Development of a Field-Deployable Voice-Controlled Ultrasound Scanner System

A Thesis submitted to the faculty
of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for
The Degree of Master of Science
in
Electrical and Computer Engineering
by

Dalys Sebastian

June 2004

APPROVED

Prof. Peder C. Pedersen, Major Advisor

Prof. William R. Michalson, Committee member

Prof. R. James Duckworth, Committee member

Prof. Fred J. Looft, Department Head

Abstract

Modern ultrasound scanners are portable and have become very useful for clinical diagnosis. However, they have limitations for field use purposes, primarily because they occupy both hands of the physician who performs the scanning. The goal of this thesis is to develop a wearable voice-controlled ultrasound scanner system that would enable the physician to provide a fast and efficient diagnosis. This is expected to become very useful for emergency and trauma applications.

A commercially available ultrasound scanner system, Terason 2000, was chosen as the basis for development. This system consists of a laptop, a hardware unit containing the RF beamforming and signal processing chips and the ultrasound transducer. In its commercial version, the control of the ultrasound system is performed via a Graphical User Interface with a Windows-application look and feel. In the system we developed, a command and control speech recognition engine and a noise-canceling microphone are selected to control the scanner using voice commands. A mini-joystick is attached to the top of the ultrasound transducer for distance and area measurements and to perform zooming of the ultrasound images. An eye-wear viewer connected to the laptop enables the user to view the ultrasound images directly. Power management features are incorporated into the ultrasound system in order to conserve the battery power. A wireless connection is set up with a remote laptop to enable real-time transmission of wireless images. The result is a truly untethered, voice-controlled, ultrasound system enclosed in a backpack and monitored by the eye-wear viewer. (In the second generation of this system, the laptop is replaced by an embedded PC and is incorporated into a photographer's vest).

The voice-controlled system has to be made reliable under various forms of background noise. Three command and control speech recognition systems were selected and their

recognition performances were determined under different types and levels of ambient noise. The variation of recognition rates was also analyzed over 6 different speakers. A detailed testing was also conducted to identify the ideal combination of a microphone and speech recognition engine suitable for the ultrasound scanner system. Six different microphones, each with their own unique methods of implementing noise cancellation features, were chosen as candidates for this analysis. The testing was conducted by making recordings inside a highly reverberant acoustic noisy chamber, and the recordings were fed to the automatic speech recognition engines offline for performance evaluation. The speech recognition engine and microphone selected as a result of this extensive testing were then incorporated into the wearable ultrasound scanner system. This thesis also discusses the implementation of the human-speech interface, which also plays a major role in the effectiveness of the voice-controlled ultrasound scanner system.

Acknowledgements

I express my sincere gratitude to my advisor, Prof. Peder C Pedersen, for his enormous help and advice, his extreme patience in reviewing all of my reports and all of the chapters of this thesis and his tremendous efforts to bring about useful results at every aspect of this thesis. His innovative ideas and his persistence for excellence have given me the motivation and confidence to persevere beyond my goals. I believe that, everything that I learnt during the course of this thesis will benefit me considerably in my professional life.

I also like to acknowledge Prof. Richard Campbell, Prof. Jim Matthews, Prof. William Michalson and Prof. James Duckworth for their help and valuable comments during different stages of the research. I also thank Prof. James Duckworth and Prof. William Michalson for being my thesis committee members.

I am grateful to Jack Coyne, who has been of tremendous help to me during the later part of this thesis. His thoughtful and useful comments and his sincere efforts in running the microphone and speech recognition engine evaluation tests, have contributed to the benefit of this thesis. I would also like to thank him for providing all the Matlab scripts required to perform the speech recognition evaluation tests.

I also wish to thank Carsten Poulsen for creating the cuing device and for his help in the microphone evaluation tests. I am grateful to all the subjects who took part in the speech recognition and microphone evaluation tests. I am also thankful to Deepti, Aditya and Carsten for the pleasant and enjoyable hours in the lab.

I am indebted to my parents for the love, encouragement and support they gave throughout my academic and professional life. Last but not least, I express my gratitude towards my husband, Deepak, for giving his love, motivation, advice and support and helping me to achieve my goals.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

Many of the deaths in an emergency environment, such as a battle field, are due to the lack of timely medical attention. As a single physician is usually assigned to several numbers of wounded soldiers a day, it becomes very important for him/her to make a fast and accurate diagnosis in an efficient manner. One of the important diagnosis tools of a physician today is the ultrasound scanner. Modern ultrasound scanners, although available in light-weight and portable models, have limitations when used in emergency situations. The physician performing the scanning has to hold the transducer and at the same time control the scanner through a keyboard. Additionally, while scanning the patient, the physician has to observe the ultrasound image on the screen of the scanner which could be located elsewhere. A physician might find the execution of all these tasks both difficult and inconvenient. This type of scanning method may result in diagnosis errors, patient dissatisfaction and the inefficient usage of time. How can a physician's job in a field hospital be made more efficient?

This thesis describes a wearable ultrasound scanner solution, aimed at making ultrasound diagnosis quicker and easier. The development was done based on the commercially available scanner system, Terason 2000. The following improvements were made: All scanner controls are made active through voice commands. The ultrasound image can be viewed by wearing an eyewear viewer. A wireless connection is established with a remote laptop, to enable real-time transmission of the ultrasound images. A mini-joystick is fitted on top of the ultrasound transducer to make distance and area measurements. The first generation of the scanner system,

which is covered in this thesis, enables the physician to carry the ultrasound scanner around by wearing it in a backpack. The second generation system, developed as a continuation of this work, has enclosed this wearable scanner in a photographer's vest, with the whole vest weighing less than 5 lbs.

The most critical element of the wearable ultrasound scanner system is its speech recognition feature. The reliability of the speech recognition feature is decided by the choices made in the selection of the speech recognition engine, the microphone and the design of the human-speech interaction interface. As the ultrasound scanner is intended for a field environment, the speech recognition feature must be designed to tolerate moderately high levels of ambient noise. This thesis is devoted to the study and implementation of the speech recognition feature of this wearable ultrasound scanner system and to make it more reliable in noisy environments.

Until recently, the ultrasound scanners used were large, heavy and mostly used inside hospitals. Portable ultrasound scanners have become a reality today with the introduction of systems like *Sonosite 180 Plus* and *iLook25* from Sonosite [1], *Optigo* from Philips and *Terason 2000* from Teratech Corporation [2]. Figure 1.1 shows pictures of these scanners.

(a) **(b)** **(c)**

Figure 1.1: Portable ultrasound scanners, (a) Sonosite 180 Plus, (b) Optigo, (c) Terason 2000

Sonosite 180 Plus is a battery-powered, light-weight, portable ultrasound scanner system. Its main features are: pulsed wave doppler mode, directional color power doppler mode, color power doppler mode and tissue harmonic imaging. All scanner controls are done by means of keyboard. Optigo system is battery-powered, light-weight, portable and supports color flow doppler mode. All screen controls are software-driven. Terason 2000 system is PC based, and consists of the SmartProbe (the transducer and a hardware unit containing the RF signal processing chips), the scanner software and the laptop. It supports power doppler mode, directional power doppler mode, pulsed wave doppler mode, color doppler mode and spectral doppler mode. In its commercial version, the operation of the Terason ultrasound system (controls and settings) is performed via a Graphical User Interface (GUI) with a Windows-application look and feel. Unlike Sonosite and Optigo, the Terason 2000 system has a PC-based architecture, making it flexible for the integration with other state-of-the-art technologies. Therefore, the commercial version of Terason 2000 has been chosen as the basis for the development of the wearable ultrasound system described in this thesis. This permitted the

incorporation of voice recognition features, integration with an eye-wear viewer and wireless transmission of ultrasound images, thus enhancing the capabilities of the wearable ultrasound scanner. The open architecture of Terason 2000 also allowed the incorporation of enhanced power management features to optimize the battery life of the wearable scanner.

There are several types of Automatic Speech Recognition (ASR) engines available today. The functionality of these engines differs according to their intended application areas. *Dictation* is one application that ASR engines are frequently designed for. This becomes useful in scenarios such as creating documents, completing forms, sending email etc. using voice. In these types of applications the speech is converted into text by the ASR engine and directly displayed as text. But an ASR engine required for our application is meant for a different purpose. Rather than dictation, the user gives voice commands in order to execute some scanner control. This type of application is called *command and control*. It is true that, in both of these applications, the ASR engine converts the speech into text. But the methods by which the ASR engines perform these tasks are functionally very different. In the dictation application, the words that an ASR engine recognizes are not predefined and can come from a very large vocabulary. But in the latter case, the ASR engine is aware of the words that the user may speak. The precise recognition of a user's speech is more critical in a command and control application than in dictation.

We have not been able to find a published comparison of the command and control ASR engines available in market today. Therefore, it was a challenge to identify the ASR engine best suited for our purpose. The ASR usually comes with manufacturer-specific software interfaces which an application can use in its software to incorporate the ASR features. Therefore, a thorough study has to be performed for all of the specific ASR features, and their software interfaces have

to be identified, before it can be programmed into our application. We reviewed available speech recognition engines from which we selected three appropriate ones, and made quantitative comparisons between them, before identifying the ideal ASR engine required for our application.

Once an ASR engine is properly chosen and implemented, the major question that remains is, "How is the recognized text used to control the ultrasound scanner?". The answer to this question may not be difficult if we had full access to the Terason scanner software. But, as the scanner software is proprietary, a method had to be devised to accomplish the same purpose. In addition, the total number of scanner controls that an ASR engine has to perform is of the order of around 100. But, if all of these commands are made active at any time, the accuracy of the ASR engine will become reduced. Therefore, the command set that an ASR engine can recognize at any time is dynamically changed depending on the command that the user spoke just prior to that time. The usefulness of a speech recognition feature is also decided to a great extent by the friendliness of the human-speech interaction interface. All these considerations bring about the need for a highly structured software architecture, which has been implemented using object-oriented design practices and is also part of this thesis.

The reliability of the speech recognition feature in an environment with realistic ambient noise depends on the ASR engine's tolerance to low speech Signal to Noise Ratios (SNRs). Another important factor that contributes to noise tolerance is the effectiveness of the microphone. Conventional airborne microphones pick up speech from the sound waves propagated in the air. Therefore, if noise is also present there is a greater chance that the speech picked up has low SNRs. In addition, in the hectic environment of emergency medicine, airborne microphones are found to be inconvenient to wear. This thesis looks into the possibilities of using other types of

microphones. A throat microphone, which detects speech from the vibration of the vocal cords propagated through the soft tissues of the throat, and a jawbone conduction microphone which detects speech from bone conduction to the ear, are a few examples. Experiments are also performed with array microphones, which can be conveniently placed in a speaker's shirt pocket or ear and detect speech from the air. Quantitative comparisons are performed in order to find the ideal combination of speech recognition software and microphone type that can perform reliably under noisy conditions.

1.1. Overview of the Thesis

Chapter 2 describes briefly Terason ultrasound scanner and its setup when used in a wearable mode. The basics of speech recognition and the various characteristics of ASR engine are also discussed. Finally, the important features of the ASR engine suited for controlling the Terason ultrasound scanner system are also discussed.

Chapter 3 explains in detail the various elements of the wearable ultrasound scanner solution implemented in this thesis. The eyewear viewer, the mini-joystick and voice-activation are some of the important topics dealt with here. An overview of the speech interface design for the wearable scanner is also discussed in this chapter. The two major components of the speech interface design are the *Voice command recognizer* and the *Voice command dispatcher*. The *Voice command recognizer* is responsible for the recognition of the voice command in the form of text. The *Voice command dispatcher* is responsible for controlling the scanner, based on the recognized voice command.

Chapter 4 discusses the implementation details of the speech recognition interfaces. As mentioned earlier, the software interface functions provided by an ASR engine is manufacturer specific. This chapter examines the architecture and working of three different speech recognition engines, and explains the implementation steps to be followed in order to integrate the ASR engine with any other software. Thus, this chapter provides the fundamentals for the implementation of the *Voice command Recognizer* component of the overall speech interface required for the ultrasound scanner.

Chapter 5 presents the architecture and the object-oriented practices used to design the *Voice command dispatcher* component of the speech interface. This chapter describes in detail the different classes involved and provides activity diagrams to explain the communication between classes.

Chapter 6 summarizes the study made on power management, and describes how the ultrasound scanner can be put into different sleep states through software interfaces. The implementation aspects of hibernate and standby states are covered here. Measurements are made to estimate the power consumed during standby and hibernate states.

Chapter 7 describes the evaluation protocol followed in order to analyze the performance of different ASRs under different types of noisy environments for different speakers and presents the results obtained. These results give valuable information regarding an ASR engine's tolerance to ambient noise in speech and its' speaker dependence. The selection of the ASR engine for the wearable ultrasound scanner was made based on the results discussed in this chapter.

Chapter 8 presents the various microphone candidates and explains the test protocol devised for identifying the best combination of ASR engine and microphone. The different microphone candidates include conventional airborne microphone, jawbone conduction microphone, physiological sensor microphone, bluetooth microphone and 4 element array microphone. Each of these microphones has unique characteristics, but its effectiveness for speech recognition in a highly reverberant (diffuse) noisy environment is analyzed here. The results are presented and conclusions are discussed with respect to each of the microphones.

Chapter 9 states the overall conclusions and gives recommendations for future work

Chapter 2. Background

2.1. PC Based Ultrasound Scanning – Terason 2000

The Terason 2000 ultrasound scanner consists of the SmartProbe (the transducer and a hardware unit containing the RF beamforming and signal processing chips), software and the laptop. A Terason 2000 scanner system is shown in Figure 2.1. The software is stored on the hard-disk drive of the laptop. The operation of the Terason ultrasound system (controls and settings) is performed via a Graphical User Interface (GUI) with a Windows-application look and feel. Unlike other ultrasound systems, which run on proprietary devices, the Terason 2000 scanner software resides on a laptop. This provides flexibility in integration with other state-of-the-art technologies to enhance its features. A few examples are integration with digital radiology, internet-based tele-radiology and medical informatics systems.

Figure 2.1: Terason 2000 ultrasound scanner system,
showing the SmartProbe (on top of the laptop) and several array transducer types

Terason 2000 supports several types of array transducers with frequencies ranging from 2-10 MHz, connected to a SmartProbe. The SmartProbe, which includes all the electronics, weighs approximately 10 ounces (0.28 kg). More details regarding the working of the SmartProbe can be found in [4]. The SmartProbe is a plug-and-play imaging device which is linked to the host PC through an IEEE 1394 (or Firewire) serial cable. In the commercially available Terason system, the power for the SmartProbe is directly supplied by the laptop. The laptop is powered by a medical grade power supply. But it is alternatively possible to supply the power to the SmartProbe using an external battery, if the laptop is unable to supply the power on its own. This will become useful when the ultrasound system is used as a wearable system, such as in a field environment. Then the laptop would be fully charged, and a fully charged external battery can drive the SmartProbe even up to 7 hours. The setup of the Terason 2000 system when the SmartProbe is powered by an external power supply is shown in Figure 2.2.

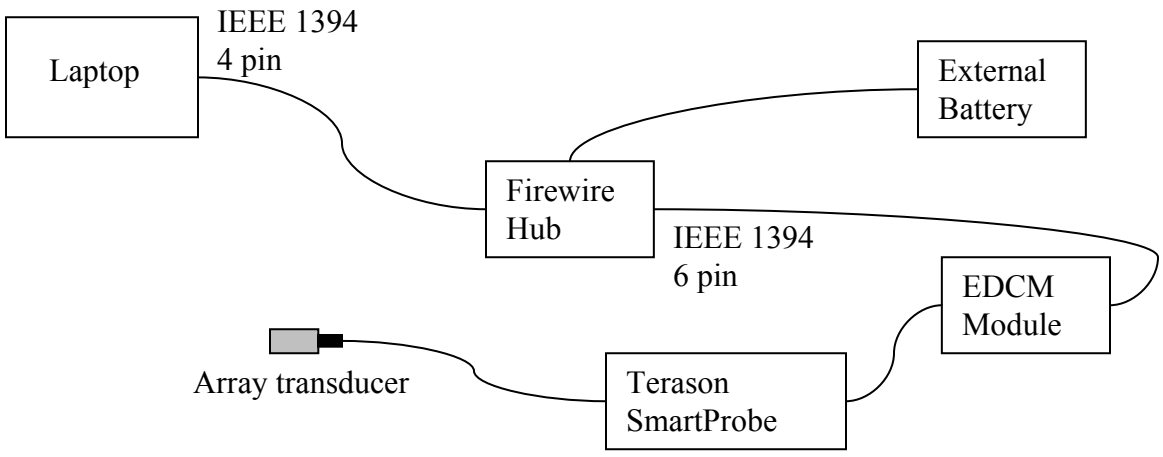


Figure 2.2: Terason Probe powered by an external battery

The laptop's 4-pin IEEE 1394 interface is connected to an external Firewire hub, by using a Firewire cable. The hub is powered by an external battery. The hub provides 6-pin Firewire outputs. The hub is connected to the SmartProbe via a voltage regulator module called the EDCM (External DC module).

The Terason 2000 system was selected for use in the ultrasound system described in this thesis, due to the opportunities it provides for integration with the other technologies. A command and control speech recognition engine was integrated with the Terason scanner software for providing voice-activation. The laptop was configured with a remote laptop for wireless real time transmission of image data. The next section provides background on speech recognition technology in general and finally identifies the requirements of a speech recognition engine suited for controlling an ultrasound scanner system.

2.2. Speech Recognition Overview

Speech recognition is the process by which a computer identifies spoken words. The computer may use this recognized word to control another machine or it may output it as text into a document. In this research, specific spoken command words are intended for the control of an ultrasound scanner system. For example, a user can say something like '*b mode*' or '*m mode*' to view different modes of operation of the scanner. He/she could say commands like '*Freeze*' in order to freeze an ultrasound image. This enables the user to perform multiple tasks at the same time and complete his/her work faster and more efficiently. Figure 2.3 shows how the voice commands are recognized by a computer, with the help of a block diagram.

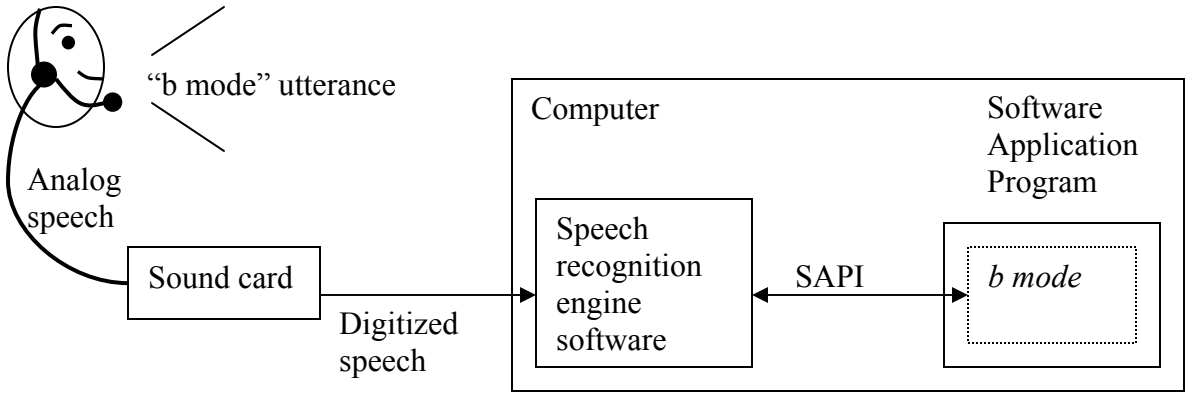


Figure 2.3: Block diagram of speech recognition and execution of voice commands

The user wears a microphone, which captures the sound and sends it to a sound card. The laptop may have its own built-in sound card or an external sound card may be used. The analog signal from the microphone is digitized by the sound card and then sent to the speech recognition engine software running on the laptop. The digital signal from the sound card must fulfill certain specifications. The sampling frequency of the digitized signal must be 11,025 Hz or 22,050 Hz depending on the speech recognition engine. The number of quantization bits of the digital speech is specified to be 16. The software application program running on the laptop informs the speech recognition engine about the vocabulary it wants the engine to recognize. The interactions of the application program with the speech recognition engine are done in the form of software functions which is specified by the speech recognition engine manufacturer. These software functions are generally called *Speech Application Program Interfaces* (SAPI). The next section examines the inside working of a speech recognition engine.

2.2.1. Inside of an Automatic Speech Recognition Engine

The main system components of automatic speech recognition (ASR) engine are shown in Figure 2.4:

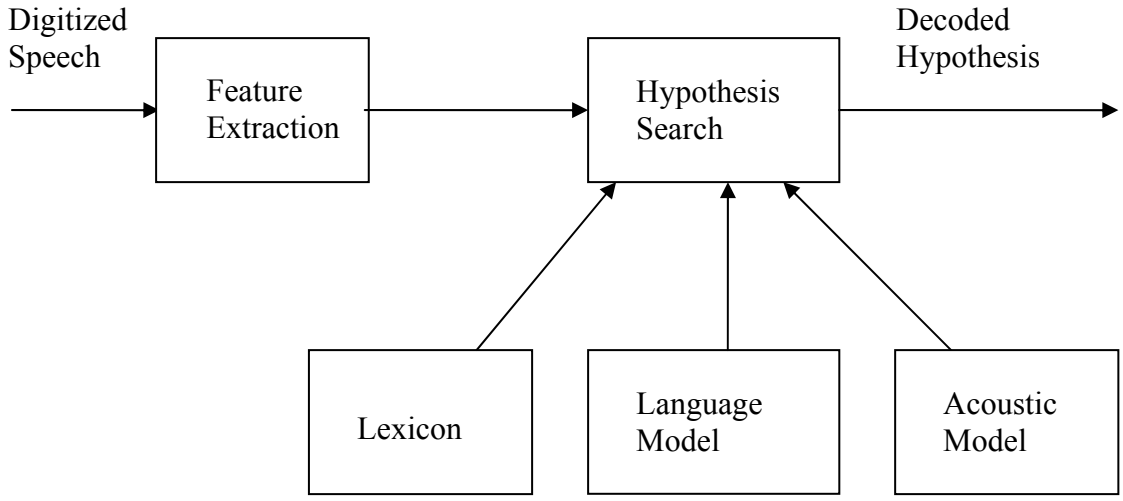


Figure 2.4: Structure of an ASR engine

The first block inside the ASR engine is a feature extraction system, which extracts a sequence of acoustic feature vectors from the speech signal. These feature vectors represent information concerning various attributes of the speech signal, e.g. its frequency content, short-term power, energy etc. These feature vectors are typically extracted every 10 ms. The lexicon defines the possible words that the search can hypothesize, representing each word as a sequence of phonemes. The language model models a linguistic structure, by computing the probability of occurrence of a particular sequence of words. The acoustic model models the relationship between feature vectors and phonemes [8].

2.2.2. Types of Automatic Speech Recognition Engines

There are a large number of ASR engines in the market. Normally ASR engines are designed for use in specific application areas. These application areas can be broadly classified into two types:

- Command and control applications
- Dictation applications.

In command and control applications, the user speaks a command, and the application recognizes the command and executes some processing based on the command. In dictation applications, the application does not interpret the user's input, but simply returns the speech as text into a document.

The ASR engine used for command and control find wide range of applications, such as manufacturing, wearable computing, telephone network, controlling personal workstations, voice surfing devices etc. Dictation applications are useful in inventory control, in medicine where voice input can significantly accelerate the writing of routine reports.

The ASR engine's design and implementation is dependent on the application it is targeted for. The ASR engine used in embedded devices usually works on the concept of Distributed Speech Recognition, based on client-server architecture. The client element of the software would reside in the embedded device, for example a mobile phone, while the server component may reside in a remote server. This is because of the limited memory capacity of the embedded device. The ASR engines suited for telephony applications are usually designed to work with limited

bandwidth. Due to this reason, the speech data is compressed before transmission. This creates additional processing tasks for such engines.

2.2.3. Speech Recognition Technology – an Overview

Speech recognition technology can be defined using 4 factors:

- Word selection
- Dependence on speaker
- Word analysis
- Vocabulary

2.2.3.1 Word Selection

The speech recognition engine must have some method of listening to a speech input, and determining when a word item has been uttered. There are three different methods of selecting speech from an input stream. They are:

- Discrete speech

Discrete speech is the simplest form of word selection. Under discrete speech, the engine requires a slight pause between each word. This pause marks the beginning and end of each word item. Discrete speech requires the least amount of computational resources. With a discrete speech system, users must speak in a halting voice. This is the format used for command and control applications, but it is not appropriate for extended periods of dictation.

- Word spotting

A much more preferred method of handling speech input is word spotting. Under *word spotting*, the speech engine listens for a list of key words along the input stream. This method allows users to use continuous speech. Since the system is listening for key words, users do not need to use unnatural pauses while they speak. The advantage of word spotting is that it gives users the perception that the system is actually listening to every word while limiting the amount of resources required by the engine itself. The disadvantage of word spotting is that the system can easily misinterpret an input. For this reason, it is important to design vocabularies for word-spotting systems that limit the possibility of confusion.

- Continuous speech

The most advanced form of word selection is the continuous speech method. Under *continuous speech*, the user is allowed to speak normally anything he/she wants out of a specific vocabulary. This is the most resource-intensive of the word selection methods. For this reason, continuous speech is usually reserved for dictation systems that require complete and accurate perception of every word.

2.2.3.2 Dependence on Speaker

The process of word selection can be affected by the speaker. This factor refers to the engine's ability to deal with different speakers. Systems could be of three types:

- Speaker dependent

Speaker-dependent systems provide the greatest degree of accuracy while using the least amount of computing resources. A speech recognition system of this type requires extensive training by each user before it can become very accurate.

- Speaker adaptive

Speaker-adaptive systems are designed to perform adequately without training, but they improve with use. The advantage of speaker-adaptive systems is that users experience success without tedious training. Disadvantages include additional computing resource requirements and possible reduced performance on systems that must serve several people.

- Speaker independent

Speaker-independent systems are needed where multiple speakers need to use the same station. The drawback of speaker-independent systems is that they require the greatest degree of computing resources.

2.2.3.3 Word Analysis

Once a word item has been selected, it must be analyzed. *Word analysis* techniques involve matching the word item to a list of known words in the engine's vocabulary. There are two methods for handling word analysis:

- Whole-word matching

Under whole-word matching, the ASR engine matches the word item against a vocabulary of complete word templates. The advantage of this method is that the engine is able to make an accurate match very quickly, without the need for a great deal of computing power. Also, these

words must be stored as spoken templates. Each word can require as much as 512 bytes of storage.

- Sub-word matching

An alternate word-matching method involves the use of sub-words called *phonemes*. Each language has a fixed set of phonemes that are used to build all words. By informing the ASR engine of the phonemes and their representations it is much easier to recognize a wider range of words. Under sub-word matching, the engine does not require an extensive vocabulary. An additional advantage of sub-word systems is that the pronunciation of a word can be determined from printed text. Phoneme storage requires only 5 to 20 bytes per phoneme. The disadvantage of sub-word matching is that it requires more processing resources to analyze input.

2.2.3.4 Vocabulary

Speech recognition system may have small vocabulary, medium vocabulary or large vocabulary. Small vocabulary systems (100 words or less) work well in command and control speech recognition. However, to handle a dictation system we need to have large vocabulary. Dictation vocabularies can reach up to tens of thousands of words. Increasing the vocabulary size will increase the amount of computation for the ASR engine.

2.2.4. Grammar Rules

One of the techniques to reduce the computation and increase accuracy for ASR engines is to specify the grammar rules. Speech grammars form a structured collection of words and phrases bound together by rules that define the set of speech streams the speech engine can recognize at a

given time. Grammar allows developers to specify flexible ways of speaking a particular command. Grammars can be divided into three types:

- Context free grammars
- Dictation grammars
- Limited domain grammars

Context Free Grammars (CFG) work by limiting the vocabulary and syntax structure of speech recognition to only those words and sentences that are applicable to the application's current state. This method is useful, if the speech input to the SR engine is not arbitrary, but is pre-defined. The important feature is that it limits what the recognizer expects to hear to a small vocabulary and tight syntax. The application specifies the vocabulary and syntax structure in a text file. The SR engine uses this file, for identifying the speech commands.

Dictation grammars offer the greatest degree of accuracy when converting spoken words into printed text. They normally work over large vocabularies. Limited domain grammar offers a compromise between context-free grammar and dictation grammar.

2.2.5. Speech Recognition Engine for Terason System

The ideal requirements suggested for a speech recognition engine suited for controlling the Terason ultrasound scanner are described below.

- Command and control ASR engine suited for wearable computing applications

- Small and extendable vocabulary systems
- Supports Context Free Grammar
- Speaker-independent/Speaker dependent
- Fast response time (less than 1 second)
- High recognition accuracy, even under high ambient noise

The chosen command and control ASR engine must be suited for PC based applications. In other words, the speech recognition engine must be simple in architecture and completely resides inside a single PC. Also, the ASR engine is preferred to be of small foot-print, i.e., the engine's vocabulary is small and requires little memory. The support of context free grammar features would allow the user to speak only those commands that are specified in the grammar. The context free grammar must be able to provide flexibility to change the ASR engine's vocabulary during run-time. This feature is very useful for controlling recognition rates and gives more scope for designing efficient human-computer voice interaction. Although speaker-independent ASR engine is more convenient to use as they need no training, it is generally found that ASR engines with some form of user-specific training gives better recognition rates for that particular user. It is recommended that the ASR engine has a fast response time, as the ultrasound scanner may be used in emergency applications. The speech recognition engine should give reasonably good performances under moderate ambient noise. This is especially important as the ultrasound scanner system in this project is targeted for field-use purposes.

2.2.5.1 Soundcard

A good quality sound card is essential for getting good recognition accuracies. The sound card must support user selectable sampling frequencies as well as user selectable bit rates. Sound Blaster Live 5.1 sound card is a recommended choice for speech recognition for use in PCs. It comes as a plug-in PCMCIA card. It is also possible to get external sound card in the form of Universal Serial Bus (USB) adapters. This option can be used if the PC's or laptop's own sound card has poor quality audio circuitry. The USB adapter converts the incoming audio into a digital signal and then presents it to the PC's USB port. In this way, the PC's or laptop's own sound card is bypassed completely.

2.2.5.2 Microphone

Excellent wideband response, noise cancellation features and ease of use are some of the important factors to consider before choosing a microphone for speech recognition. The airborne microphones available today come as headset and desktop models. The headset microphones are preferable to desktop models, as the latter are more susceptible to background noise. In addition to airborne microphones, we have tested a throat microphone, a jawbone conduction microphone, an array microphone and a bluetooth wireless microphone employing varying types of noise-canceling technologies for performance under moderate levels of ambient noise.

Chapter 3. System Architecture

3.1. Wearable Ultrasound Scanner Setup

A schematic of the first generation wearable ultrasound scanner setup is shown in Figure 3.1. The Terason 2000 ultrasound scanner is used along with a light-weight, high performance laptop. In the second generation ultrasound system, developed as a continuation of this work, the laptop has been replaced with an embedded PC with no keyboard. A command and control speech recognition engine and a noise-canceling microphone are selected to control the scanner using voice commands. An eyewear viewer connected to the laptop through a VGA cable is used for viewing the ultrasound images. A wireless connection is set up with a remote computer to enable real-time transmission of ultrasound images. A mini-joystick, mounted on top of the transducer, is used to make distance and area measurements. The laptop can be put into sleep mode through voice commands when not in use in order to conserve power. The power for the scanner is supplied from an external battery.

The laptop, the scanner units and the battery for supplying the power are enclosed in a backpack. Logon scripts are executed when the laptop is switched on, to bring up the Terason scanner software and the speech recognition applications. It also establishes the wireless connection with the remote laptop if necessary. Figure 3.2 shows a person wearing the wearable ultrasound system.

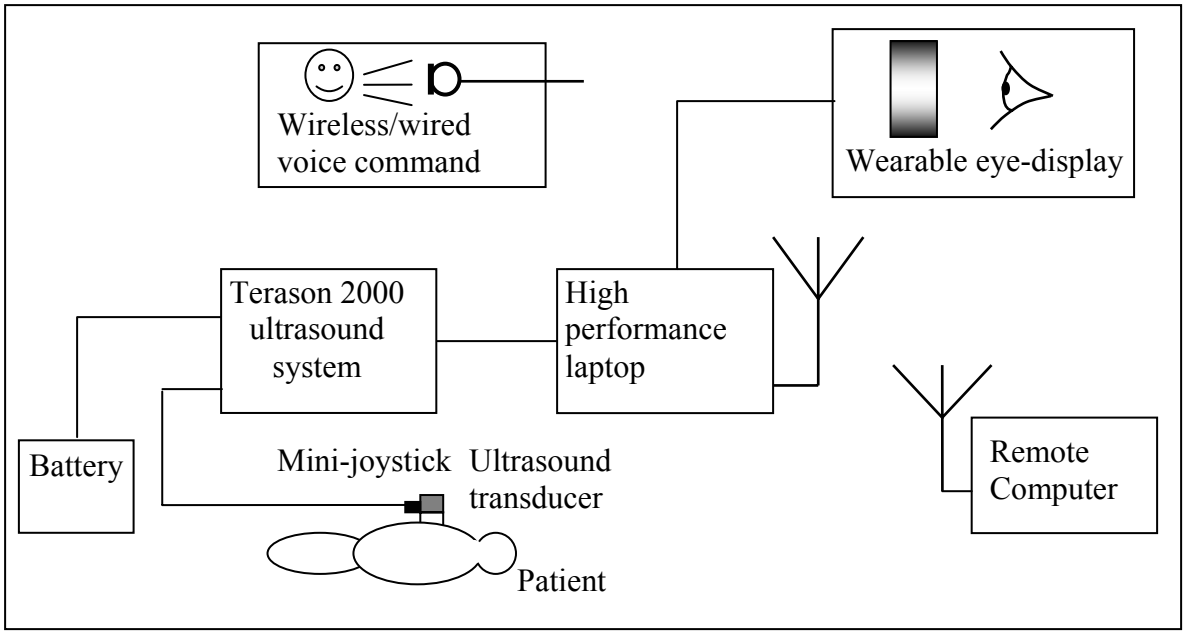


Figure 3.1: First generation ultrasound scanner setup

Figure 3.2: Photograph of a person wearing the ultrasound scanner system

3.2. Components of the Wearable Ultrasound System

This section describes each of the components involved in greater detail.

3.2.1. Terason 2000 Ultrasound System

The Terason 2000 ultrasound system and its configuration when powered by an external power supply were described in greater detail in Section 2.1. The same setup as shown in Figure 2.2 is used here also for powering the ultrasound scanner. The external battery used for powering the scanner is UltraLife LWC-L. It is a Lithium ion rechargeable battery and has a capacity of 7.5 AHrs or 114 Watt-hours. It operates at an average voltage of 15.2 V and can supply current up to 6.5 A.

3.2.2. High Performance Laptop

The laptop used was Sony VAIO V505A series. Its important specifications are: Intel Pentium 4-M processor with a clock speed of 2 GHz, 1024 MB RAM, 40 GB hard drive, integrated wireless LAN, IEEE 1394 (4-pin) interface, 2 USB ports, VGA output, headphone (stereo), monaural microphone input, internal CD-RW drive, double capacity lithium ion battery, PCGA-BP4V, providing a total battery life of around 6 hours, total weight of 4.34 lbs including the battery. The Windows XP operating system is installed in this laptop. As the laptop provides only 4-pin Firewire interface and the Terason scanner requires 6 pin Firewire connect for supplying power, a Firewire hub was used. The Firewire hub used is Belkin Firewire hub, with 6-pin Firewire sockets at either side. The hub has a socket for supplying power to which the UltraLife external battery can be connected. The setup is shown in Figure 2.2.

3.2.3. Eyewear Viewer

The SV-6 PC viewer from *MicroOptical Corporation* served as an eyewear viewer for monitoring the laptop screen. It is a small, light, head-up display and can be mounted on eye-glasses, as shown in Figure 3.3. It has a quick mount and dismount system so that it can be easily attached and detached from the eyewear. It can be used with either the left or right eye and includes a built-in focus mechanism. The battery and the controls for adjusting the image are enclosed in a small lightweight pack which can be clipped on the backpack. It is compatible with computers that provide a VGA signal at 60 Hz. The pixel format of the SV-6 is 640 columns by 480 rows. But SV-6 has the ability to view higher resolution input on the 640 by 480 display. In other words, SV-6 can accept an 800 by 600 and higher input and reformat the information to be displayed on the viewer. This is a critical feature when the software in the laptop does not conform to 640 by 480 resolutions.

Figure 3.3: SV-6 PC viewer

3.2.4. Mini-Joystick

A mini-joystick, called Minipoint from *Varatouch Technologies Inc. (VTI)*, was epoxied to the top of the ultrasound transducer to make distance and area measurements and to zoom the ultrasound images. It provides an USB interface for connection to the PC. Its operating voltage range from 2V to 35V and it can move in 360 degree directions. The hardware supports digital wake-up, i.e., it is able to wake-up the computer from a standby state. It works on VTI's proprietary R2 Resistive Rubber technology. This works by using a special material called Resistive RubberTM, conductive rubber over a resistive coating. This rubber acts as a 'wiper' over the resistive coating. By giving the conductive rubber also a specific resistance, VTI eliminates the resistive coatings on the printed circuit board. A picture of Minipoint is shown in Figure 3.4.

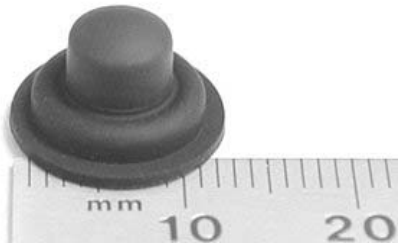


Figure 3.4: Minipoint

3.2.5. Voice Activation

The voice activation of the ultrasound scanner requires three different elements: 1) Speech recognition engine software, 2) Microphone, 3) Voice command handler software. The message flow for voice activation of the scanner is shown in Figure 3.5.

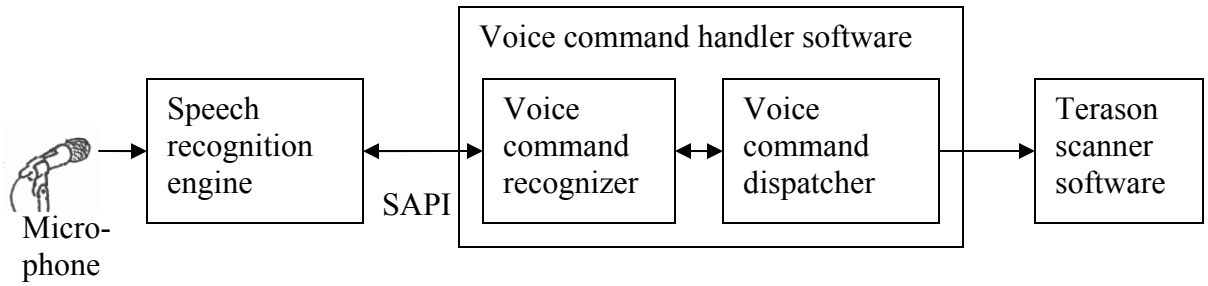


Figure 3.5: Voice activation of the scanner system

The primary speech recognition engines used for this setup are Microsoft command and control speech recognition engine, Vocon 3200 and IBM Viavoice. These engines and their features will be described in greater detail in the later chapters of this thesis. Its performance with the various microphones will be analyzed at different noise levels, and an ideal combination of microphone and speech recognition engine suitable under high ambient noise will be selected. Different types of microphones are also tested. The voice command handler software consists of two modules, the voice command recognizer and the voice command dispatcher. This software exists as a separate application in the laptop, since the functionality contained in this software cannot be integrated into the Terason scanner software due to its proprietary nature. The voice command recognizer module recognizes the command that the user has spoken. This module uses the *Speech Application Program Interfaces* (SAPI) to communicate with the engine and gets the recognition result. As the SAPI used is specific for each engine, the implementation of this module changes when the speech recognition engine is changed. More details regarding the implementation of this module is given in Chapter 4. The voice command dispatcher module receives the recognition result and dispatches this result to the Terason ultrasound scanner software. The nature of this communication is described in detail in Section 3.3. The

implementation of this module is independent of the engine used and is described in detail in Chapter 5.

3.2.6. Microphones

Different types of microphones are selected and tests are carried out in order to choose the one which gives the best performance under moderately high levels of ambient noise. The selected microphones include conventional airborne microphone recommended for speech recognition with good noise-canceling features, physiological sensor which picks up sound from the vibration of the vocal cords captured from the skin surface of the throat, jaw bone conduction microphone which detects speech from the vibrations coupled via the jawbone, a 4-element array microphone developed using adaptive beamforming techniques and a 2-element array microphone which works using bluetooth. Each of the selected microphones has their unique ways of implementing noise-cancellation features. The tests performed using them and the results obtained are presented in Chapter 8.

3.2.7. Establishing the Wireless Connection

A wireless network is set up between the Sony VAIO and another laptop located further away for the transfer of ultrasound images. This network is configured to operate in *ad-hoc* mode. A connection is established between the laptops in this ad-hoc wireless network using the software *Netmeeting* from Microsoft. When Sony VAIO is switched on, logon scripts are executed which establishes the connection with the remote laptop using Netmeeting. The feature in Netmeeting to share the desktop with another laptop in the network is activated. This enables the real time

transmission of the ultrasound images displayed on Sony VAIO's monitor to a remote laptop's monitor.

3.2.8. Logon Scripts

When Sony VAIO is switched on, logon scripts are scheduled to be executed. These scripts are responsible for setting up of wireless connection with the remote laptop, starting up of the voice command handler software and the Terason 2000 ultrasound scanner software.

3.2.9. Power Management

The laptop (Sony VAIO) can be put into low power (sleep) states when the user is not actively scanning. Terason scanner goes into low power consuming mode when the menu item 'Freeze' is invoked on the scanner software window. The scanner system can be put into two different sleep states by means of voice commands. One is the *standby state*, where the laptop consumes around 5 watts of power. When the user speaks the voice command for standby state, the voice command handler software recognizes it with the help of the speech recognition engine. It then dispatches the command 'Freeze' to the ultrasound scanner software to freeze the image on the screen. This puts the scanner into a low power mode. Finally, the voice command handler software communicates to the operating system to put the laptop into a standby state. The same sequence of operations also takes place for the second sleep state, the *hibernate state*. In this state, the laptop consumes only a few milliwatts of power. The laptop and the scanner are put back into their active modes by using the laptop's power switch, located on the outside of the laptop. More information regarding power management and its implementation will be described in Chapter 6.

3.3. Speech User Interface Design

It is desirable to manage most ultrasound scanning controls through voice commands. A command and control speech recognition engine is chosen to implement this functionality. The voice commands used for controlling the scanner are chosen to be short in length, such as two or three words. However, the selection of single-word commands is avoided due to the higher possibilities of incorrect recognitions in a noisy environment. The speech recognition engine recognizes these commands from a small fixed set of vocabulary to enhance the recognition accuracy. This is done by utilizing the context-free-grammar features of the speech recognition engine.

The speech interface is implemented such that a user can operate the Terason system with voice commands the same way he/she would operate it using a mouse. In other words, the user can initially speak commands for any of the main menu items in the Terason window. Activation of one of the main menus will bring its entire submenu into view. In this way, the user can trace through the entire menu hierarchy of the Terason window, being able to perform all scanning controls.

