Nepali Speech Recognition

Chetan Prajapati (24208)
Jiwan Nyoupane (24211)
Jwalanta Deep Shrestha (24212)
Shishir Jha (24239)

February, 2008

Acknowledgment

First of all, our sincere thanks go to the Department of Electronics and Computer Engineering for providing the opportunity to do this project.

Research in Speech Recognition is a field which needs continuous effort with in-depth research and we are indebted to our project supervisors Assistant Professors Shri Daya Sagar Baral and Shri Jeevan Kumar Pant for the interest in the project and their invaluable suggestion and ideas. Similarly we are also thankful to our Project Coordinator Assistant Professor Shri Rajendra Lal Rajbhandari for assisting and guiding us in our project.

We would like to thank our colleagues as well as teachers who have been providing encouragement and helping us with valuable comments throughout this project.

Chetan Prajapati (060/BCT/509)
Jiwan Nyoupane (060/BCT/512)
Jwalanta Deep Shrestha (060/BCT/513)
Shishir Jha (060/BCT/541)

February, 2008

Abstract

Speech recognition is the process of converting a speech signal to a sequence of words, by means of algorithms implemented as a computer program. This project analyzes the existing models for recognition of Nepali speech. Upon finding the shortcomings of the existing models, we move on to *Ear Model*, which is the simulation of how human ear and brain work together for speech recognition. This model eventually was found out to provide better accuracy than conventional methods. To further alleviate accuracy, provision for Nepali Dictionary checking and the database of syllables' frequencies from Nepali corpus has also been devised. By using these techniques we have been able to achieve a speech recognition model that can be helpful to all seeking better model of speech recognition of Nepali language and other languages as well.

Keywords

Nepali speech, Speech Recognition, Conventional methods, Ear Model

Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Program Interface
ANN	Artificial Neural Network
ASR	Automatic Speech Recognition
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
GUI	Graphical User Interface
IDFT	Inverse Discrete Fourier Transform
HMM	Hidden Markov Model
HTK	HMM Toolkit
MFCC	Mel Frequency Cepstral Coefficeint
NSR	Nepali Speech Recognition
OSS	Open Source Software
SNR	Signal to Noise Ratio
SQL	Structured Query Language

Chapter 1

Introduction

1.1 Background

Ever since the PCs started appearing in the market where it was affordable to general mass, humans have used it numerous application. PC or Personal Computer has in some way revolutionized how people do their work, because, the computers now are not only available to few elites how had the money to buy the bigger expensive pre-PC computer but also to average people. PCs not only have helped its users by reducing the their work load but also have helped them in many different aspects by providing them facilities like gaming, Internet surfing, and lot more. These aspect of computer has helped PC makers to encourage even the children to use computers.

Personal computers and technology associated with it has never looked back ever since it started in late 1980s. From its early days, when it was used for spreadsheet applications, word processing, designing and other monotonous jobs, computers have come a long way. Today, it has enough power to handle multiple programs simultaneously making it possible for general mass to enjoy high end applications in their home. Due to this computers have penetrated in many different areas as an primary tool for computing. This has caused huge amount of reliance of a large population of humans on computer for their day to day work.

But still with all that has happened all around the world, in Nepal, more than 80% of population is untouched by any form of computer access. Even though many projects are being developed by the government there has not been considerable advancement in getting the rural population to use the computer. One of the major hindrance that the government is facing in deploying the computers to the rural areas is the literacy rate of our country. With an average literacy rate of less than 45% in Nepal, many targeted users

lack basic knowledge required to operate a computer, creating a problem in using the computer through traditional method. Hence, an alternative way of computing is necessary so that this group of people along with the normal users can have a better way of utilizing the computer.

1.2 Speech as computer input

Speech is a natural mode of communication for people. Human learn all the relevant skills during early childhood, without instruction, and we continue to rely on speech communication throughout our lives. It comes so naturally to us that we don't realize how complex a phenomenon speech is. The human vocal tract and articulators are biological organs with nonlinear properties, whose operation are not just under conscious control but also affected by factors ranging from gender to upbringing to emotional state.

Therefore, as people are so comfortable with speech, that if they can interact with the computers also via speech, rather than having to resort to primitive interfaces such as keyboards and pointing devices, then probably they will embrace computers much nicely. A speech interface would even support regular users in numerous comprehensive tasks like telephone directory assistance, spoken database querying for novice users, "hands- busy" applications in medicine or fieldwork, office dictation devices, or even automatic voice translation into foreign languages. Due to these valuable applications it has motivated numerous research in automatic speech recognition since the 1950's. Great progress has been made so far, especially since the 1970's, using a series of engineered approaches that include template matching, knowledge engineering, and statistical modeling.

1.3 Need

Most of the works done till now has been focused only in the English language. Nepali computation is still in infancy stage because of the input system. Computer primarily being an English device, most people in Nepal who use computer find it convenient in English. For those who want to use computer in Nepali, the main hindrance happens to be the input system. Though input systems have been devised that provides keyboard input in Nepali Language much remains to be done to breach the hindrance and make Nepali input system widely acceptable.

This can change with a Nepali Speech Recognition system. As we all know that, if applicable, speech is a better way to communicate with com-

puter than keyboard. This can change the way we use computer in Nepali language. Furthermore, with computers reaching to remote areas of Nepal, a Nepali Speech Recognition can play an important role providing an easy and easily adaptable input system for novice users of computer. But, we will see in the following sections that speech recognition not only has a good application as a input system but has numerous other applications, such as Medical Transcription, Military usages, telephony, etc, where it can benefit from Nepali system.

Though the basic concepts of human hearing are fairly simple, the specific structures in the human ears are extremely complex. Even though scientist all over the world are making many headways in discovering new and better speech recognition algorithms every year, it's pretty astonishing how much is involved in the process of hearing. Due to this reason there has till date not been a single project which has achieved a 100% accuracy in recognizing speech.

1.4 Nepali Speech Recognition

This project is aimed at comparing different methods that already exists for speech recognition to find the method that is most suitable for Nepali Speech Recognition. In this project we have compared two different approaches of the statistical template based pattern matching recognition along with a radical new approach of recognition system, which we have named as *Ear Model*.

In the following sections we describe the structure of Nepali language along with how human hearing works since our model is entirely based on it. Further we have described what speech recognition is and what the domains are where it has found extensive use. The later sections contains details about the different types of speech recognition system along with the data from the experimentation that we have done. Thereafter, the next section contains description, detailed design information and technical details about the *Ear Model* that we have proposed for Nepali Speech Recognition.

Chapter 2

Nepali Language Structure

2.1 Classification of Vowel phonemes

In Devanagari, there are total of 14 vowels [Table 2.1]. Out of these vowels, spoken Nepali uses only six of them [Table 2.2]. Other vowels are in fact diphthongs.

Nepali vowels can be categorized in several ways.

- Height of tongue
- Activeness of tongue
- Position of lips

2.2 Height of tongue

According to height of tongue, the vowels are divided into three levels.

- High: The vowels इ and उ fall under this.
- Middle: The vowels ए and ओ fall under this
- Low: The vowels आ and अ fall under this.

2.3 Activeness of the tongue

Different vowels require activeness of different part of the tongue. In Nepali vowels, इ and ए are pronounced by the activeness of the front part of tongue. Similarly, the vowels आ , उ , ओ and अ are pronounced by the activeness of the back part of the tongue

S.No.	Sound	Symbol
1	/a/	अ
2	/aa/	आ
3	/e/	इ
4	/ee/	ई
5	/u/	उ
6	/uu/	ऊ
7	/ri/	
8	/rii/	
9	/e/	ए
10	/ai/	ऐ
11	/o/	ओ
12	/au/	औ
13	/am/	अ
14	/aha, /	अ

Table 2.1: Nepali Vowels

S.No.	Sound	Symbol
1	/a/	अ
2	/aaa/	आ
3	/e/	इ
4	/u/	उ
5	/a/	ए
6	/au/	ओ

Table 2.2: Nepali spoken vowels

2.4 Roundness of lips

According to the roundness of lips, there are two kind of vowels – rounded vowels and unrounded vowels.

- Rounded Vowels: उ, ओ
- Unrounded Vowels: इ, ए, अ, आ

Apart from these, vowels can also be categorized on the basis of muscular tension and nasalness. Vowels अ, ए and ओ fall under lax vowels, whereas इ, आ and उ fall under tense vowels.

The classification of vowels can be summarized as in [Table 2.3].

Vowels	Height of Tongue	Activeness of tongue	Roundness of tongue	Muscular tension
अ /a/	low	Back	Unrounded	lax
आ /aa/	low	Back	Unrounded	tense
इ /i/	high	Front	Unrounded	tense
उ /u/	high	Back	Rounded	tense
ए /e/	middle	Front	Unrounded	lax
ओ /o/	middle	Back	Rounded	lax

Table 2.3: Classification of Vowel

2.5 Classification of consonant phoneme

Devanagari has altogether 36 consonants. Out of these, Nepali spoken language only uses 29 consonants. They are shown in [Table 2.5.4].

The Nepali consonants are classified as follows:

- Place of articulation
- Manner of articulation
- Phonation
- Aspiration

2.5.1 Place of articulation

Bilabial

Bilabial consonant is a consonant articulated with both lips.

S.No.	**Sound**	**Spoken**
1	/ka/	क
2	/kha/	ख
3	/ga/	ग
4	/gha/	घ
5	/nga/	ङ
6	/cha/	च
7	/chha/	छ
8	/ja/	ज
9	/jha/	झ
10	/ta/	ट
11	/tha/	ठ
12	/da/	ड
13	/dha/	ढ
14	/ta/	त
15	/tha/	थ
16	/da/	द
17	/dha/	ध
18	/na/	न
19	/pa/	प
20	/pha/	फ
21	/ba/	ब
22	/bha/	भ
23	/ma/	म
24	/ya/	य
25	/ra/	र
26	/la/	ल
27	/wa/	व
28	/sa/	स
29	/ha/	ह

Table 2.4: Consonants in Nepali spoken language

Dental

Dental consonant or dental is a consonant that is articulated with the tongue against the upper teeth

Alveolar

Alveolar consonants are articulated with the tongue against or close to the superior alveolar ridge, which is called that because it contains the alveoli (the sockets) of the superior teeth.

Palatal

Palatal consonants are consonants articulated with the body of the tongue raised against the hard palate (the middle part of the roof of the mouth). Consonants with the tip of the tongue curled back against the palate are called retroflex.

Velar

Velars are consonants articulated with the back part of the tongue (the dorsum) against the soft palate (the back part of the roof of the mouth, known also as the velum).

Glottal

Glottal consonants are consonants articulated with the glottis.

2.5.2 Manner of articulation

Plosive / Stop Consonants

A stop, plosive, or occlusive is a consonant sound produced by stopping the airflow in the vocal tract.

Affricate

Affricate consonants begin as stops (most often an alveolar) but release as a fricative rather than directly into the following vowel.

Fricative

Fricatives are consonants produced by forcing air through a narrow channel made by placing two articulators close together.

Bilabial	Dental	Alveolar	Palatal	Velar	Glottal
प	त	च	य	क	ह
फ	थ	छ		ख	
ब	द	ज		ग	
भ	ध	झ		घ	
म		ट		ङ	
व		ठ			
		ड			
		ढ			
		न			
		र			
		ल			
		स			

Table 2.5: Classification of consonants according to place of articulation

Nasal

A nasal consonant (also called nasal stop or nasal continuant) is produced when the velum—that fleshy part of the palate near the back—is lowered, allowing air to escape freely through the nose.

Lateral

Laterals are consonants pronounced with an occlusion made somewhere along the axis of the tongue, while air from the lungs escapes at one side or both sides of the tongue.

Trilled

A trill is a consonantal sound produced by vibrations between the articulator and the place of articulation.

Approximant / semi-vowel

Approximants are speech sounds that could be regarded as intermediate between vowels and typical consonants. In the articulation of approximants, articulatory organs produce a narrowing of the vocal tract, but leave enough space for air to flow without much audible turbulence.

Plosive	Affricate	Fricative	Nasal	Lateral	Trilled	Semi Vowel
क ख ग घ ट ठ ड ढ त थ द ध प फ ब भ	च छ ज झ	स ह	ङ न म	ल	र	य व

Table 2.6: Classification of consonants according to manner of articulation

2.5.3 Phonation

A voiced sound is one in which the vocal cords vibrate, and a voiceless sound is one in which they do not.

Voiced Consonants	क ख च छ ट ठ त थ प फ स
Voiceless Consonants	ग घ ङ ज झ ड ढ द ध न ब भ म य र ल व ह

Table 2.7: Classification of Nepali consonants according to phonation

2.5.4 Aspiration

Aspiration is the strong burst of air that accompanies either the release or, in the case of pre-aspiration, the closure of some obstruents.

Unaspirated	क ग ङ च ज ट ड त द न प ब म य र ल व
Aspirated	ख घ छ झ ठ ढ थ ध फ भ स ह

Table 2.8: Classification of Nepali consonants according to aspiration

The categorization is summarized in Table 2.5.4:

2.6 Structure of Nepali Syllable

The structure of syllables in Nepali language can be summarized as in [Table 2.10].

