Urdu Text to Speech System for Navigation App

By
Muhammad Asfand
00000119770

Supervisor
Dr. Ali Tahir
Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of Masters in Commuter Science (MS CS)

In
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(August 2019)

Acceptance Certificate

Certified that final copy of MS/MPhil thesis written by **Muhammad Asfand**, Reg no. **00000119770**, of SEECS has been vetted by under- signed, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfill- ment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: _____

Name of Supervisor: _____
Date: _____

Signature(HOD): _____
Date: _____

Signature(Dean/Principal): _____
Date: _____

Approval

It is certified that the contents and form of the thesis entitled "**Urdu Text to Speech System for Navigation App**" submitted by **Muhammad Asfand** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Ali Tahir**
Signature: _____

Date: _____

Committee Member 1: **Dr. Faisal Shafait**

Signature: _____
Date: _____

Committee Member 2: **Dr. Safdar Abbas**

Signature: _____
Date: _____

Committee Member 3: **Dr. Qaiser Riaz**

Signature: _____
Date: _____

Abstract

Due to increase in taxi hailing service (Uber & Careem), a demand for Urdu text to speech system was increasing. This system would be used in a Pakistani navigation app which would facilitate the drivers(captains) in finding a way without keeping an eye on the mobile. The goal is to provide Pakistani people the language in which they are most comfortable i.e. Urdu. In Pakistan, most of the drivers are illiterate and don't understand English commands by Google Maps. So we developed an Urdu Text To Speech System in which the commands during turn by turn navigation would be in Urdu. Some of the main steps involved are Lexicon generation, LTS rules generation, Data labeling, Pitch marking, extracting Mel Frequency Cepstral Coefficents, building HMM model. Once the model is generated, we will export it to Android devices. We will check the quality of both models i.e. original and the one ported to Android. Our testing is done using both manual and automatic methods.
Keywords - Urdu Text to Speech, Hidden Markov Model, navigation, Mel Frequency Coefficients.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Muhammad Asfand**
Signature: _____

Acknowledgment

Thanks to Almighty Allah for giving me knowledge, power and strength to accomplish this task. I learned a lot from this project and this will certainly help me in forthcoming life.
I would like to give a big thanks to our supervisor Dr. Ali Tahir who guided me throughout. With his expert knowledge and experience we made an innovative and successful project. Also I would like to thank all Committee members for there support.
I would also like to thank my team lead Faraz Khalid for his help and guide throughout. Also I would like to thank Muhammad Ahmed from BESE 15 batch for his help in finding the cause of an issue I was facing.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

Text to speech (TTS) is a system which takes text as an input and gives corresponding audio as an output. It is also referred as speech synthesis. The end goal is to create an artificial human voice. A text to speech system is usually concatenative based or parametric based. Concatenative TTS produces a more natural sound whereas parametric TTS is less natural but with better pronunciation and more control. The reason why concatenative is more natural is because its stores recording of speech and combines them part by part to form a sentence. This obviously will require a large amount of storage. On the other hand parametric TTS stores features of speech which takes less storage. So our focus in this thesis will be parametric TTS as we want our system to be lite and have better pronunciation. In parametric TTS, we will be using Hidden markov model (HMM). HMM based speech synthesizer works in following way:

- Training part
  - Extracting mel-cepstral coefficients
  - Extracting excitation parameters
- Model training
- Synthesis
  - Conversion of text to phones
  - Parameter generation algorithm
  - Waveform Generation

Generally, in order to train such a system, one require audio files with there transcriptions. One has to then map audio files with each characters in there

respective transcription file. This will train the system to know how each character is spoken. Generally the steps involved in TTS are text processing, feature extraction and waveform generation as shown in figure 1.1. These steps will be discussed in detail later on.

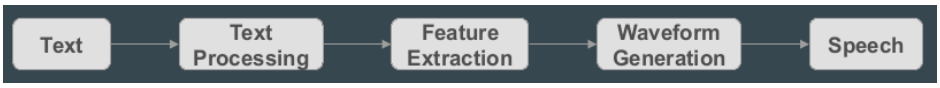


Figure 1.1: TTS General Flow

1.1 Brief History

In 1936, we saw the 1st practical use of speech synthesis. Britain General Post Office introduced a speaking clock. All the words were prerecorded and stored on optical disks. One disk each for hours, minutes and seconds. They were concatenated to givea output time. After this several improvements and methods have been invented in this domain. After that different methods were tried but mainly consist of concatenation of recorded speech. It saw an increase in 1970s when electronic storage started become cheap and robust. After that work was done to process text to speech, form text analysis, prosodic prediction, phoneme generation, and waveform synthesis. As time passes, the hardware prices were becoming cheap and people started working on different methods for concatenative speech. A much larger inventory of concatenative units were introduced where each previously unit now has multiple options and are selected based on some rules. In early 2000s, a new statistical method was developed which instead of selecting unit, they were generating them. Thus came HMM synthesis which upon the course of time has been developing.

1.2 Motivation

I am a frequent Uber and Careem user and had been using it to travel to and from university & office. During the ride, conversation with the captain(driver) normally starts and they end up asking what do I do. And I tell them that I work in TPL Maps where we make maps. They ask that maps like Google Maps? And I reply yes. Then they ask whether we provide navigation or not. To which I say of-course we do provide navigation. Then they ask is it in **Urdu**? And I say no. Then they tells me that they are

not literate and don't understand English routing instruction. And they find it difficult to use mobile during driving. So they asked me if somehow we can integrate Urdu navigation system. These type of conversation led us to think that there should be a system for it in Urdu. Upon searching on the internet we didn't found one and hence we decided to work on **Urdu Text to Speech System for Navigation app**.

1.3 Problem Statement and Contribution

So our purpose here is to create a TTS system for a navigation app which the drivers can use without looking at there phones. There were two ways to do it. One was to record and store all the possible words which could be used during navigation or to create a TTS system which would synthesize the given text. The former method would require storing of each word's audio file which would take lot of memory space. We choose the latter as it is more resource efficient, provides more control and can incorporate new words too. So our aim is to create a system which is given any Urdu routing sentence and it will synthesize it.
When we searched for Urdu TTS online, we only found one which was made by Center for Language Engineering(CLE) Lahore. But the problem was that they were charging for its use. The rate of usage is PKR 1.00 per transaction plus PKR 0.01 per character exceeding 100 characters length. Pricing is not acceptable as we can't impose pricing on captains. So we decided to make our own TTS which would be incorporated in TPL Maps and would be available free. The major limitation was that we were using University of Edinburgh's Festival tool for this purpose and they were using LISP language. It was difficult understanding LISP and getting to know the workflow. Also work on Urdu TTS was very limited so we had to figure out most things and took help from English TTS. Also another problem was collection of audio data and preparation of Urdu dictionary which is required for navigation purpose. We had to prepare Urdu data for our need and modify Festival & Flite to go with our changes.
Another major problem we faced was that we have to use Urdu in Roman style instead of Nastaliq. The reason was that we had to use data provided to us by TPL Maps. And there data was in Roman script. So we had two options; either convert Roman urdu in Nastaliq or make TTS using given style. Developing TTS in Nastaliq style would help us categorized and map characters with vowels easily but using Roman style it becomes difficult as number of alphabets don't match. English has 26 alphabets whereas Urdu has around 40. So we need to map or differentiate some vowels using two

alphabets. For example ch would be use as chay of Urdu etc. But this causes another problem as there are words which might use ch as kay of Urdu e.g. school. Now here we need to differentiate both sounds using ch. This would not have happen if we had used Nastaliq style. But still we had to choose Roman style as we had to rely on automatic conversion from Roman to Nastaliq style which in many cases were giving wrong results. Forexample Ideas by GulAhmed was converted to Gulahmed k khyallat which was wrong in a sense that this shop name had to remain same in Urdu as well. Plus manual conversion was not possible as it was a huge dataset and would require a large amount of time. Also the data was not final. TPL Maps were carrying out survey so they plan to increases there data with the passage of time. So we had to go with there standards.

Chapter 2

Literature Review

In (Tokuda and Black., 2002), authors have worked on parametric based synthesis and described HMM based speech synthesis system for English. They have described both the training and synthesis part of the method. In (Black, 2006), they have presented CLUSTERGEN method which they have integrated in FESTIVAL to build speech models. Our focus will be on this method as we will be using it in our case to build model for Urdu TTS. Also we will be using FESTIVAL toolkit developed by them in simplifying our tasks to build a parametric based Urdu voice. In (Zen et al., 2007), they have described the new and improved version of HMM-based speech synthesis system called HTS version 2.0. In this version they improved the performance, removed bugs and added new features which are helpful in synthesis process. In (Zen et al., 2009), gives an overview of HMM based synthesis method and its advantages and disadvantages compared to others. According to it, we have following reasons to go for HMM based synthesis:

- Modifiable voice characteristics
- Can be used for multiple languages
- As compared to unit selection, it requires very less data.
- Speech Recognition techniques can be used with it

In (Oord et al., 2016), a different technique was developed for synthesizing speech from text. The technique which they developed is called Wavenet and is based on deep neural net. Its similar to PixelCNN. They claim that the voice generated from it appears more natural then that concatenative or parametric based voice. In (Oord et al., 2017), Wavenet was improved as the orginal one doesn't support parrallel processing and took too long to synthesize. They claimed that it is twenty times faster than that of original

one.
In (Kabir H., 2002), they used NLP for Urdu TTS System (Preprocessing and Phonological processing). In (Hussain, 2004), author have discussed Urdu letter to sound rules. He classified the Urdu letters and aerabs in different categories and then explains about there placement in a word and how they act as vowel or consonant. In (Hussain, 2005), author discusses phonological processing for Urdu TTS system in which normalized text is converted to annotated phone string. The flow which he described is as shown in fig 2.1
In (Ijaz M., 2007), they have developed a Urdu Lexicon from a corpus which