As discussed in Section 3.2.5, the scanner is voice-activated by implementing the voice command handler software. The voice command recognizer module of this software uses *Speech Application Program Interfaces* (SAPI) for communication with the speech recognition engine. As the scanner software is proprietary and not accessible for direct speech integration, the problem of implementing all scanner controls through voice-activation is solved with the help of

the voice command dispatcher module. All the scanner software menu items can be activated using specific keystrokes in addition to mouse clicks. A given recognized voice command is translated into the respective sequence of keystrokes by the voice command dispatcher module after making the ultrasound software application the active window in the laptop. The communication between the voice command dispatcher and the scanner software is described in Figure 3.6.

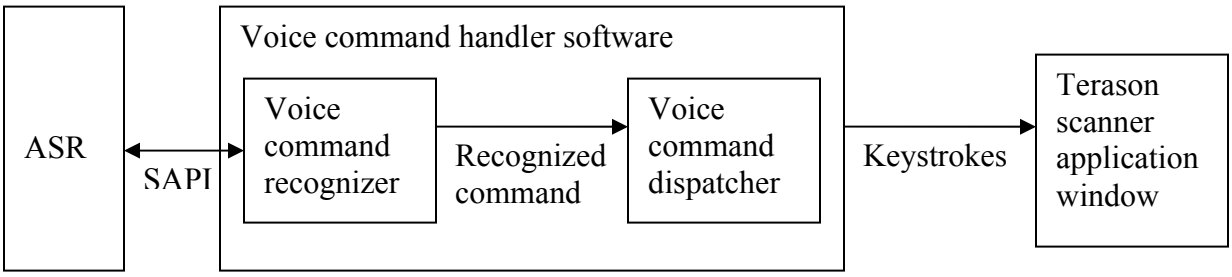


Figure 3.6: Dispatching commands to the Terason scanner application

The speech recognition engine supports context-free grammar, which allows us to specify the voice commands that are allowed to be recognized at any time. The context-free grammar is a file where different rules can be specified, and the rules can be made active or inactive at runtime. Each rule indicates the different commands that can be recognized when that rule is active. to All the menu items of the Terason software application are categorized into different sets and each set added as a rule in the context-free-grammar of the speech recognition engine. By default, only the rule containing the main menu items is kept active in the grammar. The different rules are then made active and inactive depending on the menu item that is invoked at a time. Thus the vocabulary of the speech engine is dynamically updated.

Microsoft's command and control speech recognition engine, *Vocon 3200* and IBM Viavoice from Scansoft are the primary engines used for the recognition of the voice commands. The SAPI selected for use is dependent on the chosen engine. The voice command handler software is implemented in C++.

Chapter 4. Speech Recognition Interface Development

This chapter deals with the techniques and software interfaces that are needed in order to develop speech recognition applications. Here we discuss three different speech recognition engines and their *Speech Application Program Interfaces* (SAPIs). The SAPIs are software routines that allow software developers to integrate the services of the speech recognition engine with their applications. These routines are specific to each speech recognition engine and are defined by the engine manufacturer. They are also valuable for the development of different tools aimed for analyzing the performance of an engine. The voice command recognizer software module, discussed briefly in Section 3.2.5, is implemented based on the principles described in this chapter.

4.1. Microsoft Command and Control Speech Recognition Engine

Microsoft speech recognition engine, version 5.1 is a small vocabulary engine, particularly suited for command and control applications. It is speaker-independent, but gives better recognition rates with the usage of user-specific speaker profiles (see Section 4.1.1). It offers context free grammar features, which enables us to specify a command list so that the engine recognizes commands from that list only. This engine supports Microsoft Speech Application Program Interface 5.1, for speech recognition application developers to incorporate the speech recognition engine features to their applications. This section explains each of the above-mentioned features in greater depth. The engine software is open-source and comes with a wide variety of tools

illustrating the working of Microsoft SAPI 5.1. The engine is also small in size and occupies only around 3 MB.

4.1.1. Creation of Speaker Profiles

One important fact which increases the recognition rate is the creation of user-specific speaker profiles. Each speaker who wishes to test the automatic speech recognition (ASR) features, must first train the engine for a short period of time. While the speaker speaks, the engine collects necessary information to build up its internal data models. It uses this internal data to come up with better results when the same speaker tests the engine afterwards.

The creation of a user-specific profile starts with the adjustment of microphone recording volume. This process is called *audio tuning*. During this step, the engine determines the suitable recording volume required for the speaker's sound level so that a good recognition output is obtained. This is followed by *training*, where the speaker has to read out passages for around 10 minutes while the engine collects user-data files. Before the speaker tests the engine, he/she has to set his/her speaker profile as the active profile for the engine. The option for creating user-specific profiles can be found under Programs->Control Panel->Speech, once the Microsoft SR engine is successfully installed in the computer.

4.1.2. Context Free Grammar Format

Context free grammar comes into play when we need the SR engine to recognize from only a specified command set. In Microsoft SAPI 5.1, these grammar files are edited in Extensible

Markup Language (XML) format and saved as a **.xml* file in the computer. This file is then loaded at run time by the speech-enabled application, compiled into a binary grammar format and then used by SAPI for performing the actual recognition.

4.1.2.1. Extensible Markup Language (XML)

The context free grammar in Microsoft SAPI 5.1 is an application of XML. Every XML element consists of a start tag (<SOME_TAG>) and an end tag (</SOME_TAG>) with a case-insensitive tag name and contents between these tags. The start tag and the end tag are the same if the element is empty, eg., the tag (<SOME_TAG/>). The attributes of an XML element appear inside the start tag. Each attribute is in the form of a name followed by an equal sign followed by a string which must be surrounded by either single or double quotation marks. An attribute of a given name may only appear once in a start tag. The contents of an element consist of text or sub-elements. The comments in an XML file must be enclosed between <!--and -->. For example, an XML format may look like:

```
<SOME_TAG   ATTRIBUTE=”a”>
        <Text> Hello! </Text>
        <SUB_TAG>
              …..
        </SUBTAG>
</SOME_TAG>
```

4.1.2.2. Microsoft SAPI 5.1 Grammar Format

The contents of the grammar file in Microsoft SAPI 5.1 must be enclosed between a start tag, <GRAMMAR> and an end tag, </GRAMMAR>, denoting the grammar XML element. The

grammar XML element has an attribute, called LANGID which must be a numerical value. For example, the LANGID has a value of 409 to indicate 'American English'.

The different words in the vocabulary are categorized into different *rules*. Each rule begins with the start tag <RULE> and ends with the end tag </RULE>. A *rule* may reference another *sub-rule*, which may have reference to still another *sub-rule*. A *rule* has an attribute TOPLEVEL that indicates whether the rule is enabled for speech recognition or not. The *rules* can be activated and deactivated during run time. In this way, the vocabulary of the speech engine can be changed dynamically. A reference to another rule starts with the start tag <RULEREF> and the end tag </RULEREF>. The list of phrases or rules that can be recognized are enclosed inside the tags <LIST> and </LIST> (or <L> and </L>). The <PHRASE> and </PHRASE> (or <P> and </P>) tags are used for specifying text to be recognized by the speech recognition engine. The <OPT> and </OPT> (or <O> and </O>) tags are used for specifying optional text in a command phrase. An example of a Microsoft SAPI 5.1 grammar is given below:

```
<GRAMMAR LANGID="409">
  <RULE NAME="test_phrase" TOPLEVEL="ACTIVE">
     <L>
       <RULEREF NAME="bmode"/>
        <P>freeze</P>
        <P>exit</P>
     </L>
  </RULE>

   <RULE NAME="bmode">
      <P>b mode</P>
      <O>
```

```
        <RULEREF NAME="bmode_controls"/>
      </O>
    </RULE>

    <RULE NAME="bmode_controls">
      <L>
        <P>reduce contrast</P>
        <P>increase contrast</P>
      </L>
    </RULE>
  </GRAMMAR>
```

For example, if the above grammar is loaded to a speech engine, it would recognize the following commands:

b mode
b mode reduce contrast
b mode increase contrast
freeze
exit

The grammar file used with Microsoft Command and Control for implementing the different commands in the ultrasound scanner system is given in Appendix A.

4.1.3. Using Microsoft SAPI 5.1

Microsoft Speech Application Program Interface (SAPI) 5.1 is a software layer used by speech enabled applications to communicate with Speech Recognition engines and Text-to-Speech (TTS) engines. An application could be any kind of tool that can use the speech recognition capability of a speech recognition engine. Figure 4.1 describes the SAPI architecture.

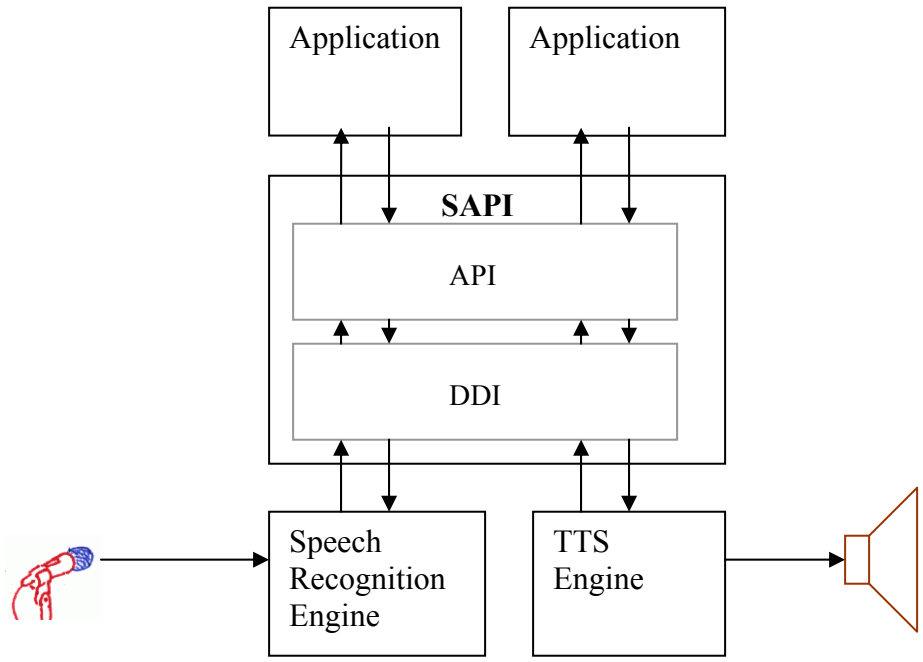


Figure 4.1: SAPI architecture

SAPI provides two different sets of software routines. One set is for use by applications that need services from the ASR or TTS engines. This set is called the *Application Programming Interface* (API). The other set is for use by SAPI to communicate with the ASR and TTS engines. This set is called *Device Driver Interface* (DDI). The API and DDI together enable the successful communication between the application and the ASR or TTS engines. With this architecture, the inner functioning of the ASR engine is concealed from the application by means of the API layer of SAPI. The recognition engine and the Text-to-Speech (TTS) engine interact directly with the sound card's audio input and output.

SAPI works on Component Object model (COM) architecture. The Component Object Model is a software architecture that allows applications to be built from binary software components [9].

SAPI provides API for applications in the form of COM interfaces. An application can use a programming language such as C++ for implementing the different APIs provided by SAPI.

4.1.3.1. Basic Speech Recognition

Figure 4.2 illustrates the sequences of steps that an application must implement using Microsoft SAPI 5.1, in order to enable speech recognition.

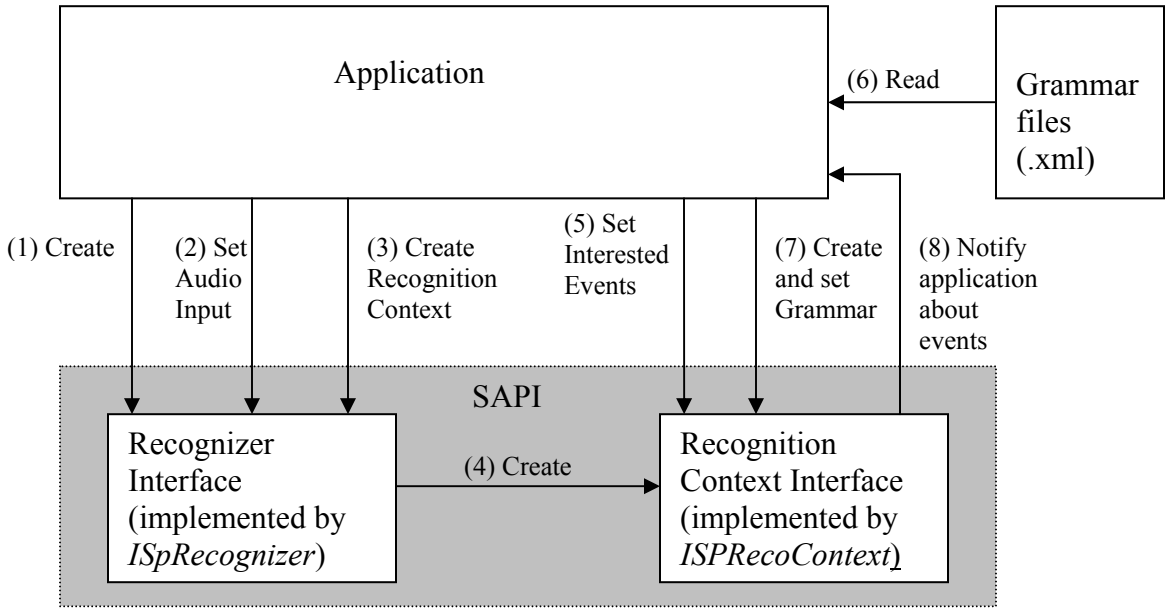


Figure 4.2: Control flow diagram

The SAPI provides a Recognizer interface named *ISpRecognizer,* which provides applications with software functions to control the different properties of the ASR engine. Some examples are setting a specific speaker profile, specifying the audio input, changing the status of the recognition engine etc. Each *ISpRecognizer* interface represents a single ASR engine. The

application should first create a Recognizer interface (*ISpRecognizer),* which can be used to setup the required audio input and the speaker profile.

There are two implementations of the *ISpRecognizer* in SAPI. One implementation is "in-process", where the ASR engine is created in the same *process* as the application (The term *process* here implies a stand-alone software program which has its own memory, stack and address space). In this case, only this application can connect to this recognizer. The other implementation is the "shared-recognizer", where the ASR engine is created in a separate *process*. There will only be one shared engine running on a system, and all applications using the shared engine connect to the same recognizer. This allows several speech applications to work simultaneously, and allows the user to speak to any application, as recognition is done from the grammars of all applications.

The *ISPRecognizer* provides interfaces to create a Recognition Context. The Recognition Context (*ISPRecoContext*) serves as the main interface for speech recognition. The application informs the Recognition Context about all the events it is interested in. The examples of some events are SPEI_START_SR_STREAM (which indicates the starting of an audio stream), SPEI_END_SR_STREAM (which indicates the end of audio stream), SPEI_HYPOTHESIS (an event indicating a hypothesis), SPEI_RECOGNITION (the recognized event) etc. The application should then indicate to the Recognition Context which grammar it is intending to use. The recognitions come back to the application by means of a notification mechanism. The events arrive in the application in the form of a *SpEvent* structure, from which the application can determine what was recognized.

An *ISpRecognizer* can have multiple *ISpRecoContext*s associated with it. Similarly, an *ISPRecoContext* can have multiple grammars associated with it. All these would enable the recognizer to recognize a wide variety of utterances. The speech engine accepts the microphone as the default audio input. It can also accept sound files (.wav) as audio inputs, which is found to be very useful for testing purposes. The preferred wav file format for Microsoft ASR engine is 22KHz 16 bit Mono.

The important steps required for the basic speech recognition to work and the respective APIs involved are given below. This list adds further details to the sequence of steps shown in Figure 4.2.

1. Initialize COM (::CoInitialize).

2. Create an ISpRecognizer interface (ISpRecognizer::CoCreateInstance(CLS_ID). The CLSID may take either of the values, CLSID_SpSharedRecognizer or CLSID_SpInprocRecognizer.

3. Set up the audio input. (ISpRecognizer::SetInput(*Input*)). When using a shared recognizer, the audio input is automatically set. However, ISpRecognizer::SetInput may be called with NULL parameter as *Input* to force the recognizer to recheck the default audio input. But for an in-process recognizer, this function has to be called in order for a non-NULL *Input* parameter to setup and start-up the audio input stream.

4. Create an ISpRecoContext interface (ISpRecognizer:: CreateRecoContext).

5. Associate the recognition messages, WM_RECOEVENT, to come to the specified window (ISpRecoContext::SetNotifyWindowMessage). It is through these messages SAPI communicates with the user's application.

6. When user's application receives the recognition message WM_RECOEVENT, it is an indication of an event that has occurred. Examples of such events are sound start, sound end,

recognition, hypothesis etc. The application can set up a list of the events it is interested in. This is done by the function ISpRecoContext::SetInterest.

7. Create an ISpRecoGrammar interface (ISpRecoContext::CreateGrammar).

8. Load the context free grammar from a file, (ISpRecoGrammar::LoadCmdFromFile (*filename*, *Options*)). The file can either be a compiled or uncompiled grammar file. To modify the rules of the grammar after it has been loaded, specify SPLO_DYNAMIC for the *Options* parameter, otherwise specify the SPLO_STATIC flag.

9. Finally the recognition can be made active by calling ISpRecognizer::SetActiveState.

10. The application should write its own function to handle the different events notified by the WM_RECOEVENT message.

11. When recognition is done, release the ISpRecoGrammar interface.

12. Release the ISpRecoContext interface.

13. Release the ISpRecognizer interface.

14. Uninitialize COM (::CoUninitialize).

More details regarding the different APIs in Microsoft SAPI 5.1 can be found in [10].

4.2. Dragon English NaturallySpeaking & IBM Viavoice Dictation

The Dragon English NaturallySpeaking and IBM Viavoice dictation are speech recognition engines tailored for dictation applications. But they can be made to work for command and control applications by making use of the Speech Application Program Interfaces that it supports. The SAPI that both engines support is Microsoft SAPI 4.0. The support for context free grammar can be achieved by using the APIs in Microsoft SAPI 4.0. Speaker profiles have to be created for better recognition rates. As both the engines support the same speech API, the development

issues that are discussed here for command and control applications are relevant to both of them. These engines have been selected for our purpose to get performance comparisons with the other command and control speech recognition engines. They both are large vocabulary engines and consume significant amount of disk space.

4.2.1. Basic Speech Recognition

Microsoft SAPI 4.0 and Microsoft SAPI 5.1, although provided by the same company *Microsoft*, are functionally very different. SAPI 4.0 is more complex compared to SAPI 5.1, and programming with it involves more work. The APIs provided by SAPI 4.0 are not as efficient as SAPI 5.1 and therefore most of the programming tasks to accomplish a specific functionality are left to the programmer. This section will not go into the details of SAPI 4.0 APIs. Interested readers may refer to [11]. However, an architectural overview will be given to get an idea regarding the different processes involved. SAPI 4.0 also uses COM interfaces, and an application can use C++ for development with the APIs. Sample applications which implement SAPI 4.0 are enclosed in the attached CD.

Here's how the process works:

1. The application sets the desired audio input for the ASR engine and creates an audio-source object through which the engine acquires the data. The audio source can either be a microphone or a sound file.

2. The application, through a speech recognition enumerator object, locates a speech recognition engine that it wants to use. It then creates an instance of the engine object and passes it the audio-source object.

3. The engine object has a dialog with the audio-source object to find a common data format for the digital audio data, such as pulse code modulation (PCM). Once a suitable format is established, the engine creates an audio-source notification handler object that it passes to the audio-source object. From then on, the audio-source object submits digital audio data to the engine through the notification handler object.

4. The application can then register a main notification handler object that receives grammar-independent notifications, such as whether or not the user is speaking.

5. When it is ready, the application creates one or more grammar objects. These objects use the context free grammar provided by the user.

6. To determine what word the user has spoken, the application creates a grammar-notification sink for every grammar object. When the grammar object recognizes a word or phrase, or has other grammar-specific information for the application, it calls functions in the grammar-notification sinks. The application responds to the notifications and takes whatever actions are necessary.

7. Typically, when a grammar object recognizes speech, it sends the grammar-notification sink a string indicating what was spoken. However, the engine may know a lot more information than this, such as alternative phrases that may have been spoken, timing, or even who is speaking the

phrase. An application can find this information by requesting results object for the phrase and interrogating the results object.

4.2.2. Microsoft SAPI 5.1 versus SAPI 4.0

Microsoft SAPI 5.1 provides simpler and easier interfaces compared to SAPI 4.0, to incorporate speech recognition into a given application. In other words, implementation using SAPI 4.0 requires more programming than that using SAPI 5.1. Although command and control applications were built using SAPI 4.0 for Dragon English NaturallySpeaking and IBM Viavoice dictation ASR engines, there were other problems. While running these applications with the audio input of the ASR engine set as directly from the microphone, it was found that the ASR engine unexpectedly switches its mode of operation from command and control to dictation. The reason for this unexpected change in behavior is still unknown.

4.3. Vocon 3200

Some of the relevant features of Vocon 3200 speech recognition engine, developed by *Scansoft Corporation*, are:

- phoneme based continuous speech recognition
- speaker-independent
- supports context free grammar
- grammar can be updated at run time
- small vocabulary
- small in size, around 1 MB

- resistance to high noise levels
- exception dictionaries to override the internal phonetic transcriptions of words.
- trained user words can be added to an exception dictionary

These features make it a valid choice for our purpose. The speech APIs used for interface with the speech recognition engine is also developed by *Scansoft*. Unlike the other engines discussed earlier, this engine does not support the creation of speaker profiles to enhance the recognition accuracy rate for a particular speaker. But, instead each speaker can create a recording which can be used to create an exception dictionary whose contents would override the internal phonetic transcription of the words during recognition.

4.3.1. Context Free Grammar Format

The core of the grammar consists of a set of grammar rules. Each rule describes an allowable structure and has a name. An example of a grammar rule may be:

<date> : the <day> of <month> <year> ;

Here, <date>, <day>, <month>, <year> all represent rules. <day>, <month> and <year> must be defined elsewhere. The grammar format uses a C-like syntax:

- grammars are composed of statements terminated by a semicolon (;)
- C++ style commenting (both /* */ and //) is supported
- Parentheses (‘( )’) can be used to group expressions together.

A typical grammar would look like:

!grammar ultrasound;
!language “American English”;

!start <Speech>;

<Speech>: Terason <commands>;

<commands> : testing | view | image;

The directive *!grammar* is followed by the grammar name. The directive *!language* specifies the language in which the grammar is written. The start statement, specified by the directive *!start*, specifies the entry point into the grammar. They specify what can be recognized. Here it specifies that the rule <*Speech*> can be recognized. The rule <*Speech*> defines that the commands all begin with the prefix *Terason*. The rule <commands> specifies all the different alternatives that can be spoken. Therefore, the above grammar allows the speaker to speak the following commands:

Terason testing

Terason view

Terason image

The directive *!activatable* is used to specify which rules can be dynamically activated and deactivated. The activation and the deactivation of the rules can then be controlled by speech APIs during runtime. This directive will come into useful when more complex grammars are designed. The grammar in text form should be stored in a file with extension *.bnf*. This grammar can be compiled into a binary format with extension *.grm* before it is used by the speech recognition engine. This can either be done by using the Grammar compiler tool provided with the engine or by using speech APIs during run time. The grammar file used with Vocon 3200 for implementing the different commands in the ultrasound scanner system is given in Appendix C.

4.3.2. Basic Speech Recognition

The steps required for the basic speech recognition to work and the respective APIs that are involved for accomplishing this purpose are given below:

1. Initialize the Speech API (lhs_Initialize).

2. Open and obtain a handle to the binary grammar. If the grammar is available only in text form, it has to be first compiled into a binary form. But if the precompiled binary grammar already exists, it can be opened by calling lhs_asrGrmOpenFromBinary.

3. Convert the grammar to a context using lhs_asrCtxOpenFromGrammarHandles. A context can be created using a single grammar or several grammars can be combined to form a context.

4. Open a recognizer (lhs_asrRecOpen).

5. Load the context on the engine (lhs_asrRecLoadContext). The engine can only have one context loaded at a time.

6. Open the audio acquisition channel on the engine (lhs_asrRecAcqOpen).

7. Tell the engine to start accepting audio samples (lhs_asrRecStart). This function is also used to register the callback function(s) that the engine will call to return results, handle signal conditions, etc.

8. Provide the engine with audio input buffers (lhs_asrRecAcquisit). This sample copies the entire contents of the sound file into a buffer and passes it to the engine in one call.

9. Tell the engine to stop listening and return a recognition result (lhs_asrRecStop).

10. Retrieve the session data (lhs_asrRecGetSessionData) and save it to a file. This file represents the information learnt by the engine during recognition. The user can decide whether the engine should use this file for subsequent recognitions.

11. Close the audio acquisition channel (lhs_asrRecAcqClose).

12. Unload the context from the engine (lhs_asrRecUnloadContext).

13. Close the instance of the recognition engine (lhs_asrRecClose).

14. Close and release the handles to the context (lhs_asrCtxClose) and to the binary grammar (lhs_asrGrmClose).

15. Uninitialize the Speech API (lhs_UnInitialize).

The application also must register an ASR_CBRESULT callback function that the engine calls when it is ready to return a recognition result.

4.3.3. Creating User Word Dictionaries

User words are speaker-dependent words. The pronunciation associated to them is specific to one user. This data is collected by the engine using a process called user word training, when each user trains a word by repeating it one or two times. The binary representation of such a speaker-dependent pronunciation is also called a *user-word transcription*. The user-word transcription is retrieved and added to an exception dictionary, which can be saved on the disk. This dictionary is given as input while compiling the grammar and the resulting output is stored onto a binary grammar file. This grammar is then converted onto a context that is loaded on the engine for use during recognition. User words typically result in better recognition if spoken by the speaker who trained them. The different steps involved and the APIs that can be used is listed below. For more details, please refer to [19].

1. Initialize the Speech API (lhs_Initialize).

2. Open a recognizer (lhs_asrRecOpen).

3. Open the audio acquisition channel (lhs_asrRecAcqOpen).

4. Initialize the user word training algorithms (lhs_asrRecUswInit).

5. Tell the engine to start the user word training process (lhs_asrRecUswStart). This function is also used to register the callback function that the engine will call to return training status, handle signal conditions, etc.

6. Provide the engine with audio input (lhs_asrRecUswAcquisit). This sample copies the entire contents of each training utterance into a buffer and passes each utterance to the engine in one call. One or two training utterances can be used for every word.

7. Force the engine to stop listening if there is not enough trailing silence detected in the training utterance (lhs_asrRecUswStop).

8. Tell the engine whether to accept or reject the previous training utterance and whether further training utterances will be sent (lhs_asrRecUswContinue). This is useful when the application or the user decides that the training utterance needs to be redone for some reason.

9. Retrieve the transcription of the user word (lhs_asrRecUswGetTranscription).

10. Tell the engine that the user word training is complete (lhs_asrRecUswTerminate).

11. Create and obtain a handle to a new dictionary (lhs_dctCreate). This sample creates a new dictionary. Alternately an existing dictionary can be opened by calling lhs_dctOpenFromBuffer.

12. Add the new user word transcription to the dictionary (lhs_dctAddWord).

13. Release the buffer for the user word transcription (lhs_asrRecUswFreeTranscription).

14. Open and obtain a handle to the compiler object (asrCplOpen).

15. Set list of dictionaries on the compiler object (lhs_asrCplSetDictionariesFromHandles).

16. Compile the grammar using the new dictionary containing the user word transcription (lhs_asrGrmOpenFromText). This function returns a handle to the binary compiled grammar.

17. Add an alternative to the grammar (lhs_asrGrmAddAlternative) to act as a placeholder for the user word in the grammar.

18. Optionally save the grammar to a buffer (lhs_asrGrmSaveToBuffer) for later archiving or immediately use it to load a context and run recognition.

19. Close the dictionary and release the handle (lhs_dctClose).

20. Close the binary grammar and release the handle (lhs_asrGrmClose).

21. Close the compiler object and release the handle (lhs_asrCplClose).

Chapter 5. Implementation of Voice Command Dispatcher

This chapter explains the architecture and design of the *voice command dispatcher* software for Terason ultrasound scanner system. The functionality of this software module was discussed briefly in Section 3.3. This module acts as the interface between the *voice command recognizer* software and the Terason ultrasound scanner software application window. It translates the recognized command into a corresponding sequence of keystrokes, after making the Terason window active on the desktop. This will execute the required menu item at the Terason window. This software is also responsible for updating the engine's grammar at run time depending on the command that the user spoke at any time.

As the objective is to implement all scanner operations through the activation of the different menu items on the Terason window, the number of voice commands that has to be handled by this module is fairly high, around 100. This necessitates the need for a structured implementation. It is made possible by applying object oriented design principles using C++.

The object oriented design knowledge to be applied for the creation of a robust and reusable software package is covered in Section 5.1. The design patterns used for the implementation of the Voice Commander software is discussed in Section 5.1.3. Finally, the design and implementation of the voice command dispatcher is discussed in detail in Sections 5.2 and 5.3, with the help of *class* and *activity diagrams*.

5.1. Design Issues

5.1.1. Definitions

In C++, a ‘class’ is a logical method to organize data and functions in the same structure. The functionality of ‘class’ is similar to that of the C keyword ‘struct’, but with the possibility of including functions as members, instead of only data. The data that a ‘class’ has to keep track is called a data member, and the operations that the ‘class’ performs on its data are called member functions. A ‘class’ is only a definition, but the ‘instance of a class’ is the actual memory that gets created. We can have multiple ‘instances of a class’ in the same program. The term ‘object’, refers to an ‘instance of a class’.

The data members or functions in a C++ class can either be *public*, *private* or *protected*. It is recommended to keep the data members as private or protected. If a data member or function of a class is private, then only the functions of that class can access them. If a data member or function is protected, then the functions of that class and all its derived classes can access them. If it is public, then any external function or object can access them by creating an object of the class.

5.1.2. Design Process

A good design is an essential requirement for the creation of a robust and reusable software package. A good design helps to add more functionalities to the package with little effort and time. It also helps others to master its implementation with ease. The steps involved in the design process are listed below:

- The different independent units involved in the execution of the software should be identified first. The tasks of each of these units and the communication between the units also need to be defined.
- Identify those design patterns (Sections 5.1.3) that can be applied to these independent units, which would make the communication between the units more structured.
- Identify the classes that can be assigned to these independent units and the class relationships it should follow
- Assign functionalities to each class.

5.1.3. Design Patterns

Patterns are techniques by which people reuse successful software design practices. They are design methods used by experts to solve specific problems in their design, and then documented to prevent the occurrence of these problems again. The *design patterns* outlined in [12] have been very popular in object oriented design over many years. The design patterns used for the design of the Voice Commander software are outlined below. More details of this can be found in [12].

5.1.3.1. Singleton

The *Singleton* pattern applies to the many situations in which there needs to be only a single instance of a class, i.e., a single object. When implemented as the pattern specifies, the class will have direct control over how many instances of itself that can be created. This is in contrast to making the programmer responsible for ensuring that there is only one instance. A second client class will always be involved to take or make requests from the Singleton class.

Here is how a generic Singleton class is written in C++. More details regarding this can be found in [12].

```
class Singleton {
public:
   static Singleton* Instance();
protected:
   Singleton();
private:
   static Singleton* _singleton;
};
```

The data member *_singleton* in the class Singleton is meant to point to the one and only instance of the class Singleton in the program. Note that this data member is made *static*, so that it is not destructed unless and until the program quits. The constructor of this class (*Singleton())* is made protected, so that it can be invoked by only this class or its derived classes. The function *Instance()* creates the single instance of this class and provides it to any client class which invokes it. A typical implementation of the function, *Instance()* will be like as below:

```
Singleton* Singleton::_singleton = 0;
Singleton* Singleton::Instance()
{
  if (_singleton == 0)
  {
    _singleton = new Singleton();
  }
  return _singleton;
}
```

An example of this design pattern in Voice command dispatcher software is the class CommandTracker (Section 5.2.2).

5.1.3.2. Command

A *Command* pattern is applied when an application receives different requests and needs to handle them independently. In this case, a request can be implemented in the form of an object. The application instantiates (creates an 'instance' of) a specific object based on the received request, and the object itself will have the knowledge on how to handle the request. In this way, the Command pattern decouples the object that invoked the operation from the one having the knowledge to perform the operation.

The key to this pattern is an *abstract* Command class (an *abstract* class cannot be instantiated), which declares an interface for executing operations. In the simplest form, this interface includes an *abstract* Execute() operation (a function becomes *abstract* if the base class do not provide implementation, but the derived classes implement it). The Command class itself may not have the knowledge to implement this operation. Every subclass of Command will override the Execute() operation to handle their own specialized way of implementing the request. As an example, menus are generally implemented with Command objects. Each choice in a Menu is an instance of a MenuItem class (say, a subclass of *Command*). When the user selects a menu item, the MenuItem::Execute() is called to carry out the operation. The execution of the different voice commands in the voice command dispatcher software follows the Command design pattern. For more details, refer to [12].

5.1.3.3. Factory Method

This pattern defines an interface for creating objects within a class, but let subclasses decide which objects to be created. Factory Method lets a class defer instantiation to subclasses.

For example, in the voice command dispatcher software, the class Command is an abstract class. It has the interface function Command::spawn() which acts as a Factory method. The abstract class Command itself does not have the knowledge on how to implement this function. All the subclasses inherited from Command should override this function and provide their own specific implementation.

5.1.4. Terason Application's Menu classification

The menu/menu items of Terason application can be closely analyzed and divided into two categories:

- A Container item - This item is represented by those menu/menu items which add more menu items/actions that can be executed. A pull down menu, which on execution brings other menu items into view, can be put into this category. A menu item which invokes a dialog box on execution can also be put into this category. This is because we may need to perform some actions on the dialog box, after this menu item is executed.

  E.g.: The pull down menu *'Modes'* come under this category. The menu item *'Open'* also come under this category as it invokes a dialog box, and a new voice command like *'Cancel'* may have to be given in order to close the dialog box.
- A non-Container item - This item is represented by a menu item which executes its operation and then brings the application to its default state. By default state, it means that the application's main menus only are active. For example, the menu item *'bmode'* comes under this category.

5.2. Architecture

The architecture of the Voice Command dispatcher software is shown in Figure 5.1. The notations follow from Unified Modeling Language (UML) [14].

represents a class

represents a package

represents inheritance

represents instantiation

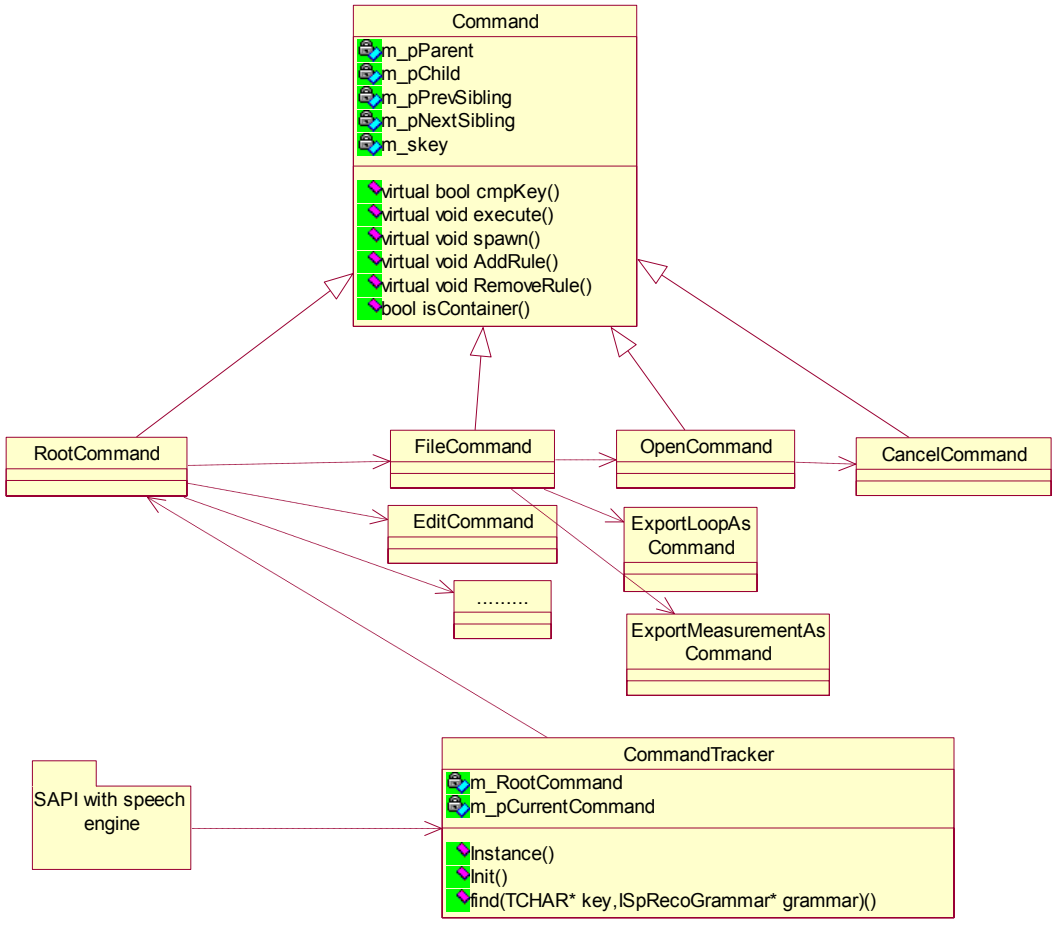


Figure 5.1: Architecture of Voice Commander software

Below, we will go through each class and package in greater detail.

5.2.1. SAPI with Speech Engine

This software package is represented by the *Voice command recognizer* module, which makes use of the *Speech Application Program Interface* (SAPI) and the speech recognition engine to recognize the voice commands.

This package creates the one and only instance of the class CommandTracker. During startup, it invokes CommandTracker::Init(..) for initialization of the system. Once a voice command is recognized, then it calls CommandTracker::find(..) to execute that specific voice command. Thus the CommandTracker class acts as the main interface class with the SAPI. The message flows that occur during initialization and recognition are given in Figure 5.2 and Figure 5.3.

5.2.2. Class CommandTracker

CommandTracker is a Singleton class, i.e., only one object of it can exist in the program. CommandTracker has the following main interface functions:

static CommandTracker Instance();*

This function is mandatory for a Singleton pattern (Section 5.1.3.1).

void Init();

initializes the Voice Commander software. This is called when SAPI starts up.

bool find(TCHAR key,CComPtr<ISpRecoGrammar> grammar);*

This function is called by the voice command recognizer after it recognizes a voice command. The first parameter *key* represents the recognized voice command. The second parameter *grammar* is a pointer to the speech recognition grammar, which may have to be updated once the command execution is performed.

CommandTracker has the following attributes:

Command m_pRootCommand*

points to the root of the command tree.

Command m_pCurrentCommand*

points to the current command that was executed if the command was a container. If the command is not a container, then *m_pCurrentCommand* is made equal to *m_pRootCommand*. By default, this attribute is set to *m_pRootCommand.*

5.2.3. Class Command

This class represents the base class of all commands. A pull-down menu/menu item can be implemented by subclassing the *abstract* class Command. The class Command only specifies the basic functions that need to be implemented and the attributes that needs to be present in order to implement a menu/menu item. This class follows the design pattern in Section 5.1.3.2.

Each Command stores its own key, which is the name for the menu/menu item. This name need not necessarily be the actual name of the menu/menu item in Terason application. It can be the name by which a user would like to invoke this menu item through voice.

To implement a new menu/menu item, you need to inherit from the base class Command, and then implement the required functions. But most of the functionalities common to all menus/menu items will be already present in the Command class itself.

The functions that are required to be implemented for a new menu/menu item to work properly are:

virtual void execute(TCHAR key, HWND hwnd) = 0*

specifies how the menu item is to be executed

virtual void spawn()

creates all the Child Commands this particular Command has. If this Command itself represents a non-container, then this function do not do anything

virtual void AddRule(CComPtr<ISpRecoGrammar>)

activates the rule for this Command in the speech recognition grammar

virtual void RemoveRule(CComPtr<ISpRecoGrammar>)

deactivates the rule corresponding to this Command from the speech recognition grammar

Other functions provided by the base class Command are:

Command(TCHAR key, bool isContainer);*

Constructor of this class. The voice key for a particular *Command* has to be specified at its construction time. Also, it is required to specify whether this *Command* acts as a container or not.

virtual bool cmpKey(TCHAR key);*

This function compares the given key with its own key. If there is a match it returns TRUE. This function can be overridden for specialized handling

Command getParent();*

void setParent(Command);*

Command getNextSibling();*

void setNextSibling(Command);*

Command getFirstChild();*

void setFirstChild(Command);*

Command getPrevSibling();*

void setPrevSibling(Command);*

Every Command is a member of a Command tree (see Section 5.2.4). The above functions help the Command to access the immediate members of this tree.

5.2.4. Class RootCommand

The class RootCommand is instantiated only once during the execution of this program. RootCommand inherits from the base class Command, and acts like a container.

CommandTracker instantiates the RootCommand class during startup. It calls RootCommand::spawn() which creates a tree of Commands, which follows the same hierarchical structure as Terason's menu structure. Thus RootCommand serves as the root of this Command tree.

5.2.5. Other Command Classes

As stated earlier, to implement a menu/menu item we need to implement a Command class. The implementation of a Command may differ slightly depending on the category to which it belongs (Section 5.2.3). Commands can be categorized into two:

- Containers

  This category of Commands has Child Commands within it. A particular Command becomes a container if it represents a pull-down menu, with several menu items within it. Or it could represent a Command which invokes a new dialog window. In this case, a Child Command has to exist to allow the user to say 'Cancel', 'Close' etc.

  A Container should specifically implement the following functions of Command class:

  virtual void spawn()

  virtual void AddRule(CComPtr<ISpRecoGrammar>)

  virtual void RemoveRule(CComPtr<ISpRecoGrammar>)

  virtual void execute(TCHAR key, HWND hwnd)*

- Non-Containers

  The implementation of a non-container is easier. Implementation of the following function alone may serve the purpose:

  virtual void execute(TCHAR key, HWND hwnd)*

All the Commands which make up the Command tree are given in Appendix B.

