Institut für
Technische Informatik und
Kommunikationsnetze

Diphone Corpus Recording and Monitoring Tool

Christoph Reller

Semester-Thesis SA-2005-21

Summer-Term 2005

Institut für Technische Informatik
und Kommunukationsnetze

Supervisors: H. Romsdorfer, Dr. B. Pfister

Contents

Abstract

Diphone concatenation stands at the basis of many of todays speech synthesis systems. When recording a diphone corpus it is usually unknown whether the synthesis later will be successful or whether there are some shortcomings in the corpus. The main concerns are spectral discontinuities and pitch mismatch at the diphone borders. This project presents the implementation of a tool which tries to monitor phone and diphone quality. The approach is to partition the task in two stages, thereby first initializing phone features (centroids and intrinsic pitch) and afterwards recording the corpus and comparing with these features. The theory behind centroids is explained and a detailed documentation of the implementation on Matlab is given.

1 Introduction

This sections describes speech processing issues which relate to this project. Some relevant points about diphone speech synthesis and diphone corpus recording are given as an overview. For a more in-depth treatment of these subjects please refer to [PHB05].

1.1 Diphone Speech Synthesis

Todays text-to-speech systems are commonly divided into two blocks. The first one is concerned with the generation of a phonological representation of the text, which roughly is a sequence of phonemes[1] together with some other parameters (e.g. stress, intonation ...) describing how they should be spoken. A second part, the phono-acoustic block, translates this information into a continuous speech signal.

Among a few other methods, concatenating procedures stand at the basis of the signal production part of the phono-acoustic block in many systems. These build on selecting slices of previously recorded speech, processing them and lining them up. When done correctly, this usually ensures with a high chance that the outcome will sound close to speech.

Diphone speech synthesis takes diphones as basic elements for concatenation. These are defined as the signal part taken from the middle of one phone[2] to the middle of the next phone. One diphone thus contains two half phones or in other words the transition from one phone to the next.

For N phones the theoretical number of such diphones is $(N+1)^2 - 1$, where the $+1$ introduces the silent “phone” and the silence-to-silence diphone is not counted. Given a selection of words for a some language this theoretical number is practically never reached. It is assumed here, that by some method all required diphones and all phones involved are determined beforehand.

With this signal production setup the major sources of unwanted results are the following:

- Wrong pronunciation by the speaker during recording
- Bad match at the joint between concatenated signal slices
- Distortion introduced by processing the signal slices

This project addresses mainly the second point, to some extent the first one and disregards the last one completely. Perceived discontinuities at the border of concatenated signals was researched in [KV01]. It is one of the aims of this project to provide a framework for monitoring these.

[1]A phoneme is defined as the smallest utter-able unit with semantically distinctive characteristics, usually written by means of the IPA (International Phonetic Alphabet)

[2]A phone is defined as one possible acoustic realization of a phoneme

1.2 Diphone Corpus Recording

Before any concatenative speech synthesis can start, some real speech signal must have been recorded to cut the slices from. This signal – called “corpus” – usually consists of a sequence of carrier words recorded from a professional speaker. An optimal or good selection of slices is usually chosen after the recording. It is assumed here that for every carrier word the potential diphone(s) to be cut are known beforehand.

Normally it is the target to have a good match at all potential joints even if the signal slices are not processed at all. The subsequent processing can only worsen the phone quality and increase discontinuities at the cuts because it is not designed for these purposes. This raises the question of how to control the following criteria, not after, but during the recording:

- What is the quality of the phones in some given diphone
- How good do the phones (diphone borders) match to each other

This project seeks ways to monitor these two criteria on-line, i.e. while recording. Since the quality of an isolated phone can be an ambiguous measure, emphasis is put on the second point in this project. Monitoring is regarded as being only practical if the tool is integrated with the recording application.

Recording a corpus demands some degree of organization. Lists defining the structure and sequence are needed to initialize a recording project. Other major parts include audio input of some defined quality, file in- and output for different formats, a consistent user interface and feedback to the operator and the speaker.

2 Overview

This report is structured into two major parts: a first theoretical part (section 3) gives the mathematical foundations while the second part (sections 4 and 5) describes in detail how the tool is implemented. All work is carried out on Matlab [3] version 7.0.1. Only a very small portion of the code is platform dependent and was implemented for a Unix as well as a PC environment.

This section outlines the most important parts of the work done by first describing the requirements and the scope of the tool followed by an overview of the solutions found from a theoretical point of view as well as how they are implemented.

2.1 Specification of the Tool

Table 1 lists the major features of the target Matlab application script. All components are meant to be written in a style which makes the script expandable in an easy way. Structured partitioning should make future developments and exchanges of single blocks straightforward. Furthermore it is important to keep the application independent from the language in which the corpus will be recorded.

[3] Matlab is a trademark of Mathworks Inc.

1	Single key choice style user interface on the Matlab command line
2	Project management to allow maintaining different recording projects
3	Input and validation of a phone list and word lists
4	Recording facility with input from a microphone
5	Simulation facility with audio file input
6	Calculation of phone features
7	Calculation of diphone features
8	Assessment of phone quality
9	Assessment of diphone quality
10	Textual and/or graphical feedback to the operator
11	Textual and/or graphical feedback to the speaker
12	Support of wave-files with corresponding label-files

Table 1: *Features*

The second point also addresses the more fundamental functionality of saving and recovering the all relevant information, while potentially keeping all files in a well structured form for later export or as simulation input.

In point 5 simulation means that instead of some microphone input, the audio information is fetched from files. In a wider sense this can include more than just the audio information. Simulation from previously recorded material together with the decisions the operator has made or from otherwise prepared data are two examples.

In points 6 until 9 lies much uncertainty as these are areas which are still being researched. The application however imposes constraints on them, such that only simple procedures can be considered here. Diphone features and diphone quality can build to some extent on phone features and phone quality.

In general it is estimated that quality, being quite a lax notion, is difficult to assess in an objective way. When regarding a single phone this issue can become utmost ambiguous. Phone features on the other hand are widely used in speech processing and their advantages should be exploited here.

Point 10 means displaying the calculated features and quality indicators as well as some guidance about the current position in the recording sequence. Point 11 can include – besides graphemic and phonemic representation of the word to be recorded – an audio feedback. This can give the opportunity to make the speaker aware of a mismatch in the pitch of the voice.

2.2 Proposed Solution

When selecting between potential alternative solutions it was important to consider feasibility given the short time. Ideas which were judged too complicated or of a too wide scope had to be dropped. To conform with the specification and to make the programming tolerant to changes, high flexibility and systematic partitioning was significant.

Another concern was to keep the processing complexity low with an acceptable performance. After all, parts of the application are meant run on-line. This ranges out all alternatives with excessive calculations during the recording session. Because a complete corpus has not a well determined length, computations which scale with word list lengths were not considered, and neither recursive algorithms.

It was decided to partition the whole diphone recording procedure up into the following distinct phases:

1. Introduction phase
2. Phone feature initialization phase
3. Diphone recording phase

In the introduction phase words are recorded but not further used neither to calculate phone features nor to extract diphones. Its purpose is to check the recording setup and bring the speaker into the right mood by possibly giving him/her some feedback about the pitch.

During the second phase, phone features are extracted from the recorded signal. Single phones or short words are recorded followed by selecting the relevant time positions for this phone and calculating the features from this information. The phones are assumed to be of high quality during this phase. Several repetitions can contribute to mean features.

The last phase comprises the actual recording of the corpus. The idea is to record one carrier word and then compare calculated phone features at the diphone borders with the ones given from phase 2. The quality of the diphone can then be assessed based on this comparison.

Since phones will be cut at their center, the focus lies on features describing the signal about some center time. This can be done by extracting perceptually relevant features around the center. These include mostly information about the spectrum, which can be assessed using spectral smoothing procedures.

By their nature there exist quite different phones. While there already exist classifications with respect to articulation and other parameters, a classification with respect to their central features is proposed here. This will be referred to as phone class. Most investigation is done on the phone classes dealing with quasi stationary phones (like vowels, fricatives . . .) are investigated.

To finally compare the initialized mean features of the phones with the ones calculated at the diphone borders a distance measure is needed. It must not be forgotten that the pitch for voiced phones is also an important phone feature. This makes the notion of distance manifold. The distance is also used to locate the diphone within the utterance, which poses some constraints on the possible sequences of phones to qualify for a carrier word.

The framework which runs all these procedures is a finite state machine (FSM), which is designed to act as a stable developing and debugging platform. Recording and simulation cycles were constructed such as to make the operation logical and flexible as well as exploiting the possibility of re-using code.

The internal data is defined to map most of the concepts and calculation inputs an outputs onto a systematic structure. When opening or creating a new recording project, this structure needs to be (re)constructed.

The concept of storing everything the operator has entered separately (as “input”) from what the entered information dictates to produce (as “output”) is followed consequently throughout the implementation. This allows reusing once recorded data as simulation input.

3 Phone Features and Diphone Quality

This section proposes ways to find and assess in a given signal features relevant to the problems outlined in the introduction. In this application these must be confined to procedures which consider single phones or single diphones only. Also, the algorithms used are mostly very common to speech processing and human speech perception.

Given some reference features, the quality of the diphone borders can be assessed by calculating features at the diphone borders and comparing them with the reference via some distance measure.

3.1 Proposed Phone Classes

The classification proposed here serves potentially two purposes at the same time:

- Assigning a phone to a distinct method for calculating central features
- Giving an indication how important co-articulation is for the speaker

The first point is the more important here, while the second one could give some indication of how more elaborate schemes than the one presented here could deal with the same problems. Table 2 lists the classes.

00	silence
01	stationary vowel, a)
02	stationary vowel, b)
03	stationary voiced nasal or fricative, a)
04	stationary voiced nasal or fricative, b)
05	stationary unvoiced fricative, a)
06	stationary unvoiced fricative, b)
11	voiced plosive, a)
12	voiced plosive, b)
13	unvoiced plosive, a)
14	unvoiced plosive, b)
21	diphthong, a)
22	diphthong, b)
a	standalone speakable
b	not necessarily standalone speakable

Table 2: *Phone Classes*

Class 00 is listed for technical completeness. To be exact, silence is not a phone but it is one possible boundary of a diphone. No features of silence are considered here since this is trivial.

For all phones two variants are listed. Standalone speakable means that this phone can be regarded as a word when preceded and followed by not more than one other phone. E.g. the word “tee” (/tʰiː/) makes the phone /iː/ qualify as standalone speakable. It is assumed here that the phone quality will not suffer for these phones when they are uttered isolated (e.g. as /iː/ instead of /tʰiː/).