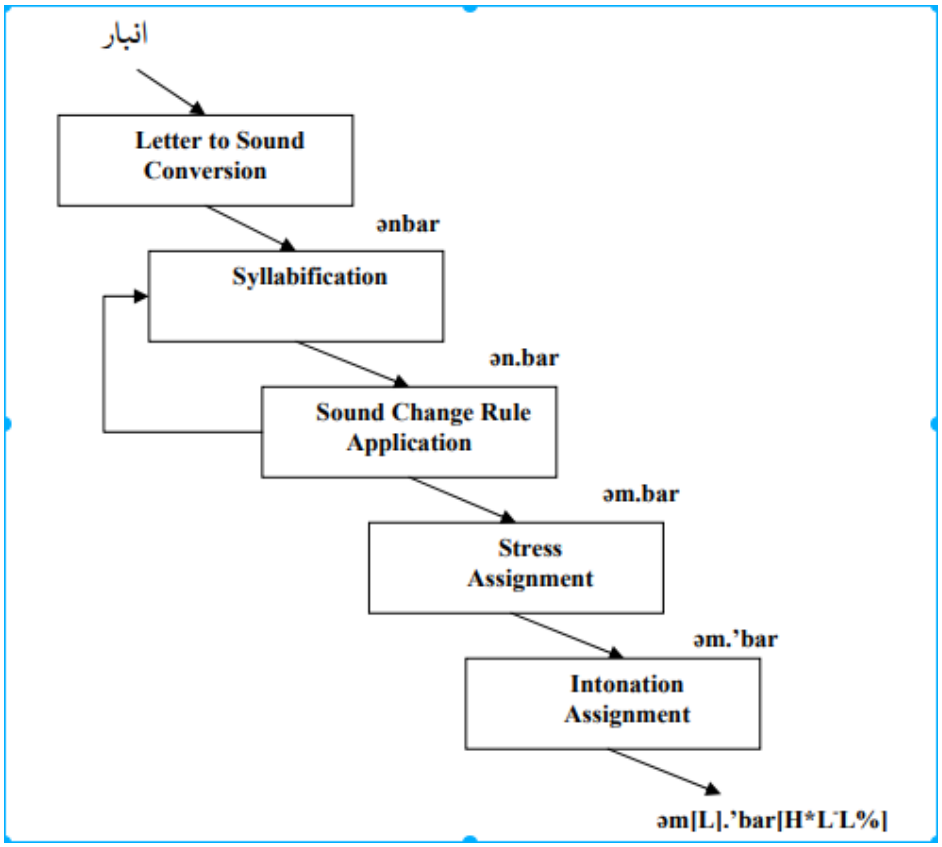


Figure 2.1: phonological processing flow chart

they collected from different sources. They followed followings steps:

- Parts of speech tags
- Lemmas
- Phonemic Transcription

- Lexicon Format

In (Basit and S., 2014), they have worked further in text processing for Urdu TTS System and converted text to 8-bit Urdu Zabta Takhti (UZT) (Tokenization, Semantic tagger, Text generation). In (O. and T., 2014), the authors have used HTS toolkit for the development of HMM based Urdu synthesizer. Also they have used greedy search algorithm to select 200 high frequency words. They have divided the development in two parts which is training and synthesis. There results were good but were very limited to selected data as there database consists of only 30 minutes of recording which is less. In (Mumtaz B. and W., 2015),they have worked on stress annotated urdu speech corpus to build female voice for TTS. In (M. and B., 2016), they have perform testing on the current TTS system of CLE. They presented the Phonological Rules and there testing results on multiple speakers dataset. In(Qasim M. and S., 2016), they have worked on the development of speech corpus for travel domain. It is similar to what we want as travel domain is closely related to navigation domain. Although there vocabulary only consists of 250 items(summary is shown in table 2.1), we are doing it on a little larger vocabulary than this. They had an accuracy of 87.21%. In (Adeba F. and K.S., 2016), they have compared both speech

Dimension	**Vocabulary size**
Confirmation	2
Bus Preference	2
Number of Seats	11
Day of Reservation	23
Destinations	44
Time of Reservation	150

Table 2.1: Statistics of speech corpus for travel domain

synthesis techniques for Urdu Text i.e. HMM based & Unit Selection. They have used Festival framework for there development. They tested the system using Urdu ASR. There results showed that HMM based synthesizer has higher accuracy than Unit Selection. There TTS architecture is shown in figure 2.2. In (R. and B., 2016), they have developed a list of diphthongs for Urdu language. Diphthongs are sound formed by the combination of two vowels. They have done this using both manual and digital identification. In (Shahid Kh. S. and Haq., 2016), they have developed a test to measure Urdu speech quality of the TTS system there center developed. For this purpose they had total of 23 urdu native speakers. They were played the recordings

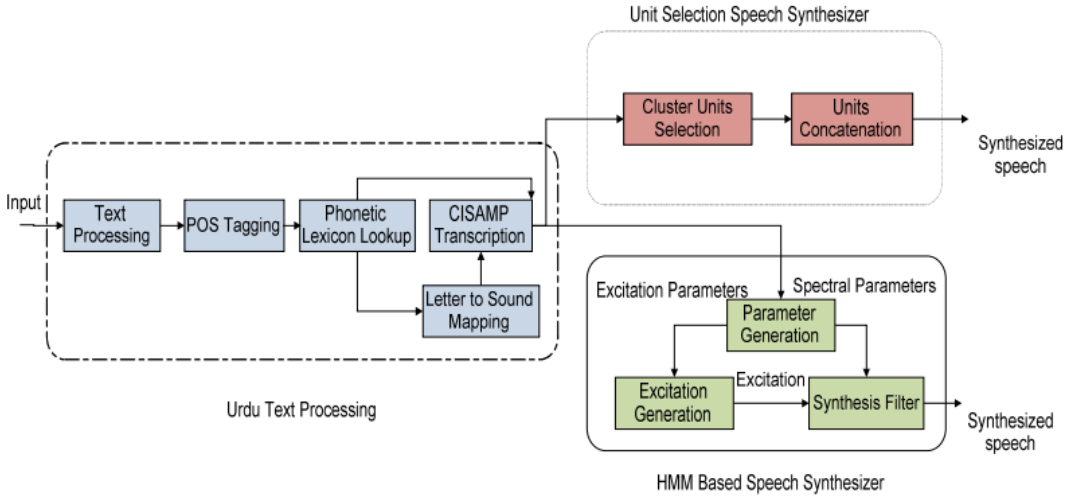


Figure 2.2: Urdu TTS architecture for HMM and Unit selection Synthesizer

and there response was recorded. Tests were divided into two categories which are as follows:

- Intelligibility test
- Naturalness test

In intelligibility test, they speakers were asked to identify the words whereas in naturalness, they were asked to rate how natural in there opinion is the sound.

Chapter 3

Design and Methodology

3.1 LEXICON

Lexicon is a list of all the words in the dictionary we have along with there pronunciations. By pronunciation, we mean that each word is segmented in to there corresponding phonemes as shown in figure 3.1. For example lets take phoneme distribution of word Lahore which is as follows:
("lahore" n (((l A h O r) 0)))
If you ignore n and 0 (will be discussed later on format section), you can see that Lahore is breakdown into five phonemes instead of six. The reason is that although its spelling contains an 'e', there is no sound of it during pronunciation so we would not include its relative phoneme in Lexicon. This would help our model to know how to pronounce these words. So to develop a text to speech system, we first need to make a LEXICON file. To make a complete and a perfect lexicon file, is a difficult task as it would require as to include all the words with there pronunciation which could possibly be used in our TTS. Now imagine if you have to do this for a ten thousands words. How much time it would require just to make a lexicon file then?

Lets consider a single word requires at-least 5 min of time for us to break it into phonemes. So ten thousands words would require at-least 35 days with non stop processing which is not possible. Lets say we work 12 hours a day then it would take 70 days just to make this lexicon file. You might think that I am over exaggerating here by saying at-least 5 min, but in truth it might take more. The reason is that you have to speak each word again and again in order to understand how it is broken down in terms of phonemes. You have to check which parts of your mouth are being used to pronounce and select corresponding phoneme then. Now this is your thinking, what about others. You have to ask others of there opinions as well. You have

```
("a" nil (((e) 0)))
("aa" nil (((A) 0)))
("aaa" nil (((t r I p) 0) ((y l) 0) ((e) 0)))
("aaapartment" nil (((A) 0) ((y p) 0) ((y r t) 0) ((m Y n t) 0)))
("aaay" nil (((A) 0) ((e) 0)))
("aab" nil (((A b) 0)))
("aaba" nil (((A b) 0) ((A) 0)))
("aaban" nil (((A b) 0) ((A n) 0)))
("aabd" nil (((A b dd) 0)))
("aabelting" nil (((A b) 0) ((Y l t) 0) ((I N) 0)))
("aabi" nil (((A b) 0) ((i) 0)))
("aabpara" nil (((A b p) 0) ((A r) 0) ((A) 0)))
("aabroo" nil (((A b) 0) ((r u) 0)))
("aabrother" nil (((A b) 0) ((r y dd) 0) ((y r) 0)))
("aabrothers" nil (((A b) 0) ((r y dd) 0) ((y r z) 0)))
("aabshaar" nil (((A b sh) 0) ((A r) 0)))
("aabshar" nil (((A b sh) 0) ((A r) 0)))
("aachi" nil (((A ch) 0) ((i) 0)))
("aad" nil (((A dd) 0)))
("aada" nil (((A dd) 0) ((A) 0)))
("aadab" nil (((A dd) 0) ((A b) 0)))
("aadcen" nil (((A dd s) 0) ((Y n) 0)))
("aadi" nil (((A dd) 0) ((i) 0)))
("aadil" nil (((A dd) 0) ((I l) 0)))
("aadmani" nil (((A dd) 0) ((m A n) 0) ((i) 0)))
("aado" nil (((A dd) 0) ((o) 0)))
("aadowal" nil (((A dd) 0) ((o v) 0) ((A l) 0)))
("aafaah" nil (((A f) 0) ((A) 0)))
("aafandis" nil (((A f) 0) ((y n dd) 0) ((i s) 0)))
("aafaq" nil (((A f) 0) ((A q) 0)))
("aafi" nil (((A f) 0) ((i) 0)))
("aafia" nil (((A f) 0) ((I) 0) ((A) 0)))
("aaftab" nil (((A f) 0) ((tt A b) 0)))
("aag" nil (((A g) 0)))
("aaghaush" nil (((A ggh) 0) ((O sh) 0)))
("aaghosh" nil (((A ggh) 0) ((o sh) 0)))
```

Figure 3.1: Subset of a Lexicon file

to refer some language professional for this purpose as well. Now you might be thinking that 5 min is less for this. In reality it is less and the number of words is not ten thousands but greater than this around fifty thousands plus. So how would you do this?
An approach recommended by University of Edinburgh is that you make lexicon file of most used words and then automatically build LTS rules from data. Then test the results of it by passing the remaining words. If the phonemes are correct then add them as it is in our lexicon otherwise correct them and then add them. This approach reduces the time for making a lexicon file. In our case, a lexicon was already developed by my supervisor Dr. Ali Tahir and his students. They developed it for speech to text recognition system. We used that lexicon, modified it our desired format and added missing words in it as well.