5.3. Implementation

5.3.1. Initialization of Voice Command Dispatcher

The flow of execution during startup can be explained by means of the activity diagram in Figure 5.2:

The symbols usually used in an activity diagram are:

- stop of the activity
- activity state
- activity flow
- comment
- decision

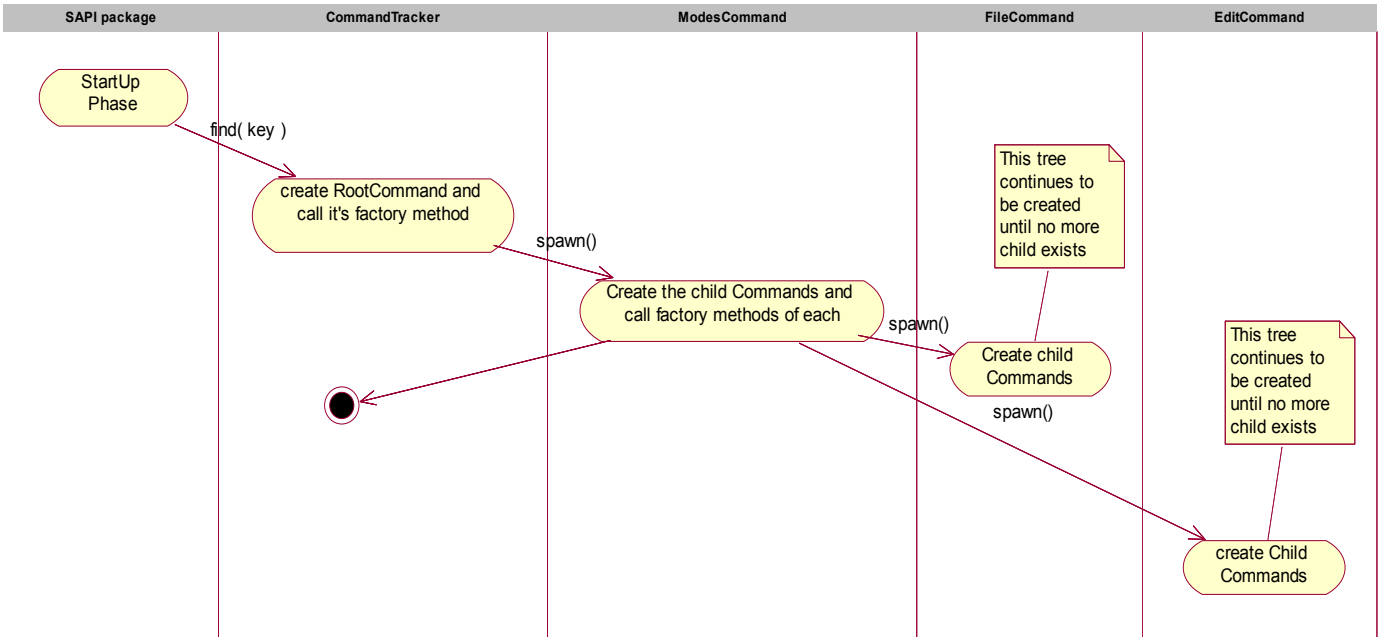


Figure 5.2: Initialization of voice command dispatcher

For initialization, the CommandTracker object calls the RootCommand object's factory method, spawn(). The RootCommand is considered to be the root of the menu structure. i.e., all the major menus represented by 'File', 'Edit', 'View', 'Insert', 'Image', 'Modes', 'Tools', 'Help' etc should be children of this object in the *Command* tree. The class FileCommand represents the menu 'File'. All the menu items underneath the menu 'File' should be children of the FileCommand object. For example, OpenCommand, ExportLoopCommand etc would be children of the FileCommand object. The Command tree is constructed in a similar way for the other Commands also. Refer to Appendix B for additional details.

5.3.2. Execution of a Container Item

The flow of execution that occurs when a Container key is recognized through voice can be better explained by the activity diagram given in Figure 5.3. For this example, we consider the recognition of the pull down menu 'Modes'.

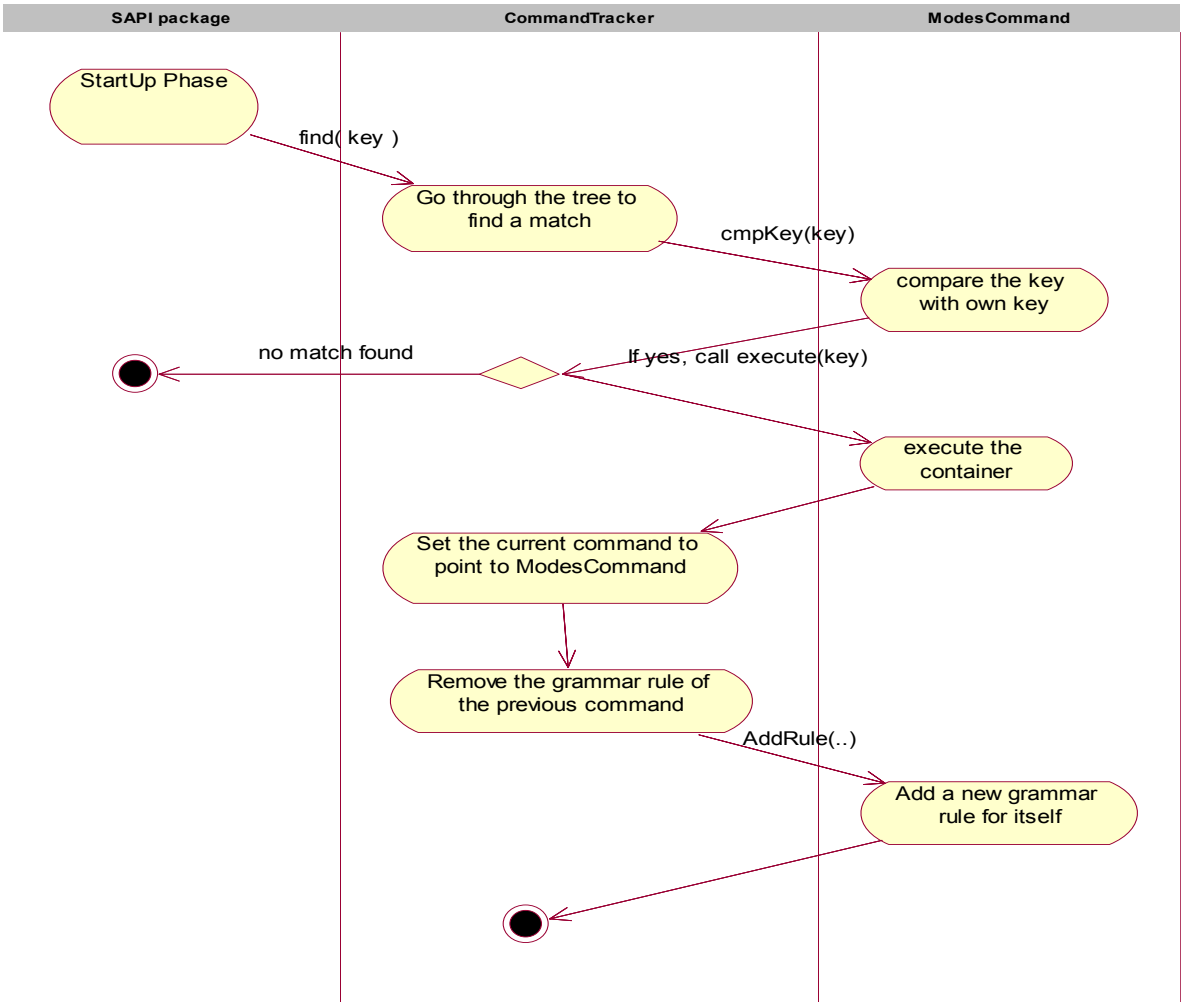


Figure 5.3: Execution of a Container item

When the SAPI package (*Voice command recognizer*) recognizes a voice command as text, it calls the CommandTracker::find() function, passing the recognized command as its parameter. The CommandTracker searches through the tree of Commands to find a match. Once a match is found (i.e., the ModesCommand object in this case), CommandTracker calls that particular Command to execute its function.

After execution, the attribute in CommandTracker class, m_pCurrentCommand, is made to point to ModesCommand. This is because the next time the user speaks a voice command, the CommandTracker considers ModesCommand as the root, and starts searching its children first. Thus its children enjoy a higher priority than its siblings or parent in the search procedure. Only if no match is found among its children, it will start searching its siblings.

The grammar rules have to be updated using the logic explained in Section 5.3.4, after a Container is recognized.

The ModesCommand::Execute() can be implemented as below:

```
void ModesCommand::Execute(TCHAR* key, HWND tHwnd)
{
    HWND tHwnd = ::FindWindow(NULL,_T("Terason Window Name"));
    if(tHwnd != NULL)
    {
            ::BringWindowToTop(tHwnd);
            ::SendMessage(tHwnd,WM_SETFOCUS,0,0);
            ::keybd_event(VK_MENU,0,0,0);
            ::keybd_event('O',0,0,0);
            ::keybd_event('O',0,KEYEVENTF_KEYUP,0);
            ::keybd_event(VK_MENU,0,KEYEVENTF_KEYUP,0);
    }
}
```

As can be seen, a voice command is send to the Terason application by the simulating keystroke events. If Terason window is the window on the foreground on the desktop, and we had pressed the keys '*Alt+O* ', the same action is done by the above section of the code.

5.3.3. Execution of a Non-Container Item

Figure 5.4 explains the execution of a non-container like the menu item ‘bmode’ in greater detail:

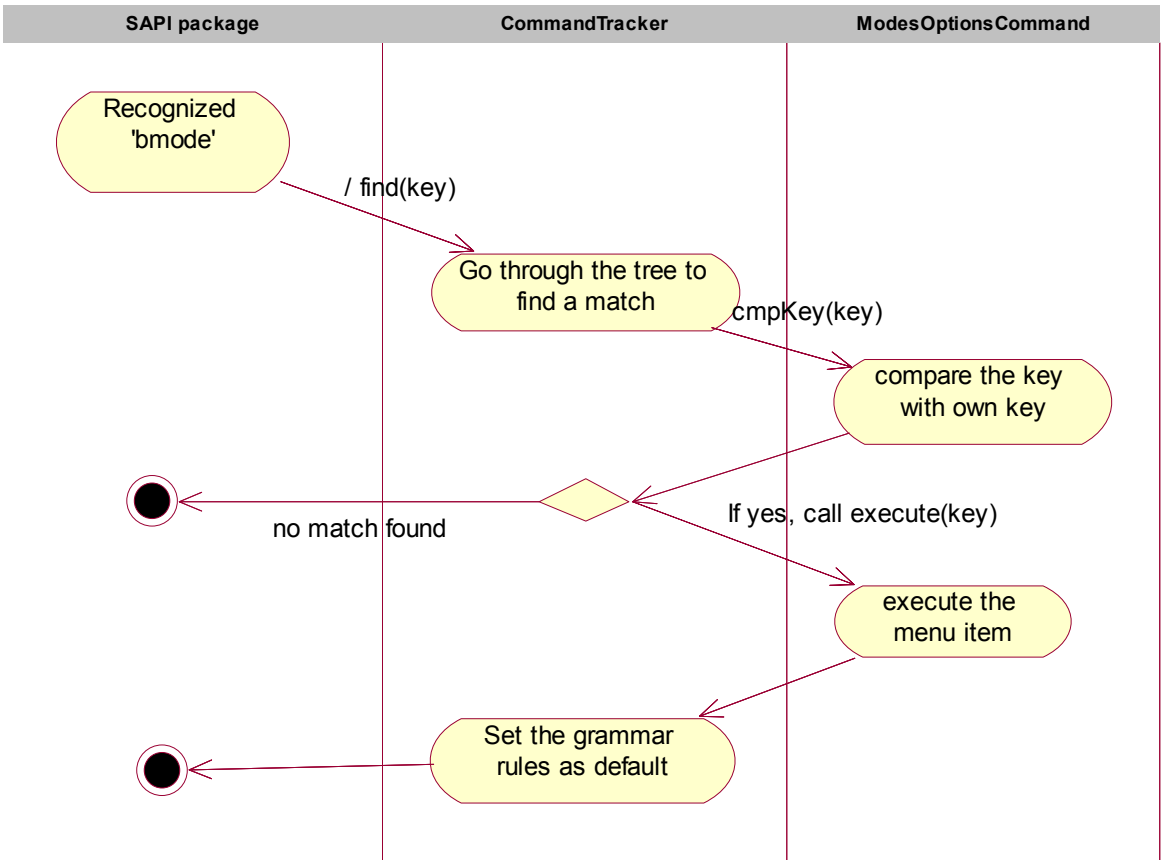


Figure 5.4: Execution of a non-Container item

The execution of a non-container item follows the same sequence of steps as that of a container item, except that the grammar updation is done in a different way. The grammar rules are set to their default state in this case, as mentioned in Section 5.3.4.

5.3.4. Grammar Updation

Appendix A shows what the speech recognition grammar file for Microsoft Command and Control looks like. Appendix C shows the grammar for Vocon 3200. The rules for the updation of the grammar during the recognition process are given below:

- When the speech engine is initialized with the grammar file, only one rule i.e., *"root"* is active. All the other rules are specified as inactive. This setting is considered as the default setting.
- When a non-container item is recognized, the grammar is always reset to its default setting.
- If the current command and the previous recognized command both are containers, and the current container is the child of the previous container, then the current container has to activate its grammar rule, and no rules have to be removed. An example of this situation is: user invokes '*Open*' after he invoked '*File*'.
- If the current command and the previous recognized command both are containers, and the current container is the sibling of the previous container, then the current container has to activate its grammar rule, and the previous container has to deactivate its rule. An example of this situation is: user invokes '*Invert*' after he invoked '*Palette*'. These containers come under the pull down menu *'Image'*.
- If the current command and the previous recognized command both are containers, and the previous container is the sibling of the parent of the current container, then the current container has to activate it's grammar rule, and the previous container has to remove it's rule An example of this situation is: user invokes '*Modes*' after he invoked '*Palette*'.

Chapter 6. Power Management

Preservation of battery power is critical for wearable computing applications. Therefore efficient handling of power consumption in wearable devices becomes particularly important. This chapter describes how power management features are incorporated into the Terason ultrasound scanner system.

6.1. System Requirements

In order to preserve battery power, the Terason scanner and the computer system (Sony V505-ACP laptop with Windows XP) have to be put into low-power mode of operation when not in use. Two different sleep modes can be supported, and the choice of the specific sleep mode depends on the time it takes for the system to go into sleep state and the time it takes for it to recover. In the maximum power-saving sleep state, the system takes longer time to go into sleep state and more time for recovery.

The ultrasound scanner system may enter into sleep modes in two different ways.

Method 1: It may respond to voice commands for going into sleep modes.

Method 2: The sleep states may be entered automatically if the scanning is not performed for a specified period of time.

It would be advantageous if the system can be brought out of sleep states by just touching the mini-joystick fitted on top of the transducer.

The different software involved in the power management are described below:

- Terason scanner software

This refers to the commercially available Terason scanner software.

- Voice command handler software

  This software interfaces with the speech recognition engine. It receives the recognized voice command from the speech recognition engine. Based on the command, it takes further action. When user invokes voice commands for activating a sleep state, it is this software which is responsible for putting the Terason scanner and the computer into low power operation.

- Operating system

  Windows XP operating system is installed on the laptop.

The different sleep states supported by the operating system are discussed in Section 6.2. The implementation steps required for putting the scanner and the laptop into sleep states with voice commands (Method 1) are covered in Section 6.3. The implementation steps required for Method 2 are not discussed in this thesis.

6.2. Power Management States

6.2.1. Power States Supported by Windows

The Windows operating system, starting from Windows 2000 onwards, supports six different power states. These power states are listed below:

- S0 / Working. The CPU is fully up and running.
- S1 / Sleep - The CPU is stopped; the system is running in a low-power mode.

- S2 / Sleep - Appears Off. The CPU has no power; the system is in a lower power mode than S1.
- S3 / Sleep (Standby) - Appears Off. The CPU has no power. The power supply is in a reduced power mode. The system is running in a lower power mode than S2.
- S4 / Hibernate (Soft Off) - The hardware is completely off and system memory has been saved to disk.
- S5 / Off - The system is completely Off. The hardware is off, the operating system has shut down; nothing has been saved. Requires a complete reboot to return to the Working state.

6.2.2. Power States Supported by Sony VAIO

The power states supported by Sony VAIO were determined by using the function GetPwrCapabilities() and the returned values in SYSTEM_POWER_CAPABILITIES structure were examined. More details regarding the usage of this function can be found in [15, [16]. It was found that Sony VAIO supports only the standby (S3) and hibernate (S4) states.

The ‘hibernate state’ consumes significantly less power than the ‘standby state’. The resume time from the ‘standby state’ normally requires a total of 5 seconds, while resuming from ‘hibernate’ requires 30 seconds.

6.3. Implementation Issues

6.3.1. Scanner Behavior

The active scanning of the scanner can be temporarily disabled, by executing the menu item *Image→Freeze* from the Terason scanner application window. This puts the scanner into a low-power situation. The ultrasound scanning can be put back into an active mode by executing the menu item *Image→Live* from the Terason scanner application window. This feature of the Terason scanner will be made use of for implementing the power management.

When the user speaks a sleep command, the Voice command handler software simulates the activation of the menu item *Image→Freeze* on the Terason scanner window. It then requests the operating system to go into a low power mode. This brings the PC into the requested sleep state.

When the operating system recovers from sleep, it requests the applications to recover to their normal state of operation. The Voice command handler software listens to these notifications, and puts the scanner back into active mode by simulating the activation of the menu item *Image→Live* on the Terason scanner window.

The above procedure had to be adopted as Terason scanner window, by itself, does not listen to power management notifications from the operating system.

The Terason scanner and the laptop can be put into standby state (S3) by giving the voice command ‘Standby’. It can be put into hibernate state (S4), by speaking ‘Hibernate’.

6.3.2. Power Management Implementation Steps

We discuss here the implementation required for putting the Terason scanner and the computer system into a low-power mode (standby/hibernate) through voice commands.

The implementation steps described below generally holds, whether the user wishes to put the system into the standby (S3) or to the hibernate (S4) state. The difference occurs only in the values of parameters specified for a particular Windows API function. This will be discussed later on. Following are the different implementation steps to be followed to put the system into low-power mode. The sequence of events shown by (1), (2) etc. in Figure 6.1 is described below.

(1) User invokes the voice command 'hibernate' or 'standby'
Voice command handler software listens to the voice input and detects the voice command with the help of the speech recognition engine.

2) Voice command handler software simulates the activation of the menu item Image->Freeze on the Terason application window. This would put the scanner into a low power situation.

3) Voice command handler software calls the Windows API function *SetSuspendState(bool state)*, to request the operating system to go into a low power mode. If the voice command invoked was 'hibernate', then *state* must be set to *TRUE*. If the voice command invoked was 'standby', then *state* must be set to *FALSE.*

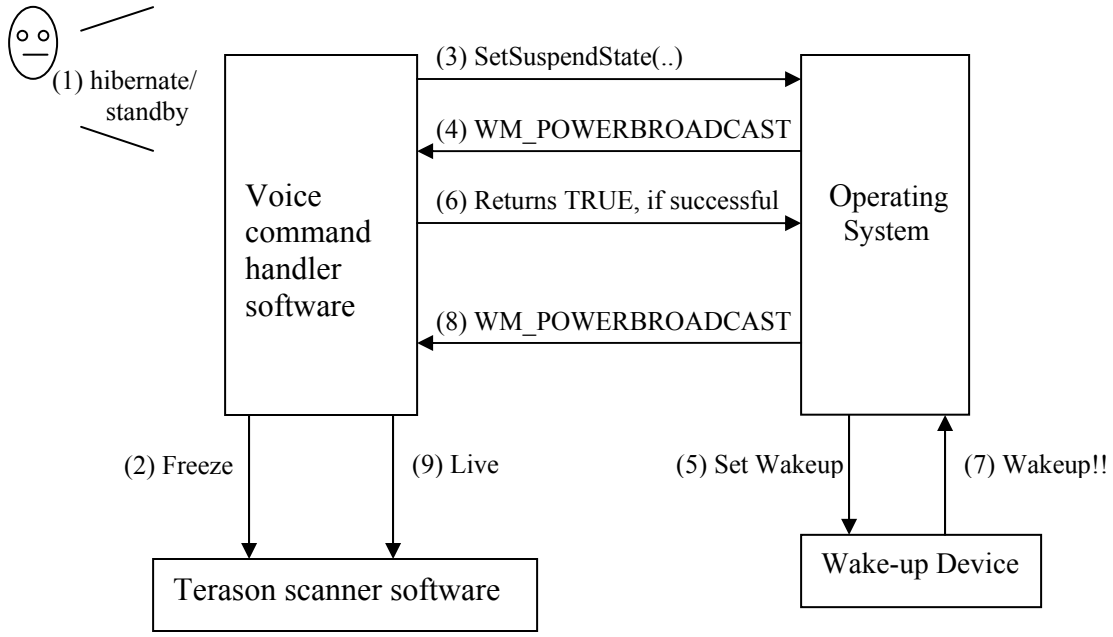


Figure 6.1: Execution steps for sleep and wake-up

4) Operating system prepares itself to go into the low-power mode. It sends a Windows Message called WM_POWERBROADCAST, which is broadcast to all applications and device drivers. This message contains a parameter called wParam, which has information about the power status change that has occurred. In this case, the value of wParam is set to PBT_APMQUERYSUSPEND event. The message is send by the operating system, as a request for permission from all applications and drivers to suspend their operations. The operating system expects each application to listen to this Windows Message, and to return TRUE if they have made sufficient preparation for a suspension of operation. Otherwise, they should return a BROADCAST_QUERY_DENY message to indicate that they have denied this request.

5 & 6) The Voice command handler software listens for WM_POWERBROADCAST and takes appropriate action. It temporarily disables the speech recognition engine, so that the minimum

RAM usage is ensured. It can also request the operating system to set up a specific device as a wake-up device. This would enable any activity at that device to wake-up the laptop. Once all this is done, it can return a TRUE to the operating system indicating that it is ready to go into the low-power mode.

If any of the applications denies this request from the operating system, then operating system issues another event called PBT_APMQUERYSUSPENDFAILED (also embedded inside the WM_POWERBROADCAST message). The Voice command handler software listens for this event also, and restart the recognition process, cancel any wake-up device enabled and put the scanner back into the active mode.

Suppose that all applications and drivers returned TRUE to step 4; then the operating system puts the laptop into the specified low-power mode. The only method by which the system can be turned on again is by pressing the power switch (if no wake-up device was defined). If a wake-up device was defined, then an activity at the device must be able to wake up the operating system.

7 & 8) Once the operating system has awakened, it will broadcast WM_POWERBROADCAST message to all applications and drivers, asking them to come back to working state. This message would have its wParam parameter set to PBT_APMRESUMESUSPEND event.

9) The Voice command handler software should listen for this message. On receiving it, it should simulate the menu item *Image->Live* on the Terason scanner application window. This would put the scanner back into active mode. The Voice command handler software should now enable the speech recognition engine, so that any further voice commands would be recognized.

6.3.3. Power Consumption in Sleep Modes for Sony VAIO

The power consumption of Sony VAIO in the two different sleep modes, standby and hibernate, was measured experimentally. The setup shown in Figure 6.2 was used for the measurement.

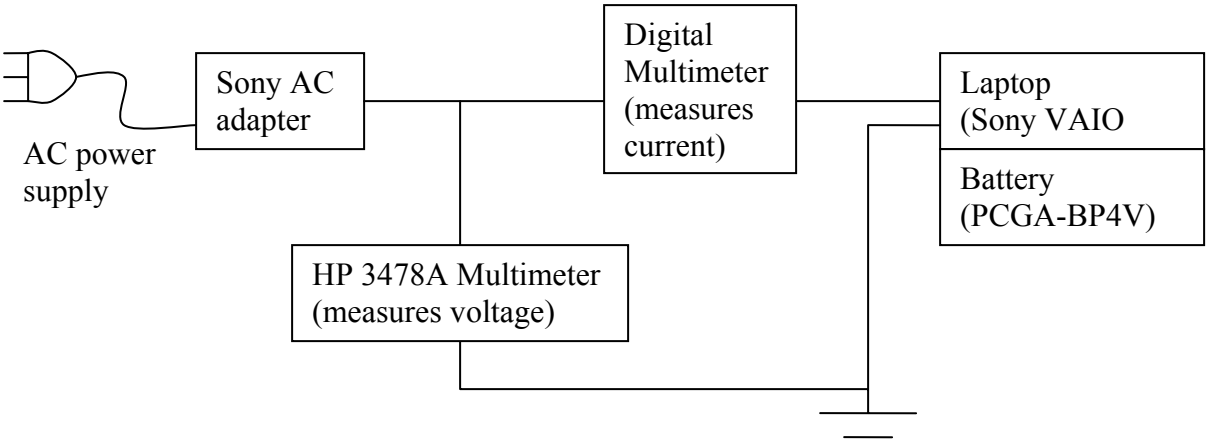


Figure 6.2: Experimental setup used for power measurement

A Digital Multimeter is connected in series between the laptop and its AC adapter. A HP 3478A Multimeter is also connected across the ac adapter and the ground, to measure the voltage. The Sony laptop has an extended battery PCGA-BP4V of capacity 11.1V/8.8A, attached to it. Measurements were made both with battery on and with the battery taken off from the Sony VAIO. The measurings were recorded using paper and pencil, after 1, 5, 10, 20 and 30 minutes. The experiments were repeated for 4 different cases: after the laptop was completely booted up, after the laptop was put into standby state, after the laptop went into hibernate state and after the laptop was shutdown. The results are shown below. *V* indicates the voltage value in Volts, and *A* indicates the current value in Amperes.

Measurements with battery: (Plots on Figure 6.3)

After / State	1 min		5 min		10 min		20 min		30 min	
	V	A	V	A	V	A	V	A	V	A
Booted up	15.51	1.33	15.51	1.13	15.51	1.14	15.51	1.14	15.51	1.14
Standby	15.74	0.16	15.75	0.15	15.75	0.16	15.74	0.15	15.74	0.15
Hibernate	15.78	0.011	15.78	0.011	15.78	0.011	15.78	0.011	15.78	0.011
Shutdown	15.78	0.011	15.78	0.011	15.78	0.011	15.78	0.011	15.78	0.011

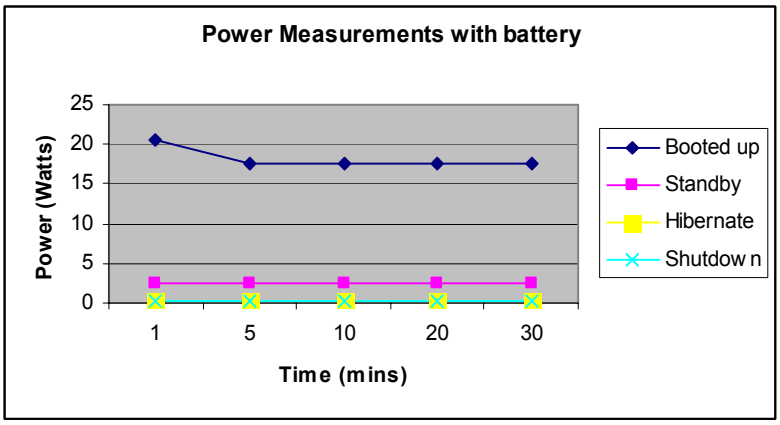


Figure 6.3: Power measurements with battery

Measurements taken without battery: (Plots on Figure 6.4)

After / State	1 min		5 min		10 min		20 min		30 min	
	V	A	V	A	V	A	V	A	V	A
Booted up	15.55	1.14	15.55	1.14	15.51	1.32	15.55	1.13	15.55	1.14
Standby	15.75	0.15	15.75	0.15	15.75	0.15	15.75	0.15	15.75	0.15
Hibernate	15.78	0.010	15.78	0.010	15.78	0.010	15.78	0.010	15.78	0.010
Shutdown	15.78	0.010	15.78	0.010	15.78	0.010	15.78	0.010	15.78	0.010

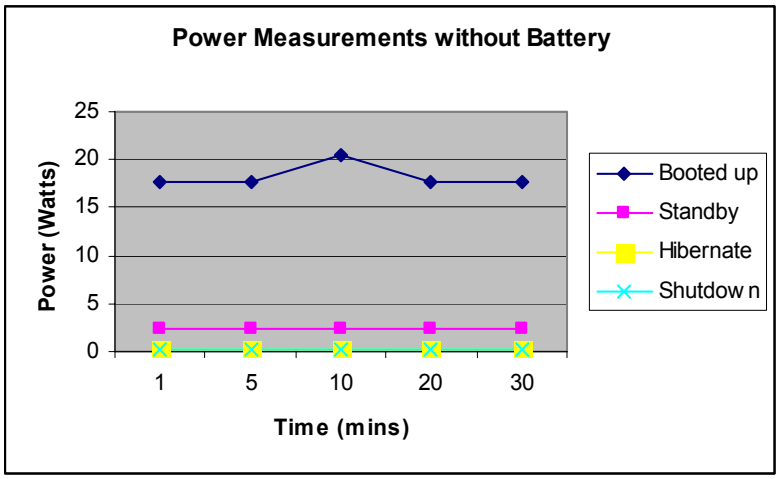


Figure 6.4: Power measurements without battery

Power consumption of Sony VAIO under normal working conditions

= 15.55 V * 1.14 A = 17.727 W.

Power consumption of Sony VAIO under Standby = 15.75 * 0.15 = 2.3625 W.

Power Consumption of Sony VAIO under Hibernate = 15.78 * 0.010 = 0.1578 W.

Power Consumption of Sony VAIO after shutdown = 15.78 * 0.010 = 0.1578 W.

The measurements also show that, the presence of the extended battery at the Sony VAIO does not affect considerably the amount of power drawn from the AC supply in the setup shown in Figure 6.2.

6.4. Open Issues

6.4.1. Wake-up from Sleep States

It has not been possible to wake the laptop from the sleep state by means of a wake-up device. The idea was to wake up the laptop from the standby state by moving the mini-joystick (manufactured by Varatouch technologies) fitted on top of the transducer. It may not be possible to wake-up from hibernate using the mini-joystick, as system hardware is completely off in this state.

Wake-up during standby was tried out using two different methods.

1) By calling the Windows API, RequestDeviceWakeup() to set a specific device as a wake-up device.

Windows Operating system returned a failure when this method was executed, indicating that the system does not support this API. This may be because the device driver or the device itself do not support the power management features.

2) By using the option 'Allow this device to bring the computer out of standby' in the Device Properties window. This window can be opened by going into Device Manager and selecting the particular device and selecting its Properties.

Although it was possible to enable this option under Device Properties window, it was found that moving the mini-joystick during the standby state did not wake up the system.

6.4.2. Setting System to Sleep when Transducer is Idle

Another potentially useful power saving mode, which is not implemented, is to put the laptop and the Terason scanner into sleep states if the scanning is not performed for a specific period of time. This was mentioned in Section 6.1. It was observed that when the SmartProbe was connected to the laptop and the Terason software running with the transducer idle, adjustment of the timer values in the Power Options window in the Control Panel of Sony VAIO did not bring the laptop to standby/hibernate states. Therefore, in order for the laptop to detect that the SmartProbe is not actively scanning at any time, some sort of image analysis may have to be done on the images acquired by the ultrasound scanner.

It may be possible that the Terason software itself may have the ability to get the information of whether the transducer is idle or not. This information can be acquired by the Voice command handler software for processing, if a suitable software interface is provided by Terason in [18].

Chapter 7. Speech Recognition Engine Evaluation

The choice of the automatic speech recognition (ASR) engine depends on how and where it will be applied. As mentioned earlier, the ASR engine for our application is targeted for use in an emergency ultrasound scanning environment. Therefore, it is essential that the ASR engine has command and control features. It must also support context free grammar, so that commands will be recognized only from a finite command set. This has a significant impact on the recognition performance.

In addition to the above features, it is important to evaluate the ASR engine's performance in an environment with high ambient noise. Furthermore, the recognition results must show only a little variation with different types of noise and speakers. The aim of this chapter is to analyze the performance of different ASR engines under different Signal to Noise Ratios (SNRs), noise and speakers.

This is achieved by making quiet room recordings using an airborne microphone from different speakers. Noise is then added electronically to each of these recordings, to create a noisy speech file. The amplitude of the noise is adjusted each time in order to get different SNRs. Different types of noise are also used for this testing. The noisy speech files are then fed to each of the ASR engine and the recognition statistics recorded.

7.1. Automatic Speech Recognition Products Tested

The ASR products that are being tested are:

- Vocon 3200 Version 1.2
- IBM Viavoice command and control
- Microsoft command and control
- Dragon English NaturallySpeaking

The key factors responsible for the selection of an ideal engine are listed below. The compliance of each engine to each of these factors is also indicated in the table. The symbol 'X' indicates that the compliance is unknown.

Selection factors	**Vocon**	**IBM**	**Microsoft**	**Dragon**
Command and control	Yes	Yes	Yes	Yes
Context free grammar support	Yes	Yes	Yes	Yes
Speaker independent	Yes	Yes	No	No
Small vocabulary	Yes	Yes	Yes	No
Different voices or pronunciations	X	X	X	X
Different noise types	X	X	X	X
Different ranges of SNRs	X	X	X	X

Table 7-1: Speech recognition engine features

This chapter deals with the analysis of the engines' performances with respect to the last three factors.

7.2. Design of the Command Set

The selected command set must contain commands that are relevant in an ultrasound scanning experiment. Each command should have a minimum of two or three words. The selection of single word commands was avoided due to the higher possibilities of incorrect recognitions, when noise is present.

It was found that having a common command prefix in front of each command was increasing the recognition rate. The prefix chosen was 'Terason'. One argument to support this result is that the prefix helps the speech recognition engine to distinguish a command from noise. Sample tests were run with Microsoft command and control speech recognition engine in order to verify this result.

It is also desired that the speech recognition engine must perform reasonably well over the entire range of SNRs for each selected command. In other words, if a specific command was consistently not recognized for SNRs above 30 dB, then that command was omitted from the command set. Some sample command sets were initially chosen and the recognition statistics for each command in the command set was analyzed using the speech recognition engines. A final command set containing 27 commands was chosen after a screening process. This is given in the Appendix D.

Figure 7.1 shows the variation of the recognition performance of the various commands in the final command set, with Vocon 3200 V1.2 ASR engine. Each command was repeated by 5

different speakers, over a wide range of SNRs and noise types. The results shown are obtained after averaging over all the different speakers, noise types and SNRs.

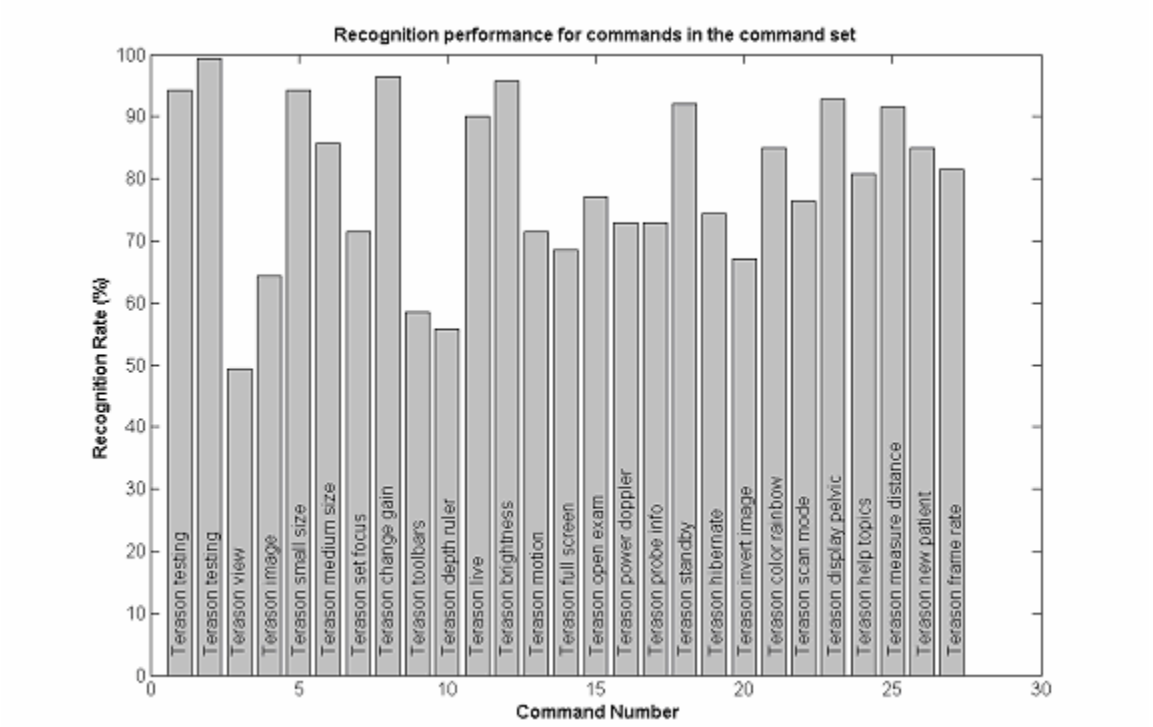


Figure 7.1: Recognition performance of the 27 commands in the command set, averaged over all speakers

7.3. Noise Types Selected for Speech Recognition Testing

It is desired that the performance of the ASR engine does not vary much with different types of noise. The noise selected for this testing resembles an environment where many people are simultaneously speaking. This noise is added electronically to quiet command utterances at specific SNRs, before being fed to the speech recognition engine to estimate performance. Four different noises are selected. They are as follows:

- Unmodulated random Gaussian noise – with spectral weighing for the male voice
- Speech modulated noise – with spectral weighing for the female voice
- 2 persons babble – (a male and female speak at a short distance)
- 6 persons babble – (a male and a female at a short distance and two females and two males at a greater distance)

All the above noise is produced by W.A. Dreschler for the International Collegium of Rehabilitative Audiology (ICRA), Netherlands [18].

7.4. Selection of Speakers

A wide variety of users may use the ASR engine in an emergency scanning scenario. Due to time-constraints, it is too difficult a task to test with a large number of speakers. Also, having too few speakers is likely to give unreliable results. Therefore, a group of 6 speakers were chosen depending on the gender and accent. The speakers include:

- 3 American native male speakers
- 1 American native female speaker
- 1 non-American male speaker
- 1 non-American female speaker

7.5. Speech Recognition Engine Test Setup

7.5.1. Test Layout

The speaker sits inside a quiet chamber with a typical Sound Pressure Level (SPL) of 43 dB. The SPL meter used for measurement was Quest Electronics SPL meter, Model-215, with A-

weighting setting. The airborne microphone Emkay Model-3345 was used to create all the recordings. The microphone was directly connected to an external sound card which is connected to the laptop via an USB input. Recording software running on the laptop creates the recording in 22 KHz, 16 bit Mono PCM format. The recording software used was TotalRecorder from *HighCriteria Inc.* The laptop used was Sony VAIO V505A and the sound card was Sound Blaster Audigy 2 NX. Figure 7.2 illustrates the test layout.

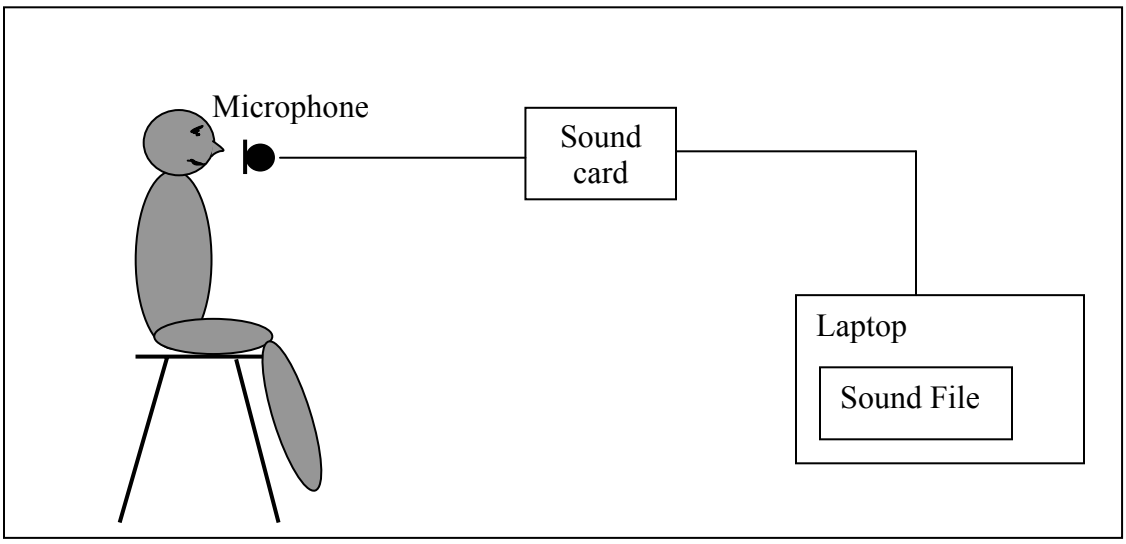


Figure 7.2: Speaker makes recordings inside a quiet chamber

Each speaker makes a total of two recordings. The first recording consists of reading a calibration passage lasting for around 30 sec, which will be subsequently used to adjust the microphone recording volume in the laptop, before running the speech recognition engine tests for that particular speaker. For the second recording, the speaker speaks all the 27 commands in the specified command set, with 5 seconds duration in between. A total of 6 speakers participate

in this testing process. Steps were taken to ensure that the speech signal is not being clipped during all the recordings.

7.5.2. Creation of Audio Tuned Sound Files

Audio tuning is the process by which the microphone recording volume of a laptop is set to the optimum level for speech recognition for a specific speaker. As mentioned above, each speaker records an initial 30 seconds passage which is used for the audio tuning. The calibration passage recorded inside the quiet chamber is played on one laptop. The audio output from the sound card of the first laptop is then fed to the audio input of the sound card of a second laptop. While the sound file is played on the first laptop, an audio tuning software running on the second laptop adjusts its microphone recording volume to the optimum value. This process is illustrated in Figure 7.3. Once the audio tuning is complete, the first laptop plays the sound file containing the command utterances for that particular speaker, and the second laptop re-records it in 22 KHz 16 bit mono PCM format. This re-recorded sound file containing all the command utterances is termed as the *audio tuned* file.

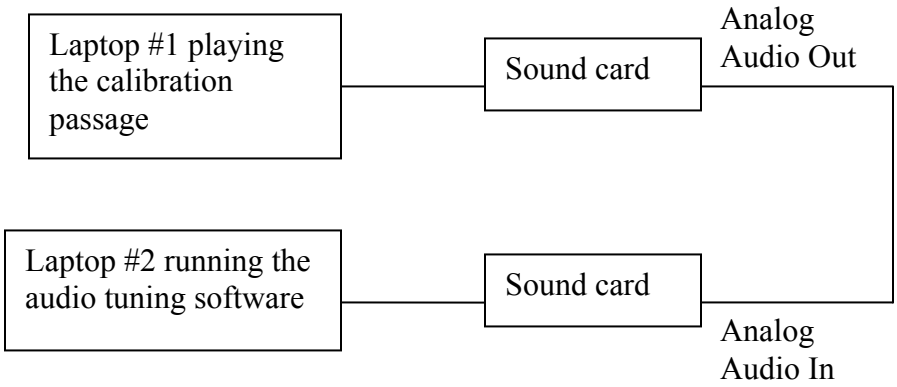


Figure 7.3: Audio tuning layout

The sound cards used were Sound Blaster Audigy 2 NXs. Laptop #1 was Dell Latitude C840 and Laptop #2 was Sony VAIO V505A. The audio tuning process was repeated for every speaker. The resultant set of 6 audio tuned sound files were used for the subsequent testing. Both Microsoft command and control and Dragon Naturally Speaking have their own audio tuning software. So, for a given recording, the final adjusted microphone recording volume after audio tuning is usually different for Microsoft command and control and Dragon Naturally Speaking. Vocon 3200 and IBM Viavoice specify that the input speech signal should lie within the acceptable range without clipping. They do not come with any specific audio tuning software. Therefore, the sound files obtained after audio tuning with Microsoft command and control were also used as input for both IBM Viavoice and Vocon 3200.