S.No.	Sound	Place of articulation	Manner of articulation	Phonation	Aspiration	Nasal
1	क	velar	plosive	voiceless	unaspirated	oral
2	ख	velar	plosive	voiceless	aspirated	oral
3	ग	velar	plosive	voiced	unaspirated	oral
4	घ	velar	plosive	voiced	aspirated	oral
5	ङ	velar	nasal	voiced	unaspirated	nasal
6	च	alveolar	affricate	voiceless	unaspirated	oral
7	छ	alveolar	affricate	voiceless	aspirated	oral
8	ज	alveolar	affricate	voiced	unaspirated	oral
9	झ	alveolar	affricate	voiced	aspirated	oral
10	ट	alveolar	plosive	voiceless	unaspirated	oral
11	ठ	alveolar	plosive	voiceless	aspirated	oral
12	ड	alveolar	plosive	voiced	unaspirated	oral
13	ढ	alveolar	plosive	voiced	aspirated	oral
14	त	dental	plosive	voiceless	unaspirated	oral
15	थ	dental	plosive	voiceless	aspirated	oral
16	द	dental	plosive	voiced	unaspirated	oral
17	ध	dental	plosive	voiced	aspirated	oral
18	न	alveolar	nasal	voiced	unaspirated	nasal
19	प	bilabial	plosive	voiceless	unaspirated	oral
20	फ	bilabial	plosive	voiceless	aspirated	oral
21	ब	bilabial	plosive	voiced	unaspirated	oral
22	भ	bilabial	plosive	voiced	aspirated	oral
23	म	bilabial	nasal	voiced	unaspirated	nasal
24	य	palatal	semi vowel	voiced	unaspirated	oral
25	र	alveolar	trilled	voiced	unaspirated	oral
26	ल	alveolar	lateral	voiced	unaspirated	oral
27	व	bilabial	semi-vowel	voiced	unaspirated	oral
28	स	alveolar	fricative	voiceless	aspirated	oral
29	ह	glottis	fricative	voiced	aspirated	oral

Table 2.9: Summary of Nepali consonants

S.No.	Syllable Pattern	Symbol	Example
1	Vowel	V	आ इ उ
2	Vowel + Consonant	VC	उस
3	Consonant + Vowel	CV	खा हौ
4	Consonant + Vowel + Consonant	CVC	तान पिर
5	Consonant + Consonant + Vowel	CCV	ल्या
6	Consonant + Consonant + Vowel + Consonant	CCVC	ख्याल
7	Consonant + Consonant + Consonant + Vowel	CCCV	स्त्री
8	Consonant + Consonant + Consonant + Vowel + Consonant	CCCVC	ब्ल्याङ

Table 2.10: Structure of Nepali syllable

Chapter 3

How human hearing works

Ears are among the extraordinary organs in human body. It is the organ that picks up all the sound and translates into signals that brain can interpret and understand. Among many different peculiarities of the ear, the most interesting fact is that it is complete mechanical as compared to other human sensory organs. Our sense of smell, taste and vision all involve chemical reactions, but Our hearing system is based solely on physical movement.

To understand how human hearing works we need to understand how sound works. An object produces sound when it vibrates in matter. This could be a solid, such as earth; a liquid, such as water; or a gas, such as air. Most of the time, we hear sounds traveling through the air in our atmosphere. When something vibrates in the atmosphere, it moves the air particles around it. Those air particles in turn move the air particles around them, carrying the pulse of the vibration through the air. To further so how sound works, let's look at a simple vibrating object, a bell. When we hit a bell, the metal vibrates, it flexes in and out. When it flexes it creates a pressure difference on each flex resulting in either compression or rarefaction. In this way, a vibrating object sends a wave of pressure fluctuation through the atmosphere. We hear different sounds from different vibrating objects because of variations in the sound wave frequency. A higher wave frequency simply means that the air pressure fluctuation switches back and forth more quickly. We hear this as a higher pitch. When there are fewer fluctuations in a period of time, the pitch is lower. The level of air pressure in each fluctuation, the wave's amplitude, determines how loud the sound is. As we know that sound travels through the air as a result of vibration in air pressure, our ear need to do the following three steps to hear the sound.

1. Direct the sound waves into the hearing part of the ear
2. Sense the fluctuations in air pressure

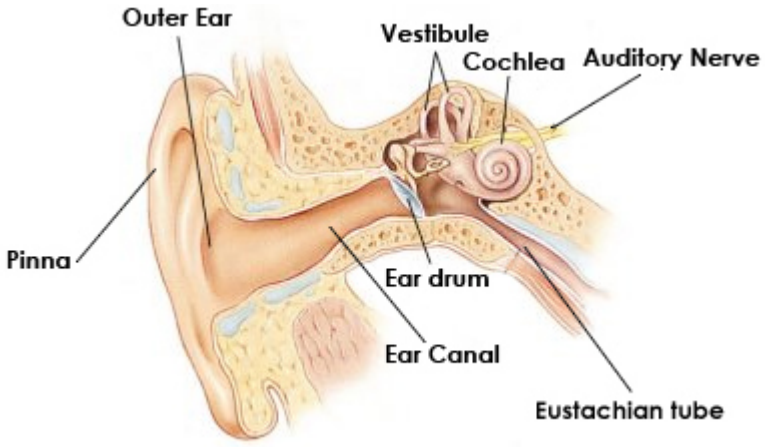

Figure 3.1: Ear Diagram

3. Translate these fluctuations into an electrical signal that the brain can understand

The pinna, the outer part of the ear, serves to 'catch' the sound waves. Our outer ear is pointed forward and it has a number of curves. This structure helps us to determine the direction of a sound. If a sound is coming from above, it will bounce off the pinna in a different way than if it is coming from in front of us or below of us. This very sound reflection alters the pattern of the sound wave. Our brain recognises distinctive patterns and determines whether the sound is in front of us, behind us, above us or below us.

Once the sound waves travel into the ear canal, they vibrate the tympanic membrane, commonly called the eardrum. The eardrum is a thin, cone-shaped piece of skin, about 10 mm wide. It is positioned between the ear canal and the middle ear. The middle ear is connected to the throat via the Eustachian tube. Since air from the atmosphere flows in from our outer ear as will as our mouth, the air pressure on both sides of the eardrum remains equal. This pressure balance lets our eardrum freely move backward and front. The eardrum is rigid and very sensitive; even the slightest air-pressure fluctuations will move it back and forth. It is attached to the tensor tympani muscle, which constantly pulls it inward. This keeps the entire membrane taut so it will vibrate no matter which part of it is hit by a sound wave. This scriptsize flap of skin acts just like the diaphragm in a microphone. The compressions and rarefactions of sound waves push the drum back and forth. Higher-pitch sound waves move the drum more rapidly, and louder sound moves the drum a greater distance. The eardrum is the entire sensory element

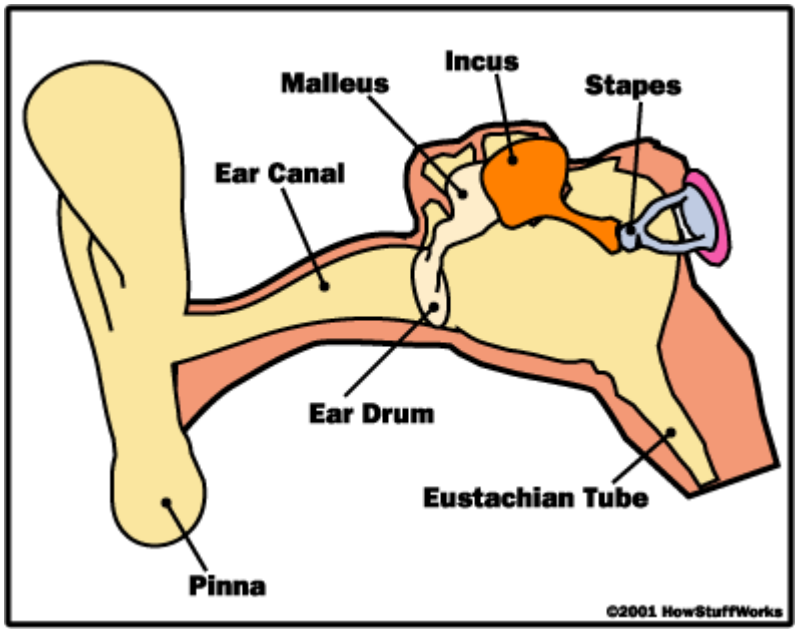

Figure 3.2: Sound waves vibrates the eardrum which stimulates the middle ear

in your ear. The rest of the ear serves only to pass along the information gathered at the eardrum.

We know that the compressions and rarefactions in sound waves move our eardrum back and forth. For the most part, these changes in air pressure are extremely small. They don't apply much force on the eardrum, but the eardrum is so sensitive that this minimal force moves it a good distance. The cochlea in the inner ear conducts sound through a fluid, instead of through air. This fluid has a much higher inertia than air – that is, it is harder to move (think of pushing air versus pushing water). The small force felt at the eardrum is not strong enough to move this fluid. Before the sound passes on to the inner ear, the total pressure (force per unit of area) must be amplified. This is the job of the ossicles, a group of scriptsize bones in the middle ear. This amplification system is extremely effective. The pressure applied to the cochlear fluid is about 22 times the pressure felt at the eardrum. This pressure amplification is enough to pass the sound information on to the inner ear, where it is translated into nerve impulses the brain can understand.

The cochlea is by far the most complex part of the ear. Its job is to take the physical vibrations caused by the sound wave and translate them into electrical information the brain can recognize as distinct sound. The

cochlea structure consists of three adjacent tubes separated from each other by sensitive membranes. In reality, these tubes are coiled in the shape of a snail shell, but it's easier to understand what's going on if we imagine them stretched out. It's also clearer if we treat two of the tubes, the scala vestibuli and the scala media, as one chamber. The membrane between these tubes is so thin that sound waves travel as if the tubes weren't separated at all.

The stapes moves back and forth, creating pressure waves in the entire cochlea. The round window membrane separating the cochlea from the middle ear gives the fluid somewhere to go. It moves out when the stapes pushes in and moves in when the stapes pulls out.

The middle membrane, the basilar membrane, is a rigid surface that extends across the length of the cochlea. When the stapes moves in and out, it pushes and pulls on the part of the basilar membrane just below the oval window. This force starts a wave moving along the surface of the membrane. The wave travels something like ripples along the surface of a pond, moving from the oval window down to the other end of the cochlea.

The basilar membrane has a peculiar structure. It's made of 20,000 to 30,000 reed-like fibers that extend across the width of the cochlea. Near the oval window, the fibers are short and stiff. As you move toward the other end of the tubes, the fibers get longer and more limber. This gives the fibers different resonant frequencies. A specific wave frequency will resonate perfectly with the fibers at a certain point, causing them to vibrate rapidly. This is the same principle that makes tuning forks and kazoos work, a specific pitch will start a tuning fork ringing, and humming in a certain way will cause a kazoo reed to vibrate.

The organ of corti is a structure containing thousands of scriptsize hair cells. It lies on the surface of the basilar membrane and extends across the length of the cochlea. When these hair cells are moved, they send an electrical impulse through the cochlear nerve. The cochlear nerve sends these impulses on to the cerebral cortex, where the brain interprets them. The brain determines the pitch of the sound based on the position of the cells sending electrical impulses. Louder sounds release more energy at the resonant point along the membrane and so move a greater number of hair cells in that area. The brain knows a sound is louder because more hair cells are activated in an area.

The cochlea only sends raw data, complex patterns of electrical impulses. The brain is like a central computer, taking this input and making some sense of it all. This is an extraordinarily complex operation, and scientists are still a long way from understanding everything about it.

In fact, hearing in general is still very mysterious. The basic concepts at work in human and animal ears are fairly simple, but the specific structures

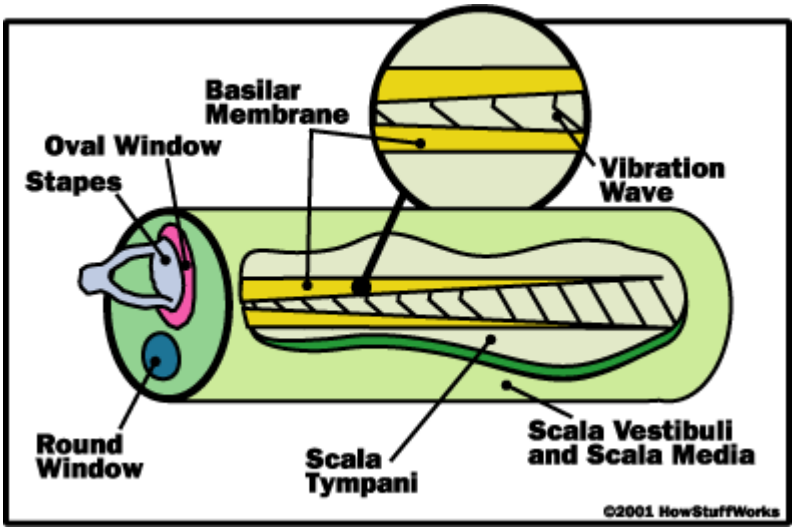

Figure 3.3: The piston action of the stapes moves the fluid in the cochlea

are extremely complex. Scientists are making rapid advancements, however, and they discover new hearing elements every year. It's astonishing how much is involved in the hearing process, and it's even more amazing that all these processes take place in such a small area of the body.

Chapter 4

Speech Recognition

4.1 Overview

Speech recognition is the process of converting an acoustic signal, captured by a microphone or a telephone, to a set of words. The recognized words can be the final results, as for applications such as commands and control, data entry, and document preparation. They can also serve as the input to further linguistic processing in order to achieve speech understanding.

Speech recognition systems can be characterized by many parameters, some of the more important of which are shown in Figure . An isolated-word speech recognition system requires that the speaker pause briefly between words, whereas a continuous speech recognition system does not. Spontaneous, or extemporaneously generated, speech contains disfluencies, and is much more difficult to recognize than speech read from script. Some systems require speaker enrollment, a user must provide samples of his or her speech before using them, whereas other systems are said to be speaker-independent, in that no enrollment is necessary. Some of the other parameters depend on the specific task. Recognition is generally more difficult when vocabularies are large or have many similar-sounding words. When speech is produced in a sequence of words, language models or artificial grammars are used to restrict the combination of words.

Speech recognition is a difficult problem, largely because of the many sources of variability associated with the signal. First, the acoustic realizations of phonemes, the smallest sound units of which words are composed, are highly dependent on the context in which they appear. These phonetic variabilities are exemplified by the acoustic differences of the phoneme /t/ in *two*, *true*, and *butter* in American English. At word boundaries, contextual variations can be quite dramatic—making *gas shortage* sound like

Parameters	Range
Speaking Mode	Isolated words to continuous speech
Speaking Style	Read speech to spontaneous speech
Enrollment	Speaker-dependent to Speaker-independent
Vocabulary	Small (20 words) to large (20000 words)
Language Model	Finite-state to context-sensitive
Sound to Noise Ratio	High (30 dB) to low (10 dB)

Table 4.1: Typical parameters used to characterize the capability of speech recognition systems

gash shortage in American English, and devo andare sound like devandare in Italian.

