HINDI SPEECH SYNTHESIS

A dissertation submitted to the University of Hyderabad in partial fulfillment of the requirements for the award of the degree of

MASTER OF PHILOSOPHY

IN

APPLIED LINGUISTICS

BY

Sumedha Seth

Centre For Applied Linguistics And Translation Studies

School Of Humanities

University Of Hyderabad

Hyderabad-500 046

December 2002

OM

For Mom and Dad

&

For all those who love me…

Acknowledgements

This dissertation is my first leap in this vast sea of Computational Linguistics. It couldn't have been done, but for the support I have received from so many people. I don't have words to thank them though, but I would still extend a little note for all the help they have given me, academic or otherwise.

First and foremost I would like to thank my supervisor Dr. Gautam Sengupta, who has been a source of constant support and guidance throughout my work period.

I extend very special thanks -

to all my professors here in the University of Hyderabad, especially Prof. Probal Dasgupta, for an ever-ready helping hand.

to all the staff-members of CALTS, Apparao garu, Murthy garu, Anand garu and all, for their assistance and advices from time to time in all academic affairs.

to all my professors at Jawaharlal Nehru University, New Delhi (where I did my M.A. in Linguistics), for the essential basics of Linguistics I have learnt from them.

to Masum for his support and contribution from day one in Hyderabad. He has been a source of reference and help both in matters related to academic and otherwise.

to Soma di, whose contribution can neither be measured nor can it be explained here in details.

to Cruz, who has always guided me even when he was away, with his comforting mails.

to Ram Babu, Sai Prasad for helping me out with my first introduction to Linux and Festival and for lending me a helping hand immediately, whenever I have asked for it.

to Hari, who has attended to all my errands and have spent hours helping me install and reinstall Linux and giving me help when I needed it most.

to Jithendra Vepa, whom I have never met, but who has helped me solve one of my most puzzling problems over the Internet.
to Rahul and Rohit, for clearing out my doubts.

to David Huggins Daines for helping me find out solutions to my problems and other people on the Festival mailing list who have promptly replied to my queries.

to Vivek, for cheering me up whenever I lost hope, for listening to all my cribbing and still being patient with me.

to all my friends at JNU and HCU who have supported me at all times. to Mani bhaiya and Vicky for helping me whenever I needed it.

I am very grateful, to all those people who could not have helped me with my work directly, but have been a constant support otherwise, without which, I couldn't have accomplished this. I extend my regards -

to my Tayaji-Tayiji, and Chachaji-Chachiji for all their love, blessings and support through the years when I was away from my parents, for teaching me small things in life that always helps.

to Sheetal didi, who has stood by me in every moment since I have come to Hyderabad. There are no words to thank her for what she has done for me. "Thank you didi, for everything". (I can't say anything better).

to my Bhaiya, for being my friend, philosopher and guide throughout and advising me in making all the important decisions. I can say just one thing, "Thank you so much. I have always counted on your support."

to my Bhabhi, who is new to our family yet so much a part of our family, for her assurances and confidence in me.

to my Mom and Dad, "I can never express it but I owe everything to you. Your love, blessings, hope, moral support and guidance has given me strength to overcome every hurdle and confidence to perform my best."

INDEX

[A CD has been submitted along with this dissertation, which has the Hindi voice built on the Festival software “festvox_hcu_hin_sms_lpc.tar” and a copy of this dissertation. The Index file is well linked with the contents in the index.]

Preface

Computational Linguistics is a platform where the "*twain do meet*" – Linguistics and Computers. Each is incomplete without the other. If we only work towards developing programs and codes for automating the process and forget the appropriate language input it may need, all we would be left with is good work with awful output. Same holds true if we only work towards analyzing language ignoring the computational work, because then all we will be left with is good analysis but with no practical use. One has to absorb both equally well. One may not specialize in both computers and linguistics, but at least have a basic understanding on how they work together to meet the same end.

The present work, Speech Synthesis in Hindi, aims at developing a diphone database for Hindi speech synthesis with the help of Festival Speech Synthesis software and is very much computational in nature. It works on a platform where computers and linguistics (phonetics and phonology) go hand in hand. In the present work, I have looked at the whole picture from a Linguistics perspective as to which phone should be included and why, how to deal with nasalization, etc. However, I have not ignored the computational aspect and have also explained the programs involved in the process of building a diphone database and what changes need to be made in the files for synthesis process.

From a computer science perspective, this work may seem not-so-important-to-discuss kinds, but it aims at explaining the step-by-step process, challenges, problems and suggestions. The chapters have been explained in simple terms to assist them who do not have an idea about the nitty-gritty details of speech synthesis and have no guidance as to where to begin.

At times is does not matter what you do but how you do it and why you do it. There is no dearth of research in this field but still we will see that Indian languages are very starved of the much-needed research. In this work I have made my best possible efforts to develop a basic database for Hindi speech synthesis. This is just a preliminary work and I would like to develop it further in future if possible.

Sumedha Seth
Hyderabad, 2002

Introduction

A Text-to-speech synthesis system converts a given text into speech. In simpler words, it is a system that “reads” out any given text. The input to such a system is a written text (computer readable) and the output is synthesized speech. The present work aims to develop a part of this entire synthesis system.

A text-to-speech synthesis system can be divided into two steps – text analysis and speech synthesis. Text analysis parses the text into various categories, disambiguates if necessary and speech synthesis converts this parsed text into synthesized speech. In other words, text analysis helps in “understanding” a given text and speech synthesis “reads” it.

The present work aims at developing a diphone database for speech synthesis in Hindi. This work has been done based on the approach used in Concatenative Speech Synthesis, according to which, co-articulatory effects are essential for continuous speech and they never go over more than two phones. Hence, if we “preserve” these co-articulatory effects or transitions and concatenate them we can get close to human-like synthesized voice.

The present system works with the help of Festival Speech Synthesis software. Festival has been developed at CSTR, Edinburgh University and provides a platform to build an entire text-to-speech synthesis system.
This dissertation has been divided into 6 chapters. Chapter 1 is an introduction to the various approaches towards text-to-speech systems, its applications in the real world and various problems that one might face in developing a text-to-speech system. Chapter 2 gives a historical overview of text-to-speech synthesis developments in the past and discusses the commercial speech systems in the market today. Chapter 3 explains the Festival software with its various modules and approaches. Chapter 4 is a detailed description of each file relevant in the process of building a diphone database and also explains each step in detail involved in the same. This chapter not only discusses what has been done but also states why it has been done and how does it affects the entire process. Chapter 5 is a Walkthrough in developing a diphone database in Hindi or any other language. It is based on the lines of a walkthrough in Festvox manual. This chapter has been included with an intension to help the reader to actually experience the entire process provided he/she has the required setup. This chapter lists step-by-step process as has been done in the present work, from compiling Festival to the making of a distribution package of the diphone database. Chapter 6 is the last chapter and explains in detail all the problems faced during the process of building a diphone database. This section has been included to help the reader to

understand the kinds of problems that can prop up in the process. One cannot state a list of all the problems; as such I have included only the ones I have faced and also how they can be solved.

The present work stresses one very important aspect of speech synthesis – the Linguistic input, because the quality of the output largely depends on how organized is the input. I have made my best efforts to look at every minute detail in the process and explain it in this dissertation. I don't claim that this work is complete in the field of speech synthesis but I would certainly say that this work is complete in what it intends to present, that is, *the process of building a diphone database for Hindi speech synthesis with specific emphasis on Linguistic input.*

[A CD has been submitted along with this dissertation, which has the Hindi voice built on the Festival software "festvox_hcu_hin_sms_lpc.tar" and a copy of this dissertation. The Index file is well linked with the contents listed in the index.]

Directory of Symbols

The following is a symbols chart that shows IPA symbols and the corresponding symbols used in the present work (Symbols used) and the Hindi alphabets.

Vowel Chart

IPA Symbols	Symbols Used	Hindi Alphabets
ə	a	अ
a	A	आ
ɪ	I	इ
i	I	ई
ʊ	u	उ
u	U	ऊ
o	o	ओ
ɔ	O	औ
e	e	ए
æ	E	ऐ
ə̃	aM	अं, अँ
ã	AM	आं, आँ
ĩ	iM	इं, इँ
i̇̃	IM	ईं, ईँ
ʊ̃	uM	उं, उँ
ũ	UM	ऊं, ऊँ
õ	oM	ओं, ओं
ɔ̃	OM	औं, औं
ẽ	eM	एं, एँ
æ̃	EM	ऐं, ऐं

Consonant Chart

IPA Symbols	Symbols Used	Hindi Aphabets
k	k	क
g	g	ग
c	c	च
ɟ	j	ज
ʈ	T	ट
ɖ	D	ड
t̪	t	त
d̪	d	द
p	p	प
b	b	ब
ɽ	R	ड़
kʰ	kh	ख
gʰ	gh	घ
cʰ	ch	छ
ɟʰ	jh	झ
ʈʰ	Th	ठ
ɖʰ	Dh	ढ
t̪ʰ	th	थ
d̪ʰ	dh	ध

IPA Symbols	Symbols Used	Hindi Aphabets
pʰ	ph	फ
bʰ	bh	भ
ɽʰ	Rh	ढ़
n	n	न
m	m	म
ŋ	ng	ङ
ɲ	ny	ञ
ɳ	N	ण
j	y	य
ɾ	r	र
l	l	ल
ʋ	v	व
s	s	स
ʃ	sh	श
ʂ	S	ष
h	h	ह
z	z	ज़
f	f	फ़

Chapter 1
Text-to-Speech Synthesis – ***An Introduction***

Text-to-Speech (TTS) is a computer-based system that reads out any machine-readable text automatically. The idea is that the computer should be able to automatically "speak" new sentences. A Text-to-speech system does not store all the words of a language but works on a system of grapheme-to-phoneme transcription of the written text. The entire synthesis procedure consists of two main phases: text analysis, where the written text is transcribed into phonetic form and the second is generation of speech waveforms, where the acoustic output is produced on the basis of phonetic information.

These two phases are also termed as *Natural Language Processing* (NLP) and *Digital Signal Processing* (DSP). NLP performs the text analysis, grapheme-to-phoneme transcription, prosody generation, morpho-syntactic analysis and syntactic processing. While DSP generates waveforms from the output of the NLP module, that it reads out and is also called speech synthesis.

NLP reduces an orthographically represented sentence into its internal structure (words, phrases etc.), identifies the part of speech of each word, extracts morphological information, describes words on the basis of other words and disambiguates them contextually. This step is very necessary because of the complexity of language.

For instance, words like 'conduct' have stress on the first or the second syllable depending on its part of speech category and the numbers like 1978 can be pronounced as '*nineteen seventy-eight*' or '*one thousand nine hundred and seventy eight*'. The choice depends on the context. Thus, contextual disambiguation has a crucial role to play. NLP also determines the phonetic transcription of the text through letter-to-sound rules and online dictionaries. Even here the main challenge comes from the pronunciation ambiguities. For example, "read" can be pronounced as both /red/ and /ri:d/ depending on the text. Such ambiguities can be resolved only by determining the context. Another aspect of NLP is prosody generation that organizes sentences into intonational phrases to make the synthesized speech sound closer to the human speech.

The final step in speech synthesis is waveform generation, which is done on the basis of series of rules that formally describe the influence of phonemes on one other.

Speech Synthesis can be achieved in two ways:

A. With the help of rules that describe the influence of phonemes on one another;

B. By storing examples of phonetic transitions and coarticulations into a speech segment database, and using them just as they are, as ultimate acoustic units (in place of phonemes).

A text-to-speech system has two main alternative synthesis philosophies: *synthesis by rule* and *synthesis by concatenation*, based on very divergent means and objectives.

Rule-based Synthesis

This works on a speech corpus that is a digital recording of consonant-vowel-consonant (CVC) sequences, read by a professional speaker. A rule-based synthesizer takes a speech corpus and performs speech analysis and gets a parametric form. This parametric form is then put through a rule finding and a rule matching stage. Finally a speech signal is produced that is synthesized into synthetic speech.

The speech corpus represents many transitions and coarticulations that help in finding rules for synthesis. A speech analyzer gives a parametric form to the digital speech data and separates the contributions of glottal folds and the vocal tract and presents the information in a compact way for further study. The speech *model* used in analysis, constrains the type of signal to be analyzed. The quality of synthesized speech largely depends on the efficiency of the speech model during analysis.

Then comes the rule finding stage. The speech data from all the speakers is then studied to get the rule form and the actual parameters (like duration or formant frequencies) are set for one speaker because the speaker specific parameters are not significant. The rules model the general articulatory features. Thus a large number of rules are gathered as a rule database by trial and error. Then synthesis begins by rule matching, where these large number of rules are matched to the phonetic inputs. The result is a parametric speech signal that is finally transformed into digital speech by a synthesizer.

The speech quality of the synthesized speech can be affected by two kinds of errors —*intrinsic errors* and *extrinsic errors.* Intrinsic errors depend on the peculiar model chosen during analysis, i.e. if the model is too simple, it may overlook the intricate speech transitions and may lead to distortions. Extrinsic errors depend on algorithms used to fix parameters and how correct are the parametric representations. These errors can be overcome by intensive trial and errors.

Rule-based synthesizers appear as *formant synthesizers* that describe speech up to 60 parameters related to formant and antiformant frequencies, bandwidths and glottal

waveforms. One can achieve high-quality speech based on formant analysis, as they are free of intrinsic errors. However there are certain problems. The large number of parameters complicates the analysis stage and result in more number of extrinsic errors and one needs a lot of trial and error to estimate the formant frequencies and bandwidths.

Still the rule-based synthesizers are a powerful approach to speech synthesis. Synthesis-by-rule facilitates speaker-dependent voice features such that switching from one voice to another is possible with the help of rules. It also reflects upon the articulatory aspects of changes in speaking styles.

Due to its potential, rule-based synthesis is widely used by various text-to-speech systems such as MITALK, JSRU synthesizer, the multilingual INFOVOX system, and the Klatt synthesizer.

Concatenation-based Synthesis

This works by concatenating sound segments such that the coarticulatory features are preserved within the segments. The knowledge about the transitions is embedded in the segments. The most crucial point in concatenation-based synthesizers is the choice of segments. If the segments are too large the size of the database will also increase and if the segments are too small then there will be more number of concatenation points that will hamper the smoothness of the synthesized speech. The segments have to be chosen to result in minimum concatenation problems and manageable size.

Concatenation based synthesis seeks a database of segmented sound units with the corresponding segmental information like duration, pitch, stress etc. These segmented units are then given a parametric form and temporal value. To produce a synthesized utterance these segmented units are concatenated as per the mapping of graphemes of the language to the phonemes. Prosody matching has to be taken care of during the process of synthesis. This is synthesis by concatenation.

On the bases of the two philosophies, i.e. *synthesis-by-rule* and *synthesis-by-concatenation*, text-to-speech synthesis systems can be divided into three main kinds – Articulatory synthesis, Formant synthesis and Concatenative synthesis systems.

Articulatory Speech Synthesis

Articulatory Speech Synthesis is a rule-based synthesis system, which attempts to model the human speech production system directly. This method involves models of human articulators and vocal cords and tries to model the human vocal organs as perfectly as

possible so it is potentially the most satisfying method to produce high-quality speech. It produces a high quality synthetic speech as it works on the basis of the functioning of human vocal organs. Though the speech quality produced by this speech synthesis system is high, so is its difficulty level because the computational load is higher. As such the implementation of this method is difficult. Therefore, this method of producing synthetic speech is not very popular and successful as others.

Human speech production system involves vocal cords, and articulators between the area of glottis and mouth (up to lips). Speech sounds are produced by various positions of the articulators in the mouth such as lips, tongue, palate, velum etc. The Articulatory model is based on the functions of vocal tract area, such as, lip aperture, lip protrusion, tongue height, velic aperture, vocal cord tension, lung pressure etc.

A general Articulatory speech synthesis can be done in *five* steps:

1) Determination of the vocal tract parameters like - angle of jaw opening, tongue center position, tongue-tip position, lip rounding, height of the hyoid and state of the velum.

2) Calculation of the 2D vocal tract outline and area function: The images show the midsaggital section of the vocal tract as taken from the X-ray images. However there is a drawback in this. The model is derived from the X-ray analysis of natural speech, which is 2-D while vocal tract is naturally 3-D. Hence it is difficult to map the motions of the articulators and properly define the complicated movements of the tongue. However, this helps to calculate the area of oral cavity, the nasal cavity and the pharyngeal cavity.

3) Next step is to calculate the vocal tract transfer rate in the frequency domain. In one technique, this is done with the help of 2x2 chain matrices.

4) The next important step is to calculate the excitation function by taking an approximation of the cross-sectional area through glottis.

5) Last step is to transform the information into the speech wave. Multiplying the transfer spectrum of the vocal tract with the excitation spectrum of the glottis finally yields the spectrum of the sound wave

All steps from 1 to 5 have to be repeated for each glottal pulse.

Speech has been synthesized by Articulatory method on the basis of various models and competing theories, such as, real time models, performance models, tube model, speech production model, the distinctive region model, and the hybrid model to name a few. The

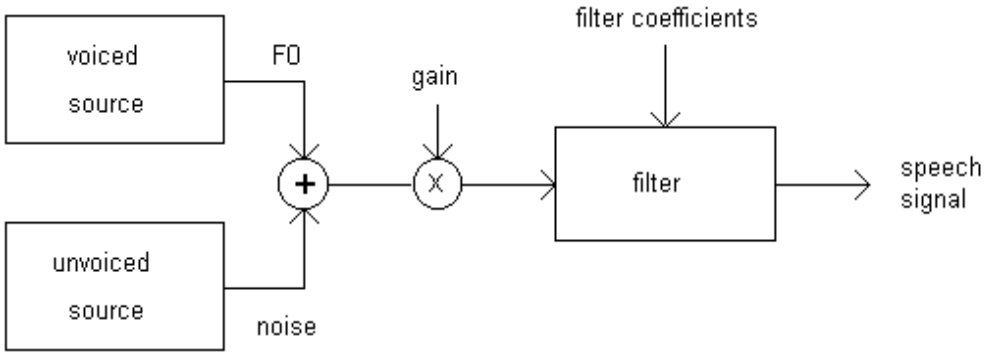
real time models contain notional time based on real time taken by the human organs to produce sound while performance models are based on an abstract time other than the notional or real time. The speech production model is based on the model of speech production that is linguistically dominated. The hypothesis is temporally adhered to that motor control dominates the syllabification of segment sequences.

Formant Synthesis

Formant synthesis is another rule-base synthesizer. It is based on the source-filter model approach. Formant synthesis models the source sounds and the formant frequencies and describes speech on the basis of formants, bandwidths, and glottal waveforms. The vocal tract (the throat from the vocal cords to the lips) has certain major resonant frequencies. These frequencies change as the configuration of the vocal tract changes, like when we produce different vowel sounds. These resonant peaks in the vocal tract transfer function (or frequency response) are known as "formants". It is by the formant positions that the ear is able to differentiate one speech sound from another.

Formant synthesis can provide infinite number of sounds that makes it more flexible. The excitation signal could be either voiced with fundamental frequency (F0) or unvoiced noise. A mixed excitation of these two may also be used for voiced consonants and some aspiration sounds. The excitation is then gained and filtered with a vocal tract filter that is made of resonators similar to the formants of natural speech. For intelligible speech three formants are required and for high quality speech up to five formants. Each formant is modeled with a two-pole resonator that enables formant frequencies and bandwidth.

Formant synthesis has been one of the most popular methods in the last few decades because of the quality of output it produces. It is not so difficult as articulatory synthesis and still it can produce a large number of sounds.



Source-filter model of speech.

Formant synthesis has two basic structures – parallel and cascade. Usually a combination of these two is required for a good quality output. The parallel formant synthesizer has resonators connected in parallel. This helps in controlling bandwidth and is useful for

nasals, fricatives, and stop-consonants. A parallel formant synthesizer specifies formant levels (peak amplitudes). In cascade formant synthesizer, the band-pass resonators are connected in series such that output of each is the input for the next one. It specifies the formant center frequencies and bandwidths. It is better for vowels and vowel-like sounds.

Concatenative Synthesis

Concatenative synthesis is based on a concatenation-based synthesizer. In this type of synthesis prerecorded samples are derived from natural speech and concatenated to produce synthetic speech. The graphemes of a language are mapped onto the phonemes in the database to account for text-to-speech conversion. This is the most convenient type of synthesis method and as such is increasing in popularity.

However, there are some limitations to this method – one it is limited to one speaker and the quality of voice once recorded is fixed and does not have much scope for variation, two, it requires more memory capacity than the other two types of synthesis methods. The most salient feature of human voice is not just sounds but how sounds are uttered in succession. As such the transition between two sounds is what gives the smoothness to speech.

In concatenative speech synthesis, the idea is to capture this transition in the form of recorded speech units, which can then be concatenated to get continuous speech. The quality of synthesis largely depends on the length of units used for concatenation. For convenience I will refer to them here as “units of concatenation”. There are longer and shorter units and one has to make a choice between them. Longer units capture the transition very well but require large amount of memory while shorter units have more concatenative points, which hamper the smoothness of continuous speech. This method requires a database of prerecorded natural utterances segmented into units such as words, syllables, demisyllables, phonemes, diphones, and triphones.

Words are used as concatenating units in very specific cases of synthesis generally in limited domain synthesis, where vocabulary required for synthesized speech is limited to some special use. Synthesis of words brings about very high naturalness and is comparatively easy as it captures the coarticulation effects within a word. However word is generally not a preferred unit because there are innumerable words in a language and a lot of them are proper words. To record all of them is not practically possible. Hence, words are not suitable units for unrestricted text-to-speech system. But words can be used more suitably for very restricted usage such as a talking clock or some announcement system where the output will be more natural speech.

Syllables are another unit of concatenation that comes under the category of longer units. A syllable is the smallest unit that is pronounced in one breath. The number of syllables in a language is smaller than the number of words, but still the database is too large for a text-to-speech system. For instance, English has 10000 syllables. The syllable database is not very reasonable.

At present there is no word or syllable based full text-to-speech synthesis system. Most of the systems are based on demisyllables, phonemes, diphones, and triphones or a combination of these.

Demisyllables are units that represent initial and final parts of syllables. Demisyllables hold an advantage on syllables that it takes about 1000 demisyllables to construct 10,000 syllables. They require less concatenation points and take into account most of the coarticulation effects. They also cover allophonic variations. But the main problem with demisyllables is that it is not possible to calculate the exact number of demisyllables in a language like phonemes and diphones. As a result if the system has demisyllables database only, it wont be able to synthesize all possible words in a language. The memory required by demisyllables is also quiet large though is less than syllable and word database.

Phonemes are the linguistic representation of speech and hence the most commonly used units in speech synthesis. They generally support rule-based systems. The number of phonemes in a language varies around 40 - 60 and so phonemes are most reasonable units in terms of the size of the database. However, phonemes do not capture the coarticulation effects and the synthesized speech may sound very mechanical. Phonemes can be used in a “mixed” unit synthesizer where various units are used in the same synthesis system as in unit selection.

Diphones are a list of all possible phone-phone combinations in a language. Diphones are units that begin in the middle of the stable state of a phone and end in the middle of the following one. They aid speech synthesis by capturing the transitions between the two phonemes and the coarticulatory effects. The maximum number of diphones in a language is the square of the number of phones of that language. However a language may not allow all possible combinations of phones and certain combinations may not exist. So the actual number of diphones may vary depending on the language. Diphones have been discussed at length in Chapter 4.

There are longer segmental units also like ***triphones***. Triphones are like diphones except that they include a complete phone between two half phones or steady states (half

phoneme – phoneme – half phoneme). The number of triphones in a language is very large and they require more concatenation points. As such the quality is not that natural sounding as with diphones.

The units of concatenation can be any or many for the speech synthesis process. The recorded utterances are segmented into the unit length, which is then stored as a database. In text-to-speech, letter-to-sound rules convert the text into the phonemes and map them to the units in the database. Concatenative synthesis has a limitation here, unlike articulatory or formant synthesis, that is, concatenative synthesis is limited to one voice and the units in the database only.

Applications of Speech Synthesis

The application field of Text-to-speech (TTS) synthesis is expanding fast and has ever increased with the improving quality of synthesized speech. The speech synthesis systems have now become affordable and easy-to-use for everyday purposes. The day is not far when TTS will be as much a part of our lives as a telephone or computer.

Speech synthesis has its utilities in various fields depending on the requirement and application. It may be used in several applications from announcements, warning systems to reading machines and e-mail readers. The vocabulary of synthesized speech depends on its application. For instance, vocabulary in announcements or talking clock is limited.

Along with the vocabulary, the quality of the synthesized speech also depends on the extent of application. The result is the various types of speech synthesis - Limited domain synthesis, which is used for announcements and Unit Selection for high quality text-to-speech synthesis and wider application. Though TTS has its utility in varied fields but it is a boon for people with communicating difficulties. For instance, reading machines for the blind or electronic mail readers can help the visually handicap to lead more independent lives.

The application of Speech synthesis is ever increasing with the upcoming new technology but to name a few it has its major application in the following fields:

1. Applications for the visually impaired:

One of the most beneficial applications of speech synthesis is for the visually impaired people for whom the computer screen can be converted into speech by way of the reading and communication aids for the blind in the form of TTS. Earlier, books were recorded on audiocassettes in the form of audio books and it was the only source of information for the blind community but to record everything on tapes is practically impossible especially with

the endless amount of reading material in the market. It is a very time consuming and expensive process. As such the visually handicap were generally deprived of the vast ocean of knowledge. TTS has appeared like a boon in many situations. It is also easier to get information from computer with speech instead of using special bliss symbol keyboard, which is an interface for reading the Braille characters.

For this particular application, few important factors for synthesized speech are quality, naturalness and speech intelligibility. It is also important to maintain the speech rates ranging from half to at least three times the normal rate. Nowadays the quality of synthesized speech has reached acceptable levels and speech synthesis systems have become more affordable.

The first commercial speech synthesizer was introduced by Raymond Kurzweil in the late 1970's, which consisted of an optical scanner and text recognition software and was capable to produce quite intelligible speech from written multifont text However, the first reading machines were far too expensive for the average user and were available only in libraries etc. With the fast pacing technology, speech synthesis systems have become easy to handle.

It is easy to construct a reading machine with a scanner and OCR (Optical Character Recognition) system. A “reading system” or a “screen reader” is software that runs alongside other programs capturing whatever is displayed on the screen. It can work with any other software. Speech recognition along with speech synthesis can become a complete computer-human communication system for visually handicap.

2. Speech Synthesis for the vocally handicapped

Those people who have speaking difficulties may require personal communication. People who are born-deaf or have hearing disorders also cannot learn to speak properly. It is also termed as “Augmentative Communication”. Through the speech synthesis technology, the user can specify utterances to communicate to the other person.

Synthesized speech gives the deafened and vocally handicap to communicate with people who cannot understand sign language. The medium of communication is synthetic speech that is replaced by real speech. For this application, the quality of voice is a very important criterion.

Talking heads are one means to improve the quality because of the inability of synthesized speech to convey emotions such as happiness, sadness, urgency, friendliness etc. One such tool that helps user to express their feelings is HAMLET (Helpful Automatic Machine

for Language and Emotional Talk). However communication through synthetic speech is generally slow with keyboards.

Speech Synthesis has expanded and improved in past decade but it still has to come a long way to facilitate for applications that replace human voice in terms of quality, intelligibility and rate of speech.

3. Speech synthesis in education

The educational applications of speech synthesis are also widespread. A computer with speech synthesizer can teach endlessly for hours. It can be programmed for special tasks, for example, for teaching spelling and pronunciation for different languages, for interactive educational applications, in games etc. Speech synthesis can also facilitate unsupervised training in a very inexpensive way.

Speech Synthesizer can also aid in proof reading especially if it is connected with a word processor as one can easily detect grammatical and stylistic errors when listening to the written text. Normal misspellings are also easier to detect.

4. Speech synthesis in Telecommunications and Multimedia

Another widespread application of speech synthesis is in the field of telecommunications and multimedia. We can find one such instance in the telephone inquiry systems. This application is catching up fast with the improving quality of the synthesized speech.

Speech synthesis is also used in electronic-mail reading. There are facilities where a person can listen to e-mail messages via a telephone line.

Synthesized speech can also be used to speak out Short Messaging Service (SMS) in mobile phones. In fact, one can develop an entirely interactive multimedia application of speech synthesis by pairing it up with speech recognition.

5. Other Applications

Speech Synthesis has scope for application in practically all human-machine interactions. For instance, it can be used in warning and alarm systems to give the information, in a talking clock, to get desktop messages and warnings from computer such as printer activity or received e-mail. Speech synthesis has its scope as language interpreters, other communication systems like videophones, video conferencing, mobile phones etc.

During last few decades the communication aids have been developed from talking calculators to modern three-dimensional audiovisual applications. The application field for

speech synthesis is becoming wider all the time, which also brings more funds into research and development areas.

Text-to-speech – ***the problem areas***

One of the main challenges of speech synthesis is to convert input text into phonetic output, i.e., grapheme-to-phoneme conversion. The extent of difficulty is language specific depending on how well the graphemes and phonemes of a language map. Graphemes of a language can map on to zero, one, two or maybe three phonemes. The more complex this conversion, the more sophisticated rules are required for it.

One of the most frequently encountered problems in text-to-speech synthesis is encountered in text processing, which is a very complex task and is specifically language dependent. The most problematic areas in this are numbers, acronyms and abbreviations. For example, 1958 can be pronounced as "*nineteen fifty-eight*" for representing a year and "*one thousand nine hundred and fifty eight*" for currency.

Similarly problems are encountered while distinguishing dates and fractions as 5/12 can be pronounced as "*five-twelfth*" for fraction and "*twelfth May*" or "*fifth December*" for date. Ordinal numbers also pose a problem for speech synthesis, the first three ordinals in English are expanded differently, 1st as *"first",* 2nd as *"second",* and 3rd *"third"*.

Abbreviations are another problematic area in speech synthesis mainly because they can be pronounced differently, as full words, as written or letter-by-letter. For instance "*St.*" can be used for "*saint*" or "*street*". Abbreviations can also be contextual, for example, "ft" can be "*foot*" or "*feet*", "kg" can be "*kilogram*" or "*kilograms*" depending on the singular or plural number. There are some abbreviations that are pronounced as full words, e.g., RAM (Random Access Memory) or letter-by-letter as SMS (Short Messaging Service). Special characters and symbols such as "$", "%", "&", "/", "+", "-", "@" also cause problems in text processing depending on the context they are used in.

Another problematic area in speech synthesis is to find correct pronunciation especially in the case of homographs. (Homographs are words that are spelled the same way but they have different meanings and pronunciations). For example, "lives" can be pronounced as /laivs/ or /livs/, "lead" can be /led/ or /li:d/.

The pronunciation may also vary according to the part of speech as in English "conduct", which is pronounced with stress at first and second syllables depending on whether it is a

verb or noun. Pronunciation of "the" also varies depending on the following sound. It becomes "thee" when preceding a vowel and "the" when preceding the consonant.

Pronunciations of certain sounds change in context from voiced to unvoiced and vice-versa depending on the context and/or adjacent sounds. Pronouncing proper names is also a challenging task especially when they are borrowed from other languages or are pronounced ambiguously. To overcome such problems, it is advisable to include these kinds of words in a dictionary.

Another challenging area in speech synthesis is prosody, that is, correct stress, intonation, duration etc. Also called the suprasegmental features and may be considered as the melody, rhythm, and emphasis of the speech at the perceptual level. Prosody is necessary to make the synthetic speech sound more natural as it largely depends on the meaning of the sentence, the intension of the speaker and speaker characteristics and emotions. The input text has to be segmented into prosodic units but in written text prosody is not marked by punctuation, hence this becomes a problem. For example the Hindi sentence "*roko mat jAne do*" "*stop don't let go*" can be interpreted in two ways – "*roko, mat jAne do*" "*stop, don't let go*" or "*roko mat, jAne do*" "*don't stop, let go*". The meaning is reversed due to the prosodic phrasing. To capture these differences is a real challenge at hand.

Many other practical problems that come across in text-to-speech synthesis arise from the method of synthesis employed. Each method has its drawbacks. For instance, in Articulatory synthesis implementation of rules and modeling of tongue movements is complex. In Formant synthesis, controlling of formant frequencies and amplitudes and bringing naturalness to the speech is a major challenge; and in Concatenative synthesis, collecting samples, recording and labeling takes time and requires large memory. Also concatenation points have to be manipulated properly to avoid distortion.

The problems listed here are the general and expected problems faced in the process of synthesis. However there can be many problems during the process. Some problems faced during the present work have been listed in Chapter 6. Those problems also include Festival software related problems.

Chapter 2
Speech Synthesis - ***History and Development***

History of Speech Synthesis

The earliest efforts to produce speech were made over two hundred years ago in 1779 by a Russian Professor Christian Kratzenstien, professor of physiology in Copenhagen, who explained the physiological differences between five long vowels /a/, /e/, /i/, /o/ and /u/ and made apparatus to produce them by constructing acoustic resonators similar to human vocal tract. He activated the resonators with vibrating reeds as in musical instruments. The basic structure of resonators is shown below:

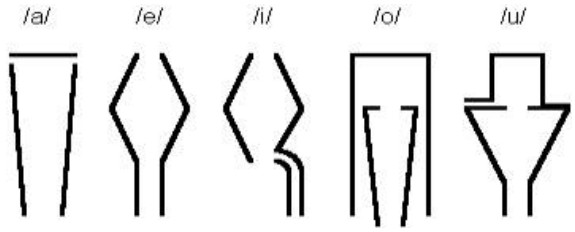


Fig. 2.1 Kratzenstein's resonators (Schroeder 1993).

Later in 1791, in Vienna, Wolfgang von Kemplen introduced his “Acoustic-Mechanical Speech Machine” that could produce single sounds as well as some sound combinations. The machine consisted of a pressure chamber for lungs, a vibrating reed for vocal cords action and a leather tube for the vocal tract action. Vowel sounds were produced by manipulating the shape of the leather tube and consonant sounds were simulated by four separate constricted passages and controlled by the fingers and plosives could be produced on the basis of a model of vocal tract with hinched tongue and movable lips. Kempelen’s studies led to the theory that the main site of acoustic articulation is the cavity between the vocal cords and lips and not larynx, which was considered the center of speech production.

In 1800’s, Charles Wheatstone constructed a more complicated version of Kemplen’s speaking machine that could now produce vowels, with a vibrating reed and most of the consonants (even nasals) with a turbulent flow through a suitable passage with reed-off, some sound combinations and even some full words.

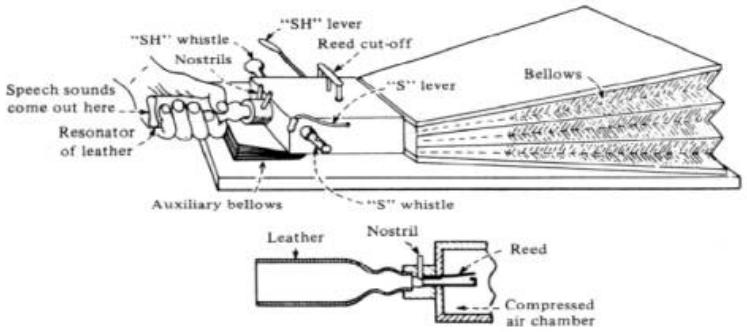


Fig. 2.2 Wheatstone's reconstruction of von Kempelen's speaking machine (Flanagan 1972).

In the 19th century, some additional machines of similar kind were constructed, but there were no really fundamental innovations in the field of speech synthesis. However, the device constructed by Joseph Faber in 1835 can be said to represent some progress in that its speech production mechanism included a model of the tongue and a pharyngeal cavity whose shape could be controlled.