7.5.3. Speech Recognition Engine Evaluation Layout

The main purpose of performing the speech recognition evaluation is to determine the ASR performance at different types of noises, SNRs and speakers. To achieve this, the noise is scaled and added electronically to each quiet command utterance of every speaker, and the resultant file is fed to the speech recognition system and the statistics recorded. Figure 7.4 shows the evaluation process in detail.

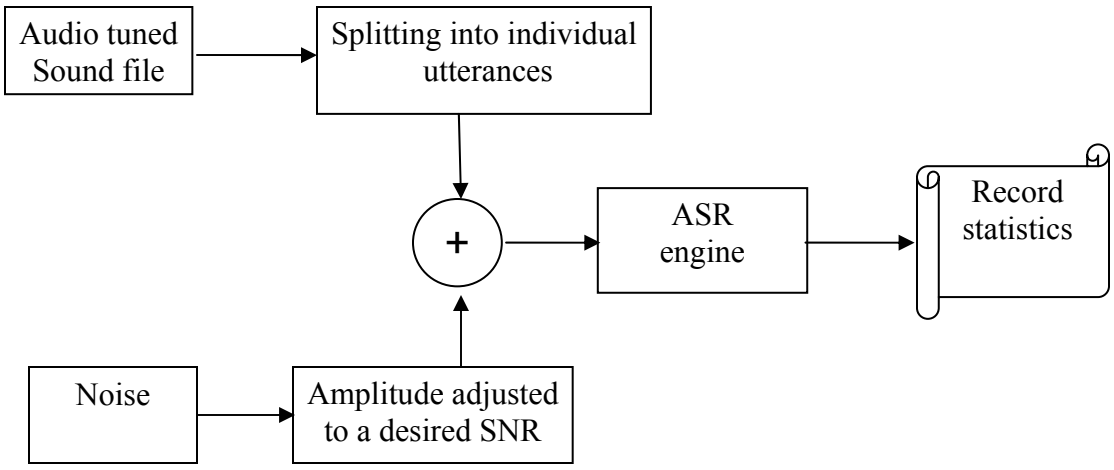


Figure 7.4: Speech recognition engine evaluation layout

The audio tuned sound file, consisting of all the 27 commands spoken in 5 seconds interval is split into individual command utterances. Noise is then added to each of these command utterances, with its amplitude scaled so as to obtain a desired SNR. This process is explained in detail in Section 7.5.4. The noisy command utterance is then fed to the ASR engine under consideration and the result recorded. To account for the random nature of noise added to the speech signal, the whole process is repeated 12 times. At each repetition, the section of the noise added to the speech is selected randomly from the original noise file. With the above method, 4 different types of noises under 7 different SNRs are tested. Finally, the statistics are collected in order to analyze the performance characteristics.

7.5.4. Creation of the Noisy Command Utterance

The audio tuned sound file contains all the commands uttered one after another with a few seconds pause in between. This has to split into individual utterance before it is given to the ASR engine for analysis. This is done as follows: The local Root Mean Square (RMS) of the speech

signal is determined by defining short duration windows over the entire speech signal and calculating the RMS value of each window. The duration of the window is set as an adjustable parameter, typically 40 milliseconds. It is assumed that when the RMS value of the signal is above a certain cutoff threshold, speech is present. This cutoff threshold is specified as a certain percent of the maximum value of the RMS and is also an adjustable parameter. By default, it is kept as 5 percent. The boundaries of each utterance are thus identified and sufficient buffer is given both at the beginning and the end of each utterance before creating a sound file containing only that utterance. This will ensure that no part of the speech is lost.

Once the individual utterance is separated out, noise has to be added at specific SNRs. For this, the signal power of each individual utterance is calculated. The amplitude of the noise is then adjusted, such that the resulting noise power is in accordance with the desired SNR. The sum of this adjusted noise signal and the speech signal is then saved as a sound file, with the same format as the original recording made in the quiet room. Seven different SNRs ranging from 40 dB up to 0 dB, are used for the testing to analyze the performance of the speech recognition engine at different noise conditions.

Once the above processing is completed on the clean speech file, $27 \times 4 \times 7 = 756$ noisy command files are created (27 commands × 4 noise types × 7 noise levels) for each speaker. These commands are fed one by one to the ASR engine and the results are recorded. As described in Section 7.5.3, the addition of noise to each command utterance is repeated 12 times, in order to reproduce the random nature of noise in speech. The final recognition result of each command utterance for a specific SNR is the average of the recognition results over all the 12 repetitions. The process described here has been implemented using Matlab scripts.

7.5.5. Training for Speaker Dependent ASR Engines

Microsoft command and control and *Dragon English NaturallySpeaking* are speaker dependent ASR engines. They have to be trained for each individual speaker before getting acceptable recognition rates for that speaker. The training is done in two different ways for each of the engines.

One is the creation of a speaker profile (Section 4.1.1) where each subject who is going to speak to the ASR engine has to complete a training session specified by the engine software. This starts with reading a short passage during which the ASR engine adjusts the microphone recording volume in that laptop to the optimum level for speech recognition. This is followed by reading a longer passage. The ASR engine uses this speech to develop acoustic models for that speaker. The whole process takes around 10-15 minutes.

The other way of training is specific-vocabulary training. The speaker reads the commands in the given command set one after another, while the ASR engine collects data for training itself. The speaker repeats the entire command set twice for this training.

Both these forms of training were done for Microsoft Command and Control and Dragon NaturallySpeaking for each of the 6 speakers. The training process was completed for both ASR engines, before performing the testing with these engines. Prior to performing tests on an engine for a particular speaker, the considered speaker's profile was made as the active profile for the engine.

Vocon 3200 and IBM Viavoice command and control ASR engines are speaker independent engines, so no forms of training were done for them.

7.5.6. Test Procedure

This section describes the different tasks that need to be performed before executing the tests for the different ASR engines. The tasks are as follows:

- Create a test directory for each of the ASR engines.
- Copy the Matlab scripts and the ASR evaluation software (Section 7.5.7) and the relevant context-free grammar file to this directory. Also copy the audio tuned recordings of each of the 6 speakers to this directory. Copy all the noise files in a subdirectory ‘./noise’. Create a subdirectory for storing the results of the Matlab scripts, by name ‘./results’.
- If the ASR engine accepts the sound files with a lower sampling rate, it has to be ensured that all the input recordings and the noise are down-sampled and saved in the proper format compatible with the ASR engine requirement.
- If the ASR engine is speaker dependent, prior to running the tests for a speaker make sure that his/her profile is the active profile on the ASR engine.
- Run the Matlab scripts. It requests, as input, the name of the recording. Once the recording file name is given, the Matlab scripts automatically generates 756 sound files (Section 7.5.4) and stores them in the subdirectory ‘./results’. The name of the sound file has the following format:

  X_Y_Z.wav , where X is the command number, Y is the noise type number and Z is the SNR number.

Once all the sound files are generated, the Matlab scripts automatically start up the ASR evaluation software. This software processes each sound file in the ‘./results’ subdirectory one by

one, and writes its output into a file by name‘./dataout.txt’. The Matlab scripts repeat this entire process of generating the sound files, processing them and writing the output, for a total of 12 times. Once the Matlab scripts have stopped running, the ‘./dataout’ file is analyzed to get the recognition performance of the considered ASR engine for that particular speaker. The ratio of the number of recognized commands to the total number of commands is the recognition accuracy.

7.5.7. ASR Evaluation Software

A software program has to be written for each ASR engine by using the respective SAPIs, in order to carry out these tests. The program must be able to recognize commands from a given set of sound files (in .wav format). The command list will be provided in the command line as input when the program is started up. While the recognition test is being done, the program writes the output to a text file. When the recognition test is completely over, the program generates statistics and writes it to the end of the output file. The program always opens the output file in append mode, so that the contents of the output file are preserved, even if the program is called a multiple number of times.

We must be able to invoke the program from the command line. A typical command line invocation would look like:

speech_test –d speaker_name –cmd [command_list]

speaker_name – the name of the speaker enclosed in double quotes.

[command_list] – the sequences of commands enclosed in double quotes, each separated from the other by a space. The commands should be specified in the same order as the order in which the sound files generated by the Matlab scripts in the ‘./results’ directory are named.

For e.g., a sample command line may look like:

speech_test –spk "speaker_name" -cmd "Terason testing" "Terason testing" "Terason view" "Terason image"……..

A few of the important statistics to be analyzed for a given ASR engine and speaker are:

Total number of recognized commands averaged over all commands, noise types and SNRs:

It is also important to know the variation of the recognition accuracy with different types of noise for a given ASR engine. Therefore, the following statistics record the number of recognized commands added over all the different noise levels for a given noise type.

No. of recognized commands for NoiseType 1 (Random Gaussian Noise (RGN)):
No. of recognized commands for NoiseType 2 (6 persons babble (6PB)):
No. of recognized commands for NoiseType 3 (2 persons babble (2PB)):
No. of recognized commands for NoiseType 4 (Speech modulated noise (SMN)):

The recognition accuracy of the ASR engine over different Signal to Noise Ratios would give a measure of the ASR engine performance in an environment with high ambient noise. The following statistics record the number of recognized commands added over all the different noise types for a given noise level.

No. of recognized commands for NoiseLevel 1 (SNR: 40 dB):
No. of recognized commands for NoiseLevel 2 (SNR: 30 dB):
No. of recognized commands for NoiseLevel 3 (SNR: 20 dB):
No. of recognized commands for NoiseLevel 4 (SNR: 15 dB):
No. of recognized commands for NoiseLevel 5 (SNR: 10 dB):
No. of recognized commands for NoiseLevel 6 (SNR: 5 dB):
No. of recognized commands for NoiseLevel 7 (SNR: 0 dB):

The following statistics record the total number of recognized commands for a given noise type and SNR.

No. of recognized commands for NoiseType 1 and NoiseLevel 1 (RGB at SNR=40 dB):
No. of recognized commands for NoiseType 1 and NoiseLevel 2 (RGB at SNR=30 dB):
No. of recognized commands for NoiseType 1 and NoiseLevel 3 (RGB at SNR=20 dB):
No. of recognized commands for NoiseType 1 and NoiseLevel 4 (RGB at SNR=15 dB):
No. of recognized commands for NoiseType 1 and NoiseLevel 5 (RGB at SNR=10 dB):
No. of recognized commands for NoiseType 1 and NoiseLevel 6 (RGB at SNR=5 dB):
No. of recognized commands for NoiseType 1 and NoiseLevel 7 (RGB at SNR=0 dB):

No. of recognized commands for NoiseType 2 and NoiseLevel 1 (6PB at SNR=40 dB):
No. of recognized commands for NoiseType 2 and NoiseLevel 2 (6PB at SNR=30 dB):
No. of recognized commands for NoiseType 2 and NoiseLevel 3 (6PB at SNR=20 dB):
No. of recognized commands for NoiseType 2 and NoiseLevel 4 (6PB at SNR=15 dB):
No. of recognized commands for NoiseType 2 and NoiseLevel 5 (6PB at SNR=10 dB):
No. of recognized commands for NoiseType 2 and NoiseLevel 6 (6PB at SNR=5 dB):
No. of recognized commands for NoiseType 2 and NoiseLevel 7 (6PB at SNR=0 dB):

No. of recognized commands for NoiseType 3 and NoiseLevel 1 (2PB at SNR=40 dB):
No. of recognized commands for NoiseType 3 and NoiseLevel 2 (2PB at SNR=30 dB):
No. of recognized commands for NoiseType 3 and NoiseLevel 3 (2PB at SNR=20 dB):
No. of recognized commands for NoiseType 3 and NoiseLevel 4 (2PB at SNR=15 dB):
No. of recognized commands for NoiseType 3 and NoiseLevel 5 (2PB at SNR=10 dB):
No. of recognized commands for NoiseType 3 and NoiseLevel 6 (2PB at SNR=5 dB):
No. of recognized commands for NoiseType 3 and NoiseLevel 7 (2PB at SNR=0 dB):

No. of recognized commands for NoiseType 4 and NoiseLevel 1 (SMN at SNR=40 dB):
No. of recognized commands for NoiseType 4 and NoiseLevel 2 (SMN at SNR=30 dB):
No. of recognized commands for NoiseType 4 and NoiseLevel 3 (SMN at SNR=20 dB):
No. of recognized commands for NoiseType 4 and NoiseLevel 4 (SMN at SNR=15 dB):
No. of recognized commands for NoiseType 4 and NoiseLevel 5 (SMN at SNR=10 dB):

No. of recognized commands for NoiseType 4 and NoiseLevel 6 (SMN at SNR=5 dB):

No. of recognized commands for NoiseType 4 and NoiseLevel 7 (SMN at SNR=0 dB):

7.5.8. Dragon English NaturallySpeaking Test Layout

This section describes the test setup for analyzing the performance of the speech recognition engines, *Dragon English NaturallySpeaking*. This ASR engine is large footprint and mainly targeted for dictation use. But it supports Microsoft Speech API 4.0, by means of which command and control operations can be supported. Therefore, a software application was developed which uses the Direct Speech Recognition API features of Speech Application Program Interface 4.0 (SAPI 4.0) to operate the ASR in command and control mode.

To perform the evaluation tests on Dragon English NaturallySpeaking, the software was initially configured to run in the same way as described in Section 7.5.7, but the recognition results from the engine were found to be very poor. In other words, the recognition rate of was very low when the speech input was given in the form of sound files. It was found that a better recognition rate was obtained when the speech input came directly from the microphone.

Therefore, for evaluation purposes, Dragon English NaturallySpeaking was configured to receive speech input directly from the microphone. The Matlab scripts were made to generate the noisy sound files, exactly as described in Section 7.5.4, and the resultant files were stored in the ‘./results’ directory. The difference in the test setup of Dragon as compared to the other ASR engines is the method by which the noisy command utterances are fed to the ASR engine. The evaluation layout of Dragon is shown in Figure 7.5.

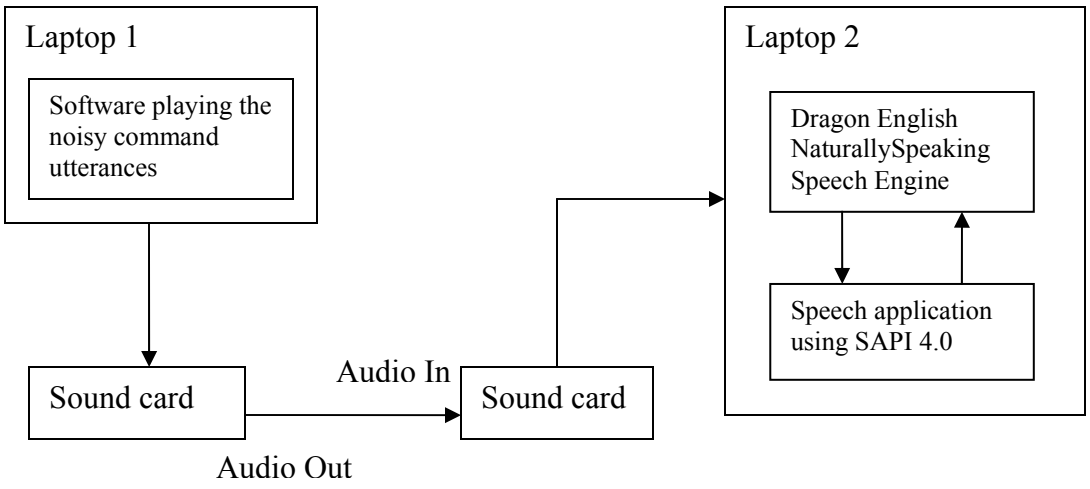


Figure 7.5: Dragon English NaturallySpeaking evaluation layout

The wav files are played on laptop 1. The audio output of the sound card of laptop 1 is fed to the audio input of the sound card of laptop 2. A speech application is made to execute on laptop 2, which sets up the Dragon engine in command and control mode, listening to the microphone input of laptop 2. The ASR engine evaluates this speech and identifies the command. The result is written to an output file by name './dataout.txt'. Once all the sound files are played, the statistics are calculated.

7.6. ASR Evaluation Results

This section summarizes the comparison results between three different ASR engines, viz. Microsoft Command and Control, Vocon 3200 and IBM Viavoice Command and Control. A few tests were performed with Dragon English NaturallySpeaking, but it was found to be a poor performer for command and control under moderately high ambient noise. The results of Dragon are briefly discussed under Section 7.6.4.

7.6.1. Recognition Performance versus SNRs

We compare here the performances of Vocon 3200, IBM Viavoice and Microsoft Command and Control under a wide range of SNRs. The x-axis specifies the SNR and the y-axis indicates the recognition rate. The results are obtained after averaging over all the different noise types. Each graph shows the results for a specific speaker. The speakers are referred in the graphs as:

Speaker #1 – Non-American male, Speaker #2 – American male, Speaker #3 – American female, Speaker #4 – American male, Speaker #5 – American male, Speaker #6 – Non-American female.

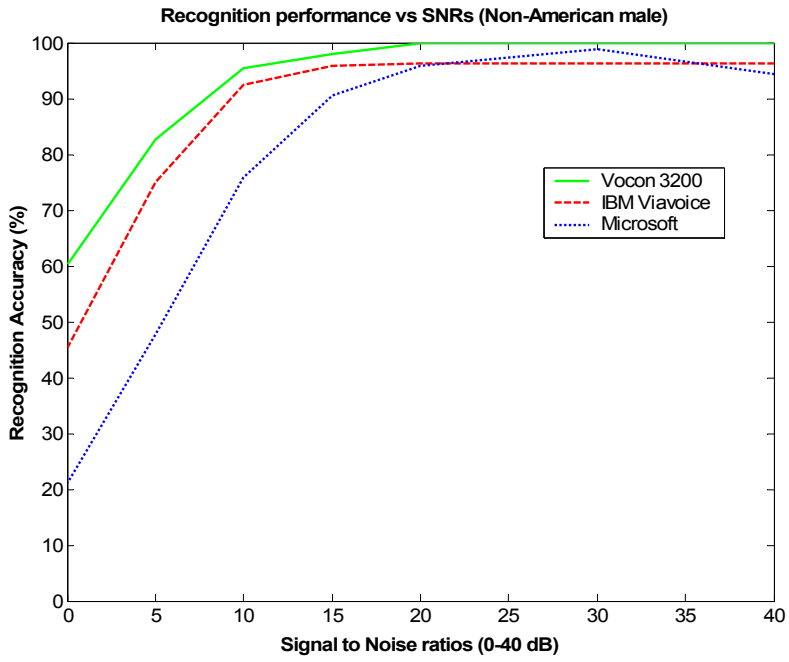


Figure 7.6: Speaker #1 - Recognition performance versus SNRs (averaged over all noise types)

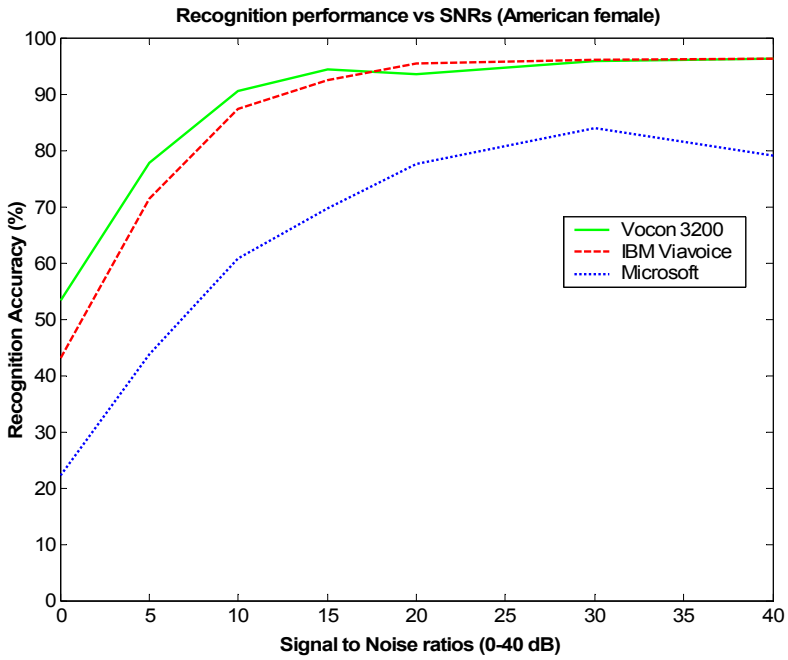


Figure 7.7: Speaker #3 - Recognition performance versus SNRs (averaged over all noise types)

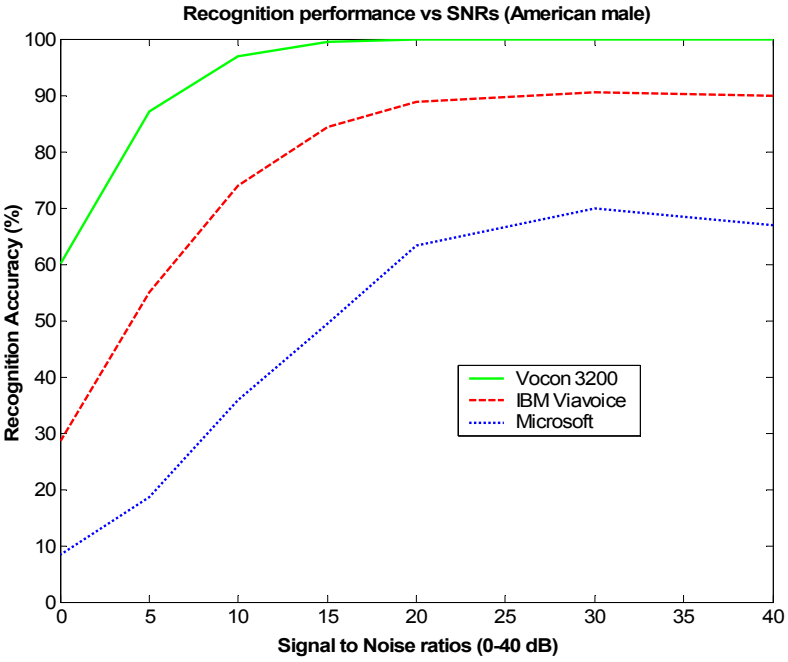


Figure 7.8: Speaker #4 - Recognition performance versus SNRs (averaged over all noise types)

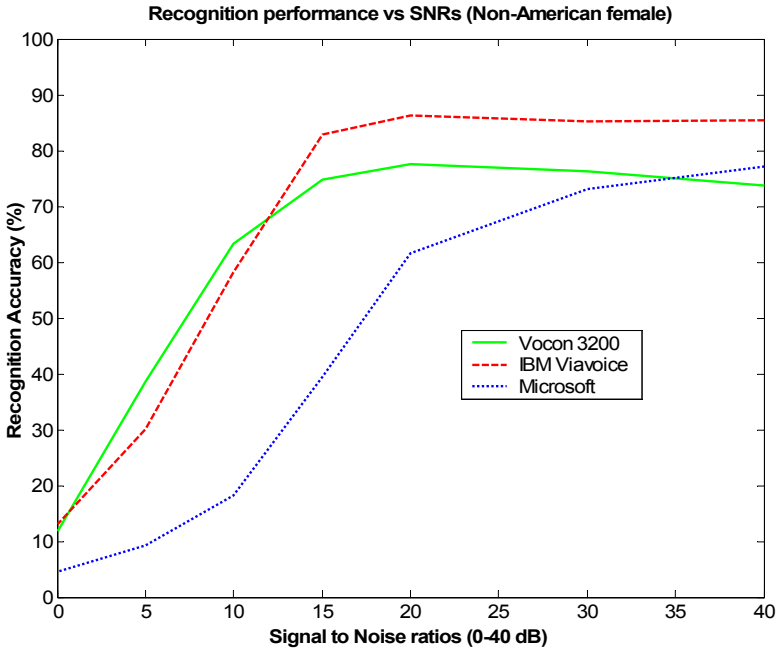


Figure 7.9: Speaker #6 - Recognition performance versus SNRs (averaged over all noise types)

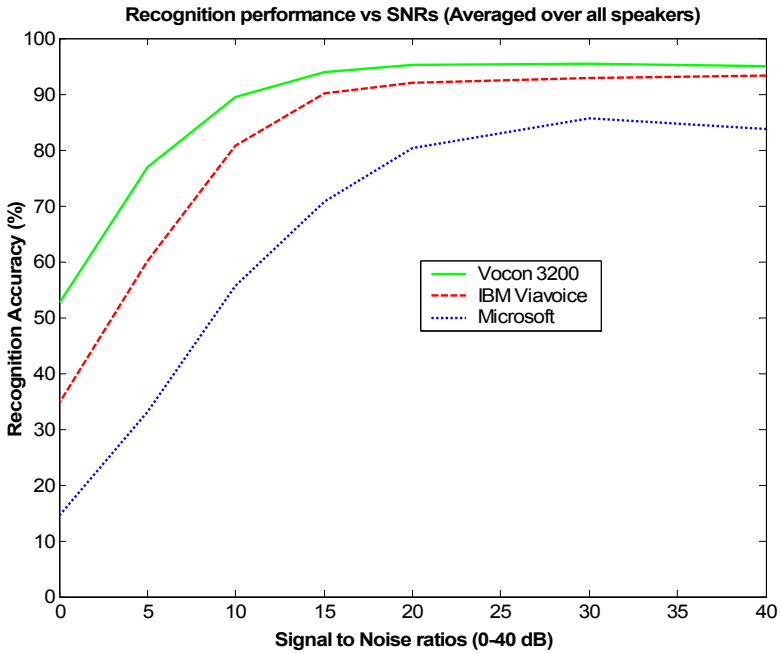


Figure 7.10: Recognition performance versus SNRs (averaged over all speakers and noise types)

The graphs show that Vocon 3200 ASR engine performs best for most speakers, especially under high ambient noise. IBM Viavoice performed better than Microsoft Command and Control in most of the cases. It can also be noticed that performance of Vocon 3200 was low for the non-American female speaker, but high for the non-American male speaker. But, when the recognition performance was averaged over all speakers, Vocon faired better than the others.

7.6.2. Comparison between Different Noise types for a Given ASR and Speaker

The performance of a given ASR to different types of noise is compared here. The x-axis specifies the SNR and the y-axis specifies the recognition rate. Each graph shows the results for Speaker A for a given ASR.

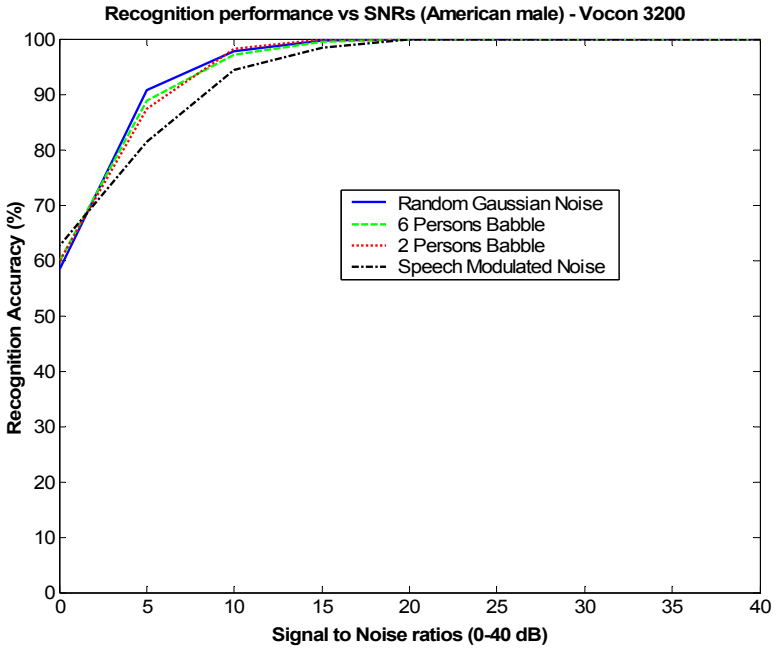


Figure 7.11: Speaker #4 – Vocon 3200 (American male)

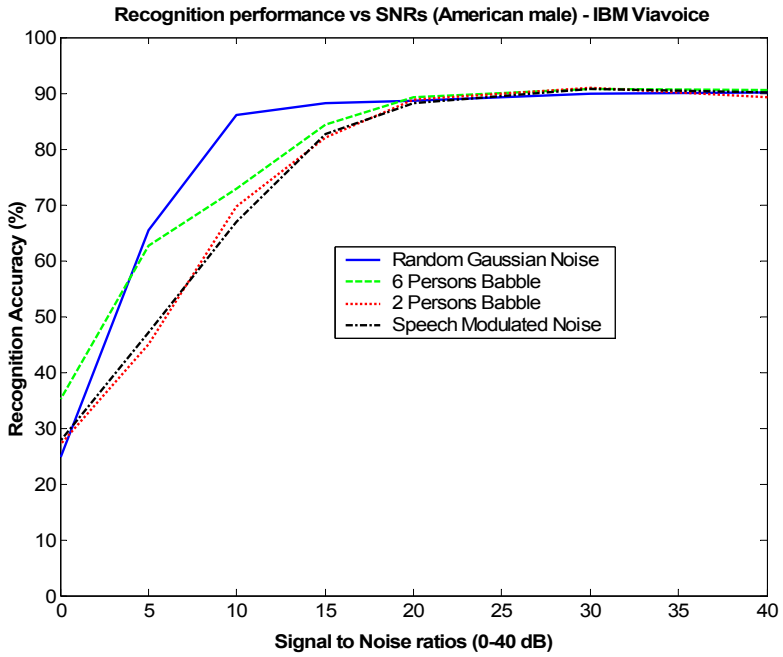


Figure 7.12: Speaker #4 – IBM Viavoice (American male)

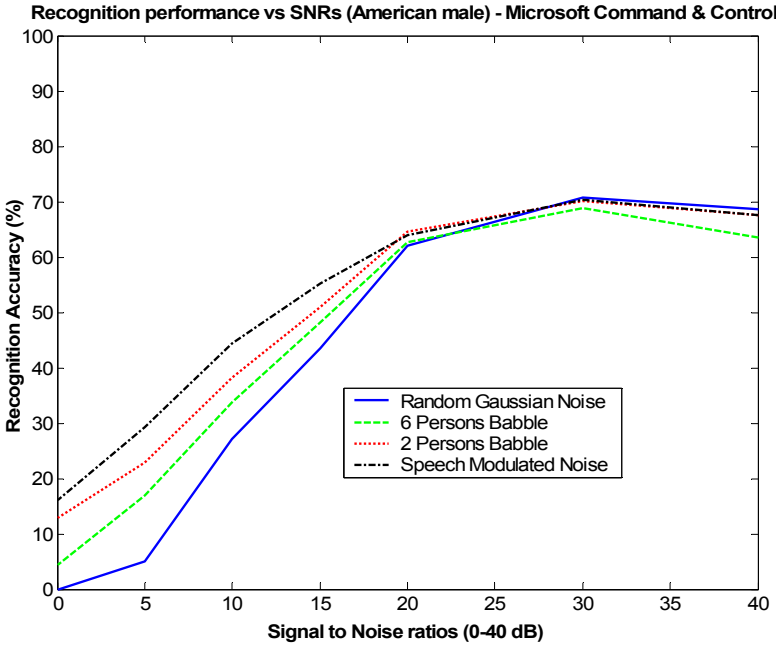


Figure 7.13: Speaker #4 - Microsoft Command and Control (American male)

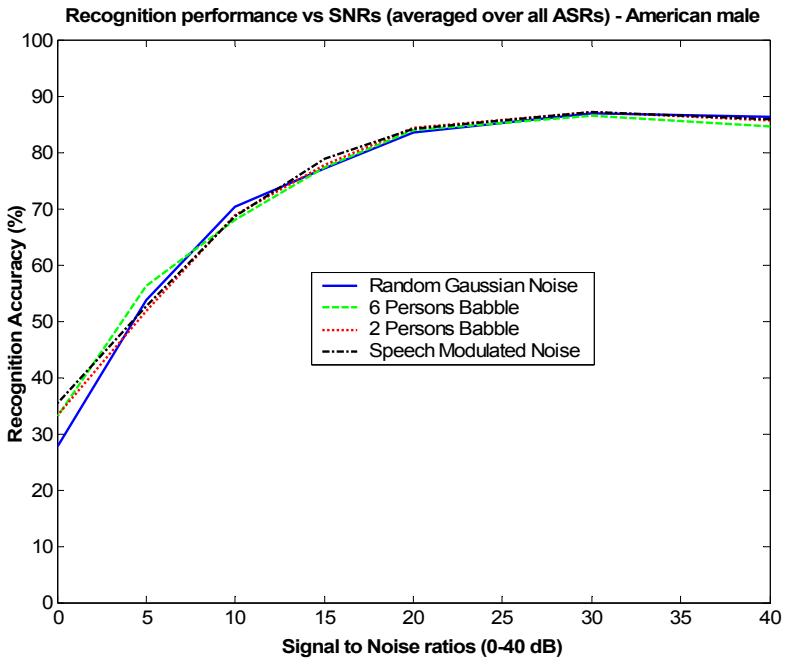


Figure 7.14: Speaker #4 - Averaged over all ASRs (American male)

The results show that the performance of Vocon 3200 does not vary much with noise types. But the performance of IBM Viavoice and of Microsoft Command and Control is dependent on the noise type. IBM Viavoice performs better under Random Gaussian noise compared to other noises, while Microsoft Command and Control performs poor under Gaussian noise especially under low SNRs. From Figure 7.14, it is evident that Speaker #4 can achieve a recognition rate of 80% or higher, if the SNR is 20 dB or higher.

7.6.3. Comparison of ASR Engines for a Given Noise Type

The performance of the ASR engines under a given noise is compared here. The x-axis specifies the SNR and the y-axis specifies the recognition rate. Each graph shows the results for Speaker #4 for a given noise.

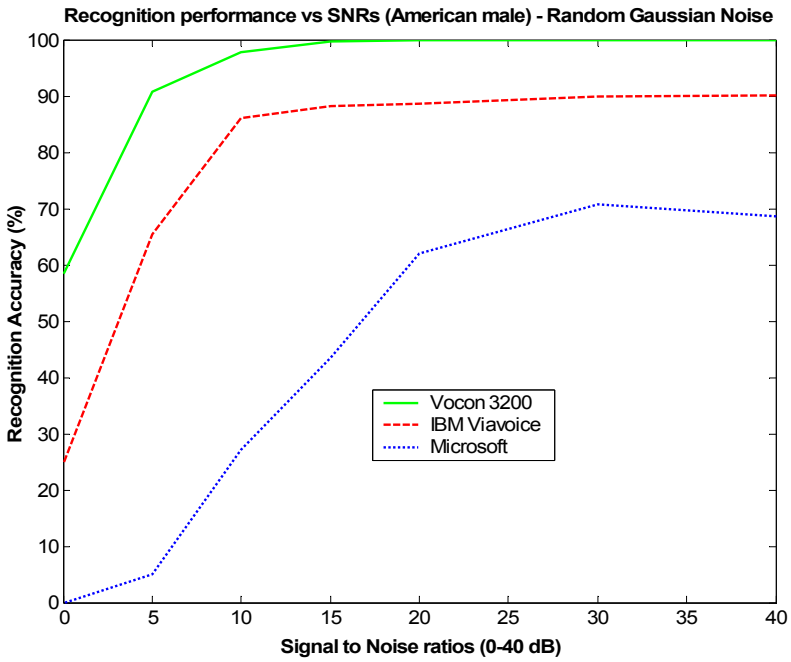


Figure 7.15: Speaker #4 (American male) – Random Gaussian Noise

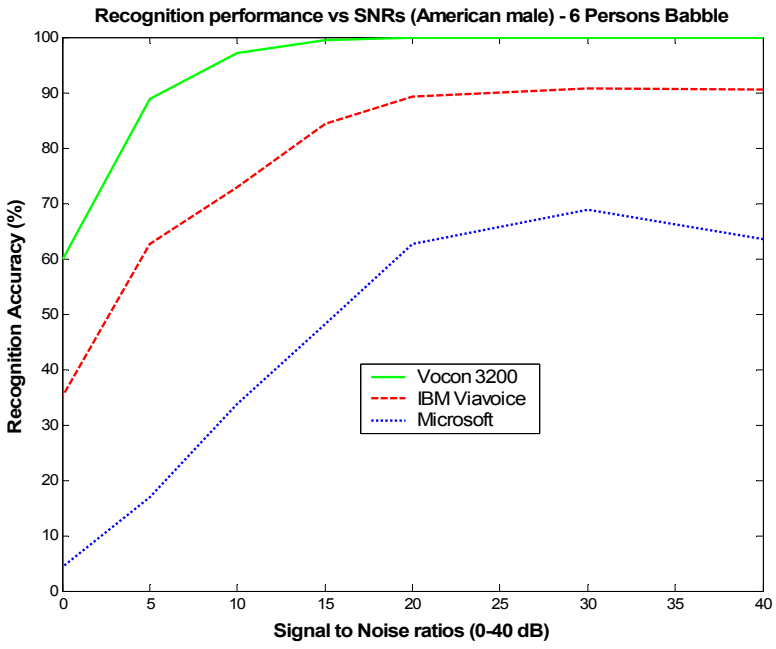


Figure 7.16: Speaker #4 (American male) – 6 Persons Babble

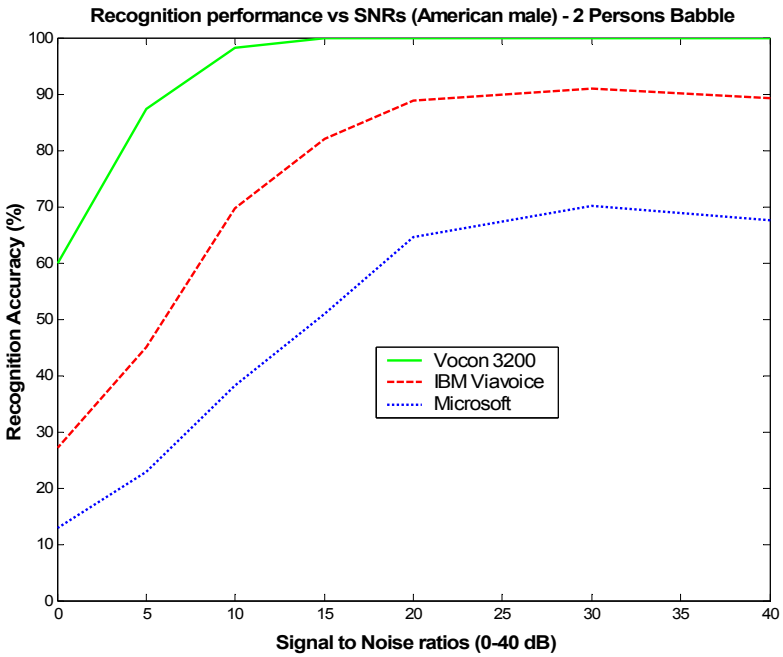


Figure 7.17: Speaker #4 (American male) – 2 Persons Babble

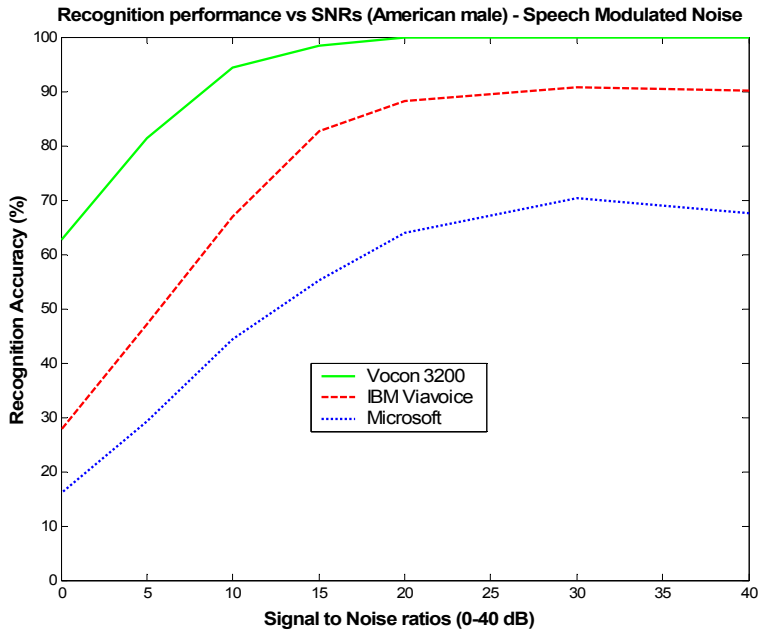


Figure 7.18: Speaker #4 (American male) – Speech modulated noise

These results show that Vocon 3200 performs best compared to other engines under all different types of noises for Speaker #4. For this speaker, Microsoft performed better than IBM Viavoice under all noises.

7.6.4. Dependency of Recognition Rates of ASRs as a Function of Signal Bandwidth

Tests were performed in order to analyze the relationship between the recognition rates of the ASRs and the bandwidth of the acquired voice commands. The setup for this testing is shown in Figure 7.19.