All other phones are considered as not necessarily standalone speakable. This implies that there is no certainty of how authentic the phone is spoken if uttered isolated. Any phone feature initialization must therefore take care of this fact by acquiring these phones as a segment of a whole word where it can be embedded appropriately.

This project almost exclusively investigates classes 01 through 06, i.e. the stationary phones. When spoken, no phone is really stationary, but there is a phase at the center of the phone in which the magnitude spectrum of the signal stays almost the same. It is this feature which makes a phone classify as “stationary”, although quasi stationary would be a more correct term.

Apart from diphthongs (classes {21, 22}) it is mainly the stationary phones which cause problems when concatenating diphones. Most plosives have a pre-plosive pause which is a convenient place to put cuts, provided that it is present and long enough. For phones from classes 01 - 06 the cut lies in the stationary phase, where some spectral stability as built up and spectral discontinuities will be more likely to jut out perceptually.

With respect to the phone features regarded here, there is no difference between classes {01, 02} and {03, 04}, while classes {01, 02, 03, 04} and classes {05, 06} are distinctly separated by their voicedness. This is the only difference in phone features applied here.

3.2 Feature Estimation of Stationary Phones

The reason for feature extraction is to separate the relevant information from all disturbing and unwanted parts. While in the unprocessed signal the whole information is present, in the feature space only the relevant aspects are represented. This is the basis for the application of a distance measure, where only relevant information should be involved. Most feature extraction methods used here are taken from [PHB05].

Since analyzing the signal as a whole usually leads to useless information, a short-time analysis is made. This is done by defining a series of – usually overlapping – consecutive analysis frames. This segmentation is therefore completely defined by a frame length N and the shift between two consecutive frames.

By cropping the signal to the frame limits a rectangular window is introduced. It is common to replace this by a window function who’s DFT exhibits more favorable characteristics. Depending on the feature extraction procedure the side effects are significantly decreased but never eliminated. Many window functions have been proposed, some of which are optimal with respect to some criteria [PM96]. Since in this application the side effects are assumed to play a minor role when handled correctly, the hamming window is used almost throughout:

$$w(n) = 0.54 - 0.46 \cos \frac{2\pi}{N-1}, \qquad n = 0, \ldots N-1 \tag{1}$$

3.2.1 Spectral Features

As stationary phones have an almost constant spectrum at their center it is sensible to consider spectral information as a phone feature. Since the auditory system is not very sensitive to phase shift, the phase information is ignored and the magnitude spectrum remains.

The most common way to approximate signal spectra in a short-time analysis is to apply the discrete Fourier transform (DFT) on a frame of length N. Matlab of course implements this in a well known fast algorithm (FFT). The DFT and its inverse (IDFT) are defined as follows:

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

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k)\, e^{j\frac{2\pi kn}{N}}, \qquad n = 0, \ldots N-1 \tag{3}$$

Applying the DFT to a signal slice which was multiplied by some window function brings about the well known side effects of leaking and smearing due to the finite signal length and the form of the window respectively. Simplified one can say that increasing the length increases the resolution in frequency but decreases the resolution in time while decreasing the length has the contrary effects.

The magnitude DFT has still got too much detail for phone feature analysis as will be seen. Since the interest is in the rough shape of the spectrum, ways of smoothing the spectrum come into play.

3.2.2 Spectral Smoothing

Figure 1 displays a small windowed signal slice of length $N = 1024$ in the top right corner taken from the vowel /aː/. A DFT of the same length is calculated and displayed as a thin line in the frequency axes, along with three thick lines indicating smoothed versions of the DFT, done by means of three different methods

The un-smoothed, "full" DFT clearly shows noise like excursions plus a periodic ripple stemming from the quasi periodicity of the signal. Neither of these effects is desired at this point. (The pitch will be analyzed as a separate feature.) This is the reason for smoothing the spectrum.

Instead of displaying the un-smoothed DFT directly, the magnitude decibel value $H(k) = 20\log_{10}(|X(k)|)$ is drawn for positive frequencies, which is more adequate when referred to the human auditory system. Along with this three smoothed spectra are displayed, each calculated diffidently from $L = 20$ coefficients, whose meaning depend on the smoothing method.

Cepstral smoothing is achieved by first transforming the DFT into its real valued cepstrum [PM96]:

$$c(m) = \mathrm{IDFT}\left[\log\left(|X(k)|\right)\right] = \frac{1}{N} \sum_{k=0}^{N-1} \log\left(|X(k)|\right) e^{j\frac{2\pi km}{N}}, \qquad m = 0, \ldots N-1 \tag{4}$$

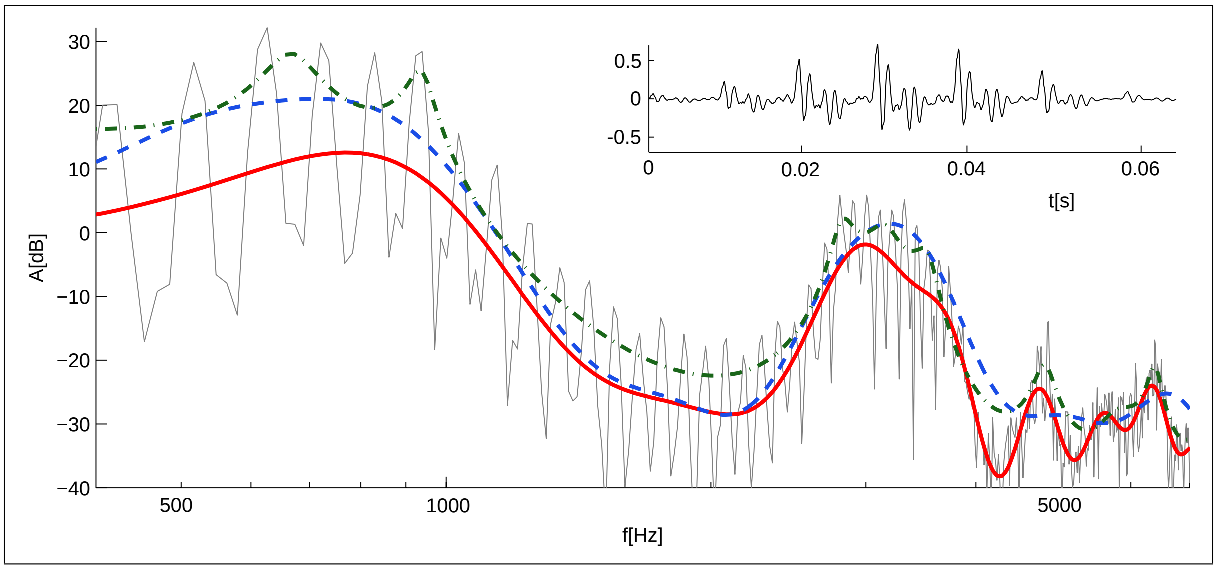


Figure 1: *Spectral Smoothing Methods – Thin line: un-smoothed spectrum, thick line: cepstral smoothing, dashed line: mel-frequency cepstral smoothing, dash-dotted line: LPC smoothing*

Then a cepstral window is applied, setting higher order cepstral components (CC's) to 0. For a cepstral window of length L the first L and the last $L-1$ coefficients are thus left untouched. The cepstral coefficients $c\left(m\right)$ are accordingly transformed into $\tilde{c}\left(m\right)$ of which many are 0. They are transformed back to the frequency domain by $\tilde{X}_c\left(k\right) = \exp\left(\mathrm{DFT}\left[\tilde{c}\left(m\right)\right]\right)$. The corresponding magnitude decibel value $\tilde{H}_c\left(k\right) = 20\log\left(\left|\tilde{X}_c\left(k\right)\right|\right)$ amounts to the cepstrally smoothed spectrum displayed in the figure. This function is completely defined by the first L coefficients in $c\left(m\right)$ (or in $\tilde{c}\left(m\right)$).

Mel-frequency cepstral smoothing is a variant of cepstral smoothing which works with mel-frequency cepstral coefficients (MFCC's). Before transforming the DFT into its real valued cepstrum a frequency transformation from linear frequency f to the mel scale $m(f)$ is done [PHB05]:

$$m\left(f\right) = 2595\log_{10}\left(1 + \frac{f}{700\mathrm{Hz}}\right) \tag{5}$$

On Matlab this is implemented as a filter bank with (in this case 24) triangular bands transforming $\left|X\left(k\right)\right|$ (the magnitude DFT) into $\left|X_{mel}\left(k\right)\right|$. Applying equation 4 to $\left|X_{mel}\left(k\right)\right|$ yields the mel-frequency cepstral coefficients $c_{mel}\left(m\right)$. Making again use of a window of length L, transforming the resulting $\tilde{c}_{mel}\left(m\right)$ back and reversing equation 5 yields the smoothed decibel magnitude spectrum $\tilde{H}_{mel}\left(k\right) = 20\log_{10}\left(\left|\tilde{X}_{mel}\left(k\right)\right|\right)$. Some further compensation due to the filter bank is applied to make the spectrum directly comparable in figure 1.

Linear prediction coefficient (LPC) smoothing relies on linear prediction. The signal is thereby approximated from a linear combination $\sum_{l=1}^{L} a_l s\left(n-l\right)$ of L preceding signal values. This is done at all N points reusing the same coefficients a_l. See [PHB05] or [PM96] for least squared error solutions to this. In the z-domain the synthesis filter $H\left(z\right) = 1/A\left(z\right)$ (where $A\left(z\right) = 1 + \sum_{l=1}^{L} a_l z^{-l}$) approximates the rough shape of the spectrum. $H\left(z\right)$ can be sampled on the unit circle to get the magnitude response $\left|\tilde{X}_{lpc}(k)\right| = \left|H\left(e^{j\frac{2\pi k}{N}}\right)\right|, k = 0, \ldots N-1$. Figure 1 displays the decibel value $\tilde{H}_{lpc}(k) = 20\log_{10}\left(\left|\tilde{X}_{lpc}(k)\right|\right)$. As with the other methods there are L coefficients a_l which specify the function completely. Due to the all-pole form of the synthesis filter this type of spectral smoothing tends to emphasize the peaks in the spectral envelope.

3.2.3 Centroids

Figure 2 visualizes a smoothed spectrogram. Along the time axis a slice of an example speech signal is displayed. A total of six frames are analyzed and the resulting (cepstrally) smoothed spectra are drawn along the frequency axis as thick lines. For the last frame a thin line shows the un-smoothed "full" spectrum for reference.

For a good display the frame length is chosen $N = 2048$. The smoothing is done with $L = 20$ coefficients as above. The frame shift $t_{shift} = 18.7$ms is larger than in the implementation for easier visual analysis. The speech signal slice corresponds roughly to the phones /. . . sø . . . /, where only the very end of the /s/ , but almost the whole of the /ø/ is contained.