In 1838, Willis found a connection between a vowel sound and the structure of the vocal tract. He discovered that the vowel quality depended on the length of the tube and not its diameter. There were many researches and experiments conducted with the help of mechanical synthesizers till 1960’s but with no significant success.

From the second decade of 20th century we saw the introduction of electrical synthesis device to produce synthetic speech and from then on progress in speech synthesis has come a long way. In 1922, Stewart introduced an electrical synthesizer that modeled the acoustic resonance of the vocal tract but could produce only single static vowel sounds with two lowest formants and no consonants or sound sequences. Wagner made a similar device, which had four electrical resonators connected in parallel with a buzz-like excitation source. The output of the four resonators was combined to produce vowel spectra.

In 1932 Japanese researchers Obata and Teshima discovered the third formant in vowels, which was considered sufficient for intelligible synthetic speech.

At the beginning of the 20th century, the progress in electrical engineering made it possible to synthesize speech sounds by electrical means. The first device of this kind that attracted the attention of a wider public, was the *VODER* (Voice Operating Demonstrator), developed by Homer Dudley and presented at the World Fair in New York in 1939. It was the first speech synthesizer that was recognized by the scientific world as a pioneer effort in producing intelligible speech artificially. Developed at Bell Laboratories in mid-thirties, it was based on source-filter-model of speech and was inspired by VOCODER (Voice Coder). VOCODER was a device that analyzed speech into slowly varying acoustic parameters, which could then drive a synthesizer to reconstruct the approximation of the original speech signal.

The VODER consisted of a wrist bar for selecting a voicing or noise source and a foot pedal to control the fundamental frequency. The source signal was passed through ten band-pass filters whose output levels were controlled by fingers. It took considerable skill to play a sentence on the device. The speech quality and intelligibility were far from good but it well demonstrated the potential for producing artificial speech such that people got more interested in speech synthesis.

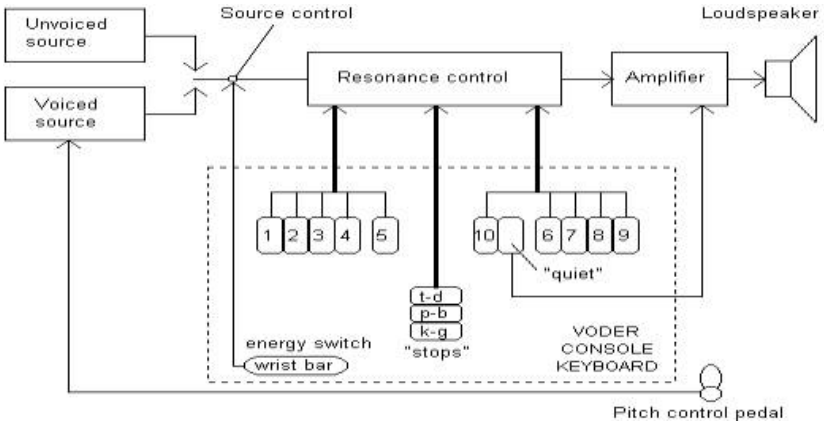
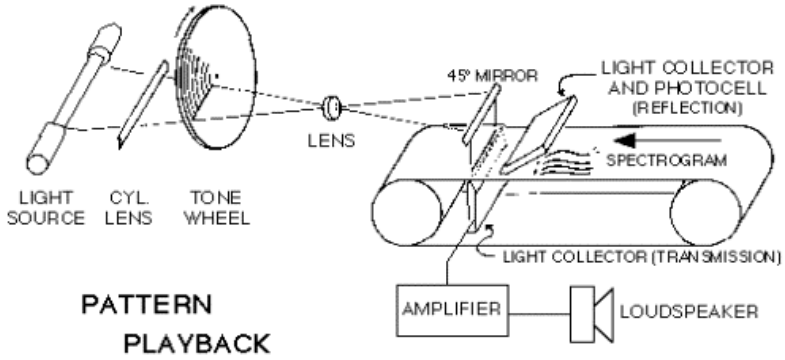


Fig. 2.3. The VODER speech synthesizer (Klatt 1987).

In 1951, Franklin Cooper developed a Pattern Playback synthesizer with his associates at Haskins Laboratories, which 'read' or reconverted recorded spectogram patterns, recorded optically on transparent belts, into original or modified sounds. This device served mainly the investigation of speech perception. It worked like an inverse of a sound spectrograph.



In the models that were developed by several researchers since the 1950-ies, an electric source signal is passed through a filter. The source signal is either a harmonic tone, as in the voiced speech sounds, or an aperiodic noise, as in the unvoiced segments.

The filter serves the purpose of simulating the resonance properties of the vocal tract. There are two distinct approaches that have been tried. In one of these, articulation is simulated with a large number of circuits connected in cascade. Each of these circuits represents a short section of the vocal tract (5 mm or so), whereby the cross sectional area of each section is crucial (*transmission line analog*). The other method uses resonance circuits to simulate each formant, i.e., the resonance of the vocal tract, irrespective of its shape (*terminal analog*).

In 1953, PAT (Parametric Artificial Talker), the first Formant synthesizer was introduced by Walter Lawrence, which consisted of three electronic formant resonators, connected in parallel. A moving glass slide was used to convert painted patterns into six time functions to control the three formant frequencies, voicing amplitude, fundamental frequency, and noise amplitude. During the same time, Gunnar Fant introduced the first cascade formant synthesizer OVE I (Orator Vebris Electris). In 1962, Fant and Martony introduced an improved OVE II synthesizer, which consisted of separate parts to model the transfer function of the vocal tract for vowels, nasals, and obstruent consonants. PAT and OVE synthesizer brought about a question as to how the transfer function of the acoustic tube should be modeled, in parallel or in cascade. In 1972, James Holmes introduced his parallel formant synthesizer and tuned by hand, the synthesized sentence “I enjoy simple life”. The quality of his synthesis was so good that an average listener could not make out any difference.

First Articulatory synthesizer, the DAVO (Dynamic Analog of the VOcal tract) was introduced by George Rosen at the Massachusetts Institute of Technology (MIT) in 1958. It was controlled by tape recording of control signals created by hand. In mid 1960s, first experiments with Linear Predictive Coding (LPC) were made. Linear prediction was first used in low-cost systems, such as TI Speak'n'Spell in 1980, and its quality was quite poor compared to present systems. Noriko Umeda developed the first full Text-to-speech system for English in Electrotechnical Laboratory, Japan, in 1968. It included a syntactic analysis module. The speech was quite intelligible but monotonous and far away from the quality of present systems.

Allen, Hunnicutt and Klatt at MIT developed MITalk text-to-speech system in 1979, which was later used in Telesensory Systems Inc, a commercial TTS system. Dennis Klatt developed the famous Klattalk system two years later. The technology used in MITalk and Klattalk systems form the basis for many synthesis systems today like DECtalk and Prose-2000.

In 1976, Kurzweil developed the first reading aid with optical scanner. The reading machines for the blind had good quality output and could read even the multifont text but were too expensive for an average customer and so were mostly used by libraries etc.

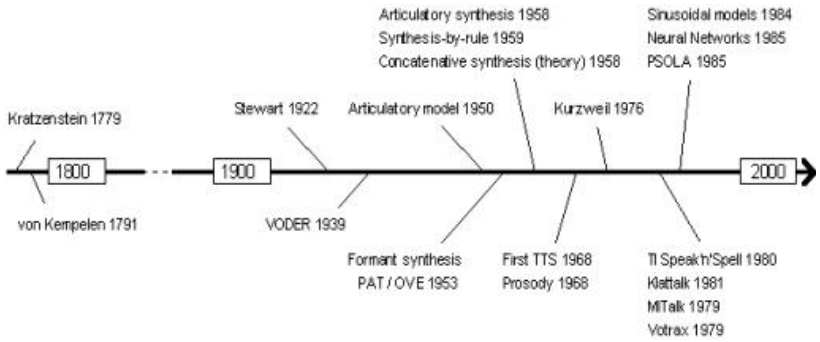
In 1970, the further development of speech synthesis was closely associated with computer technology and now it was

not enough to simulate natural speech production with electric circuits, but these circuits were also just simulated. Computers made it possible to utilize speech synthesis for practical purposes, and several systems were developed with the function of converting text into speech.

The idea is to concatenate stored words or various shorter segments in order to produce speech. However, single speech sounds (phones) cannot be successfully concatenated into words and sentences, since the acoustic properties do not provide a natural sounding speech. Better success has been achieved with so called "diphones", which consist of the second half of one speech sound and the first half of the subsequent. This results in a large number of elements, which have to be carefully selected. With such methods it is possible to achieve a high degree of naturalness, even without having a complete description and understanding of the acoustics of speech production. However, these methods lack the flexibility of synthesis by rule. Rule based synthesis does not make use of any stored segments of speech, but all the properties of the speech signal result from the application of a set of rules.

In late 1970's and early 1980's, a large number of commercial text-to-speech and speech synthesis products were introduced. The first integrated circuit for speech synthesis was probably the Votrax chip that consisted of cascade formant synthesizer and simple low-pass smoothing circuits. In 1978 Richard Gagnon introduced an inexpensive Votrax-based Type-n-Talk system. In 1980, Texas Instruments introduced linear prediction coding (LPC) based Speak-n-Spell synthesizer based on low-cost linear prediction synthesis chip (TMS-5100). It was used for an electronic reading aid for children and received quite considerable attention.

In 1982 Street Electronics introduced Echo low-cost diphone synthesizer. At the same time Speech Plus Inc. introduced the Prose-2000 text-to-speech system. A year later, first commercial versions of famous DECtalk and Infovox SA-101 synthesizer were introduced. Some milestones of speech synthesis development are shown in the figure below:



Some milestones in speech synthesis

Modern speech synthesis technologies involve quite complicated and sophisticated methods and algorithms. At the present, the limits of the achievable intelligibility and naturalness of synthetic speech are no longer set by technological factors, but rather by our limited knowledge about the acoustics and the perception of speech. In research, speech synthesis is used to test this knowledge.

Survey of present speech synthesis systems

This survey will be divided into 4 types:

- Synthesizers
- Text-to-Phoneme Systems
- Text-to-speech systems
- Commercial Multilingual Speech Synthesis systems
- Hindi Speech Synthesis Systems

Synthesizers:

A text-to-speech system consists of a speech synthesizer and a set of rules for translating text into input strings to a speech synthesizer. The following speech synthesizers are available on the net and can be downloaded for free and used for research and/or private use. The manuals are available for these synthesizers, which contain instructions about input format to the speech synthesizer. The two very famous synthesizers are Mbrola and Klatt-type synthesizer.

- ***MBrola***

MBROLA (Multi Band Resynthesis OverLap Add technique) is a high-quality, diphone-based speech synthesizer, provided by the TCTS Lab of the Faculte Polytechnique de Mons (Belgium). This synthesizer aims to obtain a speech synthesis system for as many languages as possible The MBROLA speech synthesizer is available for free for non-commercial, non-military applications. At present, diphone databases exist for the following languages: American English, Brazilian Portuguese, Breton, British English, Dutch, French, German, Greek, Hindi, Romanian, Spanish, and Swedish.

TCTS also provides a speech database labelling software: MBROLIGN, a fast MBROLA-based text-to-speech aligner. It is not a text-to-speech system because it does not accept raw text as input. More information and demos of the different voices and languages and also comparisons between MBROLA and other synthesis methods can be found at:

http://tcts.fpms.ac.be/synthesis/mbrola.html

- ***Klatt-style Synthesizer***

The Klatt synthesizer is a formant synthesizer developed by D Klatt and versions of this synthesizer are also used in some commercial systems. The synthesizer is also available for free, which runs on Unix. It does not contain any text-to-pronunciation rules; hence it is not a text-to-speech system. More information about this synthesizer is available on the following ftp site:

ftp://svr-ftp.eng.cam.ac.uk/pub/comp.speech/synthesis/klatt.3.04.tar.gz
ftp://svr-ftp.eng.cam.ac.uk/pub/comp.speech/synthesis/klatt.3.04.tar.Z

Text-to-Phoneme Systems

The translation from text-to-pronunciation is central to a full text-to-speech system. Only one example for this translation software was found on the web.

- ***Narrator Translator Library***

It is a US English text-to-phoneme translator, implemented as a resident software library, for use with the Amiga Narrator Device. It is an independent replacement for the Commodore-supplied "translator.library" which is a part of the Narrator speech synthesis package. This software was supplied as a standard part of the Amiga operating system software, where it implements multi-lingual text-to-speech. Approximately 700 translation rules are used to create the 'ARPAbet' phonemes. The translation rules, for each language, are defined in a plain text 'Accent' file and there is a provision for selecting the unique languages for text segments by inserting in-line markup codes in the text: e.g. "Hello there! \french{Bonjour} \deutsch{gute morgen}". The Archive for Narrator Translator Library includes 'Accent' files for American English, British English, Swedish, Maori, Finnish, German, Icelandic, Klingon, Polish, Italian, and Welsh languages. This software is functional on all current Amiga systems. More information on this can be found on the following ftp site:

ftp://ftp.doc.ic.ac.uk/pub/aminet/dev/src/tran42src.lha

Text-to-speech systems

A text-to-speech system converts the written text into speech and incorporates both a text-to-phoneme conversion and a speech synthesizer. The quality of documentation, manuals and speech quality differs widely between these systems. All of these systems are available for English; however, some of them can also “speak” other languages. A number of text-to-speech systems can be downloaded from the web but I will be discussing the significant few here.

❖ *Festival*

Festival is the most complete speech synthesis system available for free and includes a comprehensive manual. It offers a general framework for building speech synthesis systems as well as includes examples of various modules. Festival is multi-lingual and offers a full text-to-speech in English, Spanish and Welsh though English is the most advanced.

The system is written in C++ and uses the Edinburgh Speech Tools for low-level architecture and has a Scheme (SIOD)-based command interpreter for control.

This system is ideal for research, educational and individual use and one can build one’s own synthesis system with a lexicon containing pronunciation of words and a speech database with fundamental frequency data. The speech database is used to build up a PSOLA diphones database. It is also possible to use an MBROLA database, though the synthesis quality will be slightly lower. It has externally configurable language-independent modules: phonesets, lexicons, letter-to-sound rules, tokenizing, part of speech tagging, intonation and duration. Festival can be installed on a Unix machine, on Linux, and even Windows.

The latest details and a full software distribution of the Festival Speech Synthesis System are available through the Festival home page at

http://www.ctr.ed.ac.uk/projects/festival.html

❖ *IPOX*

IPOX is an experimental, all-prosodic speech synthesizer, developed by Arthur Dirksen, IPO, the Netherlands, and John Coleman, Oxford University, UK. Its preliminary version is available for downloading from the web. IPOX can be installed and used on a PC running Windows 3.1 or higher. Sound output requires a 16-bit Windows-compatible sound card, such as the Soundblaster 16. However, the current version of the documentation (in the form of Windows Help files) is incomplete.

IPOX can be accessed on the web at:

http://www.tue.nl/ipo/people/adirksen/ipox/ipox.htm

❖ *rsynth*

It is a public domain text-to-speech system assembled from a variety of sources and supports CMU and BEEP format dictionaries and now also uses stress marks in the dictionary to synthesize intonation. rsynth can be downloaded from the web at:

ftp://svr-ftp.eng.cam.ac.uk/pub/comp.speech/synthesis/rsynth-2.0.tar.gz

A short description and a manual with the demonstration of the synthesis can be found at

http://wwwtios.cs.utwente.nl/say/

❖ *PlainTalk*

This is Apple's text-to-speech system extension that enables applications to perform text-to-speech conversion. The Speech Manager runs on most Macs, but PlainTalk (and the high-quality voices) requires a 68020 Mac or higher. It is available at anonymous ftp from:

ftp://ftp.support.apple.com/pub/apple_sw_updates/US/Macintosh/System/PlainTalk 1.4.1/

The current release (PlainTalk 1.5) contains the English Text-To-Speech with about a dozen voices (English_Text-to-Speech.), Mexican Spanish (Mexican_Spanish_TTS.), and the English Speech Recognition software.

http://www.speech.apple.com/

❖ ***WinSpeech***

WinSpeech is a text-to-speech application that reads text and produces speech to the audio output. It features basic text editing tools, a talk from editing window, a DDE server that allows other Windows applications to send text for talking, coach mode for providing audio instructions throughout the program, and dictionary editing tools for customizing pronunciation. WinSpeech is shareware that is produced by PCWholeWare

WSPLIB text-to-speech DLL is a speech functions library for developers. One can access it at:

http://www.pcww.com/index.html

❖ ***"Speak" - a Text-to-Speech Program***

It is a Text-to-speech program based on concatenation of pre-recorded speech segments. A function library can be used to integrate speech output into other code.

"Speak" is available by ftp from:

ftp://wilma.cs.brown.edu/pub/speak.tar.Z

Commercial Multilingual Speech Synthesis Systems

There are many speech synthesis systems for different languages. This section lists some well-known commercial multilingual speech synthesis systems and does not list all existing systems. The information is mainly taken from the product descriptions found on the Internet.

❖ ***Apple - PlainTalk***

It is a Text-To-Speech system for Mac computers and can produce English and Mexican Spanish text-to-speech. It reads text files and the alert messages from the computer. There are 26 different voices to choose from, including 4 specifically designed for Mexican Spanish Text-to-Speech. It also includes an adjustable voice rate. It is included in many Mac computers, and can also be downloaded from the web:

http://www.apple.com/macos/speech/

❖ ***BaBel***

Offered by BaBel Technologies, it is a top-class speech synthesis thanks to the Multi Band Resynthesis OverLap Add technique (MBROLA). This new synthesis technology was patented in 1996 and received the European Information Technology Prize the same year for its innovative approach to concatenative speech synthesis. It is a high-quality speech synthesis at low CPU cost. This new generation of high-quality speech synthesizers does not constantly remind the listener that the speaker is a machine. A tailored time-domain diphone concatenation algorithm smoothly produces words. It is the first to allow spectral smoothing while maintaining very low computational costs. It does not require DSP and a standard Pentium 100 can run the synthesizer 20 times faster than real-time.

The MBROLA technique uses language and speaker-dependent diphone databases to produce any sentence in a given language with a given voice. The MBROLA speech synthesizer is available for English, German, French, Dutch, Brazilian Portuguese, Spanish, Swedish and Romanian. Other languages are currently under development.

More information about BaBel can be found on the web at:

http://www.babeltech.com/html/frame/synthesis/fintrosyn.html

❖ *BeSTspeech*

BeSTspeech reads ASCII text with no vocabulary limits. It is available for Dutch, English (male and female), French, German, Italian, Portuguese, Spanish, Arabic, Cantonese, Japanese, Korean, Malay, Mandarin and Russian.

BeSTspeech is a technology that generates and reads virtually any computerized text over the telephone, with a natural sounding, multi-lingual voice. The product is based on Lernout & Hauspie Speech Product (L&H) TTS algorithms.

More information about BeSTspeech is on the web at:

http://www.alternateaccess.com/products/texttospeech.html

❖ *DECtalk Speech Synthesis*

DECtalk Speech Synthesis transforms ordinary text into natural-sounding, highly intelligible speech. It provides the highest level of speech quality and accuracy in the industry, producing clear, correct pronunciation of single characters, words, phrases, and proper names. DECtalk offers personalized voices, as well as extensive user controls that ensure optimum performance in real-world applications, such as voiced Email and FAX messages or Voice Output Communication Aides (VOCA). Moreover, DECtalk includes built-in applications such as "Speak," which synthesize text in a window. DECtalk output can be directed to audio devices, into WAVE files, or into memory buffers.

DECtalk Speech Synthesis is available for Intel systems running Windows NT or Windows 95, and for Alpha systems running Windows NT or DIGITAL UNIX.

The DECtalk Express external package is an external, portable package you can plug in to any PC or serial port. The external package includes a built-in speaker and headphone jack, plus combined on/off and volume controls and a rechargeable battery pack.

As a Multi-language text-to-speech synthesis, DECtalk features nine defined voice personalities (four female, four male, and one child); special email and URL text-processing rules; ability to say letters, words, or phrases; pause, continue, and stop speaking controls; variable speaking rates, from 75 words per minute to 650 words per minute; accurate letter-to-sound pronunciation rules; large internal (fixed) dictionary; punctuation control for pauses, pitch, and stress; ability to say proper names; output stereo volume control; user-defined pronunciation dictionary; ability to generate DTMF tones for dialing; ability to generate tones of a specified frequency and duration; voice-control commands may be inserted into any text file for use by DECtalk software applications; Microsoft Speech API compliant; multiple instances of the speech object in a single process; supports any AlphaStation or Intel 486-based personal computer, 50 MHz or faster; and requires 8 MB memory.

More information about DECtalk can be accessed on the web at:

http://ww1.systems.digital.com/DDic.nsf/99269e41de6401f5802563ce0047ffd8/9c1357dd99c807d68025668a001b86c1?OpenDocument

❖ *ETI-Eloquence*

ETI-Eloquence is a software-based text-to-speech system and generates waveforms completely algorithmically instead of by concatenating waveforms, for maximum flexibility and naturalism. For instance, when the user requests a deeper voice, the software simulates a larger vocal tract, instead of simply pitch-shifting samples. It uses high-level linguistic parsing, which obviates the need for a huge dictionary. It handles numbers, acronyms, currency, etc. and includes a set of annotation symbols, for placing stress on particular words, expressing excitement/boredom, etc. It also allows phonetic input and supports MS SAPI. The system can speak seven different languages and dialects - US and UK

English, Castilian and Mexican Spanish, French, German, Italian, Canadian French and Chinese.

TTS systems are made available to application developers in the form of toolkits that can be accessed through an application program interface (API). The ETI-Eloquence toolkit is a highly flexible toolkit that takes advantage of the powerful linguistic models and development tools that underlie it. It produces highly intelligible General American English speech with exceptionally natural intonation.

The toolkit provides eight, fully customizable, voices, including those of adults and children, both male and female. It provides functions that make it well suited for a wide range of applications. These include real-time mouth position data for spoken phonemes (e.g., for creating animated talking faces), text indices for synchronizing actions with the speech output (e.g., highlighting words on the screen as they are spoken), special dialog boxes for customizing user dictionaries, multi-channel output (for telephony applications), and more.

The toolkit is presently available for Windows-based PCs, and is being ported to other platforms. In addition to ETI's own API, the system supports the Microsoft Speech Application Program Interface (SAPI). ETI is continuing to enhance the toolkit with new features. Further information can be found on the web site:

http://www.eloq.com/

❖ ***Eurovocs***

Eurovocs is a stand-alone text-to-speech synthesizer that uses the text-to-speech technology of Lernout and Hauspie Speech Products. It is presently available for Dutch, French, German and American English. One Eurovocs device can support two different languages and can be connected to any computer through a standard serial interface. It supports personal dictionaries, generation of DTMF tones, and pronunciation of special character sequences such as digit strings, telephone-numbers, date and time indications, abbreviations, alphanumeric strings, etc.

More information can be found on:

http://www.elis.rug.ac.be/t&i

❖ ***Infovox***

It is a Multilingual Text-to-speech system and the languages available are American English, British English, German, French, Spanish, Italian, Swedish, Norwegian, Icelandic, Danish and Finnish. Infovox uses both formant synthesizer and diphone synthesizer.

PC board INFOVOX 500 is a half-length board that can be used in IBM PC, AT, PS/2 model 30 or compatible computers. The board is delivered with a loudspeaker and a control device for adjustment of speech rate and output volume. INFOVOX 500 can be connected to the Dialogics telephone interface card for easy integration in telecom applications.

Desktop unit INFOVOX 700 is a serially interfaced stand-alone converter that can be attached to any computer or terminal through an RS 232 interface. OEM board INFOVOX 600 is an OEM board built with CMOS IC´s. Software for Macintosh INFOVOX 210 is a software-based converter for use in an Apple Macintosh with the Apple Speech Manager interface. INFOVOX 220 is a software-based converter for use in an IBM compatible PC with Microsoft Windows 3.1 and 95. INFOVOX 230 is a software-based converter for use in an IBM compatible PC with Microsoft Windows 95 and NT. INFOVOX 330 is a software-based converter for use in an IBM compatible PC with Microsoft Windows 95 and NT. Further information can be found at: http://www.promotor.telia.se/infovox/index.htm

❖ ***Laureate***

Laureate is a text-to-speech system developed at BT Laboratories. It was designed to generate high-quality natural synthetic speech from typed text, while maintaining high intelligibility. It is available for disabled users from Cambridge

Adaptive Communication and from the Foundation for Communication for the Disabled. For commercial applications, Laureate can be obtained from HTK Limited.

Laureate was designed to be an industrial strength TTS system. This means it includes the following features: multi-channel capabilities, architectural support for multi-processor systems, ease of maintenance and upgrading, platform independence and robustness.

Laureate includes a versatile yet simple API (application programmer's interface), which allows it to be viewed as a black box by an application. However, many parts of the Laureate system can be configured for any particular application, allowing it to be tailored to specific tasks.

Laureate uses a technology that allows the synthetic speech to be modeled on almost any speaker. Laureate currently supports British and American English, and different accents within these. There are prototype versions of Laureate for several other European languages, including French and Spanish. General information about Laureate can be found on:

http://innovate.bt.com/showcase/laureate/index.htm

The commercial system is presented on the web site:

http://www.htk.co.uk/

- ***Lernout & Hauspie Madison[tm]***

L&H Madison enables your computer to talk to you. By tightly integrating L&H Text-To-Speech (TTS) technology with the Windows 95 and Windows NT 4.0 operating systems, the entire operating environment becomes capable of producing speech. Also, any text from any application can now be spoken at the request of the user. Madison is implemented as an application toolbar, which provides an easy way to speech-enable a computer. The user can have text spoken by dragging text from almost any application and dropping it onto the application bar enabling it to be read.

The following languages are available: Dutch, English (US), English (British), French, German, Italian, Spanish

More information can be found on the web site

http://www.lhs.com/

- ***Listen2 Text Reader***

Listen2 is a multi-voice, multi-language text reader. It comes in two versions, English-only that uses high quality male and female voices, and the International version that can speak up to 5 different languages: English, German, French, Spanish or Italian, all in male voices. The basic International program comes with built-in English, and additional language fonts can be purchased separately. The English version comes complete.

Both programs are dynamically switchable and configurable. This means you can press a hot key to speed up the speech, make it louder or quieter, etc., while it is reading a file. You can also insert flags in text files to make it switch voices or switch languages, depending on what version you have.

Listen2 has all the features of the JTS Reader shareware program plus a few more. It will voice your reminder messages or appointment list on start-up. It will also speak a reminder message on shutting down. Listen2 uses the proVoice speech synthesizer.

More information can be found on the web site:

http://www.islandnet.com/jts/

❖ *Lucent Technologies Bell Labs Text-to-Speech System*

Lucent Technologies provides a website with demos and samples of their latest speech synthesis technology. The site has interactive demos in American English, German, and Mandarin Chinese, and the capability to adjust voice parameters on the fly. Pre-synthesized demos for French, Italian, Russian, and Romanian are also provided. The site includes downloadable papers with detailed system descriptions.

The Bell Labs TTS system generates synthetic speech by concatenating segments of natural speech. The majority of units in the acoustic inventory are diphones.

The unit selection and concatenation modules select and connect the acoustic inventory elements. These modules retrieve the necessary units, assign new durations, pitch contours and amplitude profiles and pass parameter vectors on to the synthesis module. Our TTS system uses vector-quantized LPC and a parameterized glottal waveform for synthesis.

One of the main areas of research at Bell Laboratories (Murray Hill, New Jersey) in the past few years was the development of a TTS system architecture that can function as a synthesizer for multiple languages. Currently, there are working systems for English, French, Spanish, Italian, German, Russian, Romanian, Chinese and Japanese. The system is multilingual in the sense that the underlying software for both linguistic analysis and speech synthesis is identical for all languages, with the exception of English.

Obviously, some language-specific information is necessary; there are acoustic inventories unique to each language and there are also special rules for linguistic analysis. These data, however, are stored externally in tables and parameter files, and are loaded by the TTS engine at run-time. Thus, in applications such as dialog or email reading, it is possible to switch voices and languages as desired at run-time. It is available in both server- or client-based options.

More information can be found on the web site below:

http://www.bell-labs.com/project/tts/

❖ *ProVerbe Speech Engine from ELAN Informatique*

The ProVerbe Speech Engine from ELAN Informatique produces natural sounding speech from written text. Naturalness is achieved by using the TD-PSOLA process from the CNET (France Telecom's research lab.), which is based on the concatenation of elementary speech units (including diphones). Supported languages are British English, American English, Russian, German, French and Spanish. For multi-channel applications Elan Informatique also provides hardware platforms.

Elan Informatique provides a SDK reference document.

More information can be found on:

http://www.elan.fr/

❖ *ProVoice Developer's Speech Toolkit from First Byte*

ProVoice Developer's Toolkits are available for DOS, Windows 3.1, Windows 95, Windows NT, OS/2, and Macintosh. ProVoice allows programmers to add synthesized speech to their applications. The program passes text strings to the ProVoice speech engine that translates text into audible speech. Male and female "SpeechFonts" are available for many languages - American English, French, German, British English, Italian, and Spanish.

ProVoice converts text to speech in two phases using a set of phonetic translation and pronunciation rules. First, the

software analyzes and translates text into "sound descriptors," a phonetic language with pitch, duration, and amplitude codes that are required to produce stress patterns in phrases and sentences. Rules are used to analyze words, numbers, and punctuation. The second phase converts the intermediate phonetic language in speech signals, algorithms drive distinct speech signals into smooth flowing, continuous, clear speech. Real-time synchronization of mouth movement and word boundaries allows animation of a graphical talking character, or highlighting of displayed text as it is spoken.

Necessary tools and examples are provided for programmers to manipulate the ProVoice speech technology; including installation instructions, extensive sample programs, and complete documentation. In addition, sample code is provided on disk to illustrate speech-programming techniques.

The following website gives more information about the system:

http://www.firstbyte.davd.com/

❖ ***VECSYS***

LA GAMME OPENVOX is speech synthesis by concatenation of different sized units. The unit size can be up to a complete sentence. It has been used in systems giving information for example about weather.

More information can be found on the following home page:

http://www.vecsys.fr/tap/produits.html

Hindi Speech Synthesis Systems

This section attempts to list the commercial Hindi speech synthesis systems as available on the web. It does not include the research systems as most of them are on a developing stage. The aim is to list the present major working speech systems in Hindi.

❖ ***HindiVani***

HindiVani, developed by Central Electronics Engineering Research Institute, New Delhi, is a PC based Unlimited Vocabulary Text-to-Speech Conversion Software for Hindi (Alpha Version) for DOS platform. The text document is generated using a Hindi Editor that supports ISCII standard. The input words are split into syllables, using a parser. An acoustic-phonetic inventory of all these syllables is available in the database, which is subsequently used to create words. The concatenation of syllables into words and the superimposition of quality features are done by developing rules. The quality features of speech, such as intonation, stress and timing patterns are then improved. A cascade-parallel formant synthesizer is used to synthesize the speech. The major components of the system are Word Parser, concatenator, parametric database, knowledge base (frequency and duration rules), voice quality manager, Klatt synthesizer and D/A converter.

The software can be used as an aid for the visually handicapped and for information retrieval. The system is being ported to Windows environment and prosody rules are being incorporated for better quality speech output.

❖ ***Webdunia SpeaKit***

The Webdunia SpeaKit is a font independent, Text To Speech Synthesis system that reads all your computer texts in any of 11 Indian Languages. It then generates an intelligent speech audio stream from the text. Text-To-Speech applications can reduce dependence on prerecorded prompts and can convert dynamic text into audible speech. Webdunia SpeaKit is extensively working towards coming up with a smooth and natural sounding Text To Speech System that helps in enhancing your interface with your applications in a language of your choice.

❖ ***dhvani***

dhvani is the Text-to-Speech effort of the Simputer Trust. The aim of this effort is to ensure that literacy and

knowledge of English are not essential for using the Simputer. Using images in conjunction with voice output in local languages makes the Simputer accessible to a larger fraction of the Indian population.

Currently ***dhvani*** has a phonetics-to-speech engine that is capable of generating intelligible speech from a suitable phonetic description in any Indian Language. In addition, it is capable of converting UTF-8 text in Hindi & Kannada to this phonetic description, and then speaking it out using the Phonetics-to-Speech engine. Development efforts are on to create a better phonetic engine, a platform independent Java port, a language independent framework etc.

Chapter 3

Festival – ***An introduction***

Festival Speech Synthesis Systems is a complete multi-lingual text-to-speech system built at CSTR (Centre for Speech Technology Research), written in C++ and uses Edinburgh Speech tools and SIOD, a Scheme based command interpreter. It provides a general framework for building speech synthesis system in any language. It is available for free on the Internet. Festival is designed as a speech synthesis system for three levels of users – one, for those who want to develop a high quality text-to-speech or a new voice, second, for those who want to develop language systems, and third, for those who want to develop and test new synthesis methods. Festival provides a common environment for implementing multiple techniques.

Alan W. Black, Paul Taylor and Richard Caley primarily wrote festival speech synthesis system. Festival uses the functionality of many subsystems, as we will see later. The three main subsystems are –

- Edinburgh Speech Tools (EST) Library,
- Scheme Interpreter (SIOD – Scheme In One Defun)
- Editline.

EST stands at the core of the festival system, Scheme Interpreter offers assistance in applications such as a scripting language and Editline provides a command line editing system.

Festival Speech Synthesis System can be divided into four parts

- Festival's architecture
- Text Processing
- Linguistic/Prosodic Processing
- Waveform Synthesis

Festival's Architecture:

Architecture of any system can be of two types – one, *pipeline architecture* and two, *blackboard architecture*.

Pipeline architecture is built in such a way that every module reads structured information from the preceding module and outputs a new structure for the following module. As such every module is dependent on the previous one. The disadvantage is that even if one wants to develop a later module, all the previous ones should be properly outputted.

Blackboard architecture, on the other hand, is such where a 'global' is accessible to all modules and any module can be added or modified as per the requirement. Festival follows the blackboard architecture in the sense that though the modules are in coordination of each other but still a researcher can develop later modules on already existing modules and need not start from the start. The researcher can develop his part and add to the existing ones.

Festival uses a number of API's for synthesis:

- Unix Shell
- Emacs
- Interactive command interpreter Scheme based read-eval- print loop
- C++ library adding modules in C++
- client/server mode

At the core Festival architecture consists of the following:

❋ *Scheme Interpreter* (a dialect of Lisp)
It uses scheme, a scripting language for easy specification of parameters and flow of control within the system.

❋ *C++/C modules*
C++ or C modules are used in core modules and are easily interfaced to the Schema interpreter.

❋ *Waveform input/output, format, resampling* – It supports many formats such that waveforms, label files, coefficient files can be easily read and written. It also offers resampling and changing formats.

❋ *Utterance Structure*
An *utterance* consists a set of *items*, which are related through a set of *relations*. An *item* may be in one or more relations. Each relation consists of a list or tree of items. *Items* are used to represent objects like words or segments, and has a number of *features* associated with it *Relations* are used to link *items* together is useful ways, such as in a list of words, or

a syntax tree or the syllable structure. Individual items may be in multiple relations.

The utterance structure in Festival is such that it gives a common regular form to all utterances. There is no fixed set of relations and new ones can easily be added at run-time, or existing ones ignored. To access information in items in an utterance a simple feature mechanism has been implemented, where each item holds features named by a string, feature values may be strings, integers or floats.

❋ *Standard data tools* – There are other basic tools available standard to the system such as Viterbi decoder, ngram support, regular expression, Cart support, weighted finite state transducers, and stochastic context free grammars.

❋ *Audio device access and spooling*

Edinburgh offers direct or indirect support for many types of output audio device. It also supports spooling, which allows the next utterance to be synthesized while the previous is actually playing. Synthesis time varies from machine to machine and synthesizer to synthesizer. There can be many factors and they are not all due to the actual algorithms used in the synthesizer. Resampling time, pauses introduced by audio hardware, and accessing files on remote disks often play as much in the overall timing as the algorithms and the machine speed itself.