Second, acoustic variabilities can result from changes in the environment as well as in the position and characteristics of the transducer. Third, within-speaker variabilities can result from changes in the speaker's physical and emotional state, speaking rate, or voice quality. Finally, differences in socio linguistic background, dialect, and vocal tract size and shape can contribute to across-speaker variabilities.

4.2 Different types of Speech Recognition approach

4.2.1 Statistical vs. Neural Network Approach

- In statistical recogniser, the pattern matcher
  - uses as explicit mathematical model of speech to calculate the probablity that a speech signal corresponds to a particular sequence of words. The mathematical model incorporates knowledge about the nature of the speech
  - select the sequence of words that yields the highest probability
- In neural network recogniser, the pattern matcher
  - has a fewer assumptions about the nature of the speech because there is no explicit mathematical model
  - Usually needs many more parameters than a statistical recogniser

- Both types use an iterative algorithm to adjust their parameters values to match training samples for speech whose corresponding word sequences are known.
- Statistical recognisers have generally been much more successful

4.2.2 Isolated vs. Continuous Speech Recognition

Interpreting isolated-words speech, in which the speaker pauses between each word is easier than interpreting continuous speech. This is because boundary effects cause words to be pronounced differently in different contexts. For example, the spoken phrase "could you" contains a j sound and despite that it contains two words there is no empty space between then in the speech wave. The ability to recognise continuous speech is very important, however since human have difficulties in speaking isolated words it is one of the most widely used form of speech.

4.2.3 Small vocabulary vs. Large vocabulary Recognition

Recognizing utterances that are confined to small vocabulary (20 words) is easier than working with large vocabulary (20,000 words). A small vocabulary helps to limit the number of word candidates for the given speech segment.

4.2.4 Real time versus Offline Speech Recognition

Highly interactive application requires that a sentence be translated into text as it is being spoken, while in other situation it is permissible to spend minutes in computation. Real time speeds are hard to achieve, especially when higher-level knowledge is involved.

4.2.5 Broad vs. narrow grammar Speech Recognition

An example of a narrow grammar is one for the phone numbers: S -¿ XXXX-XXXX, where X is any number between 0 and 9. Syntactic and semantic constants for unrestricted English are much harder to represent. The narrower the grammar is, the smaller the search space for recognition will be.

4.2.6 Speaker Independent vs. Speaker Dependent Recognition

A speaker-independent system can listen to any speaker and translate the sounds into the corresponding written text. Speaker independence is hard to achieve because of the wide variations in pitch and accent. It is easier to build a speaker-recognition system that is speaker-dependent system, which can be trained on the voice patters of a single speaker. The system will work for that one speaker. It can be trained again on another voice, but then it will no longer work for the original speaker.

Chapter 5

Existing methods

5.1 Hidden Markov model (HMM)-based speech recognition

Modern general-purpose speech recognition systems are generally based on Hams. These are statistical models which output a sequence of symbols or quantities. One possible reason why Hams are used in speech recognition is that a speech signal could be viewed as a piecewise stationary signal or a short-time stationary signal. That is, one could assume in a short-time in the range of 10 milliseconds, speech could be approximated as a stationary process. Speech could thus be thought of as a Markov model for many stochastic processes.

Another reason why Hams are popular is because they can be trained automatically and are simple and computationally feasible to use. In speech recognition, the hidden Markov model would output a sequence of n-dimensional real-valued vectors (with n being a small integer, such as 10), outputting one of these every 10 milliseconds. The vectors would consist of campestral coefficients, which are obtained by taking a Fourier transform of a short time window of speech and correlating the spectrum using a cosine transform, then taking the first (most significant) coefficients. The hidden Markov model will tend to have in each state a statistical distribution that is a mixture of diagonal covariance Gaussians which will give a likelihood for each observed vector. Each word, or (for more general speech recognition systems), each phoneme, will have a different output distribution; a hidden Markov model for a sequence of words or phonemes is made by concatenating the individual trained hidden Markov models for the separate words and phonemes.

Described above are the core elements of the most common, HMM-based approach to speech recognition. Modern speech recognition systems use var-

ious combinations of a number of standard techniques in order to improve results over the basic approach described above. A typical large-vocabulary system would need context dependency for the phonemes (so phonemes with different left and right context have different realizations as HMM states); it would use campestral normalization to normalize for different speaker and recording conditions; for further speaker normalization it might use vocal tract length normalization (VTLN) for male-female normalization and maximum likelihood linear regression (MLLR) for more general speaker adaptation. The features would have so-called delta and delta-delta coefficients to capture speech dynamics and in addition might use heteroscedastic linear discriminant analysis (HLDA)

Decoding of the speech (the term for what happens when the system is presented with a new utterance and must compute the most likely source sentence) would probably use the Viterbi algorithm to find the best path.

5.2 Neural Network-based Speech Recognition

Another approach in acoustic modeling is the use of neural networks. They are capable of solving much more complicated recognition tasks, but do not scale as well as Hams when it comes to large vocabularies. Rather than being used in general-purpose speech recognition applications they can handle low quality, noisy data and speaker independence. Such systems can achieve greater accuracy than HMM based systems, as long as there is training data and the vocabulary is limited. A more general approach using neural networks is phoneme recognition. This is an active field of research, but generally the results are better than for Hams. There are also NN-HMM hybrid systems that use the neural network part for phoneme recognition and the hidden markov model part for language modeling.

5.3 Dynamic time warping (DTW)-based speech recognition

Dynamic time warping is an approach that was historically used for speech recognition but has now largely been displaced by the more successful HMM-based approach. Dynamic time warping is an algorithm for measuring similarity between two sequences which may vary in time or speed. For instance, similarities in walking patterns would be detected, even if in one video the

person was walking slowly and if in another they were walking more quickly, or even if there were accelerations and decelerations during the course of one observation. DTW has been applied to video, audio, and graphics – indeed, any data which can be turned into a linear representation can be analyzed with DTW.

A well known application has been automatic speech recognition, to cope with different speaking speeds. In general, it is a method that allows a computer to find an optimal match between two given sequences (e.g. time series) with certain restrictions, i.e. the sequences are "warped" non-linearly to match each other. This sequence alignment method is often used in the context of hidden Markov models.

5.4 Applications

5.4.1 Health care

In the health care domain, even in the wake of improving speech recognition technologies, medical transcriptionists (MTs) have not yet become obsolete. Many experts in the field anticipate that with increased use of speech recognition technology, the services provided may be redistributed rather than replaced. Speech recognition has not yet made the skills of MTs obsolete.

Speech recognition can be implemented in front-end or back-end of the medical documentation process.

Front-End SR is where the provider dictates into a speech-recognition engine, the recognized words are displayed right after they are spoken, and the dictator is responsible for editing and signing off on the document. It never goes through an MT/editor.

Back-End SR or Deferred SR is where the provider dictates into a digital dictation system, and the voice is routed through a speech-recognition machine and the recognized draft document is routed along with the original voice file to the MT/editor, who edits the draft and finalizes the report. Deferred SR is being widely used in the industry currently.

Many Electronic Medical Records (EMR) applications can be more effective and may be preformed more easily when deployed in conjunction with a speech-recognition engine. Searches, queries, and form filling may all be faster to perform by voice than by using a keyboard.

5.4.2 Training air traffic controllers

Training for military (or civilian) air traffic controllers (ATC) represents an excellent application for speech recognition systems. Many ATC training systems currently require a person to act as a "pseudo-pilot", engaging in a voice dialog with the trainee controller, which simulates the dialog which the controller would have to conduct with pilots in a real ATC situation. Speech recognition and synthesis techniques offer the potential to eliminate the need for a person to act as pseudo-pilot, thus reducing training and support personnel. Air controller tasks are also characterized by highly structured speech as the primary output of the controller, hence reducing the difficulty of the speech recognition task. The USAF, USMC, US Army, and FAA are currently using ATC simulators with speech recognition provided by Adacel Systems Inc (ASI). Adacel's MaxSim software uses speech recognition and synthetic speech to enable the trainee to control aircraft and ground vehicles in the simulation without the need for pseudo pilots. Adacel's ATC In A Box Software provided a synthetic ATC environment for flight simulators. The "real" pilot talks to a virtual controller using speech recognition and the virtual controller responds with synthetic speech.

5.4.3 Telephony and other domains

ASR in the field of telephony is now commonplace and in the field of computer gaming and simulation is becoming more widespread. Despite the high level of integration with word processing in general personal computing, however, ASR in the field of document production has not seen the expected increases in use.

5.4.4 Disabled people

Disabled people are another part of the population that can use speech recognition programs. It is especially useful for people who can't use their hands, from the profoundly physically impaired to people with mild repetitive stress injuries. In fact, people who used the keyboard a lot and developed RSI became an urgent early market for speech recognition.

5.4.5 Further applications

- Automatic translation
- Automotive speech recognition (e.g., Ford Sync)

- Court reporting (Real time Voice Writing)
- Speech Biometric Recognition
- Hands-free computing: voice command recognition computer user interface
- Home automation
- Interactive voice response
- Medical transcription
- Mobile telephony, including mobile email.
- Pronunciation evaluation in computer-aided language learning applications
- Robotics
- Transcription (digital speech-to-text).

5.5 Speech Recognition Software

5.5.1 Free Software

- XVoice
- CVoiceControl
- Open Mind Speech
- GVoice
- ISIP
- CMU Sphinx
- Ears
- NICO ANN Toolkit
- Myers' Hidden Markov Model Software

5.5.2 Commercial Software

- IBM ViaVoice
- Vocalis Speechware
- Babel Technologies
- Nuance
- Abbot/AbbotDemo

Besides these there are other commercial products like Entropic, SpeechWorks etc

Chapter 6

Implementation using conventional methods

As explained earlier there exists three different methods for speech recogntion namely, HMM based network, Neural Network based and Dynamic Time Warping based. Among these the most widely used method is the HMM based with template matching. In this project as an initial approach we has planned the template based matching using HMM. The strategy for this approach is as explained below.

6.1 Strategy

6.1.1 Dynamic time warping

One of the earliest approaches to isolated word speech recognition was to store a prototypical version of each word (called a template) in the vocabulary and compare incoming speech with each word, taking the closest match. This presents two problems: what form do the templates take and how are they compared to incoming signals. The simplest form for a template is a sequence of feature vectors – that is the same form as the incoming speech. We will assume this kind of template for the remainder of this discussion. The template is a single utterance of the word selected to be typical by some process; for example, by choosing the template which best matches a cohort of training utterances. Comparing the template with incoming speech might be achieved via a pairwise comparison of the feature vectors in each. The total distance between the sequences would be the sum or the mean of the individual distances between feature vectors. The problem with this approach is that if a constant window spacing is used, the lengths of the input

and stored sequences is unlikely to be the same. Moreover, within a word, there will be variation in the length of individual phonemes. The matching process needs to compensate for length differences and take account of the non-linear nature of the length differences within the words.

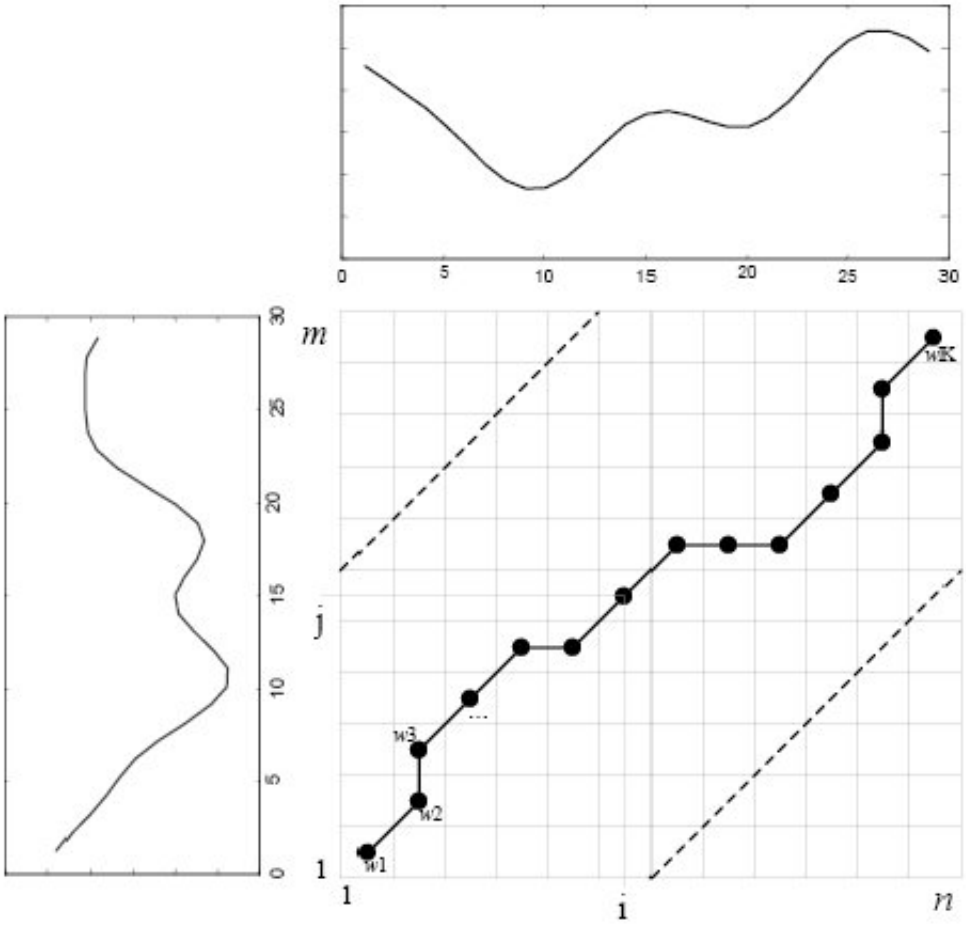

Figure 6.1: Dynamic Warping Path

The Dynamic Time Warping algorithm achieves this goal; it finds an optimal match between two sequences of feature vectors which allows for stretched and compressed sections of the sequence. Suppose we have two

time series Q and C, of length n and m respectively, where:

$$Q = \{q_1, q_2, \ldots, q_i, \ldots, q_n\} \tag{6.1}$$

$$C = \{c_1, c_2, \ldots, c_j, \ldots, c_m \Psi\} \tag{6.2}$$

To align two sequences using DTW we construct an n-by-m matrix where the (ith, jth) element of the matrix contains the distance d(q_i,c_j) between the two points q_i and c_j (Typically the Euclidean distance is used, so $d(q_i, c_j) = (q_i - c_j)^2$). Each matrix element (i,j) corresponds to the alignment between the points q_i and c_j. Warping path W, is a contiguous (in the sense stated below) set of matrix elements that defines a mapping between Q and C. The k^th element of W is defined as $w^k = (i, j)k$ so we have:

$$W = w_1, w_2, \ldots, w_k, \ldots, w_K \tag{6.3}$$

The warping path is typically subject to several constraints.