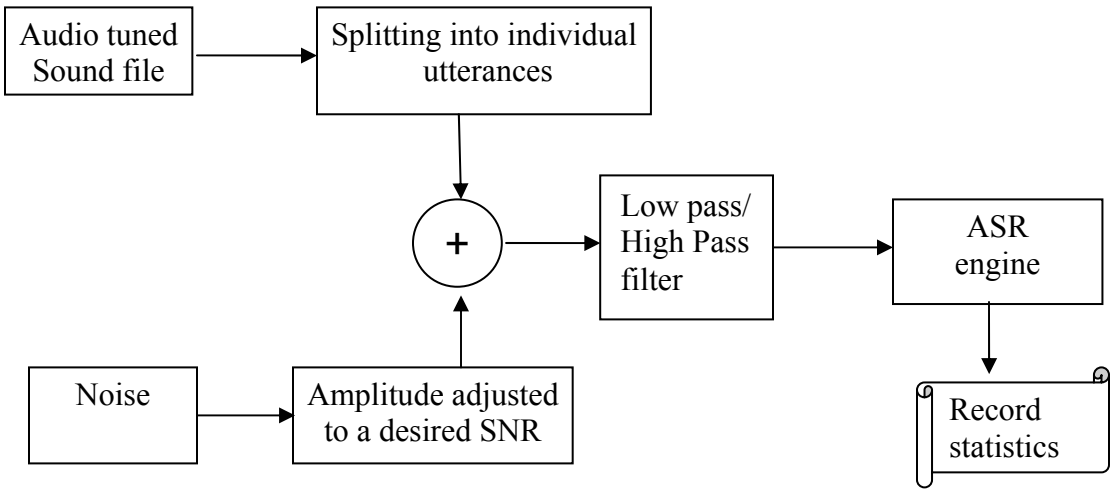
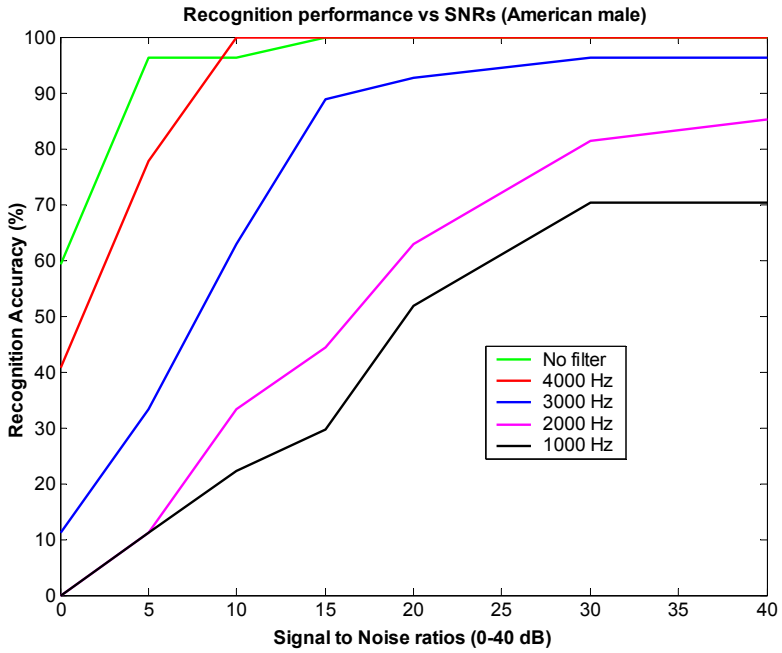


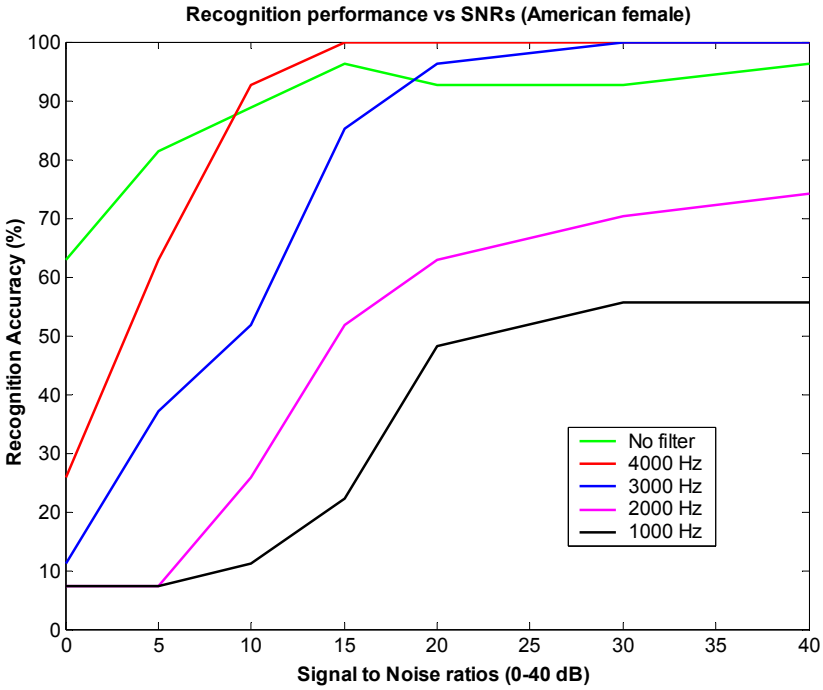
Figure 7.19: Test setup for determining the ASR dependency on speech sampling rates

Low pass or high pass filters were introduced between the ASR engine and the sampled speech in order to determine the effect of high frequency and low frequency respectively on the recognition rates. The low pass and high pass filters were chosen to be a Minimum-Phase Constrained Equiripple FIR filter, with stopband attenuation of 30 dB and passband parameter

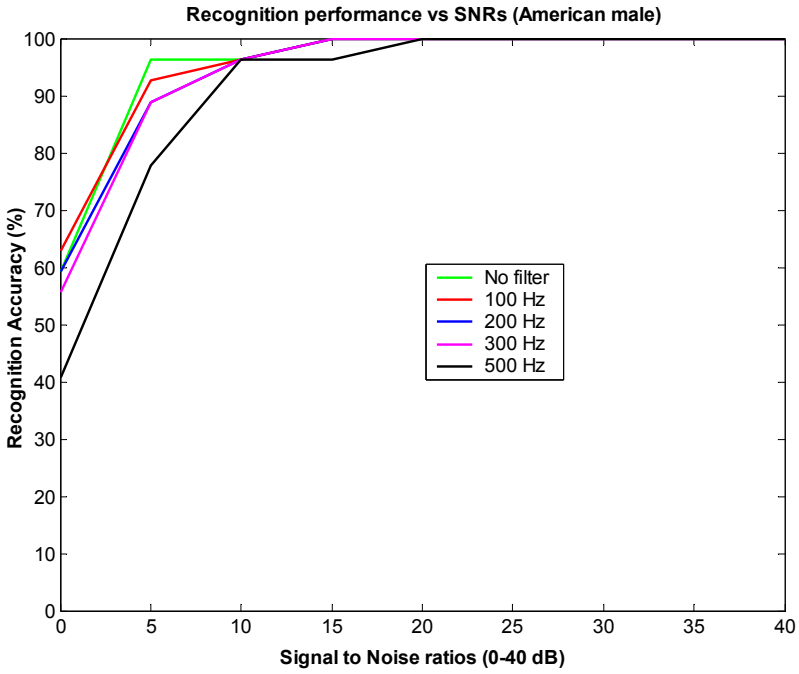
set to 1 dB. The order of the Low Pass Filter (LPF) was chosen to be 8 and for the High Pass Filter (HPF), it was chosen to be 21. LPFs were designed with passband cutoff frequencies of 4000Hz, 3000Hz, 2000Hz and 1000Hz. Different HPFs with passband cutoff frequencies of 100Hz, 200Hz, 300Hz and 500Hz were also designed. The analysis was performed on the recordings of Speaker #3 (American female) and Speaker #4 (American male), made in the quiet room with an airborne microphone (Emkay, model 3345) of reasonably wide frequency response. The recordings had a sampling rate of 11,025 kHz. Gaussian noise was added to each individual command utterance electronically, with the noise amplitude scaled to get different SNRs. The noisy speech is then passed through the low pass or high pass filter, and the result is fed to the Vocon 3200 ASR engine, as shown in Figure 7.19. Finally, the statistics are recorded. The results are plotted in Figure 7.20(a) and (b) for an American male and an American female respectively with an LPF. Figure 7.20(c) and (d) shows the corresponding results using a HPF.



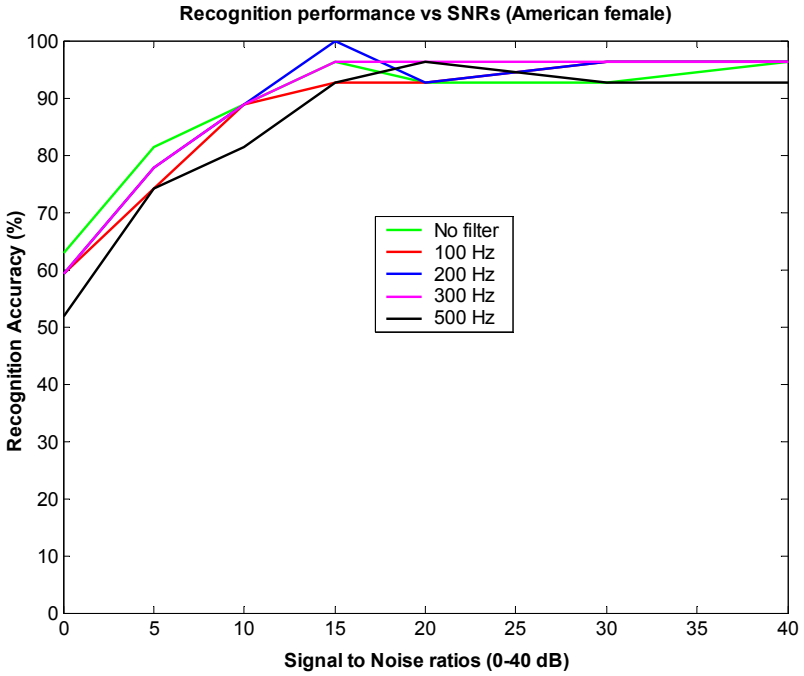
(a) Speaker #4 - Effect of recognition rate on high frequencies



(b) Speaker #3 – Effect of recognition rate on high frequencies



(c) Speaker #4 - Effect of recognition rate on low frequencies



(d) Speaker #3 - Effect of recognition rate on low frequencies

Figure 7.20: Effect of high and low frequencies in sampled speech on recognition rate, the analysis was done with Gaussian noise using Vocon 3200 ASR engine

Figure 7.20(a) and (b) indicate that the high frequency content in speech has an impact on the recognition rate. The results are similar for American male and American female speaker, with a slightly higher recognition rate for the male speaker with cutoff frequencies of 1000 Hz and 2000 Hz. The graphs show that when frequencies above 3000 Hz or 4000 Hz are filtered out, there is an appreciable fall in the recognition rate at low SNRs. When frequencies above 1000Hz or 2000Hz are filtered out, the fall in performance was evident at both high and low SNRs. These results reveal that the frequency response of the microphone has an appreciable effect on the recognition rate. Figure 7.20(c) and (d) shows that frequencies above 300-400 Hz are required to be present to get a good recognition rate. In other words, frequencies below 300 Hz do not have an appreciable effect on speech recognition. Therefore, on the basis of these experiments, it can

be concluded that the recommended bandwidth of a microphone suited for speech recognition should be between 300 and 5000Hz. In addition, the sampling rate of speech for ASR purposes has to be kept at or above 11 kHz, to ensure that good results are obtained.

7.6.5. Dragon English NaturallySpeaking Evaluation Results

Dragon English NaturallySpeaking was not subjected to a detailed testing with all speakers, because the initial tests carried out with it has not given promising results. Figure 7.21 shows the recognition performance of Dragon with a non-American female speaker (Speaker #6). It can be seen that the fall in performance is quite prominent even at high SNRs.

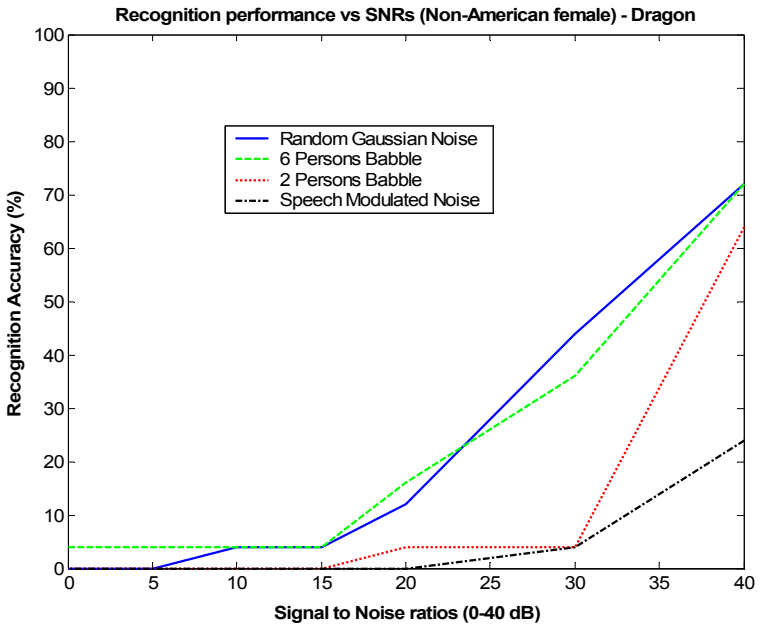


Figure 7.21: Dragon performance with different noise types (Non-American female)

7.6.6. Conclusion on ASRs

In general, Vocon 3200 has proved to give superior performance compared to the other engines. Therefore, it was selected for implementing speech recognition to the wearable ultrasound scanner system. The results indicate that it is suited for an environment with moderately high ambient noise. As the performance of Vocon is not found to be dependent on the noise type, it is expected that the recognition performance would stay consistent in a given environment.

The tests described here were performed using Vocon 3200 Version 1.2. Since the completion of the tests, Vocon 3200 Version 2.0 has become available. The main features of Vocon 3200 Version 2.0 are: speaker adaptation functionality to improve recognition accuracy for specific speakers, speed optimizations to reduce the overall CPU power consumption, better recognition accuracies than version 1.2. There have been significant changes made to the Speech APIs for Vocon Version 2.0 from Version 1.2.

Chapter 8. Microphone testing in noisy environment

The goal of this series of experiments is to determine which combination of microphone and *Automatic Speech Recognition* (ASR) engine will work best for the wearable ultrasound scanner system. Recordings are made from different speakers in an acoustic chamber, with noise of different types and levels played in the background. The acoustic chamber is highly reverberant, with a near-uniform sound pressure of non-directional sound. These recordings are then fed to the speech recognition engines and the recognition statistics recorded. We will examine each aspect of the testing in greater detail.

8.1. Microphones Tested

The different microphones selected for this testing are:

1. Emkay, model 3345 – a conventional airborne microphone.
2. Bluespoon - Bluetooth wireless microphone consisting of 2-microphone array.
3. Voiceguard – 4-microphone array, with adaptive directivity pattern for noise cancellation.
4. Physiological sensor microphone – detects speech from the vibration of the vocal cords captured from the skin surface of the throat.
5. Invisio – jawbone conduction microphone, which detects speech from bone conduction to the ear.

The details of each of the above microphones are discussed in detail below. The factors relevant for the selection of an appropriate microphone are:

- Easy to wear and comfortable to use
- Performance do not degrade in an environment with moderately high ambient noise
- Frequency response wide enough for speech recognition

Emkay, model 3345

This is a conventional air-borne microphone, recommended for speech recognition purposes. Figure 8.1 shows a picture of this microphone.

Figure 8.1: Emkay, model 3345

Emkay, model 3345 consists of a microphone and speaker and is monaural. It can be connected to any standard sound card. Tests were performed to analyze the frequency response of an airborne mike. A sweep signal of amplitude 1.0 V, frequency range of 10 Hz -10000 Hz, amplitude of 0.15 V peak-to-peak and of duration 1 second was generated and fed to two loudspeakers kept close to each other, with the airborne mike kept at 5 cm from the speaker. The microphone was connected to the Sound Blaster Audigy 2 NX sound card, which was connected to the laptop. Recording software running on the laptop records the sweep and saves it in the

form of a sound file. A Matlab script is used to plot the data in the sound file. Figure 8.2 shows the plot. The gain provided by the loudspeaker, sound card and the recording software were all kept fixed throughout the test. In this test, the frequency response measured is a product of the frequency response of the loudspeaker and that of the microphone. However, the plots obtained here are useful for a relative comparison between the different microphones.

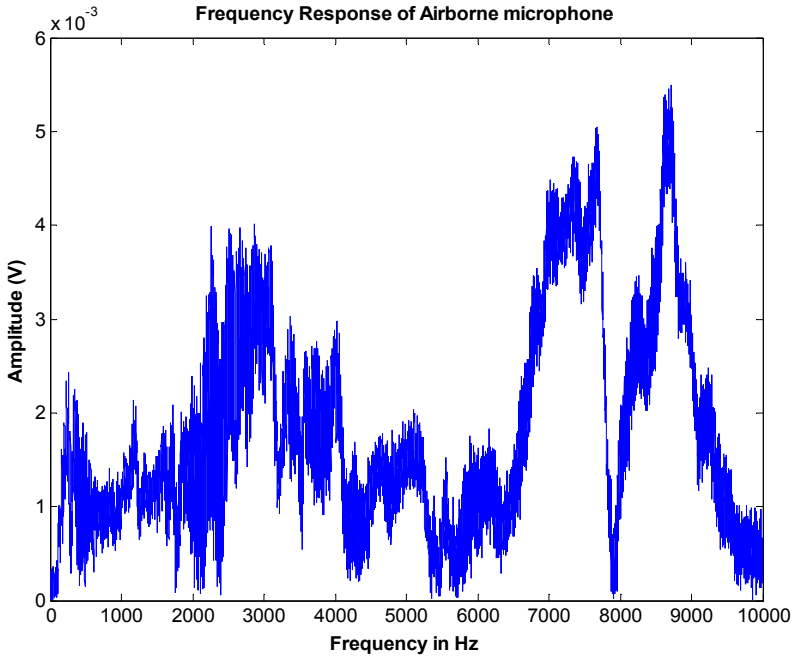


Figure 8.2: Frequency Response of Airborne Microphone

Bluespoon

This is a wireless bluetooth microphone, provided by *Nextlink,* Gentofte, Denmark. It is a 2-element array microphone and contains a speaker as well. It works together with a Bluetooth USB adapter, WML-C51APR provided by *Mitsumi*. Figure 8.3 shows a picture of Bluespoon.

Figure 8.3: Bluespoon Microphone

The USB adapter works with the help of Mitsumi Bluetooth software. The freely downloadable version 3.1 of the software, obtained from Mitsumi Europe website, supports Bluetooth headset profile, and is essential for Bluespoon to work properly. After switching on Bluespoon, the connection to the microphone from the laptop can be established using the Bluetooth software. It was also found that the sampling rate of the recordings made with Bluespoon needs to be kept at 8 KHz, to get acceptable sound quality. There exists no recording volume control feature in Bluespoon. It provides a fixed digital gain for recording. In the commercial version of Bluespoon, the digital gain provided was too low for speech recognition. Therefore, the digital gain was customized by the manufacturer for our needs.

The sweep signal test as mentioned earlier in this section was carried out for Bluespoon also. Figure 8.4 shows the plot. A comparison can be made with Figure 8.2, to analyze the difference in frequency response with the airborne. It can be observed that Bluespoon has a cutoff frequency of around 3000 Hz. Its speech recognition capability has to be still investigated.

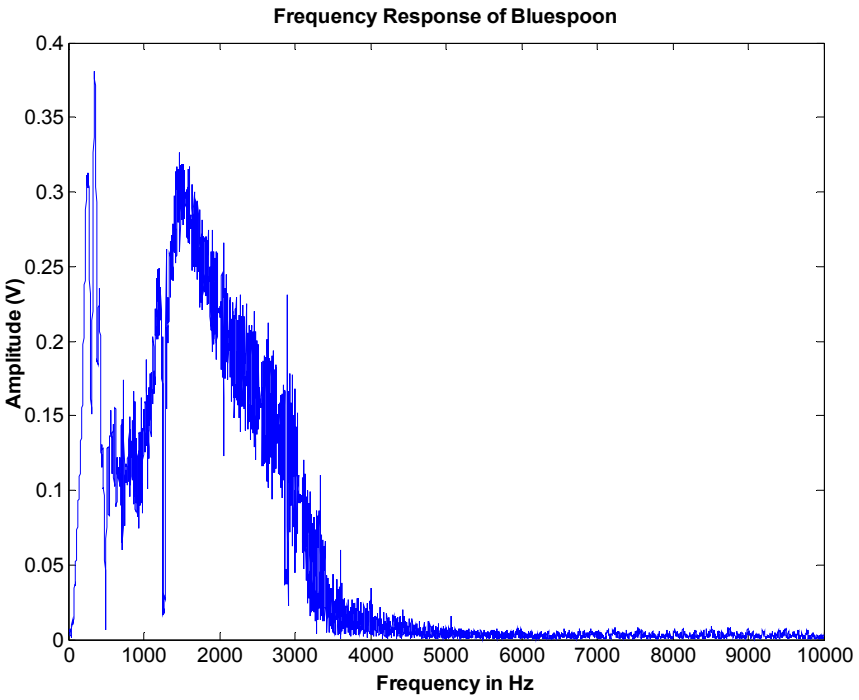


Figure 8.4: Frequency response of Bluespoon

Voiceguard

This microphone contains 4 array elements and is produced by *Planning System Incorporated* (PSI), Reston, VA. It is shaped in the form of a pen and can be worn in a shirt pocket. It is an endfired microphone, i.e., it is most sensitive towards the end which points to the speakers' mouth. It has an adaptive noise directivity pattern and is effective in shielding off directional noise. It performs poorer in diffuse noise than in directional noise. It is likely to pick up more noise than a conventional airborne microphone, as its location is farther away from the mouth. Figure 8.5 shows a picture of Voiceguard.

Figure 8.5: Voiceguard microphone

The Voiceguard microphone has to be connected to an Audio4-USB box also provided by *Planning Systems Incorporated*. The Audio-4 USB has two outputs. One is *Aout* which provides Hybrid Adaptive Beamforming (HAB), a technique proprietary to the manufacturer. The other output is *Bout* which provides HAB plus a gain control mechanism, also proprietary to the manufacturer. For all the testing described in this chapter, the output provided by *Aout* is used. In its commercial version, the output from the Audio4-USB box goes into the line input of the laptop. It was observed that the signal obtained in this way was very low in amplitude and not suitable for speech recognition. Therefore, the analog gain was adjusted by the manufacturer, so that it lies within the microphone input range of the laptop. Therefore for all the testing described here, *Aout* output was connected to the microphone input of the soundcard. Any standard sound card can be used. Some sound cards like SoundBlaster Audigy 2 NX provide a dc bias voltage of 5V at their microphone input. This bias voltage may damage the Audio4-USB if connected directly through stereo connectors, and therefore a custom-made connector must be made to connect the Audio4-USB *Aout* to the microphone input of the sound card. This connector will have the bias voltage wire disconnected internally.

In directional noise, the adaptive array processing techniques of Voiceguard microphone is specified to give an improvement in SNR of around 25 dB, as compared to an omnidirectional microphone. The corresponding improvement in SNR in the presence of diffuse noise is around 10 dB. The sweep test was carried out for Voiceguard microphone also, and Figure 8.6 shows the result. As can be verified from Figure 8.6, the Voiceguard microphone has a cutoff frequency of around 5000 Hz. However, this range of frequencies appears to be adequate for speech recognition purposes.

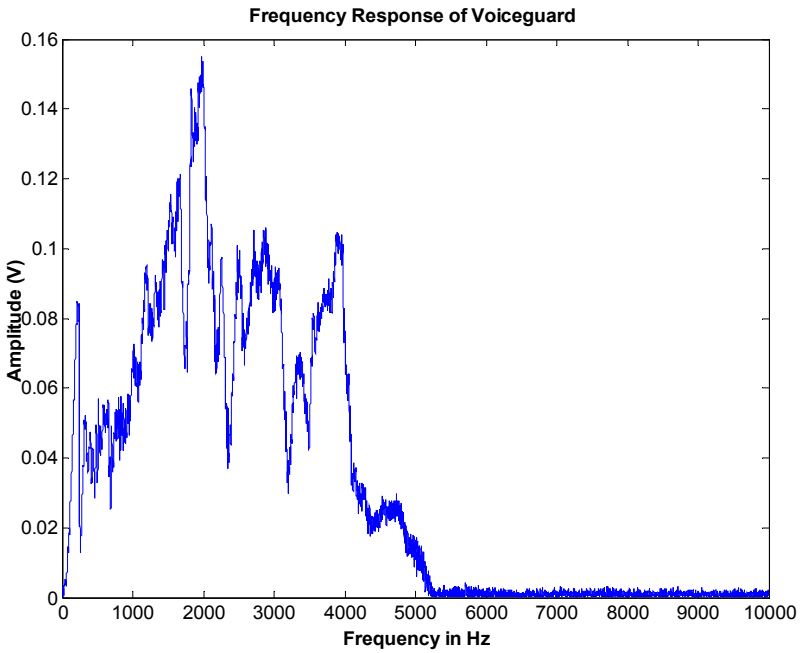


Figure 8.6: Frequency Response of Voiceguard

Physiological Sensor

Figure 8.7 shows the physiological sensor microphone. The sensor consists of a fluid or gel contained within a small, conformable, rubber bladder or pad that includes a hydrophone. This enables the collection of high SNR cardiac, respiratory, voice and other physiological data. The voice is detected from the vibration of the vocal cords conducted through the soft tissue to the skin surface of the throat. The pad also minimizes interference from ambient noise because it couples poorly with airborne noise. It is therefore meant for dual purpose, speech recognition and medical monitoring. This is manufactured by *RadioEar Corporation*, New Eagle, PA and developed by *Army Research Laboratory* (ARL). More details can be found in [20].

Figure 8.7: Physiological sensor microphone

The signal from the throat microphone has amplitude of a few millivolts range, and is therefore amplified by a custom-made pre-amplifier, so that the output lies within the microphone sensitivity range of the sound card (20 – 200 mV). The frequency characteristics of the pre-amplifier were kept such that the final output resembles the frequency response of a conventional microphone. The amplified signal is then fed to the microphone input of the sound card. Any standard sound card can be used.

Invisio

This is a jaw bone conduction microphone, developed by *NextLink*. It is designed such that it can be fitted precisely in the ear each time it is worn. It detects speech from vibrations picked up from the ear. Invisio also provides shielding against airborne sounds such as ambient speech and noise. It contains a microphone and also a loud speaker. Figure 8.8 shows the picture of Invisio.

Figure 8.8: Invisio microphone

8.2. Speech Recognition Engines

The speech recognition engines used for this testing are:

- Vocon 3200 Version 1.2 from Scansoft
- IBM Viavoice Command and Control
- Microsoft Command and Control

The factors relevant for the selection of speech recognition engines and the respective speech recognition engine details were dealt with in greater detail in Chapter 7.

8.3. Test Description

8.3.1. Acoustic Chamber

The testing was performed in an acoustic chamber, housed in the Higgins House Garage on WPI Campus. The test chamber is highly reverberant, with speakers mounted on the walls, and will thus produce a near-uniform sound pressure of non-directional sound. Figure 8.9 shows pictures of the inside and outside of the acoustic chamber.

(a) (b)

Figure 8.9: Acoustic test chamber facility, (a) inside (b) outside

The speaker participating in the testing sits alone inside the chamber (Figure 8.9(a)). There are holes at the side of the chamber for running the wires to outside of the chamber. The laptops are placed outside the chamber as shown in Figure 8.9(b) for performing the tests.

8.3.2. Test Parameters

Recordings were made from 6 different subjects under 2 different types of noises, at 3 different *Sound Pressure Levels* (SPLs), and also under quiet conditions. The noise types selected resembled forms of blended human speech. The command set used for the recording consisted of 27 command words, which are listed later in this section.

Noise Types

The noise types were chosen from an ICRA noise CD, produced by Wouter A. Dreschler [18]. The two different noise types selected for this testing are described below:

Noise type I: Unmodulated random Gaussian noise, with spectral weighing for male voice

Noise type II: 6 Persons babble (a male and a female at a short distance and two females and two males at a greater distance)

Speakers

Six subjects were selected for the testing. Of the six subjects, four are American born, and two are foreign born. The subjects are all between 20 and 30 and in good health. Of the four American born subjects, three are males and one is female. Of the foreign born subjects, one is male and the other is female. The different subjects are listed below:

Speaker #1 – Non-American male,
Speaker #2 – American male,
Speaker #3 – American female,
Speaker #4 – American male
Speaker #5 – American male,
Speaker #6 – Non-American female

The subjects all signed a document, stating that the participation is voluntary, that their identity will be kept confidential and that they will not be subjected to SPLs above 80 dB, unless provided with hearing protection. This document is reproduced in Appendix F.

Command Set

The test vocabulary consists of 25 command words and two header words, for a total of 27 words. The command words all consists of the word 'Terason' followed by one of the actual command for the Terason ultrasound scanner. The chosen command set is as given below:

Terason testing
Terason testing
Terason view
Terason image
Terason small size
Terason medium size
Terason set focus
Terason change gain
Terason toolbars
Terason depth ruler
Terason live
Terason brightness
Terason motion
Terason full screen
Terason open exam
Terason power doppler
Terason probe info
Terason standby
Terason hibernate
Terason invert image
Terason color rainbow
Terason scan mode
Terason display pelvic
Terason help topics
Terason measure distance
Terason new patient
Terason frame rate

The two repetitions of the word 'Terason testing' in the beginning could be considered as the 'header', as they are not included in the recognition count, while determining the recognition accuracy. This is because in this series of experiments, the ASR engine is configured to receive input directly from the microphone. Due to this reason, the ASR engine takes some time to adapt to the speaker, microphone and the environment. It can be noted that the headers were included in the final recognition count for the testing described in Chapter 7, although the same set of command words were used in both cases. For the testing in Chapter 7, the speech engine received the input directly from sound files, and therefore this correction was not required.

Noise Levels

Sound Pressure Level (SPL) is used as the basis for measuring noise level. The SPL is the ratio between the actual sound pressure and a fixed reference pressure. The fixed reference pressure is

usually that of the *threshold of hearing* which has been internationally agreed upon as having a value of 0.0002 $dynes/cm^2$. The SPL was measured using Quest Electronics SPL meter, Model-215 with A-weighting settings. A-weighting indicates the weight filter that is applied in order to tailor the frequency response of the SPL equipment in such a manner as to simulate the ear's perception of the sound.

Three different SPLs were used for each of the two noises selected for the testing. The SPLs were chosen such that a speech recognition system, when receiving input from an air-borne microphone, will recognize most of the spoken words (approx. 85%) at the lowest of the three SPLs and will only recognize some of the spoken words (approx. 15%) at the highest SPL. In preparation for this, tests were carried out in a quiet room, with recording of sound files of a volunteer reading the 27 command words using an airborne microphone. Noise was added electronically to the recorded sound files at various levels, before being sent to a speech recognition engine. Microsoft command and control ASR engine was used for this testing. The recognition accuracy versus SNRs obtained from this testing is shown in Figure 8.10.

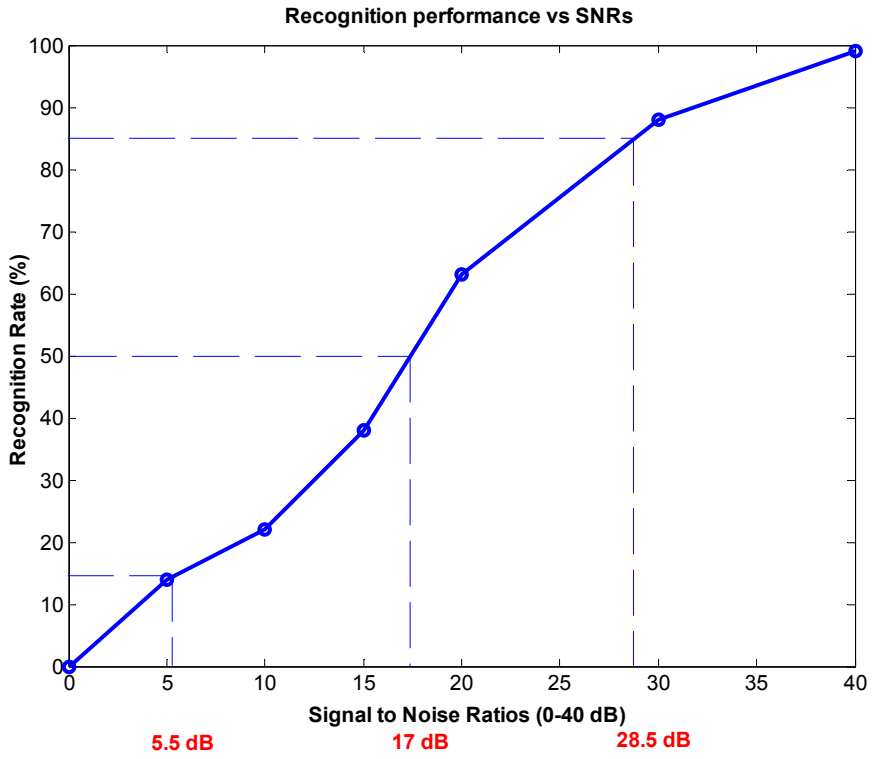


Figure 8.10: Recognition accuracy versus SNRs for Microsoft

From the result, it was determined that the 85% recognition rate, the 50% recognition rate and the 15% recognition rate were achieved at approximately SPL noise levels of REF dB, REF + 12 dB, and REF = 24 dB, respectively. However, to determine the approximate SPL level corresponding to the REF dB, similar tests had to be repeated in the acoustic chamber as well. For this, Gaussian noise was recorded on a CD and then played through the power amplifiers in the acoustic chamber. A subject wearing an airborne microphone sat inside the chamber and spoke all the 27 commands with 5 seconds duration in between. The laptop recorded this speech and sent it to Microsoft Command and Control ASR engine for evaluation. The above process was repeated with the gain of the power amplifier adjusted each time, and the output of the ASR engine recorded. It was observed that the ASR engine gave a recognition rate of 50% at an SPL level of 104 dB. The same test was repeated with the Voiceguard microphone, and a recognition

rate of near 15% was obtained at an SPL of 80 dB. Therefore, to obtain a realistic evaluation of all microphones, the SPLs were selected to be as 68 dB, 80 dB and 92 dB.

8.3.3. Test Layout

Figure 8.11 shows the experimental details of the microphone testing. The speaker sits inside the acoustic chamber wearing the microphones. Noise is played through the loudspeakers mounted on the walls. When noise achieves a near uniform sound pressure inside the chamber, the speaker starts speaking one command after another with slight pause in between. A cuing device with an LED mounted inside the chamber, indicates to the speaker when to speak a command. The laptops connected to the respective microphones create the recordings in the form of sound files.

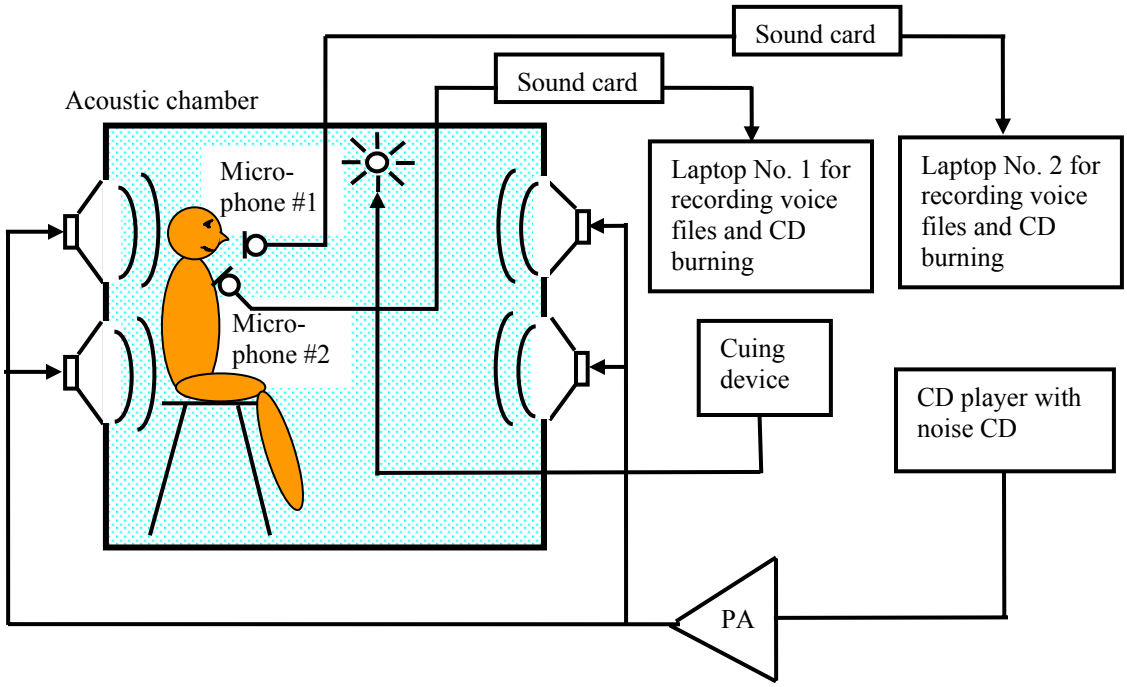


Figure 8.11: Layout of the testing system

To save time, each speaker was made to wear two microphones at a time. The pairing of the microphones was done such that the functioning of one microphone will not interfere with that of the other microphone, with respect to their position or speech capture. The microphones were paired as follows:

Emkay + Physiological sensor microphone

Voiceguard + Bluespoon

Invisio

The noise input to the power amplifier, driving the speakers in the acoustic chamber, is obtained from a *noise CD* being played in a CD player. The CD contains six noise tracks, consisting of two tracks at the REF dB level, two tracks at the REF + 12 dB level, and two tracks at the REF + 24 dB level (Section 8.3.4). The output control of the CD player and the gain setting of the power amplifier have previously been adjusted to produce the required SPL inside the chamber. The gain setting of the power amplifier and CD player gain setting were then left undisturbed throughout the testing process. Speakers are made to wear earplugs when SPL was above 80 dB, except in the case of Invisio which provides around 15 dB protection of its own.

Each of the microphones is connected through sound card to their respective laptop. The sound card used is Sound Blaster Audigy 2 NX. A software program called *IntelligentRecorder* is written has an user interface as shown in Figure 8.12, to make the recording process much easier, by organizing the sound files of each speaker under respective directories each time it is made. The speaker's name and the microphone name at any time can be selected on the window. When a recording has to be made, the button corresponding to that specific recording is selected. The

duration of each recording can also be specified in the adjacent box. This brings up the software 'TotalRecorder' from *HighCriteria Inc* and automatically records the speech for the specified number of seconds. When done, it saves the recording in the form of sound files in 22 KHz 16 bit mono PCM format. The sound files created follow the directory structure './SpeakerName/MicrophoneName/TypeofRecording'. The recordings of each speaker for all the five microphones are stored finally on to a *test CD*. The exact tracks on the test CD are described in Section 8.3.6.

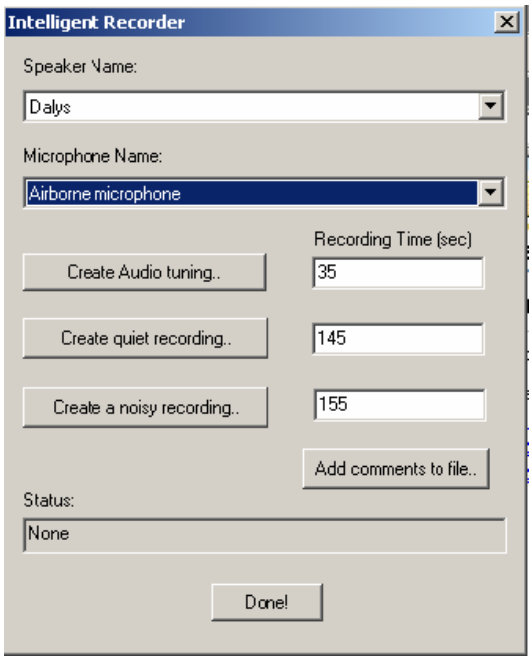


Figure 8.12: Recorder interface for the testing

A *cuing device* is connected to an LED mounted inside the acoustic chamber as well as an LED, mounted next to the laptops and CD player, outside the acoustic chamber. The LED flashes every five second for 0.5 sec., in order that the subject reads the command words at the precise

moments and so that the duration of the noise tracks corresponds to the time that the subject takes to read the 27 command words. More details are given in Section 8.3.5.

We will examine the speech signal flow in the acoustic chamber in greater detail. During the recording sessions in the acoustic chamber, the signal flow occurs as shown in Figure 8.13.

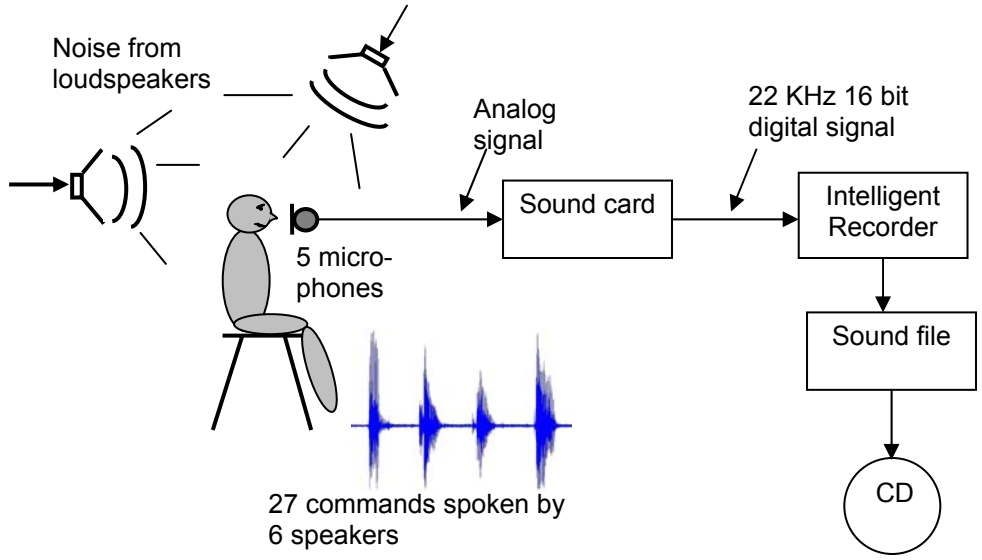


Figure 8.13: Signal flow during recording

The microphone captures the speech and sends as analog signal to the sound card. The sound card digitizes it at the specified format. *Intelligent Recorder* gets this information from the sound card and stores it on to a sound file. This sound file is finally stored on to a test CD.

Once all the recordings sessions are completed and the test CDs made, the test CD of a specific speaker is placed in one laptop, and the audio output of the sound card of that laptop is applied to the microphone input of another laptop, containing the speech recognition software packages (Section 8.3.8).

8.3.4. Layout of Tracks on Noise CD

As mentioned in Section 8.3.3, the noise for the testing is played by a CD containing the following tracks:

Track 1: Noise type I: Unmodulated Gaussian noise, at level REF dB + 0dB

Track 2: Noise type II: 6 Persons babble, at level REF dB + 0dB

Track 3: Noise type I: Unmodulated Gaussian noise, at level REF dB + 12dB

Track 4: Noise type II: 6 Persons babble, at level REF dB + 12dB

Track 5: Noise type I: Unmodulated Gaussian noise, at level REF dB + 24dB

Track 6: Noise type II: 6 Persons babble, at level REF dB + 24dB

Each track on the noise CD is of 150 sec duration. Once a track on the CD starts playing, 5 seconds are given for the noise to reach a near uniform level inside the chamber. Then the cuing device lights in every 5 seconds, indicating the speaker to speak a command. After the 27 commands are spoken, a 15 seconds quiet time is present during which the background noise is recorded. Therefore the time for each track = 5 + 26 * 5+ 15 = 150 seconds.

The tracks on the noise CD are created so that there is no clipping at the highest level (which occurs with track 6). At the same time, we ensure that the proper dB level is found for each track by calculating a relative dB level in the form of 20 log(v_{RMS}), where v_{RMS} is found as follows:

$$v_{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}[v_i]^2 \Delta t}$$

In the above expression, Δt is the sampling interval and v_i are the samples of the noise signal.

8.3.5. Cuing Device

To make sure that the subjects, whose voice is being recorded, speak each word in the list with a consistent and correct time interval, a cuing device has been designed and built, consisting of a timer circuit, a microprocessor, an LED driver and two LEDs. The cuing device is placed outside the acoustic chamber and next to the CD recording device, and one LED is mounted inside the acoustic chamber and one LED on the cuing device itself. The LEDs will produce a short green-colored blink every 5 seconds, as an indication that the subject is to read the next word in the list. More specifically, the cuing device will be turned on exactly at the moment when a given track on the noise CD is started; the LED display will produce the first blink 5 seconds after being turned on, and then will produce an additional 26 blinks with 5 seconds interval thereafter, followed by a red light as a signal to the subject that the current test has been completed. The cuing device will be turned off when one track on the noise CD is completed. The light output is shown in Figure 8.14.