❋ *Server/client model*

Festival has a server client model such that larger and more powerful machines might be used remotely by smaller programs saving on both start up time and resources required on the client end.

Text Processing

The first three tasks of speech synthesis are analysis of raw text so that it can be processed. Text processing converts an arbitrary text into identifiable words chunked into reasonable sized utterances.

Text processing is done in the following steps:

- Identify tokens in the text
- Chunk the tokens into reasonably sized sections
- Identify types for tokens (some pronounced, some not)
- Map tokens to words
- Identify types for words

Tokens are identified in a raw text on the bases of two features – one, preceding whitespace, i.e. space, tab, new line, and carriage return and two, succeeding punctuation. Festival converts text into an ordered list of tokens on the basis of these two features.

These tokens are then chunked into reasonable sized utterances, mostly a sentence. Utterance breaks depend on a rule system based on tokens and its context. The currently used decision tree for determining end of utterance is as follows:

```
((n.whitespace matches ".*\n.*\n[ \n]*") ;; A significant break in the text
((1))
((punc in ("?" ":" "!"))
((1))
((punc is ".")
 ;; This is to distinguish abbreviations vs periods
 ;; These are heuristics
 ((name matches "\\(.*\\..*\\|[A-Z][A-Za-z]?[A-Za-z]?\\|etc\\)")
 ((n.whitespace is " ")
 ((0))           ;; if abbrev signal space is enough for break
 ((n.name matches "[A-Z].*")
 ((1))
 ((0))))
 ((n.whitespace is " ")  ;; if it doesn't look like an abbreviation
 ((n.name matches "[A-Z].*")  ;; single space and non-cap is no break
 ((1))
 ((0)))
 ((1))))
 ((0)))))
```

Next step is to relate each token to zero or more words. This is done on the basis of context-sensitive rules and takes into account numbers, dates, money, some abbreviations, email addresses and random tokens. This is to identify and analyze tokens that have some internal structure. For instance pronunciation of numbers generally depends on the context:

1936 date: nineteen thirty-six.

1936 phone number: one nine three six (an extension number etc.)

1936 quantifier: one thousand nine hundred (and) thirty six

19:36 time: nineteen hours and thirty-six minutes

19 day: nineteenth

Homographs are another candidate for tokenization. They are words that have same written form but are pronounced differently. Homographs can be distinguished through different types of information –

Different part of speech: `project`.

Semantic difference: `bass, tear`.

Proper names: `Nice, Begin, Said`.

Roman Numerals: `Chapter II, James II`.

Numbers: years, days, quantifiers, phone numbers.

Some symbols: `5-3, high/low, usr/local`

Festival supports a homograph disambiguation method technique that covers all classes of homographs, those based on part of speech as well as "semantic" ones (e.g. "row" (boat and argument)), though the part of speech tagger usually deals with part of speech homographs. For this technique, first a *large* corpus of words is required from which all occurrences of homographs are extracted and labelled with its class and then contextual features are extracted that will identify the class. Finally a classification tree or decision list is built to classify occurrences.

Next is the level of mode specific processing for the selective treatment of tokens in different types of context. In this, mode specific rules are applied for a particular text. For example, a text mode can be specified for emails such that conventions for quoting (and signatures etc) can be dealt with, that make comprehension of the message much easier.

Festival supports the notion of *text modes*, following the notion of modes in Emacs, which allow customization of mode specific parameters.

For text mode, Festival offers –

filter - A Unix program filter for the file. In email-mode this removes most of the mail headers.
init_function - A Scheme function to be called when entering the mode. This allows selection of voice, addition of lexical entries, and mode specific tokenization rules to be set up.
exit_function - Called on exiting the mode, so you can tidy everything up and not leave mode specific rules that cause other synthesis modes to fail.

Festival also uses a Mark-up language - Sable, which is independent of any particular synthesizer so that the input information can be used to identify the text. In many uses of synthesis such information does exist in the application and could easily be passed on to the synthesizer if there was a well-defined way to do so.

Sable is an XML-based language developed for marking up text. A group involving AT&T, Bell Labs, Sun, Apple, Edinburgh University and CMU designed sable. It is intended as a standard that can be used across many different synthesis systems.

Input in Sable may be labelled, identifying pronunciation breaks, emphasis etc. Sable currently supports selection of language, voice, genre; marking of boundaries and emphasis; definition of pronunciation for words; pronunciation of literals (spelling) and phonemes; specification of intonation and speaking rate and inclusion of sound files.

In Festival Text Processing is done in five steps –

- Tokenizing the text into an ordered list of tokens
- Chunking the tokens into utterances
- Applying user defined functions to each utterance
- Each token in an utterance is analyzed and converted into one or more words
- The whole file is filtered in text mode and mode specific parameters are specified such as token-to-word functions.

Linguistic/Prosodic Processing

Linguistic/Prosodic processing is the translation of words to segments with durations and F0 contour. The quality of speech, its naturalness, intelligibility and acceptability largely depends on the appropriateness of the phonetic and prosodic output. Linguistic processing is specifically done in the following modules as stated below:

- Lexicons: mapping words to pronunciations
- Letter to sound rules: when no list of words is available
- Intonation: finding the tune
  - Accent assignment: where are the accents and what are their types.
  - F0 contour generation: by rule or statistical method.
- Duration: specification of length of each segment.
- Post-lexical rules: co-articulatory effects between words.

Lexicons:

The simplest way to find the pronunciation of a word is to look it up in a word list or *lexicon*, which gives the syllabic structure, lexical stress and pronunciation in some phoneme set. Some languages do not have a one-to-one mapping between graphemes and phonemes. In such cases, the only option left is to build a lexicon. For example, languages like French and English do not have a grapheme corresponding to every phoneme. The size and necessity of a lexicon is language dependent, but it is almost always convenient to have at least a small list for various acronyms and names etc.

In Festival, a lexicon consists of three parts:

Compiled Lexicon – It is a large list of words with their part of speech, syllabic structure, lexical stress and pronunciation. This is not loaded into the system but accessed as required and can be accessed efficiently even for lexicons with 100,000s entries

Addenda – It is a typically smaller list of entries specific to a task or implementation. In Festival, a typical addenda includes specialized words and names such as "ToBI", "PSOLA", "awb" and "email" which are not typically found in big lexicons.

Letter to sound rules – It is a general letter to sound rule system included in Festival allowing mapping of letters to phones by context-sensitive rules.

Letter-to-sound rules:

However, there could exist a well-defined mapping from the orthography to the pronunciation and in such cases building letter-to-sound rules is a better option. For some languages, the orthography closely maps to pronunciation (e.g. Hindi) and letter to sound rules can do almost all pronunciation. It is practically impossible to list all words in a lexicon. The solution to this is to offer a mechanism that assigns a pronunciation to words not found in the lexicon or addenda.

Festival supports a basic standard mapping rule

```
( LEFTCONTEXT [ ITEMS ] RIGHTCONTEXT = NEWITEMS )
```

eg: ([th] = th)
([ch] = ch)

Prosodic Phrasing:

It is necessary to split utterance into prosodic phrases for a natural sounding intelligible speech. Phrasing tends to be balanced, that is, we prefer to keep chunks roughly the same size, so that factors like number words in phrases make a difference.

Festival supports two major methods, one by rule and the other using a statistical model based on part of speech and phrase break context.

Prosodic phrasing by rule:

The rule system uses a decision tree, which may be trained from data or written by hand. Following is a simple example included in **`lib/phrase.scm'**:

```
(set! simple_phrase_cart_tree'
((R:Token.parent.punc in ("?" "." ":"))
((BB))
((R:Token.parent.punc in ("'" "\"" "," ";"))
((B))
((n.name is 0)  ;; end of utterance
```

((BB))
((NB))))))

This is applied to each word in an utterance. The models may specify any of three values – no break, small break and large break. This model is often reasonable for simple TTS, but for long stretches with no punctuation it does not work well.

Prosodic phrasing by rule:

This is a more complex model, which compares favorably with most other phrase break prediction methods. Most methods try to find the likelihood of a break after each word but don't take into the account the other breaks that have already been predicted. This can often predict breaks at places where there is a much more reasonable break close by.

The more elaborate model supported by Festival finds the optimal breaks in an utterance, based on the probability of a break after each word (based on its part of speech context), and the probability of a break based on what the previous breaks are. This can be implemented using standard Viterbi decoding. This model requires:

- Ngram model of B/NB distribution (up to a length of 8 or so seems optimal).
- Probabilities of breaks, given current, previous and next part of speech tags.

Intonation:

Intonation prediction can be split into two tasks

1. Prediction of accents: (and/or tones) this is done on a per syllable basis, identifying which syllables are to be accented as well as what type of accent is required (if appropriate for the theory).
2. Realization of F0 contour: given the accents/tones generate an F0 contour.

Prediction of accents:

The basic first approximation for accent prediction, especially in English, is to accent all stressed syllables in content words. This basically over assigns accents and is most noticeably wrong in compound nouns. The ToBI intonation theory states that there are only a small number of distinct accents. Therefore, we need to then choose between these accents types. Although there seems to be some linguistic difference in the use of each accent, its difficult to fully identify it. Statistically trained methods (again CART) produce

reasonable results. ToBI makes distinctions between accents and boundary tones. Tones are the intonation events that occur at the end (or start) of prosodic phrases.

Festival allows accent placement by decision tree. A much simpler example is just to have accents on stressed syllables in content words (and single-syllable content words with no stress). A decision tree to do this is as follows:

```
(set! simple_accent_cart_tree
'
((R:SylStructure.parent.gpos is content)
((stress is 1)
((Accented))
((position_type is single)
((Accented))
((NONE))))
((NONE))))
```

F0 Generation:

The F0 is the basic tune in speech. There are various theories for F0 generation, each with various advantages and disadvantages. This is of course language dependent, but a number of methods have been relatively successful over multiple languages. The task of generating a contour from accents is actually much better defined than generating the accent placement or their types.

Festival supports a number of methods that allow generation of *target* F0 points. These target points are later interpolated to form an F0 contour. The simplest model adds a fixed declining line. This is useful when intonation is be ignored in order to test other parts of the synthesis process.

The General Intonation method allows a Lisp function to be written, which specify a list of target points for each syllable. This is powerful enough to implement many simple and quite powerful theories.

The following function returns three target points for accented syllables given a simple hat pattern for accents syllables.

```
(define (targ_func1 utt syl)
"(targ_func1 UTT ITEM)
Returns a list of targets for the given syllable."
(let ((start (item.feat syl "syllable_start"))
(end (item.feat syl "syllable_end")))
(if (not (eq? 0 (length (item.relation.daughters syl "Intonation")))
(list
(list start 110)
(list (/ (+ start end) 2.0) 140)
```

```
    (list end 100)))))
```

A different method for implementing ToBI is also implemented in Festival. In this model, linear regression is used to predict three target values for every syllable using features based on accent type, position in phrase, distance from neighboring accents etc. The results appear better than the rule approach, but need data to train from.

Duration

Explicit segmental durations are necessary for synthesis. Festival supports fixed or average duration modified by rule. Fixed method is used by assigning a fixed size for all phones for example, 100 milliseconds. This is the easiest way. The next simplest model assigns the average duration for that phone (from some training data). This actually produces reasonable results. In duration modified by rule, duration is assigned based on a large number of experiments and modifications from a base duration of all the phones. These modify the basic duration by a factor based on information such as position in a clause, syllable, position in word, syllable type, etc. The results sound reasonable though a little synthetic. These rules are one of the fundamental features that people subconsciously identify when recognizing synthetic speech. In Festival, best duration modules are those trained from database of natural speech using the sorts of features used in the Klatt rules but deriving the factors by statistical methods.

A simple duration decision tree predicts a factor to modify the average duration of a segment. This tree causes clause initial and final lengthening as well as taking into account stress.

```
(set! spanish_dur_tree
 '
 ((R:SylStructure.parent.R:Syllable.p.syl_break > 1 ) ;; clause initial
 ((R:SylStructure.parent.stress is 1)
 ((1.5))
 ((1.2)))
 ((R:SylStructure.parent.syl_break > 1)  ;; clause final
 ((R:SylStructure.parent.stress is 1)
 ((2.0))
 ((1.5)))
 ((R:SylStructure.parent.stress is 1)
 ((1.2))
 ((1.0))))))
```

Post-lexical Rules:

There are some segmental effects that cannot be determined for words in isolation, as they only appear when the pronunciations are concatenated. Many of these co-articulatory

effects are ignored in synthesis, which often leads to what appears to be over-precise articulation. One such effect that can however be easily dealt with is vowel reduction. The effect often happens in non-important function words. Although this can be trained from data we can specify mappings of full vowels to schwas is specific contexts. Another form of reduction is word contractions, as in "it is" to "it's". This is very common in speech and makes a synthesizer sound more natural.

Festival supports a general method for post-lexical rules. These may be specified on a per-voice basis. Although some phenomena occurs across dialects, some are dialect specific. For example, in British English, post-vocalic "r" is deleted unless it is followed by a vowel. This can't be determined in isolated words for word final r's so this is done as a post-lexical rule. The following simple CART is applied to each segment in the segment relation after lexical look up to decide if the "r" has to be deleted or not.

```
(defvar postlex_mrpa_r_cart_tree '((name is r)
 ((n.ph_vc is -)
 ((delete))
 ((nil)))
 ((nil)))
```

Waveform Synthesis

A major part of speech synthesis process is waveform generation. Waveform synthesis is the process where the input text after being processed, is converted into speech by mapping the graphemes with the corresponding segmented units stored as waveforms in a database form. The synthesized speech is generated by putting together the database units. For instance, if we want to synthesize the word /put/, the synthesizer will concatenate the following segments:
silence-p, p-u, u-t and t-silence.

The database should be properly made such that the “joining” points or say the concatenation points should be at the same level. For this reason extracting pitchmarks is important.

Festival is a complete system where one can develop his own part with the help of already provided information in the system. Hence, if someone does not want to perform the initial steps of text processing and prosodic processing, one can still build a database and listen to a reasonable output. Festival provides default values for prosody, duration, etc. Even if there is no text processing, one can hear the synthesized speech at the level of phones in a monotone.

There are three main types of speech synthesis techniques (See Chapter 1):

Articulatory - models human vocal tract, produces high quality speech but is a difficult process.
Formant – Models the speech signal, produces reasonable quality but sounds synthetic. This also a comparatively difficult method.
Concatenative – concatenates segmented units of natural speech, which are prosodically modified to produce natural like speech.

Festival is a concatenative Speech Synthesizer. In Festival, waveform synthesis is generally done by building a database of segmented units, mostly diphones, (though longer units can be selected). The process of building a diphone database is discussed at length in Chapters 4 and 5. The phones of a particular language are defined and then a list of prompts is prepared. A schema code makes the list of diphones by pairing up each phone with the other.

After the prompts are ready, they are recorded and labelled. These labelled prompts are then segmented into diphone units and an index is prepared with the start, middle and end time for each diphone. Then the pitchmarks are extracted and normalized and LPC coefficients are calculated to give a uniform pitch and tone to the diphones for a smooth synthesized speech. This accounts for unrestricted synthesis. However, longer units can also be synthesized in Festival for Limited domain purposes.

There is also a technique to select *non-uniform units* from a database of isolated words. These are called *non-uniform* because the selected units may be phone, diphone, triphones or even longer. Thus the inventory contains many examples of each phone and acoustic measures are used to find the most appropriate units. In many cases multi-phone units would be selected especially for common phonetic clusters.

Festival supports UniSyn database module that can use database in two formats: separate, when all files are accessed individually during synthesis and grouped, when a database is collected together into a signal special file containing all relevant information. A database should consist of a set of waveforms, with pitchmarks and an index. The process of generating pitchmarks, LPC coefficients and diphone index has been explained in chapter 4.

To sum up, Festival is a complete system for Text-to-speech synthesis that allows various facilities for development of new synthesis systems. It gives a lot of space to every person who wishes to work on it, from a user's as well as a developer's point of view. It also gives

a lot of liberty in developing various modules or parts even if one does not develop the whole system. It is like a basic skeleton that only needs to be adorned and given life.

Chapter 4
Concatenative Synthesis – ***Getting started***

Concatenative Synthesis is one of the most convenient ways to produce intelligible and natural-like synthetic speech. In this method, units are concatenated from a database of prerecorded natural utterances. The database is a collection of recorded linguistic units or segmented instances of utterances that can be concatenated according to the requirement. Concatenative synthesizers are generally limited to one speaker and one voice though one can develop different male or female voices. They require more storage space than other synthesizers. In concatenative speech synthesis, a unit inventory is built from the recorded utterances that are then segmented and labeled and finally most appropriate units are chosen. Segmenting and labeling each unit is generally a very time-consuming job. Nowadays most of it can be automated but still hand labelling is preferable for a better output. Concatenative synthesizers have a fixed inventory of segmented units and they cannot produce anything beyond the elements in their vocabulary.

Philosophy

Concatenative Synthesis captures transitions in all possible phone-phone combinations in a language. It is based on the philosophy that co-articulatory effects never go over more than two phones. Hence, if we “preserve” these co-articulatory effects or transitions and concatenate them we can get close to human-like synthesized voice.

The basic idea is that when any two sounds occur together in succession, the transition between them is specific to the two sounds. To explain it more clearly, two sounds, say, /b/ and /a/ have certain transition between them, which is specific to them. Similarly /b/ and /i/ will have a specific transition, which will be different from the transition of /b/ and /a/. Since these transitions carry individual characteristics of the two sounds, they are a hallmark for the continuous speech.

Diagrammatically, every sound, as can be seen in a sound wave, shows three parts - the beginning, the middle stable part and the trailing part of the sound (as explained in non-technical terms). This is more marked in vowels, glides and sonorants. The beginning and the trailing part change in the presence of the preceding or following sound, thus giving co-articulatory effects. However the middle stable part does not change and remains stable. This idea has been captured in concatenative speech synthesis especially in diphone synthesis where the transitions are preserved from half of one phone to the half of other phone such that while concatenating, the stable portions are joined together to produce

continuous speech.

This method may sound simple but in practice, one has to face many practical problems such as the pitch of all the diphones, which should be at the same range so that while concatenating, the stable portions are well joined. Also one has to take care of the clicks and gaps that may occur while segmenting the diphones. Care has to be taken that diphones are properly labelled and segmented as it reflects on the quality of synthesized speech.

Concatenative synthesis supports varied lengths of units for concatenation such as words, syllables, demisyllables, phonemes, diphones, and triphones. The choice of units reflects on the quality and efficiency of the speech synthesis system. Two common methods of concatenative speech synthesis are diphone-based synthesizers and unit selection method.

Units of Concatenation

One of the most important aspects of concatenative synthesis is to find correct unit length. The choice has to be made whether the units under consideration are longer units or shorter units. Longer units are preferred for high naturalness as there are less concatenation points and better co-articulatory effects but such units require large memory as they are more in number. On the other hand, shorter units require less memory but have more concatenation points and hence have more clicks, giving it a synthetic effect. Also, sampling and labeling procedures are more complex for smaller units. Therefore a good speech synthesis system has to make a choice between the kinds of units to be taken into consideration. The units of concatenation are words, syllables, demisyllables, phonemes, diphones, and triphones. (These have been explained in detail in Chapter 1).

Words are used as concatenating units in very specific cases where vocabulary required for synthesized speech is limited to some special use. Word is generally not a preferred unit because there are innumerable words in a language and a lot of them are proper words. To record all of them is not practically possible.

Syllables are another unit of concatenation lesser in number than words, but still the database is too large for a text-to-speech system.

Demisyllables are units that represent initial and final parts of syllables. They require less concatenation points and take into account most of the co-articulation effects. But it is not

possible to calculate the exact number of demisyllables in a language and the memory required by demisyllables is also quiet large.

Phonemes are the linguistic representation of speech and hence the most commonly used units in speech synthesis. But phonemes do not capture the co-articulation effects and the synthesized speech may sound very mechanical.

Triphones are longer segmental units, like diphones except that they include a complete phone between two half phones or steady states (half phone – phone – half phone). The number of triphones in a language is very large. Triphones require more concatenation points; as such the quality is not that natural sounding as with diphones.

Above we have seen the limitations of all kinds of units of concatenation. Most of them cannot be made into an “only-unit” database, i.e., one cannot have a good speech synthesis system with only words, syllables, demisyllables or phonemes as the unit of concatenation due to their size or quality. However, if these are used together in a combination in something like unit selection method, where the system chooses which units to concatenate depending on the context, they make good units of concatenation in a speech synthesis system. There can be “only unit” database as well and one such database that is also widely used in synthesis is the diphone database..

Diphones are units that begin in the middle of the stable state of a phone and end in the middle of the following one. They aid speech synthesis by capturing the transitions between the two phonemes and the co-articulatory effects. They are a list of all phone-phone transitions in a language. The number of diphones in a language is the square of the number of phones. Thus if there are 50 phones in a language, the expected diphones are around 2500. The number varies depending on the valid combinations. A complete diphone database at least includes the following combinations --

#-V (silence-Vowel), V-# (Vowel-silence),
#-C (silence-Consonant), C-# (Consonant-silence),
CV (Consonant-Vowel), VC (Vowel-Consonant),
VV (Vowel-Vowel), and CC (Consonant-Consonant).

Phones, in this context, are not defined in strictly linguistic terms. Phones include any allophonic variations or whatever units one may consider, for instance, in Hindi one may include /nh/ as a phone even though it occurs in very restricted environment and is not even explicitly regarded as an allophone. It is generally considered /n/ followed by /h/. However we can find a minimal pair for it - /kAnhA/ “Name for Lord Krishna” and /kAnA/ “one-eyed”.

A diphone database may not have all phone-phone combinations in a language because of certain phonotactic constraints of that language, for instance, certain phone-phone combinations may not occur in a language for example, 'ng' the velar-nasal sound does not occur at the beginning of the word in Hindi, hence silence-'ng' is a non-valid diphone in Hindi.

In a diphone database only one occurrence of each diphone is recorded. While selecting the phone pairs in a language one must consider not only phones that occur within a word but also those phones that occur at word boundaries. This is very important because many phone pairs may be omitted and that would give unwanted clicks and breaks in continuous speech. For example, sound /ng/ in Hindi occurs only with vowels and that too at the end of the word, but suppose if a word that ends with the sound /ng/ is followed by a word that starts with the sound /t/ as in /ting-tong/, the phone pair required is /ng-t/. Thus, while making a diphone database, it becomes very important to look closely at the possible phone-phone combinations in a language, absence of which may affect the quality of sound.

In the present system we will be concentrating on building a diphone database with help of Festival software.

Festival- a brief introduction

Festival is a Speech Synthesis System built at CSTR, Edinburgh University primarily by Alan W. Black, Paul Taylor and Richard Caley. It offers a general framework for building speech synthesis system in any language. Festival is a complete text-to-speech system and has various modules for text processing, prosodic processing and waveform synthesis. It provides a basis for research and development of various speech synthesis techniques from the concatenative approach.

Steps for building a diphone database –

A brief overview:

This section covers those commands in chronological order that are needed to build a new voice. The steps have been explained in the next chapter *Walkthrough – Hindi Diphone Database* and in the US/UK English Walkthrough in the Festvox manual, chapter 8. Hence, I will only be listing the commands now as a summary to the entire process that will be explained later in this chapter.

The following commands set the environment variables to where the Edinburgh Speech Tools, Festival and Festvox have been installed.

```
export ESTDIR=/usr/local/speech tools
export LD_LIBRARY_PATH=/usr/local/speech_tools/lib
```

(This command is not stated in the walkthrough but it is important to export the speech tools library to avoid errors.)

```
export FESTDIR=/usr/local/festival
export FESTVOXDIR=/usr/local/festvox
```

Second step is to make a new directory for the new voice:

```
mkdir ~/data/hcu_hin_sms_diphone
cd ~/data/hcu_hin_sms_diphone
```

Third step is to create the directory structure with relevant files for building a new voice:

```
$FESTVOXDIR/src/diphones/setup_diphone hcu hin sms
```

Fourth, we have to generate the list of nonsense words:

```
festival –b festvox/diphlist.scm festvox/hin_schema.scm '(diphone-gen-schema
"hin" "etc/hindiph.list")'
```

Then we have to synthesize the prompts:

```
festival –b festvox/diphlist.scm festvox/hin_schema.scm '(diphone-gen-waves
"prompt-wav" "prompt-lab" "etc/hindiph.list")'
```

Record the nonsense words from the list – hindiph.list

```
bin/prompt_them etc/hindiph.list
```

The recorded prompts can be labeled by the following command:

```
bin/make_labs prompt-wav/*.wav
```

To correct the autolabelling and handlabel:

```
emulabel etc/emu_lab
```

Next step is to make the diphone index.

```
bin/make_diph_index etc/hindiph.list dic/smsdiph.est
```

To normalize the pitch:

```
bin/make_pm_fix pm/*.pm
```

To find the overall vowel mean power:

```
bin/find_powerfactors lab/*.lab
```

The last step is to build pitch-synchronous LPC coefficients:

```
bin/make_lpc wav/*.wav
```

Now a new diphone database in a new voice is ready and can be tested by the following command:

```
Festival> festvox/hcu_hin_sms_diphone.scm\ ‘(voice_hcu_hin_sms_diphone)’
```

The voice can be tested by using the commands “SayPhones”, /namaste/ ”Hindi greeting for ‘hello’”.

```
festival> (Sayphones ‘(pau n a m a s t e pau))
```

Next, create a group file to include only the relevant portions of the diphone database:

```
festival (us_make_group_file “group/smslpc.group” nil)
```

Last step is to integrate the new voice to festival:

```
cd /usr/local/festival/lib/voices/English/
ln –s ~/data/hcu_hin_sms_diphone
```

The above stated steps build a new voice in a new language or in the same language depending on what information is provided. The same steps have been explained in the next chapter in the form of a Walkthrough. These steps have been stated here in brief because the following sections will explain what work has been done at each step to build a Hindi diphone database for Hindi Speech Synthesis.

Creating a Directory Structure

After compiling Festival Speech Synthesis system successfully on the computer, first is to create a directory structure to build a new voice in your language. This can be done by the following command:

```
# mkdir ~/data/hcu_hin_sms_diphone
# cd  ~/data/hcu_hin_sms_diphone

# $FESTVOXDIR/src/diphones/setup_diphone hcu hin sms
```

The directory is named with a convention. The name is in three parts –

- The name of the institution (in this case “hcu” that stands for Hyderabad Central University),
- The name of the language (“hin” which stands for Hindi) and
- The name of the speaker whose voice will be recorded for synthesis (“sms” that stands for Sumedha in this case).

The directory structure will be created with the following directories in it:

Bin – It has all the *make* files, that is, the code for all the commands in the process of synthesis.
Cep – This directory has the *cepstral values* of the recorded utterances.
Dic – It has the *diphone index* file.
Etc – It has the *emulabel* template and the diphone list. It also gets the pitch normalization values during synthesis.
F0 – It is generally empty unless *F0 values* have been calculated. ***Festival*** – This directory is supposed to accommodate a lot of values for prosodic analysis like duration, accent etc but not the ones used in building the diphone database.
Festvox – It contains all the files and information related to the particular language.
Group – This has all the *group files* of the diphone database i.e., subparts of spoken words that contain the diphones.
Lab – It contains all the *label files* for all the recorded utterances.
Lar – It is generally empty unless EGG signal has been collected. It contains the extracted *pitchmarks* for the EGG signal.
Lpc – This has all the pitch-synchronous *LPC coefficients*.
Mcep – This does not contain anything relevant for only diphone database.
Pm – It contains the *extracted pitchmarks* from the waveform.
Pm_lab – It contains the *pitchmarks* calculated from the label files.
prompt-cep – It contains *cepstral values* for the synthesized prompts.
prompt-lab – This has the *label values* for synthesized prompts.
prompt-wav – This has the *wave files* for the synthesized prompts.
Wav – This directory has the *wave files* for the recordings of the nonsense words.

Wrd – This directory is not required for diphone database.

The most important directory that actually defines the language to be synthesized is FESTVOX. It contains 7 files in it that are named as follows thus – “diphlist”, “hcu_hin_lex”, “hcu_hin_phones”, “hcu_hin_sms_diphone”, “hcu_hin_sms_dur”, “hcu_hin_sms_int” and “hcu_hin_token”.

Festvox should have 8 files. Along with the above seven, there is another file that has to be added – “hin_schema”. This is the most important file in the system as it helps generate the diphone list and prompts. The file is available in the Festival system libraries but has to be modified to suit the requirements of the specified language.

Detailed descriptions of each of these files and the changes that have to be made for the specific language have been stated in the following section.

This directory and the changes made in it is what make for the synthetic speech in the specified language.

Festvox files and defining parameters for Hindi Language

Festvox is a directory that includes all the relevant information about the language that has to be processed to synthesize speech, from defining the phone set and its limitations to listing the durations, intonations and lexicon. The rest of the program is automated and all that is needed are few commands. However, everything depends on the information provided in the files of this directory. The better the information provided, the better is the quality of output.

In this Section, I will be discussing briefly the role of each file in the Festvox directory. Though the description of these files is not stated explicitly in any book, but I intend to do so because it leads to a better understanding of the system. One can however, follow all the steps from the section English Walkthrough in Festvox manual, chapter 8, but it does not explain the basic functioning of the system. Therefore I will briefly explain the role of each file and then, the changes that I have made in them.

Festvox contains the following eight files:

1. diphlist
2. hcu_hin_sms_diphone

3. hcu_hin_token
4. hin_schema
5. hcu_hin_phones
6. hcu_hin_sms_int
7. hcu_hin_sms_dur
8. hcu_hin_lex

Out of the eight files mentioned above, we need to change five files – hin_schema, hcu_hin_phones, hcu_hin_sms_int, hcu_hin_sms_dur and hcu_hin_lex. The three files that do not need any language specifications for building a diphone database are diphlist, hcu_hin_sms_diphone and hcu_hin_token. The first two files are basic codes that function together with other files in Festvox while the latter one defines tokens in the target language required while performing text analysis and is not needed for building diphone database primarily. I will first be explaining these three files that do not need any change and then the five files that have to be specified with the language and/or speaker related parameters.

1) Diphlist.scm

This file basically contains code for a lot of functions especially in the initial part of the process of building a diphone database. It defines the functions like “diphone-gen-schema” that generates the diphone list with the nonsense carrier, i.e. a list of nonsense words to be recorded and “diphone-gen-waves” that generates the prompts and synthesizes each nonsense word with fixed duration and fixed pitch so that they can be played during recording. The two functions are performed together with the file hin_schema. Diphlist.scm file creates a list of diphones, the filenames for each diphone, generates the prompt, carries out mapping and waveform synthesis for all the prompts and autolabels them.

These synthesized prompts are then played for the speaker while recording to maintain the recording effects (explained later). In short, diphlist.scm is involved with all the initial preparations before the actual recording of the diphones in the process of building a diphones database. This file functions like preparing a draft before starting with the work.

2) hcu_hin_sms_diphone

This file can be compared to the summing up of the entire chapter with a conclusion. It is like wrapping up the entire diphone database to make it a voice than a conglomeration of sound sequences. It locates the database integrated in Festival or runs from the path specified.

```
;;; Try to find out where we are
(if (assoc 'hcu_hin_sms_diphone voice-locations)
(defvar hcu_hin_sms_dir
(cdr (assoc 'hcu_hin_sms_diphone voice-locations)))
;;; Not installed in Festival yet so assume running in place
(defvar hcu_hin_sms_dir (pwd)))
```

```
(if (not (probe_file (path-append hcu_hin_sms_dir "festvox/")))
(begin
(format stderr "hcu_hin_sms: Can't find voice scm files they are not in\n")
(format stderr "  %s\n" (path-append hcu_hin_sms_dir "festvox/"))
(format stderr "  Either the voice isn't linked into Festival\n")
(format stderr "  or you are starting festival in the wrong directory\n")
(error)))
```

It requires all the files in festvox except hin_schema and diphlist, and allocates all the relevant information about the specified language, i.e. information on phones, intonation, lexicon, duration and also token (if information is given):

```
;;; other files we need
(require 'hcu_hin_phones)
(require 'hcu_hin_lex)
(require 'hcu_hin_token)
(require 'hcu_hin_sms_int)
(require 'hcu_hin_sms_dur)
```

It is also responsible for grouping the files before they are integrated in Festival and contains an option for grouped files, i.e., it carries reference to the files after they have been grouped:

```
;; Go ahead and set up the diphone db
(set! hcu_hin_sms_db_name (us_diphone_init hcu_hin_sms_lpc_sep))
;; Once you've built the group file you can comment out the above and
;; uncomment the following.
;(set! hcu_hin_sms_db_name (us_diphone_init hcu_hin_sms_lpc_group))
```

The file defines the voice of the diphone database to the institution, language and speaker. This is important because when one has to play the text on the festival prompt one needs to set the voice or choose one from the various voices present in the system. To set the synthesized voice, the voice is addressed in a convention as: voice_hcu_hin_sms_diphone.

```
;;;  Full voice definition
(define (voice_hcu_hin_sms_diphone)
"(voice_hcu_hin_sms_diphone)
Set speaker to sms in hin from hcu."
(voice_reset)
(Parameter.set 'Language 'hcu_hin)
;; Phone set
(Parameter.set 'PhoneSet 'hcu_hin)
(PhoneSet.select 'hcu_hin)
```

The file groups, defines and specifies the voice and adds certain features relevant to read

the text for synthesis. Ideally one should specify the prosodic features and text features (one can find out through text analysis), for a natural sounding intelligible speech in a text-to-speech system, but for a diphone database, these features are relevant only to the limit where the speech can be heard as a monotone voice.

The quality of text-to-speech, however, largely depends on the quality of the diphone database since the basic voice can be heard by mere concatenation of diphones. Though it may sound like a mechanical speech in a monotone. The speech one hears from a diphone database is a flat voice with no intonation and accent. This file provides the basic default or "nil" values to diphone database so that at least a basic monotone could be heard. If no accent, tone, post-lexical rules, part of speech information is provided, the file hcu_hin_sms_diphone, gives them a "nil" value.

```
;; token expansion (numbers, symbols, compounds etc)
(set! token_to_words hcu_hin_token_to_words)
;; No pos prediction (get it from lexicon)
(set! pos_lex_name nil)
(set! guess_pos hcu_hin_guess_pos)
;; Phrase break prediction by punctuation
(set! pos_supported nil) ;; well not real pos anyhow
;; Phrasing
(set! phrase_cart_tree hcu_hin_phrase_cart_tree)
(Parameter.set 'Phrase_Method 'cart_tree)
;; Lexicon selection
(lex.select "hcu_hin")
;; No postlexical rules
(set! postlex_rules_hooks nil)
;; Accent and tone prediction
(set! int_accent_cart_tree hcu_hin_accent_cart_tree)
(Parameter.set 'Int_Target_Method 'Simple)
(Parameter.set 'Int_Method 'General)
(set! int_general_params (list (list 'targ_func hcu_hin_sms_targ_func1)))
;; Duration prediction
(set! duration_cart_tree hcu_hin_sms::zdur_tree)
(set! duration_ph_info hcu_hin_sms::phone_data)
(Parameter.set 'Duration_Method 'Tree_ZScores)
```

The file also sets the waveform synthesizer that maps the "read" phones onto the diphones and creates a waveform for the concatenated diphones to produce speech.

```
;; Waveform synthesizer: diphones
(set! UniSyn_module_hooks (list hcu_hin_sms_diphone_fix))
(set! us_abs_offset 0.0)
(set! window_factor 1.0)
```

```
(set! us_rel_offset 0.0)
(set! us_gain 0.9)
(Parameter.set 'Synth_Method 'UniSyn)
(Parameter.set 'us_sigpr 'lpc)
(us_db_select hcu_hin_sms_db_name)
```

Finally this file automatically sets the current voice to the voice of the specified diphone database, in this case Hindi diphone database.

```
(set! current-voice 'hcu_hin_sms_diphone)
```

To summarize it, I would say this file plays a major role in organizing the entire database and giving it the status of a voice. Without this file, all we are left with is a conglomeration of a large number of recorded and segmented diphones. But `hcu_hin_sms_diphone`, assembles the information in various files and makes a voice that can be heard with the help of SayPhones or SayText.