3.1.1 Lexicon format

A lexicon files have many lexicon entries corresponding to each word. Each entry is separated by a new line. Each lexicon entry has three parts which are:

- Word
- Part of speech
- Pronunciation

Lets again take the example of Lahore but this time we will use a little advance version of it. Its lexical entry is as follows:
("lahore" n (((l A h) 0) ((O r) 0)))
Now here lahore is the word whose pronunciation we are defining. 'n' tells us the part of speech which here means its a noun. If 'v' was used then that would have mean verb. Similarly other parts of speech can be specified here. After that is the pronunciation part. Pronunciation part is further divided into small parts which are:

- Phones
- Stress marking
- Syllables

You might have noticed that pronunciation this time is different from previous and looks more complex. The reason is that we have used syllabification.

Previously, I had mentioned that l A h O r are the phonemes. But what is 0 and why they are separated this time. 0 is used to express the stress on vowels. The reason why they are separated is because of syllabification. We have divide each word on the basis of its corresponding syllable. l A h makes one syllable and O r makes another.

3.1.2 ADDENDA

Addenda is similar to lexicon but small in size. It contains words and there pronunciation which are not part of lexicon. They are usually used when some words are missing in Lexicon or you want to have small dictionary for a particular application. Abbreviations are also added here. In our case we have only added punctuation in it as we need to ignore them regarding there pronunciation as shown in fig 3.2.

```
(define (tpl_urdu_addenda)
  "(tpl_urdu_addenda)
Basic lexicon should (must ?) have basic letters, symbols and punctuation."
;; Basic punctuation must be in with nil pronunciation
(lex.add.entry '("." punc nil))
(lex.add.entry '("'" punc nil))
(lex.add.entry '(":" punc nil))
(lex.add.entry '(";" punc nil))
(lex.add.entry '("," punc nil))
(lex.add.entry '("-" punc nil))
(lex.add.entry '("\"" punc nil))
(lex.add.entry '("`" punc nil))
(lex.add.entry '("?" punc nil))
(lex.add.entry '("!" punc nil))
)
```

Figure 3.2: Addenda

3.2 Letter To Sound Rules

Now we have developed our Lexicon and Addenda, we are all good to go. But what happens if a word comes which is not in dictionary. Or what happens if a word's spelling is not right. For example instead of 'i', 'e' is used or instead of 'u', 'o' is used. Our system will reject those words as they don't match in either Lexicon or in addenda. Also you can't add all those possibilities in Lexicon. Just think of time it would take to think of all possibilities of each word and the time to add them in Lexicon. So what do we do now?
The answer is that we write rules to handle such situations. These rules are

called letter to sound rules as they will map how each letter will be pronounced with others letters. Generally when a word is not found in lexicon, an error is returned which states that word is not in dictionary. However we will not return error and will use letter to sound rules. So how to build these letter to sound rules. There are two ways to do this which are:

- Manual writing
- Automatic building

Manual writing as the name suggests is that you manually define those rules. Although this is a difficult task as it is difficult to find all the rules and then manually add them but it guarantees that all those words which fall under those rules are pronounced correctly. Rules are defined in following format:
(LEFTCONTEXT [ITEMS] RIGHTCONTEXT = NEWITEMS)
where ITEMS means the letter or group of letters whose pronunciation needs to be defined provided that it comes after a letter or group of letters or none shown by LEFTCONTEXT and is followed by a letter or group of letters or none shown by RIGHTCONTEXT. Lets explain it with an example. Consider the following two words which are school and sacha. The ch sound in both will be different. In school it will give a sound of kay where as in sacha it will give sound of chay. So to differentiate we would first identify the difference between them. We noted that all words in which ch comes after s and is followed by o, it gives sound of k. So we will make a rule of it which as follows:
(s [c h] o = k)
Now in all the words like school, scholar and schooling; k sound would be produce in place of ch. Finding all such rules is a difficult and time taking task. We generally note the common or most use rules manually. For the rest we would do it automatically by using our Lexicon. This will work by checking the relationship of letters in our Lexicon. The steps involved in it are:

- Define allowables
- Align letters
- Find probabilities
- Building CART models

First we need to define allowables. Allowable is a list of letters with its possible phone. Meaning that only these phones can replace that letter. For example letter b can be replaced either by 'b' or 'bh' or '_epsilon_' (_epsilon_

```
(set! allowables
      '
      ((a _epsilon_ A e E Y y O An yn en)
       (b _epsilon_ b bh)
       (c _epsilon_ ch k chh sh s)
       (d _epsilon_ d dd j dh ddh)
       (e _epsilon_ I i e A E Y y yy)
       (f _epsilon_ f)
       (g _epsilon_ g gh ggh)
       (h _epsilon_ h A e E)
       (i _epsilon_ I yy i e A E in en)
       (j _epsilon_ j jh)
       (k _epsilon_ kh kkh k)
       (l _epsilon_ l ll)
       (m _epsilon_ m)
       (n _epsilon_ n N An en in Yn un On on)
       (o _epsilon_ u U o O A on y)
       (p _epsilon_ p ph)
       (q _epsilon_ k q)
       (r _epsilon_ r rh d)
       (s _epsilon_ y A e s z sh)
       (t _epsilon_ t th tth tt sh ch dd)
       (u _epsilon_ u U o O A un y)
       (v _epsilon_ v)
       (w _epsilon_ v u o O)
       (x _epsilon_ k s z)
       (y _epsilon_ y Y yn Yn i I in In e E A en yy)
       (z _epsilon_ z s)
       (# #)
       ))
```

Figure 3.3: List of allowable

```
;; LTS rules
(set! tpl_urdu_lts_rules '(
(a
 ((n.name is #)
  ((p.name is e)
   ((p.p.name is s)
    (((_epsilon_ 1) _epsilon_))
    ((p.p.name is t)
     (((_epsilon_ 1) _epsilon_))
     (((A 0.833333) (_epsilon_ 0.166667) A))))
   ((p.name is o)
    ((p.p.name is l)
     (((_epsilon_ 1) _epsilon_))
     ((p.p.name is r)
      (((_epsilon_ 1) _epsilon_))
      (((A 0.818182) (_epsilon_ 0.181818) A))))
    (((A 0.990418)
      (An 0.000942507)
      (e 0.000942507)
      (_epsilon_ 0.000471254)
      (O 0.000314169)
      (y 0.00691172)
      A))))
  ((n.name is a)
   ((p.name is c)
    (((Y 1) Y))
    ((n.n.n.name is #)
     ((p.name is y)
      ((p.p.name is a)
       ((n.n.name is n)
        (((_epsilon_ 0.333333) (Y 0.666667) Y))
        (((_epsilon_ 0.5) (Y 0.5) _epsilon_)))
       ((n.n.name is t)
        (((A 0.5) (_epsilon_ 0.5) A))
        (((A 0.037037) (_epsilon_ 0.962963) _epsilon_))))
      ((n.n.name is k)
       ((p.name is t)
        (((A 0.5) (Y 0.5) A))
        ((p.name is r)
         (((A 0.25) (_epsilon_ 0.25) (y 0.5) y))
         ((p.p.name is s)
          (((A 0.5) (_epsilon_ 0.5) A))
          (((A 0.111111) (_epsilon_ 0.888889) _epsilon_)))))
       ((n.n.name is p)
        ((p.p.name is #)
         (((A 0.333333) (_epsilon_ 0.333333) (Y 0.333333) A))
```

Figure 3.4: Subset of our LTS rules

means nothing). Complete list is shown in figure 3.3. After defining allowables, check that all the letters in your lexicon are aligned with respective phonemes as defined in allowables. If not then add those phonemes in allowables or if you think Lexicon entry is wrong then correct them. Then we will find the number of each letters that appears along with the number of there pairs. This is done to find the probability of each letter to phone conversion given the adjacent letters/phones. Now using these probabilities, we will build a Classification and Regression tree (CART) to store this information. Here CART tree is build using wagon tool. CART tree will help us selecting the phonemes for a word which is not available in lexicon or addenda. Figure 3.4 shows a subset of how our LTS rules look like.

3.3 FESTIVAL based model

Festival is an open source speech synthesis system developed by University of Edinburgh. It is under the Center for Speech Technology Research (CSTR) there. It is written in C++. We will be using it as a basis for generation of our own text to speech system. The general flow of Festival is shown below in Figure: 3.5.

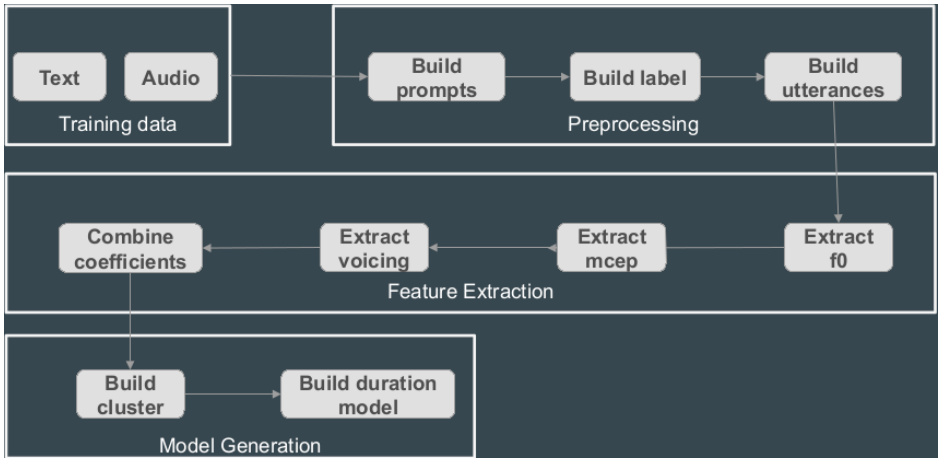


Figure 3.5: Festival General Flow

SO the basic steps involved in our speech synthesis system are:

- Training data
- Preprocessing

- Feature Extraction
- Model Generation

3.3.1 Training data

The first step in any system is to prepare the training data. As we are working on Urdu TTS, we requires recording of Urdu transcriptions from different speakers. For this purpose 2 male and 2 female speakers were used to record our data. Urdu transcriptions contained data related to routing and local area addresses. For example

- aap ki manzil aap ke baien taraf hai
- roundabout pe dasvin sarak lein
- bakhshupura gujraat

There were around 1600 lines where on average it took 4 seconds to record a line. So we had training data set of around 7 hours. All recordings were recorded at sample rate of 16000Hz. All recordings were numbered according to the there transcription number which allows us to easily map them. During recording, we made sure that speaker is not interrupted nor there should be any noise in the surrounding. This is done to make sure there is consistency while recording is going on. In below images, one can clearly see the difference between two recordings of the same sentence when recorded in noisy background 3.6 and quiet background 3.7.

