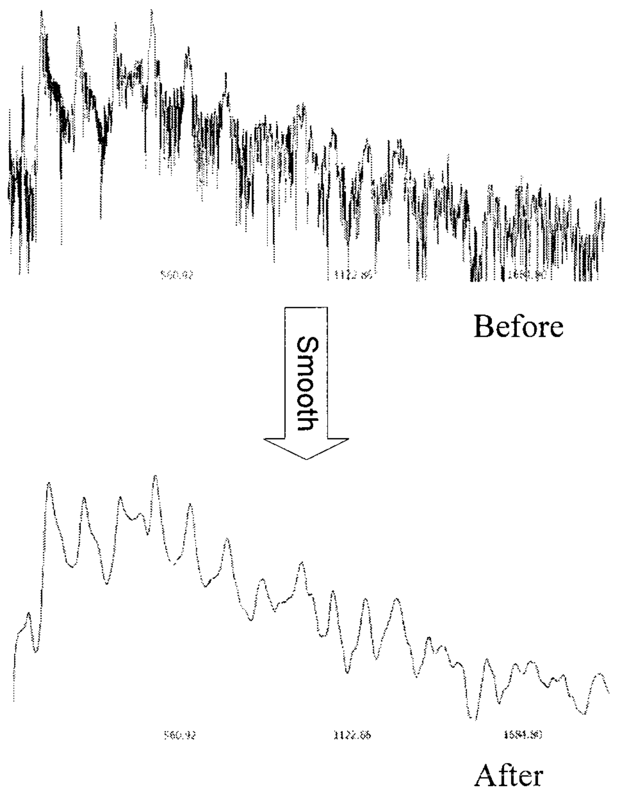


Figure 4.17. Original Signal and Filtered Signal After Smoothing Function.

Once the data is smoothed, the program finds all the possible peaks and saves them into an array. The FindPeak() function provides a procedure "Peak" that locates peaks in the FFT dataset using two vectors, x and y for example, representing frequency and counts. A peak is determined by the slope of the line between two points before the peak and two points after the peak. Figure 4.18 shows the code for FindPeak().

```
for(a=(i1+2);a<(Csize-2); a++)
if  ( (YfftSmooth[a]>YfftSmooth[a-1]) && (YfftSmooth[a]>YfftSmooth[a-2])
 && (YfftSmooth[a] > YfftSmooth[a+2]) &&  (YfftSmooth[a] >
 YfftSmooth[a+1]))
      YPeak[Psize]=YfftSmooth[a];
```

Figure 4.18. FindPeak (Code).

Figure 4.19 shows how the program finds all possible peaks by dynamically increasing the array size. An array is initially defined for storing the 10 peaks. Every time a peak is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

found, the variable "capacity" is decreased by 1 point. When capacity reaches 0, the size of the peak array will be doubled and newly found peaks will be added to it.

```
    if (capacity==0) {
     capacity=Psize;
     YPeak=(double *)realloc((double *)YPeak, Psize*2*sizeof(double));
      indexP=(int *)realloc((int *)indexP, Psize*2*sizeof(int));
   }
```

Figure 4.19. Increase Peak (Code).

4.2 Authentication Module

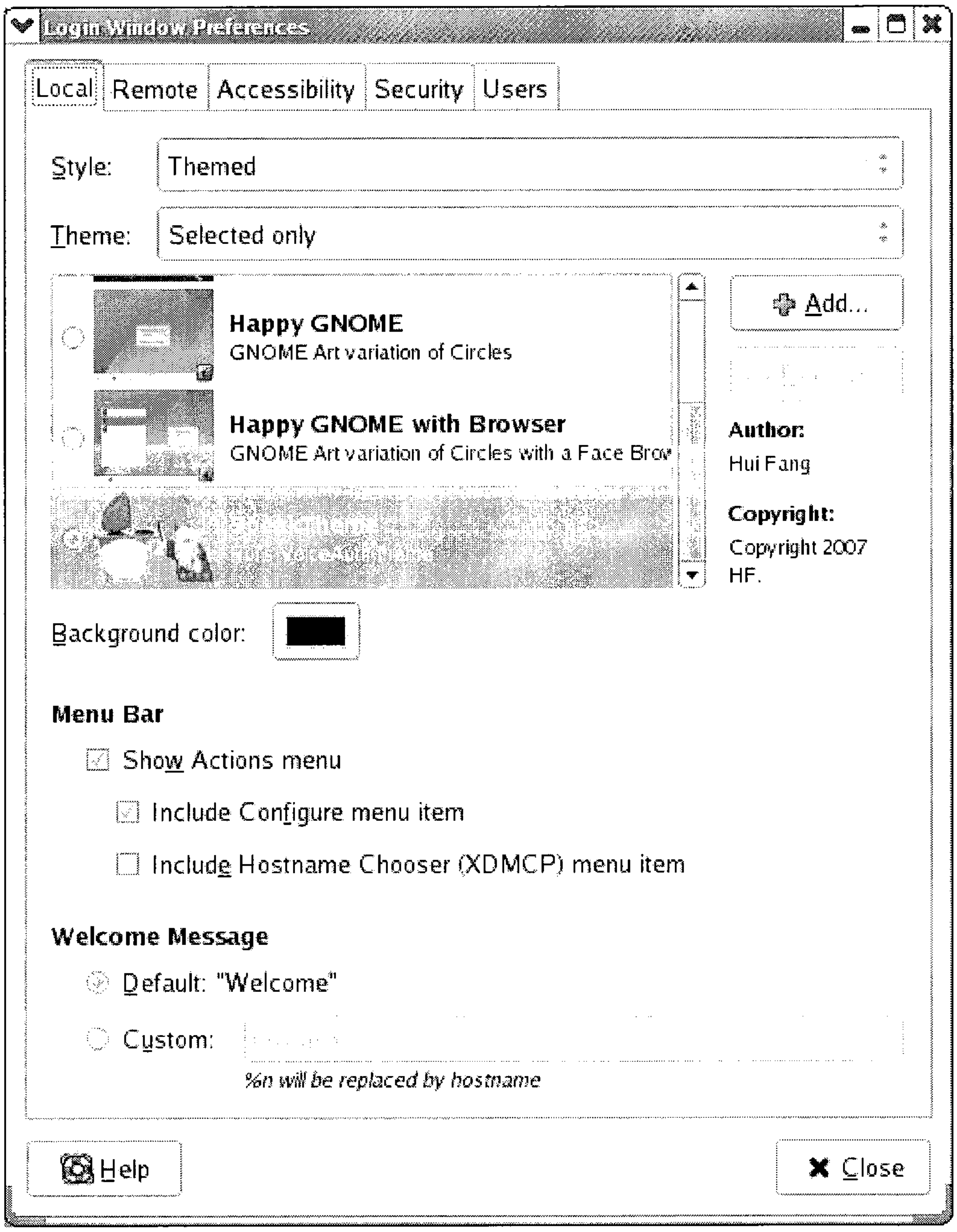
The Authentication Module is a separate program called pam_hui_voice. This program identifies a person using the voiceprint database created by Training Module. The user says the passphrase "Hello". The system then processes the input using the same technique developed in the Training Module. Next the system compares the voice sample with the stored voiceprint to identify the person. Based on the verification, the system either grants access or reject the request.

In order to authenticate a user from the login screen, the Fedora operating system was modified with a customized display theme. This is done by modifying a currently available theme and uploading it to the system. GDM Themes can be created by using an XML file that follows the specification in the Fedora system path: gui/greeter/greeter.dtd. Theme files are stored in the directory <share>/gdm/themes/<theme_name>. Usually this would be under /usr/share. The theme directory should contain a file called GdmGreeterTheme.desktop which has a similar format to other .desktop files and looks like the example below:

```
[GdmGreeterTheme]
Greeter=SpeakTheme.xml
Name=SpeakTheme
_Description=Hui's VoiceAuth theme
_Author=Hui Fang
Copyright=Copyright 2007
Screenshot=screenshot.png
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The Name, Description, Author and Copyright fields can be modified just like the other .desktop files. (GDM, 2004) Figure 4.20 illustrates the Login Window Preferences window in which you can select the customized theme that has been uploaded to the system. In this figure, the SpeakTheme is the customized theme that has been selected.



Style: Themed

Theme: Selected only

Happy GNOME
GNOME Art variation of Circles

Happy GNOME with Browser
GNOME Art variation of Circles with a Face Brow

[illegible]

Author:
Hui Fang

Copyright:
Copyright 2007
HF.

Background color: ■

Menu Bar

☑ Show Actions menu

☑ Include Configure menu item

☐ Include Hostname Chooser (XDMCP) menu item

Welcome Message

◉ Default: "Welcome"

○ Custom:

%n will be replaced by hostname

Figure 4.20. Login Window Preferences.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The Authentication Module of VALIS is an implementation of the 'login' program, which uses the PAM conversation methods to authenticate a user. To let VALIS act as the default login program, a simple modification on `/etc/pam.d` can be made, as shown below:

```
#%PAM-1.0
auth        required      pam_hui_voice.so
account      required     pam_nologin.so
account      include      system-auth
session      include          system-auth
session      required     pam_loginuid.so
session      optional         pam_console.so
```

Figure 4.21. PAM (Code).

Each non-commented line of a PAM configuration file consists of four possible arguments: a *module interface*, *control flag*, *module path*, and *module arguments*. Each line has the structure:

interface   control_flag   module_path   [module_arguments].

An example line is shown as:

```
auth          required              pam_hui_voice.so
```

where `auth` is interface, `required` is control flag, `pam_hui_voice.so` is module path. There are no module arguments in this example.

A *module interface* is the PAM jargon for the type of authorization a module performs. A PAM module may utilize one or all of four possible interfaces. These interfaces are *account, auth, password*, and *session.* Each interface will be specified in the configuration file for a service if it's appropriate for the module to use that interface. This program utilizes the *auth* interface for authentication purposes.

For each interface, the configuration file specifies a *control flag* which determines what PAM does next based upon the result of the authentication. There are four control flags which are `required`, `requisite`, `sufficient`, and `optional`. Control flags give the ability to set the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

importance of a module in respect to the modules that follow it. In VALIS, `required` is the only one used.

The *module path* tells PAM the location of the module. It is normally a full pathname. If no path is specified, PAM defaults to `/lib/security` to find the module. As shown in Figure 4.21, the module path is `/lib/security/pam_hui_voice.so`.

As previously described, VALIS uses intelligence classifiers to compare the voice sample and the voiceprint by using the T-test and correlation measures. The T-test is used to compare different sample size arrays such as peaks. Correlation is used to compare same size arrays such as frequencies and the top six peaks. For accuracy, the top six peaks were compared twice, sorted and unsorted.

Figure 4.22 shows the code that calculates the T-test and degrees of freedom according to the formula shown in Figure 3.3 and 3.4.

```
T=(SPMean-RPMean)/(sqrt(((SPStd*SPStd)/Psize)+((RPStd*RPStd)/peaksize)));

DF=(((SPStd*SPStd)/Psize+(RPStd*RPStd)/peaksize)*((SPStd*SPStd)/Psize+
(RPStd*RPStd)/peaksize))/((((SPStd*SPStd)/Psize)*((SPStd*SPStd)/Psize)
/(Psize-1.0))+(((RPStd*RPStd)/peaksize)*(RPStd*RPStd)/peaksize)/(peaksize-
1.0));
```

Figure 4.22. Degrees of Freedom and T-test (Code).

Figure 4.23 shows the implementation of the correlation formula depicted in Figure 3.5 in the previous chapter.