3) hcu_hin_token

hcu_hin_token contains information about the tokens of the target language and need to be altered but it does not play any role in the building of a diphone database. This file comes handy after a database is ready for text-to-speech synthesis system. The file has to be specified with token rules for token-to-word mapping. In very simple terms, tokens are like small categories where words can be specified for some context-specific functions. For instance, numbers can be tokenized for numerals, for currency, for time, for years etc. This is important because the same numbers can be said in different ways in different contexts, for example, 1950 can be said as “One thousand nine hundred and fifty” for money and “nineteen fifty” for year, 19:50 – “nineteen hours and fifty minutes” for time and so on.

To overcome this contextual arbitrariness, we need to list the words as tokens in this file that will allow treatment of numbers in various contexts. Similarly, abbreviations, alphanumerics, homographs (words that have the same written form but are different in meanings that can be contextually determined. e.g. “lead” can be pronounced /led/ when it appears as a noun and /liid/ when as a verb), have to be tokenized for different contexts. To list such tokens, one has to give information about the part of speech that can be listed in this file as well. Below is the file as created in the directory structure.

```
;;; Token rules to map (tokens to words)
```

```
;;; Part of speech done by crude lookup using gpos
(set! hcu_hin_guess_pos
'((fn
 ;; function words
  )
 ;; Or split them into sub classes (but give them meaningful names)
 ; (fn0 .. .. .. ..)
 ; (fn1 .. .. .. ..)
 ; (fn2 .. .. .. ..)
 ))
(define (hcu_hin_token_to_words token name)
"(hcu_hin_token_to_words TOKEN NAME)
Returns a list of words for the NAME from TOKEN.  This primarily
allows the treatment of numbers, money etc."
(cond
((string-matches name "[1-9][0-9]+")
(hcu_hin_number token name))
;; Add many other rules, for numbers, abbrev, alphanumerics, homographs
(t
(list name))))
(define (hcu_hin_number token name)
"(hcu_hin_number token name)
Return list of words that pronounce this number in hin."
;; You have to write this
)
(provide 'hcu_hin_token)
```

For Hindi, I have not made any additions or changes to this file, because the aim of this work is to create a diphone database and token-to-word rules do not have a role in the process of building a diphone database.

4) hin_schema

This is the most important file because it is responsible for creating the diphone list and prompts in the specified language. It includes a list of all the phones in a language in the form of subsets like consonants, vowels, stops, nasals, liquids etc. Basically, a consonant and a vowel list is necessary to make a diphone list. hin_schema has a major role to play in the initial steps of speech synthesis in the following order:

- Listing the phones of the language and the invalid combinations.
- Mapping them onto the phones of already existing languages i.e., bootstrapping.
- Making a list of diphones and nonsense words.
- Synthesizing the prompts in the existing language in prompt-wav and auto-labelling them in prompt-lab.

To begin with, the first and the most important step is to list the phones of the target language. The quality of output depends on the efficient input. Therefore, if the phones are chosen with care such that all possible phone combinations are included, it is most likely that the synthesis will have a better output. All possible phone combinations imply that not only phone combinations within a word but also at word boundaries should also be included. Every language allows phone-phone combinations on the basis of the phonotactic constraints that bar certain combinations of phones. Hence, it becomes important to define the invalid combinations in a language.

The phones are listed in categories – consonants, vowels, etc depending on the requirements of the language. The basic outline of this file is given in an example code in src/diphone/darpaschema.scm but one has to make many relevant changes for the target language. I will be discussing most of the changes with the explanation of the functions of hin_schema. First I will concentrate on the phones I have chosen for Hindi language and the reason for my choice.

I have listed the following phones in Hindi. The most important categories that I have used are the consonants and vowels. In all, I have listed 57 phones. We need to define silence “pau”, because it acts as a phone in silence-phone and phone-silence sequences.

```
;;; A phone list for HINDI
(set! vowels '(a A i I e E u U o O aM AM iM IM eM EM uM UM oM OM))
(set! consonants '(h k g c j T D t d p b R kh gh ch jh Th Dh th dh ph bh Rh n m
ng ny N y r l v s S sh z f))
(set! silence 'pau)
```

First I will explain the vowels selected for the present work. I have taken 20 vowels into consideration – 10 basic vowels and 10 nasalized vowels.

Basic Vowels - a, A, i, I, e, E, u, U, o, O
Nasalized Vowels - aM, AM, iM, IM, eM, EM, uM, UM, oM, OM.

Nasalization is a marked feature for vowels. It is not present in Festival examples, because it is more observed in Indian Languages. Nasalization is a feature associated with vowels only. Nasalized vowels are shown with a /bindu/ or /chandra bindu/ in orthography. It does not have separate alphabet and this makes many people think that nasalized vowels are not phones. But, I would suggest that nasalized vowels should be defined because they have an independent existence as phonemes in Hindi language. We can find minimal pairs for vowels and nasalized vowels in Hindi. For example, /mAs/ “month” and /mAMs/ “flesh”

are a minimal pair that suggests that nasalized /AM/ is a phoneme.

I am stressing this point because I have observed other synthesis systems for Indian languages that do not include nasalization for all vowels. Nasalization is a feature that cannot be separated from a vowel. One cannot record vowels and nasalization separately and then concatenate them because it is a co-articulation phenomenon, that is, vowel and the nasalization are uttered at one time. As such, I find it absolutely necessary to give nasalized vowels an independent status of phones. In Hindi, we find nasalization associated with all the vowels. This can be observed from the list of minimal pairs in the example for all vowel-and-nasalized vowels pairs. The nasalized vowel is represented with the capital M after the vowel.

/a/ - /aM/	- /rac/ - /raMc/	“Absorb”	–	“Little”
/A/ - /AM/	- /mAs/ - /mA~s/	“Month”	–	“Flesh”
/i/ - /iM/	- /citA/ - /ciMta/	“Pyre”	–	“Worry”
/I/ - /IM/	- /bIRA/ - /bIMRA/	“To take responsibility”	–	“A bundle of sticks or bamboo”
/e/ - /eM/	- /pec/ - /peMc/	“Owl”	–	“Fraud”
/E/ - /EM/	- /pETh/ - /pEMTh/	“Enter”	–	“Shop”
/u/ - /uM/	- /gunjArnA/ - /guMnjArnA/	“To make ends meet”	–	“To echo”
/U/ - /UM/	- /gUdA/ - /gUMndA/	“Pulp”	–	“Knead”
/o/ - /oM/	- /god/ - /goMnd/	“Lap”	–	“Adhesive”
/O/ - /OM/	- /mOj/ - /mOMj/	“Happiness”	–	“Made of some kind of leaves”

With nasalization being such an intricate part of the Hindi phonetics, it seems very justified to include them as separate phones even though they result in an increase in the number of diphones.

Next is the list of Consonants. Hindi Alphabet system is one of the most scientifically organized alphabetic systems and it represents all the sounds of Hindi, unlike English, where the alphabets don’t map to the sounds. Hence listing the consonants was not a difficult choice. While choosing the diphones one point must be kept in mind whether to include those foreign phones that have been incorporated in the language with time. Ideally foreign phones should be considered for the phone set but this means extending the diphone set eventually, which is not advisable because larger the diphone set more the space required. A diphone set should be kept as minimal as possible with all possible

combinations included.

To solve this dilemma, the most obvious choice was to include those foreign phones that have eventually, with time, become an integral part of the phonetic system of the language and cannot be accounted as the allophonic variation. Hindi is one language that has adapted itself to many foreign languages, mostly Arabic/Urdu, Pharsi and English.

There are many phonemes in Hindi that are foreign to the language but only some have been so deeply incorporated that it is difficult to separate them as native and foreign phones. Most of them occur as allophonic variation. To keep my list restrictive and manageable, I have included only two foreign phones in Hindi consonants - /z/ and /f/. Though /q/ - uvular 'k' and /.g/ - uvular 'g' have also made a special place in Hindi especially in Urdu words but they can be regarded for allophonic variation with velar 'k' and velar 'g'. For example, /.gazal/ can be and is pronounced as /gazal/ by most of the Hindi speakers who are not well versed with Urdu. Similarly /qalam/ is also accepted as /kalam/. However, the sounds /z/ and /f/ are more phonemic in nature and cannot have "alibis".

The list of the considered consonants is:
h, k, g, c, j, T, D, t, d, p, b, R, kh, gh, ch, jh, Th, Dh, th, dh, ph, bh, Rh, n, m, ng, ny, N, y, r, l, v, s, S, sh, z, f.

In the above list, there are some more sounds that can be considered controversial for being chosen like /ny/ and /ng/. One can easily take these as concatenation of /n/ + /y/ and /n/ + /g/. But look at the following examples:
/aMng/ (part of body) – It cannot be pronounced as /aM/ + /n/ + /g/. The /n/ and /g/ are inseparable.
/anyAy/ (injustice) – the transition between /n/ and /y/ is not that of concatenation but coarticulation. Hence it is /ny/ and not /n/ + /y/.

I have made an attempt to include all possible instances of co-articulation, and valid phone-phone combinations.

Like any other language, Hindi does not allow all phone-phone combinations. There are some pairs that are not possible because of phonotactic constraints. I have carefully listed out all such combinations by referring to Hindi dictionaries and on the basis of my intuitive knowledge about Hindi as a native speaker.

I have listed the invalid pairs in two categories vowel-vowel (vv) and consonant-consonant

(cc). I have not considered any consonant-vowel or vowel-consonant pairs because most of them occur atleast at word boundaries if not within a word or they may occur in remote possibilities. For vv most of the pairs that are invalid are vowels and nasalized vowels together. The reason behind this decision is more phonological, in the sense that one cannot utter a plain vowel followed by a nasalized vowel or vice-versa because nasalization is a strong feature in Hindi and has a tendency to cross over or assimilate. That is, a vowel tends to become nasalized in the presence of an adjacent nasalized vowel in continuous speech mainly because nasalization is a strong feature. For example, in the Hindi sentence /mAM Am khAnA hE/ “Mother I want to eat Mango”, the /A/ of /Am/ tends to become nasalized in continuous speech because the vowel preceding it is nasalized /mAM/ and also because it is followed by a nasal /m/ in /Am/. This is not represented phonetically because it is a feature of continuous speech. Since in speech synthesis our aim is to achieve natural spoken competence, these phonological factors have to be kept in mind. On the basis of this observation, the invalid vowel pairs are listed in hin_schema as follows:

```
(set! invalid_vv
'((a a aM AM iM IM uM UM eM EM oM OM)
(A a A aM AM iM IM uM UM eM EM oM OM)
(i i I aM AM iM IM uM UM eM EM oM OM)
(I i I aM AM iM IM uM UM eM EM oM OM)
(u u aM AM iM IM uM UM eM EM oM OM)
(U u U aM AM iM IM uM UM eM EM oM OM)
(e e aM AM iM IM uM UM eM EM oM OM)
(E E aM AM iM IM uM UM eM EM oM OM)
(o o aM AM iM IM uM UM eM EM oM OM)
(O O aM AM iM IM uM UM eM EM oM OM)
(aM a A i I u U e E o O aM)
(AM a A i I u U e E o O AM)
(iM a A i I u U e E o O iM IM)
(IM a A i I u U e E o O iM IM)
(uM a A i I u U e E o O uM UM)
(UM a A i I u U e E o O uM UM)
(eM a A i I u U e E o O eM)
(EM a A i I u U e E o O EM)
(oM a A i I u U e E o O oM)
(OM a A i I u U e E o O OM)
))
```

The first symbol in each line is the vowel that cannot be combined with the following vowels. Thus, /a/ in the first line is the vowel that has invalid pairs with following vowels /a/, /A/…

This invalid list has to be considered while making a list of diphones and so the part of the program that makes the diphone pairs for vowels has to be altered. To include all pairs

except for the ones listed in “invalid vv” category I have included the following line in the program:

```
(if (not(member v2 (cdr (assoc v1 invalid_vv ))))
```

The valid vv list will be generated by the following function:

```
(define (list-vvs)
(apply
append
(mapcar
(lambda (v1)
(mapcar
(lambda (v2)
(if (not(member v2 (cdr (assoc v1 invalid_vv ))))
(list
(list (string-append v1 "-" v2))
(append (car vv-carrier) (list v1 v2) (car (cdr vv-carrier))))))
vowels))
vowels)))
```

The constraints on cc pairs mostly apply in cases where one consonant does not occur at word initial positions. Hence it cannot have a cc pair even at word boundaries. All the invalid pairs are keeping in mind the word boundary valid combinations as well as combinations possible within a word.

I have chosen consonant phones like /R/, /Rh/, /ng/, /ny/ etc that occur with a vowel along with it, hence there are more number of invalid pairs in the list in combination with these consonants. Now one can argue why should these be included as phones when it has so many invalid combinations.

The point I would like to stress here is that no matter how many consonants a consonant can combine with, but the fact that it is heard as a distinct sound in speech, not only in a particular environment but otherwise, makes it valid enough to be included as a phone. This can be understood by a simple example, the sound /Rh/ does not occur with more than half the other consonants it is still a phone because it cannot be derived from concatenating /R/ + /h/. It is not just a case of aspiration in the presence of /h/ as we can see in the word /paRhAnA/ “to teach” (here we can hear /R/ and /h/ more distinctly and can be accounted by concatenating them) and /paRhAi/ “study” (here they are heard as one sound /Rh/). It is a distinct sound no matter how infrequent its occurrence is and should have a separate existence in the list.

On the other hand sounds like /nh/, /lh/ etc have not been included as phones on a second

thought because they seem to have very limited usage in very few instances that can be taken care of by concatenation. For example words like /nannhA/ “infant”, /dUlhA/ ”bridegroom” etc. These are not regarded as phones as there are not many instances of it if compared to the entire vocabulary. These minor instances can be taken care of by concatenating the phone with /h/. It also matters how well is the pair /nh/ or /lh/ recorded so that it sounds more like a phone.

The list of invalid cc’s is as follows:

```
(set! invalid_cc
'((h Th Rh ng ny N S h)
(k Rh ng ny N)
(g Rh ng ny N S)
(c Rh ng ny N S)
(j Rh ng ny N S)
(T Rh ng ny N S)
(D g c j T R kh gh ch jh Th dh ph bh Rh ng ny N S sh)
(t Th Dh Rh ng ny N)
(d Th Dh Rh ng ny N)
(p Rh ng ny N)
(b Rh ng ny N)
(R ng ny S)
(kh kh Rh ng ny N S)
(gh R Rh ng ny N S)
(ch Rh ng ny N S)
(jh R jh Rh ng ny N S)
(Th Rh ng ny N S)
(Dh R Th Dh Rh ng ny N S)
(th Rh ng ny N S)
(dh R Th Dh dh Rh ng ny N S)
(ph T D p ph Rh ng ny N S)
(bh T D b R Th Dh ph bh Rh ng ny N S)
(Rh T D R Th Dh Rh ng ny N y S)
(n Rh ng ny N S)
(m Rh ng ny N S)
(ng c j T D t d p b R kh gh ch jh Th Dh dh ph bh Rh ng ny N y r l v s S sh)

(ny k g T D d p b R kh gh c jh Th Dh dh ph bh Rh ng ny N n m y r l v s S sh h)
(N R Th Dh Rh ng ny S)
(y R Rh ng ny N S)
(r D Th Dh Rh ng ny N)
(l Rh ng ny)
(v ng ny N S)
(s D R Dh Rh ng ny N S)
(S g c j D d R kh gh ch jh Dh dh bh Rh ng ny r l s S sh h)
(sh R ch jh Th Dh Rh ng ny N S)
(z R Rh ng ny N S)
(f R Rh ng ny N S)
))
```

In hin_schema, the invalid cc pairs are listed as stated above. The first symbol in each line is the consonant that cannot be combined with the following consonants. Thus, /h/ in the first line is the consonant that has invalid pairs with following consonants /Th/, /Rh/…

This invalid list has to be considered while making a list of diphones and so the part of the program that makes the diphone pairs for consonants has to be altered. To include all pairs except for the ones listed in “invalid cc” category, I have included the following line in the program:

```
(if (not(member c2 (cdr (assoc c1 invalid_cc ))))
```

The valid cc list will be generated by the following function:

```
(define (list-ccs)
(apply
append
(mapcar
(lambda (c1)
(mapcar
(lambda (c2)
(if (not(member c2 (cdr (assoc c1 invalid_cc ))))
(list
(list (string-append c1 "-" c2))
(append (car cc-carrier) (list c1 c2) (car (cdr cc-carrier))))))
consonants))
consonants)))
```

The following matrix shows the combinations of valid and invalid vowels and consonants in Hindi.

	a	A	I	I	u	U	e	E	o	O	aM	AM	iM	IM	uM	UM	eM	EM	oM	OM
a	X	*	*	*	*	*	*	*	*	*	X	X	X	X	X	X	X	X	X	X
A	X	X	*	*	*	*	*	*	*	*	X	X	X	X	X	X	X	X	X	X
I	*	*	X	X	*	*	*	*	*	*	X	X	X	X	X	X	X	X	X	X
I	*	*	X	X	*	*	*	*	*	*	X	X	X	X	X	X	X	X	X	X
u	*	*	*	*	X	*	*	*	*	*	X	X	X	X	X	X	X	X	X	X
U	*	*	*	*	X	X	*	*	*	*	X	X	X	X	X	X	X	X	X	X
e	*	*	*	*	*	*	X	*	*	*	X	X	X	X	X	X	X	X	X	X
E	*	*	*	*	*	*	*	X	*	*	X	X	X	X	X	X	X	X	X	X
o	*	*	*	*	*	*	*	*	X	*	X	X	X	X	X	X	X	X	X	X
O	*	*	*	*	*	*	*	*	*	X	X	X	X	X	X	X	X	X	X	X
aM	X	X	X	X	X	X	X	X	X	X	X	*	*	*	*	*	*	*	*	*
AM	X	X	X	X	X	X	X	X	X	X	*	X	*	*	*	*	*	*	*	*
iM	X	X	X	X	X	X	X	X	X	X	*	*	X	X	*	*	*	*	*	*
IM	X	X	X	X	X	X	X	X	X	X	*	*	X	X	*	*	*	*	*	*
uM	X	X	X	X	X	X	X	X	X	X	*	*	*	*	X	X	*	*	*	*
UM	X	X	X	X	X	X	X	X	X	X	*	*	*	*	X	X	*	*	*	*
eM	X	X	X	X	X	X	X	X	X	X	*	*	*	*	*	*	X	*	*	*
EM	X	X	X	X	X	X	X	X	X	X	*	*	*	*	*	*	*	X	*	*
oM	X	X	X	X	X	X	X	X	X	X	*	*	*	*	*	*	*	*	X	*
OM	X	X	X	X	X	X	X	X	X	X	*	*	*	*	*	*	*	*	*	X

This is the VV combination file. *The crosses (X) denote the invalid pairs. The stars (*) denote the valid pairs.*

	k	g	c	j	T	D	t	d	p	b	R	kh	gh	ch	jh	Th	Dh	th	dh	ph	bh	Rh	ng	ny	N	n	m	y	r	l	v	s	S	sh	h	z	f
k	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	X	X	*	*	*	X	X	X	X	*	*	*	*	*	*	*	*	*	*	*	*
g	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	X	X	*	*	*	X	X	X	X	*	*	*	*	*	*	*	X	*	*	*	*
c	*	*	*	*	X	X	*	*	*	*	*	*	X	*	X	X	X	X	*	*	*	X	X	X	X	*	*	*	*	*	*	X	X	*	*	*	*
j	*	*	X	*	X	X	*	*	*	*	*	X	X	X	*	X	X	X	*	*	*	X	X	X	X	*	*	*	*	*	*	X	X	X	*	*	*
T	*	*	*	*	*	X	*	*	*	*	*	*	X	*	*	*	X	X	*	*	*	X	X	X	X	*	*	*	*	*	*	X	X	*	*	*	*
D	*	X	X	X	X	*	X	X	X	*	X	X	X	X	X	X	*	X	X	X	X	X	X	X	X	*	*	*	*	*	*	X	X	X	*	*	*
t	*	*	*	*	X	X	*	*	*	*	*	*	*	*	*	X	X	*	*	*	*	X	X	X	X	*	*	*	*	*	*	X	X	*	*	*	*
d	*	*	*	*	X	X	*	*	*	*	*	*	*	X	X	X	X	X	*	*	*	X	X	X	X	*	*	*	*	*	*	*	X	*	*	*	*
p	*	*	*	*	*	X	*	*	*	*	*	*	*	*	*	X	X	*	*	*	*	X	X	X	X	*	*	*	*	*	*	*	X	*	*	*	*
b	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	X	X	*	*	*	*	X	X	X	X	*	*	*	*	*	*	*	X	*	*	*	*
R	*	*	*	X	X	X	*	*	*	*	*	*	*	*	*	X	X	*	*	*	*	*	X	X	*	*	*	*	X	X	*	*	X	X	*	X	X
kh	*	*	*	?	?	*	*	*	*	*	*	X	X	X	X	X	X	X	X	X	X	X	X	X	X	*	*	*	*	*	*	*	X	*	X	*	*
gh	X	X	X	X	*	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	*	*	X	*	*	*	X	X	X	X	*	*
ch	X	X	X	X	X	X	*	*	*	*	*	X	X	X	X	X	X	X	X	*	X	X	X	X	X	*	*	X	*	*	*	X	X	X	X	*	*
jh	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	*	X	X	*	X	X	X	X	X	X	*	*
Th	X	X	X	*	X	X	*	*	*	*	*	X	*	X	X	X	X	X	*	X	X	X	X	X	X	*	X	*	*	*	X	*	X	*	X	*	*
Dh	X	X	X	X	X	X	*	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	*	X	*	X	X	*	X	X	X	X	*	*
th	*	X	*	X	X	X	X	X	*	X	*	X	X	X	X	X	X	X	X	X	*	X	X	X	X	*	X	*	*	*	*	X	X	X	X	*	*
dh	*	X	X	X	X	X	*	X	*	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	*	*	*	*	*	*	X	X	X	X	*	*
ph	*	X	X	X	X	X	X	X	X	X	*	X	X	X	X	X	X	X	X	X	X	X	X	X	X	*	X	*	X	X	X	X	X	X	X	*	*
bh	*	X	X	X	X	X	X	*	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	*	X	*	*	X	X	X	X	X	*	*	*
Rh	X	X	X	X	X	X	*	*	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	*	X	X	X	X	X	*	X	X	X	X	X
ng	*	*	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	*	X	X	X	X	X	X	X	X	X	X	X
ny	X	X	*	*	X	X	*	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
N	*	*	X	X	*	*	*	X	*	X	X	*	*	X	X	X	X	X	*	*	*	X	X	X	*	*	*	*	X	X	*	*	X	*	X	X	X
n	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	X	X	X	X	*	*	*	*	*	*	*	X	*	*	*	*
m	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	X	X	X	*	*	*	*	*	*	*	*	X	*	*	*	*
y	*	X	*	*	X	X	*	*	*	*	X	X	X	X	X	X	X	X	X	*	X	X	X	X	X	*	*	*	*	*	*	*	X	*	*	*	*
r	*	*	*	*	*	X	*	*	*	*	*	*	*	*	*	X	X	*	*	*	*	X	X	X	X	*	*	*	*	*	*	*	*	*	*	*	*
l	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	X	X	*	*	*	*	X	X	X	*	*	*	*	*	*	*	*	*	*	*	*	*
v	*	*	*	*	X	X	*	*	*	*	*	X	X	X	X	*	X	X	*	X	X	X	X	X	X	*	*	*	*	*	*	*	X	*	*	*	*
s	*	*	X	X	*	X	*	*	*	*	X	*	*	X	X		X	*	*	*	X	X	X	X	X	*	*	*	*	*	*	*	X	X	X	*	*
S	*	X	X	X	*	X	*	X	*	X	X	X	X	X	X	*	X	X	X	*	X	X	X	X	*	*	*	*	X	X	*	X	X	X	X	X	X
sh	*	X	*	X	X	X	*	*	*	*	X	*	*	X	X	X	X	X	*	X	X	X	X	X	X	*	*	*	*	*	*	X	X	*	X	*	*
h	*	*	*	*	*	*	*	*	*	*	*	*	X	*	X	X	X	X	X	*	*	X	X	X	X	*	*	*	*	*	*	*	X	*	X	*	*
z	*	*	*	*	*	*	*	*	*	*	X	*	*	*	*	*	*	*	*	*	*	X	X	X	X	*	*	*	*	*	*	*	X	*	*	*	*
f	*	*	*	*	*	*	*	*	*	*	X	*	*	*	*	*	*	*	*	*	*	X	X	X	X	*	*	*	*	*	*	*	X	*	*	*	*

This is the CC combination file. *The crosses (X) denote the invalid pairs. The stars (*) denote the valid pairs.*

On the basis of the list of phones, ***hin_schema*** and ***diphlist*** file together create a list of diphones and add a nonsense carrier before and after the diphone. The list is generated in an order of combinations silence-vowel, vowel-silence, silence-consonant, consonant-silence, consonant-vowel vowel-vowel and consonant-consonant. One can also have a

single list for consonant-vowel-consonant combination instead of two separate lists, whereby two diphones will be extracted from each nonsense word, consonant-vowel and vowel-consonant as has been done for Hindi in the present case. If other categories of sounds have been specified then the list will include them as well for instance nasal-vowel etc. The order of combinations can be specified in the program itself.

The list is generated in the folder called Etc in the main folder hcu_hin_sms_diphone as hindiph.list. Each line in the list is numbered; numbering is special because it has the name of the language followed by the number, for example hin_0001, hin_0002…hin_2436. Every line has one diphone with carrier and diphone name at the end. An example is given below:

```
( hin_0001 "pau a t aa pau" ("pau-a") )
```

In the above example, hin_0001 is the number of the diphone, “pau a t aa pau” is the nonsense word that has to be recorded, where “t aa pau” is the carrier and “pau-a” is the diphone.

A simple function is defined to enumerate the phone-phone combinations. Following is an example of the function that creates a consonant-consonant list. This is a simple function that specifies that every consonant be combined with every other consonant in the consonant list. The list of consonants and vowels has been explained earlier.

For instance, if there is a list (a b c) the combinations derived will be (aa), (ab), (ac), (ba), (bb), (bc), (ca), (cb), (cc). The diphones are also generated in this manner. Since the number of diphones is the square of the number of phones except for the phonotactic constraints of the target language we get all possible combinations of phones.

```
(define (list-ccs)
(apply
append
(mapcar
(lambda (c1)
(mapcar
(lambda (c2)
(list
(list (string-append c1 "-" c2))
(append (car cc-carrier) (list c1 c2) (car (cdr cc-carrier))))))
consonants))
consonants)))
```

(I have added a line for invalid pairs for Hindi phones as explained above.)

For Hindi the list has been created for the following combinations in this order:

Silence-vowel – silv
Vowel-silence – vsil
Silence-consonant – silc
Consonant-silence – csil
Consonant-vowel-consonant – cvc
Vowel-vowel - vv
Consonant-consonant – cc

The list is created with the help of the following command:

festival –b festvox/diphlist.scm festvox/hin_schema.scm '(diphone-gen-schema "hin" "etc/hindiph.list")'

*(In this the hin_schema and diphlist.scm together make the list for Hindi diphones in the **Etc** directory.)*

The carrier is a nonsense addition to the beginning and/or end of the diphone to generate a nonsense word. The carrier for each language can be specified depending on the phonotactic constraints. Generally, the carrier has the most frequent and common sounds in the language that can be pronounced with almost all other sounds. For instance, in Hindi the most suitable sounds for a carrier are /a/ and /t/. However, in language where /t/ is not so frequent, /k/ or /d/ or any other sound can act as a carrier. The idea is that the sound should be pronounceable with any other consonant or vowel in the language so that it provides for an easy pronunciation of the nonsense word.

This nonsense carrier is important for a monotone pronunciation of the diphone. The diphone is generally in the middle of the word. The reason for a nonsense word is that it is a surety that all the diphones will be recorded in a chronological order and there would not be any tone or accent in the word. The diphones will be pronounced with an acceptable consistent duration and prosody. Another factor is that if the carrier is same in all similar kind of words i.e., if all consonant-consonant combinations have same carrier for example, it will be easy to segment the diphones from the list by cutting of the carrier and extracting the desired sounds. If "real" words are taken in a language instead of nonsense words, it will be a painful process to look for words that may include all the phone-phone combinations and care will have to be taken to find out if all the diphones have been

included in the list. Also while pronouncing “real” words it is most likely that the speaker utters them in an accent. This will bring about a disparity in the pitch of various diphones and may result in a non-continuous speech. Even to segment the desired diphone from a list of “real” words is not practical because what part should be cut has to be specified for all words. Thus the use of this nonsense carrier also has a lot of “sense” in it.

The carriers are different for different set of words for example; silence-vowel will have a different carrier than vowel-silence because the carrier is an aid to make the nonsense words well pronounced and depends on the syllable pattern. Generally a consonant-vowel alternation is aimed at except for cc and vv pairs. This is for the convenience of pronunciations. Below is a list of the carrier used in Hindi with their corresponding phone-phone combination as stated in the hin_schema program. In this list we observe two bracketed carriers - ((pau t aa) (aa pau)), the first bracket indicates the carrier that occurs before the diphone and the latter occurs after the diphone. In case of silences, there is only one carrier, either at the beginning or at the end, depending on the combination.

```
(set! cvc-carrier '((pau t aa ) (aa pau)))
(set! cc-carrier '((pau t aa ) (aa t aa pau)))
(set! vv-carrier '((pau t aa t) (t aa pau)))
(set! silv-carrier '(() (t aa pau)))
(set! silc-carrier '(() (aa t aa pau)))
(set! vsil-carrier '((pau t aa t ) ()))
(set! csil-carrier '((pau t aa t aa) ()))
```

Another important feature of hin_schema is “bootstrapping”. This has to be done to setup functions to generate prompts for those languages that do not exist in the synthesizer. In such cases, the phones of an already existing language have to be overlapped onto phones of non-existing language while generating the prompts. This mapping or overlapping is called “bootstrapping”. A simple mapping is given between the target phone set and existing synthesizer’s phone set. At synthesis time each phone of the non-existing language must be overlapped to an equivalent existing phone. If the equivalent is not present, i.e. if the concerned languages have different sounds, the closest equivalent is taken into consideration. Thus if Hindi /R/, alveo-palatal retroflex is not present in US English, it is mapped with English /r/, trill. This mapping is done through a simple table.

For Hindi, bootstrapping is done in the table `hin2radio_map`, which gives closest phone for each Hindi sound in US English. In the table below we can see that the first sound symbol is the Hindi sound and the second is the closest equivalent sounds in English. One important thing that must be noted is that if one is using “pau” instead of #, then one has to map even this in the mapping table otherwise one may get errors, where after synthesis the

system cannot recognize “pau”. Hence one has to be careful for every symbol that is being used. In case the mapping is not specified, either the system uses the default mapping i.e., maps the phone to it or does not recognize it.

Finally the Define_word_prompt has to be defined to actually do the mapping. This part of the program can be taken from the Japanese example or the Festvox manual and does not need any alterations. This mapping is not only mapping of the sound symbols but actual sounds in the process of synthesis, while generating prompts, recording etc. Thus this mapping is mapping of phones as pronounced in the existing phone set with their durations. The sounds of the non-existing language take up default durations on the basis of this mapping and even the label files are generated automatically based on this mapping, whereby the labelling are based on the phones from the original language but the acoustic signals are from the target language. Therefore the nearest equivalent is chosen while mapping in case of the absence of sound in the existing language.

```
(set! hin2radio_map
'((a ah)
(A ah)
(i iy)
(I iy)
(u uw)
(U uw)
(e eh)
(E eh)
(o ow)
(O ow)
(aM ah)
(AM ah)
(iM iy)
(IM iy)
(uM uw)
(UM uw)
(eM eh)
(EM eh)
(oM ow)
(OM ow)
(kh k)
(gh g)
(c ch)
(j jh)
(T t)
(Th t)
(D d)
(Dh d)
(t th)
(d dh)
```

```
(ph p)
(bh b)
(R r)
(Rh r)
(ng n)
(ny n)
(N n)
(S s)
(sh s)
(h ah)
(y l)
(# pau)
))
```

The file hin_schema plays a very important role in the initial steps of speech synthesis especially in making the list and generating and synthesizing the prompts. We have already discussed the diphone list in detail. Now when the list is ready next step is to synthesize the prompt in an already existing voice in festival. This synthesized voice is played to the speaker while recording the prompts in the target language. The command bin/prompt_them etc/hindiph.list is used for recording the prompts.

The synthesized voice is played to the speaker who then reiterates or mimics the prompt to record in his/her voice. This is done with a dual motive – one, if the speaker mimics (in his own voice and language) the already synthesized prompt (in an existing voice), it is most likely that the prosodic style will remain constant, i.e., the speaker will keep a monotone while recording and it is less prone to pronunciation errors, and second, the speaker will less likely record a wrong diphone. However the problem occurs when the diphone database is being built in a new language and all sounds do not map, then, the nearest equivalent is used instead.

Since the synthesized existing language will not have the sound, the pronunciation of the actual sound of the target language will be alien and the person has to be very careful while recording and should avoid mimicking. The prompts are synthesized before recording with the help of the following command:

```
festival –b festvox/diphlist.scm festvox/hin_schema.scm '(diphone-gen-waves
"prompt-wav" "prompt-lab" "etc/hindiph.list")'
```

The basic code for generating and synthesizing the prompts is in diphlist.scm. hin_scema together with diphlist.scm synthesizes the prompts in the directory prompt-wav and autolables them in prompt-lab for all the diphones listed in hindiph.list. Before synthesizing, the voice in Festival has to be set in which the prompts

will be synthesized in the function Diphon_Prompt_Setup in hin_schema (for Hindi; for US English it will be us_schema). The function also sets up the F0 value for a monotone recording. This value is the middle range of the speaker and is set through the variable FP_F0. For Hindi, KAL, US male voice from the synthesizer is set with FP_F0 as 90 (for male voice).

```
(define (Diphone_Prompt_Setup)
(voice_kal_diphone)
(set! FP_F0 90)
(set! diph_do_db_boundaries nil)
 )
```

To sum up, the file hin_schema lists the phones of the language in various categories and specifies the invalid combinations that should be excluded, maps them onto the phones of already existing voice in the synthesizer (bootstrapping), creates a list of diphones and nonsense words with carriers in the etc directory (hindiph.list), synthesizes the prompts in an existing voice to aid recording in the directory prompt-wav and autolabels them in prompt-lab. The basic code for generating the list and prompts is written in diphlist.scm and the language specific information is provided in hin_schema, together both the files perform the above stated functions.

5) hcu_hin_phones:

In this file all the phones of the specified language as listed in hin_schema are defined in terms of their feature matrix. The features are in different categories like front, back, high, low, place of articulation etc. and given a sequence in numbers or the short form of the feature, for example, '-' is for absence, 1,2,3… or b, p, v are for the presence of different features say in place of articulation for bilabial, palatal, velar etc. and '0' is for not applicable. One can add more categories if required.