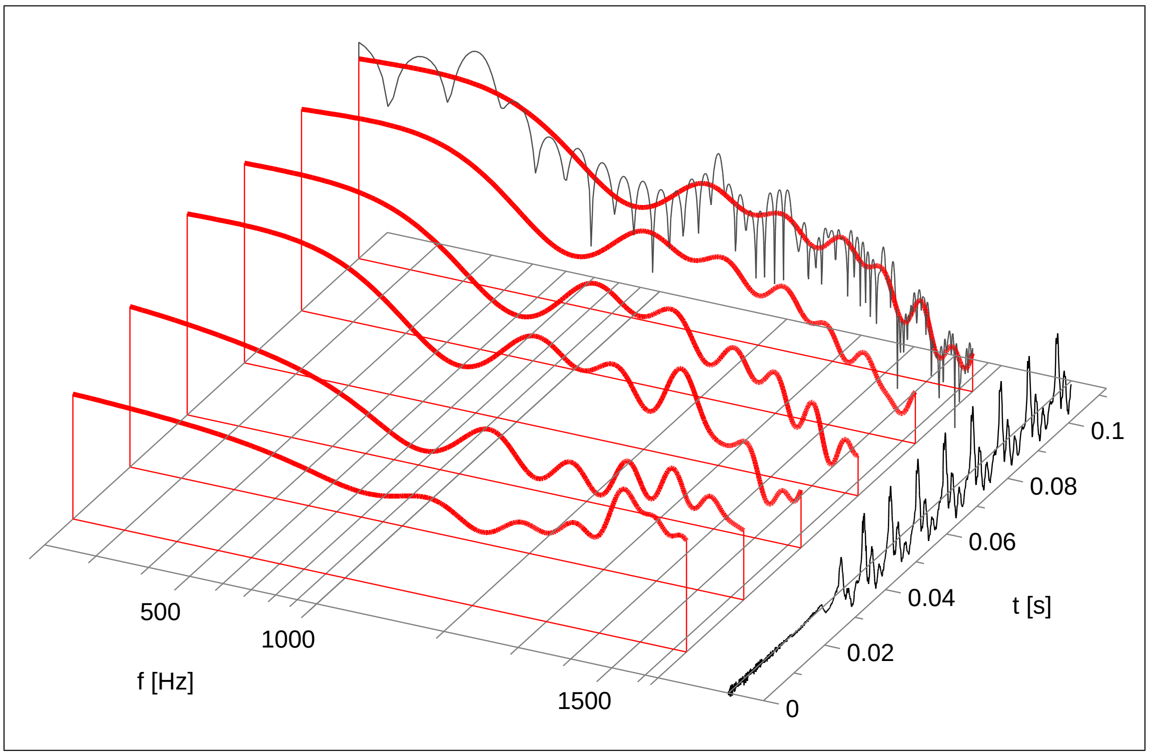


Figure 2: *Smoothed Spectrogram*

It is clearly visible that the smoothed spectrum changes during the phone transition and then settles as the stationary phase of the /ø/ establishes during the last three frames. The spectral shape of these is believed to characterize the phone /ø/, which brings us to the concept of a centroid.

A centroid is defined as the mean over potentially many frames and potentially many spoken instances of smoothed spectra in the "stationary" phase (at the center) of a phone of class

{01, 02, 03, 04, 05, 06}. This is one of the major phone features used in this project to calculate distance between phones and assess diphone quality.

3.2.4 Pitch

For the voiced phones (classes {01, 02, 03, 04, 11, 12, 21, 22}) the pitch can be extracted as another important feature.

The extraction algorithm used here is based on an analysis of the cepstrogram [PHB05]. Due to the periodicity in the speech signal, the spectrum will have a periodic ripple as can be observed in figures 1 and 2. This ripple gives rise to a peak value $c\left(m_{peak}\right)$ in the real valued cepstrum at m_{peak}, which corresponds to a frequency of $f_0 = m_{peak}/f_s$ where f_s is the sampling frequency.

Care must be taken to find the correct peak, since for instance $c\left(0\right)$, which corresponds to the signal energy, will usually be the global maximum of $c\left(m\right)$. This problem is met by specifying a range for f_0 corresponding to the capabilities of the (spoken) human voice (roughly $50 - 150$Hz). Furthermore, a lower threshold is set for the peak detection to assure that the pitch is assigned only to voiced parts of the signal.

For phones from classes {01, 02, 03, 04} the pitch should be stable during the stationary phase of the phone. Analogous to the case of the centroids, a mean pitch over the stationary phase of potentially many instances of a spoken phone can be calculated. This mean is sometimes called “intrinsic pitch”.

The stability of the pitch during the stationary phase can be an indicator for the phone quality. For the phone feature initialization phase this is not of such a high importance since only the intrinsic pitch is extracted, but it may also alter the centroid. This differs from the case when pitch is used to asses diphone border quality.

3.3 Possible Features for Other Phone Classes

Phones from classes {11, 12, 13, 14, 21, 22} have no stationary phase as such, which makes them unsuitable to direct application of the features presented above. This section gives some hints about possible features, but none of the ideas described here was implemented.

Plosives (classes {11, 12, 13, 14}) are known to have a low-energy pre-plosive pause followed by a sharp rise in energy at the beginning of the plosion. Some plosives (e.g. /t^h/) end in an additional aspiration phase. In diphone speech synthesis the cut is usually made at the end of the pre-plosive pause. The simplest feature for diphone border assessment could be the detection of this pause in the smoothed signal energy. The phone quality could possibly be assessed further by analyzing the extent of the pause and by means of the smoothed spectrogram of the plosion, although co-articulation with the succeeding phones may result in variation which maybe is not a disadvantage.

Diphthongs (classes {21, 22}) can be viewed as a succession of two (stationary) vowels, in which case the problem is reduced to phones from classes {01, 02}. However, as many diphone speech synthesis systems regard diphthongs as one phone, further solutions can be proposed. The cut region (diphone border) lies in the middle of the phone, where the spectrogram must be assumed to change strongly. One possibility could be to regard this progression of the spectrum as a feature. In a first step, the middle of the phone must be found by some method. Then a series of smoothed spectra together with a parameter indicating the width (duration) of the phone can be calculated and a mean of these taken over many instances of spoken phones could serve as a phone feature analogous to centroids.

3.4 Assessing Diphone Quality

Once reference phone features are initialized, any potential diphone boundary can be compared against these by calculating phone features of the carrier word at the time under investigation, followed by finding the distance between the features according to some distance measure. This can be done over the whole carrier word yielding a distance curve. (See figure 3 for an examples.) This curve's resolution, given by the frame shift in general differs from the one of the time signal.

If some stationary phone at a diphone boundary is spoken with some minimum quality, then the distance curve for centroid features will pass below some threshold. The total time during which the distance stays below this threshold can give an indication of where the diphone boundary would be acceptable in terms of spectral discontinuities.

For voiced phones the acceptable time window is further narrowed when the match of the pitch is assessed. Not only should the pitch be within some threshold from the intrinsic pitch for this phone, but also there is a need for some degree of flatness at a potential diphone border.

The length of the remaining acceptable time window for this diphone border must then still exceed some threshold. This is to ensure that when concatenating two diphones, both contribute to their common phone to some extent thus synthesizing the phone properly.

In total four thresholds can thus be defined:

- Maximal value for the centroid distance
- Maximal deviation from the intrinsic pitch
- Minimal flatness of the pitch
- Minimal length of the resulting acceptable time window

In this projects none of these thresholds is addressed directly, since it is believed that they depend on many factors (e.g. aptitude of the speaker, characteristics of the speech synthesis processing, target quality of the system, ...) whose estimation is very difficult. Finding values for these thresholds could be the subject of large scale tests.

3.4.1 Distance Measures for Centroids

This section introduces two distance measures used for centroids. Since there are three smoothing methods, the total number of possible distance curves amounts to six.

Euclidean distance is a common and widely used distance measure. Here it is applied directly to the coefficients which describe the smoothed spectrum, regardless of the smoothing method. The euclidean distance between L coefficients a_l and reference coefficients α_l is defined as following:

$$D_{euclid} = \sqrt{\sum_{l=1}^{L} (a_l - \alpha_l)^2} \tag{6}$$

Both cepstral methods presented in 3.2.2 have in common that the first coefficient assesses the signal energy. Since the energy can be adapted easily by means of scaling the amplitude, this first coefficient is weighted with some small number ϵ (here $\epsilon = 0.1$) in the distance calculation. This ensures that a mismatch in the signal energy is ignored to some extent. The euclidean distance then becomes:

$$D_{euclid} = \sqrt{\varepsilon (a_1 - \alpha_1)^2 + \sum_{l=2}^{L} (a_l - \alpha_l)^2} \tag{7}$$

Symmetric Kullback Leibler distance [KV01] is a variation of the Kullback Leibler distance (or relative entropy). It is applicable to two positive valued functions, which in this case are taken as the magnitude spectrum $X(k)$ and the reference magnitude spectrum $\Psi(k)$. The difference to the ordinary Kullback Leibler distance [CT91] is that the two functions are interchangeable without altering the resulting distance.

$X(k)$ and $\Psi(k)$ are calculated by sampling the smoothed spectrum at K (usually $= N/2$) points as mentioned in section 3.2.2. The symmetric Kullback Leibler (SKL) distance is calculated by first normalizing both functions, actually turning them into probability distributions:

$$\hat{X}(k) = \frac{X(k)}{\sum_{k=1}^{K} X(k)}, \qquad \hat{\Psi}(k) = \frac{\Psi(k)}{\sum_{k=1}^{K} \Psi(k)}$$

Then the SKL is calculated as follows:

$$D_{SKL} = \sum_{k=1}^{K} \left(\left(\hat{X}(k) - \hat{\Psi}(k) \right) \log \frac{\hat{X}(k)}{\hat{\Psi}(k)} \right) \tag{8}$$

This distance measure has the property of weighting spectral regions with high energy stronger than those with low energy. It is reported in [KV01] that this corresponds in a high degree to human speech perception. Intuition suggests that formants, being regions of high energy, are taken distinctly into account.

Figure 3 shows an example speech signal of the German words “die Dame” (/diːˈdaːmə/) in the top axis. In an earlier phone feature initialization stage, the centroid for the phone /aː/ was established using one of the three presented spectral smoothing methods.

In a first step, smoothed spectra have been calculated over the whole signal. Then these were compared with the centroid by means of the two distance measures above. To make the distance curves more comparable, they were additionaly scaled such that their minimum equals to 1.