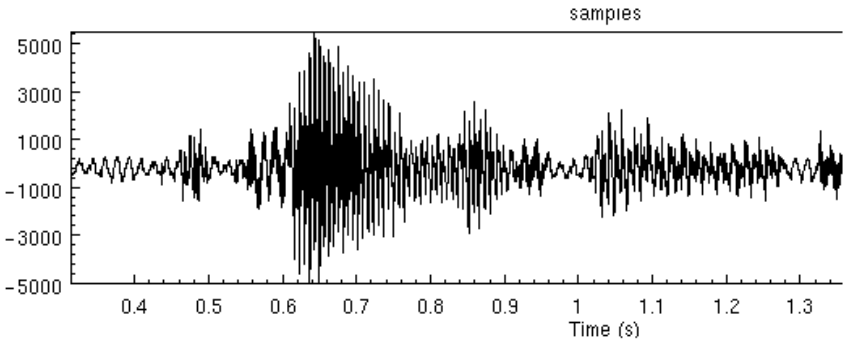


Figure 3.6: Recording with some noise in the background

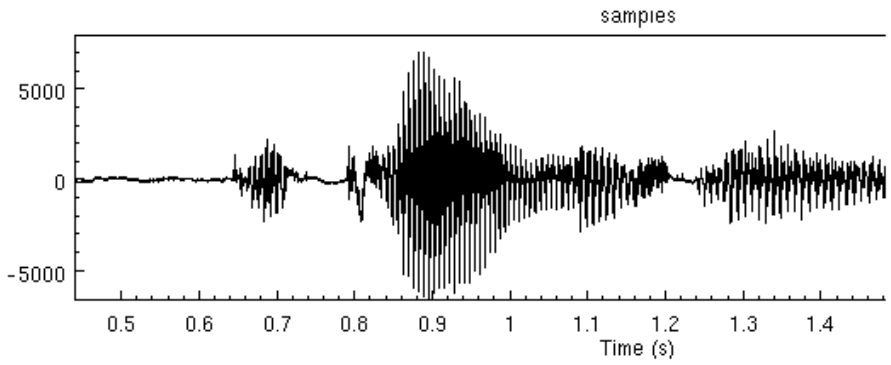


Figure 3.7: Recording without noise in the background

3.3.2 Preprocessing

Before we begin processing our data, we need to transform it into a format which is understandable by our system. First we will confirm that all the wave files are in correct format. For this we will run a simple script which will output our wavefiles in 16000Hz and in riff format. Second we need to remove extra silence from our recordings. This extra silence is either at the start of recording or at the end. We have to remove it as they are not required and only increases the duration of recordings. In image above 3.7, one can see that there is an unrequired part at the beginning of the recording. We will remove it to have only that part of the recording which is of use to us. For this purpose we will use Edinburgh speeh toolkit. This is done by extraction of fundamental frequency (f0). F0 can be used to estimate the beginning and ending of the speech. Using it we can eliminate extra seconds from our wave files.

Now our wave files have been set, we will move on to next step of preprocessing in which we will generate utterances. These are used to approximate phones for each sentences. These are calculated using Festival library. In 3.8, one can see text is broken down into its phonemes and then there relative positions in the wave files. For example the text "tractor workshop" is broken down to the phonemes as shown in the table 3.1. A lab file is also generated which only shows phonemes and there relative duration in the wave files 3.9.

```
EST_File utterance
DataType ascii
version 2
EST_Header_End
Features max_id 25 ; type Text ; iform "\"tractor workshop\"" ;
Stream_Items
1 id _1 ; name tractor ; whitespace "" ; prepunctuation "" ;
2 id _2 ; name workshop ; whitespace " " ; prepunctuation "" ;
3 id _4 ; name workshop ; pbreak BB ; pos nil ;
4 id _3 ; name tractor ; pbreak NB ; pos nil ;
5 id _5 ; name BB ;
6 id _6 ; name syl ; stress 0 ;
7 id _14 ; name syl ; stress 0 ;
8 id _22 ; name pau ; dur_factor 1 ; end 0.11 ;
9 id _7 ; name t ; dur_factor 1 ; end 0.22 ;
10 id _8 ; name r ; dur_factor 1 ; end 0.33 ;
11 id _9 ; name Y ; dur_factor 1 ; end 0.44 ;
12 id _10 ; name k ; dur_factor 1 ; end 0.55 ;
13 id _11 ; name t ; dur_factor 1 ; end 0.66 ;
14 id _12 ; name y ; dur_factor 1 ; end 0.77 ;
15 id _13 ; name r ; dur_factor 1 ; end 0.88 ;
16 id _15 ; name v ; dur_factor 1.5 ; end 1.045 ;
17 id _16 ; name y ; dur_factor 1.5 ; end 1.21 ;
18 id _17 ; name r ; dur_factor 1.5 ; end 1.375 ;
19 id _18 ; name k ; dur_factor 1.5 ; end 1.54 ;
20 id _19 ; name sh ; dur_factor 1.5 ; end 1.705 ;
21 id _20 ; name O ; dur_factor 1.5 ; end 1.87 ;
22 id _21 ; name p ; dur_factor 1.5 ; end 2.035 ;
23 id _23 ; name pau ; dur_factor 1 ; end 2.145 ;
24 id _25 ; f0 110 ; pos 2.035 ;
25 id _24 ; f0 130 ; pos 0.11 ;
End_of_Stream_Items
```

Figure 3.8: Example of an utterance file

Text	Phone
t	t
r	r
a	Y
c	k
t	t
o	y
r	r
w	v
o	y
r	r
k	k
sh	sh
o	O
p	p

Table 3.1: Phoneme distribution of "tracktor workshop"

```
0.1100 100 pau
0.2200 100 t
0.3300 100 r
0.4400 100 Y
0.5500 100 k
0.6600 100 t
0.7700 100 y
0.8800 100 r
1.0450 100 v
1.2100 100 y
1.3750 100 r
1.5400 100 k
1.7050 100 sh
1.8700 100 O
2.0350 100 p
2.1450 100 pau
```

Figure 3.9: Example of an lab file

3.3.3 Feature Extraction

After preprocessing the data, its time to extract useful features. Feature extraction involves following steps:

- Label generation
- Extract f0
- Extract mcep
- Extract voicing
- Combine coefficient

Label generation

We use EHMM labeler to label our data. It is provided and recommended by festvox as it follows there format as well. First it will extract phone sequences and save them in a text file 3.10. Then it will compute necessary features which are mel-frequency cepstrals. The resultant files are then converted to binary format. It then uses BaumWelch algorithm to find maximum likelihood of HMM's parameters. After that features are aligned and standardized.

```
0 pau s Y v y n ssil d i pau
10 pau h U m A ssil b l O k pau
100 pau b l O k ssil n A I n t i n pau
1000 pau n I u ssil ch A l i pau
1001 pau s t o r ssil b l O k pau
1003 pau b l O k ssil t U Y n t i pau
1004 pau f A I v ssil b Y h Y r I A pau
1005 pau b i ch ssil s t r i t pau
```

Figure 3.10: Example of phone sequences

Extract f0

In this step, we will first extract pitchmarks using Edinburgh speech tools and save them in .pm files. Using these pitchmarks, we will extract fundamental frequency of each wave files and store it in .f0 files. They are also known as excitation parameters.

Extract mcep

In this step, Mel Cepstral Coefficients are extracted from wave files. Tools to extract these are provided with Festvox, so no issue in this step as well. This step may take some time depending on the amount of data (number of files). The features are stored as .mcep files.

Extract voicing

This step is to detect pitch in each wave file. This is done using pda which produces fundamental frequency contours. The output is stored as .v files and contains data in the form of zeroes and ones.

Combine coefficient

In this step we combine above features in one file for each corresponding utterance we have. The data in each of corresponding files (.f0 , .mcep , .v) are extracted and pasted in new files. New files are stored in .mcep format.

3.3.4 Model Generation

After extracting features, we will create a parametric model. The model will be based on the features we extracted in the previous step. It involves following steps:

- Generate Statenames
- Generate Filters
- Clustering & Duration Model

Generate Statenames

This is a simple step in which we create a scheme file for statenames. These statenames are EHMM state labels. A subset of it is shown in Figure 3.11. It contains vowel and its corresponding three states. This step is just a prerequisite for model creation.

Generate Filters

In this step we create two types of filters. In our first type, we will split the frequency ranges into five categories which are as follows:

- Low pass filter at 500

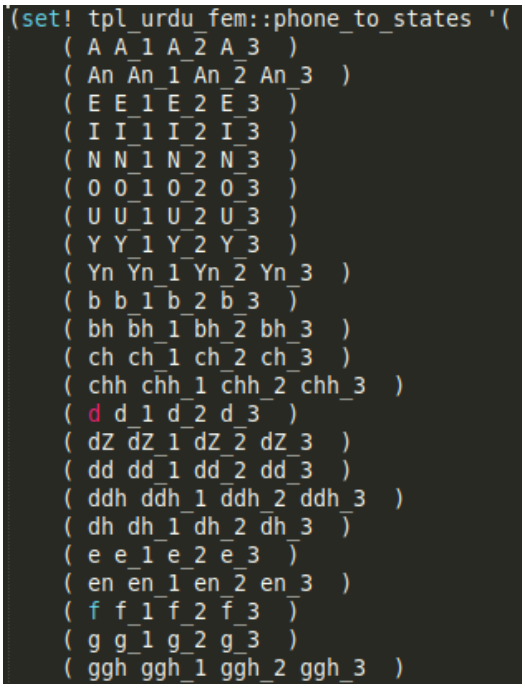


Figure 3.11: Example of Statename file

- Band pass filter between 900 and 1500
- Band pass filter between 2000 and 3750
- Band pass filter between 4000 and 6000
- Band pass filter between 6250 and 7500

And our second type of filter is a simple Low pass filter at 6000. These filters ranges are defined by University of Edinburgh.
Note: Low pass filter is a filter which only allows lower frequency (from the specified) to pass through and blocks remaining one. Whereas in Band pass filter, only the specified range is allowed to pass and remaining is blocked.

Clustering & Duration Model

This is the step in which we train our model. The method which we are using is called CLUSTERGEN. This method requires recording and there utterances which we have collected and preprocessed in previous steps. Clustering is done using Classification And Regression Tree (CART). CART tree is learned from labelled data which we have collected before. It is a supervised

learning process and it uses wagon system which University of Edinburgh has developed. It produces trees in LISP format which is desired format in FESTIVAL. CART tree is useful because it can predict both categorical values or continuous values. Categorical values are used for classification and continuous values are used for regression tree. Against categorical values, we will get probability distribution and against continuous values, we will get mean or standard deviation. Wagon requires data files from database which consists of vector. The first field of each vector determines whether to build a regression tree or classification tree. First we will build clusters using f0 and mcep features we calculated before and then we will find mean and standard deviation of duration of phones in our database. Then we will extract features from our utterances and save them in one data file.