Some features that define a vowel cannot be applied to a consonant, in such a case marking is 0, for example, the feature *vowel frontness* has three parameters – front, mid and back indicated by 1, 2, 3 respectively. But for a consonant, vowel frontness is not applicable, hence it is marked 0 for consonants. Thus the phones are listed with the feature sequences for representing the presence or absence of that particular feature and a matrix if formed. For instance, a sound /a/ will be defined in a feature matrix as:

```
(a + a 3 2 - - 0 0 0 0)
```

The + indicates presence of a feature in an either-or situation, i.e. when the feature is either there or not there; 'a', '3' and '2' indicate the feature sequence especially in cases where there are more choices for features, e.g. 'a' is for schwa; '-' indicates that the particular

feature is not present and the '0' indicates the feature is not applicable to this sound. The categories of features and their markings are defined earlier. When the directory structure is created it does not have any phone list. One can see a demo of it in us phone set in Festival.

Hindi phone set has been defined on the basis of the following features:

a) Vowel or Consonant – It indicates whether the sound is a vowel or a consonant. The feature is called vc and +, - indicate the presence or absence of the feature, i.e., if the sound is a vowel, the feature matrix shows '+' and if a consonant, it shows '–'. It is stated as follows:

```
(;; vowel or consonant
(vc + -)
```

b) Vowel Length – It indicates whether the vowel is short, long, a diphthong or a schwa. It is called vlng. As linguists would put it, vowel length could be either short or long, but here it also indicates if the vowel is a diphthong or a schwa. The symbols used to suggest the features are 's', 'l', 'd', 'a', '0' where 's' stands for short, 'l' for long, 'd' for diphthong, 'a' for schwa and '0' for not applicable.

```
;; vowel length: short long diphthong schwa
(vlng s l d a 0)
```

c) Vowel Height – Also called vheight, it indicates vowel height on a five-point scale – high, higher mid, mid, lower mid, low. Numbers 1, 2, 3, 4, 5 and 0 suggests the height of the vowel. 1 stands for high, 2 for higher-mid, 3 for mid, 4 for lower-mid, 5 for low and 0 for not applicable. It is written as follows in the file:

```
;; vowel height: high higher-mid mid lower-mid low
(vheight 1 2 3 4 5 0)
```

d) Vowel Frontness – This indicates whether the vowel is front, mid or back. This is called vfront and is ndicated by numbers 1, 2, 3, and 0. 1 for front, 2 for mid, 3 for back and 0 for not applicable.

```
;; vowel frontness: front mid back
(vfront 1 2 3 0)
```

e) Lip Rounding - Lip rounding indicates whether the vowels are rounded or unrounded. The markers are '+' for presence and '-' for absence. '0' indicates if the feature is not applicable as would be the case with consonants. This feature is specific for vowels and so it is called vrnd in short.

```
;; lip rounding
(vrnd + - 0)
```

f) Vowel Nasalization – This feature has been specifically included for Hindi language as it plays a very significant role in phonetics and phonology of Hindi. I have justified my choice for nasalized vowels while describing the list of phones in hin_schema. It is not present in Festival examples, because it is more observed in Indian Languages. Nasalization is a marked feature for vowels that cannot be separated from a vowel. One cannot record vowels and nasalization separately and then concatenate them. Nasalization is a co-articulation phenomenon, that is, vowel and the nasalization are uttered at one time. Hence nasalization has not only been included as a phone but also as separate feature category even in the feature matrix. Vowel Nasalization is called `vnal` and has a '+' and '-' marking for presence and absence respectively. '0' indicates not applicable (for consonants).

```
;; vowel nasalization
(vnal + - 0)
```

g) Consonant Type – This feature is specifically for consonants and is a non-technical term for *manner of articulation*. In short, it is called `ctype`. It categorizes consonants into 7 types – stop, fricative, affricative, nasal, lateral, approximant and flap. These are marked as 's', 'f', 'a', 'n', 'l', 'r', 'p' respectively and '0' for not applicable. It is written as follows in the file:

```
;; consonant type: stop fricative affricative nasal lateral approximant flap
(ctype s f a n l r p 0)
```

h) Place of Articulation – Place of articulation indicates the orification of the mouth while pronouncing consonants. In short, it is called `cplace`. There are seven places of articulation – labial, alveolar, palatal, labio-dental, dental, velar, and glottal. The symbols that mark these are 'l', 'a', 'p', 'b', 'd', 'v', 'g' respectively and '0' for not applicable.

```
;; place of articulation: labial alveolar palatal labio-dental dental velar ;;glottal
(cplace l a p b d v g 0)
```

i) Consonant Voicing – This feature indicates whether the consonant is a voiced or a voiceless consonant. Also called `cvox` for consonant voicing, it has markings '+' for presence, '-' for absence of the feature and '0' for not applicable.

```
;; consonant voicing
(cvox + - 0)
```

j) Consonant Aspiration – Aspiration is another important feature of Indian languages mostly Indo-Aryan Languages. This feature is present in English in allophonic variation with /k/, /p/ and /t/ when they occur at word initial position. However, in Hindi aspiration is strong, has a phonemic status and also has an orthographic representation. All stops, palatals and velar sounds have aspirated counterparts in Hindi. Aspirated consonants in Hindi are - /kh/,

/gh/, /ch/, /jh/, /Th/, /Dh/, /th/, /dh/, /ph/, /bh/, /Rh/. Aspiration is also present as an allophonic variation in /rh/, /nh/, /mh/ but these do not have a phonemic status. This feature has therefore been added in the feature matrix. It is called `casp` and is marked by '+' for presence, '-' for absence and '0' for not applicable.

```
;; consonant aspiration
(casp + - 0)
```

After all the features have been stated, a list is made with all the phones and features are marked with their specific symbols. One important point is that even silence is considered as a phone in a phone set, as such it should also be defined in the feature matrix. The entire list of feature matrix for Hindi phones is given below. The categories are in the order as explained above. For convenience, the order in short forms is – vc, vlng, vheight, vfront, vrnd, vnal, ctype, cplace, cvox, casp.

```
;; Phone set members
(
(a + a 3 2 - - 0 0 0 0)
(A + l 5 3 - - 0 0 0 0)
(i + s 2 1 - - 0 0 0 0)
(I + l 1 1 - - 0 0 0 0)
(u + s 2 3 + - 0 0 0 0)
(U + l 1 3 + - 0 0 0 0)
(e + s 3 1 - - 0 0 0 0)
(E + l 4 1 - - 0 0 0 0)
(o + s 3 3 + - 0 0 0 0)
(O + l 4 3 + - 0 0 0 0)
(aM + s 3 2 - + 0 0 0 0)
(AM + l 5 3 - + 0 0 0 0)
(iM + s 2 1 - + 0 0 0 0)
(IM + l 1 1 - + 0 0 0 0)
(uM + s 2 3 + + 0 0 0 0)
(UM + l 1 3 + + 0 0 0 0)
(eM + s 3 1 - + 0 0 0 0)
(EM + l 4 1 - + 0 0 0 0)
(oM + s 3 3 + + 0 0 0 0)
(OM + l 4 3 + + 0 0 0 0)
(k - 0 0 0 0 0 s v - -)
(g - 0 0 0 0 0 s v + -)
(c - 0 0 0 0 0 s p - -)
(j - 0 0 0 0 0 s p + -)
(T - 0 0 0 0 0 s a - -)
(D - 0 0 0 0 0 s a + -)
(t - 0 0 0 0 0 s d - -)
(d - 0 0 0 0 0 s d + -)
(p - 0 0 0 0 0 s l - -)
(b - 0 0 0 0 0 s l + -)
(R - 0 0 0 0 0 p a + -)
(kh - 0 0 0 0 0 s v - +)
(gh - 0 0 0 0 0 s v + +)
(ch - 0 0 0 0 0 s p - +)
```

```
(jh - 0 0 0 0 0 s p + +)
(Th - 0 0 0 0 0 s a - +)
(Dh - 0 0 0 0 0 s a + +)
(th - 0 0 0 0 0 s d - +)
(dh - 0 0 0 0 0 s d + +)
(ph - 0 0 0 0 0 s l - +)
(bh - 0 0 0 0 0 s l + +)
(Rh - 0 0 0 0 0 p a + +)
(ng - 0 0 0 0 0 n v + -)
(ny - 0 0 0 0 0 n p + -)
(N - 0 0 0 0 0 n a + -)
(n - 0 0 0 0 0 n d + -)
(m - 0 0 0 0 0 n l + -)
(y - 0 0 0 0 0 r p + -)
(r - 0 0 0 0 0 r a + -)
(l - 0 0 0 0 0 l a + -)
(v - 0 0 0 0 0 f b + -)
(h - 0 0 0 0 0 f g - +)
(s - 0 0 0 0 0 f a - -)
(S - 0 0 0 0 0 f p - -)
(sh - 0 0 0 0 0 f a - +)
(z - 0 0 0 0 0 f a + -)
(zh - 0 0 0 0 0 f a + +)
(nh - 0 0 0 0 0 n d + +)
(lh - 0 0 0 0 0 l a + +)
(rh - 0 0 0 0 0 r a + +)
(pau - 0 0 0 0 0 0 0 0 0)
 ) )
```

6) hcu_hin_sms_int

This file specifies the intonation. It does not require any alterations as such. This program is responsible for giving default intonation to the synthesized speech in the absence of text analysis and prosody building, that is, when text analysis has not been performed and no information on intonation, stress or prosody has been provided, it uses the defaults as provided by Festival.

However this file plays a very important role even if no information is provided on prosody. In a simple speech synthesis diphone database in a male voice, it would not even strike that this file could have such a major role to play without the language specific information on prosody. I have developed the diphone database for Hindi language in a female voice. Initially, I encountered a major problem, in spite of my utterances being recorded in female voice, after synthesis I got the output in a male voice. (This problem has been discussed in Chapter 6 in detail.) Below is a part of the program that defines the default intonation values for a monotone voice.

```
(define (hcu_hin_sms_targ_func1 utt syl)
```

```
"(hcu_hin_sms_targ_func1 utt syl)
Simple hat accents."
(let ((start (item.feat syl 'syllable_start))
(end (item.feat syl 'syllable_end))
(ulen (item.feat (utt.relation.last utt 'Segment ) 'segment_end))
nstart nend fustart fuend fuend fstart fend)
(set! nstart (/ start ulen))
(set! nend (/ end ulen))
(set! fustart '130)
(set! fuend  '110)
(set! fstart  (+ (* (- fuend fustart) nstart) fustart))
(set! fend    (+ (* (- fuend fustart) nend) fustart))
```

The fustart and fuend values define the target points for each syllable. These values are 130 and 110 respectively. As we know, pitch for a male voice has a range from 90-140, and intonation in simple terms can be said to be pitch variation, the default values normalize all the syllables to a pitch of 130-110. As such the voice sounds like a male voice. If these values are changed to 220 for fustart and 200 for fuend, the voice sounds a female voice.

In this, I have changed the fustart and fuend values as follows:

```
(set! fustart '220)
(set! fuend  '200)
```

There is another value that plays an important role in female voice, that is, the Fixed Prosody (FP), which sets the F0 values. We have to set the FP_F0 value for 220 for a female voice. FP_F0 is the Fixed Prosody for F0 value for the monotone utterance. As stated in the Festival manual, FP_F0 is: "*In using Fixed_Prosody as used in Phone type utterances and hence Say Phones, this is the value in hertz for the monotone F0*." This value has to be added only in case of a female voice because the value is already set to 100 for a male voice in the Festival files.

```
(set! FP_F0 200)
```

I realized the necessity of this addition when from the same diphone database, I got a female voice for SayText command and a male voice for SayPhones command. It is so because Fixed Prosody sets up the F0 values for the phones type of utterance as used in Say Phones.

7) hcu_hin_sms_dur

This file specifies the language and voice specific duration for each phone. These durations are basic averages that are used in the synthesized speech. Festival is based on the fixed duration model for each phone. A value of 100 milliseconds is fixed for all phones as a default. Suppose if we do not make any changes in the file, we will still hear a synthesized utterance though the speech sounds too artificial. The SayPhones function in Festival uses

a fixed value FP_Duration. The only difference will be that when no durations are specified in this file, it takes the default 100 milliseconds for all phones. However, there is another level of durations model, i.e., average durations model where we can specify durations for each phone in the language. Ideally average durations should be calculated from real utterances but writing values by hand is also reasonably acceptable whereby vowels are longer than the consonants and stops have the shortest duration.

For Hindi, I tried to find average durations for phones by recording some “real” words, calculating their durations in Praat software. Since I could not find averages for each phone from a mere one or two utterances, the best alternative than to make guesses was to find averages for various categories of sounds. Hence, for example, one can notice similar values for stops, aspirated stops, sibilants, nasals, tense and lax vowels and nasalized vowels. Since I did not have ample data to find out the averages, I tried to round up the values of similar categories, that is, suppose I got 140, 145, 159, 162, 154 milliseconds for short vowels, I rounded it up to make an average 150. Similarly for tense vowels I got higher values that I rounded up to 200 and for nasalized vowels I got values close to 300 for vowels /aM/, /eM/ and around 350 for /AM/, /EM/ etc. For stops my averages are close to 75 and for aspirated stops, 100. For nasals, values are a rounded up 125 and for sibilants 150. Thus, even though the values I have given are not exact averages, they are also not absolute guesses.

The duration values for all the phones as stated in hcu_hin_sms_dur are as follows:

```
(set! hcu_hin_sms::phone_data
'(
;;; PHONE DATA
(pau 0.0 0.250)
(a 0.0 0.150)
(A 0.0 0.200)
(i 0.0 0.150)
(I 0.0 0.200)
(e 0.0 0.150)
(E 0.0 0.200)
(u 0.0 0.150)
(U 0.0 0.200)
(o 0.0 0.150)
(O 0.0 0.200)
(aM 0.0 0.300)
(AM 0.0 0.350)
(iM 0.0 0.300)
(IM 0.0 0.350)
(em 0.0 0.300)
(EM 0.0 0.350)
(uM 0.0 0.300)
```

```
(UM 0.0 0.350)
(oM 0.0 0.300)
(OM 0.0 0.350)
(k 0.0 0.75)
(kh 0.0 0.100)
(g 0.0 0.75)
(gh 0.0 0.100)
(c 0.0 0.105)
(ch 0.0 0.125)
(j 0.0 0.100)
(jh 0.0 0.125)
(T 0.0 0.75)
(Th 0.0 0.100)
(D 0.0 0.75)
(Dh 0.0 0.100)
(t 0.0 0.75)
(th 0.0 0.100)
(d 0.0 0.75)
(dh 0.0 0.100)
(p 0.0 0.75)
(ph 0.0 0.100)
(b 0.0 0.75)
(bh 0.0 0.100)
(R 0.0 0.75)
(Rh 0.0 0.100)
(ng 0.0 0.75)
(ny 0.0 0.120)
(N 0.0 0.125)
(n 0.0 0.125)
(m 0.0 0.125)
(y 0.0 0.125)
(r 0.0 0.125)
(l 0.0 0.125)
(v 0.0 0.125)
(s 0.0 0.150)
(S 0.0 0.150)
(sh 0.0 0.150)
(h 0.0 0.150)
```

Since pause or silence is also treated as a phone in the process, we also give it a default duration value of 250.

Most of the phones are longer in stressed syllables, at the word final and even maybe at word initial positions. Since we define durations on averages, a simple multiplicative factor can be used to define the contextual durations, i.e. durations that occur in word initial or final positions. This file also has a small function, which predicts longer durations of phones in stressed syllables and in clause initial and clause final syllable.

```
(set! hcu_hin_sms::zdur_tree
'
((R:SylStructure.parent.R:Syllable.p.syl_break > 1 ) ;; clause initial
((0.6))
((R:SylStructure.parent.syl_break > 1)   ;; clause final
((0.8))
((0.5)))))
```

8) hcu_hin_lex

This file has letter-to-sound rules and the lexicon. This is a very important part of speech synthesis because it is through this medium that the synthesizer maps the diphones on to the written text. A lexicon is necessary when pronunciations cannot be predicted by the orthography, i.e. when the alphabets do not have a one-one relation with the sounds of the system and one alphabet can represent more than one or two sounds in different and unpredictable environments. In such cases the best option is to build a lexicon - a list of most frequently used words with their pronunciations. However, if the language has a one-to-one relation between sounds and alphabets, the most convenient option is to build a letter-to-sound table, i.e. map the alphabets or graphemes to the respective phonemes. Also called grapheme-to-phoneme conversion, this is the most economical way but largely depends on the orthographic system of the language concerned.
Pronunciations in Hindi can almost completely be predicted from its orthography. That is, it nearly has a one-to-one relation between graphemes and phonemes. The following are the letter-to-sound rules for Hindi.

```
(lts.ruleset  hcu_hin ( )
(
( [ a ] = a )
( [ A ] = A )
( [ i ] = i )
( [ I ] = I )
( [ u ] = u )
( [ U ] = U )
( [ e ] = e )
( [ E ] = E )
( [ o ] = o )
( [ O ] = O )
( [ aM ] = aM )
( [ AM ] = AM )
( [ iM ] = iM )
( [ IM ] = IM )
( [ uM ] = uM )
```

```
( [ UM ] = UM )
( [ eM ] = eM )
( [ EM ] = EM )
( [ oM ] = oM )
( [ OM ] = OM )
( [ k ] = k )
( [ kh ] = kh )
( [ g ] = g )
( [ gh ] = gh )
( [ c ] = c )
( [ ch ] = ch )
( [ j ] = j )
( [ jh ] = jh )
( [ T ] = T )
( [ Th ] = Th )
( [ D ] = D )
( [ Dh ] = Dh )
( [ t ] = t )
( [ th ] = th )
( [ d ] = d )
( [ dh ] = dh )
( [ ph ] = ph )
( [ p ] = p )
( [ b ] = b )
( [ bh ] = bh )
( [ R ] = R )
( [ Rh ] = Rh )
( [ ng ] = ng )
( [ ny ] = ny )
( [ N ] = N )
( [ n ] = n )
( [ m ] = m )
( [ y ] = y )
( [ r ] = r )
( [ l ] = l )
( [ v ] = v )
( [ s ] = s )
( [ S ] = S )
( [ sh ] = sh )
( [ h ] = h )
( [ z ] = z )
( [ f ] = f )
 ))
```

The Hindi letter-to-sound rules as you will see above are just a mapping of the same symbols for graphemes and phonemes. It is so because it is difficult to get Hindi font

system in Linux that would be compatible with Festival.

Since Hindi can be written in roman alphabets, I have used here arbitrary symbols to represent the phonemes and the same symbols to represent the graphemes. To make it convenient for the user of this diphone database, I have provided below a list of Hindi alphabets mapped on to the symbols I have used to represent Hindi phoneme and grapheme symbols.

A list of Hindi Vowels (alphabets) and the corresponding symbols (used in the present work).

Hindi Vowels

Hindi Alphabets	Symbols Used
अ	a
आ	A
इ	I
ई	I
उ	u
ऊ	U
ओ	o
औ	O
ए	e
ऐ	E
अं, अँ	aM
आं, आँ	AM
इं, इँ	iM
ईं, ईँ	IM
उं, उँ	uM
ऊं, ऊँ	UM
ओं, ओं	oM
औं, औं	OM
एं, एँ	eM
ऐं, ऐं	EM

A list of Hindi Consonants (alphabets) and the corresponding symbols (used in the present work).

Hindi Alphabets	Symbols Used
क	k
ग	g
च	c
ज	j
ट	T
ड	D
त	t
द	d
प	p
ब	b
ड़	R
ख	kh
घ	gh
छ	ch
झ	jh
ठ	Th
ढ	Dh
थ	th

Hindi Alphabets	Symbols Used
ध	dh
फ	ph
भ	bh
ढ़	Rh
न	n
म	m
ङ	ng
ञ	ny
ण	N
य	y
र	r
ल	l
व	v
स	s
श	sh
ष	S
ह	h
ज़	z
फ़	f

Building a Diphone Database:

In the last section we have explained the parameters that have been set for Hindi language in the relevant files in Festvox. After this initial spadework, we begin with the main task, i.e. of building a diphone database. This section will elaborate and explain the intricate details, as much as possible, of each step involved in the process of building a diphone database. The steps with the commands have been briefly stated in an earlier section in this chapter and some initial steps relevant to the changes in Festvox files have been explained in detail in the last section.

The process of building a diphone database as done for Hindi language is in the following stages:

- Defining a diphone list
- Synthesizing the prompts
- Recording the diphones
- Labelling the diphones
- Hand labelling the diphones (optional but necessary)
- Making the diphone index
- Extracting the Pitchmarks
- Building LPC parameters
- Checking and correcting diphones
- Defining a diphone voice

Defining a diphone list:

This point has been explained in detail in the last section. However, I will summarize the process again here. First, the phones of the target language (Hindi) are defined and if necessary “bootstrapped”, i.e. mapped on to the phones of the existing language in hin_schema.scm. Next, a list of valid phone-phone pairs is made from the list of phones in sets of vowel-vowel, consonant-consonant, consonant-vowel-consonant, silence-vowel, silence-consonant, and vowel-silence and consonant-silence sequences (more sequences can be specified depending on the language).

The phones and their valid combinations are language specific and largely depend on the person’s linguistic knowledge and choices. Then a nonsense carrier is added before and/or after the phone pair to make a nonsense word and a list is generated in Etc/hindiph.list. The list contains the id of file, the nonsense word to be recorded, i.e. the prompt and the diphone that will be extracted from the word.

The file id is auto-generated and the same file id is used throughout – in wav files, label files, pitchmarks etc. This file id is like a specific identity to every diphone.

```
( hin_0001 "pau a t aa pau" ("pau-a") )        ::here pau is silence and is
( hin_0002 "pau A t aa pau" ("pau-A") )        ::considered as a phone
( hin_0003 "pau i t aa pau" ("pau-i") )
( hin_0004 "pau I t aa pau" ("pau-I") )
( hin_0005 "pau e t aa pau" ("pau-e") )
```

Synthesizing the prompts:

The prompts (i.e. the nonsense words) generated in the file hindiph.list are then synthesized in an existing voice by the synthesizer so that it can be played to the speaker while recording. Synthesizing the prompts, in simple terms, is nothing but an automated process whereby the synthesizer “records” the prompts in an existing voice (whichever we choose) and labels those “recordings”. For the synthesizer to be able to “speak” the phones of the new language, we need to bootstrap (or map) the phones of our language to the existing phones of the voice we will be using. For instance, US English does not have Hindi /R/ or /Rh/, so if we map /R/ and /Rh/ both to /r/. Thus the new phones are made recognizable to the system.

The synthesizer will then have the prompts “recorded” in an existing voice in prompt-wav directory in different files and similarly label them in prompt-lab directory. Well, one question always comes up, why is this necessary? Why should prompts be synthesized and played during recordings? Festival provides a simple answer – doing this helps the speaker to maintain a fixed prosodic style while mimicking the prompts because the speaker hears a monotone played and is less likely to bring about an accent, and second, to maintain the sequence of diphones. This is important because we mentioned earlier that there is a unique file id created for every diphone when making a diphone list and all throughout the same file id is used for different processes for each diphone i.e. while recording, labelling, pitchmarking, etc. Synthesizing the prompts is more like a “guide” to the speaker during the recording of the diphones.

The following command is used to synthesize the prompts:

```
festival –b festvox/diphlist.scm festvox/hin_schema.scm ‘(diphone-gen-waves
“prompt-wav” “prompt-lab” “etc/hindiph.list”)’
```

Recording the diphones:

After setting parameters for Hindi language in the files in Festvox, generating the diphone list, and synthesizing the prompts, next step is to record the utterances or the list of nonsense words. The list is created in a chronological order in the directory ETC as hindiph.list as explained in the last section. The diphones are recorded with an objective of getting uniform pronunciation. One could also record “real” words rather than these nonsense words, but there are two problems – one has to find a large number of real words that will include all possible diphones or combinations of phones in the language, which is a very tedious job. Also one has to be very careful while recording them because it is very likely that “real” words will have pronunciation errors. The words should be pronounced with

the least prosodic variation and in as much a monotone as possible.

Recording the nonsense words is done by the following command:

```
bin/prompt_them etc/hindiph.list
```

The synthesized prompts are played to the speaker followed by the recording time where the speaker mimics the played nonsense word. This is followed by reiterating the synthesized prompt and the speakers recording. The recording is played for the speaker to check if it is as desired. The recordings are automatically saved in individual files in the directory – hcu_hin_sms_diphone/wav. The name of the files is the same as auto-generated in the hindiph.list – “hin_xxxx”. The recordings go on continuously till they are stopped by Ctrl+C. To resume the recordings from the point they have been stopped (say 300) one can play the command:

```
bin/prompt_them etc/hindiph.list  300
```

These commands play the entire hindiph.list alternating between synthesized prompts and recording time till the entire list is exhausted or stopped in between. prompt_them uses the commands na_record and na_play to record the words and play them back to the speaker alternately for every utterance. The commands are shown below:

To play the synthesized prompt:

```
$ESTDIR/bin/na_play prompt-wav/$f.wav
```

To record:

```
$ESTDIR/bin/na_record -f 16000 -time $duration -o wav/$f.wav
```

To automate the system the value $duration for the duration of recording and $f for the filename has been set earlier:

```
cat $1 |
awk '{if (NR >= '$POSITION') print $0}' |
while read lll
do
echo $lll
f=`echo $lll | awk '{print $2}'`
duration=`$ESTDIR/bin/ch_wave -info prompt-wav/$f.wav | awk '{if ($1 ==
"Duration:") printf("%d\n",$2+1.5)}'`
```

To review (or replay):

```
$ESTDIR/bin/na_play prompt-wav/$f.wav
::This will play the synthesized prompt.
$ESTDIR/bin/na_play wav/$f.wav
::This will play the recording.
```

However there is another option to record the prompts, that is, to use the commands na_record and na_play directly. But this is a tedious way because one has to change the file name for every recording. It is basically the command for a non-automated recording unlike prompt_them:

```
$ESTDIR/bin/na_record –f 16000 –time 5 –o hin_0001.wav –otype riff
```

Here one has to specify the time and the file name for every instance.

To test the recordings one can play them:

```
$ESTDIR/bin/na_play hin_0001.wav
```

Recording diphones should be done efficiently. The output reflects the quality of the recording. If the recording is done with precaution, after synthesis, the voice will sound clear with less disturbance and noise. One has to take the following precautions while recording the nonsense words.

a) The speaker should be in good health and relaxed. He/she should not be suffering from cold, cough or hangover and should not make an effort while recording because the tension in the voice becomes evident. The speaker should maintain a voice that can be easily repeated at the similar pitch as recordings may stretch on to more than one sitting or some diphones may have to be re-recorded in some cases. Ideally recording should be done in one sitting but generally it gets too tiring and so it may take a couple of days. In such a case the recording should be done at the same time of the day.

b) The recording environment should be quiet and well defined in the sense that if recordings have to be repeated or stretched to another sitting, the environment can be reconstructed with the same settings for volume, microphones, distance etc. The best environments are recording studios, however one may even try it in the privacy of one's room on a PC.

c) The distance between the speaker and microphone is also of considerable importance. Best is the head mounted mikes with the distance of 8mm from the lips rather than fixed microphones because the adjustments are more stable.

d) The quality of hardware in computers should be good and compatible to the system as the quality of recordings may go down if the hardware gives problems, for example if the sound card is not very compatible to Linux 6.2, it may give rise to unwanted disturbances.

I have built the Hindi diphone database on a standard home PC. I have used a Telex head-mounted microphone and recorded in the privacy of my room at midnight. I have purposely chosen the midnight hour because in the early morning the chirping of birds caused disturbance. The problems faced during recordings[S1] have been discussed in detail in Chapter 6.

Labelling and Hand-labelling the diphones:

Festival provides us with the facility of auto labeling, but at the same time advises us that we should check the labels in EMU labeler (preferably) and hand label or hand edit for better results. Now there are two points that I would like to explain, one, why is hand labelling advised and second, how do we do it.

What is labeling?

To begin with, first I will explain the process of autolabelling as done by Festival. Autolabelling identifies the file id as in hindiph.list and picks up each recorded prompt (nonsense word), outlines the boundary of each sound in the wave file, i.e. marks the beginning and end durations for each phone on the basis of the autolabelled synthesized prompts in prompt-wav and prompt-lab and gives each marked sound, its phone symbol as specified in hin_schema. To make it more simple, autolabeling identifies the sounds in a wave file on the basis of the information given in hin_schema, diphlist,scm and hindiph.list and more importantly prompt-wav, prompt-lab and prompt-cep and marks them with the specified symbols.

This process is done by the following command -

```
bin/make_labs prompt-wav/*.wav
```

We have explained earlier why synthesizing the prompts was such an important step. Synthesizing the prompts creates three files - prompt-wav - the synthesized prompts, prompt-lab - the autolabelled files for each prompt that has been synthesized and prompt-cep, that finds out the cepstrum features for each synthesized prompt. In autolabelling the recorded prompts, the process uses the prompt-lab and prompt-cep files and based on it labels the recorded prompts in lab and finds out the cepstrum values in cep. A part from the make_labs code:

```
if [ $# = 0 ]
then
echo align nonsense word phone labels from prompts
echo "Usage: make_labs prompt-wav/*.wav"
echo expects the following to exist:
echo "prompt-wav/*.wav      The waveform prompts"
echo "prompt-lab/*.lab      The labels for the prompts"
echo "wav/*.wav                    The spoken waveforms"
echo "cep/                  Will fill this directory with cepstrum files"
echo "prompt-cep/         Will fill this directory with cepstrum files"
echo "lab/                  Will fill this directory with label files"
exit 1
```

The actual alignment process is:

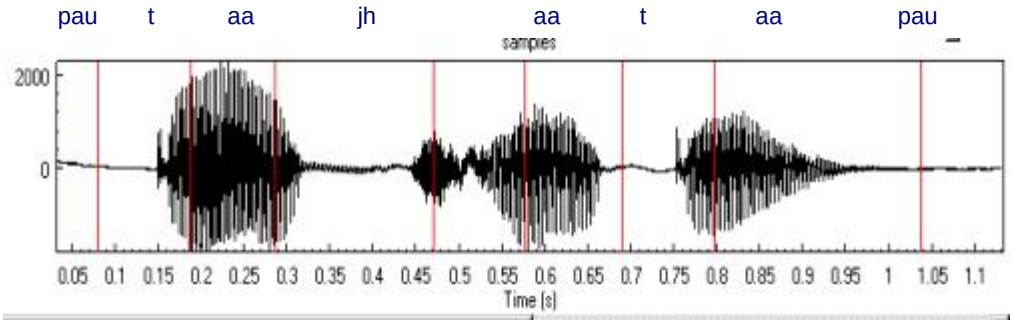
```
$PHONEALIGN -itrack prompt-cep/$fname.cep -ilabel prompt-lab/$fname.lab -
otrack cep/$fname.cep -olabel lab/$fname.lab
```

(Phones are aligned on the basis of prompt-lab and prompt-cep)

The idea is to take a prompt and the spoken word form and derive mel-scale cepstral parametrizations of the files.

Why do we need hand labeling?

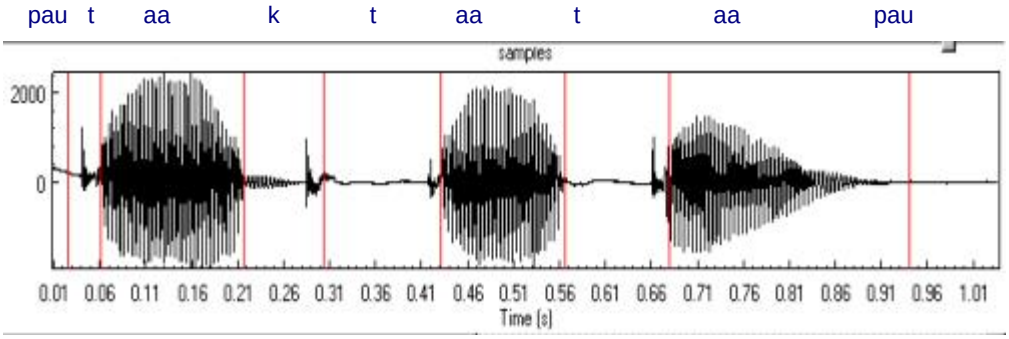
Hand labeling is not a part of the process of building a diphone database. It is optional but desired for a better quality of synthesis. After, autolabelling the entire list of recoded words; one can see the labels with the help of EMU software, which is supported by Festival. Labelling through the Festival code is not always correct, i.e. the boundaries are not correctly marked and as a result, diphones may not be properly segmented and synthesized creating gaps, clicks and errors in the synthesized speech. To avoid such unwanted instances, it is better to look at each label file and readjust (if necessary) the markings at diphone boundaries. Look at the diagram below (taken from an EMU labeler) this is an instance of autolabelling. One can see the erroneous markings.



To rectify it, it is always better to go through the tedious process of hand labeling. The minimum one has to change is at the beginning of the preceding phone to the first phone in

the diphone, the changeover, and the end of the second phone in the diphone. One has to mark the phone boundaries and the diphone is extracted by default from the middle of one phone to the middle of the other.

Following is an example of a hand labelled prompt:

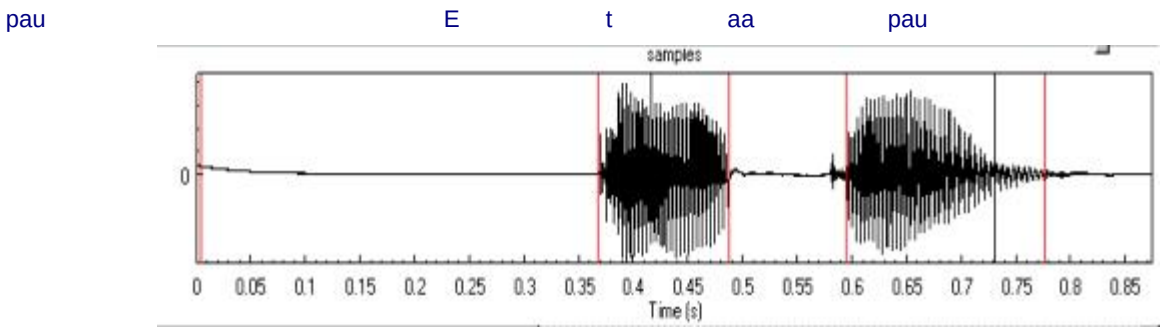


How do we hand label?

Though there are some guidelines given in the Festvox manual as to how one should hand label different kinds of phone pairs, vowel-vowel, vowel-consonant, etc., but hand labeling is mostly a self-developed skill whereby one can find out from experience and little knowledge where to mark, provided one has sufficient knowledge about sound waves and spectrograms.