```
for (i=0; i<size; i++){
      part1+=((x[i]-xb)*(y[i]-yb));
      part2+=((x[i]-xb)*(x[i]-xb));
      part3+=((y[i]-yb)*(y[i]-yb));
}

return (part1/sqrt(part2*part3));
```

Figure 4.23. Correlation (Code).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3 Voiceprint Reference File

The voiceprint reference file is a binary file which contains the following components: the size of the FFT array, the FFT array, the number of peaks, and the array of peaks. The reference file is named the same as the person's login name.

4.4 Decision Logic

To decide if a voice sample matches to a voice print, several voice features are considered. A final decision about the identity of the speaker is made by comparing unknown feature vectors to all models in the database and selecting the best matching model. VALIS compares the sample file to the reference file and stores the result into a table. The table contains the T-test results, the number of matching peaks, the correlation of FFT, and the correlation of peaks, etc. VALIS first considers the T-test results. If a T-test result of "yes" is returned, the login process will continue. Next, the correlation of the FFT results is checked. The highest correlation result located will be identified as the most likely candidate. If the highest result is lower than the correlation threshold (currently set to 85%), the program will return "nobody". If the highest result is higher than the correlation threshold, the program double checks the candidate with the correlation of peaks, the highest peak and time duration of the passphrase. If all of these match, VALIS returns the candidate's username and will login as this user automatically. This process is depicted in Figure 4.24.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

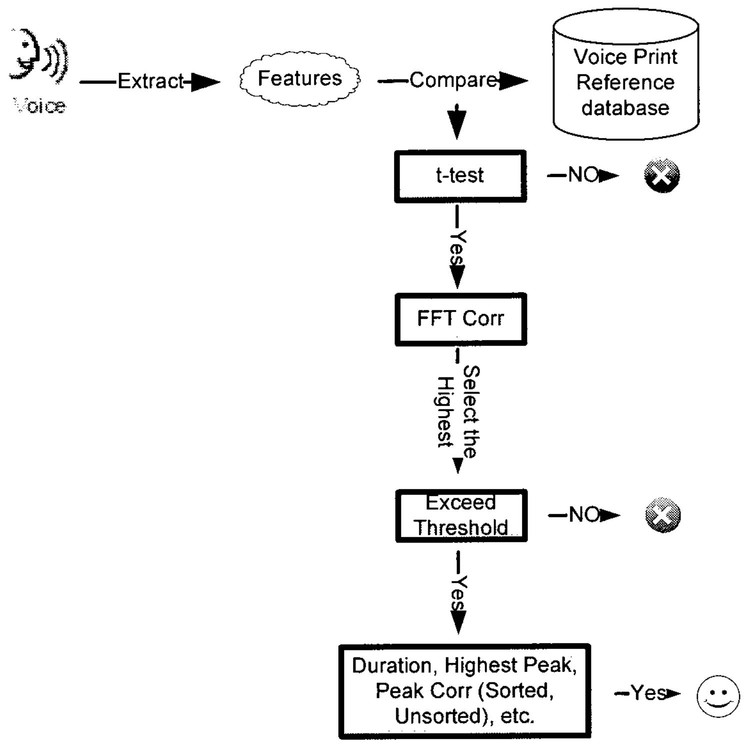

Voice
Extract
Features
Compare
Voice Print Reference database
t-test
NO
Yes
FFT Corr
Select the Highest
Exceed Threshold
NO
Yes
Duration, Highest Peak, Peak Corr (Sorted, Unsorted), etc.
Yes

Figure 4.24. Decision Logic.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 5 TEST RESULTS

To ensure consistency of background noise, testing was performed in the same office within the Computer Science Department at Southern Connecticut State University. A total of twelve people (6 men and 6 women) participated in the testing and were composed of both students and faculty members. All of the voice samples were collected using the same microphone in the same room. After a brief training, every speaker recorded seven samples. Each sample was approximately two seconds in length using the phrase "Hello". Three samples of each speaker were used to create the voice print reference file. The remaining four samples of each speaker were used for testing authentication.

The results of the voice samples and reference file comparison are shown in tables 5.1, 5.2, 5.3, and 5.4. The first row of each table is the comparison of the T-test result, the second row is the comparison of correlation of FFTs, and the third row of the table is the comparison of correlation of the first 6 peaks.

Table 5.1

Male Speaker 1 Identified As Male Speaker 1

	m1	**m2**	**m3**	**m4**	**m5**	**m6**	**f1**	**f2**	**f3**	**f4**	**f5**	**f6**
T-test	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	No	Yes	Yes
FFTC	0.9715	0.7966	0.6569	0.7896	0.6536	0.7435	0.644	0.6583	0.6294	0.4006	0.6468	0.6628
PeakC	0.9957	0.9344	0.955	0.9899	0.954	0.955	0.9728	0.9906	0.9731	0.9518	0.9198	0.9877

Table 5.1 shows results for male speaker 1 (m1). Row 1 shows that male 4, male 6 and female 4 are out of consideration due to failure of the T-test result. Rows 2 and 3 show that the highest correlation of FFTs and peaks out of the remaining seven speakers is male 1. The

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

program correctly identified male 1 as the actual speaker based on the data contained in Table 5.1.

Table 5.2

Male Speaker 2 Identified As Male Speaker 2

	m1	m2	m3	m4	m5	m6	f1	f2	f3	f4	f5	f6
T-test	No	Yes	No	Yes	No	Yes	Yes	Yes	No	Yes	Yes	No
FFTC	0.7126	0.9681	0.7459	0.8007	0.8998	0.7567	0.6744	0.8057	0.7375	0.6409	0.7979	0.6726
PeakC	0.9262	0.9946	0.9555	0.9758	0.9993	0.9992	0.9959	0.9374	0.9671	0.9954	0.9942	0.9679

Table 5.2 shows data for male speaker 2 (m2). As can be seen, males 1, 3, 5 and females 3, 6 are eliminated due to the T-test result. Rows 2 and 3 show that the highest correlation of FFTs and peaks out of the remaining 5 speakers is male 2. Male 5 has the second highest comparison on correlation of FFTs and peaks. This demonstrates that the voice of two speakers have similar biometric features. Two people could sound similar however the program was able to determine the difference and correctly identified the speaker as speaker m2.

Table 5.3

Female Speaker 1 Identified As Female Speaker 3

	m1	m2	m3	m4	m5	m6	f1	f2	f3	F4	f5	F6
T-test	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
FFTC	0.5215	0.6378	0.4944	0.5171	0.5211	0.3713	0.8195	0.6317	0.8646	0.4549	0.7083	0.6512
PeakC	0.9814	0.9435	0.9324	0.9797	0.9466	0.9487	0.9724	0.9886	0.9892	0.9418	0.9278	0.9823

Table 5.3 shows the data for female speaker 1 (f1). Based on the results in Table 5.3, the program incorrectly identified the speaker as female 3. It is possible that two individuals with similar vocal patterns may produce voice samples with high correlation especially when the samples were collect in the same room. In this case, female 1 has a closer match to female 3's reference file than to female 1's reference file. This result demonstrates the possibility of failure of the program.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 5.4

Female Speaker 4 Identified As Nobody

	m1	m2	m3	m4	m5	m6	F1	f2	f3	F4	f5	F6
T-test	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
FFTC	0.5216	0.7596	0.7212	0.7601	0.7656	0.6443	0.5049	0.5405	0.568	0.7753	0.6772	0.6958
PeakC	0.9474	0.9684	0.982	0.9822	0.9901	0.9881	0.9864	0.9495	0.9555	0.9876	0.9706	0.9823

Table 5.4 shows the data for female speaker 4 (f4). In Table 5.4, male 6 was eliminated due to the T-test result. This was inconsequential due to the fact that the highest correlation of FFTs is 0.77 which was below the minimum threshold. In this case the highest correlation belongs to the correct speaker but produced a rejection due to the threshold limitation of 0.85. In the future this error can be remedied with the addition of improved noise filtering. In this case, the program returns "nobody" as identified.

Table 5.5

Summary of FFT Correlation

	m1	m3	m4	m5	m6	F1	f2	f3	f5	f6
m1_1	0.938966	0.747004	0.787333	0.690893	0.779876	0.617975	0.570147	0.631371	0.691903	0.484179
m1_2	0.929125	0.753369	0.777669	0.722587	0.813272	0.598171	0.637784	0.637256	0.667413	0.454239
m1_3	0.971532	0.656902	0.789561	0.653627	0.796583	0.644047	0.658251	0.629387	0.646823	0.400645
m1_4	0.891584	0.696801	0.791019	0.667362	0.771057	0.601638	0.56912	0.596608	0.676424	0.454304
m3_1	0.624506	0.95021	0.642023	0.810566	0.735597	0.534504	0.527584	0.566433	0.701054	0.563135
m3_2	0.58476	0.945254	0.588801	0.825815	0.734949	0.513725	0.481768	0.586345	0.697	0.523629
m3_3	0.568322	0.926368	0.620528	0.781749	0.702655	0.543075	0.479772	0.532391	0.68585	0.480247
m3_4	0.548287	0.872033	0.497786	0.63714	0.568891	0.31845	0.26369	0.383964	0.531156	0.379644
m4_1	0.723929	0.621077	0.955233	0.762286	0.803079	0.641064	0.622236	0.608004	0.65925	0.54201
m4_2	0.779069	0.682755	0.963196	0.811833	0.840037	0.64519	0.643654	0.620825	0.677598	0.574708
m4_3	0.750003	0.680553	0.918272	0.82934	0.845059	0.646032	0.669224	0.681005	0.70122	0.587552
m4_4	0.700813	0.703849	0.902005	0.844173	0.847556	0.633342	0.652639	0.624458	0.709442	0.576337
m5_1	0.659848	0.840341	0.772344	0.961219	0.886499	0.603518	0.634512	0.704254	0.73642	0.614108
m5_2	0.641479	0.852033	0.769493	0.968956	0.893243	0.575286	0.620733	0.691899	0.753681	0.637908
m5_3	0.631907	0.801891	0.792974	0.974686	0.899551	0.585727	0.650107	0.709559	0.739626	0.727232
m5_4	0.636945	0.818973	0.734917	0.943591	0.843807	0.495633	0.570531	0.617321	0.642421	0.593621
m6_1	0.785273	0.752962	0.827994	0.889269	0.924402	0.667457	0.759454	0.730365	0.766634	0.614447
m6_2	0.761825	0.77128	0.795071	0.869198	0.961538	0.711538	0.796751	0.758082	0.807575	0.626177

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

m6_3	0.71255	0.745919	0.800658	0.899755	0.968126	0.67437	0.805692	0.737524	0.797943	0.640945
m6_4	0.752923	0.734837	0.799244	0.882961	0.959182	0.641983	0.766432	0.71908	0.7305	0.589501
f1_1	0.600342	0.639983	0.729774	0.658344	0.739492	0.912484	0.677413	0.746161	0.877029	0.468839
f1_2	0.585939	0.598894	0.646643	0.57148	0.695392	0.896997	0.594658	0.756061	0.819759	0.363037
f1_3	0.579631	0.527493	0.550278	0.558049	0.693169	0.910839	0.668676	0.815603	0.735748	0.430547
f1_4	0.521511	0.49444	0.517108	0.521128	0.637772	0.819474	0.631731	0.864617	0.708257	0.45492
f2_1	0.720103	0.66764	0.763469	0.739431	0.855381	0.763908	0.965634	0.727143	0.783429	0.597054
f2_2	0.626339	0.512427	0.627419	0.62808	0.783734	0.710623	0.9759	0.705909	0.688019	0.552369
f2_3	0.605691	0.583866	0.629241	0.636857	0.774867	0.697133	0.961377	0.66978	0.710136	0.519917
f2_4	0.441263	0.467301	0.459219	0.542059	0.657751	0.538862	0.833177	0.506319	0.511526	0.465742
f5_1	0.60676	0.669101	0.56109	0.664328	0.78135	0.833304	0.690935	0.85965	0.929096	0.499299
f5_2	0.618188	0.757562	0.675398	0.783046	0.820137	0.801688	0.70943	0.833117	0.952847	0.578193
f5_3	0.604906	0.7437	0.652686	0.718879	0.794336	0.795386	0.618427	0.835889	0.962764	0.498696
f5_4	0.598977	0.710695	0.653674	0.71596	0.793958	0.827502	0.700355	0.85271	0.951549	0.555227
f6_1	0.53463	0.674505	0.713739	0.742757	0.726612	0.537489	0.58425	0.574022	0.645427	0.907185
f6_2	0.521565	0.721201	0.760139	0.785645	0.759606	0.504873	0.540521	0.568043	0.677162	0.775319
f6_3	0.285987	0.431188	0.413166	0.603455	0.566788	0.358724	0.473425	0.482876	0.472623	0.8969
f6_4	0.320953	0.561133	0.451525	0.660326	0.616151	0.398774	0.452359	0.496347	0.526651	0.929007

Table 5.5 is a summarized correlation of FFT comparison results between 10 speakers generated from the VALIS. Each of the speakers repeated the specific word "Hello" several times, and the results are given in this table. Base on the testing, 3 out of 36 tests experienced failure. This result produced an approximate 8 percent failure rate. The highest correlation result of each row is highlighted in extra thick border. Grey background denotes the highest correlation result which did not exceed the minimum threshold.

Throughout the voice identification project development, the premise that false acceptance is much worse than false rejection was a key design goal. A false rejection error occurs when a valid identity claim is rejected. A false acceptance error consists of accepting an identity claim from an impostor. Both types of errors depend on the threshold used in the decision making process. With a low threshold, the system tends to accept every identity claim thus making few false rejections and many false acceptances. The test experienced mixed

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

success with various threshold levels. Figure 5.1 shows the false rejection rate and false acceptance rate with threshold level varying from 75% to 97%.

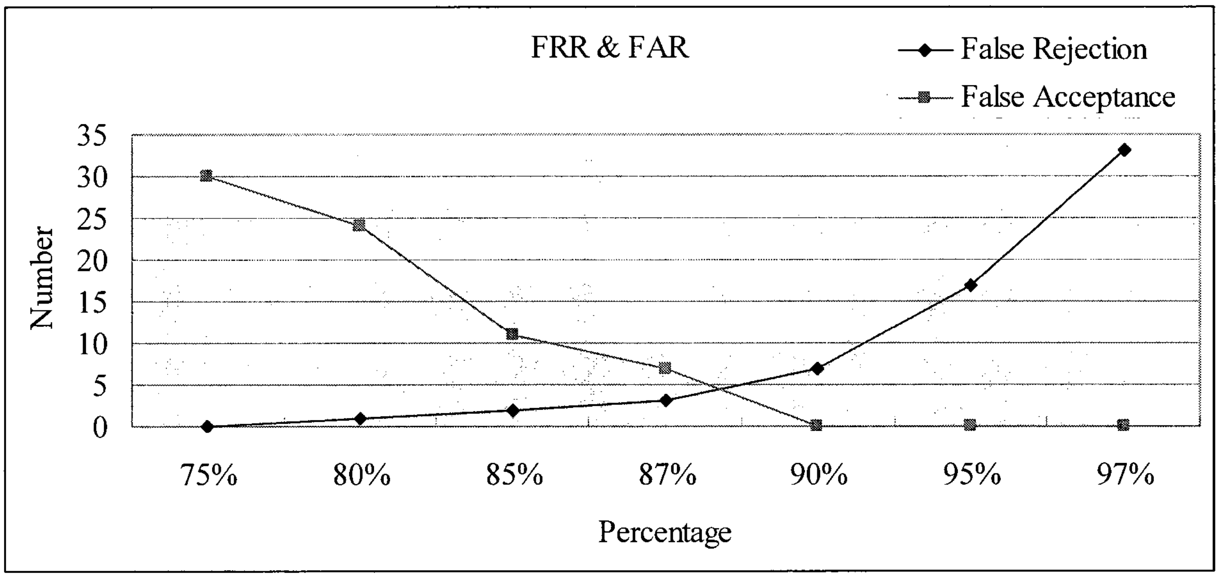


Figure 5.1. False Rejection Rate and False Acceptance Rate.

This data shows a satisfactory performance result due to the low rate of rejections and false positives. Future work on filtering and algorithms would most certainly improve performance.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 6 CONCLUSION AND FUTURE WORK

The main goal of this project was to provide an alternative method of computer user authentication. Voice authentication provides a number of benefits compared to the traditional way of username and password authentication. Some of the benefits include a more natural way of gaining computer access, no need for regular typed password changes, and spoken passphrases will not be lost, stolen or forgotten. The system also eliminates the need for keyboard entries of sensitive access credentials. Spoken passphrases are also very desirable in a situation where mobility is of the utmost importance and keyboards can not be used. As mobile-commerce applications and web-based portals become more prevalent, the voice authentication application may be used in online applications.

VALIS enables the user to add Voice Authentication to Linux systems. It is an Open Source, pluggable, easy-to-use application with the flexibility to add or remove features.

The performance of this system was tested in a single location with 12 users, 6 females and 6 males. Based on the results, VALIS performed well with satisfactory results as expected. The error rate encountered was within an acceptable range. These errors might be attributed to such factors as environmental noise and test subject variability. The question remains whether the rate of these errors would increase with the increasing use of the system. This should not be a problem due to the fact that VALIS was designed to authenticate users to individual Linux based computers. It is not designed as an enterprise level application with a great number of users.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In the future, more features could be added to the program to enhance its performance and usability. Because the most important features of a successful Voice authentication system is the accuracy and reliability, after a speaker is successfully authenticated, the voice sample could be added to the voice print reference file to strengthen the voice print. This would increase the accuracy of the user authentication as the system continues to be used.

Another area for future work could consist of the use of additional algorithms. These might include the Hidden Markov Model (HMM) or Vector Quantization (VQ) algorithms, utilizing such algorithms to either replace the current methods or to add onto VALIS would an additional level of decision making logics that could be explored.

One drawback in biometric authentication is the inevitable changes inherent in the human condition. Aging, health changes and physical damage will occur throughout a person's life time. These factors may alter an individual's physical characteristics upon which biometric authentication is based. These factors will require the updating of reference data and possibly the use of multiple authentication techniques.

Additionally, the testing phase and processing phase can both be automated to help turn this program into more of a marketable product. Presently all the repetitive collection, multi-step processing and data visualization is performed using multiple manual steps. VALIS could be improved to automatically to move the data through each one of these steps, ensuring the desired result. Human error could also be eliminated.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX A: SOURCE CODE

1. myC.c