Boundary Conditions

$w_1 = (1, 1)$ and $w_K = (m, n)$, simply stated, this requires the warping path to start and finish in diagonally opposite corner cells of the matrix.

Continuity

Given $w_k = (a, b)$ then $w_k - 1 = (a', b')$ where a–$a' £ 1$ and $b - b' £ 1$. This restricts the allowable steps in the warping path to adjacent cells (including diagonally adjacent cells).

Monotonicity

Given$wk = (a, b)$ then $wk - 1 = (a', b')$ where a–$a^3 0$ and $b - b^3 0$. This forces the points in W to be monotonically spaced in time. There are exponentially many warping paths that satisfy the above conditions, however we are interested only in the path which minimizes the warping cost:

$$DTW(Q, C) = min\{\sum_{k=1}^{k} w_k / K\} \tag{6.4}$$

The K in the denominator is used to compensate for the fact that warping paths may have different lengths. This path can be found very efficiently using dynamic programming to evaluate the following recurrence which defines the cumulative distance g(i,j) as the distance d(i,j) found in the current cell and the minimum of the cumulative distances of the adjacent elements:

$$g(i, j) = d(qi, cj) + min\{g(i - 1, j - 1), g(i - 1, j), g(i, j - 1)\} \tag{6.5}$$

6.1.2 Feature extraction

As we were dealing with noisy data, the features needed to be robust to the ambient noise in the signal. With this reason in mind we used 13 mel frequency cepstral coefficients and 12 delta cepstral coefficients. Apart from this one major thing that was done to enhance the final recognition was implementation of spectral subtraction and endpoint detection. The various feature extraction techniques are discussed below:

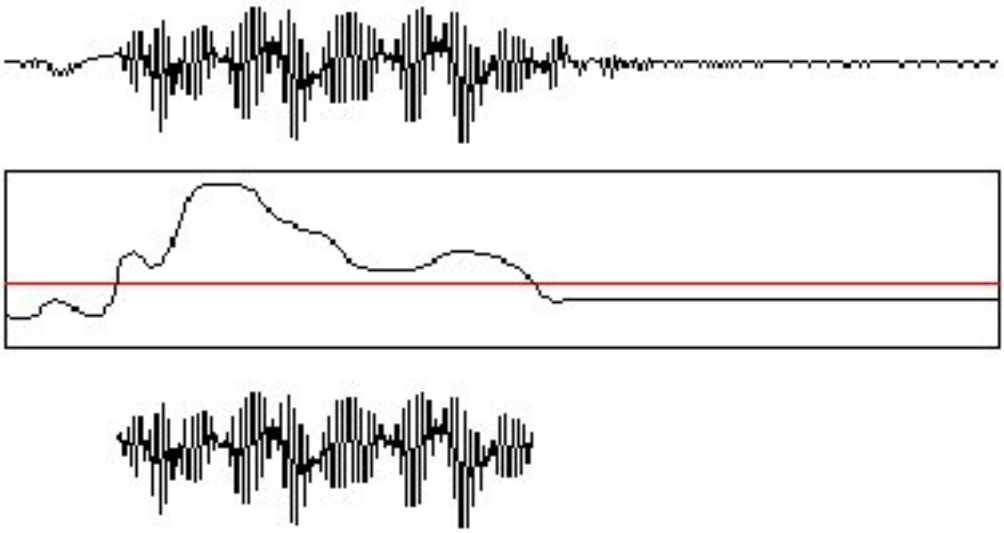

Figure 6.2: Feature Extraction

Blocking into frames

Section of N(256) consecutive speech samples are used as a single frame. Consecutive frames are spaced M(128) samples apart. Thus achieving a 50% overlap between adjacent windows.

$$X_{\lambda}(n) = \tilde{A}(M * l + n), \Psi 0 <= n <= N - 1 and 0 <= l <= L - 1 \qquad (6.6)$$

N = Total No. of samples in a frame.
M = Total No. of sample spacing between the frames. [measure of overlap]
L = Total number of frames.

Spectral subtraction and endpoint detection:

The whole speech signal was first passed to this routine which would do a endpoint detection depending whether the speech was 2.5 times greater than the threshold. This threshold was the average energy in the first four and last four frames of the signal which is assumed to be void of the command word. The threshold is applied from the beginning of the frame and from the end of the frame thus giving us two frame numbers in between which the whole utterance is present.

We calculate the energy per frame first. ie.

$$P[l \ldots m] = Sum\ k = \ldots j(s[k]^2) \tag{6.7}$$

where s[k] are the speech data in the frame. Similarly P is calculated for all the frames and an average is taken for the final noise value[E].

$$E = [Sum\ k = l \ldots m(p[k]^2)]/m \tag{6.8}$$

The threshold is set at (constant$_*$ E), as the detecting criterion. After endpoint detection of the speech signal using above two frame numbers the command word is picked up from the whole signal. The average noise energy and this truncated signal is passed to the spectral subtraction routine. This calculates a 512 point FFT per frame each 256 long and does spectral subtraction using the noise FFT directly in the frequency domain.

$$X(f) = Y(f) * Hss(f) \tag{6.9}$$

$$Hss(f) = \sqrt{(1 - 1/SNR(f))} \tag{6.10}$$

$$SNR(f) = |Y(f)|^2/|N(f)|^2 \tag{6.11}$$

where Y(f) = 512 point FFT per frame of the signal, N(f) = 512 point FFT of the noise frame passed.

6.1.3 The Problem

- Noisy computer room with background noise
- Weak fricatives: /f, th, h/
- Weak plosive bursts: /p, t, k/
- Final nasals
- Voiced fricatives becoming devoiced

- Trailing off of sounds (ex: binary, three)
- Simple, efficient processing
- Avoid hardware costs

6.1.4 The Solution

- Two measurements
- Energy
- Zero crossing rate
- Simple, fast, accurate

6.1.5 Time-Frequency relation

There exists a pretty deep relation between the time and frequency domain. Each of it is highly interdependent upon other. The relation that exists between them is as follows.

1. Any multiplication done in the time domain is equivalent to convolution in the frequency domain
2. Similarly, any convolution done in the time domain is equivalent to multiplication in the frequency domain.

In speech analysis, increasing the time resolution means reducing the frequency resolution, while increasing the frequency resolution means reducing time resolution. Hence, there exists a trade-off between the two.

6.1.6 Mel-Frequency Cepstral Coefficient

Mel Frequency Cepstral Coefficients (MFCCs) are coefficients that represent audio. They are derived from a type of cepstral representation of the audio clip (a "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are positioned logarithmically (on the mel scale) which approximates the human auditory system's response more closely than the linearly-spaced frequency bands obtained directly from the FFT or DCT. This can allow for better processing of data, for example, in audio compression.

MFCCs are commonly derived as follows:

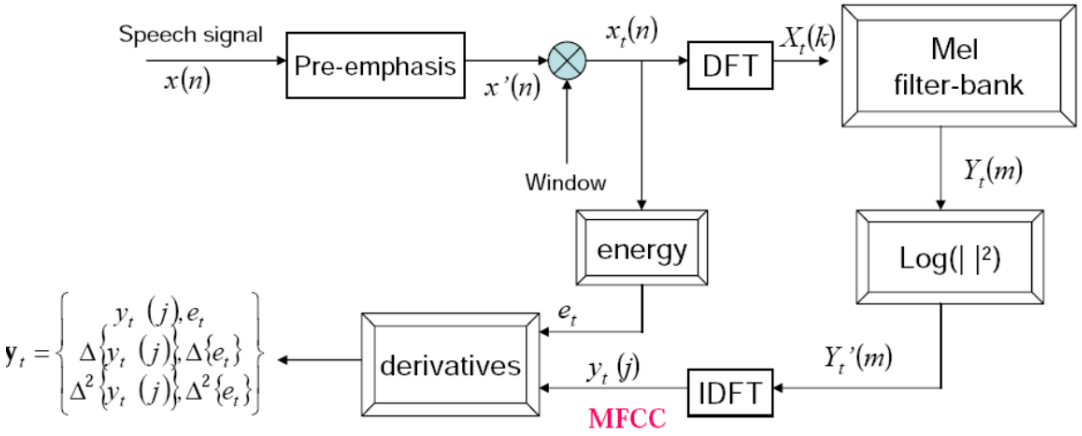

Figure 6.3: Mel Frequency Cepstral Coefficients extraction

1. Take the Fourier transform of (a windowed excerpt of) a signal
2. Map the log amplitudes of the spectrum obtained above onto the mel scale, using triangular overlapping windows.
3. Take the Discrete Cosine Transform of the list of mel log-amplitudes, as if it were a signal.
4. The MFCCs are the amplitudes of the resulting spectrum.

There can be variations on this process – e.g. differences in the mel scale conversion A detailed view of this can be seen in the figure below which illustrates the functioning and process of deriving the Mel Frequency Cepstral Coefficients.

As seen from the figure the important part of finding the MFCC are as follows:

- Taking the sample as input
- Applying the window functioning
- Taking the DFT
- Generating the Mel Filter bank
- Computing the Log energy
- Finding the IDFT

- Finally taking the derivatives

Now we look at each of these steps in detail :

6.1.7 Windowing

A window function (or apodization function) is a function that is zero-valued outside of some chosen interval. For instance, a function that is constant inside the interval and zero elsewhere is called a rectangular window, which describes the shape of its graphical representation. When another function or a signal (data) is multiplied by a window function, the product is also zero-valued outside the interval: all that is left is the "view" through the window. Applications of window functions include spectral analysis, filter design and beamforming. In speech analysis each of these windows are also known as frames. To extract the maximum features from the sample data we use successive overlapping frames. When the length of a data set to be transformed is larger than necessary to provide the desired frequency resolution, a common practice is to subdivide it into smaller sets and window them individually. To mitigate the "loss" at the edges of the window, the individual sets may overlap in time. One of the important question is why to divide speech signal into successive overlapping frames. Since speech is not a stationary signal and we prefer not to represent the entire signal by just one Fourier transform. Hence we divide the speech into frames based on two different attibutes of frame, namely

1. **Frame size:** the length of time over which the number is valid. Typically, $10ms$ $25ms$
2. **Frame shift:** the length of time between successive frame. Typically, $5ms$ $10ms$

This is system of framing of speech is more evident from the figure below which shows the overlapping successive framing of a speech signal.

Two different kind of windowing commonly used in speech analysis and processing are as follows.

Rectangular Window

$$w[n] = \begin{cases} 1 & 0 \leqslant n \leqslant L-1 \\ 0 & \text{otherwise} \end{cases}$$

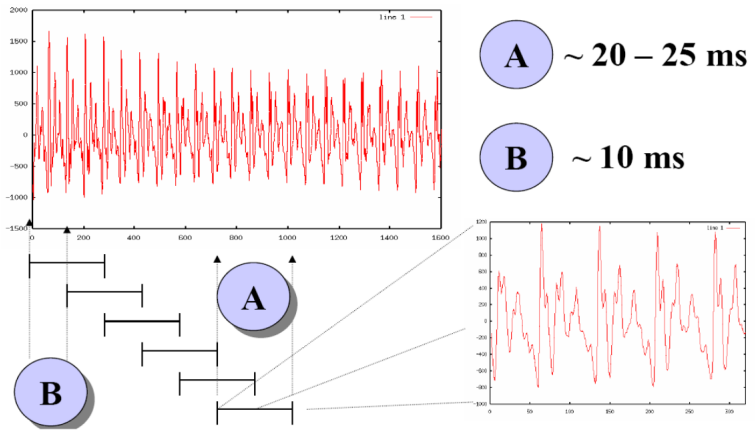

Figure 6.4: Windowing procedure

Hamming Window

$$w[n] = \begin{cases} 0.54 - 0.46 \cos \dfrac{2\pi n}{L-1} & 0 \leqslant n \leqslant L-1 \\ 0 & \text{otherwise} \end{cases}$$

6.1.8 Discrete Fourier Transform

The Fourier transform of can be computed from the z-transform as:

$$X(\omega) = X(z)|_{z=e^{j\omega}} - \sum_{n=-\infty}^{\infty} x(n)e^{j\omega n} \tag{6.12}$$

The Fourier transform may be viewed as the -transform evaluated around the unit circle.

The Discrete Fourier Transform (DFT) is defined as a sampled version of the Fourier shown above:

$$x(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}, k = 0, 1, \ldots, N-1 \tag{6.13}$$

The inverse DFT is given by:

$$x(n) = \sum_{n=0}^{N-1} X(k)e^{2\pi kn/N}, n = 0, 1, \ldots, N-1 \tag{6.14}$$

The Fast Fourier Transform (FFT) is simply an efficient computation of the DFT.