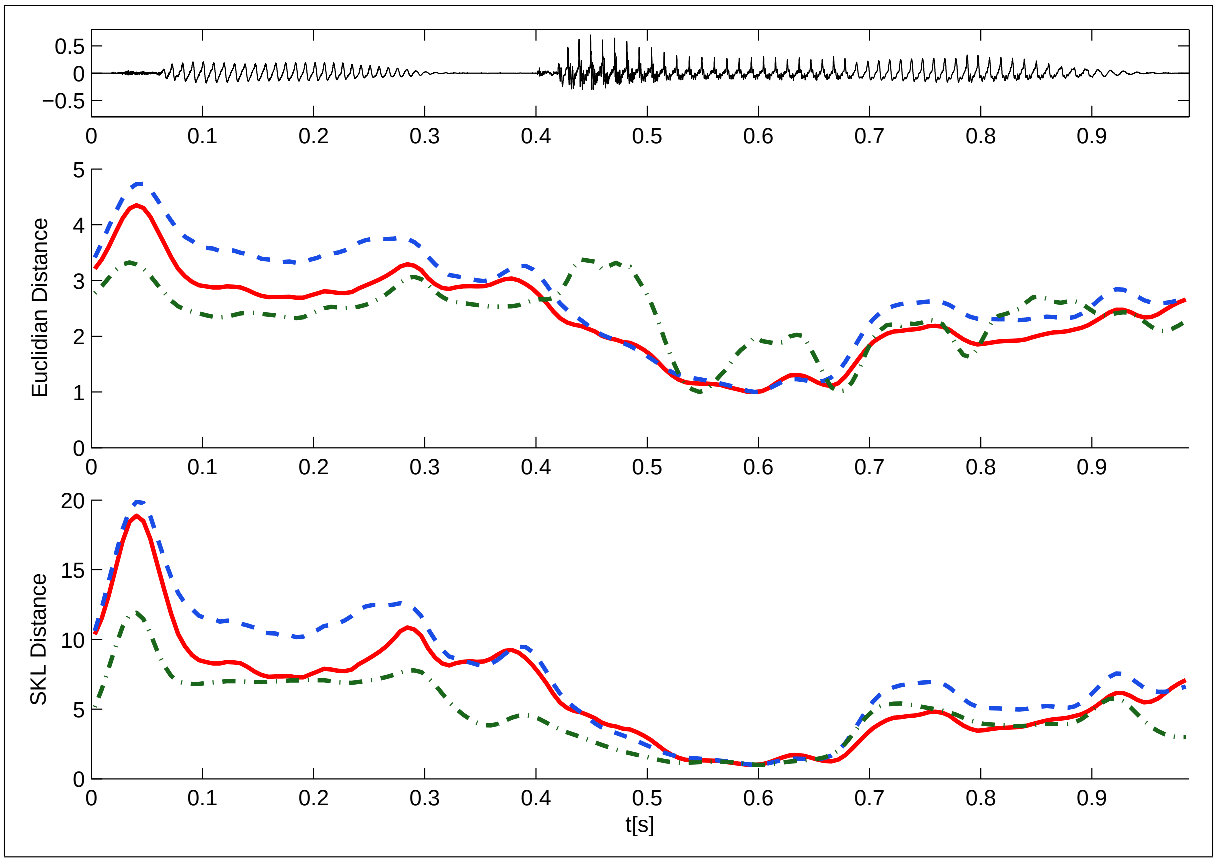


Figure 3: *Distances Compared – Solid line: cepstral smoothing, dashed line: mel-frequency cepstral smoothing, dash-dotted line: LPC smoothing*

In general SKL distance has a higher range in the values than euclidean distance, which could be due to the fact that more coefficients (i.e. $N/2$) are involved in the SKL distance than in the euclidean distance. For both distance measures the MFCC smoothing exhibits the highest distinctive character, while the LPC method seems to give smaller differences.

In the implementation the distance curves are also used to locate the phone within the word. For this purpose, euclidean distance on LPC's seems to work worst as the rise in the dash-dotted curve at 0.6s (in the middle of the vowel /aː/) indicates.

4 General Framework Implementation

This section describes all parts of the Matlab program code which are not specific to this implementation, but can be reused in similar applications. A recording and monitoring tool like this demands some degree of stability and systematic partitioning for the development as well as for in-field operation. Re-use of code is not a directly addressed issue here, but code consistency and any future development will profit from it.

4.1 Programming Conventions

Some special fonts and characters are used to distinguish between different meanings. The typewriter font "`typewriter`" is used for directory names, file names, excerpts from ASCII text files, excerpts from Matlab code and numerical values. Strings between triangular brackets "<" and ">" are names of variables for which values (usually numbers, sometimes strings) are substituted. The symbols "$\longleftrightarrow$" and "$\hookrightarrow$" are just used to wrap the line and have no further significance. The dot (e.g. in "`.field`") indicates that the string following it is an element (in Matlab also called "field") of a structure.

The program heavily makes use of Matlab structures, sometimes with quite high depths. To make the code more readable, every field in a structure which itself is again a structure begins with a capital letter. If the field name wants to hint at several words then this is indicated by capital letters, not by underscores. E.g. "spectral method" becomes "`speMethod`" since this is a string variable, while the structure "word list" has the field name "`WdList`".

All static data in this program – apart from a few global variables written in capital letters – is included in two structures:

- One structure for the program flow control (`Cntr`)
- One structure for all data (`Data`)

The program flow control part is described in the next section, while the data structure which comprises the application specific parts is explained in 5.3

Functions which operate on the whole of either the `Cntr` or the `Data` structure start with a capital letters and have capital letters to indicate several words (e.g. "`RecordNow`"). All other functions are written with small letters only and have underscores to indicate several words in the function name (e.g. "`centroid_distance`").

The starting file for the program is `DiphRec.m` in the folder `DiphRec1.5`. All Matlab code files are located in the sub-directories which are shown in table 3 together with a short description of their contents.

`FSM`	The finite state machine
`Proj`	Recording project management and initialization
`Run`	Recording cycle, calculation and display
`Sim`	Simulation cycle
`View`	Viewing cycle
`Centr`	Phone feature, distance and display calculation

Table 3: *Directories*

In the directory `lib` all used liberary functions are collected including recording, cepstral smoothing, lpc analysism pitch detection, label file access and other helper functions.

4.2 The Finite State Machine

All program flow control is implemented in the form of a finite state machine (FSM), which is held in the `Cntr` structure array. When starting the program then first this structure is initialized and then the FSM is started.

Every entry in the structure corresponds to one state the program can be in. Every state has therefore an identifying number – the index into the structure array `Cntr`. The starting state is always the first element (`Cntr(1)`). Before adding a state the state's ID number must be fetched using the global function `nextStateId`. This ID is then used for the initialization of the corresponding element of the `Cntr` structure and to indicate the target states in state transitions.

Figure 4 conceptually shows the capabilities of the finite state machine. Round boxes indicate states and all strings written in squared boxes are functions. The arrows show state transitions. Every element in the `Cntr` structure array thus potentially has three types of functions:

- "Action In" functions are executed when ever the state is entered.
- The "Catch Event" function determines which state will be next.
- "Action Out" functions are called when leaving the state.

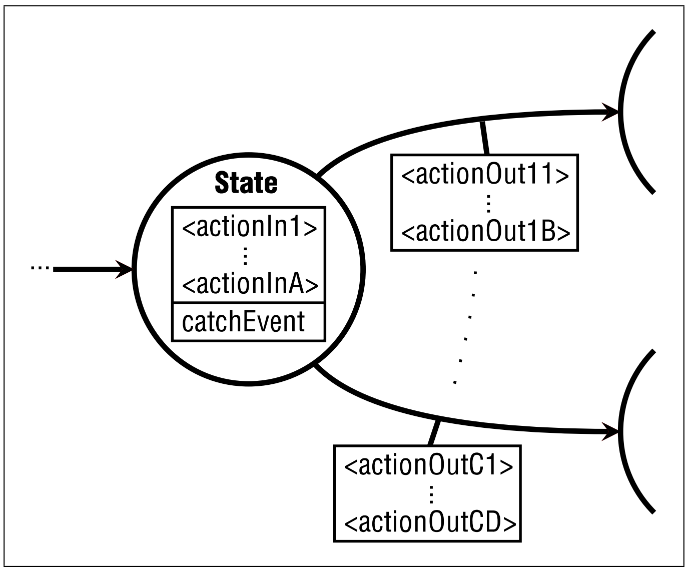


Figure 4: *Structure of one State of the FSM*

Every entry in `Cntr` must be initialized with the fields given in the list below. Some of the fields (marked with "(opt)") are optional.

`.stateName` A string which is only used for debugging purposes.

`.actionIn` (opt) A cell array of function handles – the "action in" functions, executed in the order of appearance in the cell array.

.actInType (opt) An array of integers of at most the same size as `.actionIn` which determines what arguments the corresponding `.actionIn` functions take and return. If not supplied, a default value is assumed.

.stateTrans An array of integers with state ID's (indexes into `Cntr`) of possible next states.

.catchEvent (opt) One function handle to a function which is returning at least one integer, the so called event, which is an index into `.stateTrans`. If `.stateTrans` has only one element, `.catchEvent` is not called.

.catchEvType (opt) An integer specifying whether the `.catchEvent` function takes and returns additional arguments. When not supplied a default value is assumed.

.actionOut (opt) A cell array of at most the size of `.stateTrans`. Every entry is again a cell array of function handles to be called in this order when a state transition occurs with the corresponding state ID of the new state in `.stateTrans`.

.actOutType (opt) A cell array of at most the same size as `.actionOut` holding arrays of integers, which indicate additional arguments and return values for every `.actionOut` function.

The finite state machine is run in the file `RunFSM.m`. A strong debugging environment is implemented by encapsulating all calls of function handles with a `try` and `catch`. The startup of the FSM and one complete cycle from one state to the next is structured as follows:

1. `RunFSM.m` is invoked with initialized `Cntr` and `Data` structures.
2. The state ID `state` is set to `1`.
3. If `Cntr(state).actionIn` is not existing then jump to step 9.
4. The next element in `Cntr(state).actionIn` is taken as the "action in" function to be called.
5. `Cntr(state).actInType` (if existing) is used to determine whether `Cntr` and/or `Data` is passed as an argument.
6. The "action in" function is executed within a `try` block, an error message is printed out in the `catch` block.
7. If the it fails and there are further elements in `Cntr(state).actionIn` the user is asked whether these should be called.
8. If `Cntr(state).actionIn` contains more elements, go back to step 4.
9. If `Cntr(state).stateTrans` has only one element, then `event` is set to 1 and jump to step 14. If additionally `Cntr(state).catchEvent` is not empty a warning is issued.