It is a lengthy process and requires many features to train a tree. But for simplicity, I'll be using some features to explain how it works. Consider one of the statement from our database which is:

abu bakar

Each statement is divided into segments.For each segment we will have vectors in our data file. Each vector contains many properties but for simplicity lets say it uses 5 properties which are:

- name (phone)
- p.name (previous phone)
- n.name (next phone)
- pos_in_syl (position in word)
- seg_onsetcoda (segment type)

So for above statement we will have vector as shown in table 3.2. Lets consider the second row from the table 3.2 to explain these properties. The first property is the name of phoneme which is currently in focus in the segment which in this case is "y". The second property tells us the previous phoneme of the segment which here is "pau". Similarly third property tells us the next phoneme in the segment which is "b". Fourth property tells us the position of the phoneme in the word. Zero means first here. Fifth property tells us if it is a vowel or it comes before or after an vowel. Coda means it is vowel or it comes after vowel whereas onset means it comes before vowel.

Row	name	p.name	n.name	pos_in_syl	seg_onsetcoda
1	pau	0	y	0	coda
2	y	pau	b	0	onset
3	b	y	u	1	onset
4	u	b	b	2	coda
5	b	u	y	0	onset
6	y	b	k	1	onset
7	k	y	k	2	onset
8	k	k	y	3	onset
9	y	k	r	4	coda
10	r	y	pau	5	coda
11	pau	r	0	0	coda

Table 3.2: Simplified vector example"

Similarly there is a list of properties which is used for training our model. We get these properties from the previous steps we have used so far. Using such data file, we can train our model.

3.4 FLITE based model

Once our model in Festival is ready, it is time to export it to flite so that it can be used on mobile devices. Flite is designed as a run time engine written in ANSI C. Any working voice in FESTIVAL can be converted to flite. Voice can't be build in flite; it can only be converted from festival voice. In order for our voice to work on mobile devices, we need following conversions:

- Language conversion
- Lexicon conversion
- Voice conversion

3.4.1 Language conversion

This is one of the simple steps. We just need to convert our phone-set from scheme format to C. A simple script separates phones and its features and stores them in array form. For example figure 3.12 shows our phone-set in LISP format. Here "tpl_urdu"is our phoneset name. Followed by phone features (clst, vc, vlng, vheight, vfront, vrnd, ctype, cplace, cvox, asp, nuk) and there possible values. After that is the vector of phones with there feature

value described above.

```
(defPhoneSet
 tpl_urdu
 ;;;  Phone Features
 (;; vowel or consonant
  (clst + - 0)
  (vc + - 0)
  ;; vowel length: short long nasalized dipthong schwa
  (vlng s l d a 0)
  ;; vowel height: high mid low
  (vheight 1 2 3 0 -)
  ;; vowel frontness: front mid back
  (vfront 1 2 3 0 -)
  ;; lip rounding
  (vrnd + - 0)
  ;; consonant type: stop fricative affricative nasal liquid approximant
  (ctype s f a n l r 0)
  ;; place of articulation: labial alveolar palatal
  ;; labio-dental dental velar glottal
  (cplace l a p b d v g 0)
  ;; consonant voicing
  (cvox + - 0) ; voiced or voiceless
  (asp  + - 0) ; aspiration
  (nuk + - 0) ;
  )
 (
  (    pau    -   -   0   0   0   0   0   0   -   -   -   )
  (    A      -   +   l   3   3   -   0   0   -   -   -   )
  (    b      -   -   0   0   0   0   s   l   +   -   -   )
  (    bh     -   -   0   0   0   0   s   l   +   +   -   )
  (    ch     -   -   0   0   0   0   s   p   -   -   -   )
  (    chh    -   -   0   0   0   0   s   p   -   +   -   )
  (    d      -   -   0   0   0   0   s   a   +   -   -   )
  (    dh     -   -   0   0   0   0   s   a   +   +   -   )
  (    dd     -   -   0   0   0   0   f   d   +   -   -   )
  (    ddh    -   -   0   0   0   0   f   d   +   -   -   )
  (    e      -   +   l   2   2   -   0   0   -   -   -   )
  (    E      -   +   s   2   1   -   0   0   -   -   -   )
  (    f      -   -   0   0   0   0   f   b   -   -   -   )
  (    ggh    -   -   0   0   0   0   s   g   +   -   -   )
  (    g      -   -   0   0   0   0   s   v   +   -   -   )
  (    gh     -   -   0   0   0   0   s   v   +   +   -   )
  (    h      -   -   0   0   0   0   f   g   -   +   -   )
  (    i      -   +   l   1   1   -   0   0   -   -   -   )
  (    I      -   +   s   1   1   -   0   0   -   -   -   )
  (    dZ     -   -   0   0   0   0   a   a   +   -   -   )
  (    jh     -   -   0   0   0   0   a   a   +   +   -   )
  (    k      -   -   0   0   0   0   s   v   -   -   -   )
  (    kkh    -   -   0   0   0   0   s   g   -   -   -   )
```

Figure 3.12: Definition of Phones in LISP

Now the conversion script will separate phone names and features name from it and make them separate arrays as shown in figure 3.13. Similarly it will make separate arrays corresponding to each phones to store its feature values as shown in figure 3.14. Feature values and phoneset are linked by there indexes.

```
static const char * const tpl_urdu_featnames[] = {
 "clst", "vc", "vlng", "vheight", "vfront", "vrnd", "ctype", "cplace", "cvox", "asp", "nuk", NULL };

static const char * const tpl_urdu_phonenames[] = {
 "pau", "A", "b", "bh", "ch", "chh", "d", "dh", "dd", "ddh", "e", "E", "f", "ggh", "g", "gh", "h", "i", "I",
 "dZ", "jh", "k", "kkh", "kh", "l", "ll", "m", "n", "N", "o", "O", "p", "ph", "q", "r", "rh", "s", "sh", "t",
 "tt", "tth", "th", "u", "U", "v", "y", "Y", "yy", "z", "in", "en", "Yn", "un", "on", "On", "An", NULL };
```

Figure 3.13: Phones and features arrays in C

```
static const int tpl_urdu_fv_000[] = { 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_001[] = { 0, 2, 3, 4, 4, 0, 1, 1, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_002[] = { 0, 0, 1, 1, 1, 1, 5, 3, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_003[] = { 0, 0, 1, 1, 1, 1, 5, 3, 2, 2, 0,   -1 };
static const int tpl_urdu_fv_004[] = { 0, 0, 1, 1, 1, 1, 5, 6, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_005[] = { 0, 0, 1, 1, 1, 1, 5, 6, 0, 2, 0,   -1 };
static const int tpl_urdu_fv_006[] = { 0, 0, 1, 1, 1, 1, 5, 7, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_007[] = { 0, 0, 1, 1, 1, 1, 5, 7, 2, 2, 0,   -1 };
static const int tpl_urdu_fv_008[] = { 0, 0, 1, 1, 1, 1, 8, 9, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_009[] = { 0, 0, 1, 1, 1, 1, 8, 9, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_010[] = { 0, 2, 3, 10, 10, 0, 1, 1, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_011[] = { 0, 2, 5, 10, 11, 0, 1, 1, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_012[] = { 0, 0, 1, 1, 1, 1, 8, 12, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_013[] = { 0, 0, 1, 1, 1, 1, 5, 13, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_014[] = { 0, 0, 1, 1, 1, 1, 5, 14, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_015[] = { 0, 0, 1, 1, 1, 1, 5, 14, 2, 2, 0,   -1 };
static const int tpl_urdu_fv_016[] = { 0, 0, 1, 1, 1, 1, 8, 13, 0, 2, 0,   -1 };
static const int tpl_urdu_fv_017[] = { 0, 2, 3, 11, 11, 0, 1, 1, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_018[] = { 0, 2, 5, 11, 11, 0, 1, 1, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_019[] = { 0, 0, 1, 1, 1, 1, 7, 7, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_020[] = { 0, 0, 1, 1, 1, 1, 7, 7, 2, 2, 0,   -1 };
static const int tpl_urdu_fv_021[] = { 0, 0, 1, 1, 1, 1, 5, 14, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_022[] = { 0, 0, 1, 1, 1, 1, 5, 13, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_023[] = { 0, 0, 1, 1, 1, 1, 8, 14, 0, 2, 0,   -1 };
static const int tpl_urdu_fv_024[] = { 0, 0, 1, 1, 1, 1, 15, 6, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_025[] = { 0, 0, 1, 1, 1, 1, 15, 6, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_026[] = { 0, 0, 1, 1, 1, 1, 16, 3, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_027[] = { 0, 0, 1, 1, 1, 1, 16, 7, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_028[] = { 0, 0, 1, 1, 1, 1, 16, 14, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_029[] = { 0, 2, 3, 10, 4, 2, 1, 1, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_030[] = { 0, 2, 3, 4, 4, 2, 1, 1, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_031[] = { 0, 0, 1, 1, 1, 1, 5, 3, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_032[] = { 0, 0, 1, 1, 1, 1, 5, 3, 0, 2, 0,   -1 };
static const int tpl_urdu_fv_033[] = { 0, 0, 1, 1, 1, 1, 5, 14, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_034[] = { 0, 0, 1, 1, 1, 1, 3, 7, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_035[] = { 0, 0, 1, 1, 1, 1, 3, 6, 2, 0, 0,   -1 };
static const int tpl_urdu_fv_036[] = { 0, 0, 1, 1, 1, 1, 8, 7, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_037[] = { 0, 0, 1, 1, 1, 1, 8, 6, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_038[] = { 0, 0, 1, 1, 1, 1, 5, 7, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_039[] = { 0, 0, 1, 1, 1, 1, 5, 9, 0, 0, 0,   -1 };
static const int tpl_urdu_fv_040[] = { 0, 0, 1, 1, 1, 1, 5, 9, 0, 2, 0,   -1 };
```

Figure 3.14: Phones and there features values in C

3.4.2 Lexicon conversion

Here we need to convert our lexicon file and letter to sound(lts) rules to C. Lets start with first conversion of lexicon file. First total number of entries are counted and stored in a variable. Then phones used in lexicon file are stored in an array. All lexicon entries are then indexed and stored in a separate file. The number of bytes which our lexicon take is also noted down to allocate memory in future. This is done using flite's tool with following code line:

```
$FESTIVAL --heap 10000000 -b $FLITEDIR/tools/make_lex.scm
'(lextoC "tpl_urdu" "tpl_urdu_lex.out" "c")' ;
```