```
//Correlation

#include <math.h>
#include <stdlib.h>

double myC(double *x, double *y, int size)
{
  int i;
  double xb=0;
  double yb=0;
  double r=0;
  double part1, part2, part3;

  for (i=0; i<size; i++){
    xb+=x[i];
    yb+=y[i];
  }

  xb=xb/size;
  yb=yb/size;

  part1=part2=part3=0;

  for (i=0; i<size; i++){
    part1+=((x[i]-xb)*(y[i]-yb));
    part2+=((x[i]-xb)*(x[i]-xb));
    part3+=((y[i]-yb)*(y[i]-yb));
  }

  return (part1/sqrt(part2*part3));

}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. myquad.c

```
#include <math.h>
#include <stdlib.h>

#ifndef M_LN2
#define M_LN2     0.69314718055994530942
#endif

#ifndef M_PI
#define M_PI      3.14159265358979323846
#endif

/* this holds the data required to update samples thru a filter */
typedef struct {
   double a0, a1, a2, a3, a4;
   double x1, x2, y1, y2;
}
biquad;

extern double BiQuad(double sample, biquad * b);
extern biquad *BiQuad_new(int type, double dbGain, /* gain of filter */
                      double freq,           /* center frequency */
                      double srate,          /* sampling rate */
                      double bandwidth);      /* bandwidth in octaves */

/* filter types */
enum {
   LPF, /* low pass filter */
   HPF, /* High pass filter */
   BPF, /* band pass filter */
   NOTCH, /* Notch Filter */
   PEQ, /* Peaking band EQ filter */
   LSH, /* Low shelf filter */
   HSH /* High shelf filter */
};

/*  Computes a BiQuad filter on a sample */
double BiQuad(double sample, biquad * b)
{
   double result;

   /* compute result */
   result = b->a0 * sample + b->a1 * b->x1 + b->a2 * b->x2 -
      b->a3 * b->y1 - b->a4 * b->y2;

   /* shift x1 to x2, sample to x1 */
   b->x2 = b->x1;
   b->x1 = sample;

   /* shift y1 to y2, result to y1 */
   b->y2 = b->y1;
   b->y1 = result;

   return result;
}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
/* sets up a BiQuad Filter */
biquad *BiQuad_new(int type, double dbGain, double freq, double srate, double
bandwidth)
{
    biquad *b;
    double A, omega, sn, cs, alpha, beta;
    double a0, a1, a2, b0, b1, b2;

    b = (biquad *)(malloc(sizeof(biquad)));
    if (b == NULL)
        return NULL;

    /* setup variables */
    A = pow(10, dbGain /40);
    omega = 2 * M_PI * freq /srate;
    sn = sin(omega);
    cs = cos(omega);
    alpha = sn * sinh(M_LN2 /2 * bandwidth * omega /sn);
    beta = sqrt(A + A);

    switch (type) {
    case LPF:
        b0 = (1 - cs) /2;
        b1 = 1 - cs;
        b2 = (1 - cs) /2;
        a0 = 1 + alpha;
        a1 = -2 * cs;
        a2 = 1 - alpha;
        break;
    case HPF:
        b0 = (1 + cs) /2;
        b1 = -(1 + cs);
        b2 = (1 + cs) /2;
        a0 = 1 + alpha;
        a1 = -2 * cs;
        a2 = 1 - alpha;
        break;
    case BPF:
        b0 = alpha;
        b1 = 0;
        b2 = -alpha;
        a0 = 1 + alpha;
        a1 = -2 * cs;
        a2 = 1 - alpha;
        break;
    case NOTCH:
        b0 = 1;
        b1 = -2 * cs;
        b2 = 1;
        a0 = 1 + alpha;
        a1 = -2 * cs;
        a2 = 1 - alpha;
        break;
    case PEQ:
        b0 = 1 + (alpha * A);
        b1 = -2 * cs;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
        b2 = 1 - (alpha * A);
        a0 = 1 + (alpha /A);
        a1 = -2 * cs;
        a2 = 1 - (alpha /A);
        break;
    case LSH:
        b0 = A * ((A + 1) - (A - 1) * cs + beta * sn);
        b1 = 2 * A * ((A - 1) - (A + 1) * cs);
        b2 = A * ((A + 1) - (A - 1) * cs - beta * sn);
        a0 = (A + 1) + (A - 1) * cs + beta * sn;
        a1 = -2 * ((A - 1) + (A + 1) * cs);
        a2 = (A + 1) + (A - 1) * cs - beta * sn;
        break;
    case HSH:
        b0 = A * ((A + 1) + (A - 1) * cs + beta * sn);
        b1 = -2 * A * ((A - 1) + (A + 1) * cs);
        b2 = A * ((A + 1) + (A - 1) * cs - beta * sn);
        a0 = (A + 1) - (A - 1) * cs + beta * sn;
        a1 = 2 * ((A - 1) - (A + 1) * cs);
        a2 = (A + 1) - (A - 1) * cs - beta * sn;
        break;
    default:
        free(b);
        return NULL;
    }

    /* precompute the coefficients */
    b->a0 = b0 /a0;
    b->a1 = b1 /a0;
    b->a2 = b2 /a0;
    b->a3 = a1 /a0;
    b->a4 = a2 /a0;

    /* zero initial samples */
    b->x1 = b->x2 = 0;
    b->y1 = b->y2 = 0;

    return b;
}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3. smooth.c

```
#include <math.h>
#include <stdlib.h>

void smooth3(double * sample,int size)
{
  int i;

  double tmp[2];

  if(size>=3){
    tmp[0] = sample[0];
    for(i=1;i<size-1;i++){
      tmp[1] = sample[i];
      sample[i] = (tmp[0] + tmp[1] + sample[i+1])  / 3.0;
      tmp[0] = tmp[1];
    }
  }
}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4. p7.h

```
#include "/root/kde/p7/p7Iface.h"
#include <kmainwindow.h>
#include <qcanvas.h>
#include <qpainter.h>
#include <kurl.h>
#include <kparts/browserextension.h>
#include <qvaluestack.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>


#define WISDOM "triwis.txt"
#define XYZ 0
#define MAXIS 10 //number of peaks
#define SMOOTHIE 100 //smooth times  was 250
#define COMPRESS 10 //compression for FFT
#define R 5 //radius around the peak
#define f(a) (cons*((double)(a)))
#define dB(a) (10.0 * log10((a)/(   (double)(half) * (double)(half)   )))
#define N 6  //number of top peaks

class QLineEdit;
class KHTMLPart;
class MainWindow;

class MainWindow : public KMainWindow, virtual public p7Iface
{

  Q_OBJECT
  public:
    MainWindow ( const char * titulo );

    virtual void setURL ( const QString url );

  public slots:
    void fileSetDefaultPage();
    void changeLocation();
    void bookLocation();
    void gotoPreviousPage();
    void openURLRequest(const KURL &url, const KParts::URLArgs & );
    void readPCM();
    void playPCM();
    void readData();
    void readMe();
    void drawData();
    void findDuration();
    void myFFT();
    void chopFFT();
    void drawFFT();
    void zoomFFT();
    void saveFFT();
    void averageFFT();
    void borderMe();
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    void recordMe();
    void filterData();
    void smoothFFT();
    void findPeak();

  protected:
    void resizeEvent( QResizeEvent* e);
    void contentsMousePressEvent(QMouseEvent* e);
    void mousePressEvent(QMouseEvent* e);
    void mouseReleaseEvent(QMouseEvent* e);
    void mouseMoveEvent(QMouseEvent* e);

  private:
    QCanvas *mywin;
    QCanvasView *myView;
    QPainter *myPaint;
    QPainter *pp;
    short int *soundD, *soundDD, *soundDDD;
    int ii,cnt;
    off_t total;
    QPixmap *tt;
    int ww, wh,mwh;
    int x1,x2,y1,y2;
    double scaleA,scaleB, scaleC,scaleD;
    double *Xfft, *Yfft, *XfftAvg, *YfftAvg;
    int *indexP;
    long int Psize;
    long int fftmany;
    long int fftsize;
    long int half;
    unsigned int SRATE;//sample rate
    int flag;
    int trainingF;
    int color;
    bool louder;
    double Duration;
    double P[N]; //Peak
    double Distance;
    QValueStack <QString> history;
};
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5. p7.cpp

```
/**
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation.