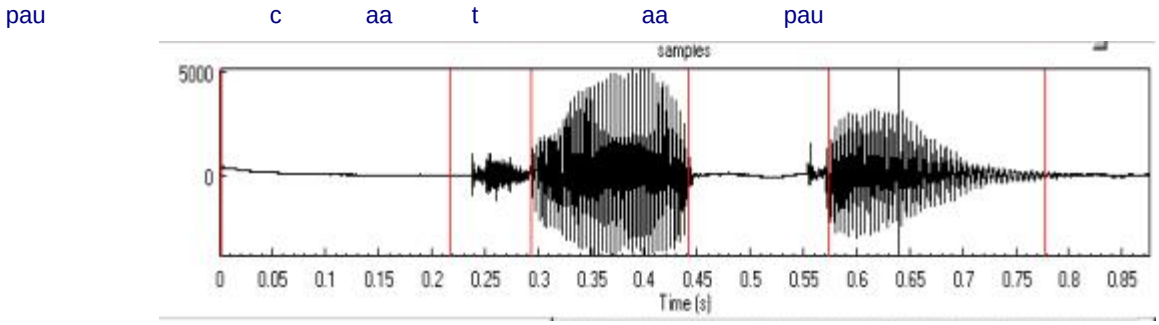
In this section, rather that reiterating what has been already said in the manuals, I will show the way I have hand labeled most of the phone-phone pairs with the help of examples. There are a lot of instances where it is difficult to decide where to mark the boundary; such problem areas and challenges have been explained separately in Chapter 6, as it is difficult to summarize everything here. For convenience, I will give one example for each phone combination here and explain the problematic areas later in Chapter 6.
First I will begin with the silence-phone pairs:

Silence-vowel –



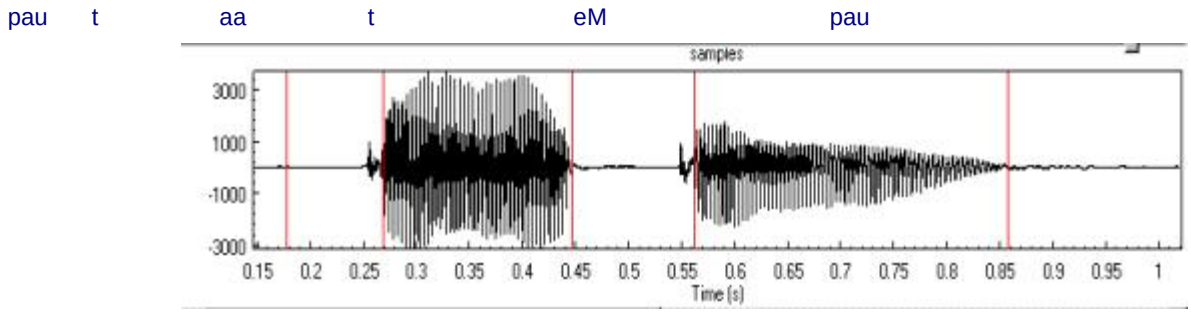
In the diagram above, one can see that the markings, from the silence to the point where the vowel begins, and the end of the waves where vowel ends. I have marked from the point I can distinguish where the vowel begins and ends

Silence-consonant –



Marking a silence-consonant is a little trickier because consonants do not have well-formed formants in spectrograms nor do they have a waveform. The problem areas in labeling have been explained in Chapter 6.

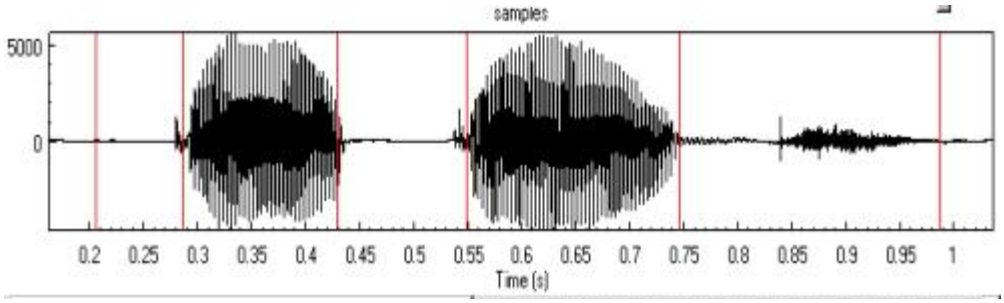
Vowel-Silence –



The phones at the end of the word are longer than word internal phones; as such phone-silence sequences have to be marked with care such that the boundaries of the phone are marked explicitly at the end of the signal i.e. the trailing part of the phone must also be included. The energy levels drop due to the following silence, and will cause an energy jump while concatenating with the other diphones. So if the phone boundary is explicitly marked at the beginning of the phone, the energy levels can be maintained.

Consonant-Silence -

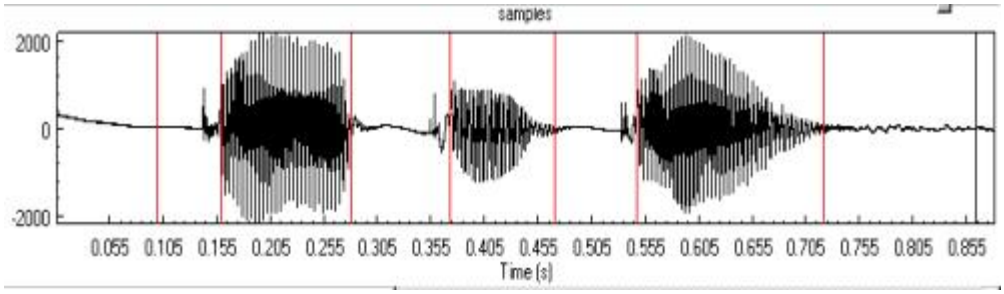
pau t aa t aa j pau



Consonants, generally, except for the sonorants, do not have waveforms and neither do they trail in silence. It is easier to mark the consonant-silence pair as the boundaries can be clearly seen e.g. in stops the burst marks the boundary. Markings should be at the end of the last signal of the phone and then mark the silence.

Consonant-Vowel-Consonant –

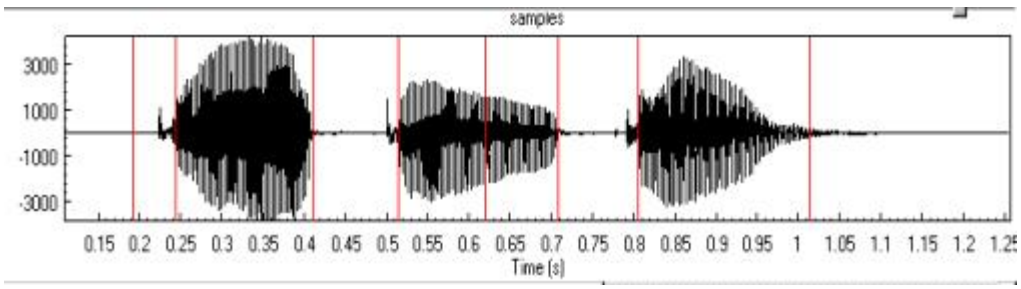
pau t aa p u p aa pau



To mark a c-v-c combination, one has to be careful in some cases because, the consonant may not look the same before and after the vowel in all cases, sometimes the vowel trails off more into the consonant, hence, the second consonant may appear smaller than the first one. Such combinations need a little extra caution while marking the boundaries (See chapter 6).

Vowel-Vowel –

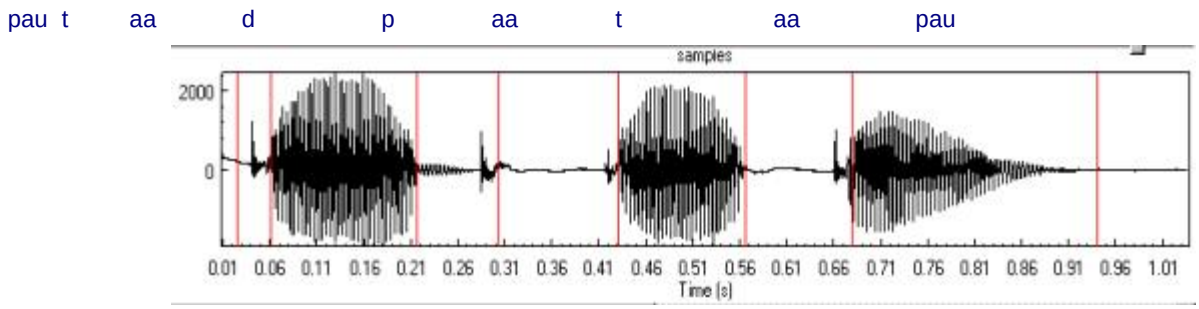
pau t aa t e u t aa pau



Vowel-vowel boundaries are generally one of the difficult areas because both the vowels have their own waveforms. The first vowel very smoothly trails off into the second one such that it is very difficult to find out where one ends and the other starts. The best bet is to

mark in the middle of the changeover. (See chapter 6 for detailed example). In the above example the boundary for /e-u/ has been marked at the center of the changeover between the two vowels.

Consonant-Consonant –



To mark consonant-consonant pairs is also difficult in most cases, as both the phones here do not have well formed waveform. The level of difficulty depends on what kinds of consonants are combined together, stops-stops, stops-sonorants or stops-sibilants etc. Like consonants cause more confusion. This has been discussed in detail in chapter 6.

Making the Diphone Index:

After labelling the recordings, we have to make the diphone index. This takes the list of diphones from etc/hindiph.list and finds out the starting, mid and end values for each diphone and makes an index list in dic/smsdiph.est. These values are basically the durations at which a diphone starts, its middle and end. For example, we can see below a part of code from bin/make_diph_index that states how the index has to be listed:

```
(format dout "%s %s %s %s %s\n"
(nth 0 a) ; diphone
(nth 1 a) ; file
(nth 2 a) ; start
(nth 3 a) ; mid
(nth 4 a) ; end
))
diphindex)
```

The values start, mid and end are durations taken from the lab files after the labeling has been done for each diphone. The code matches the label file and hindiph.list, with the help of file id and returns the index entry. The diphone boundaries are then found with the help of label files.

The index is built in the reverse order, i.e., first the last diphone in hindiph.list is indexed and this continues in the descending order. The index is numbered same as the file id in

hindiph.list. Following command is used to make the diphone index:

```
bin/make_diph_index etc/hindiph.list dic/smsdiph.est
```

The output of this command is saved in “dic” directory in smsdiph.est file as follows:

```
EST_File index
DataType ascii
NumEntries 2614
EST_Header_End
pau-pau hin_2436 1.34592 1.895 1.9475
sh-sh hin_2435 0.348078 0.39 0.44
sh-s hin_2433 0.263718 0.315249 0.365124
sh-v hin_2432 0.280088 0.336981 0.36349
sh-l hin_2431 0.237948 0.281148 0.328074
sh-r hin_2430 0.316193 0.371992 0.400439
sh-y hin_2429 0.440872 0.503187 0.522929
sh-m hin_2425 0.443108 0.514224 0.542112
...
...
pau-A hin_0002 0.205689 0.413567 0.480307
pau-a hin_0001 0.275712 0.551424 0.595188
```

The index first names the diphone e.g. sh-sh, states the file id in hindiph.list e.g. hin_2435 and lists three values for each diphone. The order is from last diphone to the first. This step basically segments the diphones from the nonsense words, i.e. “cuts” the carrier and explicitly indexes these diphones.

Extracting the Pitchmarks:

Pitchmakrs are extracted to find the default pitch periods within the speaker’s maximum and minimum range and finally the unvoiced section is filled with the default pitchmarks. Festival supports two systems to extract pitchmarks – LPC, Linear Predictive Coding (distributed in public version) and PSOLA, (not in public version). Both are pitch synchronous techniques and require information on where pitch periods occur in the acoustic signal. Pitchmarks can be extracted from the EGG – electroglottograph signal (if the recordings are done with an electroglottograph EGG or laryngograph) as well as from the wave files of the recordings by a microphone. However, EGG is a better and easier option, if possible, because it records electrical activity in the glottis during speech.

The following command is used to extract pitch periods EGG:

```
bin/make_pm lar/*.lar
```

We have not used this in our present system.

If one does not have EGG recordings, the following command is used to extract the pitch periods:

```
bin/make_pm_wave wav/*.wav
```

The code in make_pm_wave extracts the pitchmarks from the waveform files in the wav directory. It filters in the incoming waveforms with a low and high band filter and then finds the pitch marks by correlating to the maximum and minimum values specified for the speaker and then fills the unvoiced section with the new found default values.

The value of pitch varies from speaker to speaker but it can be grouped in basically two – male and female voice. The file specifies the minimum and maximum arguments of pitch range, which is speaker dependent. The file, by default, gives values for a male speaker 0.005 and 0.012 in the range 200 to 80 (for high and low pass filter).

```
PM_ARGS='-min 0.005 -max 0.012 -def 0.01 -wave_end -lx_lf 200 -lx_lo 111 -lx_hf
80 -lx_ho 51 -med_o 0'
```

For a female speaker the values min and max values have to be changed to 0.0033 and 0.7 and the range for filters is changed to 300 and 140:

```
PM_ARGS='-min 0.0033 -max 0.7 -def 0.01 -wave_end -lx_lf 300 -lx_lo 111 -lx_hf
140 -lx_ho 51 -med_o 0'
```

After the pitchmarks have been extracted, next post-processing step is to predict pitchmarks to the nearest waveform peaks. However, one has to be careful with this step whether the signal or wave is upside down. In such cases, the code will predict the troughs rather than the peaks of the waveform. To overcome this problem, one can invert the signal by using –inv argument to the arguments of pitchmark.

The quality of synthesis is strongly affected if pitch extraction is not done properly.

Building LPC parameters:

LPC, i.e. Linear Prediction Coding, is the only signal processing method in the public version of Festival. One has to extract LPC parameters and LPC residual files in a diphone database. This should be done after the pitch marks have been extracted. First we need to find the general power difference through a set of diphones because there is often a power mismatch:

```
bin/find_powerfactors lab/*.lab
```

This finds the mean power for each vowel in each file and calculates a factor with respect to overall vowel-mean power. The calculated power factors are saved in ETC directory as

etc/powerfacts. These power factors are used to normalize the power of each waveform file during the making of the LPC coefficients.

The following command is used to extract the LPC values:

```
bin/make_lpc wav/*.wav
```

LPC coefficients are saved in two files (two files for each diphone) in the directory LPC with the extension .lpc and .res, where .lpc is the LPC coefficients and .res is the LPC residual.

Power normalization eliminates the artifacts caused by variations in recording environment or variations in speaker's pronunciation of different diphones. After, pitchmarks have been extracted and pitch normalization, LPC finds out the pitch periods for the diphone database. This is the last step of building a diphone database and now the next job is checking the diphones and defining and integrating the voice in Festival.

Checking and correcting diphones:

At this stage we need to check the diphones. This is done with the help of the file Festvox/hcu_hin_sms_diphone.scm. The basic function of this file and the process of checking the diphones has been explained earlier in the chapter. This voice can be tested at the festival prompt.

It is important to check the diphones because most likely there may be "mistakes" due to mispronunciation while recording, mislabeling, missing diphones etc. To check the diphones, we use the command SayPhones. Checking all the diphones becomes easy if the nonsense words (prompts) are synthesized and played, in this way, all the diphones can be tested in an order. To make it easy, one can even write a small code to make the process automated. Since this is basically a diphone database and no text analysis has been done yet so the output will be a monotone speech.

One can test the diphones as follows:

```
festival> (SayPhones '(pau n a m a s t e pau))
::(/namaste/- greeting in Hindi equivalent to "hello")
```

One can save the synthesized utterance:

```
festival> (set! utt1  (SayPhones '(pau n a m a s t e pau)))
```

```
(utt.save.wave utt1 “namaste.wav”)
```

After checking the diphones, one must find out where the problem occurs (if it occurs), by looking at the whole process step-by-step. One can verify if the diphone is listed in hindiph.list, can listen to the recording to check if it is all right, check the entry in the diphone index, check the labeling and the waveform, pitchmarks etc. Problems can be anywhere. After detecting the problem one needs to rectify it with the possible solutions at each stage. However if the problem persists, one option is to re-record the problematic areas with care such that the environment is the same.

The problems that I faced during my work have been discussed in Chapter 6.

Defining a diphone voice:

After the final diphone database is ready and needs no more improvement, we come to the last step – defining the diphone database as a voice. For this, first we need to group the files of the diphone database so that only the relevant files containing the sub parts of words that contain a diphone are “bunched together” to be integrated in the Festival system.

```
festival festvox/hcu_hin_sms_diphone.scm‘(voice_ hcu_hin_sms_diphone)’
...
festival (us_make_group_file “group/smslpc.group” nil)
```

After these files have been grouped, next step is to integrate the voice so that festival can find it. A symbolic link is added from the voice directory of Festival’s English voices to the directory containing the new voice:

```
cd /usr/local/festival/lib/voices/english/
ln –s /data/hcu_hin_sms_diphone
```

Finally a distribution file has to be made so that the voice may be installed on other festival installations and can be used by anyone. One must add a “COPYING” file to the database directory and state the terms and conditions in which the voice should be used, distributed and modified.
The distribution file must be generated in the directory that contains festival and speech tools:

```
cd /usr/local/
```

```
tar zfvc Festvox_hcu_hin_sms_lpc.tar.gz \
festival/lib/voices/english/ hcu_hin_sms_diphone/Festvox/*.scm \
festival/lib/voices/english/ hcu_hin_sms_diphone/COPYING \
festival/lib/voices/english/hcu_hin_sms_diphone/group/smslpc.group
```

This is the final step and now the voice is ready to be distributed and used in various applications.

[A CD has been submitted along with this dissertation, which has the Hindi voice built on the Festival software “festvox_hcu_hin_sms_lpc.tar” and a copy of this dissertation. The Index file is well linked with the contents listed in the index.]

[S1]Doesn’t seem necessary

Chapter 5
Walkthrough

To build a Hindi Speech synthesis system on a diphone-based synthesizer, we need to build a Hindi Diphone database. The relevant changes, processes, steps, their explanations in detail and justifications for all the choices made in the process have been explained in the last chapter. In this chapter I will reiterate the steps of building a diphone database as stated in earlier chapters, in a chronological and crisp manner but without any details or descriptions. The aim of this chapter is to give a step-by-step development of the process of building a Hindi diphone database in such a manner that the reader can follow it up practically to build his/her own voice in Hindi or in a new language if relevant data is provided.

A walkthrough for US/UK English has been given in Section 8.10 Chapter 8, Diphone Databases in Festvox manual. This walkthrough has been written on similar guidelines though there are a number of additions.

To begin with, first, one must make a list of the tasks that are involved in building a database. This is important as it aids a smooth development because one knows what comes next. Most of these are listed in Festvox:

- Constructing basic template files
- Generating the prompts
- Recording utterances (nonsense words)
- Auto labeling the recorded utterances
- Hand labeling the auto labelled utterances with the help of EMU labeller (this step is not stated in Festvox as it is optional but required if one aims to get a better output)
- Generating the diphone index
- Testing the diphones and hand fixing if necessary
- Building diphone group files and distribution

Before we head on to building a diphone database, I believe we should start from the start and so I consider it relevant to state the processes involved in compiling Festival on a personal computer with RedHat Linux 6.2.

Installing Festival:

In order to compile Festival we need to have RedHat Linux 6.2 on a computer. (Though one can have other versions of Linux as well, but since the present system is built on RedHat 6.2, I will stick to it.) If Festival, Festvox, and Speech Tools have been successfully installed on the system (whichever version of Linux), then you can skip this and need not read this section.

To compile Festival, we need the following source packages:

festival-1.4.2-release.tar
festlex_CMU.tar
festlex_OALD.tar
festlex_POSLEX.tar
festopt_clunits.tar
festvox_don.tar
festvox_kallpc16k.tar
festvox_kedlpc16k.tar
festvox-1.2-beta.tar
speech_tools-1.2.2-release.tar

First we have to unzip all the above packages. We can do that by using the following command:

tar –xzvf <filename>

After unzipping all the files we will get three directories –

- Speech tools
- Festival
- Festvox.

First we will have to install the Speech Tools, then Festival and at last Festvox.

To compile Speech Tools, “cd” (i.e. Change directory) into the speech tools directory and run the following commands. This will configure speech tools.

cd speech tools
./configure
make

To compile festival, “cd” (Change directory) into the festival directory and run the following commands. This will configure festival.

cd festival
./configure

make

To compile Festvox, “cd” (Change directory) into the Festvox directory. Before Festvox is configured, one must have Edinburgh Speech Tools and Festival installations. Next, one has to edit the “config” file and set ESTDIR to point to the locally installed version of the Speech Tools and also set the path for Festival distribution, FESTDIR if necessary. This can be done as follows:

cd festvox
cd config
cat config-dist > config

Now edit the config file and specify the paths for speech tools anf festival installation. Finally we can install festvox:

cd ..
make

This should build the phone aligner and build the scripts.

Setting up the variables:

To begin with our work, first we need to set up the following variables to where the festival, Festvox and speech tools have been installed:

export ESTDIR=/usr/local/speech_tools
export LD_LIBRARY_PATH=/usr/local/speech_tools/lib
export FESTDIR=/usr/local/festival
export FESTVOXDIR=/usr/local/festvox

Making a new directory:

Create a new directory and “cd” into it. One needs approximately 500MB of space for it.

mkdir /data/hcu_hin_sms_diphone
cd /data/hcu_hin_sms_diphone

The directory is named with a particular convention with the name for institution, (here hcu for Hyderabad Central University), name for the language, (here hin for Hindi), and name for the speaker, (here sms for Sumedha).

Creating a basic Directory Structure:

Next step is to build a basic directory structure and copy the relevant skeleton files. By default the files are copied for languages US and UK English.

$FESTVOXDIR/src/diphones/setup_diphone hcu hin sms

Constructing basic template files:

For Hindi one has to make a lot of changes to provide for the necessary linguistic input. These alterations have been described at length in Chapter 4. However, for convenience I will again list the directories and the relevant changes in brief as they are elementary in building the diphone database.

The basic directory structure will have 19 sub directories – Bin, Cep, Dic, Etc, F0, Festival, Festvox, Group, Lab, Lar, Lpc, Mcep, Pm, Pm_lab, prompt-cep, prompt-lab, prompt-wav, Wav and Wrd. Amongst these, the Festvox directory is the one which contains language specific data and most of the files in this directory need linguistic input to create the list of diphones and build a database.

The details about the Festvox files and the changes have been explained earlier, this is just a brief summing up of the work.

hin_schema – Specify the list of phones, the invalid pairs if any and radio map or "bootstrap" these phones with the phones of an existing language.

hcu_hin_sms_phones – Define the phones of the language in terms of a feature matrix specifying the height, frontness, manner of articulation, place of articulation, etc. of the phones.

hcu_hin_sms_dur – In this file, one has to specify the average durations of all the phones of the language in question including the pause or silence.

hcu_hin_sms_int – This file generally does not need any input or alterations, but if the diphone database is being built in a female voice, one has to specify the average range of the speaker and Fixed Prosody - (200-220 for female voice, by default it is set at 110-130 for a male voice).

hcu_hin_sms_lex – One has to specify the letter-to-sound rules for the language in this file. If the graphemes do not have one-to-one mapping in a language, one has to make a lexicon.

hcu_hin_sms_token – If one wants to do some text analysis, tokens have to be specified in this file for numbers, abbreviations, etc. For building a diphone database this information is not important and may be excluded for the time being.

There are two more files, diphlist.scm and hcu_hin_sms_diphone.scm, which do not need any alterations and they work in tune with other files.

Generating the diphone list:

After the files in Festvox have been modified, our basic spadework is done and we are ready for the process of building a diphone database. From the information in Festvox, we can now create a list of utterances or prompts. These utterances are nonsense words with a nonsense carrier added before and/or after the diphones to produce a sequenced list for recording. The list can be created by the following command:

festival –b festvox/diphlist.scm festvox/hin_schema.scm '(diphone-gen-schema "hin" "etc/hindiph.list")'

The list is created in Etc directory as hindiph.list and will contain the possible phone-phone combinations in a language.

Synthesizing the prompts:

The synthesizer then synthesizes the prompts generated in the hidiph.list in an existing voice. These synthesized prompts are used for prompting the speaker while recording and for alignment. Prompts can be synthesized by the following:

festival –b festvox/diphlist.scm festvox/hin_schema.scm '(diphone-gen-waves "prompt-wav" "prompt-lab" "etc/hindiph.list")'

The voice for synthesis is by default set in hin_schema to voice_kal_diphone as:

```
(define (Diphone_Prompt_Setup)
(voice_kal_diphone)
(set! FP_F0 90)
(set! diph_do_db_boundaries nil)
```

Record the utterances:

While recording, the synthesized prompts are "spoken out" to the speaker for mimicking because this will keep the articulatory effects to minimum and all the diphones will be recorded in a sequence. The recording is also iterated to the speaker so that he/she can check if the recording is as desired.

Recording environment should be carefully set as explained earlier such that in case of re-recording some instances, the environment can be duplicated in future.

Recording can be done by:

bin/prompt_them etc/hindiph.list

This command plays the synthesized prompt, gives the recording time, and repeats them for the speaker to review it. To stop recording at any point one has to press Ctrl+C and to resume recording from a certain point, the prompt number can be specified as follows:

bin/prompt_them etc/hindiph.list 300

Labelling the recorded prompts:

The recorded prompts can be labelled by:

bin/make_labs prompt-wav/*.wav

This auto aligns, i.e. marks the boundaries of all the phones in the recorded utterances from which diphones will be segmented later.

One should always check the labelling and rectify if necessary the autolabelled prompts in EMU labeler.

emulabel etc/emu_lab

This will open a window, select FILE and OPEN from the top menu bar and place the other dialog box and click inside it and hit return. A list of all label files will be given. Double-click on each of these to see the labels, spectrograms and waveforms. One can move the pointer to readjust the phone boundaries, click on save and open another label file.

Making a diphone index*:*

After all the labels have been corrected, a diphone index has to be built. A diphone index lists all the diphones, their file id's, and their start, mid and end durations in the reverse order of the diphone list. To make a diphone index:

bin/make_diph_index etc/hindiph.list dic/smsdiph.est

The diphone index is made in the directory "Dic" as "smsdiph.est" (In this case, sms stands for the speaker's name)

Extracting pitchmarks:

Next step is to extract the pitchmarks from the recorded utterances to find the default pitch periods within the speaker's maximum and minimum range so that finally the unvoiced section is filled with the default pitchmarks. Extracting pitchmarks is important for the quality of synthesis and better the pitchmarks better will be the synthesis.

If the recordings have been done with an electroglottograph (EGG), pitchmarks have to be extracted from the EGG signal as follows:

bin/make_pm lar/*.lar

However, if the utterances have not been recorded by EGG, pitchmarks can be extracted from the waveform as follows:

bin/make_pm_wave wav/*.wav

The extracted pitchmarks can be predicted to the nearest peak in waveform:

bin/make_pm_fix pm/*.pm

Power normalization:

In a set of diphones, there is often a mismatch or a general power difference between files. This difference has to be neutralized and can be done by finding the mean power for each vowel in each file and then calculating a factor with respect to overall mean vowel power. This can be calculated by:

bin/find-powerfactors lab/*.lab

The power factors calculated by this are saved in etc/powerfacts

Building LPC coefficients:

This is the last step in building a diphone database that will be ready for testing. After the power factors have been calculated, build the pitch-synchronous LPC (Linear Prediction Coding) coefficients and the residual files:

bin/make_lpc wav/*.wav

LPC finds out the pitch periods for the diphone database. LPC coefficients are saved in two files (two files for each diphone) in the directory LPC with the extension .lpc for the LPC coefficients and .res for the LPC residual.

Testing the diphones:

Now the database is ready for its initial tests. It is a good idea to test all the diphones before wrapping them up into group files and integrating them in the system. The basic functionality of voice and the diphones are tested with the SayPhones command.

festival festvox/hcu_hin_sms_diphone.scm '(voice_hcu_hin_sms_diphone)'
festival > (SayPhones '(pau n a m a s t e))

It is a good idea to listen to all the diphones to shortlist the erroneous ones if any. One idea is to listen to the entire list of nonsense words in the new voice at the festival prompt, as this would ensure that all the diphones have been tested.

If there are some erroneous diphones one can check them with EMU labels by giving the diphone id. e.g. hin_0001:

emulabel etc/emu_lab hin_0001

Rectify the labels and check the diphone indexes for proper start, mid and end values. After correcting all the faulty labels run the entire process from make_diph_index, make_pm_wave, find_powerfacts to make_lpc.

However if the problem still persists, the only option is to re-record the problematic utterances. This should be done only if the fault is in a large number of diphones. While re-recording one has to take care that the environment is exactly the same and so is the pitch and volume so that the difference is minimal. After re-recording, the new recordings have to be labelled and the whole process has to be run from make_diph_index for all the recordings, previous and new.

Making group files:

After all the diphones have been tested, wrong ones have been rectified and the diphone database is absolutely ready with no more alterations due, next step is to group this diphone database. Making group files omits the unnecessary portion and includes only the

subparts of the spoken words, which contain the diphones. It is more like wrapping up the whole diphone database into a voice.

To do this first run the following command to make the group files:

festival festvox/hcu_hin_sms_diphone.scm '(voice_hcu_hin_sms_diphone)'

…

festival (us_make_group_file "group/smslpc.group" nil)

The "us" here stands for the UniSyn, unit concatenation sub-system in Festival.

After this, edit the file Festvox/hcu_hin_sms_diphone.scm and comment out the line (around line 81)

Comment the line

(set! hcu_hin_sms_db_name (hin_diphone_init hcu_hin_sms_lpc_sep))

and uncomment the line (around line 84)

(set! hcu_hin_sms_db_name (hin_diphone_init Hcu_hin_sms_lpc_group))

Integrating the new voice in Festival:

The next stage is to integrate this new voice in the festival so that festival can find it automatically. To do this, add a symbolic link from the voice directory of Festival's English voices to the directory containing the new voice. For this, first "cd" (Change directory) to Festival's voice directory -

cd /usr/local/festival/lib/voices/English

- and add a symbolic link back to where the new voice is built:

ln –s /data/hcu_hin_sms_diphone

This new voice will be now available to anyone running this Festival version.

Generating a distribution file:

The last step is to make distribution files in case you want others to use your voice on their Festival installations. Before making distribution files, make a file COPYING in the directory that contains your voice and state the terms and conditions in which people may use, distribute and modify the voice.

Now generate the distribution tar file in the directory where the festival, festvox and speech tools have been installed (In this case /usr/local)

cd /usr/local/
**tar zxvf Festvox_hcu_hin_sms_lpc.tar.gz **
**festival/lib/voices/English/hcu_hin_sms_diphone/Festvox/*.scm **
**festival/lib/voices/English/hcu_hin_sms_diphone/COPYING **
festival/lib/voices/English/hcu_hin_sms_diphone/group/smslpc.group

Now the new voice and diphone database is ready to be distributed and can be written on a CD as well and can be used by any festival installation.

Chapter 6
Problems and Conclusion

In this chapter I will be discussing the problems I have faced during the course of my work and what developments can be made further in future.

This chapter concludes my work and suggests further improvements. You will be inquisitive why I did not implement these improvements and suggestions myself. Well, I can just say that only when one does something will one realize what is missing and the same holds in my case. After I did this work, I could manage to write this chapter and could express what I feel I could or should have done for a better output. Even though this work is computational in nature but I stress my point here that you look at it from a completely non-computer science outlook because that is the way it has been done (being a non-computer science student myself). For a computer science student, this chapter may seem not-so-important-to-discuss kinds, but it aims at explaining the challenges, problems (as experienced) and suggestions to those who do not have any idea about the nitty-gritty details of speech synthesis. In developing a diphone database for Hindi, my aim is not to modify codes (though I had to do at some places) or write new ones, but to highlight the fact that Computational Linguistics is a marriage between computers + linguistics and one ought to absorb the interface. Knowing one and assuming the other wont do any good. In speech synthesis, a lot depends on what linguistic input is provided to the computational system. I am not implying here that my work is complete and without flaws but the fact is that many speech synthesis systems I have come across, lack the required linguistic input and the result is that even though computationally their systems are far advanced, linguistically they are very starved. For instance, including the nasalized vowels /aM/, /AM/, etc. is very important for a continuous speech because nasalization is a co-articulatory feature (both /a/ and /M/ sounds are pronounced together) and cannot be concatenated /a/ + /M/. Similarly, choosing the invalid pairs need a lot of linguistic input like what kinds of geminates are allowed in Hindi, which phone pairs cannot occur at intra-word or inter-word boundaries, and so on.

One more interesting point is the question - what are “half-consonants” and how do we account them in Hindi diphone database. I have come across this question many times in last one year. In Hindi, there is nothing called “half-consonants” from the point of view of linguistics. Half-consonants is just an orthographic representation i.e. alphabetic representation of sounds. Hindi is a syllable timed language and has a syllabic alphabetic system. In other words, Alphabets in Hindi are actually syllables. All consonants have a sound /a/ pronounced with them. Thus, we have alphabet “क” - “k” pronounced as /ka/.

The actual sound is /k/ but cannot be pronounced without a vowel or other consonant. Now look at the two words given below –

/kahanA/ - Here orthographically we have complete alphabet “k” - “क“. But in pronunciation it is /k/ followed by /a/.
/kyA/ - Here the orthographic representation is half-consonant “k” - “क्” but pronunciation wise it is only the sound /k/.

It depends on the person what choice he/she will make. Spoken language is represented through sounds and not orthography but most people believe that language is the alphabetic system. Orthography is an arbitrary representation of sounds and may or may not follow a pattern or rule. For example, English alphabet system does not represent the sounds of the language and has one-to-many mapping between graphemes and phonemes. There is no rule of mapping one alphabet to many sounds. It is an arbitrary system. Hindi however, is not so arbitrary a system. The alphabets represent sounds sans the intrinsic vowel spoken to utter the consonant. A complete consonant (alphabetically), is consonant+/a/ in pronunciation and half-consonant (alphabetically), is only consonant as pronounced in speech. The so-called “half consonants” are not pronounced individually. Actually in building a database, what we should actually include are these so-called “half-consonants”. To make “full-consonants” out of these we just need to concatenate “half-consonant” with /a/. The actual phones of a language are these “half-consonants” and the “full consonants” are actually graphemes. We need to reach the platform where we can distinguish between the two. What we pronounce are actually sounds but we perceive them as written symbols of our language. One has to realize this difference between perception and reality. We need to understand the speech system of our language and languages in general before we head on to developing speech synthesis software. Unless we know our own speech system very well how can we create an artificial system and expect it to be as good as real. To develop a diphone database, one certainly needs a fair idea about the sounds and their graphic representations. This linguistic knowledge reflects on various aspects - on the choice of phones for the diphone database, defining those phones, short-listing the valid and invalid pairs (phone-phone combinations), recording, labeling, giving intonation and duration and making letter-to-sound rules or lexicon.

The choice of phones in a language should reflect on the allophonic variation, co-articulation effects, and phonemic status apart from the graphemic representation of sounds. In Festival Speech Synthesis System, this information comes handy in feeding the input in “hin_schema”. (Refer chapter 4 for details). If one is sure about the sound system, defining them in terms of a feature matrix is not difficult (in “hcu_hin_sms_phones.scm” – refer chapter 4 for details). Similarly, information about duration and intonation is a part of

phonetic and phonological knowledge of the language that helps in providing information for the “hcu_hin_sms_dur.scm” and “hcu_hin_sms_int.scm” files in Festvox (chapter 4 for details). How efficiently are the prompts recorded largely depends on how much aware a person is about “speech”. Though a native speaker’s competence of the language is very essential but added to it should be the linguistic knowledge about one’s language. Recordings should be clear, stable and in a monotone. Labelling the recoded words requires substantial knowledge of the physical representation of speech, i.e. sound waves, spectrograms etc. To segment the sounds or mark phone boundaries, one should be able to read the waveforms, and map the changing waves with the sounds. This is especially important when marking the vowel-vowel combinations, as it is generally difficult to find out where one ends and the other starts. (See chapter 4). The input for the letter-to-sound rules and/or lexicon (as in “hcu_hin_sms_lex.scm” – refer chapter 4 for details) depends on the kind of mapping between graphemes and phonemes of a language and how much a person can map reality to perception.

Selecting phonemes in a language may seem very trivial but if done the right way it needs a deep understanding of the sound system of one’s language. Once the list of phones is ready, and has been properly defined in Festvox, next step is to make a diphone list. If one intends to include all the phone-phone combinations, then creating a list is not a problem at all but suppose if one decides to include only the possible combinations in the specified language and exclude the “unwanted” or invalid pairs, to make a sorted list is a challenge at hand. To shortlist the valid pairs is the right approach as the diphone database should be of reasonable and manageable size and having invalid combinations will unnecessary increase the size of the database. For this, one must first make a list of invalid pairs in the language, which should be a careful decision considering the word internal and external boundaries. Then one has to modify the code “hin_schema”, such that while making the list of valid phones, the list of invalid ones is excluded. This has been done in the following way:

For vowel-vowel pairs:

```
(define (list-vvs)
(apply
append
(mapcar
(lambda (v1)
(mapcar
(lambda (v2)
(if (not(member v2 (cdr (assoc v1 invalid_vv ))))
(list
(list (string-append v1 "-" v2))
```

```
(append (car vv-carrier) (list v1 v2) (car (cdr vv-carrier))))))
vowels))
vowels)))
```

For consonant-consonant pairs:

```
(define (list-ccs)
(apply
append
(mapcar
(lambda (c1)
(mapcar
(lambda (c2)
(if (not(member c2 (cdr (assoc c1 invalid_cc ))))
(list
(list (string-append c1 "-" c2))
(append (car cc-carrier) (list c1 c2) (car (cdr cc-carrier))))))
consonants))
consonants)))
```

While making the list, the code will check for every combination if it occurs in the invalid list. Those in the invalid list will be skipped and the next pair will be checked. In fact the code sums up as, *list those that do not occur in the invalid list, if they do, skip it*. I have made invalid pairs list only for vowel-vowel and consonant-consonant combinations because most of the restrictions occur in such instances only. Vowels and consonants can occur in most of the word internal and (if not) in word external situations. Also Hindi mostly has a C-V-C syllabic pattern, so I did not want to take a risk excluding any such pairs that may be used even in remote possibilities.

