# Festival Based Maiden TTS System for Tamil Language

**Sreekanth Majji**[†], **Ramakrishnan A. G.**[*]

Electrical Engineering Department
Indian Institute of Science, Bangalore, India
[*]ramkiag@ee.iisc.ernet.in
[†]m.sree4u@gmail.com

**Abstract**

In this paper we describe the first Text To Speech (TTS) system for the Tamil language based on Festival architecture. We discuss practical implementation details and we have created and made use of an unit selection database using a phonetically rich Tamil speech corpus with 1027 sentences. The paper gives comprehensive details about the various modules in the TTS developed using Festival. In other words, we deal with how to add a new language to the existing set of languages in Festival, keeping the mathematical detail at minimum. The prosody models supplied by Festival have been incorporated into TTS. An optimal coupling technique for joining the units has been used for concatenation and waveform generation.

## 1. Introduction

Voice User Interfaces for IT applications and services have become more and more prevalent for languages like English, and are valued for ease of access, especially in telephony-based applications. In a country like India, where there is a rich multitude of languages and relatively lower literacy rates, local language speech interfaces can provide access to IT applications and services, through internet and/or telephone, to the masses. Given the increased availability of digital content in local languages, there is a real demand for speech synthesis systems for Indian languages.

Many commercial and laboratory TTS systems exist today, producing speech of varying quality. Festival speech synthesis (FSS) system is one such system, developed at University of Edinburgh by Alan W Black et al. Festival provides an environment for both speech synthesis research and development. Festival can also be used as a fully usable TTS system. The existing FSS system has inbuilt American-English dialect, British-English dialect and Spanish dialect language models. In the process of building a Tamil TTS, we also build a language model for Tamil and integrate it with Festival for its portable use.

## 2. Goal of the paper

The primary motivation of this work is to make a technological contribution that will form part of facilitating blind people to read printed material in their own mother tongue (to begin with, Tamil) from a book.

The fact that no good, high-quality TTS systems exist for any Indian language, is also a motivating factor for our endeavor. Recently, HP Labs India has developed a high quality TTS system for Hindi. We plan to build a high quality TTS system in a south Indian language for our laboratory research. We have created a phonetically rich Tamil speech corpus by selecting 1027 sentences from a huge Tamil text corpus, and developed a Tamil TTS system. In this project, we also aim to understand the processes involved in the development of TTS for any Indian language. This will aid us in the future development of TTS in other Indian languages.

Though work has been reported on text-to-speech synthesis in Indian languages, they mostly use diphone/polyphone databases. Unit selection databases contain multiple instances of different units, and hence cater for the variability of speech in different contexts. So, unit selection synthesizers have a better quality speech output as compared to diphone or other concatenative type synthesizers. Since Festival has a ready to use unit selection module, we build a Tamil voice from a large speech corpus, and proceed with the waveform synthesis by concatenation.

## 3. Building a Tamil Voice in Festival

This section describes the training stage, where we use a labelled database to make a system capable of producing a speech output. Training results in arrangement of the units extracted from the database in a format from which the units can be selected meaningfully and with ease. As soon as the database is recorded and properly segmented and labelled, training is carried out once for all. In Festival this process is termed as *building a voice.*

For building a voice model, we use a single speaker data corpus of 1027 sentences. A native Tamil speaker's voice was recorded in noise free conditions and sampled at 16 kHz. The corpus was manually segmented and labelled using Praat software tool. We create a phoneset for Tamil language which is required by the different sub-systems of Festival

### 3.1. Text Analysis

The text sentences are written in Tamil using a software tool iLEAP. iLEAP exports the Tamil text into the standard ISCII format. We have developed a G2P (grapheme to phoneme converter) tool for phonetically transcripting the ISCII text. The G2P tool uses some syntax and letter to sound (LTS) rules for converting the ISCII format into Tamil phonetic format. There is no text normalization done by G2P and text normalization has to be done separately. In our work, we use normalized text for training and testing purposes.