 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY.  See the GNU General Public License for
 * more details.
 *
 **/


#include "p7.h"
#include <qvbox.h>
#include <qlineedit.h>
#include <dcopclient.h>
#include <kfiledialog.h>
#include <kapp.h>
#include <kmenubar.h>
#include <ktoolbar.h>
#include <klocale.h>
#include <kpopupmenu.h>
#include <khtml_part.h>
#include <kdebug.h>
#include <kconfig.h>
#include <kiconloader.h>
#include <alsa/asoundlib.h>
#include <sys/types.h>
#include <dirent.h>
#include <fftw3.h>

#include "myquad.c"
#include "smooth.c"
#include "myC.c"

#define ALSA_PCM_NEW_HW_PARAMS_API
#define TOOLBAR_ID_ADDBOOKMARK 1
#define TOOLBAR_ID_BACK 2
#define TOOLBAR_ID_QUIT 3

MainWindow::MainWindow (const char * name) : KMainWindow (0L, name),
DCOPObject ("browser")
{
  setCaption("Training");

  resize(800, 400);
  QPopupMenu * filemenu = new QPopupMenu;
  filemenu->insertItem( i18n( "&Quit" ), kapp, SLOT( quit() ) );
  QString about = i18n("Hui Fang\n\n" "Training Program\n");
  QPopupMenu *helpmenu = helpMenu(about);
  KMenuBar * menu = menuBar();
  menu->insertItem(i18n( "&File" ), filemenu);
  menu->insertSeparator();
  menu->insertItem(i18n( "&Help" ), helpmenu);
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  KToolBar *toolbar=new KToolBar(this);

  toolbar->insertButton(UserIcon("/root/kde/p7/pics/record.gif"), 0,
     SIGNAL(clicked(int)),this,SLOT(recordMe()),TRUE, i18n("Record Voice"));

  toolbar->insertButton(UserIcon("/root/kde/p7/pics/play.gif"), 0,
     SIGNAL(clicked(int)),this,SLOT(playPCM()),TRUE, i18n("Play Voice"));

  toolbar->insertButton(UserIcon("/root/kde/p7/pics/showGraph.gif"), 0,
     SIGNAL(clicked(int)),this,SLOT(readMe()),TRUE, i18n("Read File"));

  toolbar->insertButton(UserIcon("/root/kde/p7/pics/save.gif"), 0,
     SIGNAL(clicked(int)),this,SLOT(saveFFT()),TRUE, i18n("Save File"));

  toolbar->insertButton(UserIcon("/root/kde/p7/pics/filter.gif"), 0,
     SIGNAL(clicked(int)),this,SLOT(filterData()),TRUE, i18n("Bandpass
Filter"));

  toolbar->insertButton(UserIcon("/root/kde/p7/pics/smooth.gif"), 0,
     SIGNAL(clicked(int)),this,SLOT(smoothFFT()),TRUE, i18n("Smoothing
Filter"));

  toolbar->insertButton(UserIcon("/root/kde/p7/pics/showFFT.gif"), 0,
     SIGNAL(clicked(int)),this,SLOT(myFFT()),TRUE, i18n("FFT"));

  toolbar->insertButton(UserIcon("/root/kde/p7/pics/findMe.jpg"), 0,
     SIGNAL(clicked(int)),this,SLOT(findPeak()),TRUE, i18n("FindMe"));

  toolbar->insertButton(BarIcon("exit"), TOOLBAR_ID_QUIT,
     SIGNAL(clicked(int)),kapp,SLOT(quit()),TRUE, i18n("Quit the
application"));

  addToolBar(toolbar);

  myView = new QCanvasView( this );
  QSize mySize;
  mySize = maximumSize();
  ww = width()-2;
  mwh = menu->height();
  wh = height()-mwh-40;
  tt = new QPixmap(ww,wh);
  pp = new QPainter(tt);    // painter
  pp->setPen(white);
  pp->setBrush(white);
  pp->drawRect(0,0,ww,wh);
  pp->end();

  mywin = new QCanvas(*tt,1,1,ww,wh);
  mywin->setBackgroundPixmap( *tt );
  myView->setCanvas(mywin);

  KConfig *config=kapp->config();
  config->setGroup("Settings");
  setCentralWidget(myView);

  DCOPClient *client = kapp->dcopClient();
  client->attach();
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  client->registerAs("p7");
}

void MainWindow::recordMe()
{
  flag=3;
  readPCM();
  readData();
  drawData();
}

void MainWindow::readMe()
{
  readData();
  drawData();
}

void MainWindow::readPCM()
{
  long loops;
  int rc;
  int size;
  int myfile;
  int dir;
  char *buffer;

  snd_pcm_t *handle;
  snd_pcm_hw_params_t *params;
  snd_pcm_uframes_t frames;
  /* Open PCM device for recording (capture). */
  rc = snd_pcm_open(&handle, "default", SND_PCM_STREAM_CAPTURE, 0);
  if (rc < 0) {
    fprintf(stderr, "unable to open pcm device: %s\n", snd_strerror(rc));
    exit(1);
  }
  /* Allocate a hardware parameters object. */
  snd_pcm_hw_params_alloca(&params);
  /* Fill it in with default values. */
  snd_pcm_hw_params_any(handle, params);
  /* Set the desired hardware parameters. */
  /* Interleaved mode */
  snd_pcm_hw_params_set_access(handle, params, SND_PCM_ACCESS_RW_INTERLEAVED);
  /* Signed 16-bit little-endian format */
  snd_pcm_hw_params_set_format(handle, params, SND_PCM_FORMAT_S16_LE);
  /* One channels (mono) */
  snd_pcm_hw_params_set_channels(handle, params, 1);

  /* 44100 bits/second sampling rate (CD quality) */
  SRATE = 44100;
  snd_pcm_hw_params_set_rate_near(handle, params, &SRATE, &dir);
  /* Set period size to 32 frames. */
  frames = 32;
  snd_pcm_hw_params_set_period_size_near(handle, params, &frames, &dir);
  /* Write the parameters to the driver */
  rc = snd_pcm_hw_params(handle, params);
  if (rc < 0) {
    fprintf(stderr, "unable to set hw parameters: %s\n", snd_strerror(rc));
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    exit(1);
  }
  /* Use a buffer large enough to hold one period */
  snd_pcm_hw_params_get_period_size(params, &frames, &dir);
  //size = frames * 4; /* 2 bytes/sample, 2 channels */
  size = frames * 2; /* 2bytes/sample, 1 channels */
  buffer = (char *) malloc(size);
  /* We want to loop for 2 seconds */
  snd_pcm_hw_params_get_period_time(params, &SRATE, &dir);
  loops = 2000000 / SRATE;
  myfile = open("saveme",O_WRONLY | O_CREAT);

  while (loops > 0) {
    loops--;
    rc = snd_pcm_readi(handle, buffer, frames);
    if (rc == -EPIPE) {
      /* EPIPE means overrun */
      fprintf(stderr, "overrun occurred\n");
      snd_pcm_prepare(handle);
    } else if (rc < 0) {
      fprintf(stderr, "error from read: %s\n", snd_strerror(rc));
    } else if (rc != (int)frames) {
      fprintf(stderr, "short read, read %d frames\n", rc);
    }
    rc = write(myfile, buffer, size);
    if (rc != size)
      fprintf(stderr, "short write: wrote %d bytes\n", rc);
  }
  snd_pcm_drain(handle);
  snd_pcm_close(handle);
  free(buffer);
}

void MainWindow::playPCM()
{
  long loops;
  int rc;
  int size;
  int dir;
  int myfile;
  char *buffer;

  snd_pcm_t *handle;
  snd_pcm_hw_params_t *params;
  snd_pcm_uframes_t frames;
  /* Open PCM device for playback. */
  myfile = open("saveme",O_RDONLY);

  rc = snd_pcm_open(&handle, "default", SND_PCM_STREAM_PLAYBACK,0 );
  if (rc < 0) {
    fprintf(stderr, "unable to open pcm device: %s\n", snd_strerror(rc));
    exit(1);
  }
  /* Allocate a hardware parameters object. */
  snd_pcm_hw_params_alloca(&params);
  /* Fill it in with default values. */
  snd_pcm_hw_params_any(handle, params);
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  /* Set the desired hardware parameters. */
  /* Interleaved mode */
  snd_pcm_hw_params_set_access(handle, params, SND_PCM_ACCESS_RW_INTERLEAVED);
  /* Signed 16-bit little-endian format */
  snd_pcm_hw_params_set_format(handle, params, SND_PCM_FORMAT_S16_LE);
  /* One channels (mono) */
  snd_pcm_hw_params_set_channels(handle, params, 1);

  /* 44100 bits/second sampling rate (CD quality) */
  SRATE = 44100;
  snd_pcm_hw_params_set_rate_near(handle, params, &SRATE, &dir);
  /* Set period size to 32 frames. */
  frames = 32;
  snd_pcm_hw_params_set_period_size_near(handle, params, &frames, &dir);
  /* Write the parameters to the driver */
  rc = snd_pcm_hw_params(handle, params);
  if (rc < 0) {
    fprintf(stderr, "unable to set hw parameters: %s\n", snd_strerror(rc));
    exit(1);
  }
  /* Use a buffer large enough to hold one period */
  snd_pcm_hw_params_get_period_size(params, &frames, &dir);
  size = frames * 2; /* 2 bytes/sample, 1 channels */

  buffer = (char *) malloc(size);
  /* We want to loop for 2 seconds */
  snd_pcm_hw_params_get_period_time(params, &SRATE, &dir);
  /* 2 seconds in microseconds divided by * period time */
  loops = 2000000 / SRATE; //set to 2 second

  while (loops > 0) {
    loops--;
    rc = read(myfile, buffer, size);
    if (rc == 0) {
      fprintf(stderr, "end of file on input\n");
      break;
    } else if (rc != size) {
      fprintf(stderr, "short read: read %d bytes\n", rc);
    }
    rc = snd_pcm_writei(handle, buffer, frames);
    if (rc == -EPIPE) {
      /* EPIPE means underrun */
      fprintf(stderr, "underrun occurred\n");
      snd_pcm_prepare(handle);
    } else if (rc < 0) {
     fprintf(stderr, "error from write: %s\n", snd_strerror(rc));
    }  else if (rc != (int)frames) {
      fprintf(stderr, "short write, write %d frames\n", rc);
    }
  }
  snd_pcm_drain(handle);
  snd_pcm_close(handle);
  free(buffer);
}

void MainWindow::readData()
{
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  int rc,size,myfile;
  snd_pcm_uframes_t frames;
  char mybyte, yourbyte;
  struct stat fileData;
  short int ztmp,zztmp;
  short int max;
  int i;

  SRATE = 44100;
  frames = 32;
  size = frames * 2; /* 2 bytes/sample, 1 channels */
  myfile = open("saveme",O_RDONLY);
  rc = fstat(myfile, &fileData);
  if(rc == -1) return;
  total = fileData.st_size/2 ;

  if (soundDDD) free(soundDDD);{
    soundDDD = (short int*)(malloc(total*sizeof(short int)));}

  cnt=0;
  while (rc = read(myfile,&mybyte,1)) {
    if (rc == 0) { fprintf(stderr, "end of file on input\n"); break; }
    rc = read(myfile,&yourbyte,1);
    if (rc == 0) { fprintf(stderr, "end of file on input\n"); break; }

    //convert to byte
    ztmp=yourbyte;
    ztmp = ztmp << 8;
    zztmp = mybyte;
    zztmp = zztmp & 0xff;
    soundDDD[cnt++] = ztmp  + zztmp;
  }

  //chop first half second off (44100/2) sample rate fftsize
  if (soundDD) free(soundDD);
  soundDD=(short int *)(malloc(sizeof(short int)*(cnt-(SRATE/2))));

  for (i=(SRATE/2); i<cnt; i++)
    soundDD[i-(SRATE/2)]=soundDDD[i];

  cnt=cnt-(SRATE/2);

  //chop second half second off (44100/2) sample rate fftsize for noise
deduction
  if (soundD) free(soundD);
  soundD=(short int*)(malloc(sizeof(short int)*(cnt-(SRATE/2))));

  short int* noiseD;
  noiseD=(short int*)(malloc(sizeof(short int)*(SRATE/2)));

  for (i=0; i<(SRATE/2); i++)
    noiseD[i]=soundDD[i];
  for (i=(SRATE/2); i<cnt; i++)
    soundD[i-(SRATE/2)]=soundDD[i];

  cnt=cnt-(SRATE/2);
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  int n=cnt/(SRATE/2);
  int j;

  for (i=0; i<(SRATE/2); i++)
    for (j=1; j<=n; j++)
      soundD[i*j]=soundD[i*j]-noiseD[i];

  max=0;
  for (i=0; i<cnt; i++){
    if (soundD[i]>max) max=soundD[i];
  }
  printf ("max=%d\n", max);
  printf ("cnt=%d\n", cnt);

  if (max<1000){
    printf ("I can't hear you.  Please speak louder.\n");
    louder=1;
  }
}

void MainWindow::drawData()
{
  int i;
  char message[30];

  tt->resize(ww,wh);
  mywin->resize(ww,wh);
  mywin->setBackgroundPixmap( *tt );
  myView->setCanvas(mywin);

  pp->begin(tt);
  pp->setBrush(white);
  pp->drawRect(0,0,ww,wh);
  pp->setPen(green);

  int maxlimit;
  int nsnd = cnt / ww;
  int nrnd = cnt % ww;
  maxlimit = cnt - nrnd;

  if(cnt > 0)

    for(i=1; i<ww;i++){
      pp->drawLine(i, (int)( wh*0.5 - wh*0.5*  (soundD[nsnd*(i-1)] / 30000.00)),
i+1,
                      (int)( wh*0.5 - wh*0.5*  (soundD[nsnd*i] / 30000.00)) );
  }
  pp->end();

  mywin->setBackgroundPixmap( *tt );
  myView->setCanvas(mywin);
}

void MainWindow::findDuration()
{
  double Duration_start, Duration_end;
  double max = 0.0;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  int i;

  for (i=0; i<cnt; i++){
    if (soundD[i]>max)
      max = soundD[i];
  }

  i=0;
  while (soundD[i]<(max/2))
    i++;
  Duration_start=i;

  i=cnt;
  while (soundD[i]<(max/2))
    i--;
  Duration_end=i;

  Duration=Duration_end-Duration_start;
  if (Duration<0.0) Duration==0;
}


void MainWindow::filterData()
{
  biquad *myb;
  myb = BiQuad_new(BPF, 1, 261.626, 44100, 3);
  for (int a=0; a<cnt; a++)
    soundD[a] = BiQuad(soundD[a], myb);

  drawData();
}


void MainWindow::myFFT()
{
  long int i,j,startI, stopI;
  double cons, *in, *out;

  fftw_plan plan;
  FILE *wisdomptr;

  startI = 0;
  stopI= cnt -1;

  fftmany  = stopI/COMPRESS-(startI -1)/COMPRESS;

  in  = (double *) fftw_malloc( fftmany * sizeof(double));
  out = (double *) fftw_malloc( fftmany * sizeof(double));
  fftsize = fftmany/2+1;
  cons = 0.5 * SRATE /(double)(fftsize) / (double)(COMPRESS);

  i=0;
  j=0;
  while(i < fftmany){
    j++;
    if(j%COMPRESS == 0 && (j>= startI) && (j <= stopI) ){
      in[i++]=soundD[j-1];
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    }
  }