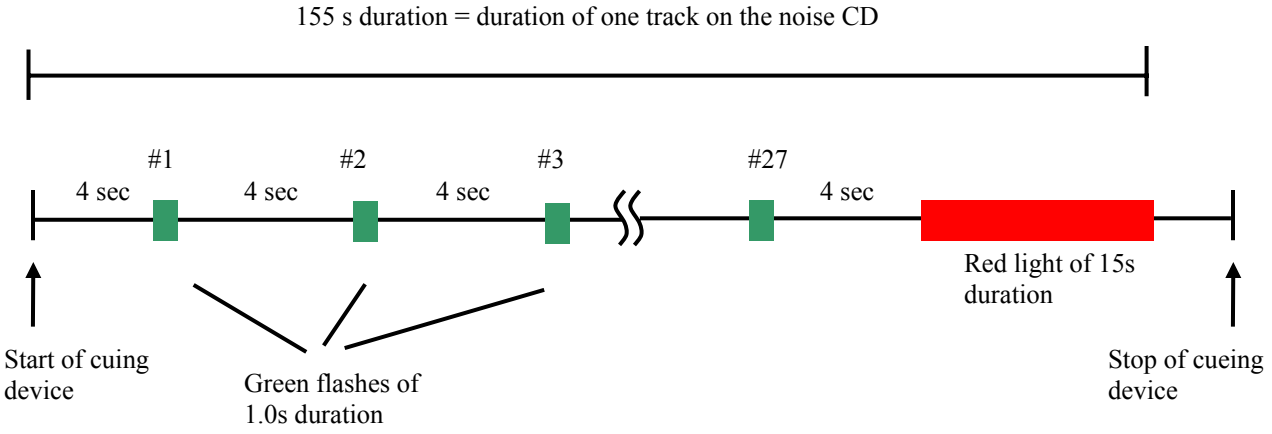


Figure 8.14: Sequence of light from the cuing device

8.3.6. Recordings to be made

After the subject has become familiarized with the testing protocol and with the 27 command words (by reading the set of words aloud a couple of times for practice), the subject was seated in the acoustic chamber, as shown in Figure 8.11. The subjects were asked to speak with nearly the same loudness even if noise is playing in the background. The subjects were shown how to wear the microphones and they were also provided with earplugs. When they were ready, the following sequence of voice files were recorded on both the laptops. The name of the sound file given here, is the name given by the *IntelligentRecorder* to the corresponding file. For every speaker and microphone, *IntelligentRecorder* organizes the files under the directory structure, './SpeakerName/MicrophoneName'.

Audiotuning1.wav: The subject reads thirty seconds of a specific text passage, under quiet conditions, to be used for audio tuning of the speech recognition system. During this time, the cuing device is off, and the noise CD is not played.

QuietRecording1.wav: The subject reads the 27 command words under quiet conditions. Thus, the cuing device is on and produces the sequence of light signals shown in Fig 8.13, but the noise CD is not played.

NoiseRecording1.wav: The subject reads the 27 command words while track 1 of the noise CD is played (noise type I: unmodulated Gaussian noise, at level REF dB + 0dB). The cuing device is on and produces the sequence of light signals shown in Figure 8.14.

NoiseRecording2.wav: The subject reads the 27 command words while track 2 of the noise CD is played. The cuing device is on and produces the sequence of light signals.

- - - - -

- - - - -

NoiseRecording6.wav: The subject reads the 27 command words while track 6 of the noise CD is played. The cuing device is on and produces the sequence of light signals.

Each subject takes around 21 – 23 minutes to complete one session, i.e., to collect the data for two microphones. Each subject repeats the same session twice more, to get data from all 5 microphones and thus completing the whole testing. Thus, a total of around 1.5 hours is required from each subject to perform the testing.

The recorded voice files are subsequently stored on CDs, with one CD per speaker.

8.3.7. Microphone and Recording Software Settings for Testing

This section describes the various settings used for creating the recordings and can be used as a checklist if a recording has to be reproduced with a given microphone in the acoustic chamber. To get proper recording gain adjustments for each microphone, the recording software *TotalRecorder* has to be first configured properly and then shut down, before starting the software *IntelligentRecorder*. All the software required for creating the recordings is provided in the attached CD under the directory ‘./noisychamber_testing’.

Bluespoon

The laptop must first be bluetooth enabled, by connecting the USB adapter and installing the correct Bluetooth software version on it. After this is done, an icon appears on the laptop indicating the Bluetooth connection status. When a bluetooth device is not yet connected, this icon is white in blue background. After a connection is established between Bluespoon and the laptop, this icon changes its color to green. It can be verified now that the recording device and the playback device on the laptop are automatically set to 'Bluetooth Audio'. The following procedure has to be followed before making a recording.

1. Go to Control Panel/Sounds and Audio Devices. Check whether the recording and playback devices are set to Bluetooth Audio.
2. Open *TotalRecorder*. Check whether the Recording device is set to 'Bluetooth Audio'. Enable the 'Use directly' flag. Under 'Recording Source and Parameters', ensure that 'Analog Connector' is selected. Note that the recording level cannot be adjusted for Bluespoon. Make the sampling frequency to 8KHz 16 bit mono PCM format.
3. Shut down *TotalRecorder* and start up *IntelligentRecorder*. Select the appropriate speaker name and microphone name on *IntelligentRecorder*.

Once recordings are done, Bluespoon can be disconnected from the laptop. The icon on the laptop will now turn back to white.

Invisio

1. Go to Control Panel/Sounds and Audio Devices. Check whether the recording and playback devices are set to Sound Blaster Audigy 2 NX.

2. Open *TotalRecorder*. Check whether the Recording device is set to 'SoundBlaster Audigy'. Enable the 'Use directly' flag. Under 'Recording Source and Parameters', 'Line In/Mic In' has to be selected. Recording format can be set to 22KHz 16 bit mono PCM format. Recording level can be kept at -10 dB.
3. On the Sound Blaster Audigy 2 NX, the analog gain adjustment for microphone input was kept at its maximum.
4. Shutdown *TotalRecorder*. Start up *IntelligentRecorder*. Select the speaker's name and microphone name appropriately.

Physiological Sensor Microphone

All the settings provided for Invisio can be followed, except that the recording level in *TotalRecorder* software can be kept at -20 dB.

Airborne Microphone

All the settings provided for Invisio can be followed, except that the recording level in *TotalRecorder* software can be kept at -20 dB.

Voiceguard

The Voiceguard microphone, received from *Planning Systems Incorporated*, came with two power options. It can either be powered directly from the battery or it can be powered using a laptop via an USB-connect. When Voiceguard was powered directly from the battery, some random spontaneous spikes were observed on the speech waveform. Therefore, it is recommended to power it from the laptop. All settings provided for Invisio can be followed for Voiceguard except:

1. On SoundBlaster Audigy, the analog gain for microphone input, is set at 75% of the maximum recording gain.

Two laptops were employed for the recording, Dell Latitude C840 (Laptop 1) and Sony VAIO V505A (Laptop 2). The recordings were taken in three separate sessions, and the sequence followed and the pairing are given in Table 8-1.

Sessions	**Laptop 1 (Dell)**	**Laptop 2 (Sony VAIO)**
1	Throat	Airborne
2	Bluespoon	Voiceguard
3	Invisio	-

Table 8-1: Microphone order and pairing

8.3.8. Microphone and ASR Evaluation Layout

Once the recordings in the acoustic chamber are completed, the evaluation of these recordings using the ASR engines will begin. It should be mentioned that there will be a quality check for all the recordings. All the recordings are listened to and verified using Matlab to make sure that they are of the expected quality.

The evaluation of the recordings will be performed as illustrated in Figure 8.15. The CD containing the voice files is played on one laptop computer, and the analog audio output is fed to the microphone input/line input of a second laptop computer. The selection of the microphone input or line input is based on the amplitude range of the signal coming from the first laptop. It

can be checked whether the audio tuning software running on the second laptop gets the full dynamic range of the recording gain adjustment. The second laptop runs the ASR engines to be tested. The software for the evaluation of the recordings is provided in the attached CD.

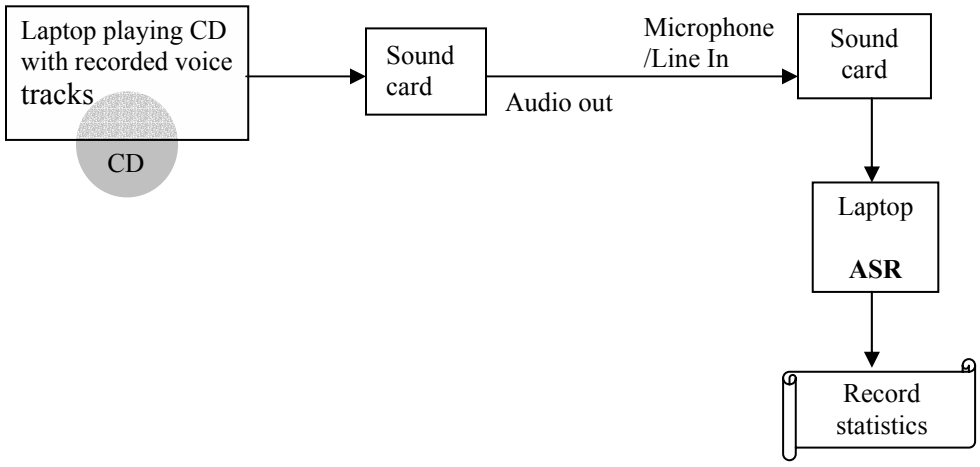


Figure 8.15: Signal flow in evaluation

The 30 second of reading on track 1 is used for audio tuning, after which the remaining seven tracks are played and sent to the chosen ASR engine. The statistics are recorded and finally analyzed.

It has to be mentioned that for all the tests in this chapter, the ASR engine was used in the always-on mode. i.e., the ASR engine remains active when an entire recording is played. In other words, irrespective of whether a command is present in the recording at any time or not, the ASR engine continuously tries to recognize speech. Therefore, the ASR engine gets activated just before the first command is uttered, continuously trains itself while the recording is being played, and then is deactivated shortly after the last command is uttered. This was true for all the 3 ASR engines used.

8.3.9. Evaluation Procedure

The different steps followed for conducting the evaluation of the recordings is given here.

1. Set up the laptop and the sound cards as in Fig 8.15. The sound cards used are Sound Blaster Audigy 2 NX. By default, when Audigy is switched on, the *Creative Multi Speaker Surround* (CMSS) feature, indicated by the yellow light glowing on the top is enabled. The CMSS feature reads specific bands of frequencies from an audio file and places them into unique speaker positions, to create a virtual surround effect. The button corresponding to it has to be switched off, so that the sound card does not introduce any undesired audio effects into the speech signal. By default, the line input of Laptop 2 was used.

2. All the subjects taking part in the testing where assigned a number. The microphones were assigned numbers in the following manner:

Microphone 1 – Airborne
Microphone 2 – Invisio
Microphone 3 – Physiological sensor
Microphone 4 – Bluespoon
Microphone 5 – Voiceguard

Numbers were also assigned for Noise types and SPLs. The sound files recorded in the chamber has the following noise type and noise level numbers.

NoiseRecording1.wav – Noise type 1 (Gaussian Noise), Noise level 1 (68 dB)
NoiseRecording2.wav – Noise type 2 (6 Persons babble), Noise level 1 (68 dB)
NoiseRecording3.wav – Noise type 1 (Gaussian Noise), Noise level 2 (80 dB)
NoiseRecording4.wav – Noise type 2 (6 Persons babble), Noise level 2 (80 dB)
NoiseRecording5.wav – Noise type 1 (Gaussian Noise), Noise level 3 (92 dB)
NoiseRecording6.wav – Noise type 2 (6 Persons babble), Noise level 3 (92 dB)

QuietRecording1.wav – Noise type 3 (Quiet), Noise Level 1 (Quiet)

The recordings made for each subject and each microphone were all collected and renamed. The naming format is as follows:

SpeakerNo_MicrophoneNo_NoiseTypeNo_NoiseLevelNo.wav

3. On Laptop 1 (Dell Latitude), create a directory corresponding to each of the ASR engine. Copy the Matlab scripts and the engine software given in the attached CD to their respective directories. Make a subdirectory by name ‘./input’ and copy all the renamed sound files to that directory. Copy the desired microphone and speaker files alone to the current directory each time before running. The Matlab script laptop1.m can be used to play the sound files one by one. Before running the script, always open it and make sure that the correct microphone number and speaker number is specified in it.

4. On Laptop 2 (Sony VAIO), create a directory for each of the ASR engine. Copy the Matlab scripts and the engine software given in the attached CD to their respective directories. Make a subdirectory, by name ‘./results’, for storing the file containing the recognition results. The Matlab script playengine.m is used to play the engine software. Once this script finishes playing the results of the recognition are written to output files following the naming format,

SpeakerNo_MicrophoneNo_NoiseTypeNo_NoiseLevelNo.txt

5. Start the audio tuning software of Microsoft command and control ASR engine. The same software was used for performing the audio tuning of Vocon and IBM. Play the Audiotuning sound file for the specific speaker and microphone on Laptop 1. Observe the

recording gain adjustment on Laptop 2, and check whether the ASR gets the full dynamic range. If not, the microphone input of Laptop 2 may have to be used instead of the line input. Once the recording adjustment is done, go to Control Panel->Sounds and Audio Devices->Recording volume and check whether it is set to the same value as that shown by the Speech window.

6. Run the playengine.m on Laptop 2, and immediately after that run the laptop1.m on Laptop 1. At a time, 7 files are processed, so the overall running time is estimated to be around 20 minutes. Once the scripts finish running, the output files are formed in the current directory of Laptop 2, which can be copied to the ‘./results’ directory, before starting the next series of tests.

7. Finally, each of the output files is analyzed and the statistics recorded manually and plots generated.

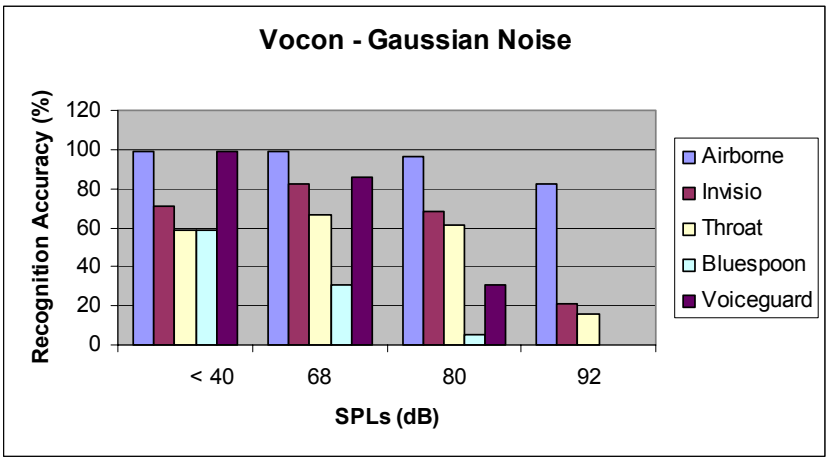
8. While performing tests for Microsoft command and control ASR engine, the active speaker profile on Laptop 2 has to be properly set to the currently considered speaker.

8.4. Evaluation Results

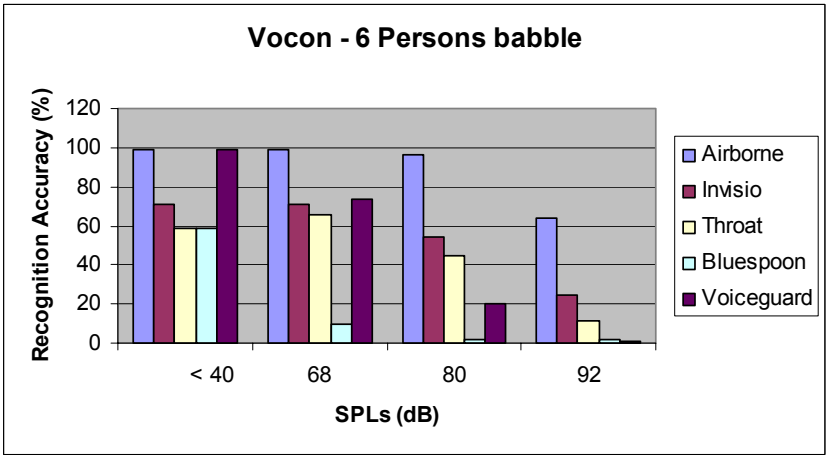
Some of the sample results of the evaluation are presented here. Detailed results are given in Appendix E. The results are categorized into two sections.

8.4.1. Performance of Different Microphones

This section shows the performance of the 5 different microphones averaged over all speakers for Vocon 3200, IBM Viavoice and Microsoft Command and Control under Gaussian Noise and 6 Persons babble. Figure 8.16(a) shows the performance of Vocon 3200 with different microphones averaged over all speakers for Gaussian Noise. Figure 8.16(b) shows the results for 6 Persons babble.



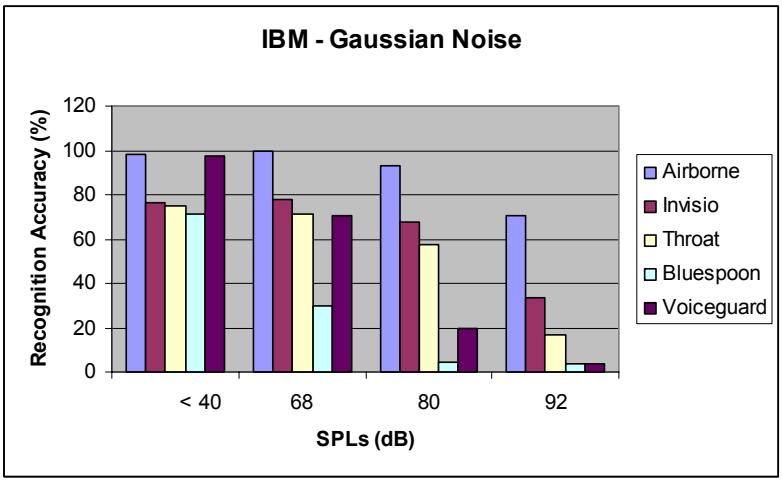
(a) Vocon 3200 – Gaussian Noise



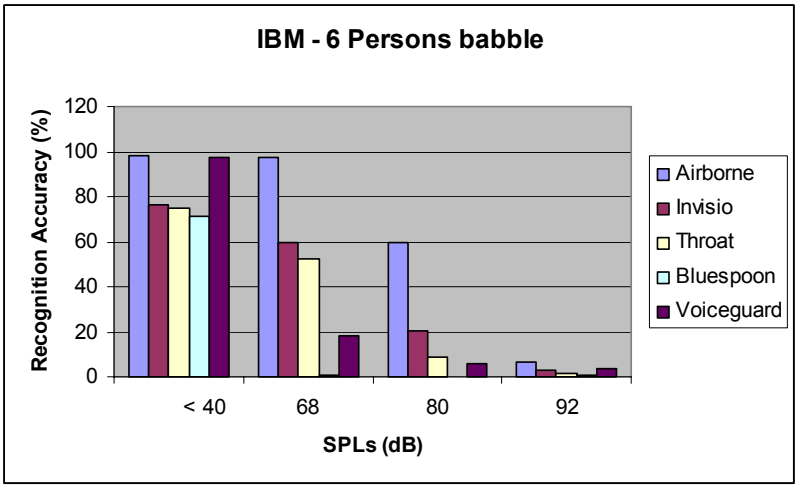
(b) Vocon 3200 – 6 Persons babble

Figure 8.16: Performance of different microphones – Vocon 3200. These results are an average over all speakers)

Figure 8.17(a) shows the performance of IBM with different microphones averaged over all speakers for Gaussian Noise. Figure 8.17(b) shows the results for 6 Persons babble.

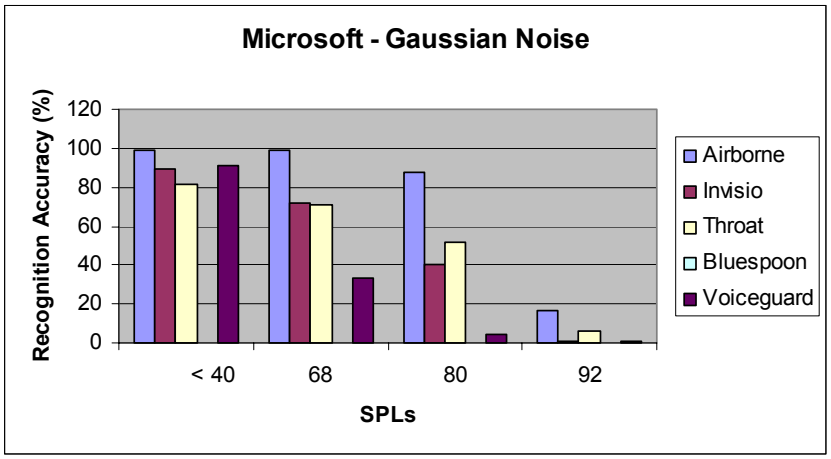


(a) IBM Viavoice- Gaussian Noise

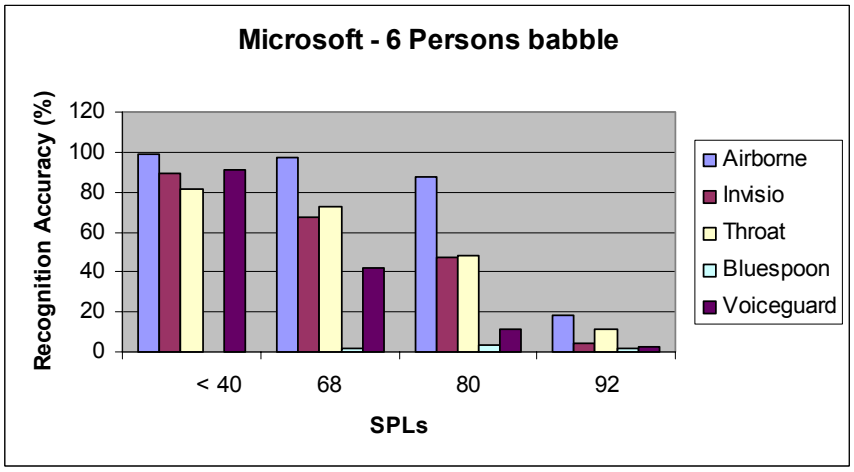


(b) IBM Viavoice - 6 Person babble

Figure 8.17: Performance of different microphones – IBM Viavoice. These results are an average over all speakers



(a) Microsoft Command and Control – Gaussian Noise



(b) Microsoft Command and Control – 6 Persons Babble

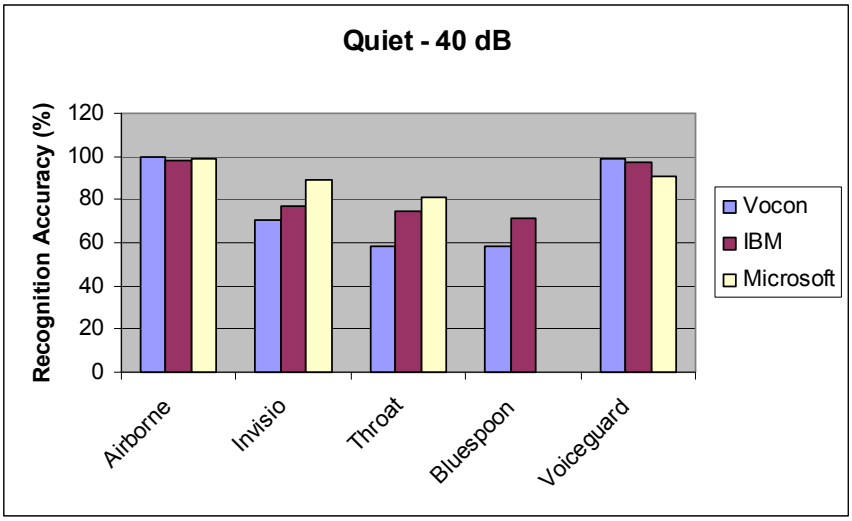
Figure 8.18: Performance of different microphones – Microsoft Command and Control. These results are an average over all the speakers

Figure 8.18(a) shows the performance of Microsoft Command and Control with different microphones in Gaussian noise and Figure 8.18(b) shows the results in 6 Persons babble. The results indicate that all microphones fair better in Gaussian noise than in 6 Persons babble for all ASR engines. The airborne microphone gave better recognitions under all the SPL levels. Invisio and physiological sensor microphones did not deteriorate much with increasing SPLs, as

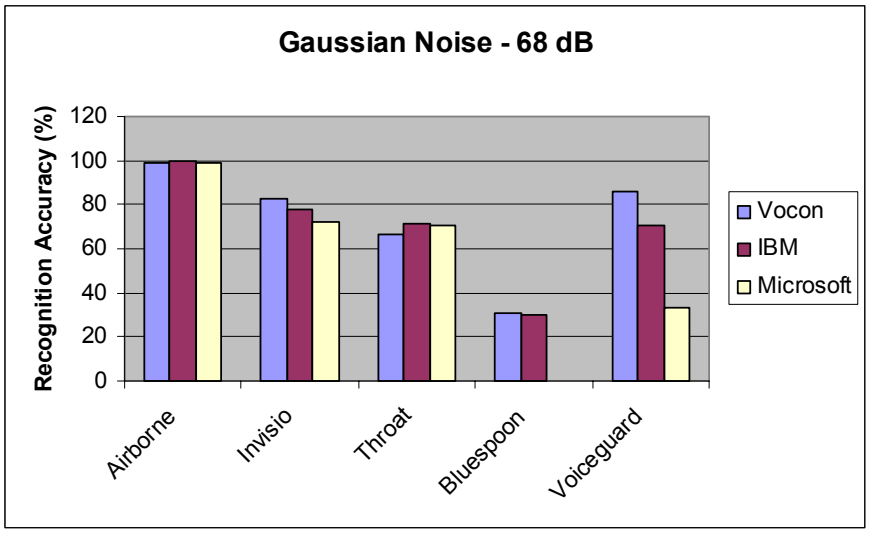
compared to Bluespoon or Voiceguard. But the recognition rates of Invisio and Physiological sensor were around 60-75% even under normal conditions. Voiceguard gave high recognition rates comparable to airborne microphone under quiet conditions, but its performance fell to 30% at 80 dB and very low at 92 dB. Bluespoon performance was poor overall, perhaps due to other reasons which are discussed in Section 8.4.4.

8.4.2. Comparison between ASR Engines and Microphones at Different SPLs

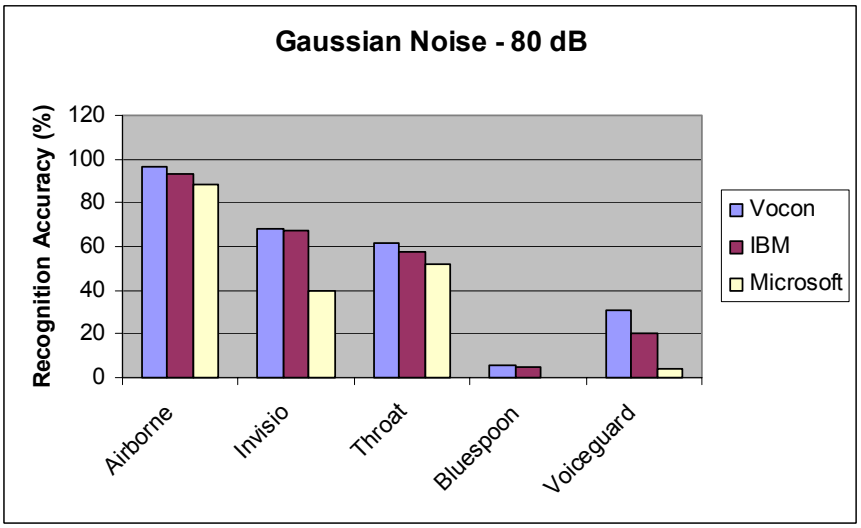
The same results presented in Section 8.4.1 are discussed in a different perspective, to get a better comparison between ASR engine and its performance at different SPLs. Figure 8.19Figure 8.19(a) gives the performance of ASRs and microphones for quiet conditions. Figure 8.19(b) shows the results in Gaussian Noise at 68 dB and Figure 8.19(c) shows the results at 80 dB.



(a) ASRs and microphones under quiet conditions



(b) ASRs and microphones in Gaussian Noise at 68 dB



(c) ASRs and microphones in Gaussian Noise at 80 dB

Figure 8.19: Comparison of ASRs and microphones at different SPLs. These results are an average over all speakers

Vocon 3200 and IBM Viavoice seem to perform almost similar for both Airborne and Voiceguard microphones under quiet conditions. But, at higher SPLs, IBM Viavoice seems to be

slightly more sensitive to noise than Vocon 3200. The performance of Microsoft Command and Control appears to degrade fast at higher SPLs. For Invisio and Physiological sensor microphone, IBM Viavoice performed better than Vocon 3200 during quiet conditions. However, Microsoft Command and Control gave the best performance with Invisio and Physiological Sensor, perhaps due to its speaker-dependent training. It can also be noticed that Voiceguard gives better performance with Vocon at higher SPLs.

8.4.3. Variability of ASR Performance with Different Speakers

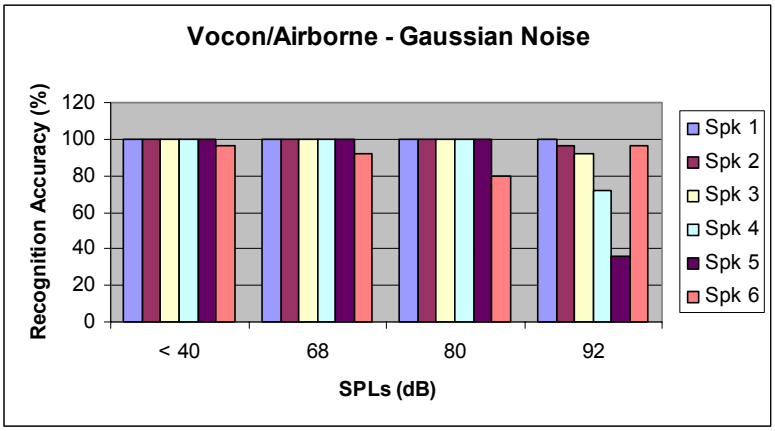
This section analyzes the recognition variability of the different ASR engines with speakers, for some specific microphones in Gaussian noise. Results for all the different microphones and ASR engines are given in the Appendix E. The details of the different speakers are:

Speaker #1 – Non-American male, Speaker #2 – American male

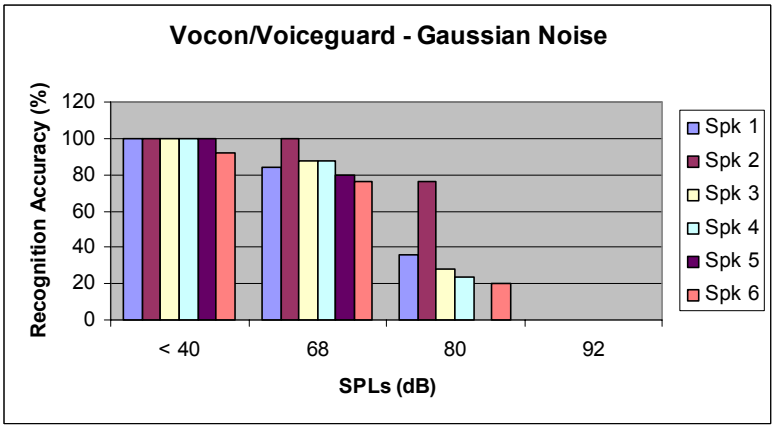
Speaker #3 – American female, Speaker #4 – American male

Speaker #5 – American male, Speaker #6 – Non-American female

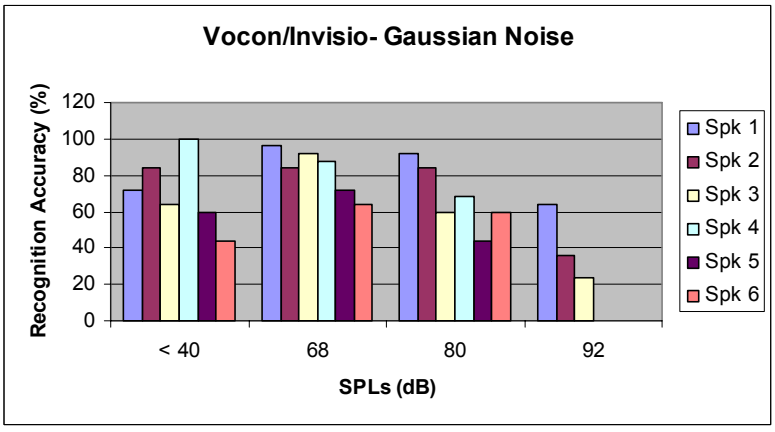
Figure 8.20 gives the results for Vocon 3200 with an airborne microphone, Voiceguard and Invisio. Figure 8.21 gives the results for IBM Viavoice with airborne microphone, Voiceguard and Physiological Sensor microphone.



(a) Vocon/Airborne – Gaussian Noise

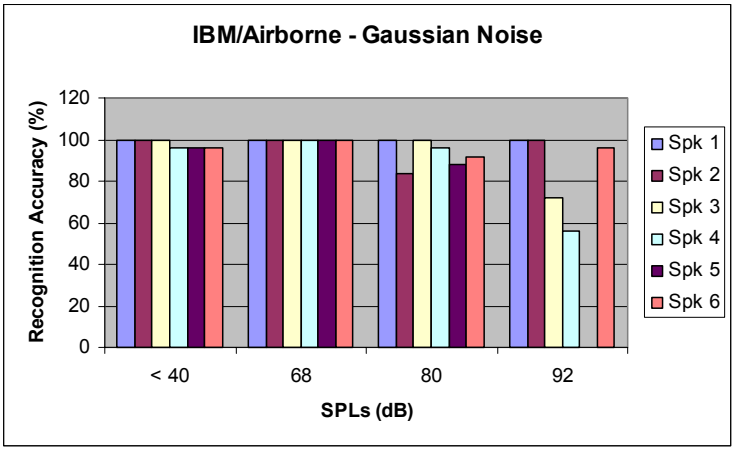


(b) Vocon/Voiceguard – Gaussian Noise

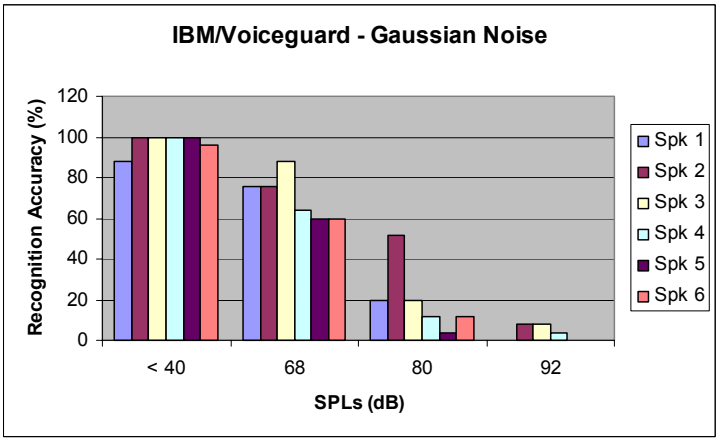


(c) Vocon/Invisio – Gaussian Noise

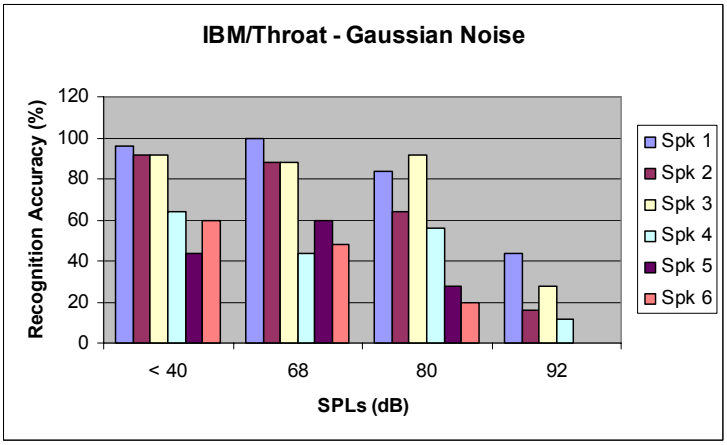
Figure 8.20: Performance of different speakers with Vocon



(a) IBM/Airborne - Gaussian Noise



(b) IBM/Voiceguard - Gaussian Noise



(c) IBM/Physiological Sensor - Gaussian Noise

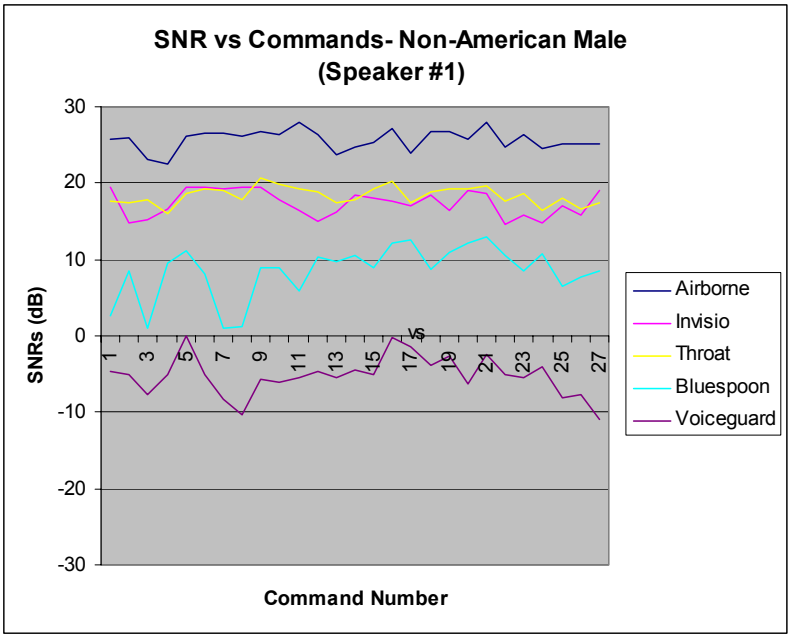
Figure 8.21: Performance of different speakers with IBM

It can be observed that the airborne microphone gave 100% recognition rates for all the five speakers up to 80 dB for Vocon 3200. The recognition rate was poorer for IBM Viavoice at 80 dB. At 92 dB, the performance of airborne seems to be dependent on the speakers. Voiceguard performance was at an average of 100% under quiet conditions, but fell to 80% under 68 dB for Vocon 3200 and even lesser for IBM Viavoice. Voiceguard seems to become dependent on the speakers at 80 dB and at 92 dB. The performance of Invisio was dependent on the speakers even under quiet conditions. But, the performance seemed to stay consistent even at 80 dB. It can be noticed that for some speakers, Invisio gave 80-100% recognition rate even at 80dB SPL. IBM Viavoice seems to give very good performance with throat microphone for some speakers even at 80 dB.

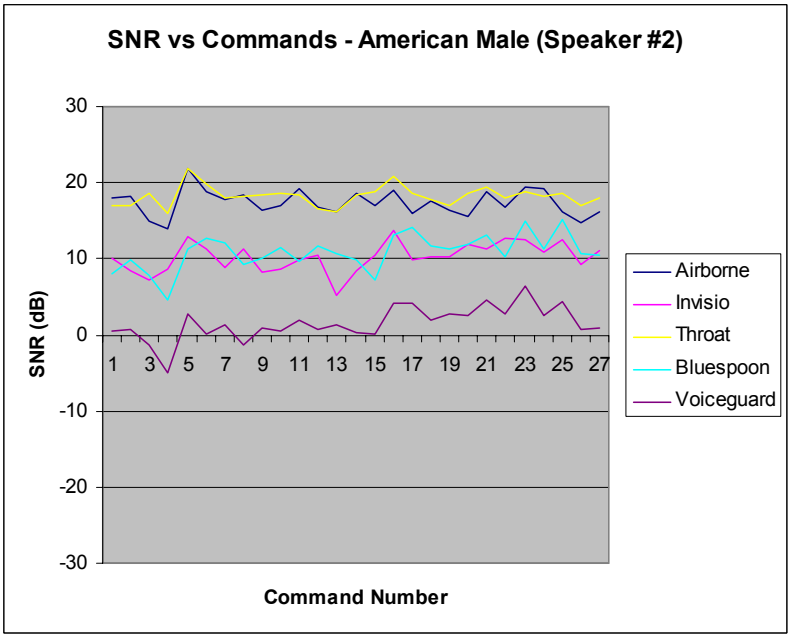
8.4.4. Analysis of the Quality of the Measured Signals with Different Microphones

The SNR of the recordings of the different microphones is determined at a given SPL to make a comparison among the signal quality, achieved with the different microphones. The analysis was done for recordings made from all the 5 microphones at an SPL level of 68 dB for all 6 speakers in Gaussian noise. To calculate the SNR of the commands in a recording, the short-time RMS values are calculated over the entire recording and these values are plotted in Matlab. The noise floor can be estimated by the user by observing the RMS plot of the recording. Once this is performed, the user is prompted again to specify the beginning and end of the first command in the recording on the plot. These points can also be determined by the user from direct observation of the plots. All the subsequent commands are then separated out from the recording by the addition of 5 seconds to the beginning of the previous command. (Each subject spoke the commands with 5 seconds duration in between, with the help of the cuing device). The SNR of

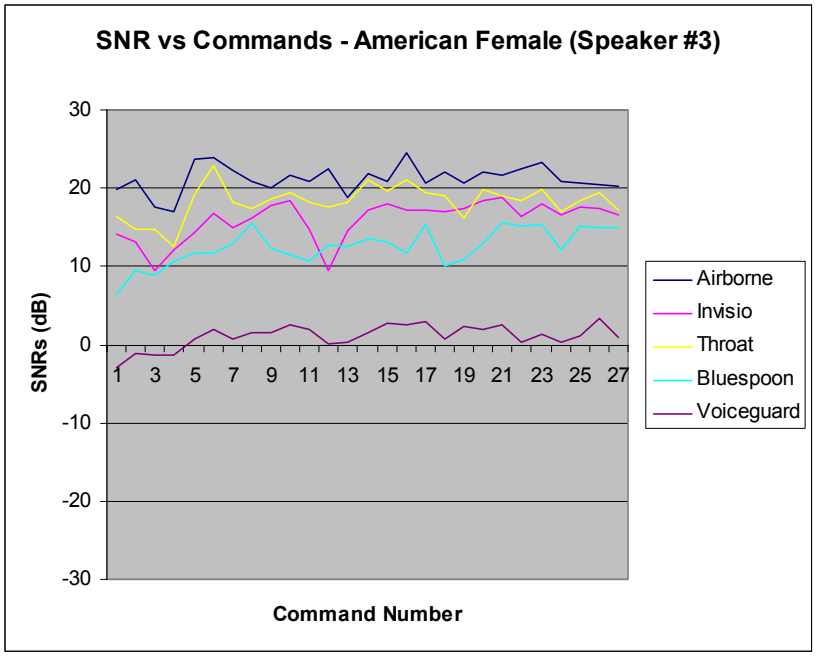
each individual command was then estimated and the results obtained from the different recordings are plotted in Figure 8.22.