### 3.2. Utterance Structure

The utterance structure lies at the heart of Festival. Festival's basic object for synthesis is utterance, which represents some chunk of text that is to be rendered as speech. In general, the process of text to speech is to take an utterance which contains a simple string of characters and convert it step by step, filling out the utterance structure with more information until a waveform is built that says what the text contains (Black, 2002).

The following information is essential for creating an utterance structure

**Segment Information** For our Tamil TTS, we have created a manually segmented and labelled speech corpus. Segmental boundary is obtained directly from this information.

**Syllable Information** We syllabify the segmented labels into VC, CV, CVC, CCVC, CVCC combinations using a scheme script. At every instance of pause (#) character, we begin or break the text into a new syllable. We actually require a linguistic syllabification module.

**Word Information** The database used has word boundaries marked with the word limiters (#) in the text sentences. By this, the words in each of the sentences, along with their timings, have been extracted.

**Intonation Events** Intonation events describe the fundamental frequency (F0) contour in a speech waveform. We quantize the intonation in terms of peak F0, positive tilt (slope towards the peak), and negative tilt (slope from the peak) features.

**Target-F0** Target-F0 describes the F0 values for a segment (phoneme). We calculate F0 values at the start, mid and end point of the phoneme unit. The pitch is detected using the builtin function (pda) in EST library. We use the maximum peak F0 value as the Target-F0 feature. Other alternative is to take average F0 value as the value of Target-F0.

### 3.3. Clustering the Units

Festival uses cluster based unit selection, which requires that we cluster the units in the database. The basic approach (Alan W Black, 1997) is to cluster units within a unit type (i.e. a particular phone) based on questions concerning prosodic and phonetic context. Specifically, these questions relate to information that can be produced by the linguistic component, e.g. is the unit phrase-final, or is the unit in a stressed syllable. Thus for each phone in the database, a decision tree is constructed, whose leaves are a list of database units that are best identified by the questions which lead to that leaf.

To cluster the units in the database, we first define an acoustic measure to quantify the distance between two units of the same phone type. We use an acoustic vector which comprises Mel frequency cepstrum coefficients, F0, power, and delta cepstrum. The acoustic distance between two units is simply the average distance for the vectors of all the frames in the units plus $x\%$ of the frames in the previous units, which helps ensure that close units will have similar preceding contexts. More formally, we use a weighted Mahalanobis distance metric to define the acoustic distance $A_{dist}(U, V)$ between two units $U$ and $V$ of the same phoneme class as:

$$if \quad |V| > |U|$$

$$A_{dist}(U,V) = \frac{W_d * |U|}{|V|} * \sum_{i=1}^{|U|} \sum_{j=1}^{n} \frac{W_j(abs(F_{ij}(U) - F_{(i*|V|/|U|)j}(V)))}{SD_j * n * |U|} \tag{1}$$

where,
$|V|$ = No. of frames in $V$
$|U|$ = No. of frames in $U$
$F_{ij}(U)$ = Parameter $j$ of frame $i$ for unit $U$
$W_j$ = Weight factor for parameter $j$
$SD_j$ = Standard deviation for parameter $j$
$n$ = No. of parameters

This measure gives the mean weighted distance between units, with the shorter unit linearly interpolated to the longer unit. $W_d$ is the duration penalty weighing the difference between the lengths of the two units. This acoustic measure is used to define the impurity of a cluster of units as the mean acoustic distance between all members. The objective is to split clusters based on questions to produce a better classification of the units.

Although the available questions are the same for each phone type, the tree building algorithm will select only the questions that are significant in partitioning that particular type. The features used for CART questions include only those features that are available for target phones during synthesis. In our experiments, these were: previous and following phonetic context (both phonetic identity and phonetic features), prosodic context (pitch and duration including that of previous and next units), position in syllable, and position in phrase. Different features are significant for different phones by their very nature, for example, a feature representing pitch is only rarely used in unvoiced consonants.