However, this process of excluding the invalid pairs is not so easy and simple as it seems. “hin_schema” and “diphlist.scm” together create the list of diphones, where “hin_schema” provides the language input and “diphlist.scm” provides the format. Each diphone is listed in one line and is given a unique file id that is used in other processes in later stages of building a diphone database. The file id is provided by diphlist.scm that is nothing but a loop that creates numbers is serial order for every match in the “hin_schema” lists. For instance, the list goes like this:

```
( hin_0001 "pau a t aa pau" ("pau-a") )
( hin_0002 "pau A t aa pau" ("pau-A") )
( hin_0003 "pau i t aa pau" ("pau-i") )
( hin_0004 "pau I t aa pau" ("pau-I") )
( hin_0005 "pau e t aa pau" ("pau-e") )
( hin_0006 "pau E t aa pau" ("pau-E") )
```

Therefore, even if invalid pairs match, diphlist.scm gives it a file id and instead of a diphone, lists “nil”. A such, even though we exclude the invalid pairs we cannot exclude their “listing” in the list. If the “nil” items in the list are not removed, it creates problems in later stages, whereby default files are created for them while synthesizing the prompts, recording etc., which brings about lots of errors. One important observation is that in “hin_schema”, nowhere is it specified that list “nil” if invalid, the system by default gives “nil” value.

(hin_1619 "pau t aa R n aa t aa pau" ("R-n"))
(hin_1620 "pau t aa R m aa t aa pau" ("R-m"))
(hin_1621 "nil")
(hin_1622 "nil")
(hin_1623 "pau t aa R N aa t aa pau" ("R-N"))
(hin_1624 "pau t aa R y aa t aa pau" ("R-y"))

To overcome this problem I had two options, one to change the code in “diphlist.scm” in such a way that every time a category is matched in the invalid list in “hin_schema”, it should skip the numbering. But “hin_schema” and “diphlist.scm” are not so explicitly linked that I could make changes. Instead I chose an easier and an amateur option. I deleted all the lines with “nil” in “hindiph.list”. As a result, I could not get a sequential list, I had “missing” numbers, something like:

(hin_0997 "pau t aa t O u t aa pau" ("O-u"))
(hin_0998 "pau t aa t O U t aa pau" ("O-U"))
(hin_0999 "pau t aa t O o t aa pau" ("O-o"))
(hin_1022 "pau t aa t aM AM t aa pau" ("aM-AM"))
(hin_1023 "pau t aa t aM iM t aa pau" ("aM-iM"))
(hin_1024 "pau t aa t aM IM t aa pau" ("aM-IM"))

Earlier I thought this will create problems in later stages but it did not. The file id’s in “hindiph.list” are important for later stages, so if any changes have been made in the list itself, it will not be a problem. The other processes are linked to these file id’s and whether these file id’s are in a sequence or not, is not an issue. Thus, a simple deleting of “nil’s” made this problem simple even though it is not the best option, I agree, but it solved the purpose.

Recording Problems

The prompts in the diphone list are synthesized in an existing voice by the synthesizer so that they can be played for mimicking during recording and provide for the alignment in labeling. Recording the diphones is important but recording them “properly” is more important. By properly I mean, with the minimum possible errors, in the best possible

environment. The care required during recording has been explained in detail in the Chapter 4. The problems that occur while recording can be many and are mostly due to speaker's mispronunciations if the speaker is not maintaining a consistent monotone, noise in the environment, unstable distance of the microphone and so on. In fact noise is one the most common problem encountered in recording. The problems and/or errors due to speaker's idiosyncrasies are very common and have mostly been discussed in earlier chapters. In the present work, I faced some not-so-common kind of problems related to noise; hence, in this section, I will concentrate on those problems that occurred due to noise.

For my own explanatory adequacy, I will divide noise into two types – we can call them "external" and "internal" noise. By "external" noise I mean the noise in the environment or around the place of recording. For instance, if someone is recording in the early hours of morning in his/her own room on a PC, there can be noise of birds, traffic outside, people moving in and out. This is when the room is not a sound proof room and the microphone system is sensitive enough to capture the disturbances. This problem largely depends on how well is the recording environment set up and can be taken care of by recording in privacy for instance, I had to record at midnight to avoid the chirping of birds etc. However, what I am concerned is with the internal noise. By "internal" I mean that noise that is caused because of some internal reasons to the system. It may be due to distorted signals, worn out recording equipment or can be because of any other reason. *Noise* is exactly what you think it is. All transistors create a faint background hissing noise, and since transistors are used everywhere in sound recording and reproduction, the hissing adds up to a noise you can actually hear, especially if you turn your volume up when there is no music or speech being reproduced. Noise is a term that covers many different things. There are many different types of noise - Some of the most common noise types are:

- ***Electrical Noise*** - introduced by mains power supplies in your equipment and takes the form of a low hum based on the frequency of AC (alternating current) power supply.
- ***DC Bias*** – 0Hz audio introduced by DC (direct current) power supplies. This is usually inaudible, but can cause a sharp pop at the beginning or end of a recording.

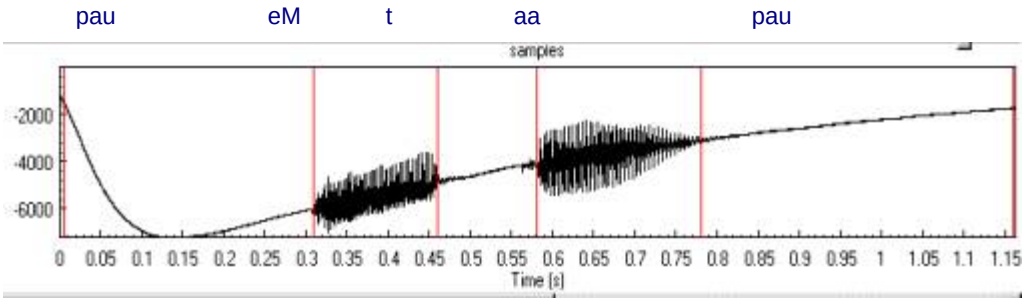
- ***Tape Hiss*** – caused by dirt on tapes or just low grade tape. It sounds like hiss or *white noise* – a random collection of different frequencies at different amplitudes.
- ***Rumble*** – caused by the mechanics of your tape deck or recorder.
- ***Warping*** – caused by old records
- ***Clicks and Pops*** – caused by dirty or scratched records.

In the present system, I have recorded the diphones on a personal computer with a Telex head mountable microphone. So there is no question of distorted tapes. However the first two types of noise explained above are a possibility in a system like this and can cause problems in recording.

I have a reason to explain all this 'noisy business' in so much detail and the reason is that after recording all the diphones (about 2000), which sounded absolutely fine, I faced a "unique" problem when I saw the waveforms in EMU. I am calling this problem "unique" because it took me a long time to find out the reason behind the problem. I consulted many people for this and eventually got an answer from some friends in Edinburgh University who are working on Digital Signal Processing.

First, I will explain the problem. In any waveform, there is a straight x-axis line, which represents the time and the y-axis line, which represents the amplitude. General noise, or say "external" noise, shows as small amplitude or as scattered frequency along the x-axis. An example of "external" noise is shown below:

My problem was very different. The x-axis or the time scale is supposed to be a straight line but in my recordings the line was a distorted line with a large dip as seen below:



However, the recordings could be heard absolutely fine, except that the waveforms did not look like sound waves.

This seemed like a ***DC Bias***, a 0Hz audio introduced by DC (direct current) power supplies. This is usually inaudible, but can cause a sharp pop at the beginning or end of a recording. But the question was how did it happen?

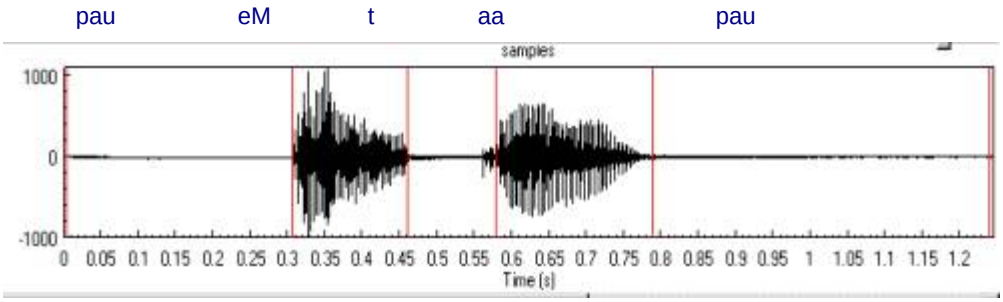
The most possible and likely explanation is that the room where the recordings have been done had earthing but probably there was a leak in the direct current, due to which, charge developed in the mic. This caused a low frequency noise, though not audible, to effect the recordings. As a result the recordings got an impulse, which has a general shape of lightening pulse. This was most probably due to leakage in power from earthing connection. Well, if the recorded wav files are filtered from a high pass filter, it removed all the low frequencies and gave a straight x-axis line.

The following script has to be run through the entire wave files to get the filtered waveforms:

```
for i in wav/*.wav
do
/usr/local/speech_tools/bin/sigfilter -hpfilter 1000 -forder 51 $i -o out
done
```

In the above script, the command `sigfilter` filters the waves through a high pass filter – `hpfilter 1000` for all files in the directory `wav` and the result is saved in the directory `out`.

An example of a filtered wave is given below:

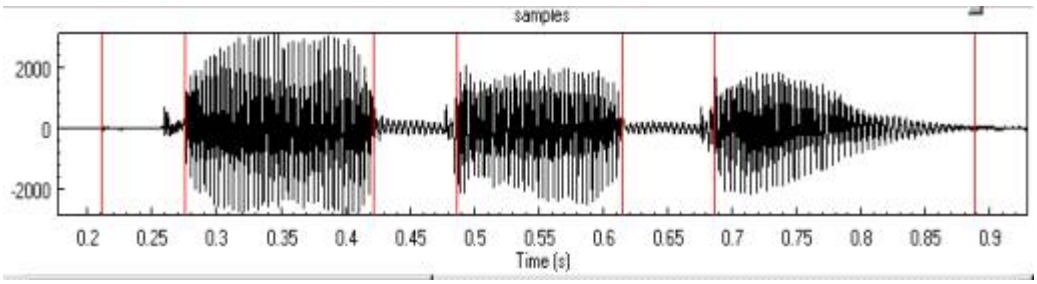


By the time I had reached this stage of filtering the waves, I had already re-recorded all the 2000 utterances. But still I processed the filtered waves to see what the synthesized voice will sound like. The filtered recordings as it is sounded a little mechanical because while filtering, with low frequency disturbance, the relevant low frequencies of recorded sound were also deleted. As such the synthesized filtered voice sounded very high pitched and mechanical. It was not of acceptable quality and had to be discarded. Even though I had to redo my entire work from scratch (that means changing the computer system and place of work and installing the operating system and software again), I learnt a lot of new things about sound and sound waves. This problem is very peculiar and most of the people may not experience it in their course of work but I consider it important to share my experiences with the readers because it is not only a problem but also a deeper look in the sound system.

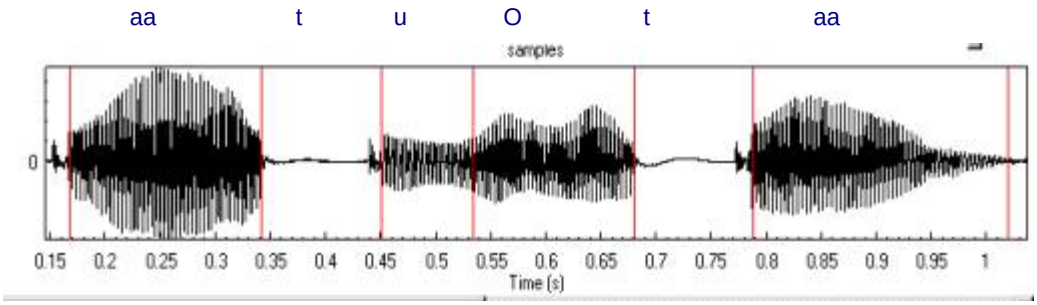
Recording errors can be any and/or many. But one must make sure that the recording has been done properly with as few errors as possible because a good recording leads to a good quality of synthesized speech.

The recorded files then have to be labeled to further the synthesis process and the labelling must be checked and corrected by hand labeling them. One has to have the intricate knowledge not only of the sound system i.e. phonetics and phonology but should also have a fair idea of the experimental part of it. From experimental phonology, I mean that part of phonetics and phonology, which deals with physical representation of sounds in the form of waveforms, spectrums, spectrograms, etc.

One should be able to look at a waveform or a spectrogram and tell the basic feature of the sound, whether it is a vowel, consonant, sonorant or semi-vowel, voiced or unvoiced consonant, stop or fricative and so on. If one knows even this much about waves, it becomes very easy and efficient to hand label the diphones because then one knows exactly where to draw the line between two phones. Look at the example below:

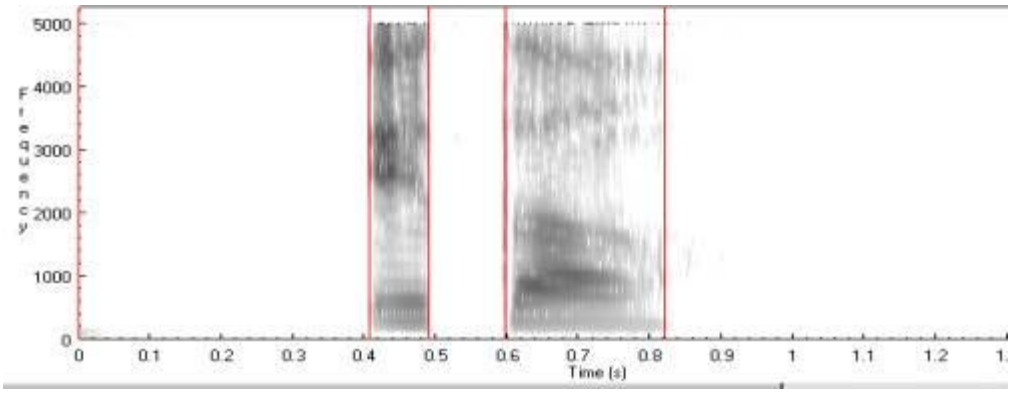


In the above diagram we can see the changing waves, we may not tell exactly what sound is it from the wave but we can say for sure that it is a consonant, a voiced stop and may be a /b/, /g/ or /d/. Now look at the waveform below:



In this picture there are two consecutive vowels. Even though we do not know which two vowels they are, but we can make out where one ends and the other starts. However if one

knows how to read a spectrogram, even the type of vowel and its other details like height, frontness, roundedness can be decided, though this information may not be required to that extent for hand labeling. Look at the picture below:



In this spectrogram, we can make out from the height of the formants that it is a high, front, unrounded vowel and should be /i/ or /I/ and the second one is /A/.

Labelling all diphones is not easy. Some cases pose problems because it is difficult to decide where to mark the boundary of phones especially in cases where the phones do not have clear waveforms as in consonants or when the waveforms merge into one another as in vowels. In this part, I will be discussing the basic challenges that have to be faced in the process of hand labeling the diphones.

Hand labeling is to mark the phone boundaries in a diphone distinctly. But when it is difficult to define the boundaries, one has to listen to the recording again and again in parts, which is also not a sure shot. All one is left with is to make guesses or own assumptions to find the best possible way to mark the boundaries. Below I have discussed some cases where I felt I had no other way out but to make my own decisions and have also explained my reasons to do the same.

Problems occur in silence-consonant pair, especially when the consonant is a stop because it is difficult to find out where the consonant starts as all we can see is a short burst towards the end of the phone. Next problem comes in a consonant-vowel-consonant combination, more so when the consonant is a sonorant or glide as they also have waveforms like vowels and it becomes difficult to demarcate them, e.g. in r-a-r, l-A-l, y-A-y etc. Problems also occur in vowel-vowel pairs because at times it is a challenge to decide when one starts and other ends. It is not so difficult in different vowels u-a but problem occurs in like vowels (vowels with minimal difference) like u-U etc.

One of the most problematic cases is to mark a geminate separately into individual phones.

Geminates, two same phones consecutively are very common in Hindi for example: /pattA/ “leaf”, /khaDDA/ “ditch”. Some problem also occurs when we get stop-aspirate pair, i.e. when a stop and its aspirate come together, as in, /buDDhA/ “old man”, here /D/ and /Dh/ come together. Similarly demarcating stops from stops, sonorants from sonorants, sonorants from glides, glides from glides, etc. in a diphone are a challenge at hand.

If one looks at these “delicate” cases, one conclusion can be made – that it is a problem to mark phone boundaries in cases where the phones are very alike in their waveform representations (whether they have or do not have a well formed wave), i.e. when they look alike in a graphical representation. The best bet then is to read the spectrogram (given just above the waveform in EMU-look at the appendix) carefully. Knowledge of reading spectrograms is not elementary but helps in making decisions based on facts rather than wild guesses. For instance, just by looking at the spectrogram one can find out whether it is a voiced or voiceless stop, aspirate or sibilant, whether a particular sound is a sonorant or vowel or whether a vowel is high or low, rounded or unrounded and so on.

It is difficult to explain all this at length here with examples as I don’t want to deviate from my main discussion i.e. identifying like-phones and hand label them but for those who have no idea about spectrograms, I will briefly present an overview of what points should one look at in a spectrogram if one should look at all.

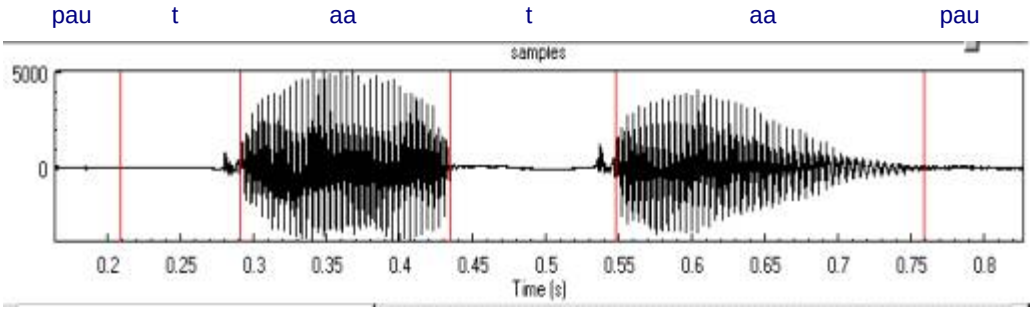
In a spectrogram, if the formants are well formed the sound is a vowel and the placement of the formant can tell the type of vowel, its height, roundedness etc. If the frequencies are formed but does not have well defined formants then the sound is most probably a sonorant or a glide and if there are no frequencies, the sound is a consonant. If there is one vertical line showing energy concentration, then the sound is a stop, if the line has small vertical lines showing scattered energy, it is an aspirate stop and if the energy is scattered as small vertical lines all over the high frequency zone, the sound is a sibilant. The voicing can be seen as small frequencies at the bottom of the spectrogram in a consonant. This is a brief overview of basic characteristics that can be observed in a spectrogram at one glance, however, there is a much deeper study that can be done but is not required for this purpose. If one can identify this much, it is enough to make decisions to mark boundaries.

Now I will explain the problem and suggest a solution for hand labeling certain phone-phone combinations:

- silence-consonant:

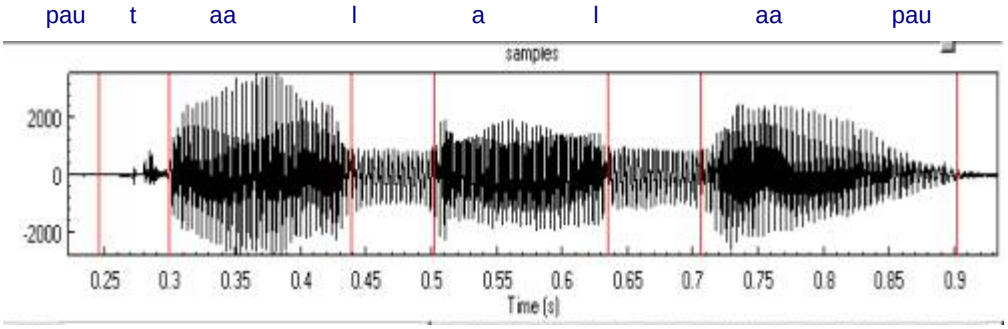
The following is a waveform for the pair pau-t. It is difficult to decide where the /t/ starts. To

mark it one should hear this part repeatedly and mark little silence before the burst. The real silence should be marked separately before the slience-burst of the stop /t/.

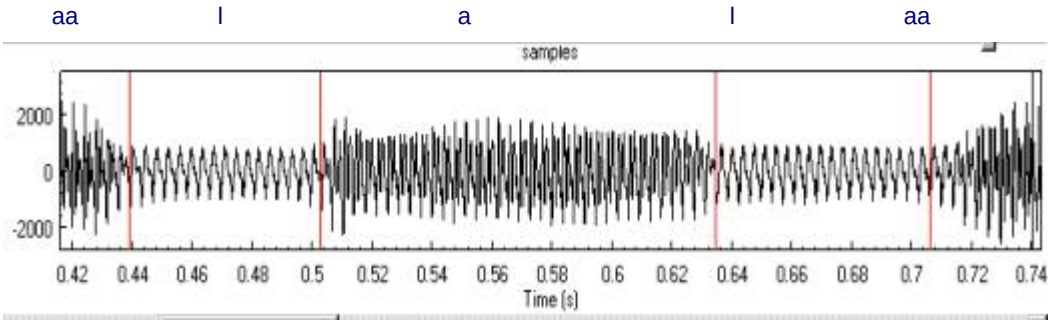


- cononant-vowel-consonant:

The following is a sonorant-vowel-sonorant combination /l-a-l/. Here the waves of /l/ merge with those of /a/ atleast at the crossover.

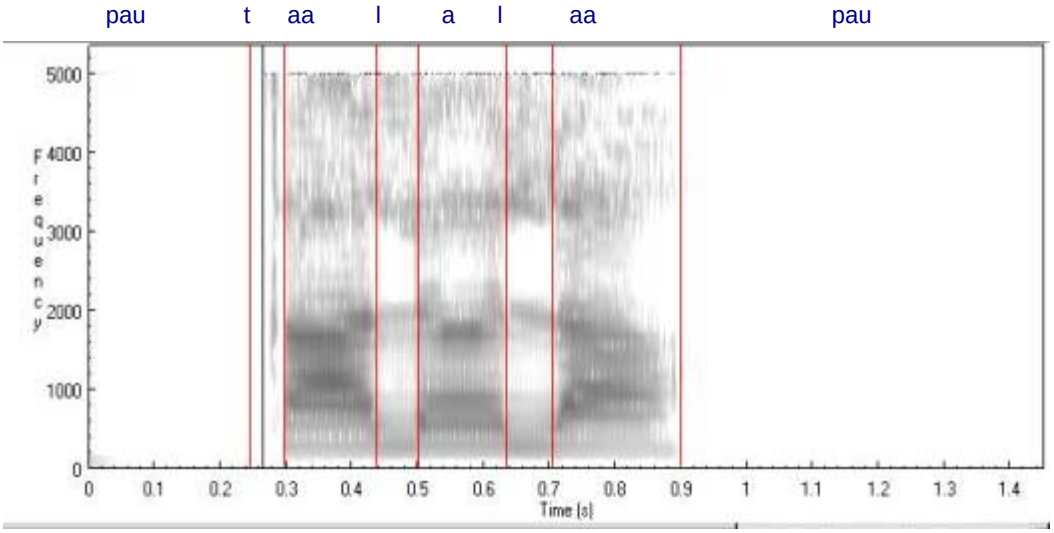


To mark the boundaries, one should enlarge the view (as seen below) and mark the place just before the well formed /a/ starts.



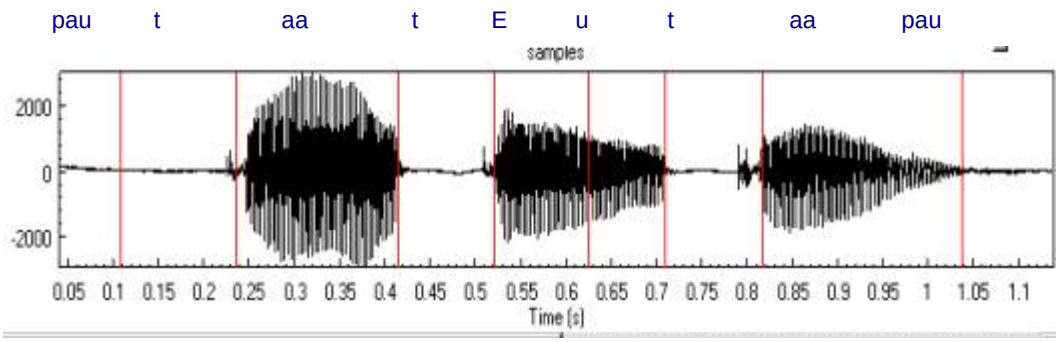
To make it simple, say there is an area called crossover where it is difficult to decide which wave is for the end of /l/ and which one is for the starting /a/. In this crossover, as can be seen in the second picture (the enlarged view), best is to mark in the middle of the

crossover area. It is however easier to look at the spectrogram, picture 3. One can easily mark out the well-formed formants from the frequencies. However, the markings should be a little before the format, you can say the at the crossover area.



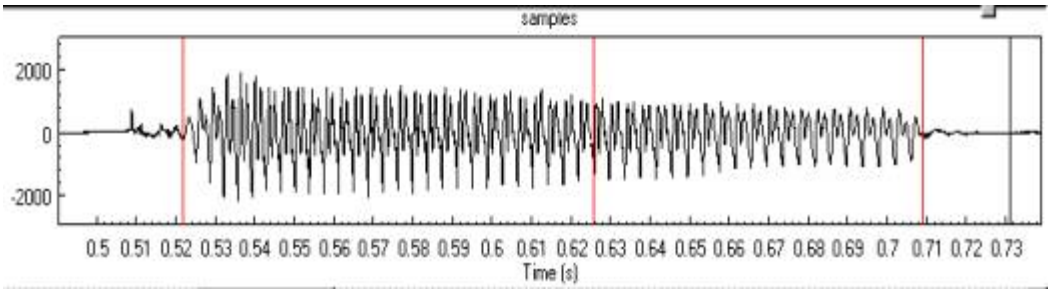
- vowel-vowel:

Following is a vowel-vowel pair /a-A/.

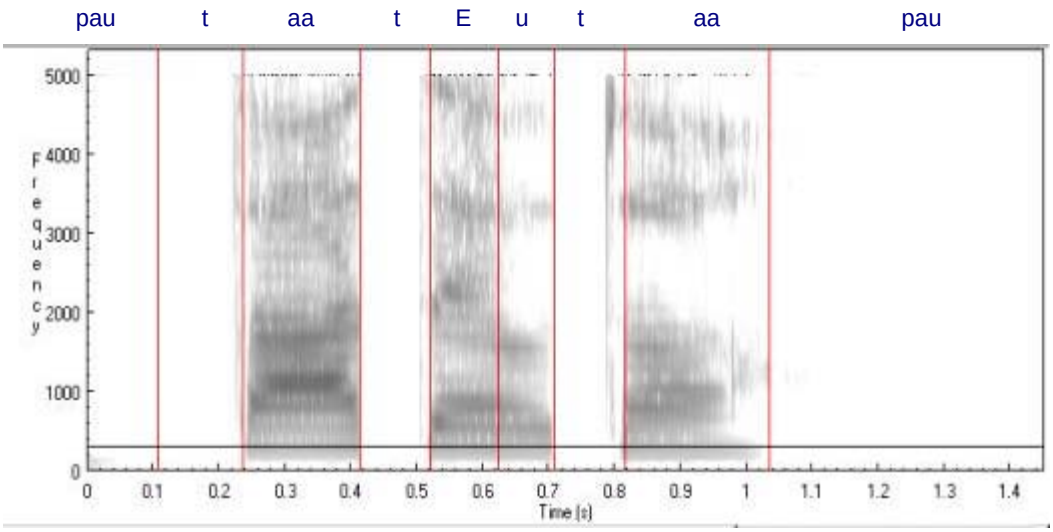


The waves are so smooth that is difficult to find the borderline. To label them, first enlarge the view as in the second diagram, and locate the crossover area and mark in the middle.

t E u t



For better results, also consult the spectrogram, picture 3 and observe where the formants change their position. Since both the sounds are similar, the variation in formants will be minimal.

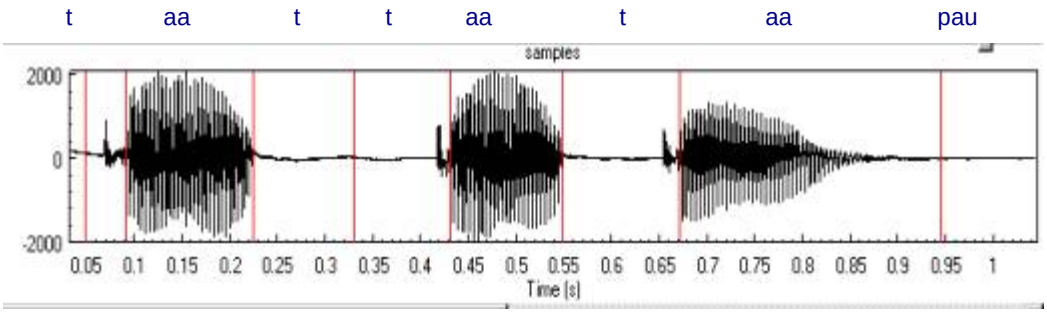


- consonant-consonant

This section explains various consonant-consonant pairs separately that pose a challenge in hand labeling.

➢ stop-stop (geminate)

Following is a stop-stop pair /t/-/t/. In the picture below we can only see a long plain area followed by a high-energy concentration.

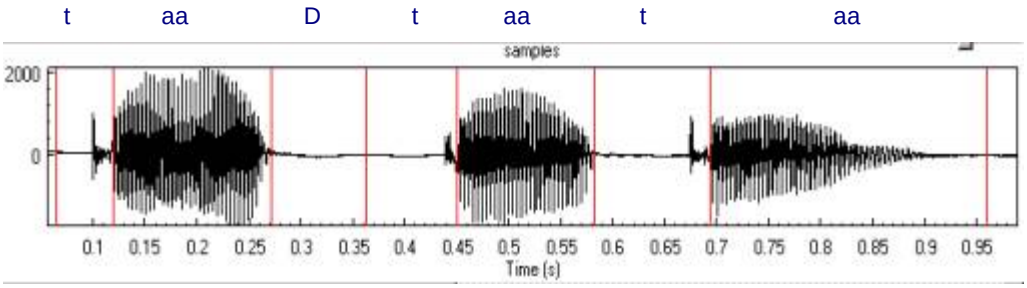


Now the most obvious choice would be to mark one phone till the first concentration and

second immediately afterwards at the second concentration. However, this would lead to problem, when segmenting a diphone. The plain area is also very important because it represents the building up of energy. To incorporate both in a diphone, one should mark the first phone in the silence area and the other at the end of the energy concentrations. Only then will the building up of energy and sudden release be captured in a geminate.

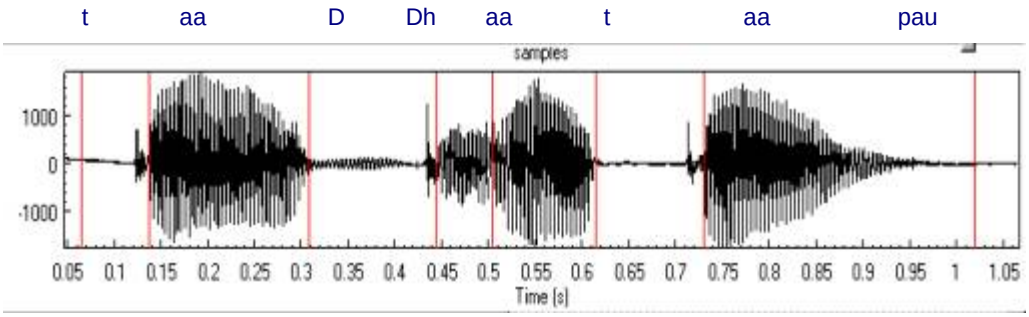
➢ stop-stop (non-geminate)

Following is a non-geminate stop hence the plosion may not be identical and in cases like /k/-/t/ may not be visible also. To mark the boundaries one must consistently mark in the middle if no energy concentration is visible. However if the two bursts are visible it is not very difficult to mark as in /D/-/t/

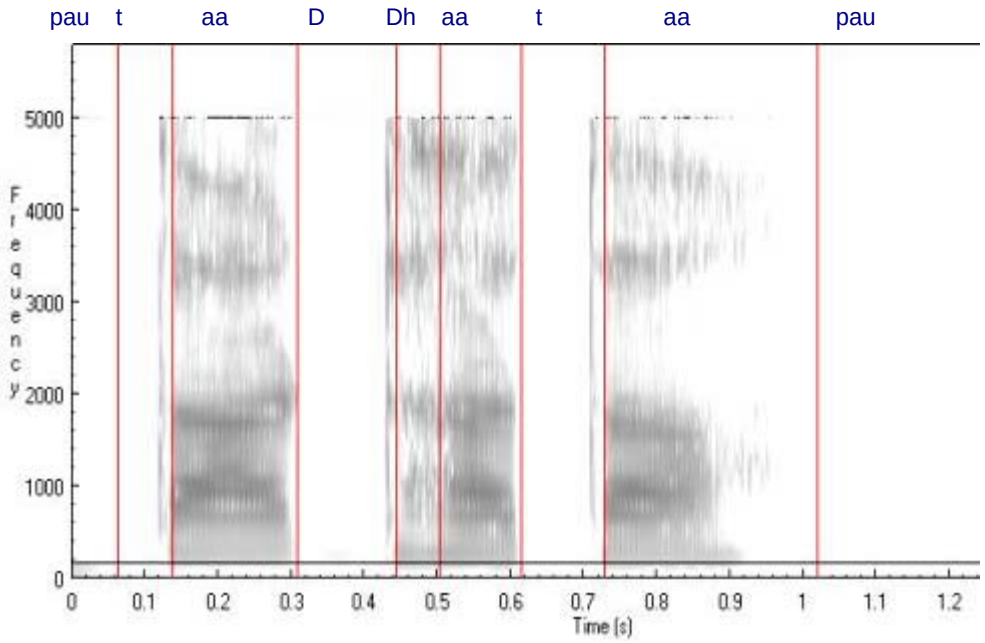


➢ stop-aspirate

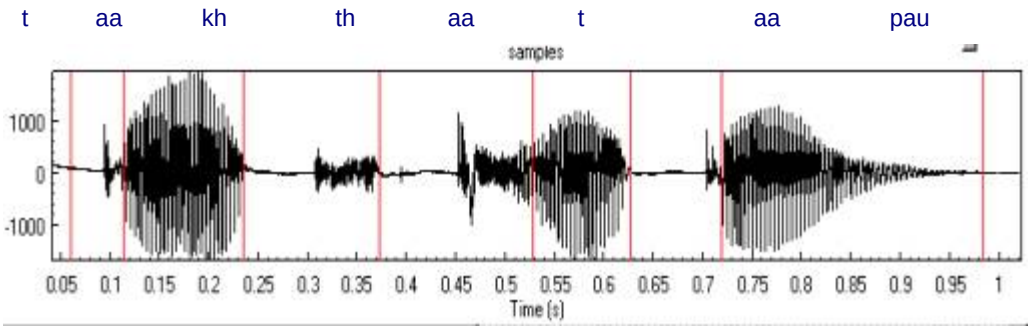
In a stop-aspirate pair, problem comes only when the stop and aspirate are alike except for the aspiration. For example, /t-th/, /d-dh/, /D-Dh/ etc.