  /* get wisdom */

  if(wisdomptr = fopen(WISDOM,"r")){
    fftw_import_wisdom_from_file(wisdomptr);
  }
  plan = fftw_plan_r2r_1d(fftmany,in,out,FFTW_R2HC,FFTW_MEASURE);

  fftw_execute(plan);
  Xfft = (double *)(malloc(fftsize*sizeof(double)));
  Yfft = (double *)(malloc(fftsize*sizeof(double)));
  Xfft[0]= 0.0,
  Yfft[0]= dB(out[0]*out[0]);

  for( i=1;i< fftsize; ++i){
    Xfft[i]=f(i);
    Yfft[i]=dB(out[i]*out[i]+out[fftmany-i]*out[fftmany-i]);
  }

  if(fftmany%2 == 0){
    Xfft[fftmany/2]=f(fftmany/2);
    Yfft[fftmany/2]=dB(out[fftmany/2]*out[fftmany/2]);
  }

  fftw_destroy_plan(plan);

  /* save wisdom */
  wisdomptr = fopen(WISDOM,"w");
  fftw_export_wisdom_to_file(wisdomptr);
  fclose(wisdomptr);

  mywin->resize(ww,wh);

  chopFFT();
  drawFFT();
}

void MainWindow::chopFFT()
{
  int i;
  double TNTMIN, TNTMAX, NEWMIN;
  char message[30];
  tt->resize(ww,wh);
  mywin->resize(ww,wh);
  TNTMIN = 2000000000;
  TNTMAX = - TNTMIN;

  for(i=0;i<fftsize; ++i){
    if (Yfft[i]>TNTMAX) TNTMAX=Yfft[i];
    if (Yfft[i]<TNTMIN) TNTMIN=Yfft[i];
  }

  NEWMIN=TNTMAX-((TNTMAX-TNTMIN)*0.7);

  for (i=0; i<fftsize; i++){
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    if (Yfft[i]<NEWMIN) Yfft[i]=NEWMIN;
  }
}

void MainWindow::smoothFFT()
{
  for (int smt=0; smt<SMOOTHIE; smt++)
    smooth3(Yfft,fftsize);

  drawFFT();
  flag = 6;
}

void MainWindow::saveFFT()
{
  char *fileName=(char*)(malloc(100*sizeof(char)));
  strcpy(fileName, "/root/kde/p7/FFT/");
  fileName=strcat(fileName, getlogin());
  fileName=strcat(fileName, "fft.bin");
  FILE *pFile;
  pFile = fopen (fileName, "awb");
  fwrite ((&fftsize), sizeof(int), 1, pFile);
  fwrite (&Xfft[0], sizeof(double), fftsize, pFile);
  fwrite (&Yfft[0], sizeof(double), fftsize, pFile);
  fclose (pFile);
}

void MainWindow::drawFFT()
{
  int i;
  double TNTMIN, TNTMAX;
  char message[30];
  tt->resize(ww,wh);
  mywin->resize(ww,wh);
  TNTMIN = 2000000000;
  TNTMAX = - TNTMIN;

  for(i=0;i<fftsize; ++i){
    if (Yfft[i]>TNTMAX) TNTMAX=Yfft[i];
    if (Yfft[i]<TNTMIN) TNTMIN=Yfft[i];
  }

  pp->begin(tt);
  pp->setBrush(white);
  pp->drawRect(0,0,ww,wh);
  pp->setPen(green);

  int maxlimit;
  int pixelskip =200;
  int nsnd = cnt / ww;
  int nrnd = cnt % ww;
  maxlimit = cnt - nrnd;

  scaleA = wh/(TNTMIN - TNTMAX);
  scaleB = - scaleA * TNTMAX;
  scaleC = ww/(Xfft[fftsize-1] - Xfft[0]);
  scaleD = - scaleC * Xfft[0];
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  pixelskip = (pixelskip) /scaleC;

  if(cnt>0)
    for(i=1; i<fftsize;i++){
      pp->drawLine(scaleC*Xfft[i-1]+scaleD,scaleA * Yfft[i-1] + scaleB ,
                   scaleC*Xfft[i  ]+scaleD,scaleA * Yfft[i  ] + scaleB);

      if( i%pixelskip ==0 ){
        sprintf(message,"%5.2f",Xfft[i-1]) ;
        QString *alpha = new QString(message);
        pp->drawText(scaleC*Xfft[i-1]+scaleD - 20,
                     wh - 10,*alpha);
      }
    }
  pp->end();

  mywin->setBackgroundPixmap( *tt );
  flag = 2;
}

void MainWindow::zoomFFT()
{
  int i,zez, cx1,cx2,cy1,cy2;

  KMenuBar * menu = menuBar();
  mwh = menu->height();

  if(x1<x2) {cx1 = x1; cx2=x2;}
  else{cx1 = x2; cx2=x1;}

  if(y1<y2) {cy1 = y1; cy2=y2;}
  else{cy1 = y2; cy2=y1;}

  //adjust for the menu
  cy1 -= (mwh+42);
  cy2 -= (mwh+42);

  double TNTMIN, TNTMAX;
  double X1,X2;

  char message[30];
  tt->resize(ww,wh);
  mywin->resize(ww,wh);

  X1 = (cx1 - scaleD) / scaleC;
  X2 = (cx2 - scaleD) / scaleC;

  TNTMAX = (cy1 - scaleB) / scaleA;
  TNTMIN = (cy2 - scaleB) / scaleA;
  mywin->setBackgroundPixmap( *tt );
  myView->setCanvas(mywin);

  pp->begin(tt);
  pp->setBrush(white);
  pp->drawRect(0,0,ww,wh);
  pp->setPen(green);
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  int maxlimit;
  int pixelskip =200;

  int nsnd = cnt / ww;
  int nrnd = cnt % ww;
  maxlimit = cnt - nrnd;

  double div = (TNTMIN - TNTMAX);
  if(div == 0) div =1.0;
  scaleA = wh/div;
  scaleB = - scaleA * TNTMAX;

  div = (X2-X1);
  if(div == 0) div =1.0;

  scaleC = ww/(div);
  scaleD = - scaleC * X1;

  double pixperindex = scaleC*(Xfft[1]-Xfft[0]);
  int indexskip = ( pixelskip  / pixperindex) ;

  if(cnt>0)
    zez=1;

  int counter = 0;
  for(i=1; i<fftsize;i++){
    if(((X1<=Xfft[i-1]) && (Xfft[i-1] <= X2)) ||
       ((X1<=Xfft[i  ]) && (Xfft[i  ] <= X2))){
         counter++;
         pp->drawLine(scaleC*Xfft[i-1]+scaleD,scaleA * Yfft[i-1] + scaleB ,
                    scaleC*Xfft[i  ]+scaleD,scaleA * Yfft[i  ] + scaleB);
           if( ((indexskip!=0) && (zez%indexskip ==0)) ||  (indexskip<=5)  ){
                      zez++;
              sprintf(message,"%5.2f",Xfft[i-1]) ;
              QString *alpha = new QString(message);
              pp->drawText(scaleC*Xfft[i-1]+scaleD-20, wh -10, *alpha);
                 }
      } //if
  } //for

  pp->end();
  mywin->setBackgroundPixmap( *tt );
  myView->setCanvas(mywin);
  flag = 2;
}

void MainWindow::averageFFT()
{
  int fftsize1, fftsize2, fftsize3;
  double *YfftA, *YfftB, *YfftC;
  double *XfftA, *XfftB, *XfftC;
  XfftA=(double *)(malloc(fftsize*sizeof(double)));
  YfftA=(double *)(malloc(fftsize*sizeof(double)));
  XfftB=(double *)(malloc(fftsize*sizeof(double)));
  YfftB=(double *)(malloc(fftsize*sizeof(double)));
  XfftC=(double *)(malloc(fftsize*sizeof(double)));
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  YfftC=(double *)(malloc(fftsize*sizeof(double)));
  XfftAvg=(double *)(malloc(fftsize*sizeof(double)));
  YfftAvg=(double *)(malloc(fftsize*sizeof(double)));
  int counter=0;
  char *fileName=(char*)(malloc(100*sizeof(char)));
  strcpy(fileName, "/root/kde/p7/FFT/");
  fileName=strcat(fileName, getlogin());
  fileName=strcat(fileName, "fft.bin");
  FILE *pFile;
  pFile = fopen (fileName, "rb");
  if (pFile==NULL) {fputs ("File error",stderr); exit (1);}
  fread ((&fftsize1), sizeof(int), 1, pFile);
  fread (&XfftA[0], sizeof(double), fftsize1, pFile);
  fread (&YfftA[0], sizeof(double), fftsize1, pFile);
  counter++;
  fread ((&fftsize2), sizeof(int), 1, pFile);
  fread (&XfftB[0], sizeof(double), fftsize2, pFile);
  fread (&YfftB[0], sizeof(double), fftsize2, pFile);
  counter++;
  fread ((&fftsize3), sizeof(int), 1, pFile);
  fread (&XfftC[0], sizeof(double), fftsize3, pFile);
  fread (&YfftC[0], sizeof(double), fftsize3, pFile);
  counter++;
  fclose (pFile);

  for (int i =0; i<fftsize; i++){
    XfftAvg[i]=((XfftA[i]+XfftB[i]+XfftC[i])/counter);
    YfftAvg[i]=((YfftA[i]+YfftB[i]+YfftC[i])/counter);
  }
}

void MainWindow::findPeak()
{
  averageFFT();

  char *fileName=(char*)(malloc(100*sizeof(char)));
  FILE *pFile;

  double *YfftSmooth;
  YfftSmooth=(double *)(malloc(fftsize*sizeof(double)));

  int a, smt, i;
  for (a=0; a<fftsize; a++){
    YfftSmooth[a]=YfftAvg[a];
  }

  for (smt=0; smt<SMOOTHIE; smt++)
    smooth3(YfftSmooth, fftsize);

  double cons;
  cons = 0.5 * SRATE /(double)(fftsize) / (double)(COMPRESS);

  int i1, i2, Csize;
  i1= (int) (50/cons);
  i2= (int) (1100/cons);
  Csize=i2-i1;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  double *YPeak;
  YPeak=(double *)(malloc(10*sizeof(double)));
  indexP=(int *)(malloc(10*sizeof(int)));
  int capacity;
  Psize=0;
  capacity=10;

  for(a=(i1+2);a<(Csize-2); a++){
    if  ( (YfftSmooth[a]>YfftSmooth[a-1]) && (YfftSmooth[a]>YfftSmooth[a-2])
        && (YfftSmooth[a] > YfftSmooth[a+2]) &&  (YfftSmooth[a] >
YfftSmooth[a+1])){
      YPeak[Psize]=YfftSmooth[a];
      indexP[Psize]=a;
      Psize++;
      capacity--;
      if (capacity==0) {
        capacity=Psize;
        YPeak=(double *)realloc((double *)YPeak, Psize*2*sizeof(double));
        indexP=(int *)realloc((int *)indexP, Psize*2*sizeof(int));

        if (YPeak==NULL){
          printf ("memory problem");
          exit(1);
        }

        if (indexP==NULL){
          printf ("indexP memory problem");
          exit(1);
        }
      }
    }
  }

  int l, myIndex;
  double myMax;
  for (a=0; a<Psize; a++){
    myMax=YfftAvg[indexP[a]];
    myIndex=indexP[a];
    for (l=(indexP[a]-R); l<=(indexP[a]+R); l++){
      if (YfftAvg[l]>myMax){
        myMax=YfftAvg[l];
        myIndex=l;
      }
    }
    YPeak[a]=myMax;
    indexP[a]=myIndex;
  }

  int b, indexT;
  double *YPSorted, t;
  int *indexS;
  YPSorted=(double *)(malloc(Psize*sizeof(double)));
  indexS=(int *)(malloc(Psize*sizeof(int)));

  for (b=0; b<Psize; b++){
    YPSorted[b]=YPeak[b];
    indexS[b]=indexP[b];
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  }

  for (a=1; a < Psize; a++)
    for (b=Psize-1; b >= a; b--)
      if (YPSorted[b-1] < YPSorted[b]) {
        t=YPSorted[b-1];
        YPSorted[b-1]=YPSorted[b];
        YPSorted[b]=t;
        indexT=indexS[b-1];
       indexS[b-1] = indexS[b];
        indexS[b] = indexT;
      }

  for (i=0; i<N; i++)
    P[i]=0.00;
  int s;
  if (N>Psize) s=Psize;
  else s=N;

  for (i=0; i<s; i++)
    P[i]=XfftAvg[indexS[i]];

  strcpy(fileName, "/root/kde/p7/ref/");
  fileName=strcat(fileName, getlogin());
  fileName=strcat(fileName, ".bin");
  pFile = fopen (fileName, "awb");
  if (pFile==NULL) {fputs ("File error",stderr); exit (1);}

  fwrite ((&Psize), sizeof(int), 1, pFile);
  fwrite (&P, sizeof(double), N, pFile);
  fwrite ((&fftsize), sizeof(int), 1, pFile);
  fwrite (&XfftAvg[0], sizeof(double), fftsize, pFile);
  fwrite (&YfftAvg[0], sizeof(double), fftsize, pFile);

  fclose (pFile);
  drawFFT();
}

void MainWindow::borderMe()
{
  int tempX;
  int tempY;
  char message[30];