### 3.4. Computing the Cepstrum Parameters

In order to cluster similar units in a database, we build an acoustic representation of them. This is also still a research issue but in the current work, we used Mel cepstrum. Interestingly we do not generate these at fixed intervals, but at pitch marks. Thus we have a parametric spectral representation of each pitch period. We have found this a better method, though it does require that pitchmarks are reasonably identified. The script file `make_mcep` takes wave files as input and gives the MFC Coefficients of the signal. It assumes that the pitchmarks (obtained using `make_pm_wave`) are already obtained for that corresponding waveform for which MFCC's are being calculated.

### 3.5. Building the Clusters

The function `build_clunits` is used for building the unit clusters, and the function runs through the following stages:

- The first stage is to load all the utterances in the database, sort them into segment type and name them with individual names.

- The next stage is to load the acoustic parameters and build the distance tables. The acoustic distance between each segment of the same type is calculated and saved in the distance table. Precalculating this saves a lot of time as clustering will require this value many times.
- The next stage is to feed the features that are used to index the clusters. The function `dump_features` does this job. It calculates the feature values for each unit in the database. The clusters are defined with respect to the acoustic distance between each unit in the cluster, but they are indexed by these features. These features then become available at synthesis time when no acoustic information is available. Thus they include features like phonetic and prosodic context rather than spectral information.
- Now that we have the acoustic distances and the feature descriptions of each unit, the next stage is to find a relationship between the features and the acoustic distances. This we accomplish using the CART tree builder `wagon` (Black, 1996). It finds out questions about which features best minimize the acoustic distance between the units in that class.

  During tree building, `wagon` uses two important parameters. The first one is prune_reduce. This gives the number of elements in each cluster to be removed in pruning. This removes the units in the cluster that are furthest from the center. This helps in pruning the tree without sacrificing the performance of clustering, and at the same time saves time. The next parameter is `cluster_prune_limit`, which defines the maximum number of units present in a cluster at a tree leaf. This parameter is actually used in post wagon build operation on the generated trees. This is useful when there are large numbers of a particular unit type which cannot be differentiated.
- The final stage in building a cluster model is to collect the generated trees into a single file and loading the unit catalogue, i.e. the list of unit names and their files and position in them.

### 3.6. Summary of the Training Steps

The basic processes involved in building a waveform synthesizer for the clustering algorithm are as follows.

- Collecting the database of general speech.
- Building utterance structures for the database
- Building coefficients for acoustic distances, typically, some form of cepstrum plus F0.
- Building distances tables, precalculating the acoustic distance between each unit of the same phone type.
- Providing each unit type with the selection features (phone context, prosodic, positional).
- Building cluster trees with the features and acoustic distances dumped by the previous two stages
- Building the voice description itself

## 4. Synthesizing an Utterance - Testing

In testing phase, the system is given a text input, and a waveform is synthesized using the voice already built during training. This section explains the processes involved in the implementation of the waveform synthesizer.

### 4.1. Grapheme-to-Phoneme Conversion

The input to the Tamil TTS is Tamil sentence in ISCII format. The phonemic transcription of the sentence is found out using a program called G2P. This module developed by us uses a set of Letter-to-Sound rules, and a mapping table to get the final output as a sequence of phonemes in ASCII format. These phones are the ones which are defined in the Tamil phoneset . The G2P inserts # symbols between words which can be later used to delineate the words in the input sentence.

### 4.2. Intonation Model

We follow the tree intonation method, which is a sophisticated and correct approach to intonation modeling. In this model, we assign *F0* values for each segment of the input text according to a tree-intonation model. We treat syllables as the units of tree-intonation model.