The Discrete Cosine Transform (DCT) is simply a DFT of a signal that is assumed to be real and even (real and even signals have real spectra):

$$X(k) = x(0) + 2\sum_{n=1}^{M-1} x(n)cos2\pi kn/N, k = 0, 1, \ldots, N-1 \tag{6.15}$$

Note that these are not the only transforms used in speech processing (wavelets, fractals, Wigner distributions, etc.). In general, we choose N=512 or 1024 for getting the required features for the analysis

6.1.9 Mel-scale

The mel scale is a perceptual scale of pitches judged by listeners to be equal in distance from one another. The reference point between this scale and normal frequency measurement is defined by equating a 1000 Hz tone, 40 dB above the listener's threshold, with a pitch of 1000 mels. Above about 500 Hz, larger and larger intervals are judged by listeners to produce equal pitch increments. As a result, four octaves on the hertz scale above 500 Hz are judged to comprise about two octaves on the mel scale. The name mel comes from the word melody to indicate that the scale is based on pitch comparisons. Mel-scale is approximate linear below 1 kHz and logarithmic above 1 kHz .

6.1.10 Log energy computation

Log energy computation is the computation of the logarithm of the square magnitude of the output of Mel-filter bank. We use the log energy instead of the linear energy or phase information because of the following points: Phase information is not very helpful in speech. The logarithm compresses the dynamic range of values, which is a characteristic of human hearing system. The logarithm also makes feature extraction less sensitive to variations of input. Can be used to remove convolution channel noise.

6.1.11 Mel Frequency cepstrum

The next step now after calculating the log energy is to calculate the inverse DFT of the log energy. This is calculated as follows:

$$y_t[k] = \sum_{m=1}^{M} log(|Y_t(m)|)cos(k(m-0.5)\frac{\pi}{M}), k = 0, 1, \ldots, J \tag{6.16}$$

Since the log power spectrum is real and symmetric, inverse DFT reduces to a Discrete Cosine Transform (DCT)

6.1.12 Typical MFCC features

Mentioned below certain features of the MFCC. They basically give us the window size window hopping and many other similar information.

- Window size: 25ms
- Window hopping: 10ms
- Pre-emphasis coefficient: 0.97
- 12 MFCC coefficients + energy
- delta and double-delta
- Total 39-dimensional features

6.2 Implementation and Results

The project started with the experimentation of two models for recognition of phoneme in the Nepali speech in Matlab. Both the approaches were based on the template matching with a standard library established by sampling a set of normalized and equilength sample data. The methods use two basic procedure of feature extraction for the given sample of sound, namely, direct FFT, and Mel Frequency Cepstral Co-efficient. To extract the features, each voice sample is first divided into equal length frames to form a uniform window of data. The frames ware then overlapped and features are extracted from the data using the two methods explained above. The purpose of overlapping, as explained above, is to keep the feature of the speech constant over the sampling period. This ensured that features extracted re accurate as far as possible.

The data represented in Apendix A depicts the results of experiment done with overlapping windows using Mel Frequency Cepstral Co-efficient and Direct FFT. The five set of data presented in theApendix represent experiment with different window size and no. of windows. In this attempt, as evident from the results shown below we can see that the maximum effciency is just above 67% at 67.3327, with samples of 48 data points per frame/window. As explained earlier, to maximize our effieciency we have used windowng with 50% overlap in each window. Furthermore, each window has been windowed

S.No.	Methods	Window Size (Samples/frame)	No. of Windows	Accuracy
1	MFCC	64	12	0.656327
2	MFCC	48	12	0.628571
3	MFCC	32	14	0.632653
4	FFT	48	12	0.673010
5	FFT	64	12	0.657595

Table 6.1: Comparision table of different methods

using hamming function to smoothen and make the end point continuous. Looking at the data we can determine that changing the window size and number of windows does change the accuracy of the overall system. The tables in the appendix A shows a detail view of data for the different windows size for both the approaches. The table 6.1 shows a summary view of the data for both the approaches. As seen from the table that the second method, the direct FFT method seems to have better accuracy as compared to the MFCC. Though there is not much of difference between the two methods the fact that FFT has a much lighter calculation and is more resistant to changes in the voice and of the speaker makes FFT a much beter option for realtime soud recognition applications. But nevertheless, MFCC as a speech recognition tool in the background of speech recongition can definitely not be rejeccted on the basis of the experimentation carried out by us as its accuracy is almost same as FFT.

Even though the experimentation was done in the MATLAB, the code for the upper mentioned method was also coded in C and python for further trials. But even those trials didn't yield any encouraging result. As we can see that, despite the fact that the experimentation is done in a very narrow domain with a single speakers data and sample input, the result is not encouraging enough. Even in such narrow domain of the speech recognition, the maximum accuracy attained is below 70% which is not the best result. Hence as an alternative to these methods we have propose a completely new and a radical approach to mimic human hearing as far as possible. This model, which we have named as 'Ear Model', is an effort to mimic the basic process of human hearing. This model to some extent simulates a virtual ear working in conjuction with a virtual mind. The detail of this new model is present in the next section.

Chapter 7

Ear Model

7.1 Overview

From previous section, we have come to know that human ear can be considered a complex natural Fourier analyzer. Depending upon the size of the hair, each of them act as detector of a particular frequency in the audible frequency spectrum. The information collected by each of these hair is transmitted to brain, which (parallel) processes the information for its recognition. The Ear Model tries to simulate the same procedure.

7.2 Hairs in cochlea as objects

In the process of speech recognition by the brain, each hair in the cochlea are responsible for detecting the vibration at different frequencies. For each and every frequencies in the audible range, objects are created which pick up pattern of amplitude for that particular frequency it is tuned to. The object then checks the pattern with previously trained data and gives out the recognized character/word as decision.

These hair-objects have the following properties:

7.2.1 Frequency

Just like each and every hair inside cochlea have their own fundamental frequency depending upon the length of the hair, the object-models of the hair also have their characteristics frequencies which determine which frequency it is tuned to.

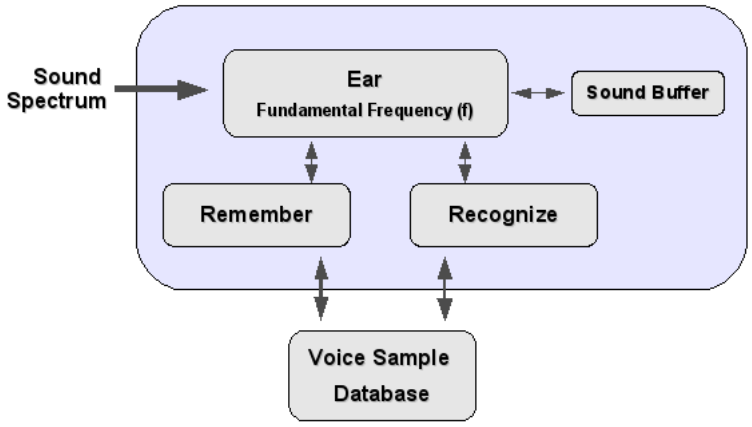

Figure 7.1: Object Representing fiber in cochlea

7.2.2 Memory

The hair-objects also have individual memory of their own, which is used both for training and recognition purposes. During the training phase, each object stores the pattern of the amplitude of the frequency it is tuned to. The object also can store multiple data for same syllable and also for different persons. This dataset is used for comparison during the recognition mode.

7.3 Recognition

Objects tuned to different frequencies can have varied recognition of the same syllable provided to it. So a central recognition system is required to analyze the outputs of the objects and give a concrete decision based on the output pattern and past results. Most of the speech related features are concentrated in the lower frequency range, so the recognition system also takes account of this.

7.4 Efficiency

The efficiency of this method is dramatically increased from other methods as there are more parameters available to make decision about the recognition.

- Recognition algorithm is applied to each and every frequency of the human audible range, and for each frequency a decision about recogni-

tion is made. In traditional methods, the spectrogram is considered as a whole which gives only one decision on recognition of the syllable.

- Analyzing the speech input frequency by frequency gives the recognition algorithm an opportunity to focus on pattern matching of just a single dimension of data. This is vital in terms of recognition of syllable, as the pattern of data for a single frequency is less likely to change than the whole spectrum of data.

- The lower range of frequencies in the human audible range has more information about the syllable than the upper part. When the whole spectogram is considered, this property of speech cannot be implemented properly (albeit methods like mel-spectrum has been devised). The analysis of pattern frequency by frequency gives the freedom to focus more on the lower range of frequencies, hence yielding better decisions on recognition.

7.5 Computational limitations

Although this model seems to more-or-less emulate the speech recognition model of human ear and brain, there are some computational limitations for implementing this.

Human audible range is upto 4000Hz, so there lies the need to create 4000 models of the cochlea-hair for proper emulation. Although the computational speed and capacity has been substantially increased in recent days, creating models for all these 4000 points is impossible and impracticable as there lies the necessity of handling the computational and memory part for all these objects. Although human brain is capable of doing such type of parallel procesing, computers aren't.

7.5.1 Solutions

There can be two possible ways to cope with the computational limitation.

1. Instead of performing pattern matching for each and every frequencies in the audible range, we can take a limited number of samples that lie with equal spacing in the same range.

2. Since all the objects of cochlea-hair operate in the same way, they can be parallel processed over number of computers.

Chapter 8

Design

8.1 Recogniser Model

The model developed that emulates the cochlea's hair inside our ear has the following functionalities.

- Take sound input from microphone or sound file.
- The Fourier Analyzer calculates the pattern of amplitude for all frequencies in the audible range.
- Fourier analyzer passes this information to the objects that represent the hairs inside the cochlea.
- Each such hair's object has its own memory which it uses to store the sample sound pattern that belongs to its frequency.
- The object can go into Training mode or Recognition mode.
- In Training Mode, the object simply stores the pattern in its memory (which it has just heard) into the database (brain)
- In Recognition Mode, the object compares the sample pattern in the memory with the samples in the database and gives the best matched syllable.
- All the objects pass the syllable they have recognized to the Decision Tool
- Decision tool, based on previous experience and statistics, detemines the most probable syllable.

- The recognized syllable is passed back to frontend, which further uses spelling checking methods to determine the correct word.

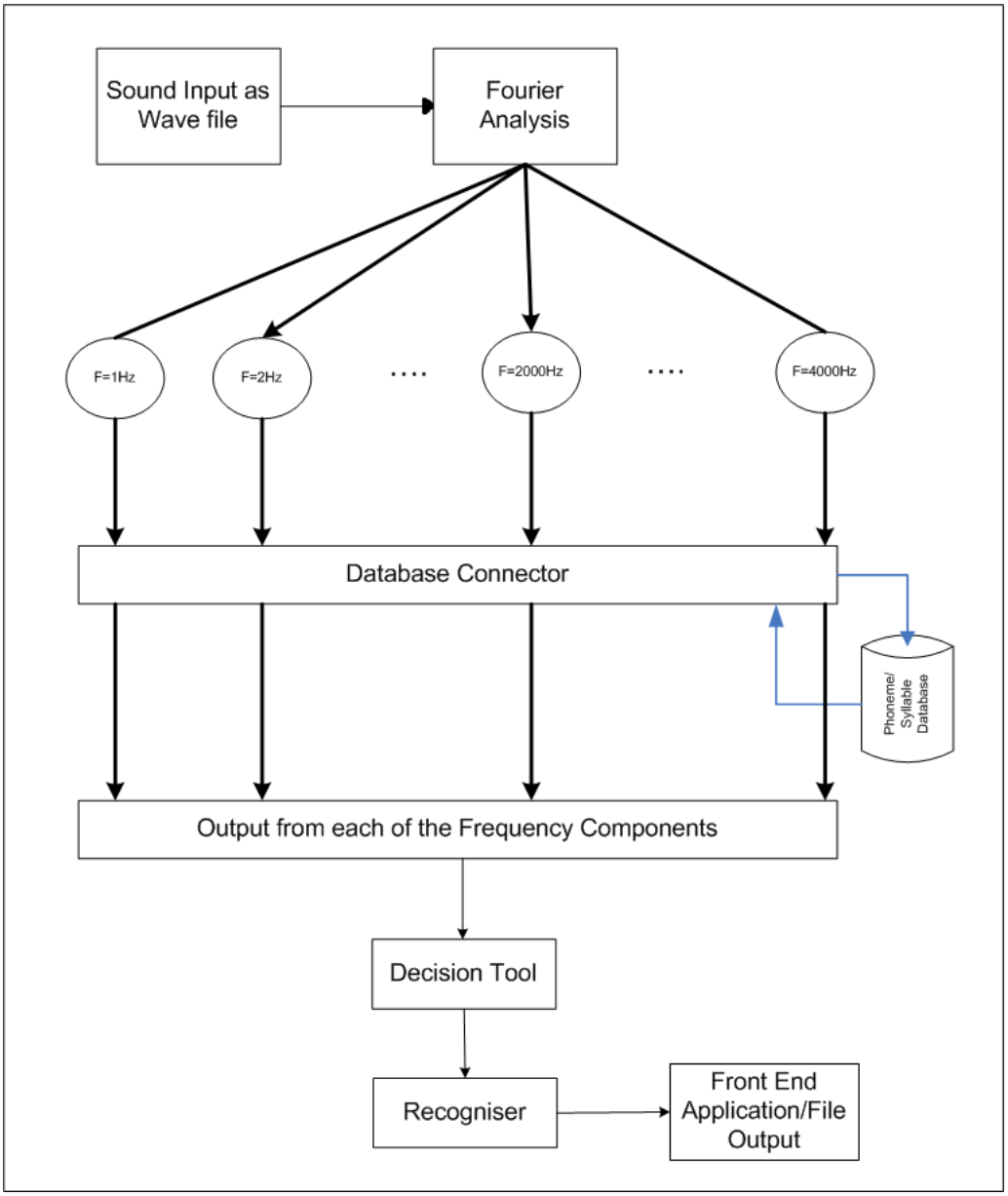

Figure 8.1: Speech Recognition using Ear Model

8.2 Database

A common database is used for all the objects. The database has the following fields.

Attribute	Description
id	Primary key
name	Name of the speaker
syllable	Syllable the sample represents
frequency	Frequency #
sample	Sample pattern for the specified frequency

Table 8.1: Table *samples*

The SQL syntax for creating the table for storing the samples of data is as follows.

```
CREATE TABLE `samples` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(255) default NULL,
  `syllable` varchar(15) NOT NULL,
  `sample` text,
  PRIMARY KEY  (`id`)
);
```

Figure 8.2: SQL Syntax

As we can see from the above table, the provision for multiple speaker is also provided. While searching the match for the input pattern, if the speaker is mentioned, the program first searches for the matching rows for the specified speaker. If no speaker is mentioned, the program searches all the data for the particular frequency.