10. `Cntr(state).catchEvent` is taken as the "catch event" function to be called. `Cntr(state).catchEvType` (if existing) is used to determine the presence of the additional arguments `Cntr` and/or `Data`.

11. The `event` variable is set to `0`.

12. The function is executed within a `try` block, and execution goes to debug break mode if it fails. The user then can do some alterations and continue the program.

13. If `event` is `0` or exceeds the size of `Cntr(state).stateTrans` jump to step 10.

14. If `Cntr(state).actionOut{event}` is not existing then jump to step 20.

15. The next element in `Cntr(state).actionOut{event}` is taken as the "action out" function to be called.

16. `Cntr(state).actInType{event}` (if existing) is used to determine whether `Cntr` and/or `Data` is passed as an argument.

17. The "action out" function is executed within a `try` block, an error message is printed out in the `catch` block.

18. If it fails and there are further elements in `Cntr(state).actionOut{event}` the user is asked whether these should be called.

19. If `Cntr(state).actionOut{event}` contains more elements, go back to step 15.

20. `state` is set to `Cntr(state).stateTrans(event)`.

In `query_user_choice` a single key user input is implemented, which is meant to be used as `.catchEvent` function. It supports a default key (the return key) and the special input "`@`", which results in `event` to be `0` and puts the FSM into debug break mode. The program developer then can investigate the `Cntr` and `Data` structures and even alter them or manually call functions on them. The program can then continued with the Matlab command `dbcont` or stopped entirely with `dbquit`.

Before starting the program or while in debug break mode the Matlab command `dbstop if error caught` makes Matlab go to debug break at the place where the function fails, which is handy for locating errors. This functionality is cleared by `dbclear all`.

While running, four types of messages are generated by the FSM:

Error messages These are preceded by "`#ERRO:`" and include failures of functions and failures due to wrong initialization of `Cntr`.

Warnings These are preceded by "`#WARN:`" and include wrong number of function arguments.

Informations are preceded by "`#INFO:`" and include the active state's ID and the caught event.

Verbose informations are preceded by "`#VERB:`" and include assumed default values.

All messages together with their generation date and time are written into a log file named “`FSMlog.txt`” in the current Matlab directory. This is convenient for analyzing unexpected behavior of the FSM.

The messages can also be displayed on the Matlab command line. As default, only error messages are displayed. This behavior can be altered by providing an integer argument to `DiphRec.m`, ranging from 0 (for no messages) to 4 (for all messages), or in the menu `Home>Application Settings`. (See appendix B for a menu map.)

4.3 Wave Files Management

As the specification demands wave-file support, this is the only audio file format supported by the program for output as well as for input. The program makes use of wave-files in a recurring manner, repeatedly adding more data to it, which is not supported in Matlab by default. To have consistent file access for the specialized wave format a Matlab class was written.

The wave format is an elaborate or even “overcooked” standard (see [wavb] or [wava]), of which only a small portion is implemented here.

The class supports the following audio formats:

Audio channels	1
Sampling frequencies	48000, 44100, 32000, 24000, 22050, 16000, 12000, 11025, 8000 Hz
Bits per sample	16, 8

The following functionality is implemented:

- Access in read or read/write mode
- Displaying file information
- Writing arbitrary length data into any file position
- Reading arbitrary length data from any file position
- Erasing arbitrary length data at any file position
- Appending data to the end

After constructing the wave-file object with the constructor function `wavfile` the data is accessed using subscript assignment, i.e. by means brackets “`(`” and “`)`” with any contiguous range of integers in between them.

When writing data into a file position which exceeds the file length, zeros are padded at positions in between. Read data at positions exceeding the file length will be zero too. The data must be provided as (and when read has the format of) doubles ranging from `-1` to `+1`.

An example session could look like this:

```
>> wf = wavfile('filename.wav', false, true);
>> wf(:) = rand(1, 4);
>> wf(1:2) = []
wf.name      = filename.wav
wf.fid       = 3
wf.readonly  = false
wf.leaveopen = true
wf.autoclean = true
wf.len       = 2
wf.flen      = 4
wf.fs        = 48000
wf.bps       = 16
>> wf = close(wf);
>> clear wf;
```

Type `help wavfile` at the Matlab command line for more information on the supported modes of operation.

5 Application Specific Implementation

This section describes the actual implementation of the application (version 1.5). Within the Matlab program, the meaning of the word “centroid” differs from the one presented in the theoretical part (section 3.2.3). In this section the term “centroid” refers to any kind of diphone border feature or phone feature. This includes the “centroids” introduced in section 3.2.3 but also intrinsic pitch and potentially other forms of data.

The term “phase” addresses the partitioning presented in the overview section 2.2. A “step” is one item to be recorded for a given phase, distinctly identified by the graphemic and phonemic representation of the word together with the phone or diphone(s) to be extracted. The term “repetition” is used to identify the recording of one step within several such recordings. Potentially every step can be recorded several times. The word “position” means a valid combination of phase and step and possibly repetition.

With every step usually a decision has to be made, telling whether the recording is accepted or not. Here, this is termed “keeping” or “discarding” a recording. This segregation produces two streams of data, one called “input” and one called “output”. The input contains all recorded steps, whether kept or discarded, together with all decisions. This includes types 1 to type 3 data in the enumeration below. The output contains only the kept steps and does not store any decision data, which corresponds to data types 1 and 2.

Conceptually, four types of data are produced while running the program:

1. Audio data
2. Time markers for the audio data
3. Decisions

4. Calculation results

Utterations are recorded recurrently, producing type 1 data. Potentially all the audio data is stored in one wave file. This is one of the reasons why type 2 data is necessary.

Type 2 data is stored in so called “label files” following an “extended label file format” as used at the “Gruppe für Sprachverarbeitung” at the TIK-ETH. A tool was developed there to graphically edit a label-file wave-file pair. They have usually the same name but different file extensions (`.extlab` and `.wav`). This type of information can therefore be edited outside the program described here. Information from several sources is gathered in this type of data:

- For each recording an utterance detection is done[4]. This brings about a well defined length of silence at the beginning and the end of the audio signal. This information is stored as a word label.
- For the second phase, where phone features are initialized, a centroid label is produced. Thereby the segment of the signal which enters the calculation is marked with `start` and `end` labels as well as any intermediate labels used for the centroid in question. All centroid labels label must be completely contained within the word label.
- Although not implemented, it would be the intention to indicate acceptable diphone border regions for phase 3 with corresponding labels.

Type 3 data contains decisions made by the operator (either “keep” or “discard”) for each recording. Some decisions made by the program which at this point seem obvious but are important for simulation fall also in this category. This includes for instance whether the centroid times are set, which for phase 2 is always the case when in recording mode. All this information is stored in a text log or sequence file.

All calculations done (type 4 data) are completely defined by the project settings and type 1 through type 3 data. This includes centroids, pitch, signal energy, distances and graphics. Type 4 data is therefore never stored. This leaves also the opportunity to make changes to project settings or even to the calculation procedures which then are reflected automatically in type 4 data.

5.1 Recording Project Management

The term project here refers to a set of files in a distinct directory structure which carry all the information allowing the program to run in as many sessions as required. The following directories and files are located in a parent project directory whose name equals the name of the project:

`<projname>_projset.txt` The project settings. (`projname` is the name of the project.)

[4]On a Unix environment this is done on-line, meaning that the recording duration is adapted to the input signal. On a PC environment the recording itself is done with a fixed time which is estimated from the phonemic representation of the word to be recorded. An offline utterance detection is then run to finally come up with the same format for the audio information.

`sampleword.wav` The wave-file created by the command "`concatenate word`" in the `Project>Tools` menu.

`extinput` The directory containing the phone list and all three word lists.

- **`<projname>_phonelist.txt`** The phone list.
- **`<projname>_wordlist1.txt`** The word list for phase 1.
- **`<projname>_wordlist2.txt`** The word list for phase 2.
- **`<projname>_wordlist3.txt`** The word list for phase 3.

`input` The directory into which all generated files of the "input" stream are stored.

- **`<id>.wav` or `<projname>_inp<phase>.wav`** Depending on the project settings a single wave-file is produced for each step with a running number or one wave-file holding the complete input.
- **`<id>.extlab` or `<projname>_inp<phase>.extlab`** The corresponding label files for the wave-files.
- **`<projname>_inplog.txt`** The input log file.

`output` The directory into which all generated files of the "output" stream are stored.

- **`<projname>_otp<phase number>.wav`** The output wave-files.
- **`<projname>_otp<phase number>.extlab`** The corresponding label files.

`siminput` The directory from which the simulation input is fetched.

- **`<any name>.wav`** The simulation input wave-file(s).
- **`<any name>.extlab`** The corresponding input label file(s) (optional).
- **`<projname>_siminp.txt`** The simulation sequence file.

To create a new project, a project creation file must be provided (see appendix C.5 for the file format definition). If a file named `create.txt` exists in the current Matlab directory it is assumed that this is a project creation file, otherwise the user is asked for the filename. Once the creation file is validated the directory structure described above is created.

Besides the possibility of providing all project settings, this file includes information about a set of "external input" files. These consist of one phone list (see appendix C.1 for the file definition) and a word list for each phase (see appendix C.2, C.3 and C.4 for the file definitions). These give a definition for each step and their target repetitions. They are copied into the project-local directory `extinp` and renamed. Also, the project's settings file and the three output wave and label files are created as well as the input log file.

If a project is opened, then first the directory structure and mandatory files are checked. From the output data all the centroid means are calculated for existing steps. This can take a while for larger projects. An alternative would have been to store them as a binary file and let the operator choose whether he wants to recalculate everything or just load the latest version.

5.2 Program Flow

This section describes the major activities implemented. They are steered by the operator by means of menu navigation (see appendix B for a menu map) and monitored on screen on two windows each containing one Matlab figure, one for the operator and one for the speaker[5].

The operator window shows depending on the phase the following information:

- Outside any phase with just the project opened:
  - total number of recorded steps for each phase.
  - total of the target number of steps to be recorded for each phase.
- In all phases:
  - The position in terms of phase, step and repetition.
  - The graphemic representation of the word.
  - The phonemic representation of the word.
  - The signal (if existing) and it's smoothed energy in a time-amplitude axis.
  - The pitch detected (if a signal exists), in a time-frequency axis.
  - The smoothed spectrogram (if a signal exists), in a time-frequency axis.
- Additionally in phase 2:
  - The phonemic representation of the phone, whose features are to be initialized.
  - The centroid start and end time as a green patch in the time-frequency axis.
  - Frequency information about the centroid in a amplitude-frequency axis next to the spectrogram.
  - The distance over the whole word from the centroid calculated in the time-distance axis at the bottom.