A CART tree is built for the parameters *F0 peak*, *postilt* and *negtilt* for all the syllables in the training corpus. The CART tree contains decision weights for the positive (postilt) and negative slope values (negtilt). We build *three linear regression models* (LR) to compute F0 values for start, mid-point and end-point targets of each segment (phoneme) within a syllable. The F0 values are computed by considering the contributions of the postilt, negtilt and *peak F0* values of the parent syllable. We assume the following;

- Start F0 : influenced by Peak F0 and positive slope (postilt) of syllable unit
- Mid F0 : influenced by positive (postilt) and negative slopes (negtilt) of syllable unit
- End F0 : influenced by Peak F0 and negative slope (negtilt) of syllable unit

The three target F0 values of each segment within a syllable are computed using the LR formula

$P = P_0 + w_i \star F_i + ...$

$P \rightarrow$ Target Pitch value

$F_i \rightarrow$ postilt, negtilt and *peak F0* weights (computed from CART tree)

$P_0 \rightarrow$ Default $P$ if $F_i = 0$

$w_i \rightarrow$Weighing coefficients computed using *least squares technique* (best fit for the observed data by minimizing the sum of squares of the vertical deviations from each data point in the line).

The whole idea is to find the appropriate F0 target value for each syllable based on available features (`postilt` and `negtilt`), by training from data. A set of features are collected for each syllable and a linear regression model is used to model three points on each syllable - start, mid, and end (Alan W Black, 2003). The predicted F0 values can be used to compute the target costs used in the unit selection module.

### 4.3. Duration Model

Given a reasonable sized database we can measure durations and features for each segment in the database. Then we can train a model using those measurements. The advantage is that we can include the contextual information in predicting the duration. The features used for building the duration tree reflect the phonetic context. Once we have the features and their descriptions, we use the program `wagon` to build the tree, which is later used for predicting the duration. The CART tree implicitly contains the context information, because of the way the splitting occurs.

It has been found that a better representation for duration modeling is zscores, i.e. number of standard deviations from the mean. In order to train a zscore model we need to convert the absolute segment durations. To do that we need the means and standard deviations for each unit type in the phoneset.

For each phoneme in the input text, the features are extracted, and the phoneme traverses through the tree depending on the questions (about the features) at the nodes. When it reaches a leaf node, the zscore values of the units at the node are taken, their average is found out, and is taken as the zscore of the phoneme under consideration. To get the absolute value of duration from zscore, the following relation is used:

$$dur = zscore * stddev + mean$$

where, mean and stddev correspond to the phoneme class to which the current phoneme belongs.

The phonetic relations (segment, syllable, word) are directly obtained from the input text that is converted as a sequence of phonemes. Once all the relations (phonetic and prosodic) required to create an utterance structure are obtained, we build the utterance structure for the input sentence.

### 4.4. Units in Festival

Though Festival has a unit selection synthesizer, the scope of it is limited. This is because the size of units is uniform - it is a phone. Hence the performance of this system would be less as compared to the one using non-uniform units. Even though there are multiple occurrences of the phonemes in different contexts throughout the database, ultimately we will be concatenating phonemes. Choosing phonemes as the units has the obvious disadvantage that the number of concatenation points would be equal to the number of phonemes in the sentence. The other drawback is that the number of candidate units satisfying a given criterion would be very large. So it affects the time consumed in finding the best sequence of units using Viterbi search. Even it increases the time to build the cluster selection trees, because the number of distance calculations is expensively high.

One more serious problem is that the co-articulation will not be effectively captured when phonemes are joined. However, this has been partially overcome in Festival by cleverly choosing the unit. Though the basic unit is phoneme, it is not always from the start of the phoneme to the end of the phoneme. In fact, it is from the mid of the previous phoneme to the end of the current phoneme. The unit so selected is similar to a diphone, except that it is not from middle of one phoneme to middle of the next phoneme. Also, the number of occurrences is large as compared to the single occurrence of a diphone in a database.