  KMenuBar * menu = menuBar();
  mwh = menu->height();

  QPixmap ttbt;
  ttbt = *tt;

  QPainter *ppbt = new QPainter(&ttbt);
  ppbt->setPen(red);
  ppbt->drawRect(x1,y1-mwh-40,x2-x1,y2-y1);

  ppbt->end();
  mywin->setBackgroundPixmap( ttbt );
}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
void MainWindow::changeLocation()
{
}

void MainWindow::setURL( const QString url )
{
}

void MainWindow::openURLRequest( const KURL &url, const KParts::URLArgs & )
{
  setURL( url.url() );
}

void MainWindow::gotoPreviousPage()
{
}

void MainWindow::bookLocation()
{
}

void MainWindow::fileSetDefaultPage()
{
  KConfig *config=kapp->config();

  config->setGroup("Settings");
  config->writeEntry( "defaultPage",  "abbaspeak" );
}

void MainWindow::resizeEvent(QResizeEvent *e)
{
  static int i=0;
  ww = width()-2;
  wh = height()-mwh-40;

  if (flag==1)    drawData();
  if (flag==2)    drawFFT();
}

void MainWindow::mousePressEvent(QMouseEvent* e)
{
  x1=e->x();
  y1=e->y();
}

void MainWindow::mouseReleaseEvent(QMouseEvent* e)
{
  x2=e->x();y2=e->y();
  if(flag==1)  borderMe();
  if(flag==2)  zoomFFT();
}