  In phase 2 all lines in red indicate information due to the recorded repetition of this step. Dashed black lines show mean values when averaged over all repetitions of this step without the recently recorded one. Solid black lines show updated mean values when this repetition would be added to all other repetitions of this step.
- Additionally in phase 3:
  - Possible (not optimal!) diphone borders as a green patch in the time-frequency axis.
  - The frequency information of the initialized centroids of the left and right phone as black lines in the axes to the left and right of the spectrogram.
  - The frequency information of the suggested diphone borders as green and red lines in the same axes as above.

[5]The contents speaker figure is not yet implemented. It is meant to display the word to be spoken in a graphemic and phonemic representation as it is done in the operator figure. The speaker figure should also contain a recording indicator with three colors indicating “ready” (e.g. green, during the “Prepare” state), “recording” (e.g. red, during the “Record” state) and “wait” (gray, during all other states).

  - The normalized distances over the whole word from the left and the right centroid as green and red lines respectively in the time-distance axis at the bottom.

Once a project is open, the user has the possibility to choose between three modes:

- "Record" mode
- "View" mode (only available if the output data is not empty)
- "Simulate" mode

These modes are described in the following sections.

5.2.1 Recording

When choosing `go to record` at the `project` menu, the recording mode is entered. This is the only mode in which the order of the steps is exactly as written in the word list files. Figure 5 shows a simplified state transition diagram for this mode which is valid for all phases. The shorthand expression `q_u_c` stands for the function `query_user_choice`.

Recording starts in the "Prepare" state, where the position can be selected and a feedback can be given to the speaker[6]. `SetPosStr` constructs the information given in the menu (see appendix B for the exact string issued), while `InitGraphics` creates the necessary graphic objects in both the speaker and the operator figure. `DisplayStep` draws all step-specific information which exists before the recording is made. This includes the position information, the mean centroid for phase 2 (if existing) and initialized centroids for phase 3 (if existing).

Once the operator has decided to embark on a recording the "Record" state is entered, where the actual recording and utterance detection is done in `RecordNow`. Depending on the project settings the operator is asked in `RecCatchEvent` whether to discard the recording right away or whether to go on to "Decide". On the transition to decide the audio data is stored as input, all calculations are done and displayed.

In the "Decide" state the operator tells whether this repetition of the step is to be accepted. On the following transitions `StoreInput` is called once again, but this time to store the decision to the input log file. If the operator chooses "keep" then the audio and label data is stored as output in the function `StoreOutput`, as opposed to when "discard" is chosen. In phase 2 `StoreOutput` adds the newly calculated centroid for this repetition to the mean centroid for the phone which will be used in phase 3 as reference.

In phase 2, before doing this decision, the operator can edit the centroid start and end times by clicking with the left and right mouse button respectively on the time-amplitude axes, thus altering the extension of the green patch in this axis. Initially the start and end times for the

[6]The feedback to the speaker is not yet implemented. Giving the right pitch is a nontrivial task, especially for phase 3, where both diphone borders can have a different target intrinsic pitch. Since speakers never really "sing" the words to be spoken on a stable pitch it would be the best to give a word melody as a feedback. But this means controlling the duration, pitch and possibly amplitude of the voiced phones plus some solution for unvoiced phones. This approach would mean that a sort of speech would have to be synthesized from only knowing the phone features.

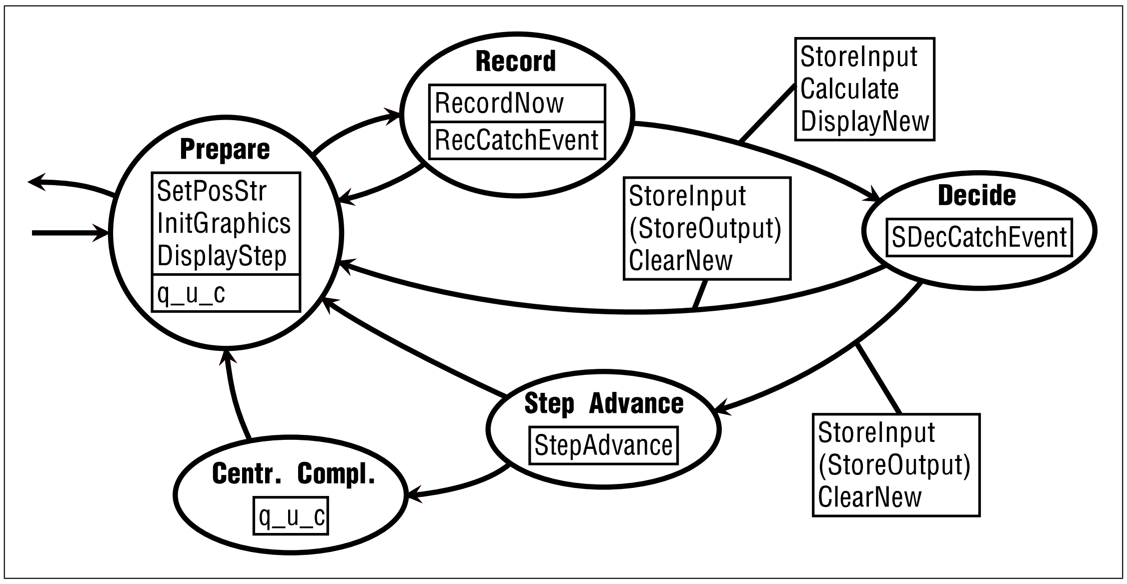


Figure 5: *State Transition Diagram for Recording Mode*

centroid are estimated for classes {01, 02, 03, 04, 05, 06} by means of observing deviations in the spectral coefficients and in the smoothed signal energy. To reflect the changes made in the other axes, the operator must choose `calculate again`. Also the operator has the ability to have an audio feedback of the selected time slice by choosing `feedback to operator`.

In phase 3, diphone borders are suggested by the program. This is actually not part of the specification and was just implemented to have an impression of the information buried in the distances. The real task would be to give potential time slices for the left and right diphone borders. The choice of the exact cutting times is meant to be an offline optimization task which can be done after recording the corpus.

Because of the reasons given in section 3.4 this functionality for phase 3 is not implemented, but the suggested borders are calculated and stored in the label files as diphone labels. This information is used by the command `concatenate word` in the menu `Project>Tools`, where any sequence of phones in ETHPA[7] notation can be entered (without the leading and trailing silence). The entered word will simply be formed by a concatenation of the "diphones" calculated in phase 3, with a small cross-fade time.

In "Decide" the operator also specifies where he wants to continue for the next cycle. If he wants to repeat this step then the FSM loops back to "Prepare". If the operator decides to go on then "Step Advance" is entered, where the next step in the step order is chosen. This might be the same as before, depending solely on the word list provided.

Once all steps are recorded at least as many times as specified by their target repetitions, the function `StepAdvance` detects this and the state transition to "Centroids Completed" informs the operator of the situation. This could also be the right point to let the operator choose whether he wants to discard recordings which in some way or the other adversely affect the mean centroids calculated. The program could possibly detect statistical outsiders and suggest them. No such functionality is implemented however.

[7]The ETHPA is a set of strings which map the IPA (International Phonetic Alphabet) into computer manageable ASCII characters. The ETHPA was developed at the ETH.

5.2.2 Viewing the Output

When re-opening a project, there is no way to gain visual access to the recorded and calculated data in the record or simulate mode. This is why a simple viewing mode was implemented. When entering "Choose Position", the position to be viewed is entered in the form `<phase>.<step>.<repetition>`, where `<repetition>` or `<step>.<repetition>` is optional.

In the "View" state all the necessary calculation to view the step and repetition specific information is done. Most of the functions have been described in 5.2.1. `GetSignal` fetches the signal from the output data instead of recording them as in record mode. Since this is only a viewing mode, the centroid start and end times can not be altered with the mouse.

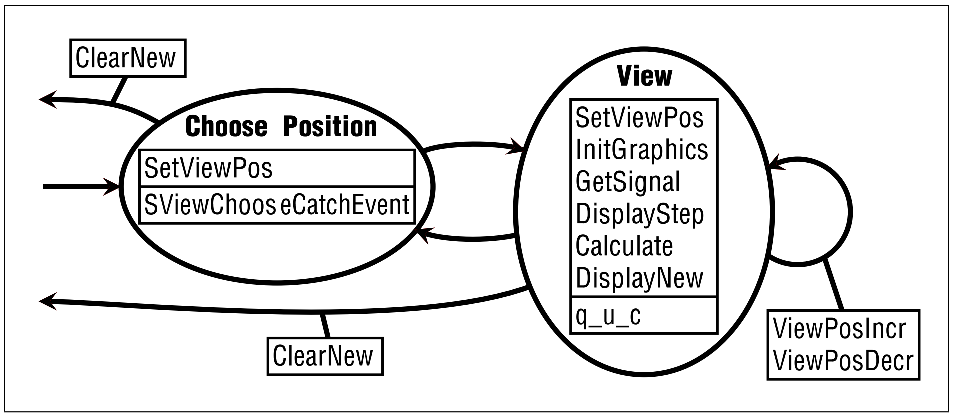


Figure 6: *State Transition Diagram for Viewing Mode*

The display of information does in general not equal the one the operator has seen during recording. This is because now the full output data is present and used to calculate centroid means while when recording, the means are building up gradually.

5.2.3 Simulation

As demanded by the specification a simulation mode is supported to create new input and output data from either of the following:

- External input data
- Previously recorded input data
- Existing simulation input data

This choice is done in "Query Simulation" (see figure 7 for a simplified state transition diagram). The simulation always needs at least two kinds of data: a simulation sequence file (see appendix C.7 for the file format definition) and some audio/label data. The sequence file is compatible with the input log file and contains all steps at least by indicating the wave file source for each step.