8.3 Modes

The Recognition Engine has two modes of operation.

1. Training Mode
2. Recognition Mode

8.3.1 Training Mode

Any SR application needs to be trained before it can function properly in normal conditions. The training mode involves speaker to provide his/her voice samples for the Speech Recognition to collect enough characteristics measurment. These measurements are then used to train the SR to search for the most likely word candidate, making use of constraints imposed by different models. Throughout this process, training data are used to determine the values of the model parameters. This mode has been designed to train multiple speakers, if needed, each of them having there separate library in the database. There is no output from this mode.

The training mode is invoked as,

```
nsr -t soundfile syllable [speaker]
```

8.3.2 Recognition Mode

This is the mode that a SR usually works to recognise the users speech. In this mode the usual process of breaking the speech into syllable's and preforming the recogntion is done. Apart from this, to continuously adapt the system to users characteristics, SR collects characteristics measurment in Recognition mode too. As explained earlier in training mode, these measurments are used to train the SR. In this mode the sample input wave is compared with the database to find most likely matches for the input. The recogniser then analyses the matched outputs to provide the most probable output based on decision made by a decision tool.

The recognition mode is invoked as,

```
nsr -r soundfile [speaker]
```

8.4 Spelling Checking

A part of our project is spelling checking module which performs spelling checking on the words recognized by our engine to maximize accuracy. We are trying to choose the most likely spelling correction for that word (the "correction" may be the original word itself). There is no way to know for sure (for example, should "kalama" be corrected to "kalama" or "kalpana"?), which suggests we use probabilities. We are trying to find the correction c, out of all possible corrections, that maximizes the probability of c given the original word w:

$$argmax_c P(c|w)$$

By using Bayes theorem this is equivalent to

$$argmax_c P(c|w) = argmax_c P(w|c)P(c)/P(w)$$

Since P(w) is the same for every possible c, we can ignore it, Thus we can write it as:

$$argmax_c P(c|w) = argmax_c P(w|c)P(c)$$

There are three parts of this expression. From right to left, we have:

1. P(c), the probability that a proposed correction c stands on its own. This is called the language model: we can think it as answering the question "how likely is c to appear in an Nepali text?" So P("ghara") would have a relatively high probability, while P("xxxxy") would be near zero.

2. $P(w|c)$, the probability that w would be typed in a text when the author meant c. This is the error model: we can think it as answering "how likely is it that the author would type w by mistake when c was intended?"

3. argmaxc, the control mechanism, which says to enumerate all feasible values of c, and then choose the one that gives the best combined probability score.

One obvious question is: why take a simple expression like $P(c|w)$ and replace it with a more complex expression involving two models rather than one? The answer is that $P(c|w)$ is already conflating two factors, and it is easier to separate the two out and deal with them explicitly. Consider the misspelled word w="kalava" and the two candidate corrections c="kalama" and c="kalpana". Which has a higher $P(c|w)$? Well, "kalama" seems good because the only change is "va" to "ma", which is a small change. On the other hand, "kalpana" may also be correct. The point is that to estimate $P(c|w)$ we have to consider both the probability of c and the probability of the change from c to w anyway, so it is cleaner to formally separate the two factors. Now let us describe how the process works. First $P(c)$, We have read a big text file, nepaliwords.txt, which consists of about a million words. The file is a Nepali text of open office (a software package like Microsoft office) which contains almost all Nepali words. We will count how many times each word appears in the file. Next we train a probability model, which is a fancy way of saying we count how many times each word occurs, using a function.

There is one complication: novel words. What happens with a perfectly good word of Nepali that wasn't seen in our training data? It would be bad form to say the probability of a word is zero just because we haven't seen it yet. There are several standard approaches to this problem; we take the easiest one, which is to treat novel words as if we had seen them once. This general process is called smoothing, because we are smoothing over the parts of the probability distribution that would have been zero, bumping them up to the smallest possible count.

Now let's look at the problem of enumerating the possible corrections c of a given word w. It is common to talk of the edit distance between two words: the number of edits it would take to turn one into the other. An edit can be a deletion (remove one letter), a transposition (swap adjacent letters), an alteration (change one letter to another) or an insertion (add a letter). Here's a function that returns a set of all words c that are one edit away from w:

```
def edits1(word):
n = len(word)
return
   # deletion
   set([word[0:i]+word[i+1:] for i in range(n)] +
   # transposition
   [word[0:i]+word[i+1]+word[i]+word[i+2:] for i in range(n-1)] +
   # alteration
   [word[0:i]+c+word[i+1:] for i in range(n) for c in alphabet] +
   # insertion
   [word[0:i]+c+word[i:] for i in range(n+1) for c in alphabet])
```

This is not really a small set. But to increase accuracy we need to consider edit distance 2. That's easy: just apply *edits1* to all the results of *edits1*: Since we aren't going beyond edit distance 2, to reduce the set we can do a small optimization: only keep the candidates that are actually known words. We still have to consider all the possibilities, but we don't have to build up a big set of them.

Now the only part left is the error model, $P(w|c)$. Here's where we ran into difficulty. For this we made some assumptions: mistaking one vowel for another is more probable than mistaking two consonants; making an error on the first letter of a word is less probable, etc. So we defined a model that says all known words of edit distance 1 are infinitely more probable than known words of edit distance 2, and infinitely less probable than a known word of edit distance 0. By "known word" I mean a word that we have seen in the

language model training data – a word in the dictionary. We can implement this strategy as follows:

```
def known(words): return set(w for w in words if w in NWORDS)
def correct(word):
    candidates = known([word]) or known(edits1(word)) or
    known_edits2(word) or [word]
    return max(candidates, key=lambda w: NWORDS[w])
```

The function *correct* chooses as the set of candidate words the set with the shortest edit distance to the original word, as long as the set has some known words. Once it identifies the candidate set to consider, it chooses the element with the highest $P(c)$ value, as estimated by the NWORDS model. This is the way how it works.

8.5 Nepali corpus analysis

To augment the accuracy in speech recognition process, we also did the frequency analysis of Nepali language on letter and syllable level.

8.5.1 Corpus

The corpus was created using gathering texts from leading Nepali e-newspaper and wiki web-sites like eKantipur, Hamro Samachar and Nepali Wikipedia. Altogether 7797 articles were collected for this purpose which consisted of 2452937 words.

8.5.2 Analysis

The frequency analysis has been done in both letter and syllable level. The result are too big to be included with this report. It can be found at http://groups.google.com/group/nepalispeech/files. The Python scripts that were used to calculate the frequencies are in Appendix C.

Chapter 9

Implementation

The overall system has been divided into two parts.

1. Backend
2. Frontend

9.1 Backend

The backend consists of a component that runs in the background and acts as a server constantly listening for any inputs to it. The chief function of backend software is as follows:

- Run in the background and wait the client for the input
- Receive the input in the form of wave file which consists of waveform of one word that is to be recognized.
- Recognize the word and send the text equivalent of it to the client

9.2 Frontend

The frontend consists of a graphical interface, whose function is to take input form the user either in the form of file or directly through microphone. The function of frontend software is as follows:

1. Provide a graphical interface to the user
2. Receive the input from the user either in the form of wave file or directly from microphone.

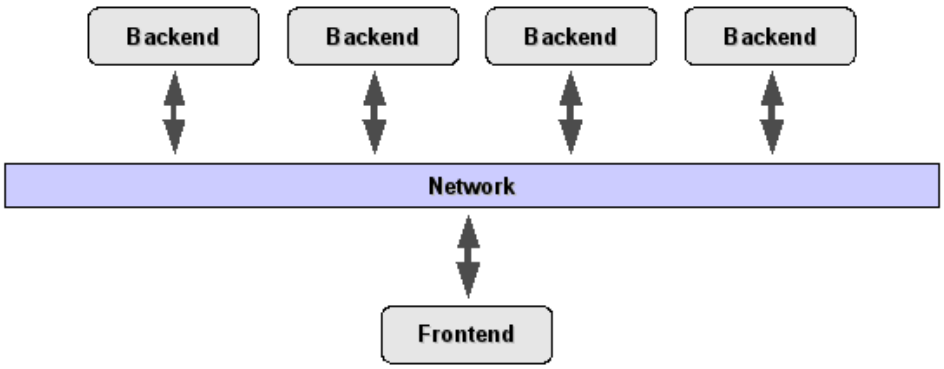

Figure 9.1: Multiple backends

3. Separate the speech into separate words
4. Pass the word to the backend server
5. Get the recognized word (in the form of text) from the server
6. Display the text or pass it to the foreground application.

9.3 Use of frontend-backend architecture

In a simpler sense, the use of frontend-backend architecture allows to focus on individual aspects of the project separately.

But on the other hand, it also allows to put these two components in separate computers allowing to have multiple backends if more processing speed is desired or the feature to have multiple clients if it is required. Since the model requires a lot of processing and the currently available computation power is insufficient, the only solution is to divide the components that can be parallelly processed and run then in different CPUs.

The engine that we've developed is illustrated in Figure 9.1. As we can see, the frontend and backend parts are connected through network. So there can be multiple number of backends. Whenever a speech input is received, it can be broken down to its constituent syllables sounds and each can be sent to different backends available, which can parallelly process the samples and send back the recognized syllable.

Chapter 10

Technical Details

10.1 Platform

The software has been designed to work on multiple platforms. The selection of tools and programming languages have been selected accordingly.

10.2 Frontend

The frontend has been programmed in Python. The use of Python is due to the following reasons.

- Simple
- Easy to Learn
- Free and Open Source
- High-level Language
- Portable
- Interpreted
- Object Oriented
- Extensible
- Embeddable
- Extensive Libraries

The libraries used with python are as follows:

10.2.1 wxPython

wxPython is a GUI toolkit for the Python programming language. It allows Python programmers to create programs with a robust, highly functional graphical user interface, simply and easily. It is implemented as a Python extension module (native code) that wraps the popular wxWidgets cross platform GUI library, which is written in C++.

10.2.2 PortAudio

PortAudio is a free, cross platform, open-source, audio I/O library. It lets you write simple audio programs in 'C' that will compile and run on many platforms including Windows, Macintosh (8,9,X), Unix (OSS), SGI, and BeOS. PortAudio is intended to promote the exchange of audio synthesis software between developers on different platforms. PortAudio provides a very simple API for recording and/or playing sound using a simple callback function.

10.3 Backend

The backend component has been programmed in C++. The use of C++ is due to the following reasons.

- Portable
- Fast
- Widely used
- Power and Flexibility
- Efficiency

10.4 Database

Currently, SQLite is database the database of choice for this project. Although the selection of this database is on the fact that it it flat file based, there are several reasons too.

- Zero-configuration - no setup or administration needed.
- Implements most of SQL92.

- A complete database is stored in a single cross-platform disk file.
- Supports terabyte-sized databases and gigabyte-sized strings and blobs.
- Small code footprint: less than 250KiB fully configured or less than 150KiB with optional features omitted.
- Faster than popular client/server database engines for most common operations.
- Simple, easy to use API.
- Available as a single ANSI-C source-code file that can be easily droped into another project.
- Self-contained: no external dependencies.
- Cross-platform: Linux (unix), MacOSX, OS/2, Win32 and WinCE are supported out of the box. Easy to port to other systems.

Chapter 11

Output and accuracy

The Ear Model turned out to be fairly accurate than the conventional methods of speech recognition. For the same set of data that we used for conventional speech recognition, the Ear Model gave us more accurate and reliable results.

For a single speaker and with two training set of data, we achieved the following results.

While trying to recognize */ka/*, the Ear Model engine gave the following top five possibilities

Syllable	Accuracy %
/ka/	33.8462
/ga/	12.3077
/pa/	6.15385
/ta/	4.61538
/tta/	4.61538

Table 11.1: Recognition for /ka/

For */kha/*, the output was as,

Syllable	Accuracy %
/nga/	12.3077
/wa/	9.23077
/kha/	9.23077
/gnya/	9.23077
/pha/	6.15385

Table 11.2: Recognition for /kha/

For */ga/*, the output was as,

Syllable	Accuracy %
/ga/	24.6154
/ka/	9.23077
/ynga/	4.61538
/ttha/	4.61538
/cha/	4.61538

Table 11.3: Recognition for /ga/

For */gha/*, the output was as,

Syllable	Accuracy %
/gha/	24.6154
/kchya/	6.15385
/tta/	6.15385
/la/	4.61538
/tha/	4.61538

Table 11.4: Recognition for /gha/

For */ta/*, the output was as,

Syllable	Accuracy %
/tta/	16.9231
/ta/	16.9231
/ka/	12.3077
/pa/	6.15385
/ttha/	6.15385

Table 11.5: Recognition for /ta/

From above tables, we can see that the Ear Model seems to be quite promising model for speech recognition. In conventional methods, there used to be just one comparision with the stored data, but in this model, the stored pattern is checked in the whole frequency range. Due to high number of matching and taking the most recognized syllable as the final recognized syllable, the accuracy is multiplied.

We can see that for syllables /ka/, /ga/, /gha/, etc the model has been successful to recognize them correctly. For /kha/ and /ta/ we can see that

they are recognized as the second top recognized syllable. By further application of statistical methods like HMM or Neural Network, the accuracy of the model can be enhanced further.

Hence we can conclude that the model is fairly accurate. For the same speech samples, we had achieved not more than 68% accuracy but in this model, we have achieved more than 80% accuracy.

Chapter 12

Limitations and Future Enhancement

We spent half of our time experimenting on the conventional methods of speech recognition and the other half was used for researching and tuning the Ear Model. So there lies a lot of places that can be worked on.

12.1 Narrow Domain

Currently the speech recognition engine that we have developed works in a very narrow domain. Due to high amount of processing to be done, by virtue of several cochlea-fiber simulation, the result is instantenous only if small number of syllables are trained and recognized. The time required for processing multiplies as the training data increases for same or multiple speakers. Hence to make it effective, the current model has been designed to work in narrow domain. However, these shortcomings can be overcomed by using the enhancement methods described in following sections.