void MainWindow::mouseMoveEvent  ( QMouseEvent * e )
{
  x2=e->x();y2=e->y();
  borderMe();
}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6. voice_hui_pam.c

```
/**
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation.

 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY.  See the GNU General Public License for
 * more details.
 *
 **/

/*
 * File: pam_hui_auth.c
 *
 * Author: Hui Fang
 *
 */

#define PAM_SM_AUTH
#define PAM_SM_SESSION
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <security/pam_modules.h>
#include <security/pam_appl.h>
#include <security/pam_misc.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <alsa/asoundlib.h>
#include <math.h>
#include <stdbool.h>
#include <fftw3.h>
#include <dirent.h>
#include "myquad.c"
#include "smooth.c"
#include "myC.c"
#include <string.h>

#define ALSA_PCM_NEW_HW_PARAMS_API
#define WISDOM "/root/kde/p7/triwis.txt"
#define XYZ 0
#define SMOOTHIE 100 //smooth times  was 250
#define COMPRESS 10 //compression for FFT
#define R 5 //radius around the peak
#define f(a)  (cons*((double)(a)))
#define dB(a)  (10.0 * log10((a)/(   (double)(fftsize) * (double)(fftsize)   )))
#define NOBODY "nobody"

short int *soundD, *soundDD, *soundDDD;
unsigned int SRATE;//sample rate
off_t total;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
int cnt;
bool louder;
int *indexP, Psize;
double *Xfft, *Yfft;
long int fftmany, fftsize;
double Peak;

void readData();
void myFFT();
void chopFFT();
void findPeak();
void filterMe();
char* compareRef();
void small(int);

/*______PAM stuff_____*/
PAM_EXTERN int
pam_sm_authenticate (pam_handle_t *pamh, int flags, int argc, const char
**argv)
{
  const char *prompt = NULL, *user = NULL, *hostname = NULL, *tty_name = NULL;
  const char *service;
  const char *myselect = NULL;
  struct pam_conv *conv;
  struct pam_message *pmsg[3], msg[3];
  struct pam_response *response;
  FILE *fp, *fg;
  int i, rc, size, dir, fd, choice, retval, len;
  long loops;
  snd_pcm_t *handle;
  snd_pcm_hw_params_t *params;
  unsigned int val;
  snd_pcm_uframes_t frames;
  char *buffer, *ptoken;

  srand48(time(NULL));
  fp = fopen("/tmp/results.txt", "w");
  buffer = (char*) malloc(128*sizeof(char));
  choice=0;

  while (choice==0){
    sleep(2);
    fg = fopen("/var/gdm/:0.GreeterInfo","r");
    if (fg==NULL) perror ("Error opening file");
    else {
      while (fgets(buffer, 127, fg)){
        ptoken=strtok(buffer, "=\n");
        if (ptoken && (strcmp(ptoken, "custom-config")==0)){
          ptoken=strtok(NULL, "=\n");
        if (strcmp(ptoken, "text") ==0){choice=1;}
        if (strcmp(ptoken, "voice")==0){choice=2;}
        }
      }
    }
    fclose(fg);
    fprintf(fp, "choice=%d\n", choice); fflush(fp);
  }
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  free(buffer);

  if (choice==1){
    const char *user;
    // We always need to know who the user is
    user = NULL;
    retval = pam_get_user(pamh, &user, NULL);
    if (retval != PAM_SUCCESS)
      return (retval);
    if (user == NULL || *user == '\0')
      pam_set_item(pamh, PAM_USER, (const void *)NOBODY);
      return (PAM_SUCCESS);
    //  exit(0);
    }

  if (choice==2){
    pam_set_item(pamh, PAM_USER_PROMPT, "Speak something: ");
    //fprintf(fp,"1\n"); fflush(fp);
    pam_get_item (pamh, PAM_SERVICE, (const void **) &service);
    //fprintf(fp,"2\n"); fflush(fp);
    pam_get_item (pamh, PAM_USER_PROMPT, (const void **) &prompt);
    //fprintf(fp,"3\n"); fflush(fp);
    pam_get_item (pamh, PAM_RHOST, (const void **) &hostname);
    //fprintf(fp,"4\n"); fflush(fp);
    pam_get_item (pamh, PAM_TTY, (const void **) &tty_name);
    //pam_get_item(pamh, "custom-config", (const void **) &myselect);
    for (i=0;i<argc;i++) fprintf(fp,"argv %d = %s\n",i,argv[i]);
      fflush(fp);
    user = NULL;
    if (user == NULL || *user == '\0')
      pam_set_item(pamh, PAM_USER, (const void *)"root");

    // open file
    fd = open("/root/temp/abbaspeak",O_RDONLY);
    // Open PCM device for playback.
    rc = snd_pcm_open(&handle, "default", SND_PCM_STREAM_PLAYBACK, 0);
    if (rc < 0) {
      fprintf(stderr, "unable to open pcm device: %s\n",
      snd_strerror(rc));
    //    exit(1);
    }
    // Allocate a hardware parameters object.
    snd_pcm_hw_params_alloca(&params);
    // Fill it in with default values.
    snd_pcm_hw_params_any(handle, params);
    // Set the desired hardware parameters.
    // Interleaved mode
    snd_pcm_hw_params_set_access(handle, params,
SND_PCM_ACCESS_RW_INTERLEAVED);
    // Signed 16-bit little-endian format
    snd_pcm_hw_params_set_format(handle, params, SND_PCM_FORMAT_S16_LE);
    // Two channels (stereo)
    snd_pcm_hw_params_set_channels(handle, params, 2);
    // 44100 bits/second sampling rate (CD quality)
    val = 44100;
    snd_pcm_hw_params_set_rate_near(handle, params, &val, &dir);
    // Set period size to 32 frames.
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    frames = 32;
    snd_pcm_hw_params_set_period_size_near(handle, params, &frames, &dir);
    // Write the parameters to the driver
    rc = snd_pcm_hw_params(handle, params);
    if (rc < 0) {
      fprintf(stderr, "unable to set hw parameters: %s\n", snd_strerror(rc));
    // exit(1);
    }
    // Use a buffer large enough to hold one period
    snd_pcm_hw_params_get_period_size(params, &frames, &dir);
    size = frames * 4; // 2 bytes/sample, 2 channels
    buffer = (char *) malloc(size);
    // We want to loop for 5 seconds
    snd_pcm_hw_params_get_period_time(params, &val, &dir);
    // 5 seconds in microseconds divided by period time
    loops = 5000000 / val;
    while (loops > 0) {
      loops--;
      rc = read(fd, buffer, size);
      if (rc == 0) {
        fprintf(stderr, "end of file on input\n");
        break;
      } else if (rc != size) {
        fprintf(stderr, "short read: read %d bytes\n", rc);
      }
      rc = snd_pcm_writei(handle, buffer, frames);
      if (rc == -EPIPE) {
      // EPIPE means underrun
        fprintf(stderr, "underrun occurred\n");
        snd_pcm_prepare(handle);
      } else if (rc < 0) {
        fprintf(stderr, "error from write: %s\n",
        snd_strerror(rc));
      } else if (rc != (int)frames) {
        fprintf(stderr, "short write, write %d frames\n", rc);
      }
    }
    snd_pcm_drain(handle);
    snd_pcm_close(handle);
    free(buffer);
//save
 // close(fd);
  fd = open("/tmp/saveme",O_WRONLY | O_CREAT);

//  loops = 2000000 / SRATE; //set to 2 seconds
  //sleep(5);
  //while( (
  rc = snd_pcm_open(&handle, "default", SND_PCM_STREAM_CAPTURE, 0);
  //  < 0 )
  //{ sleep(1); }


  snd_pcm_hw_params_alloca(&params);
  snd_pcm_hw_params_any(handle, params);
  snd_pcm_hw_params_set_access(handle, params, SND_PCM_ACCESS_RW_INTERLEAVED);
  snd_pcm_hw_params_set_format(handle, params, SND_PCM_FORMAT_S16_LE);
  //snd_pcm_hw_params_set_channels(handle, params, 2);
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  /* One channels (mono) */
  snd_pcm_hw_params_set_channels(handle, params, 1);

  SRATE = 44100;
  snd_pcm_hw_params_set_rate_near(handle, params, &SRATE, &dir);
  frames = 32;
  snd_pcm_hw_params_set_period_size_near(handle, params, &frames, &dir);
  rc = snd_pcm_hw_params(handle, params);

  snd_pcm_hw_params_get_period_size(params, &frames, &dir);
  //size = frames * 4; /* 2 bytes/sample, 2 channels */
  size = frames * 2; /* 2bytes/sample, 1 channels */
  buffer = (char *) malloc(size);
  /* We want to loop for 5 seconds */
  snd_pcm_hw_params_get_period_time(params, &SRATE, &dir);

  loops = 2000000 / SRATE; //set to 2 seconds

  while (loops > 0) {
    loops--;
    rc = snd_pcm_readi(handle, buffer, frames);
    if (rc == -EPIPE) { snd_pcm_prepare(handle);
    } else if (rc < 0) { } else if (rc != (int)frames) { }
    rc = write(fd, buffer, size);
  }

  snd_pcm_drain(handle);
  snd_pcm_close(handle);
  free(buffer);
  close(fd);

  readData();
 // small(cnt);
  filterMe();
  myFFT();
  chopFFT();
  findPeak();

  char* item;
  item = compareRef();

 // pam_set_item(pamh, PAM_USER, (const void *)NOBODY);
  pam_set_item(pamh, PAM_USER, (const void *)item);
    return (PAM_SUCCESS);
  }
  fclose(fp);
}

void readData()
{
  int rc,size,myfile,i;
  snd_pcm_uframes_t frames;
  char mybyte, yourbyte;
  struct stat fileData;
  short int ztmp,zztmp,max;

  SRATE = 44100;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  frames = 32;
  //size = frames * 4; // 2 bytes/sample, 2 channels
  size = frames * 2; // 2 bytes/sample, 1 channels
  myfile = open("/tmp/saveme",O_RDONLY);
  rc = fstat(myfile, &fileData);
  if(rc == -1) return;
  total = fileData.st_size/2 ;

  if (soundDDD) free(soundDDD);{
    soundDDD = (short int*)(malloc(total*sizeof(short int)));}

  cnt=0;
  while (rc = read(myfile,&mybyte,1)) {
    if (rc == 0) { fprintf(stderr, "end of file on input\n"); break; }
    rc = read(myfile,&yourbyte,1);
    if (rc == 0) { fprintf(stderr, "end of file on input\n"); break; }

    //convert to byte
    ztmp=yourbyte;
    ztmp = ztmp << 8;
    zztmp = mybyte;
    zztmp = zztmp & 0xff;
    soundDDD[cnt++] = ztmp  + zztmp;
  }
  //chop first fftsize second off (44100/2) sample rate fftsize
  if (soundDD) free(soundDD);
  soundDD=(short int *)(malloc(sizeof(short int)*(cnt-22050)));

  for (i=22050; i<cnt; i++)
    soundDD[i-22050]=soundDDD[i];

  //soundD=soundDD+22050;
  cnt=cnt-22050;

  //chop second fftsize second off (44100/2) sample rate fftsize for noise
deduction
  if (soundD) free(soundD);
  soundD=(short int*)(malloc(sizeof(short int)*(cnt-22050)));

  short int* noiseD;
  noiseD=(short int*)(malloc(sizeof(short int)*22050));

  for (i=0; i<22050; i++)
    noiseD[i]=soundDD[i];
  for (i=22050; i<cnt; i++)
    soundD[i-22050]=soundDD[i];

  cnt=cnt-22050;
  int n=cnt/22050;
  int j;

  for (i=0; i<22050; i++)
    for (j=1; j<=n; j++)
      soundD[i*j]=soundD[i*j]-noiseD[i];

  max=0;
  for (i=0; i<cnt; i++)    if (soundD[i]>max) max=soundD[i];
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  if (max<1000){
    printf ("I can't hear you.  Please speak louder.\n");
    louder=1;
  }
}

void myFFT()
{
  long int i,j,startI, stopI;
  double cons;
  fftw_plan plan;
  double *in,*out;
  FILE *wisdomptr;

  startI = 0;
  stopI= cnt -1;

  fftmany  = stopI/COMPRESS-(startI -1)/COMPRESS;

  in  = (double *) fftw_malloc( fftmany * sizeof(double));
  out = (double *) fftw_malloc( fftmany * sizeof(double));
  fftsize = fftmany/2+1;
  cons = 0.5 * SRATE /(double)(fftsize) / (double)(COMPRESS);

  i=0;
  j=0;
  while(i < fftmany){
    j++;
    if(j%COMPRESS == 0 && (j>= startI) && (j <= stopI) ){
      in[i++]=soundD[j-1];
    }
  }
  // get wisdom

  if(wisdomptr = fopen(WISDOM,"r"))    fftw_import_wisdom_from_file(wisdomptr);

  plan = fftw_plan_r2r_1d(fftmany,in,out,FFTW_R2HC,FFTW_MEASURE);
  //small();
  fftw_execute(plan);

  Xfft = (double *)(malloc(fftsize*sizeof(double)));
  Yfft = (double *)(malloc(fftsize*sizeof(double)));
  Xfft[0]= 0.0,
  Yfft[0]= dB(out[0]*out[0]);

  for( i=1;i< fftsize; ++i){
    Xfft[i]=f(i);
    Yfft[i]=dB(out[i]*out[i]+out[fftmany-i]*out[fftmany-i]);
  }

  if(fftmany%2 == 0){
    Xfft[fftmany/2]=f(fftmany/2);
    Yfft[fftmany/2]=dB(out[fftmany/2]*out[fftmany/2]);
  }

  fftw_destroy_plan(plan);
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  // save wisdom
  wisdomptr = fopen(WISDOM,"w");
  fftw_export_wisdom_to_file(wisdomptr);

  fclose(wisdomptr);
}

void chopFFT()
{
  int i;
  double TNTMIN, TNTMAX, NEWMIN;
  char message[30];
  TNTMIN = 2000000000;
  TNTMAX = - TNTMIN;

  for(i=0;i<fftsize; ++i){
    if (Yfft[i]>TNTMAX) TNTMAX=Yfft[i];
    if (Yfft[i]<TNTMIN) TNTMIN=Yfft[i];
  }

  NEWMIN=TNTMAX-((TNTMAX-TNTMIN)*0.3);

  for (i=0; i<fftsize; i++){
    if (Yfft[i]<NEWMIN) Yfft[i]=NEWMIN;
  }
}

void filterMe()
{
  biquad *myb;
  int a;
  myb = BiQuad_new(BPF, 1, 261.626, 44100, 3);//middle C
  //BiQuad((double)*soundDD, myb);
  for (a=0; a<cnt; a++)
    soundD[a] = BiQuad(soundD[a], myb);
}

void findPeak()
{
  double P[5];
  double *YfftSmooth;
  YfftSmooth=(double *)(malloc(fftsize*sizeof(double)));

  int a, smt, i;
  for (a=0; a<fftsize; a++){
    YfftSmooth[a]=Yfft[a];
  }

  for (smt=0; smt<SMOOTHIE; smt++)
    smooth3(YfftSmooth, fftsize);

  double cons;
  cons = 0.5 * SRATE /(double)(fftsize) / (double)(COMPRESS);

  int i1, i2, Csize;
  i1= (int) (50/cons);//0;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  i2= (int) (1100/cons);//4000/cons;
  Csize=i2-i1;
  printf("i1=%di2=%dCsize=%d\n", i1, i2, Csize);

  double *YPeak;
  int* indexP;
  YPeak=(double *)(malloc(10*sizeof(double)));
  indexP=(int *)(malloc(10*sizeof(int)));
  int capacity;
  Psize=0;
  capacity=10;

  for(a=(i1+2);a<(Csize-2); a++){
    if  ( (YfftSmooth[a]>YfftSmooth[a-1]) && (YfftSmooth[a]>YfftSmooth[a-2])
        && (YfftSmooth[a] > YfftSmooth[a+2]) &&  (YfftSmooth[a] >
YfftSmooth[a+1])){
      YPeak[Psize]=YfftSmooth[a];
      indexP[Psize]=a;
      Psize++;
      capacity--;
      if (capacity==0) {
        capacity=Psize;
        YPeak=(double *)realloc((double *)YPeak, Psize*2*sizeof(double));
        indexP=(int *)realloc((int *)indexP, Psize*2*sizeof(int));

        if (YPeak==NULL){
          printf ("memory problem");
          exit(1);
        }

        if (indexP==NULL){
          printf ("indexP memory problem");
          exit(1);
        }
      }
    }
  }
  printf("..........Psizebefore=%d\n", Psize);

  int l, myIndex;
  double myMax;
  for (a=0; a<Psize; a++){
    myMax=Yfft[indexP[a]];
    myIndex=indexP[a];
    for (l=(indexP[a]-R); l<=(indexP[a]+R); l++){
      if (Yfft[l]>myMax){
        myMax=Yfft[l];
        myIndex=l;
      }
    }
    YPeak[a]=myMax;
    indexP[a]=myIndex;
  }

  //sort Peak
  int b, indexT;
  double *YPSorted, t;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  int *indexS;
  YPSorted=(double *)(malloc(Psize*sizeof(double)));
  indexS=(int *)(malloc(Psize*sizeof(int)));

  for (b=0; b<Psize; b++){
    YPSorted[b]=YPeak[b];
    indexS[b]=indexP[b];
  }

  for (a=1; a < Psize; a++)
    for (b=Psize-1; b >= a; b--)
      if (YPSorted[b-1] < YPSorted[b]) {
        t=YPSorted[b-1];
        YPSorted[b-1]=YPSorted[b];
        YPSorted[b]=t;
        indexT=indexS[b-1];
       indexS[b-1] = indexS[b];
        indexS[b] = indexT;
      }

  for (i=0; i<5; i++)
    P[i]=0.00;
  int s;
  if (5>Psize) s=Psize;
  else s=5;

  for (i=0; i<s; i++)
    P[i]=Xfft[indexS[i]];
}

char* compareRef()
{
  int smt, i, j, aa, peaksize, i1, i2, Csize, fftsizeR, a, *indexP, capacity,
b, indexT, *indexS, s;
  DIR *df;
  struct dirent *mydir;
  char *buffer, *ptoken;
  FILE *pFile, *pFile1, *pFile2, *pFile3;
  double *XfftR, *YfftR, *peakR, cons, C, peakC, peakCSorted, *YfftSmooth,
*YPeak;
  double pR[5], pS[5];
  int mergedSize, index1, index2, index3, mergedIndex;
  double *mergedPeak, *YPSorted, t;

  cons = 0.5 * SRATE /(double)(fftsize) / (double)(COMPRESS);
  i1= (int) (50/cons); //0;
  i2= (int) (1100/cons);
  Csize=i2-i1;

  YfftSmooth=(double *)(malloc(fftsize*sizeof(double)));
  buffer = (char*) malloc(128*sizeof(char));

  for (a=0; a<fftsize; a++)          YfftSmooth[a]=Yfft[a];
  for (smt=0; smt<SMOOTHIE; smt++)    smooth3(YfftSmooth, fftsize);

  YPeak=(double *)(malloc(10*sizeof(double)));
  indexP=(int *)(malloc(10*sizeof(int)));
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
Psize=0;
capacity=10;

long time_taken = time(NULL);

pFile3 = fopen ("/tmp/log.txt","awb");
fprintf (pFile3, "Login: %s\n", ctime(&time_taken));