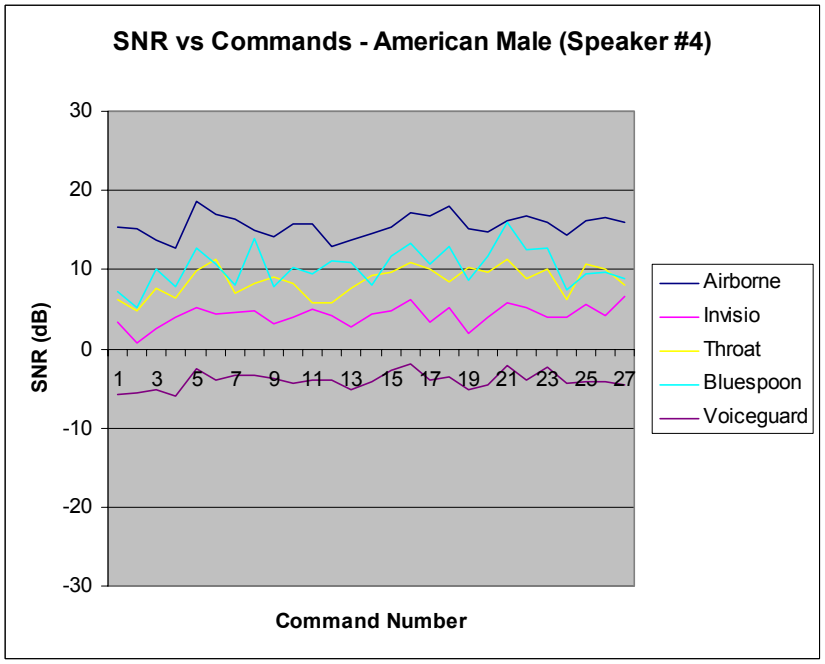
(a) SNR versus Commands – Speaker #1 (Non-American male)



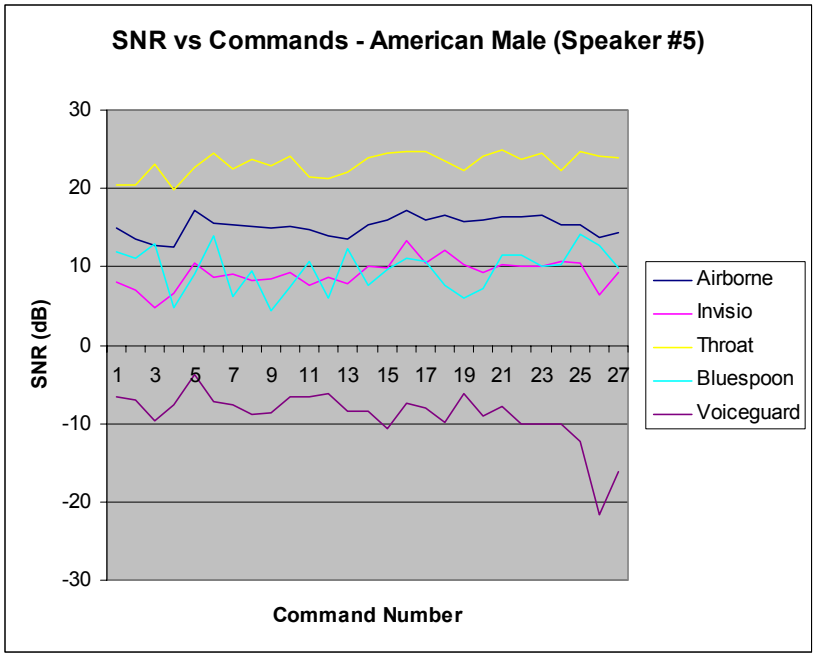
(b) SNR versus Commands – Speaker #2 (American male)



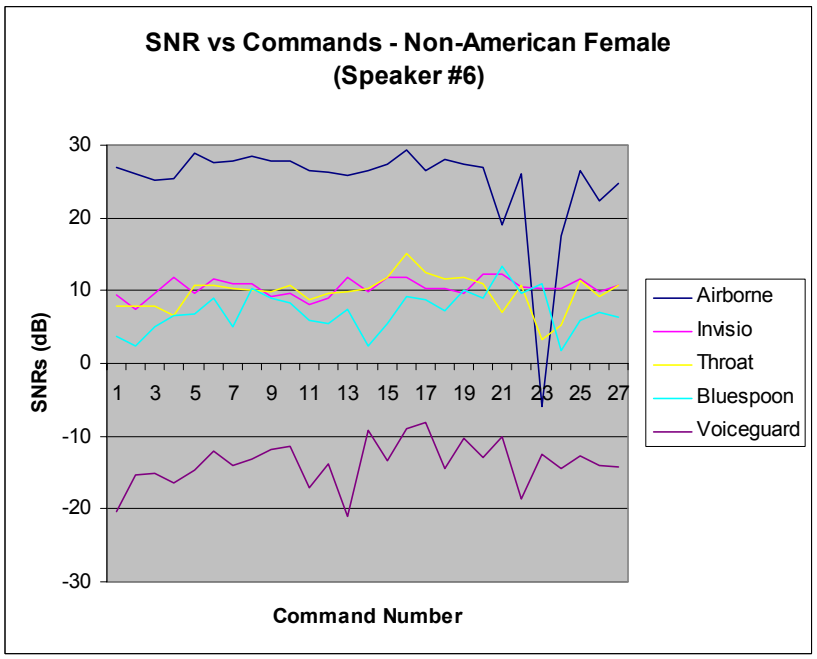
(c) SNR versus Commands – Speaker #3 (American female)



(d) SNR versus Commands – Speaker #4 (American male)



(e) SNR versus Commands – Speaker #5 (American male)



(f) SNR versus Commands – Speaker #6 (Non-American female)

Figure 8.22: SNR of the recordings of the microphones at 68 dB SPL

The graphs indicate that the SNR of the airborne microphone is higher than the other microphones. The Invisio, Physiological Sensor and Bluespoon microphones also give reasonably good SNRs. The SNR of Voiceguard microphone is noticeably poor. A more detailed analysis of these results can be found in Chapter 9.

8.4.5. Conclusions on Microphones

This section presents the conclusions made for each microphone based on the results illustrated in the Sections 8.4.1 – 8.4.4.

Airborne microphone, Emkay Model 3345

Emkay has performed quite well even under high noise conditions. It was observed to be dependent on the speakers at 92 dB. The only disadvantage with the airborne microphone is its unsuitability of use in a field environment.

Invisio and Physiological Sensor Microphone

The results tend to give almost identical observations regarding these two microphones. Their performance was found to be consistent until 80 dB. Speaker variability was also observed even in quiet conditions. The reason for this kind of performance could be because the method of speech capture for these types of microphones differs from the airborne microphone. A recorded sound file from Invisio when played, sound very different as compared to one recorded using airborne. This necessitates the need for tuning speech recognition engines specifically for these types of microphones. A possible alternative would be using the Hidden Markov model Toolkit (HTK) [21] for implementing ASR features, which enables to fine-tune the speech recognition

parameters. This could possibly increase the recognition rates and lessen the speaker variability. Another alternative is to use the user-word training features in Vocon 3200 to get customized pronunciation dictionaries for these microphones. This can be made for each speaker as well. At high SPLs such as 92 dB, recognition performance can be improved by using the *push-to-talk* features of Vocon 3200 and IBM Viavoice. As the ASR engines were operated in an always-on manner for a given recording, it was getting constantly trained from the environment from one command to the next and even while the user was silent. This must have resulted in loss in performance. Under *push-to-talk*, the ASR engine gets activated only when an user presses a button for talking and it gets deactivated when he/she releases the button. So the training of the ASR occurs only when the user speaks and not when he/she is silent. This mode of operation has given a better performance than always-on mode at high SPLs.

Bluespoon

Bluespoon gave a poor performance overall. One reason for this could be that, to get an acceptable sound quality from Bluespoon, the sampling rate had to be kept at 8 KHz 16 bit mono PCM. But in general, ASR engines require a higher sampling rate. A solution is to get a Bluetooth microphone compatible with the ASR sampling rates. Bluetooth later standards are known to support it, and therefore it may be introduced commercially to the market in the near future. Another disadvantage of Bluespoon for speech recognition purposes is that it does not support digital gain adjustment.

Voiceguard Microphone

Voiceguard gives excellent results under quiet conditions. One biggest advantage it possesses is its ease of use. Its performance is found to deteriorate at 80 dB. One solution for this is the use of

push-to-talk feature in the ASR engine. It has been found that this mode of ASR engine gives acceptable performance even for SPLs around 80 dB. Additionally, Voiceguard microphone is designed for better performance in directional noise than in diffuse noise. Therefore, the performance may be slightly better than indicated in the graphs for Voiceguard, in the presence of directional noise.

8.4.6. Conclusion

Vocon 3200 with Airborne microphone has given the best performance both under quiet and high noise conditions in all the results presented here. But if the convenience of use is a major factor in the selection of a microphone, then Vocon 3200 with Voiceguard microphone could be a possible option. But its usage may be acceptable only till 68 dB, but can be improved at 80 dB using *push-to-talk* features of the ASR engine.

Chapter 9. Conclusions and Future Work

9.1. Conclusions

This thesis has described the redesign of an existing portable ultrasound scanner, the Terason 2000, into a wearable ultrasound scanner system suitable for field-use purposes. The ultrasound scanner configuration, which was produced as a result of this research, is targeted to make a physician's job in an emergency situation both fast and efficient. Most controls of the wearable ultrasound scanner system are operated through voice commands. The mini-joystick attached to the top of the transducer serves as a useful interface to make distance and area measurements. The ultrasound images can be viewed through an eye-wear viewer, which also serves as an effective addition over the existing ultrasound systems. Power management features incorporated into the ultrasound system help to conserve the battery power. A wireless connection, based on IEEE 802.11b, is established with a remote laptop to enable real-time transmission of ultrasound images. These additional extensions to the existing ultrasound system are expected to enhance its usage in emergency situations. Clinical trials are planned with the new ultrasound scanner system in order to get valuable feedback from physicians.

The implementation details of the speech interface used to control the ultrasound scanner system by voice commands were also described. The menu items on the Terason scanner window are invoked by sending the appropriate sequence of keystrokes from the *voice command handler* software, which is external to the Terason scanner software. A user might wish to speak a command, which is different from the corresponding menu item text on the Terason scanner window. This can be facilitated by altering the context-free grammar file of the *voice command*

handler software. The incorporation of new voice commands will require only minimal implementation effort, due to the object-oriented design of the *voice command handler* software. Additionally, as the *voice command handler* software consists of two functionally separate modules, viz. v*oice command recognizer* and *voice command dispatcher*, the migration to a new ASR engine only means that appropriate changes are made to the *voice command recognizer* module. This is because it is only this module which interacts directly with the ASR engine through its respective APIs.

Power management issues were also briefly discussed in this thesis. The different low power states of operation of the ultrasound scanner system were identified and measurements were made to determine the approximate power consumed in these states. The implementation details involved in bringing a system to a low power state and for its subsequent wake-up process were also examined. In this discussion, the low power states were invoked using voice commands. This method of invoking the power state is not actually preferred, as it is highly probable that a false recognition of a voice command will occur at some point in time, which may bring the system to a low power state at an undesired moment. Therefore, more reliable ways of invoking the low power states of the system have to be investigated.

This thesis also discussed the extensive tests carried out in order to select the ASR engine which is most tolerant to realistic levels of ambient noise. The reliability of the ASR engines under different types of noise and speakers was also investigated. Vocon 3200, IBM Viavoice and Microsoft Command and Control served as candidates for the testing. The evaluation was done by collecting recordings from 6 different speakers in a quiet chamber using an airborne microphone. Noise was scaled and added electronically to these recordings to create 7 different

Signal-to-Noise ratios. 4 different types of noise were also considered for this testing. The resulting noisy recordings were fed to the ASR engines and the statistics were collected and analyzed before reaching a conclusion. The results indicate that Vocon 3200 performs better than the other ASR engines under moderately high ambient noise conditions. Its performance under different noise types remained consistent. It showed similar performances with almost all of the speakers. It can also be noted from the results that, the performance of Vocon 3200 remained above 95% for SNRs of 25 dB or higher, for most speakers.

Tests were also performed to investigate the range of audio frequencies relevant for speech recognition. Low-pass filters with cutoff frequencies in the range 1000-4000Hz were applied to the sound files containing noise and speech, before they were fed to the ASR engine, to study the impact of removing the high audio frequencies on recognition rate. The graphs show that when frequencies above 3000 Hz or 4000 Hz are filtered out, there is an appreciable fall in the recognition rate at low SNRs. When frequencies above 1000Hz or 2000Hz are filtered out, the fall in performance was evident at both high and low SNRs. The results were consistent for both male and female American speakers. A similar high-pass filter test revealed that the recognition rate stayed high only when frequencies above 300 Hz were present in the recording. These tests give useful information regarding the frequency response of a microphone suited for speech recognition purposes. On the basis of these experiments, it can be concluded that the minimum bandwidth required for a microphone suited for speech recognition is between 300 Hz and 5000 Hz. These results also indicate that, sampling rates of 11 kHz or higher are essential for obtaining good recognition results. Furthermore, noise below 300 Hz can be filtered out before speech recognition is actually performed, without affecting the recognition rate.

The evaluation of the ASR engines with 5 different types of microphones was conducted in a real reverberant environment with noise at a prescribed level introduced. The different microphone candidates include a physiological sensor microphone, a jaw-bone conduction microphone (Invisio), a 4-element array microphone (Voiceguard), a 2-element array microphone that works using Bluetooth (Bluespoon) and a conventional airborne microphone (Emkay). The recordings were made using 6 different speakers in 2 different noise types at 3 different SPLs, and they were fed subsequently to the ASR engines to estimate performance. The results show that Vocon 3200 together with an Emkay microphone gives the best performance under SPL levels of 80 dB or higher. At low noise levels, an alternative setup would be to use Vocon 3200 with the Voiceguard microphone (4-element array microphone). The tests conducted in the reverberant chamber have tested the microphones in a diffuse noise environment. As the Voiceguard microphone is designed to perform better in directional noise than in diffuse noise, its performance in directional noise will be better than the results presented in this thesis. The performance of the ASR engine itself can be improved with the help of the *push-to-talk* feature. With the incorporation of this feature, Vocon 3200 has shown to give good performance with Voiceguard even at high SPLs. Invisio and Physiological Sensor microphones showed potential for speech recognition, but microphone-specific ASR training is necessary before they can be reliably used. Bluespoon gave poor recognition rates, possibly due to its poor frequency response and low sampling rates.

The determination of SNRs of the recordings of all the different microphones at an SPL of 68 dB has shown that the quality of the signal, as measured by the SNR of the signal from the microphone, varies a great deal from microphone to microphone. The Voiceguard microphone captures speech signal at an SNR of around 20 dB lower than the speech signal captured by the

airborne microphone in the same environment. The adaptive array processing techniques of the Voiceguard microphone, when applied to the original signal captured from all its 4 microphone elements, is expected to give an improvement of around 10 dB in diffuse noise. Therefore, the actual difference in SNRs between the signals captured by the airborne microphone and the microphone elements in Voiceguard is around 30 dB at 68 dB SPL. This seems to explain why the performance of Voiceguard degrades faster in more noisy environments. Invisio and Physiological sensor microphones have shown to give SNRs comparable to that of the airborne microphone even at 68 dB SPLs. Although, Bluespoon has given poor recognition performance with most ASR engines, the speech signal captured by Bluespoon has a reasonably good SNR.

Thus, two different methods of evaluation were performed to identify the ideal combination of ASR engine and microphone that can give good performances in moderately noisy environments. In the first evaluation, noise was added electronically to speech and in the second, noise was played together with speech in an acoustic chamber. On the basis of the SNR of the recording of a specific speaker with an airborne microphone at an SPL of 68 dB, a comparative study was done between the recognition performances obtained in the two evaluation tests for this speaker at this SNR. The analysis shows that the recordings made in the acoustic chamber give a better performance for most ASR engines at the same SNR than the quiet room recordings with electronic noise added. One argument to support this variation in performance could be that, in one evaluation, noise is added electronically to speech, while in the other, noise signal is convolved with speech signal in the acoustic chamber. The method by which the ASR engine handles the resultant waveforms in each of the above two cases may possibly be different.

9.2. Future Work

One of the important improvements that can be made to the wearable ultrasound scanner system described in this thesis is replacing the laptop with an embedded PC. This work has recently been completed, and the resultant ultrasound scanner can now be incorporated into a photographer's vest with the whole system weighing less than 5 lbs. Figure 9.1 shows a picture of a person wearing the vest, with the scanner incorporated into it.

Figure 9.1: Second generation ultrasound scanner system

The new ultrasound scanner system supports a wide variety of scanner controls through voice commands. However, further enhancement is achieved by implementing more voice commands. The ability to enter patient information through voice commands is an example.

It would be useful to evaluate the reliability of Voiceguard microphone compared to a conventional airborne microphone in directional noise. This could be done by carrying out

testing in a reverberant chamber equipped with directional noise or in a more realistic environment like in an ambulance.

Fine-tuning speech recognition parameters for effective performance under noisy conditions is indeed another area which can benefit from further work. One possibility is to use the push-to-talk feature of the ASR engine, so that the recognition is active only when a user presses a button to talk and it gets deactivated when the user releases the button. This feature has been implemented in the scanner system shown in Figure 9.1. The mini-joystick epoxied on top of the transducer, also provides left and right buttons. The right button fitted towards the right of the transducer, is ergonomically designed for use together with the push-to-talk feature of the ASR engine. The user can easily press the button to invoke a voice command, while the ultrasound scanning is being performed.

The power management feature is also an area for further improvement. The mini-joystick, fitted on top of the transducer, could serve as a device for waking the system from a low power state. However, the drivers for this device currently do not support this feature. Obtaining the right drivers which can support this feature could be a solution. It is also possible that the USB interface version that the mini-joystick used, version 1.0, does not support this feature and a higher version like USB 2.0 could solve this problem.

Currently, the system is configured in such a way that it always remains in active mode, unless the user explicitly puts it into a sleep state. One possibility for improvement here is to put the ultrasound system into a sleep state when the transducer is idle. In other words, when the user is not actively performing the scanning, the system should detect it and automatically go into a

sleep state. This may bring in the need for some form of image analysis. It may be possible that the Terason scanner software itself may have the ability to determine the information of whether the transducer is idle or not. This information can be acquired by the *voice command handler* software for processing, if a suitable software interface is provided by Terason.

References

[1] Major Suzanne Le P Langlois, "Portable ultrasound on deployment", *ADF Health*, Vol 4, pp. 77-78, Sep 2003.

[2] Chiang, A.M., Chang, P.P., Broadstone, S.R., "PC-based ultrasound imaging system in a probe", *IEEE Ultrasonics Symposium Proceedings*, Vol 2, pp. 22-25, Oct 2000.

[3] Peacocke R.D, Graf D.H, "An Introduction to Speech and Speaker Recognition", *IEEE Computer*, Vol. 23, Issue: 8, pp. 26-33, Aug 1990.

[4] Chiang A.M., Chang P.P., Broadstone S.R., "PC-Based Ultrasound Imaging System in a Probe", *IEEE Ultrasonics Symposium*, Vol 2, pp. 22-25, Oct 2000.

[5] Srinivasan S., Brown E., "Is Speech Recognition Becoming Mainstream? ", *IEEE Computer*, Vol. 35, Issue 4, pp. 38-41, March 2002.

[6] "How Speech Recognition Works", *http://project.uet.itgo.com/speech.htm*

[7] Ouellette J., "Speech Recognition: Humanizing the Interface", *The Industrial Physicist*, pp. 20-23, March 1998.

[8] Padmanabhan M., Picheny M., "Large Vocabulary Speech Recognition Algorithms", *IEEE Computer*, Vol. 35, Issue 4, pp. 42-50, March 2002.

[9] COM, *http://www.microsoft.com/com/tech/com.asp*

[10] Microsoft Speech SDK 5.1 Help manual, http://www.microsoft.com/speech/techinfo/apioverview/

[11] Microsoft Speech SDK 4.0 C++ Development guide, http://www.microsoft.com/speech/download/old/sdk40a.asp

[12] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns, Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

[13] "Microsoft Developer Network - MSDN Library Network Help files", *http://msdn.microsoft.com/*

[14] "Rational Rose tool for Object Oriented Design Help file", *http://www-306.ibm.com/software/rational/*

[15] "OnNow Power Management Architecture for Applications", Microsoft Developer Network (MSDN) documentation, *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/power/base/onnow_power_management.asp*

[16] Microsoft Platform SDK documentation – Core SDK, *http://www.microsoft.com/msdownload/platformsdk/sdkupdate/*

[17] Terason 2000 Ultrasound COM Automation Interface, *http://www.terason.com*

[18] W.A. Dreschler, H. Verschuure, C. Ludwigsen and S. Westermann, "ICRA Noises: Artificial Noise Signals with Speech-like Spectral and temporal properties for Hearing Instrument Assessment," *Audiology* 2001*;* 40:148-157.

[19] Vocon 3200 Version 1.2 Documentation, *http://www.scansoft.com/automotive/vocon3200/*

[20] J.D. Bass, M.V, Scanlon, T.K. Millis, J.J. Morgan, "Getting Two Birds with One Phone: An acoustic sensor for both speech recognition and medical monitoring", Presented in poster format at the 138th Meeting of the Acoustical Society of America, Nov 1999.

[21] Hidden Markov Model toolkit, *http://htk.eng.cam.ac.uk/*

Appendices

Appendix A.

The context free grammar file described here can be used with Microsoft SAPI 5.1 for the recognition of the menu items on Terason window using voice commands. The grammar is divided into several rules and each rule can be made active and deactive by the software program with the help of APIs provided by Microsoft Command and Control.

```
<GRAMMAR LANGID="409">

  <RULE NAME="root" TOPLEVEL="ACTIVE">
    <L>
      <P>file</P>
      <P>edit</P>
      <P>view</P>
      <P>exam</P>
      <P>image</P>
      <P>insert</P>
      <P>image</P>
      <P>modes</P>
      <P>tools</P>
      <P>help</P>
      <P>small</P>
      <P>medium</P>
      <P>large</P>
      <P>focus</P>
      <P>depth</P>
      <P>gain</P>
    </L>
  </RULE>

  <RULE NAME="file_rule" TOPLEVEL="INACTIVE">
    <L>
      <P>open</P>
      <P>export image as</P>
      <P>export loop as</P>
      <P>export measurements as</P>
      <P>exit</P>
    </L>
   </RULE>
```

```
<RULE NAME="cancel_rule" TOPLEVEL="INACTIVE">
  <L>
    <P>cancel</P>
  </L>
</RULE>

<RULE NAME="view_rule" TOPLEVEL="INACTIVE">
  <L>
    <P>toolbars</P>
    <P>status bar</P>
    <P>image control bar</P>
    <P>terason explorer</P>
    <P>patient info</P>
    <P>image display</P>
    <P>depth ruler</P>
    <P>frame rate</P>
    <P>reference bar</P>
    <P>probe info</P>
    <P>measurement value</P>
    <P>orientation logo</P>
    <P>patient information</P>
    <P>zoom thumb nail</P>
    <P>zoom</P>
    <P>refresh</P>
    <P>t g c display</P>
    <P>full screen</P>
  </L>
</RULE>

<RULE NAME="toolbars_rule" TOPLEVEL="INACTIVE">
  <L>
    <P>file</P>
    <P>freeze</P>
    <P>measurement</P>
    <P>play back</P>
    <P>scan mode</P>
  </L>
</RULE>

<RULE NAME="tgcdisplay_rule" TOPLEVEL="INACTIVE">
  <L>
    <P>on</P>
    <P>off</P>
    <P>timeout</P>
  </L>
</RULE>
```

```
<RULE NAME="exam_rule" TOPLEVEL="INACTIVE">
   <L>
      <P>open exam</P>
      <P>save exam</P>
      <P>save exam as</P>
      <P>delete exam</P>
      <P>abdominal</P>
      <P>cardiac</P>
      <P>obstetrical</P>
      <P>pelvic</P>
      <P>prostate</P>
   </L>
</RULE>

<RULE NAME="close_rule" TOPLEVEL="INACTIVE">
   <L>
      <P>close</P>
   </L>
</RULE>

<RULE NAME="image_rule" TOPLEVEL="INACTIVE">
   <L>
      <P>freeze</P>
      <P>size</P>
      <P>depth</P>
      <P>focus</P>
      <P>live</P>
      <P>invert</P>
      <P>palette</P>
      <P>smoothing</P>
      <P>persistence</P>
      <P>map</P>
   </L>
</RULE>

<RULE NAME="invert_rule" TOPLEVEL="INACTIVE">
   <L>
      <P>left right</P>
      <P>up down</P>
   </L>
</RULE>

<RULE NAME="palette_rule" TOPLEVEL="INACTIVE">
   <L>
      <P>gray</P>
      <P>tan</P>
      <P>flame</P>
```

```
      <P>sepia</P>
      <P>magenta</P>
      <P>sage</P>
      <P>rainbow</P>
      <P>cobalt</P>
   </L>
</RULE>

<RULE NAME="smoothing_rule" TOPLEVEL="INACTIVE">
   <L>
      <P>a</P>
      <P>b</P>
      <P>c</P>
      <P>d</P>
      <P>e</P>
   </L>
</RULE>

<RULE NAME="persistence_rule" TOPLEVEL="INACTIVE">
   <L>
      <P>zero</P>
      <P>one</P>
      <P>two</P>
      <P>three</P>
      <P>four</P>
   </L>
</RULE>

<RULE NAME="map_rule" TOPLEVEL="INACTIVE">
   <L>
      <P>a</P>
      <P>b</P>
      <P>c</P>
      <P>d</P>
      <P>e</P>
      <P>f</P>
   </L>
</RULE>

<RULE NAME="insert_rule" TOPLEVEL="INACTIVE">
   <L>
      <P>text</P>
      <P>measurement label</P>
   </L>
</RULE>

<RULE NAME="modes_rule" TOPLEVEL="INACTIVE">
```

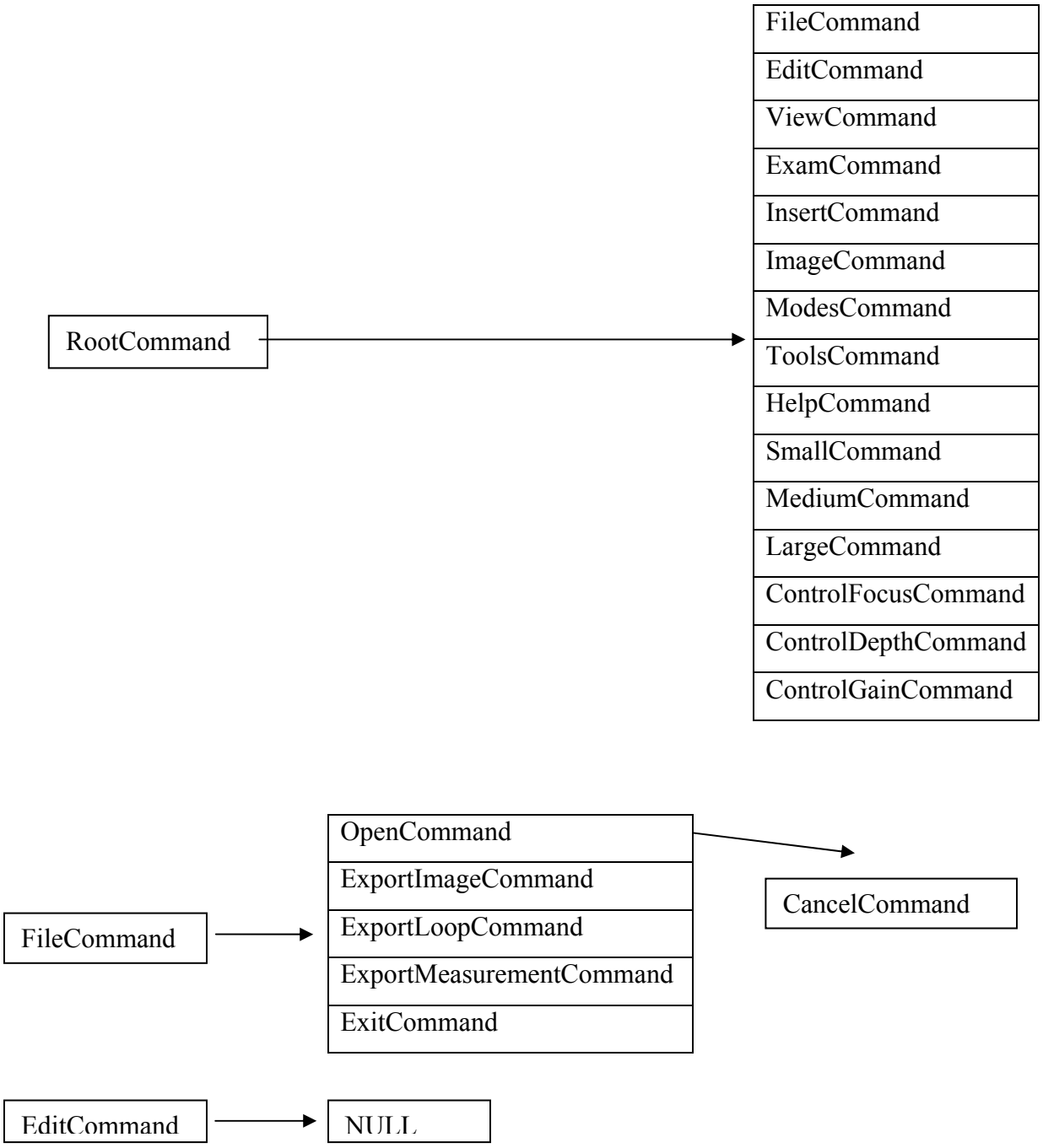
```
   <L>
      <P>b mode</P>
      <P>m mode</P>
      <P>directional power doppler</P>
      <P>power doppler</P>
   </L>
</RULE>

<RULE NAME="tools_rule" TOPLEVEL="INACTIVE">
   <L>
      <P>gamma correction</P>
      <P>probe verification test</P>
      <P>system logging</P>
      <P>always on top</P>
      <P>auto freeze</P>
      <P>set auto freeze wait time</P>
      <P>show color doppler saving warning</P>
   </L>
</RULE>

<RULE NAME="help_rule" TOPLEVEL="INACTIVE">
   <L>
      <P>help topics</P>
      <P>technical support</P>
      <P>license</P>
      <P>about terason</P>
   </L>
</RULE>

<RULE NAME="depth_rule" TOPLEVEL="INACTIVE">
   <L>
      <P>seven</P>
      <P>eight</P>
      <P>nine</P>
      <P>ten</P>
      <P>eleven</P>
      <P>twelve</P>
      <P>thirteen</P>
      <P>fourteen</P>
      <P>fifteen</P>
      <P>sixteen</P>
      <P>seventeen</P>
      <P>eighteen</P>
      <P>nineteen</P>
      <P>twenty</P>
      <P>twenty one</P>
      <P>twenty two</P>
```

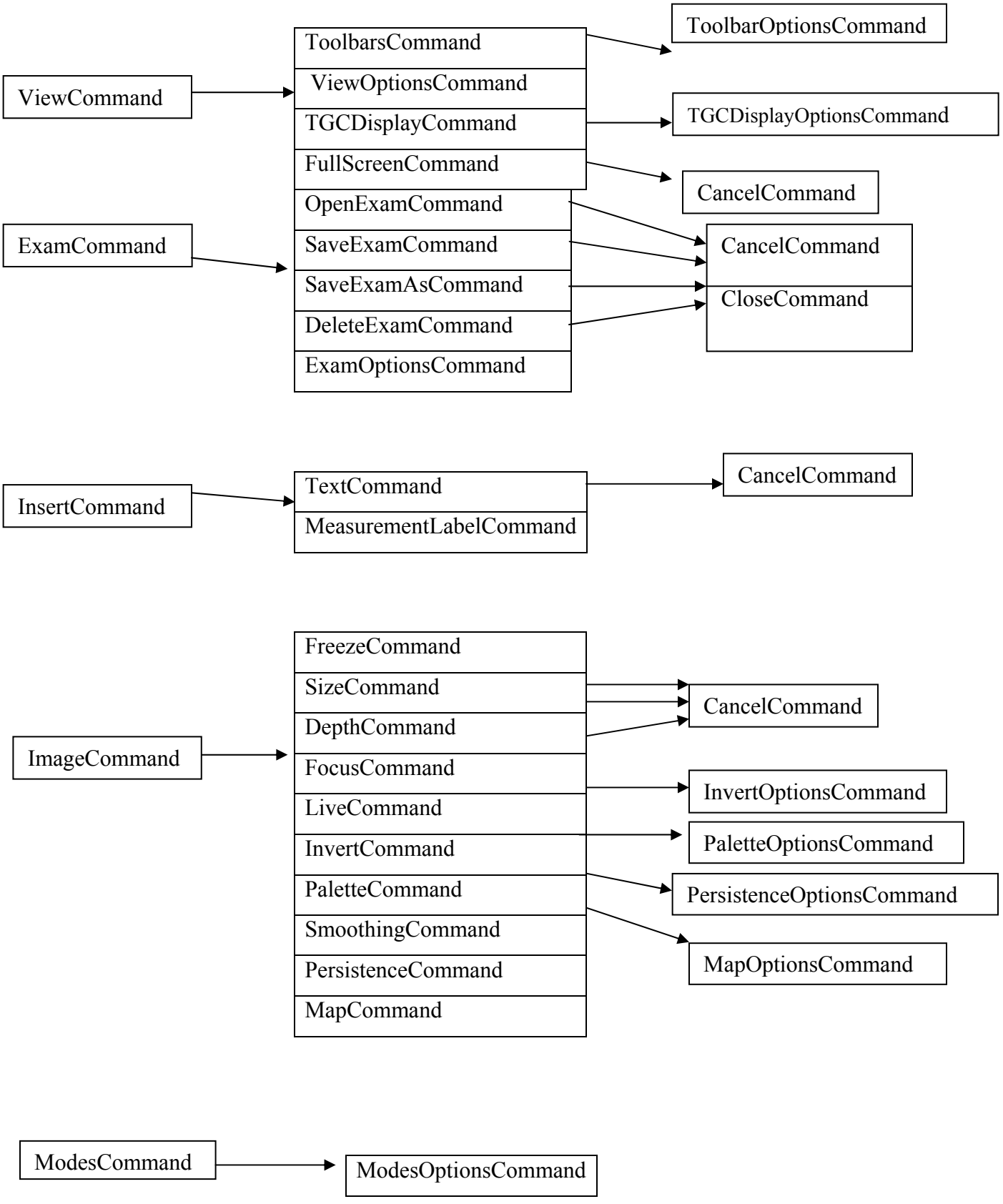
```
        <P>twenty three</P>
        <P>twenty four</P>
      </L>
    </RULE>

    <RULE NAME="focus_rule" TOPLEVEL="INACTIVE">
      <L>
        <P>three</P>
        <P>four</P>
        <P>five point five</P>
        <P>seven</P>
        <P>eight point five</P>
        <P>ten</P>
        <P>thirteen</P>
      </L>
    </RULE>

    <RULE NAME="gain_rule" TOPLEVEL="INACTIVE">
      <L>
        <P>increase</P>
        <P>decrease</P>
      </L>
    </RULE>

</GRAMMAR>
```

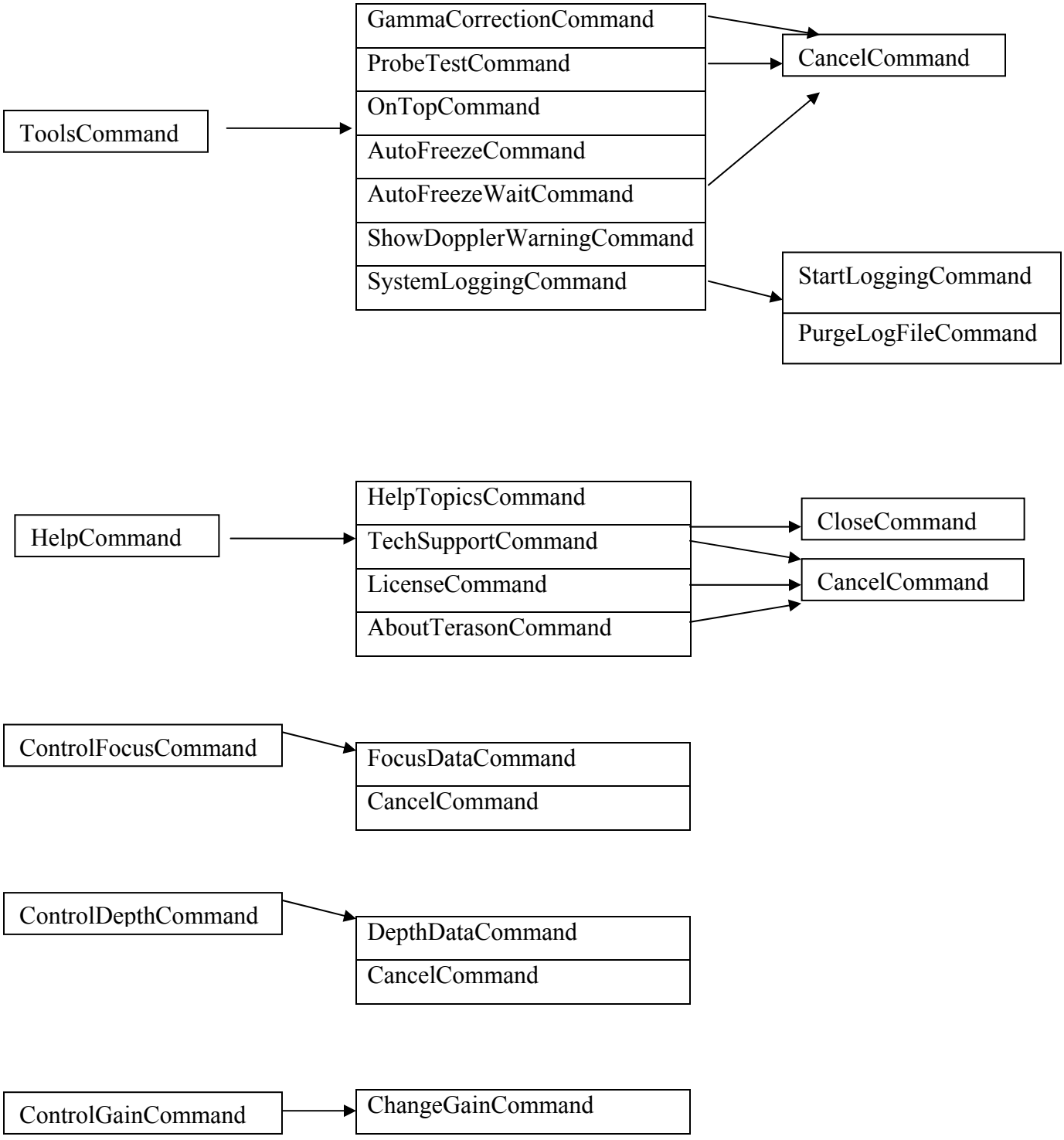
Appendix B.

The Command Tree:

The complete list of Command classes used in the voice command dispatcher, along with how they are organized in the Command tree is given here. The arrow always starts at the box containing the parent and ends at the box containing the child Command(s).




ViewCommand
ToolbarsCommand
ViewOptionsCommand
TGCDisplayCommand
FullScreenCommand
OpenExamCommand
SaveExamCommand
SaveExamAsCommand
DeleteExamCommand
ExamOptionsCommand
ExamCommand
ToolbarOptionsCommand
TGCDisplayOptionsCommand
CancelCommand
CancelCommand
CloseCommand
InsertCommand
TextCommand
MeasurementLabelCommand
CancelCommand
ImageCommand
FreezeCommand
SizeCommand
DepthCommand
FocusCommand
LiveCommand
InvertCommand
PaletteCommand
SmoothingCommand
PersistenceCommand
MapCommand
CancelCommand
InvertOptionsCommand
PaletteOptionsCommand
PersistenceOptionsCommand
MapOptionsCommand
ModesCommand
ModesOptionsCommand


ToolsCommand
GammaCorrectionCommand
ProbeTestCommand
OnTopCommand
AutoFreezeCommand
AutoFreezeWaitCommand
ShowDopplerWarningCommand
SystemLoggingCommand
CancelCommand
StartLoggingCommand
PurgeLogFileCommand
HelpCommand
HelpTopicsCommand
TechSupportCommand
LicenseCommand
AboutTerasonCommand
CloseCommand
CancelCommand
ControlFocusCommand
FocusDataCommand
CancelCommand
ControlDepthCommand
DepthDataCommand
CancelCommand
ControlGainCommand
ChangeGainCommand

Appendix C.

The grammar file used with Vocon 3200 for the recognition of the different menu items on the ultrasound scanner software window is given below. The grammar is divided into several rules and each rule can be made active and deactive by the software program with the help of APIs provided b y Vocon 3200.

```
#BNF+EM V1.0;

!grammar "Terason Grammar";

!language "American English";

!start <Speech>;

!activatable <file_rule> <cancel_rule> <view_rule> <toolbars_rule> <tgcdisplay_rule>
<exam_rule> <close_rule> <image_rule> <invert_rule> <palette_rule> <smoothing_rule>
<persistence_rule> <map_rule> <insert_rule> <modes_rule> <tools_rule> <help_rule>
<depth_rule> <focus_rule> <gain_rule>;

<Speech>: Terason <Rules>;

<Rules>: <root> | <file_rule> | <cancel_rule> | <view_rule> | <toolbars_rule> |
<tgcdisplay_rule> | <exam_rule> | <close_rule> | <image_rule> | <invert_rule> | <palette_rule> |
<smoothing_rule> | <persistence_rule> | <map_rule> | <insert_rule> | <modes_rule> |
<tools_rule> | <help_rule> | <depth_rule> | <focus_rule> | <gain_rule>;

<root>: file | edit | view | exam | image | insert | modes | tools | help | small | medium | large |
focus | depth | gain | hibernate |standby;

<file_rule>: open | "export image as" | "export loop as" | "export measurements as" | exit;

<cancel_rule>: cancel;

<view_rule>: toolbars | "status bar" | "image control bar" | "terason explorer" | "patient info" |
"image display" | "depth ruler" | "frame rate" | "reference bar" | "probe info" | "measurement
value" | "orientation logo" | "patient information" | "zoom thumb nail" | "zoom image" | refresh |
"t g c display" | "full screen";

<toolbars_rule>: file | freeze | measurement | "play back" | "scan mode";

<tgcdisplay_rule>: on | off | timeout;
```

<exam_rule>: "open exam" | "save exam" | "save exam as" | "delete exam" | abdominal | cardiac | obstetrical | pelvic | prostate;

<close_rule>: close;

<image_rule>: freeze | size | depth | focus | live | invert | palette | smoothing | persistence | map;

<invert_rule>: "left right" | "up down";

<palette_rule>: gray | tan | flame | sepia | magenta | sage | rainbow | cobalt;

<smoothing_rule>: a | b | c | d | e;

<persistence_rule>: zero | one | two | three | four;

<map_rule>: a | b | c | d | e | f;

<insert_rule>: text | "distance measurement" | "measurement label";

<modes_rule>: "b mode" | "m mode" | "pulsed wave doppler" | "color doppler" | "directional power doppler" | "power doppler";

<tools_rule>: "gamma correction" | "probe verification test" | "system logging" | "always on top" | "auto freeze" | "set auto freeze wait time" | "show color doppler saving warning";

<help_rule>: "help topics" | "technical support" | license | "about terason";

<depth_rule>: seven | eight | nine | ten | eleven | twelve | thirteen | fourteen | fifteen | sixteen | seventeen | eighteen | nineteen | twenty | "twenty one" | "twenty two" | "twenty three" | "twenty four";

<focus_rule>: three | four | "five point five" | seven | "eight point five" | ten | thirteen;

<gain_rule>: increase | decrease;

Appendix D.