12.2 Offline

Due to lack of time and major focus on tuning up the Ear Model's realiazation, the current system works offline. That is, it can't directly take input from microphone and do the recognition task directly but instead takes the speech sample from a Wave file and processes it to give the recognized syllables. However this can only be considered as incomplete task and not a shortcoming of the model.

12.3 Parallel Processing

The Ear Model was designed with parallel processing in mind from the beginning. As we know our brain is a complex parallel processing device and it processes the signals that come from fibers in the cochlea parallelly to recognize the syllables and words. Since our model also simulates the same behaviour, the objects that simulates the fibers of cochlea can be executed parallelly in different CPUs. As the number of training samples increases, this also becomes inevitable so as to increase execution speed and efficiency. Currently the speech recognition engine only supports parallel processing in the engine level and it remains as a future enhancement to support parallel processing in the object level (representing fibers of cochlea).

12.4 Frontend

The GUI of our system also has lots of places for improvement. As mentioned earlier, the project was focused on research of the Ear Model and entirely neglected the development of a good interface, which hence becomes the chief future enhancement to be done.

Chapter 13

Conclusion

Speech Recognition is here to stay. As a result, an increasing number of application are vying to use the functionality of Speech Recognition for inputs. Though the original objective of this project was to build a Nepali Speech Recognition program, in the course of the project, we encountered many obstacles, which prompted us to rather investigate the different methods available to find the most appropriate method for Nepali Speech Recognition. This is important because different methods available for Speech Recognition comes with some trade-offs that needs to be analyzed for efficiency and adaptability. In this work, we focus on two major methods, FFT and MFCC, for their efficiency and adaptability. And further on finding the shortcomings in these methods we propose an alternative to these methods.

First, understanding the behavior of FFT and MFCC is in itself difficult, because it involves a complex interaction between time and frequency domains of the speech signal. Though these methods have been widely used in many speech recognition applications, we found that it was not the best suited method for Nepali Speech Recognition. We experimented both the models to compare speech signals based on template matching with a set of standard library. Nevertheless, though the initial findings were satisfactory in a narrow domain, for larger library the accuracy plummeted to an unacceptable level. With an average accuracy of 67% achieved from these conventional methods, we were certain that it was not enough for the proposed Nepali Speech Recognition. However, the addition of statistical and probabilistic methods like HMM and ANN, could have improved the overall efficiency of the recognition, nonetheless, the preliminary accuracy of 67% was not encouraging enough to move ahead with these methods.

Next, in an attempt to improve the accuracy of the upper mentioned methods, we experimented with different parameters required for analyzing the speech signals. For this, we manipulated the frame size and the window

size of the signals to establish a norm for the best possible exactness. Even with all these attempts we could not cross the 68% accuracy mark.

Due to the above mentioned issue with the accuracy, as an alternative to these methods we have proposed a completely radical approach to mimic human hearing as far as possible. This model, which we have named as 'Ear Model', is an effort to mimic the basic process of human hearing. The model to some extent, simulates a virtual ear working in conjunction with a virtual mind. We have worked hard on making this model as sound as possible for the use in Nepali Speech Recognition. From the experimentation we have achieved fairly good accuracy in recognizing isolated alphabets. Due to impressive accuracy of 80% obtained just from the ear model without the use of any statistical or probabilistic method, we can be sure that with the use of statistical methods like HMM and ANN we can further enhance the accuracy. Thus we can propose this method to be used as a back bone for the Nepali Speech Recognition.

Even though we have created a base for new approach to the speech recognition, there is a huge scope for improvement in this model. Hence we expect to see more work done on this model in future so that this model emerges as good model for the future of Speech Recognition attempts; in particular the Nepali Speech Recognition, which is our primary goal for proposing this method.

Appendix A

Experiment Data

Sound	Accuracy of Recognition
/ka/	0.97143
/kha/	0.77143
/ga/	0.94286
/gha/	0.88571
/ynga/	0.62857
/cha/	0.68571
/chha/	0.34286
/ja/	0.00000
/jha/	0.28571
/ynda/	0.97143
/tta/	0.91429
/ttha/	0.74286
/da/	0.02857
/dha/	0.91429
/ynda/	0.85714
/ta/	0.94286
/tha/	0.62857
/the/	0.25714
/dha/	0.88571
/na/	0.42857
/pa/	0.77143
/pha/	0.42857
/ba/	0.51429
/bha/	0.74286
/ma/	0.97143
/ya/	0.94286
/ra/	0.28571
/la/	0.48571
/wa/	0.97143
/sa/	0.60000
/sha/	0.94286
/ha/	0.54286
/kchya/	0.97143
/tra/	0.62857
/gnya/	0.08571
"The total average is 0.656327"	

Table A.1: Sample data for MFCC based feature extraction pattern with 64 sample/frame 12 frames with 50% Overlap

Sound	Accuracy
/*ka*/	0.91429
/*kha*/	0.80000
/*ga*/	0.97143
/*gha*/	0.88571
/*ynga*/	0.74286
/*cha*/	0.60000
/*chha*/	0.34286
/*ja*/	0.11429
/*jha*/	0.40000
/*ynda*/	0.57143
/*tta*/	0.91429
/*ttha*/	0.74286
/*da*/	0.08571
/*dha*/	0.85714
/*ynda*/	0.77143
/*ta*/	0.94286
/*tha*/	0.77143
/*the*/	0.02857
/*dha*/	0.88571
/*na*/	0.31429
/*pa*/	0.94286
/*pha*/	0.45714
/*ba*/	0.48571
/*bha*/	0.85714
/*ma*/	0.97143
/*ya*/	0.97143
/*ra*/	0.51429
/*la*/	0.48571
/*wa*/	0.97143
/*sa*/	0.05714
/*sha*/	0.22857
/*ha*/	0.68571
/*kchya*/	0.97143
/*tra*/	0.68571
/*gnya*/	0.05714
"The total average is 0.628571"	

Table A.2: Sample data for MFCC based feature extraction pattern with 48 sample/frame 12 frames with 50% Overlap

Sound	Accuracy
/*ka*/	0.94
/*kha*/	0.91
/*ga*/	0.94
/*gha*/	0.77
/*ynga*/	0.57
/*cha*/	0.46
/*chha*/	0.37
/*ja*/	0.06
/*jha*/	0.26
/*ynda*/	0.23
/*tta*/	0.94
/*ttha*/	0.71
/*da*/	0.4
/*dha*/	0.83
/*ynda*/	0.86
/*ta*/	0.89
/*tha*/	0.8
/*the*/	0.06
/*dha*/	0.89
/*na*/	0.54
/*pa*/	0.86
/*pha*/	0.37
/*ba*/	0.37
/*bha*/	0.89
/*ma*/	0.97
/*ya*/	0.97
/*ra*/	0.43
/*la*/	0.57
/*wa*/	0.97
/*sa*/	0.23
/*sha*/	0.2
/*ha*/	0.69
/*kchya*/	0.97
/*tra*/	0.66
/*gnya*/	0.57
"The total average is 0.632653	

Table A.3: Sample data for MFCC based feature extraction pattern with 32 sample/frame 14 frames with 50% Overlap

Sounds	Accuracy
/*ka*/	0.82353
/*kha*/	0.67647
/*ga*/	0.88235
/*gha*/	0.73529
/*ynga*/	0.00000
/*cha*/	0.41177
/*chha*/	0.05882
/*ja*/	0.32353
/*jha*/	0.97059
/*ynda*/	0.52941
/*tta*/	0.91177
/*ttha*/	0.58824
/*da*/	0.29412
/*dha*/	0.91177
/*ynda*/	0.85294
/*ta*/	0.91177
/*tha*/	0.61765
/*the*/	0.20588
/*dha*/	0.97059
/*na*/	0.52941
/*pa*/	0.94118
/*pha*/	0.44118
/*ba*/	0.64706
/*bha*/	0.76471
/*ma*/	0.88235
/*ya*/	0.94118
/*ra*/	0.23529
/*la*/	0.64706
/*wa*/	0.88235
/*sa*/	0.76471
/*sha*/	0.88235
/*ha*/	0.50000
/*kchya*/	0.97059
/*tra*/	0.70588
/*gnya*/	0.47059
"The total average is 0.673010"	

Table A.4: Sample data for the plain FFT based template matching using Windows with 48 sample and total of 12 windows

Sound	Accuracy
/*ka*/	0.97143
/*kha*/	0.62857
/*ga*/	0.94286
/*gha*/	0.85714
/*ynga*/	0.00000
/*cha*/	0.85714
/*chha*/	0.37143
/*ja*/	0.14286
/*jha*/	0.17143
/*ynda*/	0.94286
/*tta*/	0.88571
/*ttha*/	0.71429
/*da*/	0.02857
/*dha*/	0.94286
/*ynda*/	0.82857
/*ta*/	0.82857
/*tha*/	0.65714
/*the*/	0.57143
/*dha*/	0.97143
/*na*/	0.57143
/*pa*/	0.80000
/*pha*/	0.28571
/*ba*/	0.65714
/*bha*/	0.80000
/*ma*/	0.97143
/*ya*/	0.88571
/*ra*/	0.20000
/*la*/	0.42857
/*wa*/	0.91429
/*sa*/	0.91429
/*sha*/	0.88571
/*ha*/	0.54286
/*kchya*/	0.91429
/*tra*/	0.65714
/*gnya*/	0.28571
"The total average is 0.657959"	

Table A.5: Sample data for the plain FFT based template matching using Windows with 64 sample and total of 12 windows

Appendix B

Matlab Codes

Feature extraction using MFCC

```
%The experimentation system of finding the accuracy of the MFCC
%co-efficients in template based matching. Coded as a part of final year
%project on "Nepali SR".

%filename of the library
filename={'ba.WAV','bha.WAV','cha.WAV','chha.WAV','da.WAV','ddha.WAV','dha.WAV',
 'ga.WAV','gha.WAV','gnya.WAV','ha.WAV','ja.WAV','jha.WAV','ka.WAV','kchya.WAV',
 'kha.WAV','la.WAV','ma.WAV','na.WAV','nga.WAV','pa.WAV','pha.WAV','ra.WAV',
 'sa.WAV','sha.WAV','ta.WAV','tha.WAV','the.WAV','tra.WAV','tta.WAV','ttha.WAV',
 'wa.WAV','ya.WAV','ynda.WAV','ynga.WAV'};
%filenamefor the naming of the consonants
name={'ka.wav','kha.wav','ga.wav','gha.wav','ynga.wav','cha.wav','chha.wav',
  'ja.wav','jha.wav','ynda.wav','tta.wav','ttha.wav','da.wav','dha.wav',
  'ynda.wav', 'ta.wav','tha.wav','the.wav','dha.wav','na.wav','pa.wav',
  'pha.wav','ba.wav','bha.wav', 'ma.wav','ya.wav','ra.wav','la.wav',
  'wa.wav','sa.wav','sha.wav','ha.wav','kchya.wav','tra.wav','gnya.wav'};
%filename of the input sample data
consonant={'consonant0.wav','consonant1.wav','consonant2.wav', 'consonant3.wav','consonant4.wav',
'consonant5.wav','consonant6.wav','consonant7.wav',  'consonant8.wav','consonant9.wav',
'consonant10.wav','consonant11.wav','consonant12.wav',
 'consonant13.wav','consonant14.wav','consonant15.wav','consonant16.wav','consonant17.wav',
 'consonant18.wav','consonant19.wav','consonant20.wav','consonant21.wav','consonant22.wav',
 'consonant23.wav','consonant24.wav','consonant25.wav','consonant26.wav','consonant27.wav',
 'consonant28.wav','consonant29.wav','consonant31.wav','consonant32.wav','consonant33.wav',
'consonant34.wav','consonant35.wav'};
%length(filename)
n=64;%samples sper frame
frame=12; % number of frames
%Writing headers in the output files
ID=fopen('e:\consonants1.txt','w');
fprintf(ID, 'This data is the sample data for %f samples per frame, %f number of frames\n',n,frame)
fclose(ID);
ID1=fopen('e:\consonants1final.txt','w');
fprintf(ID1, 'This data is the sample data for %f samples per frame, %f number of frames\n',n,frame)
fclose(ID1);
total=0;
% reading each of the syllable as sliced by the py program
for k = 1 :length(consonant)
```

```
cd e:\consonants
wav=wavread(consonant{1,k});
wavlength=length(wav);
fftarrsamp=[];
wavsamp=wav(1:(n-1));
%applying the hamming window
wavsamp = wavsamp .* hamming((n-1));
%finding the wavsample's MFCC co-effecient
 fft_wav=melfcc(wavsamp);
 fftarrsamp=[fftarrsamp; fft_wav];
loop=abs(wavlength/n);
%segmenting it and finding the 39 co-efficients
for i = 1:frame
     wavsamp=wav(((i-0.5)*n)+1:(i+0.5)*n);
     wavsamp = wavsamp .* hamming(n);
     fft_wav=melfcc(wavsamp);
     fft_delta=deltas(fft_wav,n);
      fft_d_delta=(deltas(fft_delta,n));
     fftarrsamp=[fftarrsamp; fft_wav;fft_delta;fft_d_delta];
end
fftarr_sam_transpose=transpose(fftarrsamp);
corrarr=[];
% reading the library
for j = 1:length(filename)
    cd e:\matlab7\work\sound\consonants
    wav=wavread(filename{1,j});
    wavlength=length(wav);
    %reading each data syllable from the library
    fftarr=[];
    wavsamp=wav(1:n);
    %applying the hamming window
    wavsamp = wavsamp .* hamming(n);
    %finding the wavsample's MFCC Co-efficeint
    fft_wav=melfcc(wavsamp);
    fftarr=[fftarr; fft_wav];
    loop=abs(wavlength/256);
    for i = 1:frame
         wavsamp=wav(((i-0.5)*n)+1:(i+0.5)*n);
         wavsamp = wavsamp .* hamming(n);
         fft_wav=melfcc(wavsamp);
         fft_delta=deltas(fft_wav,n);
         fft_d_delta=(deltas(fft_delta,n));
         fftarr=[fftarr; fft_wav;fft_delta;fft_d_delta];
    end
    fftarr_orig_trans=transpose(fftarr);
    %comparing it with each other
    cor=corr2(fftarr_orig_trans,fftarr_sam_transpose);
    corrarr=[corrarr;cor];
    %sprintf('%s,  %f ' ,filename{1,j},cor)
end
data=[];
corrarr;
% writing the output to the file,ID for detailed output, ID1 for
% summarised output
ID=fopen('e:\consonants1.txt','a');
ID1=fopen('e:\consonants1final.txt','a');
k
% sorting the output to put the syllable the most matching percentage
% on the top
for i = 1:35
    for j=1:(35-i)
        if (corrarr(j)<corrarr(j+1))
```

```
                temp = corrarr(j+1);
                corrarr(j+1)=corrarr(j);
                corrarr(j)=temp;

                tempstr=filename{1,j+1};
                filename{1,j+1} = filename{1,j};
                filename{1,j} = tempstr;
            end
        end
    end
    %Findng to see where the required syllable is in the sorter data
    for i = 1:35
        if (strcmp(lower(name{1,k}),lower(filename{1,i}))==1)
              break;
        end
    end
    %writing the data to the file
    fprintf(ID,'\n\nfor the consonant %s',consonant{1,k});
    fprintf(ID,'\taverage is \t%f\n',((35-i)/35));
    fprintf(ID1,'%s\t%f\t%f\n',lower(name{1,k}),((35-i)/35),i);
    for j = 1:35
        fprintf(ID,'%s\t%f\n',filename{1,j},corrarr(j));
    end
    fclose(ID);
    %fclose(ID1);
    %fining the overall average accuracy of the whole system
    total=total+(35-i)/35;
end
%Writing the value to the summary file
fprintf(ID1,'The total average is %f\n',(total/35));
fclose(ID1);
```