When simulating from external input data, the audio file(s) needn't necessarily be accompanied by corresponding label file(s), and if label files are given then they needn't contain any

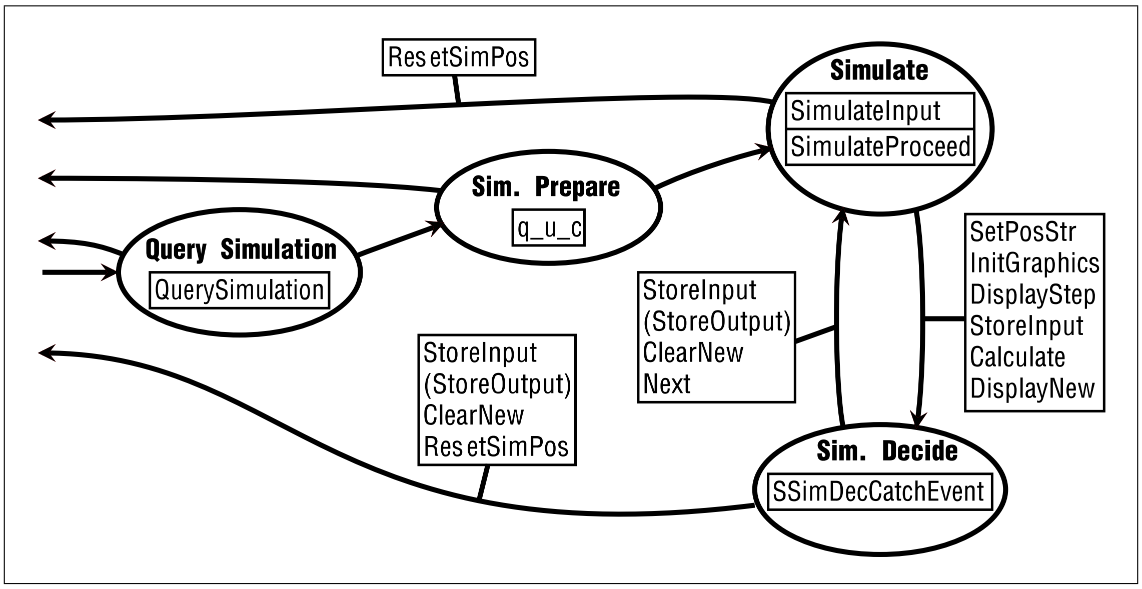


Figure 7: *State Transition Diagram for Simulation Mode*

utterance detection and/or centroid information. In the latter case the utterance detection and the centroid start and end time detection is run.

When simulating from previous input, the contents of the `input` directory is taken as is for the simulation. After having chosen the simulation data source, the given simulation sequence file is validated and the existence of required wave files is checked. The all data is copied into the `siminp` directory after deleting any files found there. When the choice was to re-use the existing simulation, the data already present in the `siminp` directory is left untouched.

In addition to the three simulation source options, two simulation modes – as set in the project settings file – are offered:

- `'SilentAll'`
- `'DecideAll'`

In the first case the FSM keeps on recurring to the “Simulate” state, doing all the un-given decisions silently until all steps are worked through and then quits simulation mode. In the second case a transition is made into “Simulate Decide” for each recording. The new information is displayed as in recording mode and the user has the option of keeping or discarding as well as altering the centroid start and end times (in phase 2) with the mouse.

5.3 The Data Structure

The `Data` structure contains all the application specific information which needs to survive a single function call. Since this is quite a large piece of data it needs to be structured well lest one looses the overview. Remember that all field names starting with capital letters again are structures. Not all fields are described here, but the ones presented give a more detailed insight into how internal calculations are done. This section therefore also describes functionality on a lower level of detail as well as mentioning some program features, which were not yet described.

`.Fn` Project related file names as strings.

`.Set` Project settings. This structure is written to the project settings file and some fields can be altered manually there safely. Only the fields to which this applies are listed here. Simulation settings:

- **`.simExtInpDir`** The directory for external simulation input.
- **`.simExtInpFn`** The simulation sequence filename for external simulation input.
- **`.simulation`** The simulation mode. Must be one of the following:
  - `'decideAll'` – Every simulation step is displayed and the user is asked for the decision whether to keep or discard.
  - `'silentAll'` – The whole simulation is run without interruption. If the simulation sequence file does not contain any information whether to keep or discard steps, all steps are kept.

Recording settings:

- **`.askAfterRec`** A logical value. If it is true (`1`) then after each recording but before any calculation is done the signal is displayed and the operator is asked to confirm the recording.

Display settings:

- **`.dispCepsWinlen`** The length of the cepstral window used to smooth the displayed spectrogram. This has no effect on any calculations.

Calculation setting for the pitch detection:

- **`.f0nfft`** DFT length used for the pitch detection. This will usually be higher than `.nfft` because more resolution in frequency is needed to gain accurate information about the pitch.
- **`.maxf0`** The maximal allowed pitch in Hz.
- **`.zlim`** The detection threshold.

Calculation settings for the utterance detection:

- **`.maxDur`** Maximal utterance duration in seconds.
- **`.maxWait`** Maximal wait time for the utterance in seconds.

Calculation settings for the centroid calculations:

- **`.winlen`** The length of the analysis frame in terms of samples.
- **`.winshift`** The distance between the start samples of two consecutive frames.
- **`.nfft`** Length ($= N$) for the DFT. This can be longer than `.winlen` in which case a zero padding is applied to the windowed signal.
- **`.numSpecCoeff`** The number of coefficients used for spectral smoothing.
- **`.nfilt`** Number of filters in the Mel filter bank.

.specMethod Defines the spectral smoothing method:

- `'cc'` – Cepstral smoothing.
- `'mfcc'` – Mel frequency cepstral smoothing.
- `'lpc'` – Smoothing by linear prediction.

.distance The distance measure, must be either of:

- `'euclidian'` – The euclidian distance is used.
- `'SKL'` – Symmetric Kullback Leibler distance is used.

.PhList(k) This is the structure array where the phone list is held. Each phone is identified by its position `k` in the array. The helper functions `index2phonemic` and `phonemic2index` convert between an array of indexes into this structure and an ETHPA string, and vice versa. The structure array has the following fields

.ethpa The ETHPA ASCII string.

.class The diphone boundary feature class as specified in section 3.1.

.char An array of characters which are character codes of some font. The font name to be used by Matlab is stored in the global variable `IPAFONT` and initialized in `DiphRec.m`.

.CeSt A centroid statistics structure. (See below for a description.)

.WdList{<phase>}(<step>) This is a cell array with three cells holding structure arrays which represent word lists, one for each phase. `step` is the index into the structure arrays and together with `phase` identifies one recordable element uniquely. To some extent the three structure arrays are similar with the following fields.
For all phases:

.graphemic The graphemic representation of the word as a string.

.phonemic The phonemic representation of the word as an array of indexes into `.PhList`.

.rep The total target repetitions for this step.

Additionally for phase 2:

.phone The phonemic representation of the phone whose features are being initialized as an index into `.PhList`.

Additionally for phase 3:

.Diph(<diphnum>) A structure array who holds all diphones for this carrier word.

.l An index into `.PhList` specifying the left phone.

.r An index into `.PhList` specifying the right phone.

.wdListOrder{<phase>} This is a cell array with three cells – one for each phase – each holding an array of indexes into `.WdList{<phase>}` (i.e. steps), which is used to determine the next step to be recorded in the `StepAdvance` function. The recording order of steps within one phase is constructed from the given word lists (see appendix C.2 - C.4 for the file format definitions).

`.Otp{<step>}(<repetition>)` Three cell arrays (one for each phase), which hold one cell for each step, where a structure array is kept to store every single repetition of the output data. The following field are present.
For all phases:

- **`.start`** The sample number of the start of the utterance.
- **`.end`** The sample number of the end of the utterance, stored together with `.start` as a word-level label in the corresponding output label file.

Additionally for phase 2:

- **`.CeTi`** A centroid times structure. (See below for a description.)

Additionally for phase 3:

- **`.Diph(<diphnum>)`** A structure array holding information about every diphone of this carrier word.
  - **`.CeTi`** A diphone times structure which is identical with a centroid times structure in this implementation. (See below for a description.)

`.otpWav{<phase>}` The wave file objects, one for each phase in a cell array.

`.Inp` A structure holding information about the current input with (among others) the following fields:

- **`.number`** Identification number for this input, as in the input log file.
- **`.keep`** A logical value telling whether the recording is being kept or discarded.

`.Sim` A structure holding information for the simulation, where some of the fields are:

- **`.Seq(<seqnum>)`** A structure array with one entry for each step in the simulation sequence, reflecting the information given in the simulation sequence file. Depending on the simulation mode, some of the fields are optional.
  - **`.phase`** The phase, a mandatory field.
  - **`.step`** The step, a mandatory field.
  - **`.file`** A string holding the name of the source wave file, located in the `siminput` directory; a mandatory field.
  - **`.start`** The start sample within the wave file; not required if the wave file contains only one recording.
  - **`.end`** The end sample within the wave file; not required if the wave file contains only one recording.
  - **`.uttdet`** A logical value telling whether an utterance detection was already applied to the audio information; an optional field with default value `false`.
  - **`.tidet`** A logical value telling whether the centroid or diphone times were already determined and stored in the label file; an optional field with default value `false`.
  - **`.keep`** A logical value telling whether to keep this recording; an optional field with default value `true`.

- **`.start`** The starting point of the simulation given as an index into `.Seq`.
- **`.end`** The ending point of the simulation given as an index into `.Seq`.
- **`.pos`** The current simulation point given as an index into `.Seq`.

`.mode` This is a string, identifying the mode in which the program is right now. This an important information for functions which are called in different modes. If the string is empty then we are in the “project” mode (the `Project` menu) from which one of the following modes can be entered:

- `'record'`
- `'simulate'`
- `'view'`

`.Pos` A structure identifying the position for recording with the following fields:

- **`.phase`** The phase.
- **`.step`** The step.
- **`.rep`** The repetition.

`.ViewPos` A structure identifying the position for viewing with the same fields as `.Pos`.

`.New{<phase>}` This is a cell array that contains information about the newly recorded (or simulated or viewed) step one structure per phase, with fields as follows.
For all phases:

- **`.sig`** The audio signal.
- **`.sigt`** An array containing the times in seconds corresponding to the audio signal.

Additionally for phase 2:

- **`.Centr`** A centroid structure. (See below for a description.)
- **`.CeTi`** The centroid start, end and intermediate times in a centroid time structure. (See below for a description.)
- **`.CeSt`** The updated centroid statistics if this recording were accepted, held in a centroid statistics structure. (See below for a description.)

Additionally for phase 3:

- **`.CeTi`** The diphone start and end times, held in a centroid times structure. (See below for a description.)

`.Disp{<phase>}` A cell array with three cells, one for each phase, containing additional information needed to display the data.
For all phases:

- **`.Time`** Contains additional information for the signal in the time-amplitude domain.
  - **`.energy`** The smoothed signal energy.

- **`.Spec`** Contains information about the spectrogram to be displayed.
- **`.F0`** Contains the pitch information.
  - **`.val`** The pitch values in Hz.
  - **`.t`** The corresponding times in seconds.

Additionally for phase 2:

`.CeDi` This is a structure which holds the following three centroid display structures. (See below for a description.):

- **`.OldMean`** The old centroid as if this recording was discarded.
- **`.NewMean`** The new centroid as if this recording was accepted.
- **`.New`** The centroid of only this recording alone.