Now we will convert lts rules. Here we need to convert decision trees of lts rules to a format understood by flite. We will first convert the LTS trees to regular grammer and then build weighted finite state transducers (wfst). Then we will convert wfst in to C compilation structure. This is done using flite's tool with following code line:

```
$FESTIVAL --heap 10000000 -b $FLITEDIR/tools/make_lts_wfst.scm
lts_scratch/lex_lts_rules.scm
'(lts_to_rg_to_wfst lex_lts_rules "wfst/")';

$FESTIVAL --heap 10000000 -b $FLITEDIR/tools/make_lts.scm
```

```
lts_scratch/lex_lts_rules.scm
'(ltsregextoC "'$LEXNAME'" lex_lts_rules "wfst/" "c")';
```

3.4.3 Voice conversion

This is an automatic process where FESTIVAL has provided the script to easily convert FESTIVAL voice into FlITE voice using Language and Lexicon file we converted in previous steps. We just need to use "build_flite"for conversion. In our FESTIVAL voice directory we need to run following code:

```
./bin/build_flite cg
cd flite
make
```

This will result in our flite voice named as flite_tpl_urdu_female3. After renaming to match flite android app format, it can be used in the app. The naming format which android needs is as follows:

"female;female3.cg.flitevox"

Where female is the gender, female3 is the voice name, cg tells its clustergen voice and flitevox is the format. Apart from this, Language and Lexicon files are also linked with flite app. Without linking the app will either crash or load default files.

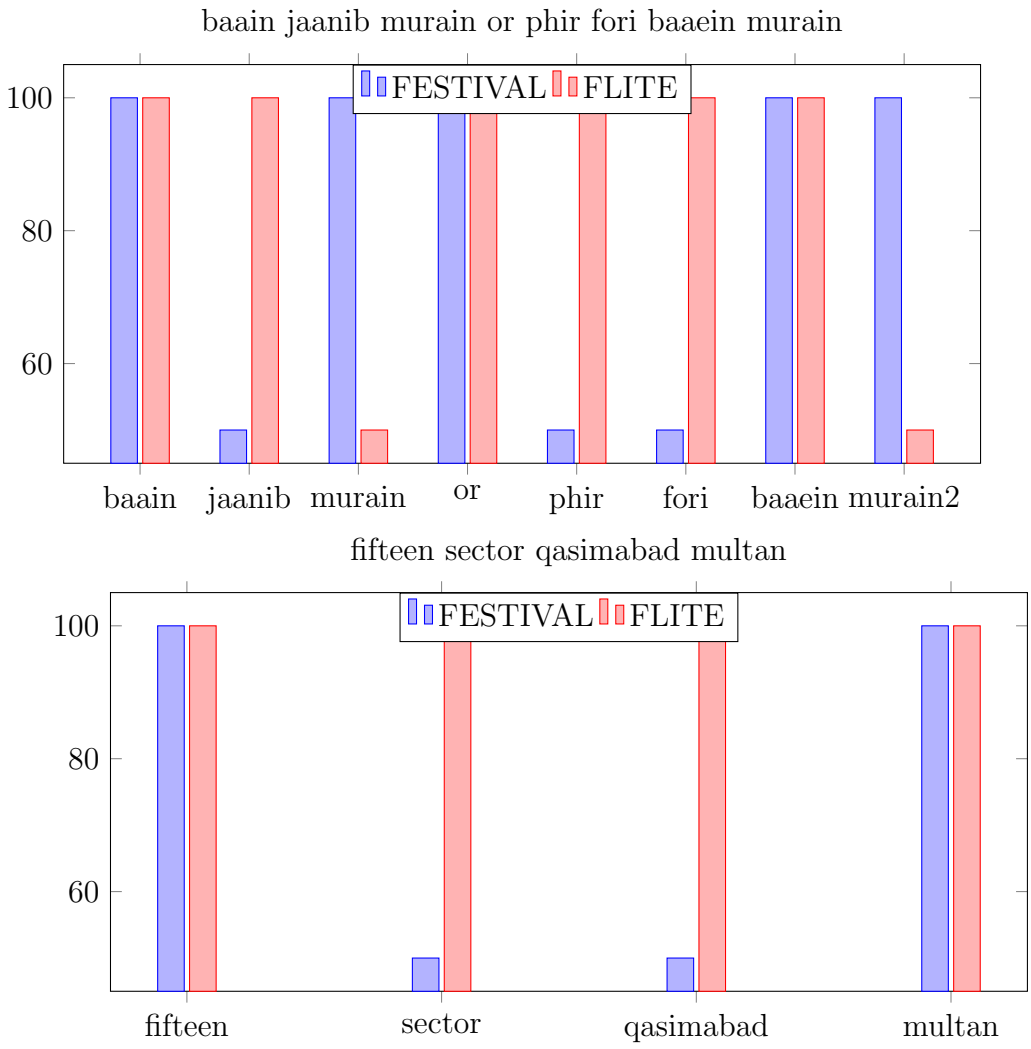
Chapter 4

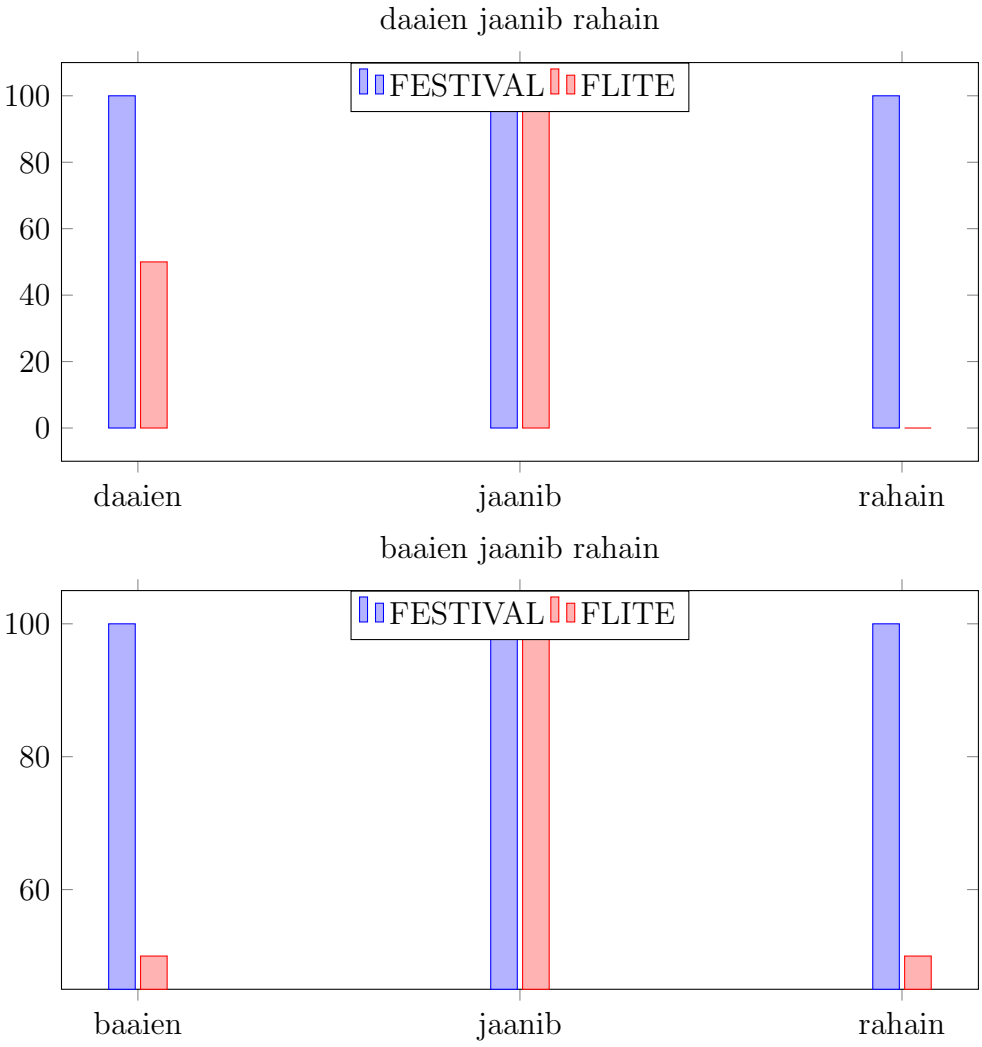
Implementation and Results

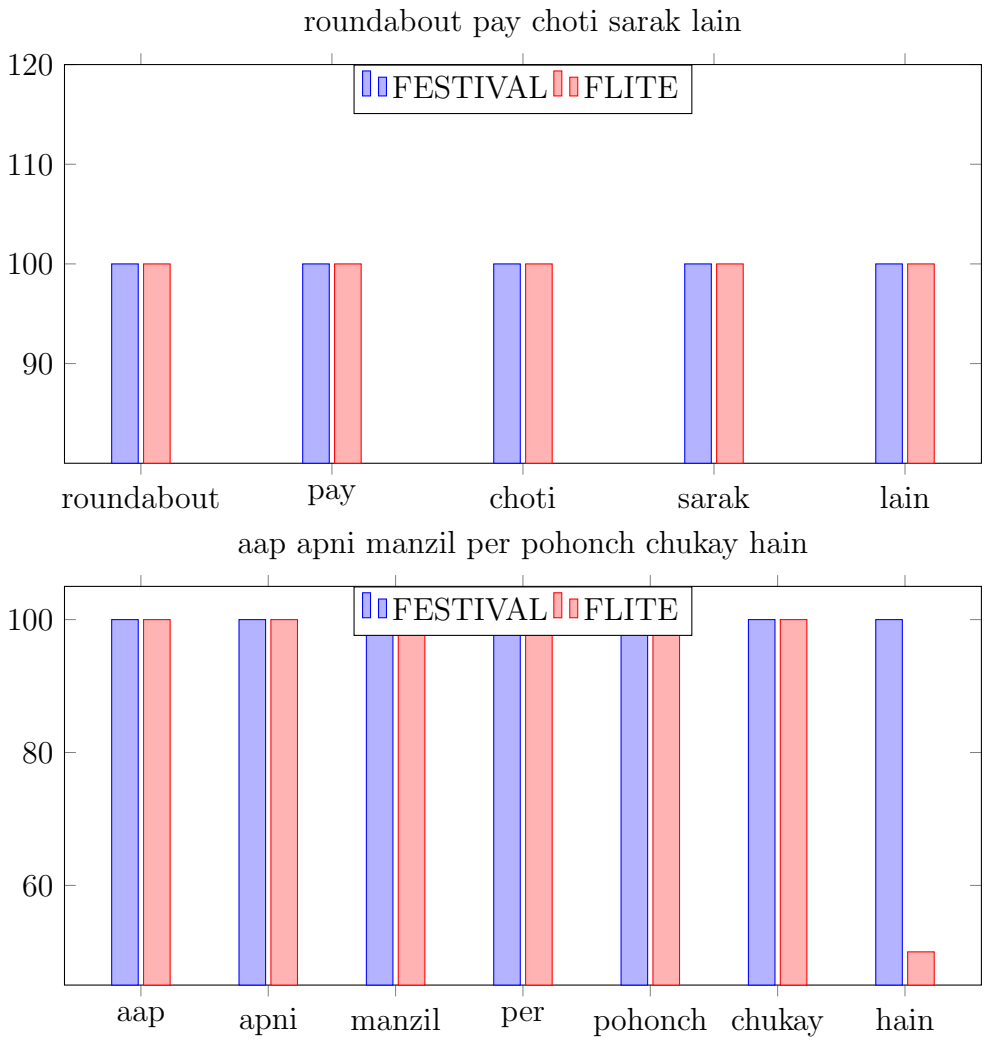
I am using the same testing method as used by (Shahid Kh. S. and Haq., 2016) i.e. both through survey and through our own speech to text system. So in first phase, to check the accuracy of our system, we had to ask different people to listen to our results and give us feedback. The idea of the feedback is to see whether the general public understand our TTS's result or not. Also to find any shortcomings in it for future updates. Here lets discuss six random sentences which we played to users and took there feedback regarding the understanding of each word in the sentence. The sentences were:

- baain, jaanib murain or phir fori baaein murain
- fifteen sector qasimabad multan
- daaien, jaanab rahain
- baain, jaanab rahain
- roundabout pay chothi sarak lein
- aap apni manzil per pohonch chukay hain

The audios were of both the Festival and Flite voice so that we could also compare them. Lets plot the results to better understand them

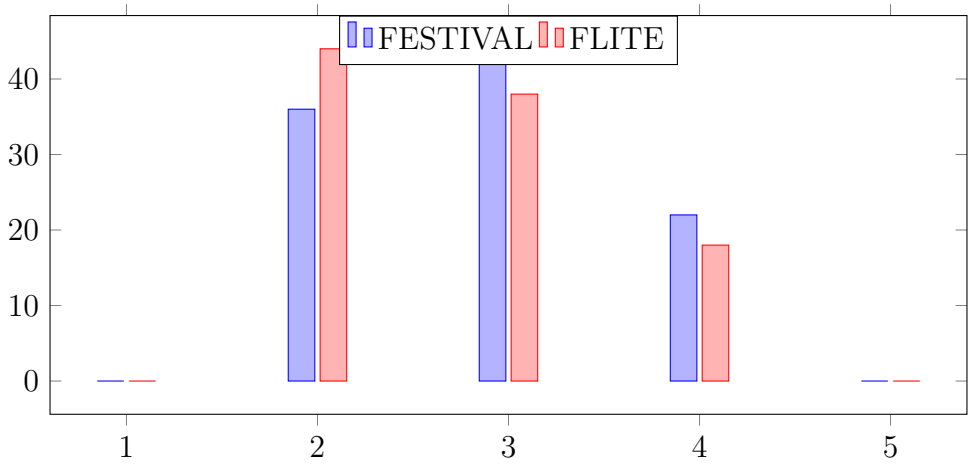

baain jaanib murain or phir fori baaein murain
FESTIVAL
FLITE
100
80
60
baain
jaanib
murain
or
phir
fori
baaein
murain2
fifteen sector qasimabad multan
FESTIVAL
FLITE
100
80
60
fifteen
sector
qasimabad
multan


daaien jaanib rahain
FESTIVAL
FLITE
100
80
60
40
20
0
daaien
jaanib
rahain
baaien jaanib rahain
FESTIVAL
FLITE
100
80
60
baaien
jaanib
rahain



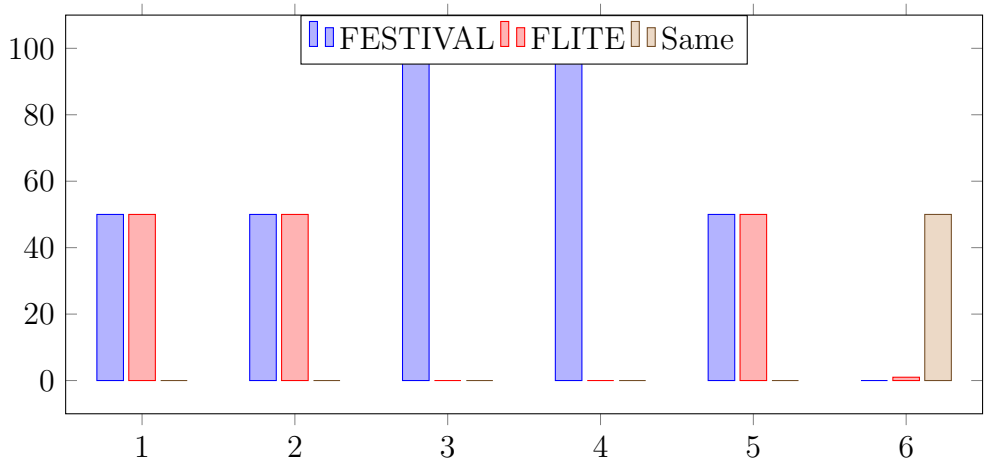
We also asked the users to tell us how natural or robotic the sound feels. For this we gave them scale from 1-5 to rate each voice. Summary for that is:

naturalness factor with 1 being lowest and 5 being highest



We also asked the users, which voice in there opinion is better or are they of same quality. The results for this are as follows:

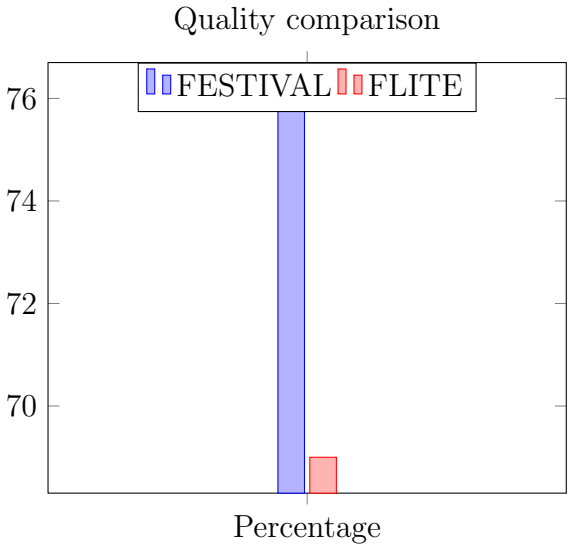
Quality comparison



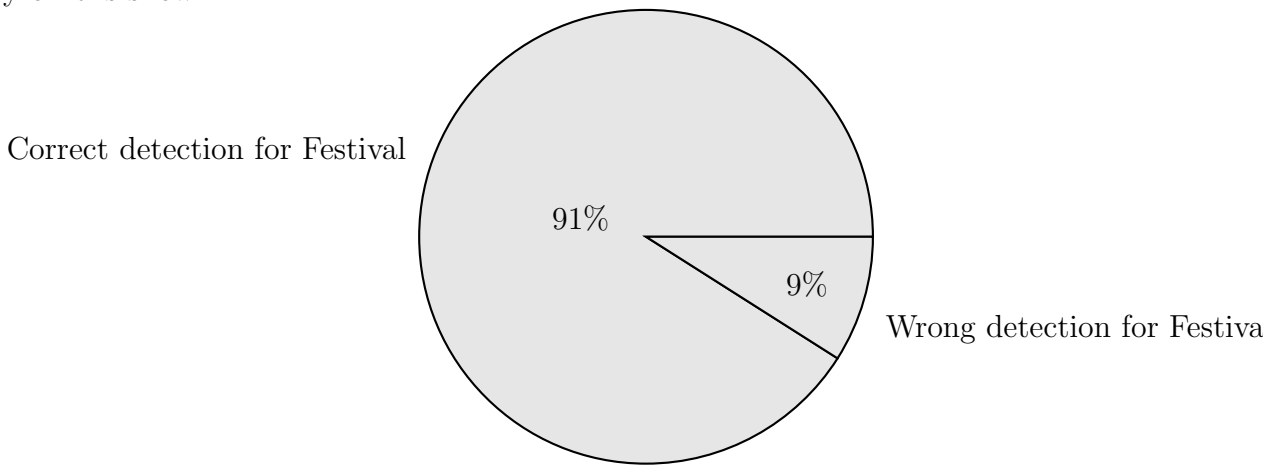
Number of words correctly understood by users is shown in 4.2

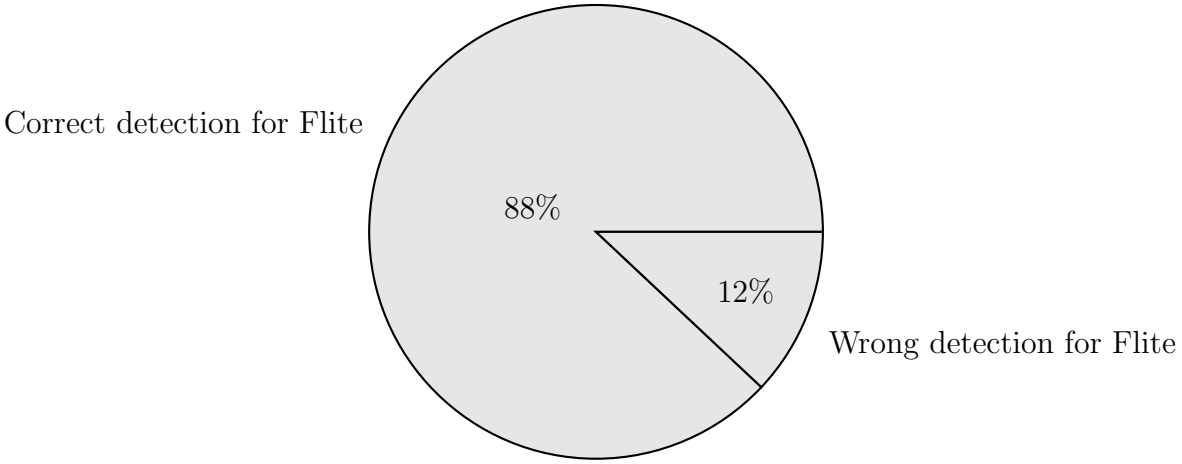
Voice	**Total number of words**	**Correct number of words**
Festival	60	55
Flite	60	53

Table 4.1: Summary of result



In second phase, we played the synthesized audio in front of a speech recognition program. Speech recognition program for the same database has already been developed by my supervisor Dr. Ali Tahir and his students. The summary of it is shown in





So overall accuracy of our system is given in table

Voice	**Manual Testing**	**Automatic Testing**
Festival	91.37	76
Flite	88.33	69

Table 4.2: Summary of result

We compared our results with CLE HTS voice as shown in table 4.3. We are only comparing our festival result with them as they haven't developed TTS for mobile.