It is so because in such cases the burst of the first stop in not complete and is generally extended into aspirated stop, as such the burst is not clearly visible. To mark it, one should carefully listen this area repeatedly and look at the spectrograms. The spectrogram may show the burst as a thin dark line followed by small scattered vertical lines. The first dark line will represent the first stop and the scattered lines will represent the aspiration. See the spectrogram below.



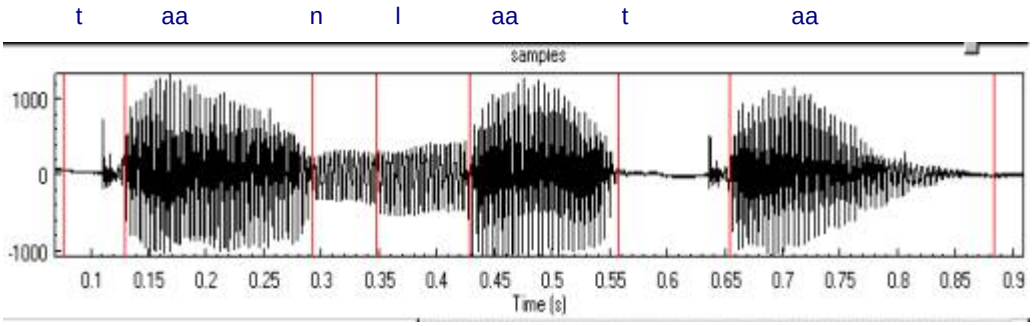
➢ aspirate-aspirate



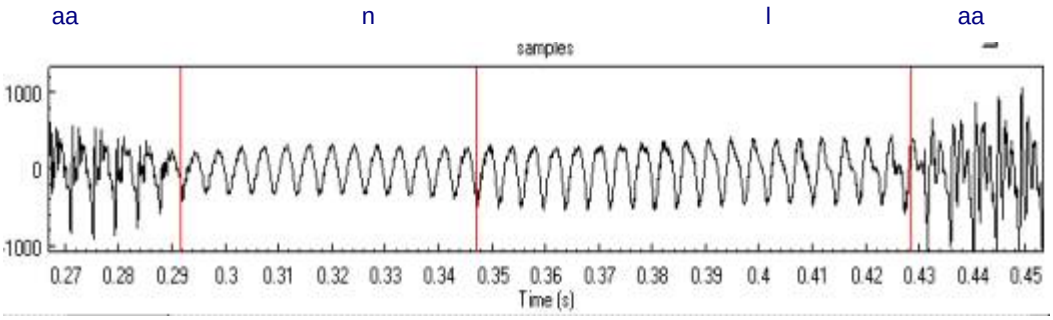
Aspirate-aspirate stops are comparatively easy because the energy concentration is fairly visible. But the energy release in the first aspirate is less than the energy release in the second aspirate. However, its best to mark the boundaries just after the energy release in both.

➢ sonorant-sonorant, glide-glide or sonorant-glide

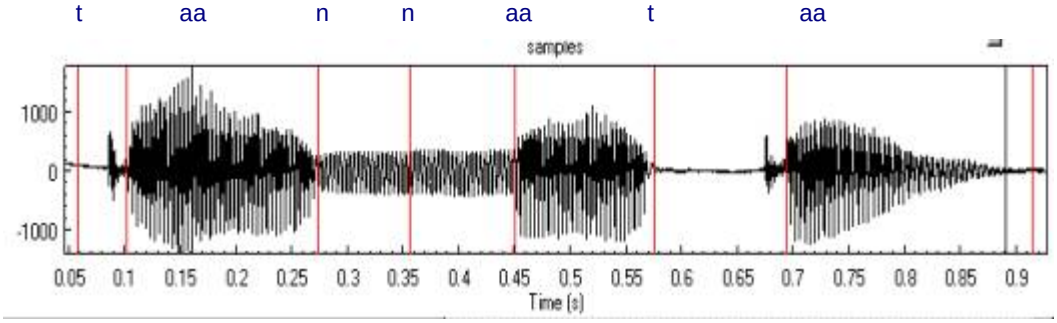
To demarcate two sonorants or glides or a sonorant-glide combination especially when they are geminates is another problem as it is difficult to find the borderline. Well, if we look at the first picture of two separate sonorants, we can see a crossover area, which is very short and not very well formed. One should mark in the middle of this crossover area.



Following is the enlarged view of the above diagram. We can clearly see the crossover area.



In the following picture one can see a geminate sonorant. To label the two separately, one should mark the way two stop geminates are marked, in he middle of the two phones, approximately half the distance between starting of the first phone to the end of the second phone.



Diphone Index Errors:

After a good recording and successful labeling next step is to create a diphone index. This step is simple as the command is stated in the Festvox manual but problems can crop in any time at any stage. So it is the case here. First time when I made the diphone index, what I got was a huge list with all zero values.

sh-sh hin_2435 0.0 0.0 0.0
sh-s hin_2433 0.0 0.0 0.0
sh-v hin_2432 0.0 0.0 0.0
sh-l hin_2431 0.0 0.0 0.0

The problem was very simple. In Linux, all files have permissions, to read, write and execute. If a file is not given executable permissions, it will not be able to execute. The files in Bin are executable files but suppose by default if the permissions are denied, the files wont perform the function. The solution is simple; one has to give permissions to the directory Bin before executing its commands. Permissions can be given by:

cd /data/hcu_hin_sms_diphone/bin
chmod 777*

This is not a major problem but can become one if someone is not aware of it.

Some problems may occur if the variables for Festival, Speech Tools and Festvox are not properly set. It is so because the commands “make_pm”, “make_pm_fix” and “make_lpc” all require Speech Tools and Festival libraries. If the variable is not set they will not be able to function with the commands specified in the code, like “ch_wave” and “sigfilter” etc. For example, one generally gets the following errors when finding the powerfactors.

bin/find_powerfactor lab/*.lab
hin_0001
sh:ch_wave :command not found
sh:ch_wave :command not found
hin_0002
sh:ch_wave :command not found
sh:ch_wave :command not found

If one gets any errors after using the commands while extracting pitchmarks, power normalizing and/or creating LPC coefficients, one must check if the following have been exported and the path is right according to one’s setting:

export $ESTDIR=/usr/local/speech_tools
export $LD_LIBRARY_PATH=/usr/local/speech_tools/lib
export $FESTDIR=/usr/local/festival
export $FESTVOXDIR=/usr/local/festvox

The path /usr/local/ has been specified for the present system. This path can be different for different systems depending on where Festival, Festvox and Speech Tools have been

installed. However, if the problem still persists, the only probability is that the system cannot locate the file for "ch_wave" and the only solution is to look for the file "find_powerfactors" file in the directory bin and specify the path for the command "ch_wave" on your computer as follows:

cd /data/hcu_hin_sms_diphone/bin
vi find_powerfactors

In find_powerfactors locate the following:

```
printf("echo START %s %s\n",fname,$3);
printf("ch_wave -start %f -end %f wav/%s.wav -otype raw -ostype
ascii\n",ltime,$1,fname);
```

The code for the command ch_wave is present in the bin directory under speech tools. The path where speech tools and its directories are installed has to be specified in the file find_powerfactors - /usr/local/speech_tools/bin/ch_wave (this is specific to the present work).

```
printf("echo START %s %s\n",fname,$3);
printf("/usr/local/speech_tools/bin/ch_wave -start %f -end %f wav/%s.wav -otype raw -
ostype ascii\n",ltime,$1,fname);
```

This solves the problem as the system can then locate where the command is.

After the synthesis process is over, one has to test the diphones using the SayPhones command. After a thorough checking and correcting the diphones if necessary, the voice is ready in the new language and can be integrated in the Festival system. Even if everything goes well in the synthesis process, problems can still persist. You must be surprised "*How?"* and "*Where?".* Well, so was I when I recorded a female voice, synthesized it carefully going through every process but got a male voice as an output. The reasons could be vaguely related to some parameters but what exactly was the problem was a big question. At last I put up my question in detail on the Festival Mailing List and finally I got an answer. I have explained the changes I have made for a female voice in Chapter 4. The answer lies in the file "hcu_hin_sms_int.scm" in Festvox. In this file, the speaker's pitch range has to be defined. The relevant changes that have to be made have been stated in Chapter 14 in Festvox manual:

However, in the present case, the start and end values of the speaker's pitch has been set by default at 110 – 130 for a male speaker. These values have to be changed for a female speaker to 200 – 220.

This will give a female voice for the command SayText. But SayPhones command uses Fixed Prosody for F0. This value is by default set to 100 in Festival. For a female voice,

this value has o be set to 200 or so in the file “hcu_hin_sms_int.scm”.

(set! FP_F0 200)

The problems I have discussed above are all related to the process of building a diphone database. However, one major problem area that I should have discussed first, but I did not on purpose, is the problems that occur with installing and compiling Festival, Festvox, Speech tools and EMU. Ideally this should have been the first point because unless one installs successfully, one can’t work on a diphone database. But I prefer to discuss it towards the end because questions like - *How to install EMU? What version of festival to install on which version of Linux? How to install Festival on Linux 7.2? How to install festival on Windows? What sound card to use?* - and so on are very often asked and well answered questions in the festival mailing list. As everyone faces some or the other installation problem at some stage, the best hope for answers is the mailing list. However, I will briefly present the problems I faced while compiling festival on my system. My system earlier had RedHat Linux 7.2 and even though I could successfully install Festival 4.2.1, it was not compatible with the sound card drivers. The problem was that the Festival speech was nearly 10 times faster than the normal speech and absolutely unintelligible. To work on such a system was not possible. After referring to the festival mailing list I realized that others have faced similar problems on this version of Linux basically because the version of Festival was older than the version of Linux and so the compatibility was not adequate. Eventually, I had to switch to Red Hat Linux 6.2, which was absolutely compatible with the software. Installing Festival, Festvox and Speech Tools was simple enough as explained in the README files, however, the main problem occurred while compiling EMU on the system. EMU was necessary for reviewing the diphones and hand labeling. To install EMU, the system should have TCL/TK 8.3.3.3 and RedHat Linux had problems installing them. After a lot of trial and error, I finally got the solution that my version of EMU was not correct. There are many versions of EMU available on the net and one has to be sure of the version in respect to the operating system and version of Festival. If Festival and Speech Tools have been successfully installed, the correct version of EMU can be installed with its requirements.

Conclusion

In this section, I intend to end my dissertation, which I assume is just a beginning step and bring to light some issues that will lead to another building block.

In the present work, we have observed the basic steps to build a diphone database for speech synthesis. To sum up the whole thing, till now we have selected phones, created a

list of nonsense words, recorded them, extracted diphones from it, given them a fixed pitch and level intonation and finally received an output of a synthesized monotone speech. The main aim of this exercise was to understand the basic system of speech synthesis and its linguistic requirements. However, this is not the end of speech synthesis, but I should say the beginning, because it is from here that we begin our journey in the world of synthetic speech generation, also called text-to-speech synthesis systems. In the last 200 odd pages we have seen everything one should know at the beginning of this journey, from the meaning of speech synthesis to a complete diphone database. Now I would say that our beginning ends here and we can step into the world of explorations and further research, where there is no dearth of new ideas to make things simpler and more efficient and still there is a lot more space for every development.

A complete text-to-speech synthesis system should be able to convert any given text into a high quality, intelligible and natural sounding synthesized speech. To reach this end, we can divide text-to-speech in two parts – text analysis and speech synthesis.

Text-analysis is a field that requires a lot of research in analyzing the syntactic and morphological structure of a given language such that it could be parsed by a computer. The computer should be able to “understand” any text in a given language, the sentence structure, the numbers in their context, the homophonous words, and so on. Without a text analysis, a text-to-speech system is incomplete.

Speech synthesis is the next step in text-to-speech, which converts a given parsed text into synthesized speech. This requires a deep knowledge of the speech system of a given language. There are various approaches for speech synthesis (this has been discussed earlier), however I will stick to concatenative speech synthesis, as this work is based on this approach. In concatenative speech synthesis, at the core is a speech database that “reads” the input text.

In the present work I have developed this speech database from diphones. There are other ways of developing this database that also reflects the quality of speech. This is one area that needs more research because more the efficiency of this database, better will be the quality of the synthesized output. One upcoming development in this aspect is the Unit Selection Database that includes different kinds of units of concatenation, i.e. syllables, diphones, triphones etc., and the synthesis system “chooses” the most suitable kind of unit according to the context and presence of that unit in the database. For instance, it works something like this – The synthesis system will first look for a syllable to map to the particular context. If a syllable is not available it will look for a diphone or triphone. It is left to the synthesis system to choose the best possible unit. However there are some

limitations for this system and it needs in-depth research.

In speech synthesis, the speech database is just the foundation on which the entire building is built. That is to say that once a database is ready, one needs to apply various prosodic rules, intonation, stress, accent etc to make the synthesized speech sound more natural. There is also some research in emotional synthesized speech nowadays. The motive is to bring about a synthesized speech that sounds like human speech. As such there is an entire new field open to research in prosodic analysis. Without prosody, the synthesized speech sounds very mechanical and monotonous (The present work is sans prosodic analysis and so it sounds monotonous. Well, I would stress one point here that I intended to first make a diphone database that could give me speech output for Hindi so that I could get a clear understanding of the process of speech synthesis. The field of prosody is yet untouched and I intend to do it in my future research projects if possible.)

If you go to “Goggle Search” on the internet and search for “speech synthesis”, you will come across innumerable sites presenting, explaining, demonstrating speech synthesis products. But give it a close look and you will realize that it still does not meet your expectations, especially if you are looking for speech synthesis in Hindi or other Indian languages. I agree a lot of work has been done, but the Indian languages are still starved of this research and need an inquisitive eye. As a research student I would say that every language is unique and needs to be dealt with uniquely.

The fact that we haven’t reached the end shows that we are still progressing. However, we have come to an end of the present work but this is still not the end. This work is just a beginning, a preliminary step necessary for everyone who intends to enter the world of speech synthesis. The journey begins from here. Research and exploration, I believe, is something that starts but never ends.

THINGS TO KNOW

Diphone

Diphones are units that begin in the middle of the stable state of a phone and end in the middle of the following one. They aid speech synthesis by capturing the transitions between the two phonemes and the co-articulatory effects.

Diphone database

It is a database of sound units called diphones (see Diphone). The database contains all possible phone-phone combinations in a language and captures all co-articulatory effects to generate continuous synthetic speech. The maximum number of diphones in a language is the square of the number of phones of that language but due to phonotactic constraints the number may vary from language to language.

Labelling

Labelling is a term very commonly used in Speech processing especially the concatenative approach. Labelling is basically marking phone boundaries. The recorded units are labeled, that is, the segment boundaries are explicitly marked and labeled with the specific symbol. The labeled segments can be of any length – phoneme, syllable, word and so one depending on the requirement.

Rule-based Synthesis

A rule-based synthesizer takes a speech corpus and performs speech analysis and gets a parametric form. This parametric form is then put through a rule finding and a rule matching stage. Finally a speech signal is produced that is synthesized into synthetic speech.

Concatenation-based Synthesis

Concatenation based synthesis seeks a database of segmented sound units with the corresponding segmental information like duration, pitch, stress etc. These segmented units are then given a parametric form and temporal value. To produce a synthesized utterance these segmented units are concatenated as per the mapping of graphemes of the language to the phonemes. Prosody matching has to be taken care of during the process of synthesis. This is synthesis by concatenation.

Lexicon

A lexicon is a word list that gives the syllabic structure, lexical stress and pronunciation of words in some phoneme set and helps to find the pronunciation of a word especially for languages do not have a one-to-one mapping between graphemes and phonemes. Basically,

a lexicon is a list of words with additional syntactic (and possibly morphological and semantic) information, which is supposed to be useful for text processing.

Letter-to-sound rules

A letter-to-sound rule is a well-defined mapping between the graphemes, i.e., alphabets of a language and the phonemes or sounds of that language. Letter-to-sound rules can be built when the language has a direct one-to-one mapping between orthography and sound system. These rules are very useful in converting text into speech in a text-to-speech synthesis system.

Waveform synthesis

Waveform synthesis is the final step in speech synthesis where the sound waves (of synthetic speech) are produced by the synthesizer on the basis of the input information. Waveform synthesis is that step where we get the output of synthetic speech. The input information can be on the basis of articulatory, concatenative, or formant synthesis techniques.

Power normalization

In a set of diphones, there is often a mismatch or a general power difference between files. This difference has to be neutralized and can be done by finding the mean power for each vowel in each file and then calculating a factor with respect to overall mean vowel power. Power normalization eliminates the artifacts caused by variations in recording environment or variations in speaker's pronunciation while recording the diphones.

Text processing or Natural Languae Processing (NLP)

Text processing or NLP converts an arbitrary text into identifiable words chunked into reasonable sized utterances. It performs the text analysis, grapheme-to-phoneme transcription, prosody generation, morpho-syntactic analysis and syntactic processing. It reduces an orthographically represented sentence into its internal structure (words, phrases etc.), identifies the part of speech of each word, extracts morphological information, describes words on the basis of other words and disambiguates them contextually.

Linguistic processing or Digital Signal Processing (DSP)

Linguistic/Prosodic processing is the translation of words to segments with durations and F0 contour. The quality of speech, its naturalness, intelligibility and acceptability largely depends on the appropriateness of the phonetic and prosodic output. DSP generates waveforms from the output of the NLP module, which it reads out and is also called speech synthesis.

Utterance structure

An utterance consists a set of items, which are related through a set of relations. An item may

be in one or more relations. Each relation consists of a list or tree of items. Items are used to represent objects like words or segments, and has a number of features associated with it Relations are used to link items together is useful ways, such as in a list of words, or a syntax tree or the syllable structure. Individual items may be in multiple relations.

Radio mapping or bootstrapping

It is done to setup functions to generate prompts for those languages that do not exist in the synthesizer. At synthesis time each phone of the non-existing language must be overlapped to an equivalent existing phone. If the equivalent is not present, i.e. if the concerned languages have different sounds, the closest equivalent is taken into consideration. This mapping or overlapping is called “bootstrapping”. A simple mapping is given between the target phone set and existing synthesizer’s phone set.

Tokenization

Tokenization is to relate each token to zero or more words. This is done on the basis of context-sensitive rules and takes into account numbers, dates, money, some abbreviations, email addresses and random tokens.

Co-articulatory effects

Co-articulation is simultaneous articulation of two successive phonological units such that the two units cannot be separated. However, in speech processing, co-articulatory effects have a little extended meaning – coarticulatory effects are the transitions between two successive sounds that are unique to those two sounds and give them continuity in speech. In simple terms, it means the transition in speech between the two successive sounds created by trailing off of one sound and beginning of the other.

Fixed Prosody

Fixed Prosody (FP) sets the F0 values. The FP_F0 value is the Fixed Prosody for F0 value for utterance. As stated in the Festival manual, FP_F0 is: “*In using Fixed_Prosody as used in Phone type utterances and hence Say Phones, this is the value in hertz for the monotone F0*.” In short, fixed prosody the monotone gives a monotone to the synthesized speech in the absence of actual prosody, accent, stress and intonation.

DC Bias

DC Bias is a kind of noise where 0Hz audio introduced by DC (direct current) power supplies. This is usually inaudible, but can cause a sharp pop at the beginning or end of a recording.

Sonorants

A speech sound in which air flows smoothly through the vocal tract. The feature [+sonorant] is

defined by Chomsky and Halle as the sound "*produced with a vocal tract configuration in which spontaneous voicing is possible*."

Demisyllables

Demisyllables are units that represent initial and final parts of syllables. They are formed by splitting the syllables in two at the vocalic nucleus, which acts as a co-articulation barrier in most cases. They require less concatenation points and take into account most of the coarticulation effects. But it is not possible to calculate the exact number of demisyllables in a language.

Phonotactic constraints

Phonotactic constraints are constraints on the relations of sequences of phonemes or phonological units in a language. as such, the language does not permit some phone sequences in certain contexts. The phonology of the language restricts some phonemes to occur together. For example, in English, "str" is allowed word initially but "sfl" is not.

Triphones

Triphones are longer segmental units, which are like diphones except that they include a complete phone between two half phones or steady states (half phoneme – phoneme – half phoneme). The number of triphones in a language is very large and they require more concatenation points.

Pitch

Pitch is the property of sounds as perceived by a hearer that corresponds to the physical property of frequency. Frequency is a physical concept, while pitch is a psychological concept, i.e. the ear's response to frequency. The higher the frequency, the higher the hearer perceives the pitch.

Phoneme

Phoneme is the smallest distinct sound unit in a given language. A phoneme is identified in a language as the smallest unit that distinguishes the two words, that is, in case of minimal pairs e.g., in "lame" and "lane", /m/ and /n/ distinguish the two words.

Duration

Duration is the physical length of sounds as measured on a time scale. Length in phonology is a feature that defines the sounds as long or short, but duration is the actual length of a sound as in natural speech and may vary from the parameters of length, example, for some speakers the realization of /a/ is longer than /i/ though both are phonologically short.

SIOD

The Scheme interpreter, SIOD – Scheme In One Defun, was written by George Carrett. It offers a basic small Scheme (Lisp) interpreter and is used for embedding applications such as Festival as a scripting language.

Editline

Editline is a complete command line editing system.

LPC coefficients

LPC, Linear Predictive Coding, is a pitch synchronous technique for power normalization. The LPC parameters and residual files for each diphone in the diphone database are extracted on the basis of the extracted pitchmarks or pitch periods. Before LPC coefficients are calculated, a script calculates the powerfactors that are used to normalize the power of each waveform during the making of the LPC coefficients.

Pitchmarks

Pitch marks are pitch periods extracted from the raw waveform or EGG (Electroglottograph) signal of all the diphones for the purpose of power normalization. These pitchmarks are important for finding LPC coefficients. Finding pitchmarks is similar to finding the F0 contour. In Festival the script 'make_pm' contributes in extracting pitchmarks from the diphones. The quality of speech synthesis largely depends on how well the pitchmarks have been extracted.

Speech synthesis

Speech synthesis system aims at "reading" any given text, whether directly introduced or scanned. It should be able to convert any text into an intelligible, natural sounding speech. It is also called a text-to-speech system and has two parts – Text processing, where a text broken into units like tokens and types such that they can be understood by the computer and DSP, digital signal processing, which converts the processed text into speech.

Articulatory speech synthesis

Articulatory synthesis system attempts to model the human speech production system directly. This method involves models of human articulators and vocal cords. It tries to model the human vocal organs as perfectly as possible so it is potentially the most satisfying method to produce high-quality speech. It uses models of the human articulators and vocal cords with a set of area functions between glottis and mouth. For instance, the parameter may be set for tongue height, tongue position, lip orification, etc.

Concatenative speech synthesis

Concatenative synthesis uses prerecorded samples derived from natural speech and concatenates them to produce synthetic speech. This method requires a database of prerecorded natural utterances such as syllables, demisyllables, phonemes, diphones, and sometimes even triphones. In synthesizing speech, units from the database are concatenated on the basis of the grapheme-to-phoneme rules.

Formant speech synthesis

Formant Synthesis is based on the source-filter model approach. This method models the source sounds and the formant frequencies and describes speech on the basis of formants, bandwidths, and glottal waveforms. It can provide infinite number of sounds that makes it more flexible. The excitation signal could be either voiced with fundamental frequency (F0) or unvoiced noise. A mixed excitation of these two may also be used for voiced consonants and some aspiration sounds. The excitation is then gained and filtered with a vocal tract filter that is constructed of resonators similar to the formants of natural speech. For intelligible speech three formants are required and for high quality speech up to five formants.

Prosodic phrasing

Prosodic Phrasing in speech synthesis leads to an understandable and intelligible speech. A natural continuous speech does not occur non-stop, but depends on the size of the breath. With each breath, we can say, we can define prosodic phrases. The intonation and accent in a speech is a result of these prosodic phrases. The prosodic phrases may or may not be marked by punctuation.

UniSyn database

The UniSyn synthesis module uses databases in basic two formats – separate and grouped. A separate format is such where, during synthesis, all the files (made in the process of synthesis like pitchmarks, LPC etc.) can be accessed individually. Group format, on the other hand is such where a database is collected together into one file containing all the relevant information necessary for waveform synthesis.

Prosody

Prosody refers to properties of a speech signal such as changes in pitch, loudness, syllable length etc. Prosody also includes aspects related to speech timing, rhythm, and speech rate. . Prosodic events are time-aligned with syllables; hence they are also called suprasegmental features. The most noticeable components of prosody are – stress, syllabic length, F0, intensity and duration.

Syllable

A syllable is the smallest unit that can be pronounced in one breath. It consists of a nucleus,

that is, a vowel or a sonorant sound and other sounds preceding and following it. The smallest syllable is just one vowel. Every language has its own syllable structure where, it provides certain number of consonants (which is generally maximum four) before or after the vowel. For instance, a possible syllable in any language can be (C)(C)(C)(C)V(C)(C)(C)(C), the C in brackets represent consonant that may or may not occur.

Nasalization

Nasalization is a phonological feature observed in vowels, where a vowel is pronounced with a nasal effect. Nasalization can be a phonological process whereby, a vowel assimilates the nasal quality of the adjacent nasal consonant, or it can occur at the phonemic level, such that nasalized vowels form a part of the phonemes of a language. It is more of a co-articulatory phenomenon with the vowel and nasal sound occurring together.

Intonation

Intonation is the term given to variation in pitch of a speaker's voice over the whole of sentence.

BIBLIOGRAPHY

A. W. Black and N. Campbell. (1995): "*Optimizing selection of units from speech databases for concatenative synthesis*". In Proc. Eurospeech '95, volume 1, pages 581--584, Madrid, Spain.

Ahuja, R., Bondale, N., Furtado, X., Jose, T., Krishnan, S., Poddar, P., Bhiksha, R., Rao P.V.S., Samudravijay, K. (1992): "*Recognition and Synthesis in Hindi Language"*", Proceedings of Workshop on Speech Technology, IIT Madras, India.

Arun Chandra. (1998): "*Compositional experiments with concatenating distinct waveform periods while changing their structural properties*." In SEAMUS'98, Urbana, IL, School of Music, University of Illinois.

C. H. Coker, N. Umeda, and C. P. Browman. (1973): "*Automatic Synthesis from Ordinary English Text.*" IEEE Trans. Audio Electroacoust. 293-97.

D. H. Klatt. (1976): "*Linguistic Uses of Segmental Duration in English: Acoustic and Perceptual Evidence.*" Journal of the Acoustical Society of America, 59, 1208-1221.

D. H. Klatt. (1987): "*Review of Text-to-Speech Conversion for English*." J. Acoust. Soc. Am 82, no.3 737-93.

D .H. Klatt, (1980): *"Software for a cascade /parallel formant synthesizer"*, Journal of the Acoustical Society of America, Vol 67.

Fry, D. B. (1979): "*The Physics of Speech*." Cambridge University Press, Cambridge, England.

H. Dudley, & T. H. Tarnoczy, (1950). "*The speaking machine of Wolfgang von Kempelen.*" Journal of the Acoustical Society of America, 22, 151-166.

H. Dudley, R. R. Riesz, and S. A. Watkins. (1939): "*A Synthetic Speaker*." J. Franklin Inst. 227 739-64.

J. Allen, M. S. Hunnicutt, and D. H. Klatt, (1987) "*From Text to Speech: The MITalk System*." New York: Cambridge University Press.

J. Holmes, I. Mattingly, J. Shearme, (1964): "*Speech synthesis by rule",* Language and Speech, Vol 7.

J. P. Olive and L. H. Nakatani. "*Rule Synthesis of Speech by Word Concatenation: A First Step.*" Journal of the Acoustical Society of America 55 (1974): 660-66.

J. P. Olive, A. Greenwood, and J. Coleman. (1993): "*Acoustics of American English Speech"*. New York:

Springer.

J. N. Holmes. (1983): "*Formant synthesizers: Cascade or Parallel*". In Speech Communication, volume 2, pages 251-273.

J. L. Flanagan, (1972): *Speech Analysis, Synthesis, and Perception*, Springer Verlag, pp. 204-210.

John E. Clark and Colin Yallop. (1996): "*An Introduction to Phonetics and Phonology".* Blackwell, Oxford.

Krishnamurti, Bh., C. P. Masica and A. K. Sinha (eds.) (1986): "*South Asian Languages: Structure, Convergence and Diglossia"*. Motilal Benarsidass, Delhi.

Mark Beutnagel, Alistair Conkie, and Ann K. Syrdal. (1999): "*Diphone Synthesis using Unit Selection".* In The 3rd ESCA/COCOSDA Workshop on Speech Synthesis, Jenolan Caves, Australia.

Masica, Colin P. (1991): "*The Indo-Aryan Languages*". Cambridge University Press, Cambridge.

N. Campbell. (1996): CHATR, "*A high-definition speech re-sequencing system* ". Acoustical Society of America and Acoustical Society of Japan, Third Joint Meeting.

N. Chomsky and M. Halle. (1968): "*The Sound Pattern of English"*. Harper & Row, New York, NY

R. Carlson and B. Granströum. (1976): "*A Text-to-Speech System Based Entirely on Rules*." Proc. Int. Conf. Acoust. Speech Signal Process. ICASSP-76, 686-88.

R. Carlson, B. Granström, S. Hunnicut, "*A multi-language Text-To-Speech module*", ICASSP 82, Paris, vol. 3

S. Hunnicut, "*Grapheme-to-Phoneme rules : a Review*", Speech Transmission Laboratory, Royal Institute of Technology, Stockholm, Sweden, QPSR 2-3, pp. 38-60.

T. Dutoit, (1996) "*An Introduction to Text-To-Speech Synthesis"*¸ Kluwer Academic Publishers, 326 pp.

T. Dutoit, H. Leich, "MBR-PSOLA, (1993): "*Text-To-Speech Synthesis based on an MBE Re-Synthesis of the Segments Database",* Speech Communication, Elsevier Publisher, vol. 13, n°3-4.

WEB SITES REFERRED

Alan W Black and Kevin A. Lenzo, *Building Voices in the Festival Speech Synthesis System Processes and issues in building speech synthesis voices*, Speech Group at Carnegie Mellon University, from the Web site,
URL: http://festvox.org/festvox/

Articulatory Speech Synthesis by Rule: *Implementation of a Theory of Speech Production*, from the Web site,
URL: http://www.essex.ac.uk/speech

Center for Speech Technology Research, University of Edinburgh. (1999): from the Web site,
URL: http://www.cstr.ed.ac.uk/.

CSLU Speech Synthesis Research Group, Oregon Graduate Institute of Science and Technology. (1999): from the Web site,
URL: http://cslu.cse.ogi.edu/tts.

CSTR Centre for Speech Technology Research, from the Web site,
URL: http://www.ircam.fr/equipes/analyse-synthese/schwarz/thesis/allbib/allbib005.html

Description to TTS, from the Web site,
URL: http://www.kgw.tu-berlin.de/~felixbur/ttsDemos.html

Eric Keller, Gérard Bailly, Alex Monaghan, Jacques Terken and Mark Huckvale, (eds.) *Improvements in Speech Synthesis*, from the Web site,
URL: http://www.unil.ch/imm/cost258volume/cost258volume.htm

Examples of Synthesized Speech, from the Web site,
URL: http://www.ims.uni-stuttgart.de/~moehler/synthspeech/examples.html

Historical images of speech synthesis, from the Web site,
URL: http://www.ims.uni-stuttgart.de/~moehler/ISCA-SynSIG/historical.html

Information on BaBel, from the Web site,
URL: http://www.babeltech.com/html/frame/synthesis/fintrosyn.html

Information on BeSTspeech, from the Web site,
URL: http://www.alternateaccess.com/products/texttospeech.html

Information on DECtalk Speech Synthesis, from the Web site,
URL: http://www.systems.digital.com/
URL: http://www.worklink.net/products/dectalk.html

Information on ETI-Eloquence, from the Web site,
URL: http://www.eloq.com/

Information on Eurovocs, from the Web site,
URL: http://www.elis.rug.ac.be/t&i

Information on Infovox, from the Web site,
URL: http://www.promotor.telia.se/infovox/index.htm

Information on IPOX, from the Web site,
URL: http://www.tue.nl/ipo/people/adirksen/ipox/ipox.htm

Information on Klatt-style Synthesizer, from the Web site,
URL: ftp://svr-ftp.eng.cam.ac.uk/pub/comp.speech/synthesis/klatt.3.04.tar.gz

Information on Laureate, from the Web site,
URL: http://www.htk.co.uk/

Information on Lernout & Hauspie Madison[tm] , from the Web site,
URL: http://www.lhs.com/

Information on Listen2 Text Reader, from the Web site,
URL: http://www.islandnet.com/jts/

Information on MBROLA, from the Web site,
URL: http://tcts.fpms.ac.be/synthesis/mbrola.html

Information on Narrator Translator Library, from the Web site,
URL: ftp://ftp.doc.ic.ac.uk/pub/aminet/dev/src/tran42src.lha

Information on PlainTalk, from the Web site,
URL: http://www.speech.apple.com/

Information on ProVerbe Speech Engine, from the Web site,
URL: http://www.elan.fr/

Information on ProVoice Developer's Speech Toolkit, from the Web site,
URL: http://www.firstbyte.davd.com/

Information on rsynth, from the Web site,
URL: http://www.tios.cs.utwente.nl/say/

Information on "Speak" - a Text-to-Speech Program, from the Web site,
URL: ftp://wilma.cs.brown.edu/pub/speak.tar.Z

Information on VECSYS, from the Web site,
URL: http://www.vecsys.fr/tap/produits.html

Information on WinSpeech, from the Web site,
URL: http://www.pcww.com/index.html

JOINT SPEECH RESEARCH UNIT (JSRU), from the Web site,
URL: http://www.mindspring.com/~ssshp/ssshp_cd/ss_jsru.htm

Multilingual Text-to-Speech Systems, from the Web site,
URL: http://www.bell-labs.com/project/tts/

On-line Publications for Festival and Speech Synthesis Topics,
from the Web site,
URL: http://www.cstr.ed.ac.uk/projects/festival/papers.html

Selection of the Speech Units, from the Web site,
URL: http://www.ias.et.tu-dresden.de/kom/lehre/tutorial

Speech Synthesis & Recognition, from the Web site,
URL:
http://www.infoweblinks.com/content/speechsynthesis&recognition.htm

Speech synthesis under Linux, from the Web site,
URL: http://www.freeos.com/articles/2613/2/1-3/

Text-to-Speech, from the Web site,
URL: http://www.commweb.com/article/COM20001003S0023/2

The Festival Speech Synthesis System, from the Web site,
URL: http://www.cstr.ed.ac.uk/projects/festival/

The Linguist List, from the Web site,
URL: http://www.linguistlist.org/

The Naturalness of Synthetic Speech, from the Web site,

URL: http://www.unil.ch/imm/docs/LAIP/COST_258/cost258.htm

Units of Representation for Speech Synthesis, from the Web site,
URL: http://www.essex.ac.uk/speech/archive/ss-units/ss-units.html