### 4.5. Unit Selection

At synthesis time, we have a stream of target segments that we wish to synthesize. For each target, we use the CART for that unit type, and ask the questions to find the appropriate cluster which provides a set of candidate units. The function $T_{dist}(U)$ is defined as the distance of a unit $U$ to its cluster center, and the function $J_{dist}(U_i, U_{i-1})$ as the join cost of the optimal coupling point between a candidate unit $U_i$ and the previous candidate unit $U_{i-1}$ it is to be joined to. The actual measure of join cost is a frame based Euclidean distance. The frame information includes F0, Mel frequency cepstrum coefficients, and power. Although this uses the same parameters as used in the acoustic measure used in clustering, now it is necessary to weight the F0 parameter to deter discontinuity of local F0, which can be particularly distracting in synthesized examples.

We use a Viterbi search to find the optimal path through the candidate units that minimizes the following expression:

$$\sum_{i=1}^{N} T_{dist}(U_i) + W * J_{dist}(U_i, U_{i-1}) \qquad (2)$$

*W* allows a weight to be set optimizing join cost over target cost. Given that clusters typically contain units that are very close, the join cost is usually the more important measure and hence is weighted accordingly.

### 4.6. Joining the Units

To join consecutive candidate units from clusters selected by the decision trees, we use an optimal coupling technique (Van Santen, 1997) to measure the concatenation costs between two units. We take the last three frames of the previous unit and the first three frames of the current unit, and then find the acoustic distance between these frames taken two at a time. This gives a total of 9 distances for a pair of adjacent units. The pair of frames (called join frames) which give the minimum distance are taken and they are overlap added to concatenate. The frames in the first unit to the right of the join frame are removed before concatenation. Similarly, the frames in the second unit to the left of the join frame are removed.

This technique offers two results: the cost of a join and a position for the join. Allowing the join point to move is particularly important when our units are phones: initial unit boundaries are on phone-phone boundaries which probably are the least stable parts of the signal. Optimal coupling allows us to select more stable positions towards the center of the phone. If the previous phone in the database is of the same type as the selected phone, we use a search region that extends 50% into the previous

phone; otherwise the search region is defined to be within the phone boundaries of the current phone.

| | Intelligibility | Naturalness |
|---|---|---|
| subject-1 | 3.7 | 3.2 |
| subject-2 | 3.2 | 3.5 |
| subject-3 | 2.7 | 2.65 |
| subject-4 | 3.5 | 3.33 |
| subject-5 | 2.8 | 2.67 |
| subject-6 | 3.0 | 2.8 |
| avg. scores | 3.2 | 3.0 |

Table 1: Average MOS scores given by subjects for 6 different synthesized Tamil sentences

## 5. Results and Conclusion

To evaluate the overall performance, perceptive evaluation was carried out. Six native Tamil speakers were asked to rate the synthesized speech in terms of intelligibility and naturalness on six different sentences. The result of this evaluation (listed in Table 1) showed good intelligibility and average naturalness.

Though the values of intelligibility and naturalness seem to be encouraging, they are still low. This can be attributed to the fact the output quality gets hampered by restrictions posed by Festival to use phones as units. Such type of units severely affect the coarticulation and the discontinuities at the joins, as explained in section 4.4.

The pitch contour of the synthesized speech showed sudden jumps at joining points. Post processing techniques are a definite alternative to smooth out such pitch discontinuities. This would help improve the intelligibility of the speech further.

Energy normalization of the output waveform can be done to reduce large variations in the loudness levels of the words across the sentence. The prosody models available with Festival may not suit the Tamil language. Research is yet to be done on the prosody models for Indian languages. When these things are taken care, there would be an improvement in the naturalness of the synthetic speech.

## 6. References


Alan W Black, Kevin Lanzo, 2003. *Building Synthetic Voices*. Carnegie Mellon University.

Alan W Black, Paul Taylor, 1997. Automatically clustering similar units for unit selection in speech synthesis. *Proceedings of Eurospeech 97*, 2:601–604.

Black, Alan W, 1996. *Edinburgh Speech Tools Documentation*. University of Edinburgh.

Black, Alan W, 2002. *The Festival Speech Synthesis System*. Carnegie Mellon University.

Van Santen, Richard Sproat, 1997. *Progress in Speech synthesis*. Springer.