Voice	**Our**	**CLE blind**	**CLE un-blind**
Festival	91	81	96

Table 4.3: Summary of result

Chapter 5

Conclusion and Future Work

5.1 Conclusion

We were successful in making a Urdu TTS for navigation app. It achieved an accuracy of 91% on FESTIVAL and 88% on FLITE which is very good considering no such previous work was done in this domain. Also the accuracy is good considering that we used Roman Urdu instead of traditional Nastaliq style. The accuracy on FLITE was less than that of FESTIVAL due to the fact that its model was derived and converted from FESTIVAL and files were compressed to make it a lighter version so that it could be used in mobile devices. CLE's accuracy is little higher than us in unblind test as they prepared everything in Nastaliq style.
Also we couldn't use Wavenet due to the fact that it was computationally expensive. Training took alot of times i.e. in weeks. Plus the synthesis also took alot of time i.e. 15 mins to generate for 10 words sentence or 5 mins when using Fast wavenet.

5.2 Future Work

For future, we need to improve it as there were some words which the users were unable to understand and also to work on its naturalness. Users complained that although they understood most of the words, they didn't sound natural enough. So works need to be done on this regards. Also work is required at FLITE side to make it sound as cleaner as possible with less or no loss. After some modification and improvement in it, this will become part of TPL Maps navigation system.
Furthermore, our system currently supports Urdu numbers from 0-9. That is it will only synthesize these number. Numbers greater than 9 are synthe-

size in parts. For example twelve (12) will be synthesized as aik(1) do(2). So in future we plan to first include numbers till 100 as from 0-100. The reason for only including till hundred is that each of these number has a unique name. After that, repetition starts so we will write rules to deal with numbers greater than 100.

Appendix A

Phoneme Properties

Table A.1 and Table A.2 show all the phonemes with there properties which we have used in development of our TTS system. These properties are:

- vc to define vowel or consonant
- vlng to define vowel length
- vheight to define vowel height
- vfront to define vowel frontness
- vrnd to define lip rounding
- ctype to define consonat type
- cplace to define place of articulation
- cvox to define consonant voicing
- asp to define aspiration

You would notice that each phoneme has either properties of vowel or consonant not both. If a phoneme is a vowel than its consonant properties would be zero and if its a consonant, its vowel properties would be zero.

Phone	**vc**	**vlng**	**vheight**	**vfront**	**vrnd**	**ctype**	**cplace**	**cvox**	**asp**
pau	-	0	0	0	0	0	0	-	-
A	+	l	3	3	-	0	0	-	-
b	-	0	0	0	0	s	l	+	-
bh	-	0	0	0	0	s	l	+	+
ch	-	0	0	0	0	s	p	-	-
chh	-	0	0	0	0	s	p	-	+
d	-	0	0	0	0	s	a	+	-
dh	-	0	0	0	0	s	a	+	+
dd	-	0	0	0	0	f	d	+	-
ddh	-	0	0	0	0	f	d	+	-
e	+	l	2	2	-	0	0	-	-
E	+	s	2	1	-	0	0	-	-
f	-	0	0	0	0	f	b	-	-
ggh	-	0	0	0	0	s	g	+	-
g	-	0	0	0	0	s	v	+	-
gh	-	0	0	0	0	s	v	+	+
h	-	0	0	0	0	f	g	-	+
i	+	l	1	1	-	0	0	-	-
I	+	s	1	1	-	0	0	-	-
dZ	-	0	0	0	0	a	a	+	-
jh	-	0	0	0	0	a	a	+	+
k	-	0	0	0	0	s	v	-	-
kkh	-	0	0	0	0	s	g	-	-
kh	-	0	0	0	0	f	v	-	+
l	-	0	0	0	0	r	p	+	-
ll	-	0	0	0	0	r	p	+	-
m	-	0	0	0	0	n	l	+	-
n	-	0	0	0	0	n	a	+	-
N	-	0	0	0	0	n	v	+	-
o	+	l	2	3	+	0	0	-	-
O	+	l	3	3	+	0	0	-	-

Table A.1: Phoneme with there properties part 1

Phone	**vc**	**vlng**	**vheight**	**vfront**	**vrnd**	**ctype**	**cplace**	**cvox**	**asp**
p	-	0	0	0	0	s	l	-	-
ph	-	0	0	0	0	s	l	-	+
q	-	0	0	0	0	s	v	-	-
r	-	0	0	0	0	l	a	+	-
rh	-	0	0	0	0	l	p	+	-
s	-	0	0	0	0	f	a	-	-
sh	-	0	0	0	0	f	p	-	-
t	-	0	0	0	0	s	a	-	-
tt	-	0	0	0	0	s	d	-	-
tth	-	0	0	0	0	s	d	-	+
th	-	0	0	0	0	s	a	-	+
u	+	l	2	3	+	0	0	-	-
U	+	s	3	3	+	0	0	-	-
v	-	0	0	0	0	f	b	+	-
y	+	s	3	1	-	0	0	-	-
Y	+	s	3	1	-	0	0	-	-
yy	-	0	0	0	0	f	p	+	-
z	-	0	0	0	0	f	a	+	-
in	+	l	1	1	-	n	0	-	-
en	+	l	2	1	-	n	0	-	-
Yn	+	l	3	1	-	n	0	-	-
un	+	l	1	3	+	n	0	-	-
on	+	l	3	3	+	n	0	-	-
On	+	l	3	3	+	n	0	-	-
An	+	l	3	3	-	n	0	-	-

Table A.2: Phoneme with there properties part 2

Appendix B

Phoneme and Urdu Characters

Figure B.1 and B.2 shows phoneme we used and there corresponding characters in Urdu. You may be wondering that why there is more than one phoneme corresponding to same urdu character. The reason is that although they are corresponding to same urdu character, there properties is different. We speak urdu characters in different ways. For example we ,usually speak in regular way but some time we speak using our nose. There are other cases where we use different part of our mouth to pronounce a word. So to differentiate, we have different phonemes corresponding to same Urdu characters.

Phone	Urdu character
A	ا
b	ب
bh	بھ
ch	چ
chh	چھ
d	د
dh	دھ
dd	ڈ
ddh	ڈھ
e	ی
E	ے
f	ف
ggh	غ
g	گ
gh	گھ
h	ہ
i	ی
I	ے
dZ	ج
jh	جھ
k	ک
kkh	کھ
kh	کھ
l	ل
ll	ل
m	م
n	ن
N	ن

Figure B.1: Phoneme with there corresponding urdu characters part 1

Phone	Urdu character
o	و
O	و
p	پ
ph	پھ
q	ق
r	ر
rh	رہ
s	س
sh	ش
t	ت
tt	ٹ
tth	ڈھ
th	تھ
u	ا
U	ا
v	و
y	ی
Y	ی
yy	ی
z	ز
in	ں
en	ں
Yn	ں
un	وں
on	وں
On	وں
An	اں

Figure B.2: Phoneme with there corresponding urdu characters part 2

Bibliography

Adeba F., Hussain S., H. T. H. E. and K.S., S. (2016). Comparison of urdu text to speech synthesis using unit selection and HMM based techniques. *19th Oriental COCOSDA Conference 2016, Bali, Indonesia.*

Basit, H. R. and S., H. (2014). Text processing for urdu TTS system. *Language and Technology Conference, Pakistan.*

Black, A. W. (2006). Clustergen: a statistical parametric synthesizer using trajectory modeling. In *INTERSPEECH.*

Hussain, S. (2004). Letter to sound rules for urdu text to speech sytem. *Workshop on Computational Approaches to Arabic Script-based Languages, Geneva, Switzerland.*

Hussain, S. (2005). Phonological processing for urdu text to speech system. *Contemporary Issues in Nepalese Linguistics, Linguistics Society of Nepal.*

Ijaz M., H. S. (2007). Corpus based urdu lexicon development. *Conference on Language Technology (CLT07), University of Peshawar, Pakistan.*

Kabir H., Raza S., S. A. H. S. (2002). Natural language processing for urdu TTS system. *IEEE International Multi-Topic Conference.*

M., F. and B., M. (2016). Urdu phonological rules in connected speech. *Conference on Language and Technology 2016 (CLT 16), Lahore, Pakistan.*

Mumtaz B., Urooj S., H. S. and W., H. (2015). Stress annotated urdu speech corpus to build female voice for TTS. *18th Oriental COCOSDA/CASLRE Conference, Shanghai, China.*

O., N. and T., H. (2014). Hidden markov model (HMM) based speech synthesis for urdu language. *Conference on Language and Technology 2014 (CLT14), Karachi, Pakistan.*

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499.*

Oord, A. v. d., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Driessche, G. v. d., Lockhart, E., Cobo, L. C., Stimberg, F., et al. (2017). Parallel wavenet: Fast high-fidelity speech synthesis. *arXiv preprint arXiv:1711.10433.*

Qasim M., Rauf S., H. T. and S., H. (2016). Urdu speech corpus for travel domain. *19th Oriental COCOSDA Conference 2016, Bali, Indonesia.*

R., B. and B., M. (2016). Identification of diphthongs in urdu and their acoustic properties. *Conference on Language and Technology 2016 (CLT 16), Lahore, Pakistan.*

Shahid Kh. S., Habib T., M. B. A. F. and Haq., E. U. (2016). Subjective testing of urdu text-to-speech (TTS) system. *Conference on Language and Technology 2016 (CLT 16), Lahore, Pakistan.*

Tokuda, Keiichi, H. Z. and Black., A. W. (2002). An HMM-based speech synthesis system applied to english. *IEEE Speech Synthesis Workshop.*

Zen, H., Nose, T., Yamagishi, J., Sako, S., Masuko, T., Black, A. W., and Tokuda, K. (2007). The HMM-based speech synthesis system (HTS) version 2.0.

Zen, H., Tokuda, K., and Black, A. W. (2009). Statistical parametric speech synthesis. *speech communication*, 51(11):1039–1064.