The command set chosen for performing the testing described in Chapter 7 and Chapter 8 is described here.

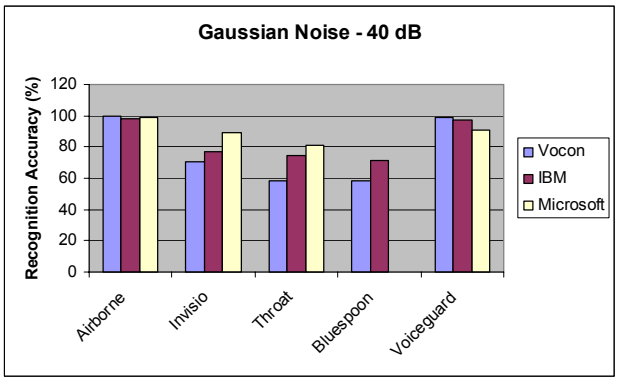
Terason testing
Terason testing
Terason view
Terason image
Terason small size
Terason medium size
Terason set focus
Terason change gain
Terason toolbars
Terason depth ruler
Terason live
Terason brightness
Terason motion
Terason full screen
Terason open exam
Terason power doppler
Terason probe info
Terason standby
Terason hibernate
Terason invert image
Terason color rainbow
Terason scan mode
Terason display pelvic
Terason help topics
Terason measure distance
Terason new patient
Terason frame rate

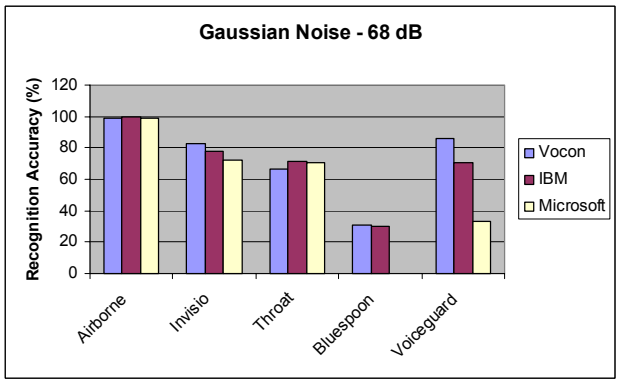
Appendix E.

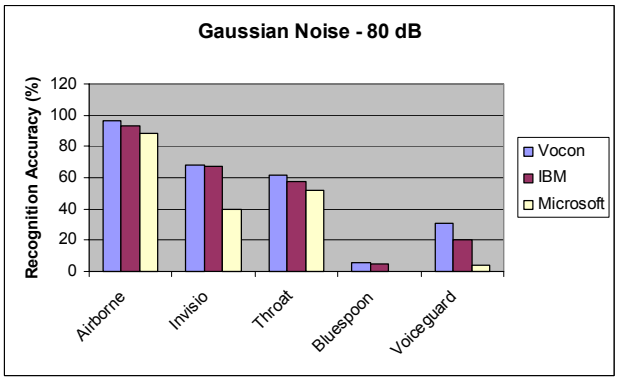
The complete results of microphone and ASR evaluation are given in this section.

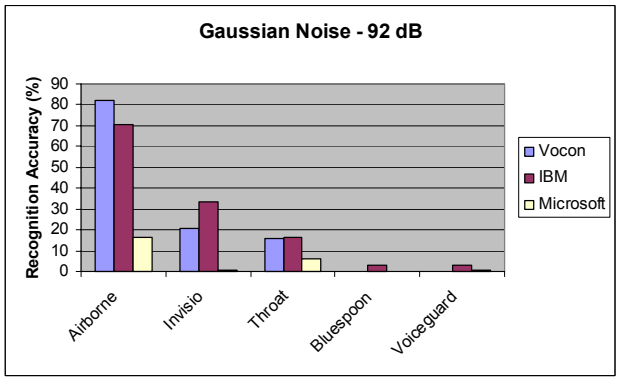
E.1. Comparison of between ASR engines and microphones at different SPLs

The results obtained from the comparison of the ASR engines, Vocon 3200, IBM Viavoice and Microsoft Command and Control with all the 5 different microphones at different SPLs are given here.



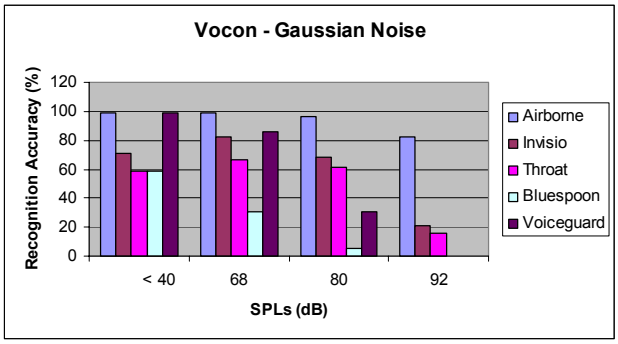


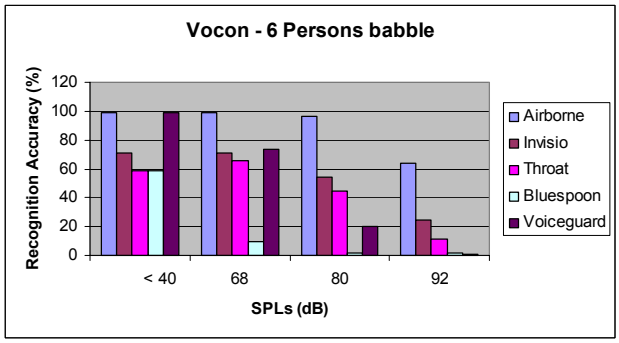


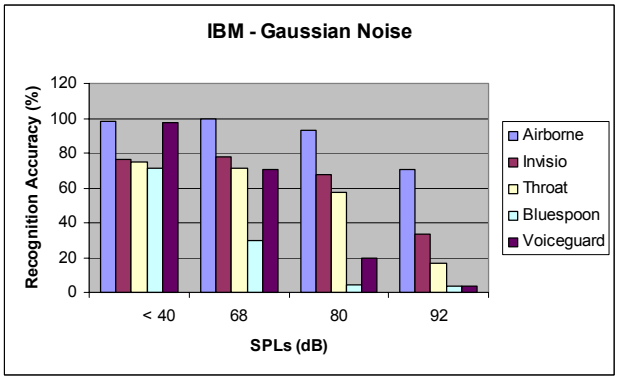


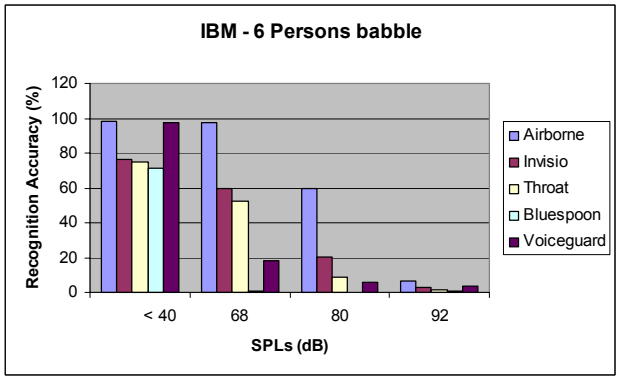
E.2. Performance with different microphones

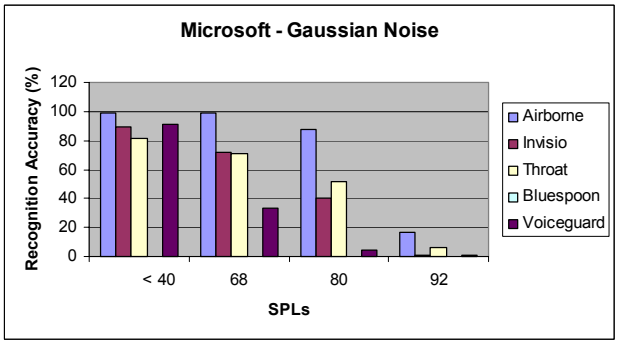
The recognition performance of the three different ASR engines, Vocon 3200, IBM Viavoice and Microsoft Command and Control at two different noise types are plotted here. The results indicate the variation with the different microphones at different SPLs.

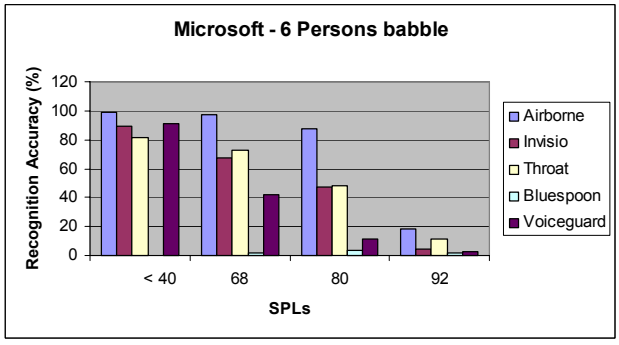






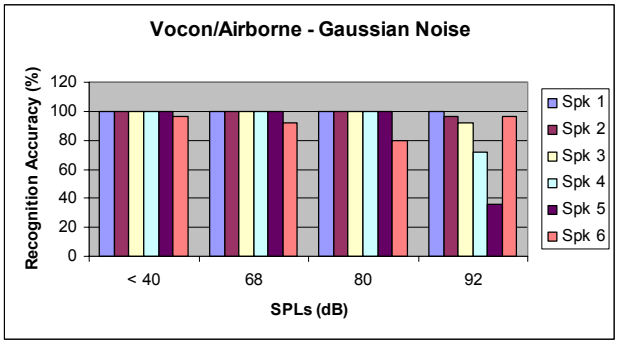


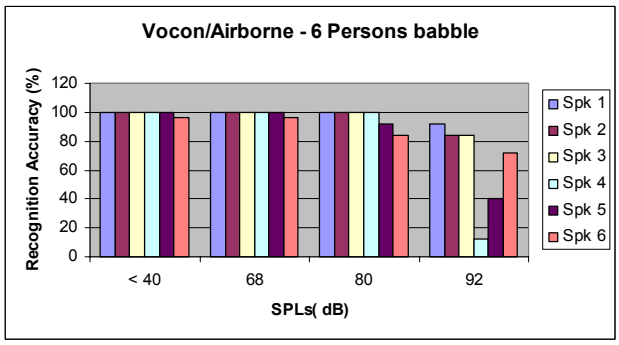


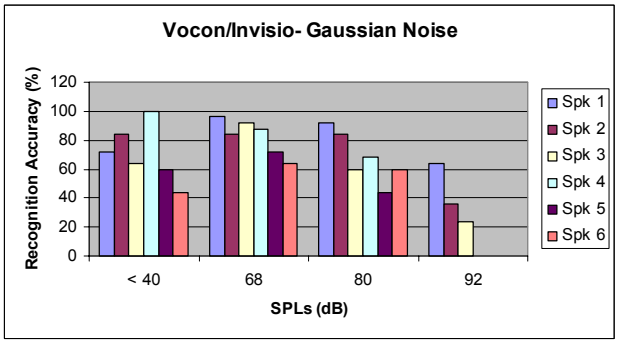


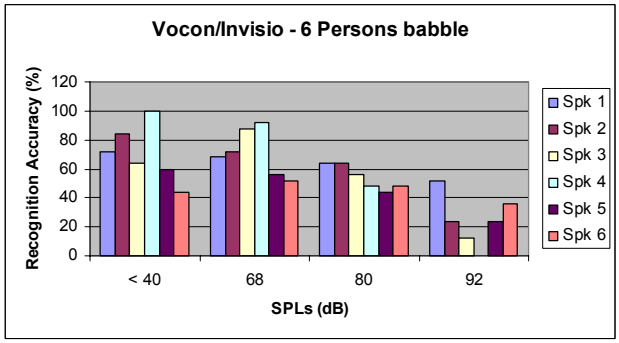
E.3 Performance with different speakers

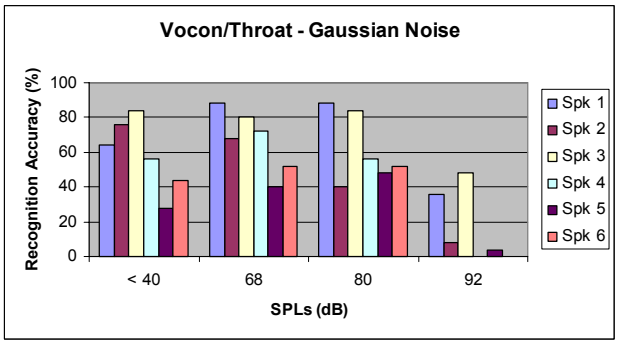
The performance of Vocon 3200 with different speakers for each microphone in two different noise types at various SPLs is given below.

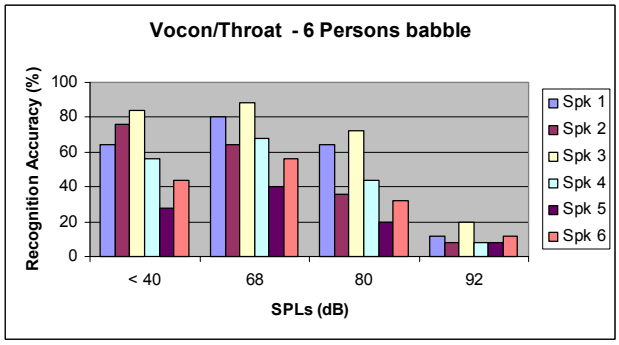


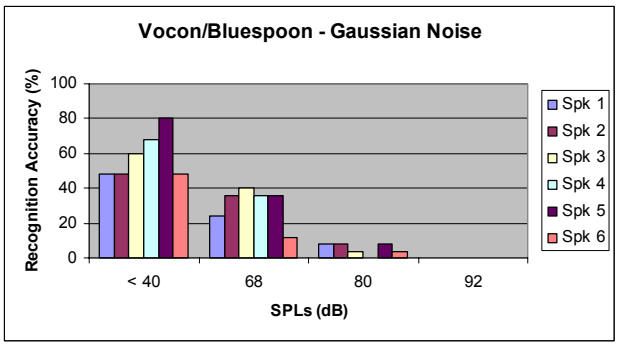


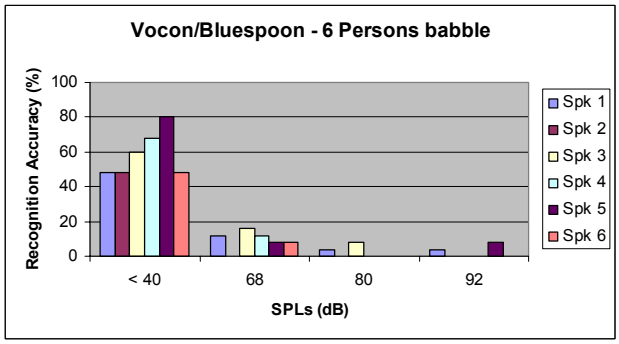


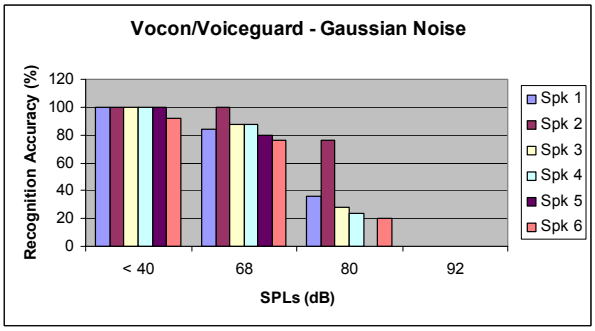


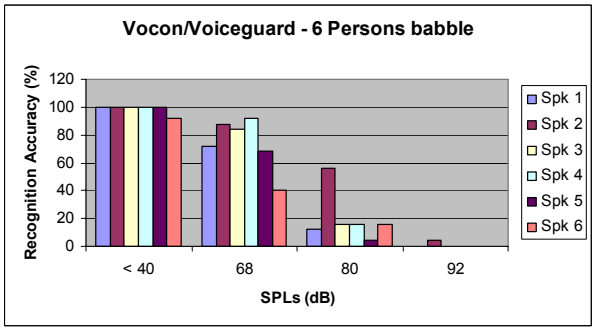




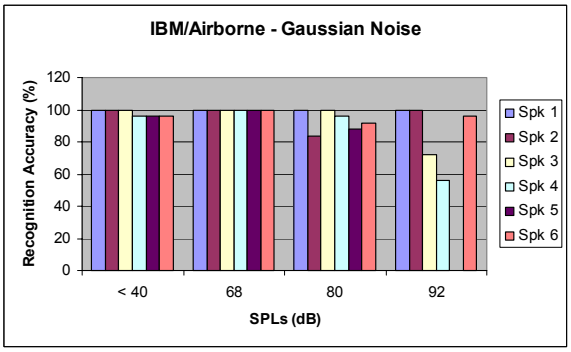


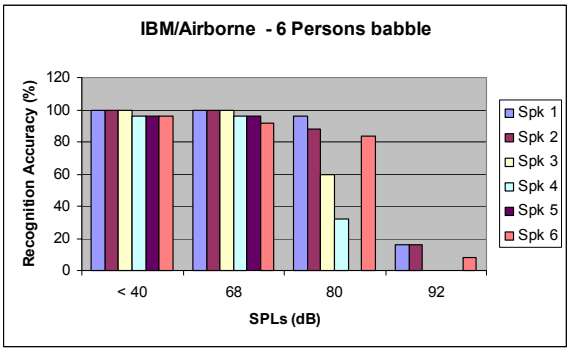


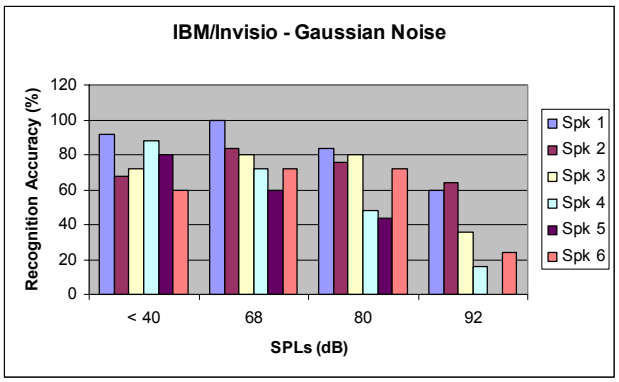


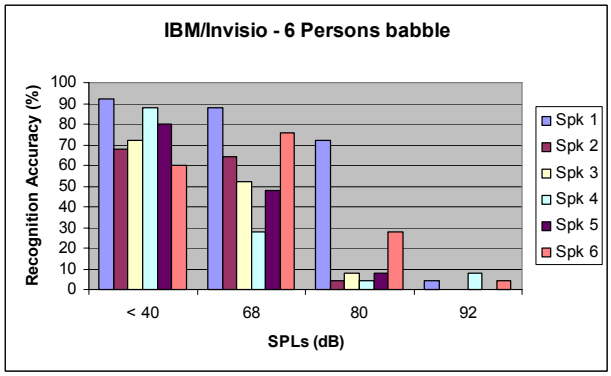


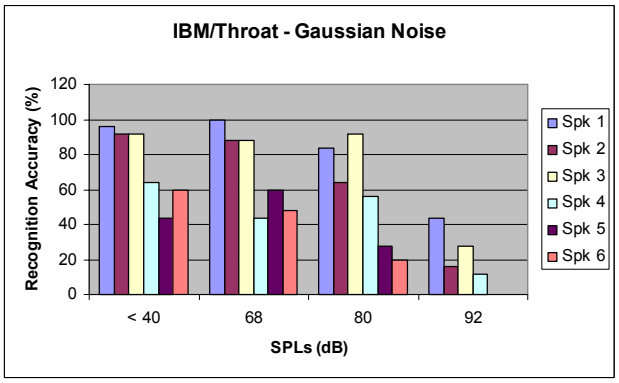
The performance of IBM Viavoice with different speakers for each microphone in two different noise types at various SPLs is given below.

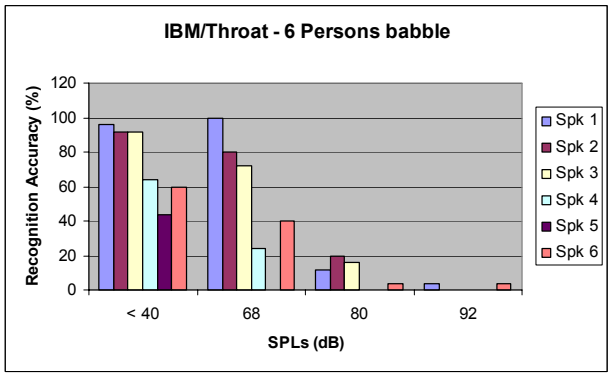


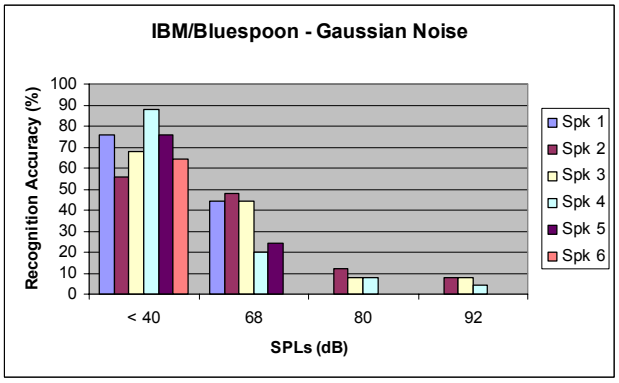


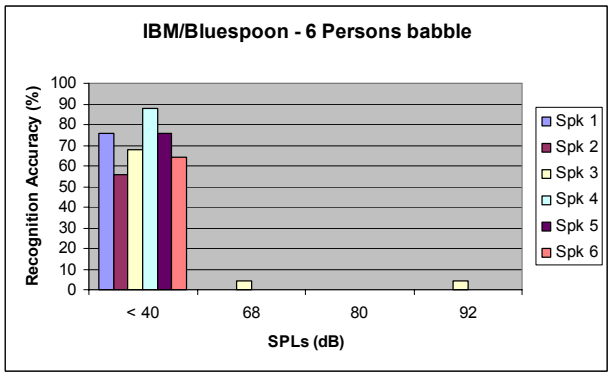

IBM/Invisio - Gaussian Noise
Recognition Accuracy (%)
120
100
80
60
40
20
0
< 40
68
80
92
SPLs (dB)
Spk 1
Spk 2
Spk 3
Spk 4
Spk 5
Spk 6

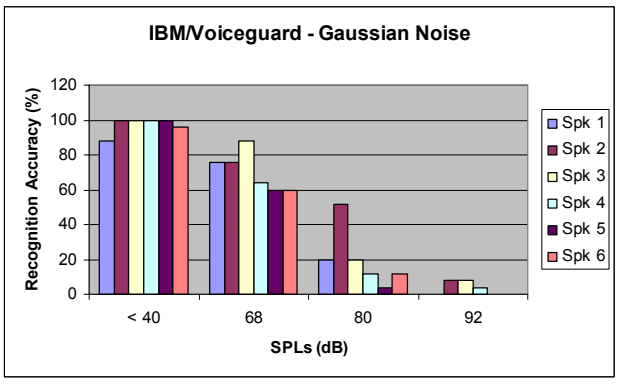

IBM/Invisio - 6 Persons babble
Recognition Accuracy (%)
100
90
80
70
60
50
40
30
20
10
0
< 40
68
80
92
SPLs (dB)
Spk 1
Spk 2
Spk 3
Spk 4
Spk 5
Spk 6

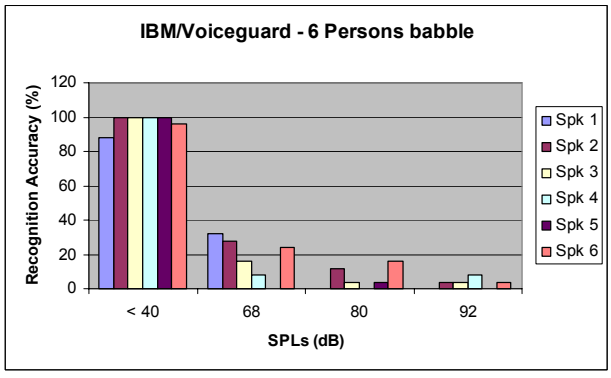

IBM/Throat - Gaussian Noise
Recognition Accuracy (%)
120
100
80
60
40
20
0
< 40
68
80
92
SPLs (dB)
Spk 1
Spk 2
Spk 3
Spk 4
Spk 5
Spk 6


IBM/Throat - 6 Persons babble
Recognition Accuracy (%)
120
100
80
60
40
20
0
< 40
68
80
92
SPLs (dB)
Spk 1
Spk 2
Spk 3
Spk 4
Spk 5
Spk 6

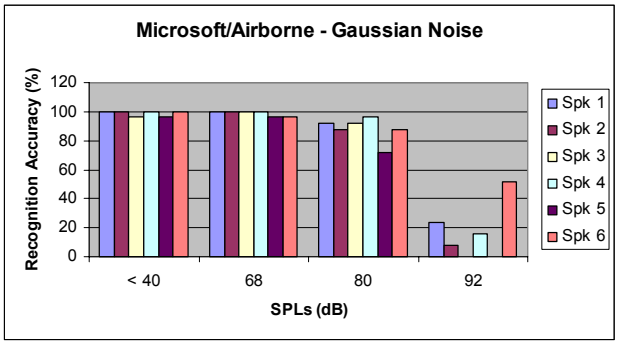

IBM/Bluespoon - Gaussian Noise
Recognition Accuracy (%)
100
90
80
70
60
50
40
30
20
10
0
< 40
68
80
92
SPLs (dB)
Spk 1
Spk 2
Spk 3
Spk 4
Spk 5
Spk 6

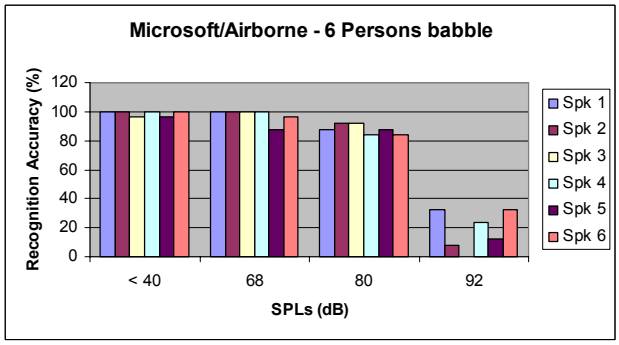

IBM/Bluespoon - 6 Persons babble
Recognition Accuracy (%)
100
90
80
70
60
50
40
30
20
10
0
< 40
68
80
92
SPLs (dB)
Spk 1
Spk 2
Spk 3
Spk 4
Spk 5
Spk 6

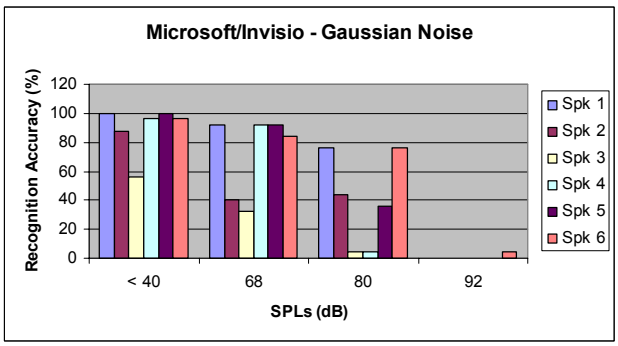

IBM/Voiceguard - Gaussian Noise
Recognition Accuracy (%)
120
100
80
60
40
20
0
< 40
68
80
92
SPLs (dB)
Spk 1
Spk 2
Spk 3
Spk 4
Spk 5
Spk 6

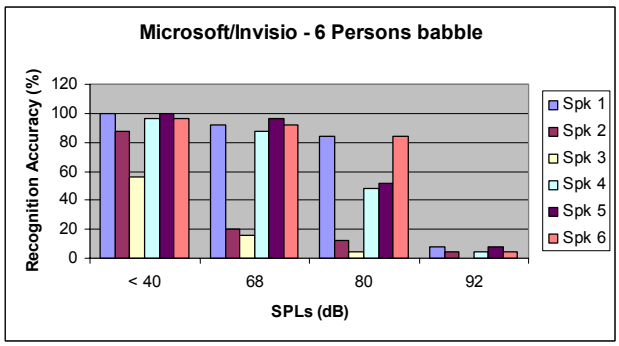

IBM/Voiceguard - 6 Persons babble
Recognition Accuracy (%)
120
100
80
60
40
20
0
< 40
68
80
92
SPLs (dB)
Spk 1
Spk 2
Spk 3
Spk 4
Spk 5
Spk 6

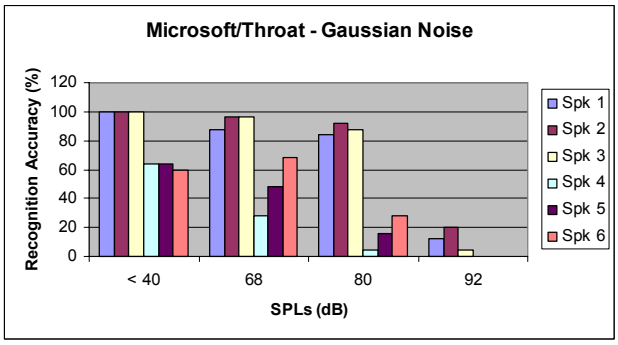
The performance of Microsoft Command and Control with different speakers for each microphone in two different noise types at various SPLs is given below.

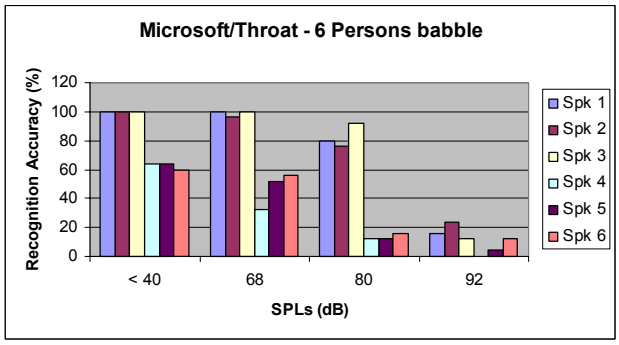


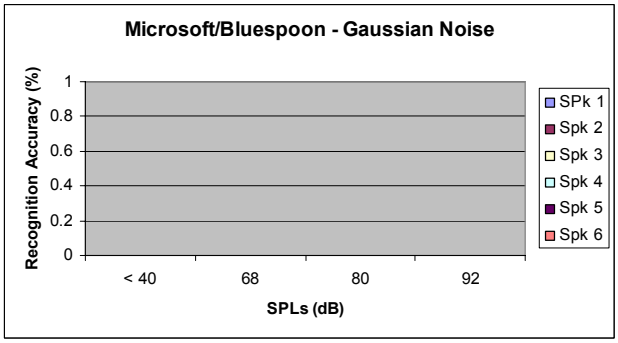


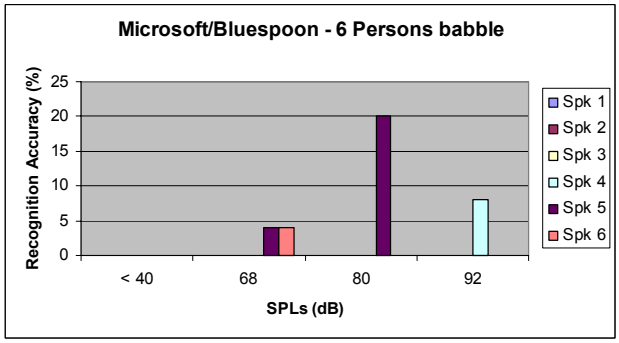


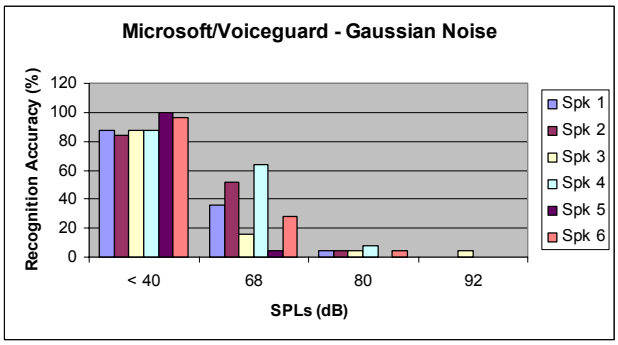


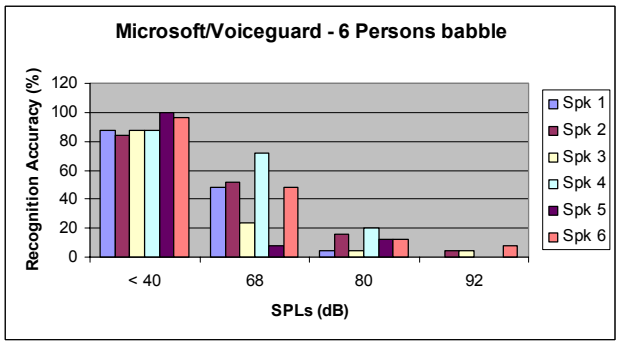









Microsoft/Voiceguard - Gaussian Noise
Recognition Accuracy (%)
120
100
80
60
40
20
0
< 40
68
80
92
SPLs (dB)
Spk 1
Spk 2
Spk 3
Spk 4
Spk 5
Spk 6


Microsoft/Voiceguard - 6 Persons babble
Recognition Accuracy (%)
120
100
80
60
40
20
0
< 40
68
80
92
SPLs (dB)
Spk 1
Spk 2
Spk 3
Spk 4
Spk 5
Spk 6

Appendix F.

INFORMED CONSENT TO TAKE PART IN A RESEARCH STUDY

TITLE OF STUDY: **Evaluation of microphones and speech recognition engines in noisy environment**

INVESTIGATOR: **Peder C. Pedersen, Dept. of Electrical and Computer Eng., Worcester Polytechnic Institute, 508-831-5641**

SPONSOR: **Telemedicine and Advanced Technology Research Center (TATRC)**

Introduction
You are asked to participate in a research study. It is important that you read the following explanation of this research study. This form describes the purpose of the study, the experimental protocol, the description of risks, benefits and precautions. It also describes your right to withdraw from the study at any point in time.

Purpose of Study
The purpose of the study is to evaluate the voice recognition performance of various combinations of microphones and automatic speech recognition (ASR) engines, under noisy conditions, in order to be able to select the combination that offers the best performance for control of a wearable medical ultrasound scanner system.

Experimental Protocol
The testing will be performed in an acoustic chamber, belonging to Prof. Campbell and housed in the Higgins House Garage on WPI Campus. The chamber is illustrated in figure 1. The test chamber is highly reverberant, with speakers mounted in the walls, and will thus produce a near-uniform sound pressure of non-directional sound. The plan is to carry out tests with two different types of noise, at three different *Sound Pressure Levels* (SPLs), and also carry out tests under quiet conditions. The noise types are created to resemble forms of blended human speech.

The SPLs have been chosen so that a speech recognition system, when receiving input from an air-borne microphone (boom microphone), will recognize most of the spoken words (approx. 85%) at the lowest of the three SPLs and will only recognize some the spoken words (approx. 15%) at the highest SPL. It was determined that the 85% recognition rate, the 50% recognition rate and the 15% recognition rate were achieved at SPL noise levels of 68 dB, 80 dB, and 92 dB, respectively. For SPLs above 80 dB, efficient hearing protection is provided for the subject.

Apart from a 30 second initial reading by the subject, for the purpose of audio tuning, the duration of the testing of one microphone with one subject is equal to the time for playing all the tracks on the noise CD. As will be discussed later, this time is about 21 minutes. The experiments will be run test with two microphones at the same time, in order to cut down on the time that the subject has to spend. The output of the two microphones (or the single microphone) that the subject is wearing is stored as voice files on the two laptops, and these voice files will later be recorded onto test CDs.

When the actually testing of the level of speech recognition is carried out later (after the testing with a given subject has been completed), one test CD is placed in one laptop, which produces an audio output that is applied to the microphone input to another laptop, containing the speech recognition software packages.

A *cueing device* is connected to an LED mounted inside the acoustic chamber as well as an LED, mounted next to the laptops and CD player, outside the acoustic chamber. The LED flashes every five second for 0.5 sec., in order that the subject reads the command words at the precise moments and so that the duration of the noise tracks corresponds to the time that the subject takes to read the 27 command words.

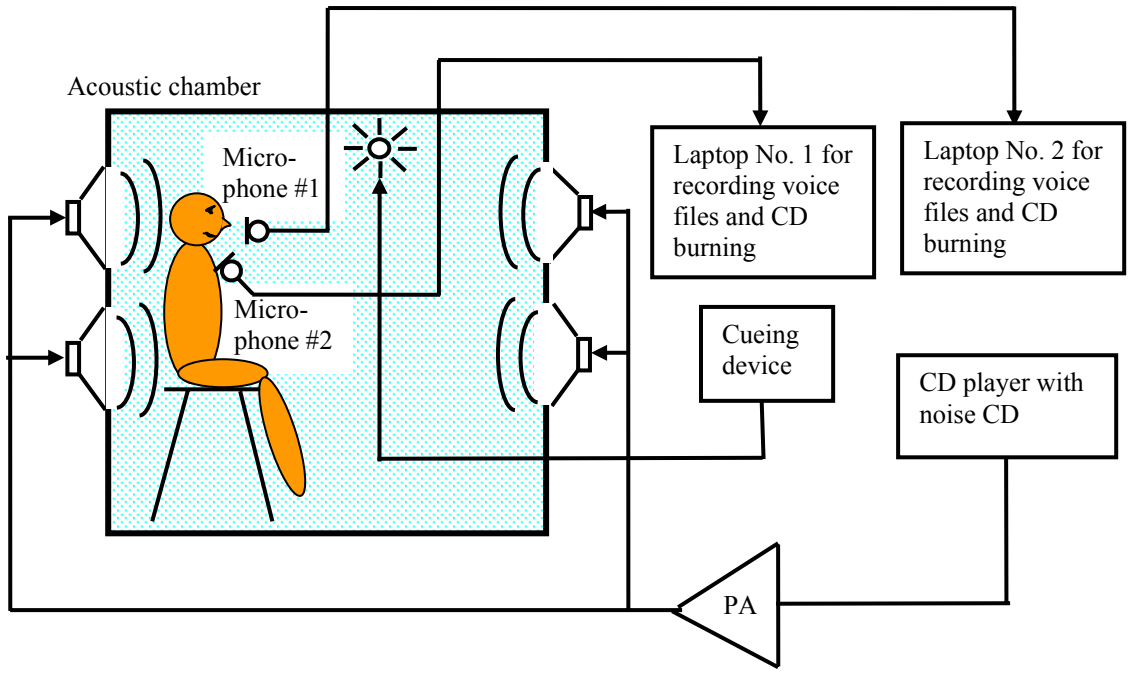


Figure 1. Lay-out of the acoustic chamber

The voice recording sequence

After the subject has become familiarized with the 27 command words (by reading the set of words aloud a couple of times for practice) and with the testing protocol, the subject is seated in the acoustic chamber, as shown in Figure 1. When ready, the following sequence of voice files are produced on one of the laptops, later to be transferred to CD. The 27 command words are listed later in this document.

Voice file 1: The subject reads thirty seconds of a specific text passage, under quite conditions, to be used for audio tuning of the speech recognition system. Thus, the cueing device is off, and the noise CD is not played.

Voice file 2: The subject reads of the 27 command words under quiet conditions. Thus, the cueing device is on and produces the sequence of light signals shown in Figure 2, but the noise CD is not played.

Voice file 3: The subject reads of the 27 command words while track 1 of the noise CD is played (noise type I: unmodulated Gaussian noise, at level REF dB + 0dB). The cueing device is on and produces the sequence of light signals shown in Figure 2.

Voice file 4: The subject reads of the 27 command words while track 2 of the noise CD is played. The cueing device is on and produces the sequence of light signals.

- - - - -
- - - - -
Voice file 8: The subject reads of the 27 command words while track 6 of the noise CD is played. The cueing device is on and produces the sequence of light signals.

The total duration of these voice files are 30 sec. + 8 x 155 sec. = 1270 sec. = 21 minutes and 10 sec. Thus, the voice files for five different microphones can fit on one CD.

Selection of subjects for testing
Six subjects will be selected for testing. Of the six subjects, four will be American born, and two will be foreign born. The subjects are all between 20 and 30 and in good health. Of the four American born subjects, three will be males and one will female. Of the foreign born subjects, one will be male and one will be female.

Benefits
The only direct benefit associated with your participation in this experiment is the compensation, if any, received. Indirect benefits to society may be derived from implementation of the scientific knowledge gained during this research project.

Risks
As you will be equipped with hearing protection when the SLP exceeds 80 dB, there will be no risks to you. The hearing protection is rated to give an SPL reduction of about 30 dB, so that at 92 dB, you will be exposed to noise levels around 62 – 65 dB.If you have tendencies to claustrophobia, you may decide not to participate, as the testing takes place in a closed chamber.

Participation
The participation in this study is entirely voluntary. You are free to withdraw consent and discontinue at any time. You are free to seek further information regarding the experiments at any time. Similarly, the investigators have the right to cancel or postpone the testing.

Smoking Policy
Worcester Polytechnic Institute is a smoke-free workplace. No smoking of any kind is permitted inside nonresidential campus buildings

Confidentiality
Data obtained in this experiment will become the property of the investigators and WPI. The subject will remain anonymous in all publications related to this research.

Supervision
This study will be directly supervised by Peder C. Pedersen and Dalys Sebastian (graduate student). Questions or comments about participation should be directed to Peder C. Pedersen at (508) 831-5641

Consent
Please read the following statement and sign on the line below if you agree to participate in the study

Cost/Payment
You will receive $100 for completion of this study. If the experimental session is not completed, you will be paid $10 per completed hour.

VOLUNTEER'S STATEMENT

I have read this document and understand the purpose of the study, and what will be expected of me if I agree to participate. By signing this consent form, I agree freely to participate as a research subject without any pressure having been placed on me to do so.

I hereby grant the investigators right to use the results of the speech recognition experiments in publications, presentations and theses; however, without any reference to my name.

I understand that participation in the study is strictly voluntary. I know that I may quit at any time without losing the benefits to which I am entitled. I also understand that the investigators in charge of the study may decide at any time that I should no longer participate in the study.

By signing this form, I have not waived any of my legal rights.

I have read and understand the above information. I agree to participate in this study. I have been given a copy of this signed and dated document for my own records.

_____ _____
Study participant (signature) Date

_____
Print participant's name

_____ _____
Person who explained this study (signature) Date