Feature Extraction using direct FFT

```
%The experimentation system of finding the accuracy of the simple FFT based
%in template based matching. Coded as a part of final year
%project on "Nepali SR".


%filename of the library
filename={'ba.WAV','bha.WAV','cha.WAV','chha.WAV','da.WAV','ddha.WAV','dha.WAV',
  'ga.WAV','gha.WAV','gnya.WAV','ha.WAV','ja.WAV','jha.WAV','ka.WAV','kchya.WAV',
  'kha.WAV','la.WAV','ma.WAV','na.WAV','nga.WAV','pa.WAV','pha.WAV','ra.WAV',
  'sa.WAV','sha.WAV','ta.WAV','tha.WAV','the.WAV','tra.WAV','tta.WAV','ttha.WAV',
  'wa.WAV','ya.WAV','ynda.WAV','ynga.WAV'};

%filenamefor the naming of the consonants
name={'ka.wav','kha.wav','ga.wav','gha.wav','ynga.wav','cha.wav','chha.wav',
  'ja.wav','jha.wav','ynda.wav','tta.wav','ttha.wav','da.wav','dha.wav',
  'ynda.wav','ta.wav','tha.wav','the.wav','dha.wav','na.wav','pa.wav','pha.wav',
  'ba.wav','bha.wav','ma.wav','ya.wav','ra.wav','la.wav','wa.wav','sa.wav',
  'sha.wav','ha.wav','kchya.wav','tra.wav','gnya.wav'};

%filename of the input sample data
consonant={'consonant0.wav','consonant1.wav','consonant2.wav','consonant3.wav',
'consonant4.wav','consonant5.wav','consonant6.wav','consonant7.wav',
'consonant8.wav','consonant9.wav','consonant10.wav',
'consonant11.wav', 'consonant12.wav','consonant13.wav','consonant14.wav','consonant15.wav',
```

```
'consonant16.wav','consonant17.wav','consonant18.wav','consonant19.wav','consonant20.wav',
'consonant21.wav','consonant22.wav','consonant23.wav','consonant24.wav','consonant25.wav',
'consonant26.wav','consonant27.wav','consonant28.wav','consonant29.wav',
'consonant31.wav','consonant32.wav','consonant33.wav','consonant34.wav','consonant35.wav'};
%length(filename)
n=64;%samples sper frame
frame=12; % number of frames
%Writing headers in the output files
ID=fopen('e:\sampcorr64s12f.txt','w');
fprintf(ID, 'This data is the sample data for %f samples per frame, %f number of frames\n',n,frame)
fclose(ID);
ID1=fopen('e:\sampcorr64s12fsum.txt','w');
fprintf(ID1, 'This data is the sample data for %f samples per frame, %f number of frames\n',n,frame)
fclose(ID1);
total=0;
%reading the sample wave file
for k = 1 :length(consonant)
    cd e:\consonants
    wav=wavread(consonant{1,k});
    wavlength=length(wav);
    fftarrsamp=[];
    wavsamp=wav(1:(n-1));
    %applying the hammming window
    wavsamp = wavsamp .* hamming((n-1));
    fft_wav=abs(fft(wavsamp,n));
    fftarrsamp=[fftarrsamp; fft_wav];
    loop=abs(wavlength/n);
    %calculating thefftvalues of the windowed wave samples with 50% overlap
    for i = 1:frame
         wavsamp=wav(((i-0.5)*n)+1:(i+0.5)*n);
         wavsamp = wavsamp .* hamming(n);
         fft_wav=abs(fft(wavsamp,n));
         fftarrsamp=[fftarrsamp; fft_wav];
    end
    fftarr_sam_transpose=transpose(fftarrsamp);
    corrarr=[];
    % comparing the data with the sample in the library
    for j = 1:length(filename)
        cd e:\matlab7\work\sound\consonants
        wav=wavread(filename{1,j});
        wavlength=length(wav);
        fftarr=[];
        wavsamp=wav(1:n);
        wavsamp = wavsamp .* hamming(n);
        fft_wav=abs(fft(wavsamp,n));
        fftarr=[fftarr; fft_wav];
        loop=abs(wavlength/256);
        for i = 1:frame
             wavsamp=wav(((i-0.5)*n)+1:(i+0.5)*n);
             wavsamp = wavsamp .* hamming(n);
             fft_wav=abs(fft(wavsamp,n));
             fftarr=[fftarr; fft_wav];
        end
        fftarr_orig_trans=transpose(fftarr);
    Ψ%performing the correlation on the sample and library data
        cor=corr2(fftarr_orig_trans,fftarr_sam_transpose);
        corrarr=[corrarr;cor];
        %sprintf('%s,  %f ' ,filename{1,j},cor)
    end
    %Opening the files for writing the data, one for the detail data, one for summary of the experiment.
    ID=fopen('e:\sampcorr64s12f.txt','a');
    ID1=fopen('e:\sampcorr64s12fsum.txt','a');
```

```
Ψ%sorting the result according to maximum value of accuracy

    for i = 1:34
        for j=1:(34-i)
            if (corrarr(j)<corrarr(j+1))
                temp = corrarr(j+1);
                corrarr(j+1)=corrarr(j);
                corrarr(j)=temp;

                tempstr=filename{1,j+1};
                filename{1,j+1} = filename{1,j};
                filename{1,j} = tempstr;
            end
        end
    end
    %computing the current position of the sound in the calculated and sorted data
    for i = 1:35
        if (strcmp(lower(name{1,k}),lower(filename{1,i}))==1)
               break;
        end
    end
Ψ%writing the data to the file for review
      fprintf(ID,'\n\nfor the consonant %s',consonant{1,k});
      fprintf(ID,'\taverage is \t%f\n',abs(((35-i)/35)));
      fprintf(ID1,'%s\t%f\t%f\n',lower(name{1,k}),abs(((35-i)/35)),i);

    for j = 1:35
        fprintf(ID,'%s\t%f\n',filename{1,j},corrarr(j));
    end
    fclose(ID);
    total=total+abs(((35-i)/35));
end
%writing the overall percentage of accuracy to the file.
ID1=fopen('e:\sampcorr64s12fsum.txt','a');
fprintf(ID1,'The total average is %f\n',(total/35));
fclose(ID1);
```

Appendix C

Nepali corpus analysis code

```
#! /usr/bin/env python
# -*- coding: UTF-8 -*-

import os

# not value of variables not shown due to
# unicode incompatibilities
consonants = u"..."
vowels = u"..."
joiners = u"..."
othersymbols = u".."

# file and folder path
textfolder = "./nepaliarticles/" # folder in which files are kept
#resultfile = "freqency.txt" # file in which frequency is updates

filelist = os.listdir(textfolder)


# module definition
# this module calculates the freq of given str in all the files and updates the result to resultfile
def calculateAndUpdate(findString):
Ψcount = 0
Ψfor f in filelist:
ΨΨin_file = open(textfolder+f, "r")
ΨΨtext = in_file.read()
ΨΨin_file.close()

ΨΨcount += text.count(findString.encode('utf-8'))

Ψprint findString, count

Ψout_file = open(resultfile, "a")
Ψout_file.write(findString.encode('utf-8')+"\t"+str(count)+"\n")
Ψout_file.close()



# calculate consonants
resultfile = "frequency-consonants.txt"
for i in range(len(consonants)):
ΨcalculateAndUpdate(consonants[i])
```

```
# calculate vowels
resultfile = "frequency-vowels.txt"
for i in range(len(vowels)):
ΨcalculateAndUpdate(vowels[i])

# calculate joiners
resultfile = "frequency-joiners.txt"
for i in range(len(joiners)):
ΨcalculateAndUpdate(joiners[i])

# calculate othersymbols
resultfile = "frequency-othersymbols.txt"
for i in range(len(othersymbols)):
ΨcalculateAndUpdate(othersymbols[i])

# calculate consonant+joiner
resultfile = "frequency-consonant-joiner.txt"
for i in range(len(consonants)):
Ψfor j in range(len(joiners)):
ΨΨcalculateAndUpdate(consonants[i]+joiners[j])


# calculate consonant+consonant
resultfile = "frequency-consonant-consonant.txt"
for i in range(len(consonants)):
Ψfor j in range(len(consonants)):
ΨΨcalculateAndUpdate(consonants[i]+consonants[j])
```

Appendix D

Resources

Nepali Speech Recognition Google Groups

http://groups.google.com/group/nepalispeech

Bibliography

[1] Nalini Vasudevan, Anushruthi Rai, Arjun Jain: *A Connectionist Framework For Feature Based Speech Recognition System Using Artificial Neural Networks*

[2] M. Kumar, N. Rajput, A. Verma: *A Large-Vocabulary Continuous Speech Recognition System For Hindi*

[3] L.R. Rabiner, M.R. Sambur: *An Algorithm For Determining The Endpoints For Isolated Utterances*

[4] Lori F. Lamel, Lawrence R. Rabiner, Aaron E Rosenberg, Jay G. Wilpon: *An Improved Endpoint Detector For Isolated Word Recognition*

[5] Austin Marshall: *Artificial Neural Network For Speech Recognition*

[6] Lawrence R. Rabiner: *A Tutorial On HMM And Selected Application In Speech Recognition*

[7] Don Colton: *Automatic Speech Recognition Tutorial*

[8] Hugo Meinedo, Joao P. Neto: *Combination Of Acoustic Models In Continuous Speech Recognition Hybrid Systems*

[9] Rita H Wouhaybi, Mohamad Adnal Al-Alaoui: *Comparison Of Neural Networks For Speaker Recognition*

[10] Cameron Elliott, Jeff A. Bilmes: *Computer Based Mathematics Using Continuous Speech Recognition*

[11] Eamonn J. Keogh, Michael J. Pazzani: *Derivative Dynamic Time Warping*

[12] John Fry, San Jose State University: *Digital Signal Processing*

[13] Stan Salvador And Philip Chan: *Fast DTW: Toward Accurate Dynamic Time Warping In Linear Time And Space*

[14] Dr. Joseph Picone: *Fundametals Of Speech Recognition: A Short Course*

[15] Barbara Resch: *Hidden Markov Models*

[16] Roberto Gemello, Franco Mana, Dario Albesano., *Hybrid HMM/Neural Network based Speech Recognition in Loquendo ASR*

[17] Speech Analysis : *Tony Robinson, http://svr-www.eng.cam.ac.uk/ãjr/sa95/speechanalysis.html*

[18] Udhyakumar.N, Swaminathan.R and Ramakrishnan.S.K., *Multilingual Speech Recognition for Information Retrieval in Indian context*

[19] Ram Prasad Gyawali, et al. *Nepali Vyakaran Tatha Rachana*

[20] Biing Hwang Juang: *Pattern Recognition In Speech And Language Processing*

[21] Joseph Keshet, Shai Shalev-Shwartz, Yoram Singer, Dan Chazan: *Phoneme Alignment Based On Discriminative Learning*

[22] Dan Ellis *PLP and RASTA (and MFCC, and inversion) in Matlab, http://labrosa.ee.columbia.edu/matlab/rastamat/*

[23] Isaac Saldana, David Ginsberg: *Remote Speaker And Speech Recognition: A Senior Design Project*

[24] Dongsuk Yuk: *Robust Speech Recognition Using Neural Networks And Hidden Markov Models: Adaptations Using Non-Linear Transformations*

[25] Seonho Kim, Seungwon Yang: *Speaker Identification System Using HMM And Mel Frequency Cepstral Coefficient*

[26] Stephen Cook: *Speech Recognition HOWTO*

[27] Joe Tebelskis: *Speech Recognition Using Neural Networks*

[28] Waleed H. Abdulla, Nikola K. Kasabov: *The Concepts Of Hidden Markov Model In Speech Recognition*

[29] Josh Mcdermott: *The Emergence Of Orientation Selectivity In Self-Organizing Neural Networks*

[30] Stan Salvador, Philip Chan: *Toward Accurate Dynamic Time Warping In Linear Time And Space*

[31] J.A. Haigh, J.S. Mason: *Voice Activity Detector Based On Cepstral Analysis*

[32] Adrian Abordo, Jon Liao: *Voice Command Recognition: Robokart*

Index