for (a=(i1+2);a<(Csize-2); a++){
  if ( (YfftSmooth[a]>YfftSmooth[a-1]) && (YfftSmooth[a]>YfftSmooth[a-2])
    && (YfftSmooth[a]>YfftSmooth[a+2]) && (YfftSmooth[a]>YfftSmooth[a+1])){
    YPeak[Psize]=YfftSmooth[a];
    indexP[Psize]=a;
    Psize++;
    capacity--;
    if (capacity==0) {
      capacity=Psize;
      YPeak=(double *)realloc((double *)YPeak, Psize*2*sizeof(double));
      indexP=(int *)realloc((int *)indexP, Psize*2*sizeof(int));

      if (YPeak==NULL){
        fprintf (pFile3, "%s\n", "memory problem");
        exit(1);
      }

      if (indexP==NULL){
        fprintf (pFile3, "%s\n", "indexP memory problem");
        exit(1);
      }
    }
  }
}

int l, myIndex;
double myMax;
for (a=0; a<Psize; a++){
  myMax=Yfft[indexP[a]];
  myIndex=indexP[a];
  for (l=(indexP[a]-R); l<=(indexP[a]+R); l++){
    if (Yfft[l]>myMax){
      myMax=Yfft[l];
      myIndex=l;
    }
  }
  YPeak[a]=myMax;
  indexP[a]=myIndex;
}

//sort Peak
YPSorted=(double *)(malloc(Psize*sizeof(double)));
indexS=(int *)(malloc(Psize*sizeof(int)));

for (b=0; b<Psize; b++){
  YPSorted[b]=YPeak[b];
  indexS[b]=indexP[b];
}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  for (a=1; a < Psize; a++)
    for (b=Psize-1; b >= a; b--)
      if (YPSorted[b-1] < YPSorted[b]) {
        t=YPSorted[b-1];
        YPSorted[b-1]=YPSorted[b];
        YPSorted[b]=t;
        indexT=indexS[b-1];
        indexS[b-1] = indexS[b];
        indexS[b] = indexT;
      }

  for (i=0; i<5; i++)    pS[i]=0.00;
  if (5>Psize) s=Psize;  else s=5;
  for (i=0; i<s; i++)    pS[i]=Xfft[indexS[i]];

  //put data into log file /tmp/log.txt
//  fprintf (pFile3, "pS1=%f  pS2=%f  pS3=%f  pS4=%f  pS5=%f\n", pS[0], pS[1],
pS[2], pS[3], pS[4]);
//  for (a=0; a<Psize; a++)
//    fprintf (pFile3, "sample peak\n Peak=%f    YPeak=%f   indexP=%d\n",
Xfft[indexP[a]], YPeak[a], indexP[a]);

  double lookup[4][31]=
{{0, 6.314, 2.920, 2.353, 2.132, 2.015, 1.943, 1.895, 1.860, 1.833, 1.812,
1.796, 1.782, 1.771, 1.761,
1.753, 1.746, 1.740, 1.734, 1.729, 1.725, 1.721, 1.717, 1.714, 1.711, 1.708,
1.706, 1.703, 1.701, 1.699, 1.697},
 {0, 12.71, 4.303, 3.182, 2.776, 2.571, 2.447, 2.365, 2.306, 2.262, 2.228,
2.201, 2.179, 2.160, 2.145,
2.131, 2.120, 2.110, 2.101, 2.093, 2.086, 2.080, 2.074, 2.069, 2.064, 2.060,
2.056, 2.052, 2.048, 2.045, 2.042},
 {0, 31.82, 6.965, 4.541, 3.747, 3.365, 3.143, 2.998, 2.896, 2.821, 2.764,
2.718, 2.681, 2.650, 2.624,
2.602, 2.583, 2.567, 2.552, 2.539, 2.528, 2.518, 2.508, 2.500, 2.492, 2.485,
2.479, 2.473, 2.467, 2.462, 2.457},
 {0, 63.66, 9.925, 5.841, 4.604, 4.032, 3.707, 3.499, 3.355, 3.250, 3.169,
3.106, 3.055, 3.012, 2.977,
2.947, 2.921, 2.898, 2.878, 2.861, 2.845, 2.831, 2.819, 2.807, 2.797, 2.787,
2.779, 2.771, 2.763, 2.756, 2.750}};


  double SPMean=0.00;
  double RPMean=0.00;
  double SPStd=0.00;
  double RPStd=0.00;
  double DF, T, temp;

  char *fileName=(char*)(malloc(100*sizeof(char)));
  pFile = fopen ("/tmp/filename.txt","w");

  df=opendir("/root/kde/p7/ref");
  while (mydir=readdir(df))
    fprintf (pFile, "%s\n", mydir->d_name);
  fclose(pFile);
  closedir(df);

  int count=0;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  pFile = fopen("/tmp/filename.txt","r");
  if (pFile==NULL) perror ("Error opening file");
  else {
    while (fgets(buffer, 127, pFile)){
      ptoken=strtok(buffer, "\n");
      if ((ptoken[0]!= '.') && (ptoken[0]!= '\0')){
        ptoken=strtok(ptoken, ".");
        count++;
      }
    }
  }
  fclose(pFile);
  count=count+1;
  char* namelist[count][2];
  double table [count][8];
  int k=1;

  pFile = fopen("/tmp/filename.txt","r");
  if (pFile==NULL) perror ("Error opening file");
  else {
    while (fgets(buffer, 127, pFile)){
      ptoken=strtok(buffer, "\n");
      if ((ptoken[0]!= '.') && (ptoken[0]!= '\0')){
        strcpy(fileName, "/root/kde/p7/ref/");
        fileName=strcat(fileName, ptoken);
        pFile1 = fopen (fileName, "rb");
        if (pFile1==NULL) {fputs ("File error",stderr);  exit (1);}

        ptoken=strtok(ptoken, ".");
        namelist[k][0]=strdup(ptoken);

        fprintf (pFile3, "\n\nProcessing with User: %s\n", namelist[k][0]);

        while (fread (&peaksize, sizeof (int), 1, pFile1)   ){
          peakR=(double *)(malloc(peaksize*sizeof(double)));
          for (aa=0; aa<peaksize; aa++){
            fread (&peakR[aa], sizeof(double), 1, pFile1);
          }
          fread (&pR, sizeof(double), 5, pFile1);
          //fprintf (pFile3, "pR1=%f   pR2=%f   pR3=%f   pR4=%f   pR5=%f\n",
pR[0], pR[1], pR[2], pR[3], pR[4]);
          fread (&fftsizeR, sizeof (int), 1, pFile1);
          XfftR=(double *)(malloc(fftsizeR*sizeof(double)));
          YfftR=(double *)(malloc(fftsizeR*sizeof(double)));
          fread (&XfftR[0], sizeof (double), fftsizeR, pFile1);
         fread (&YfftR[0], sizeof (double), fftsizeR, pFile1);
       }
        fclose(pFile1);

       //T-test
       for (i=0; i<peaksize; i++)
         RPMean=RPMean+peakR[i];
       RPMean=RPMean/peaksize;

       for (i=0; i<peaksize; i++)
         RPStd=RPStd+pow((peakR[i]-RPMean), 2.0);
       RPStd=sqrt(RPStd)/(peaksize-1.0);
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
      for (i=0; i<Psize; i++)
        SPMean=SPMean+Xfft[indexP[i]];
      SPMean=SPMean/Psize;

       for (i=0; i<Psize; i++)
        SPStd=SPStd+pow((Xfft[indexP[i]]-SPMean), 2.0);
      SPStd=sqrt(SPStd)/(Psize-1.0);

      DF=(((SPStd*SPStd)/Psize + (RPStd*RPStd)/peaksize)*((SPStd*SPStd)/Psize
+ (RPStd*RPStd)/peaksize))/
          ((((SPStd*SPStd)/Psize)*((SPStd*SPStd)/Psize)/(Psize-1.0)) +
          (((RPStd*RPStd)/peaksize)*(RPStd*RPStd)/peaksize)/(peaksize-1.0));
       fprintf (pFile3, "DF=%f\n", DF);
      int roundup;
      roundup=DF+0.5;
       if (roundup>30) roundup=30;
      T=(SPMean-
RPMean)/(sqrt(((SPStd*SPStd)/Psize)+((RPStd*RPStd)/peaksize)));
       fprintf (pFile3, "T=%f\n", T);
      temp=lookup[3][roundup];
       fprintf (pFile3, "temp=%f\n", temp);
       //save T-test to namelist[count][1]
       if (fabs(T)<=temp)        namelist[k][1]="yes";
       else                      namelist[k][1]="no";

       table[k][1]=T;
       table[k][2]=temp;

      //T-test end
      //compare peakS and peakR

      mergedSize=peaksize+Psize;
      mergedPeak=(double *)(malloc(mergedSize*sizeof(double)));

       index1=0;
       index2=0;
       index3=0;

       while ((index1 < peaksize)&&(index2< Psize)){
         if (peakR[index1]<Xfft[indexP[index2]]){
          mergedPeak[index3]=peakR[index1];
          index1++;
        }
        else{
          mergedPeak[index3]=Xfft[indexP[index2]];
          index2++;
        }
      index3++;
      }

      if (index1<peaksize){
        while(index1<peaksize){
          mergedPeak[index3]=peakR[index1];
          index1++;
          index3++;
        }
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
        }

        if (index2<Psize){
          while(index2<Psize){
            mergedPeak[index3]=Xfft[indexP[index2]];
            index2++;
            index3++;
          }
        }

        int match, exact, newmatch;
        match=0;
        for (i=0; i<(mergedSize-1); i++){
           if ((mergedPeak[i+1]-mergedPeak[i])<10){ match++; }
        }
        exact=0;
        for (i=0; i<(mergedSize-1); i++){
          if ((mergedPeak[i+1]-mergedPeak[i])<0.0001) {exact++;}
        }
      //  fprintf(pFile3, "match=%d\nexact=%d\npeaksize=%d\n", match, exact,
peaksize);

        double percentage=0.00;
         newmatch=0;
         double temp1[Psize];
         for (i=0; i<Psize; i++) temp1[i]=Xfft[indexP[i]];
         double temp2[peaksize];
         for (j=0; j<peaksize; j++) temp2[j]=peakR[j];

         for (i=0; i<Psize; i++){
           if (temp1[i]!=0){
             for (j=0; j<peaksize; j++){
               if (temp2[j]!=0){
                 int result=fabs(temp1[i]-temp2[j]);
                 if (result<10){
                   newmatch++;
                   temp1[i]=0;
                   temp2[j]=0;
                 }
               }
             }
           }
         }
         double temp3;
         for (i=1; i<Psize; i++){
           if (temp1[i]!=0 && temp1[i-1]!=0) {
             temp3=(temp1[i]+temp1[i-1])/2;
           for (j=0; j<peaksize; j++)
             if ((temp2[i]!=0)&&(  (fabs(temp3-temp2[j])<10))){
               newmatch++;
               temp1[i]=0;
               temp1[i-1]=0;
               temp2[j]=0;
             }
           }
         }
         for (j=1; j<peaksize; j++){
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    if ((temp2[j]!=0)&&(temp2[j-1]!=0))  {
      temp3=(temp2[j]+temp2[j-1])/2;
    for (i=0; i<Psize; i++)
      if ((temp1[i]!=0)&&(fabs(temp3-temp1[i])<10)){
        newmatch++;
        temp2[j]=0;
        temp2[j-1]=0;
        temp1[i]=0;
      }
    }
  }

  fprintf (pFile3, "new match=%d\n", newmatch);
  percentage = (double) (newmatch-exact)/(double) (peaksize-exact);
  fprintf (pFile3, "new percentage=%f\n", percentage);

  table[k][3]=newmatch;
  table[k][4]=peaksize;
  table[k][5]=percentage;

 //step3 C
 C=myC(Yfft, YfftR, fftsize);

  for (smt=0; smt<SMOOTHIE; smt++){
    smooth3(Yfft,fftsize);
    smooth3(YfftR, fftsize);
  }

  C=myC(Yfft, YfftR, fftsize);
  table[k][6]=C;
  fprintf (pFile3, "smoothedC=%f\n", C);

  peakC=myC(pR, pS, 5);
  fprintf (pFile3, "peakC=%f\n", peakC);

  double tempR, tempS;
  for (i=0; i<5; i++){
    for (j=0; j < (5-1); j++){
      if (pR[j+1] > pR[j]){
        tempR = pR[j];            // swap elements
        pR[j] = pR[j+1];
        pR[j+1] = tempR;
      }
      if (pS[j+1] > pS[j]){
        tempS = pS[j];
        pS[j] = pS[j+1];
        pS[j+1] = tempS;
      }
    }
 }

  peakCSorted=myC(pR, pS, 5);
  table[k][7]=peakCSorted;
  fprintf (pFile3, "peakCSorted=%f\n", peakCSorted);

  //step4
 double refPeak, Distance;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
        refPeak=pS[0]-pR[0];
          if (fabs(refPeak)<15) fprintf (pFile3, "%s", "**********Peak match");
        Distance=0.0;
        for (j=0; j<fftsize; j++)
          Distance=Distance+pow((YfftR[j]-Yfft[j]),2.0);

        k++;
      }
    }
  }
  fclose(pFile);

  bool user=0;
  double maxi=0.00;
  for (k=1; k<count; k++){
     if ((table[k][5]>=maxi)&&(strcmp(namelist[k][1],"yes")==0))
       maxi=table[k][5];
  }
  double maxiC=0.00;
  for (k=1; k<count; k++){
    if ((table[k][6]>=maxiC)&&(strcmp(namelist[k][1],"yes")==0))
      maxiC=table[k][6];
  }
  for (k=1; k<count; k++){
    if (strcmp(namelist[k][1],"yes")==0){
      if
((table[k][5]>0.35)&&(table[k][5]==maxi)&&(table[k][6]>0.70)&&(table[k][6]==m
axiC)){
       fprintf (pFile3, "*******************The user is: %s\n",
namelist[k][0]);
       user=1;
       fclose (pFile3);
       return namelist[k][0];
      }
    }
  }
  fclose (pFile3);
  return "root";
}

PAM_EXTERN int pam_sm_setcred (pam_handle_t *pamh, int flags, int argc, const
char **argv)
{
  return PAM_SUCCESS;
}

/* --- account management functions --- */
PAM_EXTERN int pam_sm_acct_mgmt (pam_handle_t *pamh, int flags, int argc,
const char **argv)
{
  return PAM_SUCCESS;
}


/* --- password management --- */
PAM_EXTERN int pam_sm_chauthtok (pam_handle_t *pamh, int flags, int argc,
const char **argv)
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
{
  return PAM_SUCCESS;
}


/* --- session management --- */
PAM_EXTERN int pam_sm_open_session (pam_handle_t *pamh, int flags, int argc,
const char **argv)
{
  pam_set_item(pamh, PAM_USER_PROMPT, "Speak something: ");
  return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_close_session (pam_handle_t *pamh, int flags, int argc,
const char **argv)
{
  return PAM_SUCCESS;
}

/*___________E_N_D________________________*/
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

REFERENCES

Biometric. (2009). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/Biometrics

Carnegie Mellon University. (2006). *Speech at CMU.* Retrieved from http://www.speech.cs.cmu.edu/

Correlation. (2010). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/Correlation_and_dependence

Covetek. (2010). *Overall Biometrics Market in 2007.* Retrived from http://www.covetek.com.au/Info/Info3.htm

St.Denis, T. (2001). *Myquad.c.* Retrived from http://www.musicdsp.org/files/biquad.c

Digital Wenture. (2010). *Fingerprint Biometric Market Growth.* Retrieved from http://www.wenturedigital.com/component/content/article/35-latest-headlines/46-fingerprint-biometric-market-growth.html

Federal Bureau of Investigation (FBI). (n.d.). *The Integrated Automated Fingerprint Identification System (IAFIS).* Retrieved from http://www.fbi.gov/hq/cjisd/iafis.htm

FFTW. (n.d.). *FFTW.* Retrived from http://www.fftw.org

FFTW-Tutorial. (2003). *FFTW-Tutorial.* Retrived from http://www.nanophys.kth.se/nanophys/fftw-info/fftw_2.html

Fourier Transform. (2010). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/Fourier_transform

Frommer, D. (2006). *New biometric devices promise better ways to eyeball your identity.* Retrieved from http://www.msnbc.msn.com/id/11523709

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

GDM. (2004). *GDM user manual.* Retrived from http://www.jirka.org/gdm-documentation/x1259.html

Jamison, N. (2009). *Your voice is your key.* Retrived from http://www.jamison-consulting.com

Jones, K. (2008). *Lockheed Gets $1 Billion FBI Biometrics Contract.* Retrieved from http://www.informationweek.com/news/security/showArticle.jhtml;jsessionid=C0DEJFDP4EC0AQSNDLPCKHSCJUNN2JVN?articleID=206503290&_requestid=402519

Kawamura, A., Fujii, K., Itoh, Y., Fukui, Y., (2001). *A new noise reduction method using linear prediction and system Identification.* Retrieved from http://www.iwaenc.org/proceedings/2001/main/data/kawamura.pdf

KDE. (2010). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/Kde

KDE Tutorial. (2002). *KDE Tutorial.* Retrived from http://developer.kde.org/~larrosa/tutorial/

Lucas, V. (2003). *VoiceAuth.* Retrieved from http://web.archive.org/web/20060217095644/http://cscience.org/ ~lucasvr/projects/voiceauth.php

Myers, L. (2004) *An Exploration of Voice Biometrics.* Retrived from http://www.sans.org/reading_room/whitepapers/authentication/exploration-voice-biometrics_1436

Noise. (2010). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/Noise

Nuance. (2007). *Nuance Website Home Page.* Retrieved from http://www.nuance.com/.

Peng, H. (2009). *Digital Signal Processing For Denoising Electrophysiological Data.* (Master's thesis) Southern Connecticut State University, New Haven, CT.

State of Rhode Island. (2010). *VoiceVerify FAQ.* Retrived from http://www.dlt.state.ri.us/ui/voiceverifyfaqs.htm

Tranter, J. (2004). *Introduction to Sound Programming with ALSA.* Retrieved from

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

http://www.linuxjournal.com/article/6735

T-test. (2010). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/T-test

VoiceID. (2004). *VoiceID.* Retrieved from http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci944937,00.html

VoiceVault. (2009). *Caller Authentication.* Retrieved from http://www.voicevault.com/ca.html

VoiceVerified. (2008). *Voice Authentication Suite 2.0.* Retrieved from http://www.voiceverified.com/prod-vas.htm

Weber, D. (2007). *Fingerprint is school-lunch ticket.* Retrieved from http://articles.orlandosentinel.com/2007-04-27/news/SCANS27_1_scanners-pine-crest-elementary-seminole-schools

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.