Additionally for phase 3:

- **`.CeDi`** A structure which holds centroid display structures, which can hold information in the time as well as in the frequency domain. (See below for a description.):
  - **`.Meanl`** The centroid of the left phone.
  - **`.Meanr`** The centroid of the right phone.
  - **`.Diphl`** The centroid calculated from the recorded signal only, at the suggested time for the left diphone border.
  - **`.Diphr`** The centroid calculated from the recorded signal only, at the suggested time for the right diphone border.
  - **`.Distl`** The centroid distance from the left phone's centroid calculated over the whole signal.
  - **`.Distr`** The centroid distance from the right phone's centroid calculated over the whole signal.

`.Fig` This is the structure where all the Matlab handle graphics is kept.

- **`.Spk`** The speaker figure.
  - **`.hFig`** The figure handle.
- **`.Op`** The operator figure.
  - **`.hFig`** The figure handle.
  - **`.PrGfx`** A structure which contains the textual project information as well the position information, which both are given in the top left corner.
  - **`.Gfx{<phase>}`** Here are all the axes and their graphic contents.
    For all phases:
    - **`.TA`** The time-amplitude axis in the top center.
    - **`.TF0`** The time-pitch axis just underneath the time-amplitude axis.
    - **`.TF`** The time-frequency axis displaying the smoothed spectrogram in the middle of the figure.

    Additionally for phase 2 (In phase 2 more graphic objects are also added to the axis described above):

`.CF` The centroid-frequency axis, to the left of the spectrogram.

`.TC` The time-centroid axis, at the bottom of the figure.

Additionally for phase 3 (Like phase 2, phase 3 also adds more graphic objects to existing axis):

`.CFl` The left centroid-frequency axis, to the left of the spectrogram.

`.CFr` The right centroid-frequency axis, to the right of the spectrogram.

`.TD` The time-distance axis, at the bottom of the figure

In the following some structures which are often used are described. They appeared above in the description of the `Data` structure and are used consistently for all centroid related functions located in the `Centr` subdirectory of the Matlab program tool.

`Centr` This is a structure which is meant to store any information regarding diphone border features. Conceptually these can be centroids as defined in section 3.2.3 or any other data relevant. The fields present in this structure are thus meant to be dependent on the phone class. This implementation knows only two slightly differing types of centroids (meaning diphone border features):
Classes {01, 02, 03, 04}:

- **`.spec`** A two dimensional array with the spectral coefficients obtained by one of the methods described in section 3.2.2. The first dimension is the index of the spectral coefficient and the second dimension (which potentially is 1) is the index of the analysis frame.
- **`.f0intr`** The intrinsic frequency.

Classes {05, 06}:

- **`.spec`** A two dimensional array with the spectral coefficients like for classes {01, 02, 03, 04}.

`CeTi` This structure contains information about centroid times or diphone times, which is meant to be storable in a label file. It has the following mandatory fields:

- **`.start`** The starting sample of the centroid/diphone relative to the word starting sample.
- **`.end`** The ending sample of the centroid/diphone.

When used for centroids, this structure can contain more fields depending on the centroid class. For classes {01-06} one field was added in this implementation:

- **`.mid`** This is the time between the centroid start and end where the centroid distance, when calculated over the whole duration of the centroid, is smallest. It is not used further in any calculations, but it is displayed as a vertical line in the time-amplitude axis. For stationary phones this can possibly give a hint at skewness in the signal.

`CeSt` The centroid statistics holds the following fields to describe an initialized centroid:

- **`.Mean`** This is a `Centr` structure describing the mean centroid. For classes {01-06} its field `.spec` will therefore have a second dimension of size 1.

- **`.Mass`** This is also a `Centr` structure, but the fields are all numeric values indicating how much data has entered the `.Mean` until now. This is used for updating the mean without a whole recalculation when an element is added.

`CeDi` The centroid display structure holds information to be displayed about a centroid.

- **`.CF`** Information about the centroid in the frequency domain. For class {01-06} phones this is the smoothed spectrum representing the centroid.
  - **`.val`** The values.
  - **`.f`** The corresponding frequencies.
- **`.TC`** Information about the centroid in the time domain. In this implementation this is only used to store centroid distance information. Diphone border features for classes {21, 22} (diphthongs) might have real time domain information.
  - **`.val`** The values.
  - **`.t`** The corresponding times.

6 Recommendations

This section gives some hints about where future development might be needed and in which direction it could go. It also points out some shortcomings of the presented project and ways to overcome them.

Although the approach of a two phased scheme with separate phone feature initialization provides considerable advantages in terms of ease of implementation, is also has some dangers. When spoken isolated, stationary phones will be somewhat different from the case when spoken embedded in a word. Although it is hoped that the difference mainly lies in the duration, it makes the system strongly dependent on the speakers abilities to utter phones from classes {02, 04, 06} properly.

Some phones of these classes naturally exhibit strong co-articulation effects, which means that depending on the context the phone is spoken distinctly different. This raises the question of what a good quality phone feature for this phone is, or whether it would be wiser to split this phones into several variants which poses many follow-up problems.

It can therefore be recommended to embed difficult phones in words and record them several times in a different context during the initialization phase. The implementation, though not preventing this, does not give a good support for this approach since during phase 2 there is no segmentation algorithm provided to roughly locate the phone.

The same segmentation problem shows up in phase 3 again, where now it is “solved” by means of the distances. It is therefore recommended that the segmentation issue is separated from calculating the diphone border qualities. Any segmentation can of course still build on phone features though.

A completely different way of approach would be to do initialization and diphone quality calculation in one phase. This would mean that the word list has to follow some hierarchy such that at every step mostly initialized phones appear in a word. While, with a systematic statistical concept, this approach could solve the problem of co-articulation, it might also have some serious disadvantages. This is because when implemented correctly it might come down to optimization procedures which grow with the number of phones whose features are already initialized. So computation time could blow up as the corpus recording proceeds.

To correctly judge diphone quality it is suggested in section 3.4, that tests are needed to be able to give some decisive thresholds for quality criteria. Maybe a statistical approach based on the information given for this particular speaker could replace these tests. This however might mean that the tool judges quality with respect to a mean for this particular speaker, which does not make any absolute statement about the usability of the corpus for diphone concatenation.

7 Conclusion

A possible implementation of a tool for recording and assessing diphone corpora was presented. Despite some shortcomings and a few unfinished parts the tool was shown to work satisfactory and as expected.

Some theory on phone features was presented here, most of which is already well known and some direction regarding phone features for plosives and diphones is given. For for quasi stationary phones centroids were successfully applied and exhibited their expected behavior.

In order to make this a stable and well structured tool a great effort had to be put in the framework. Sensible partitioning makes the program modular clearly segregates the different tasks. This also eases any further development considerably. In total about 7300 lines of Matlab code was written, some parts with a high density, distributed over 82 files.

Last but not least the author has done some lengthy tests with his own voice on this tool. They showed that the speaker is put under some stress to pronounce properly and hit the right pitch. It is estimated that the application of this tool could result in usable corpora with low discrepancies in terms of spectra and pitch mismatch.

References

[CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., 1991.

[ipa] A free ipa font:
http://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&id=encore-ipa.

[KV01] E. Klabbers and R. Veldhuis. Reducing audible spectral discontinuities. In *IEEE Transactions on Speech and Audio Processing*, volume 9, pages 39–51, January 2001.

[PHB05] B. Pfister, H.-P. Hutter, and R. Beutler. *Sprachverarbeitung I*. Vorlesungsskript für das Wintersemester 2004/05, Departement ITET, ETH Zürich, 2005.

[PM96] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing*. Prentice Hall, 1996.

[wava] Simple wave file format description:
http://ccrma.stanford.edu/courses/422/projects/WaveFormat/.

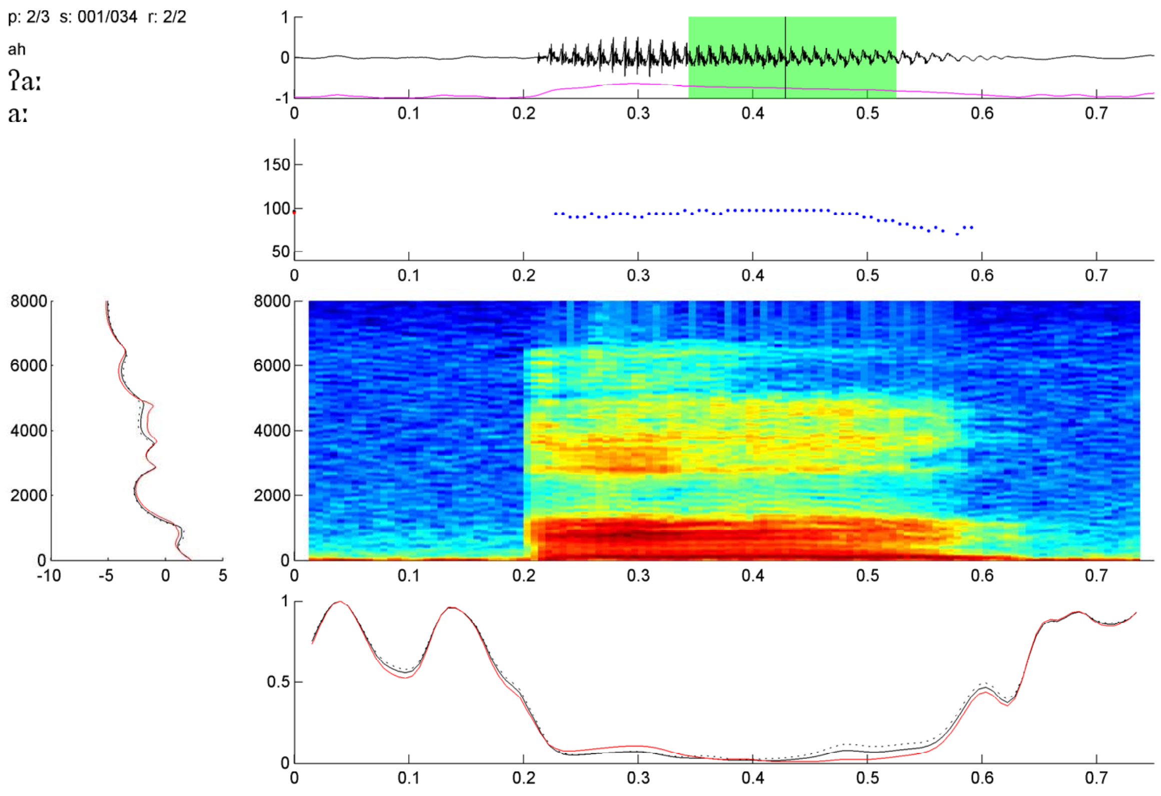
[wavb] Wave file format description:
http://www.borg.com/ jglatt/tech/wave.htm.

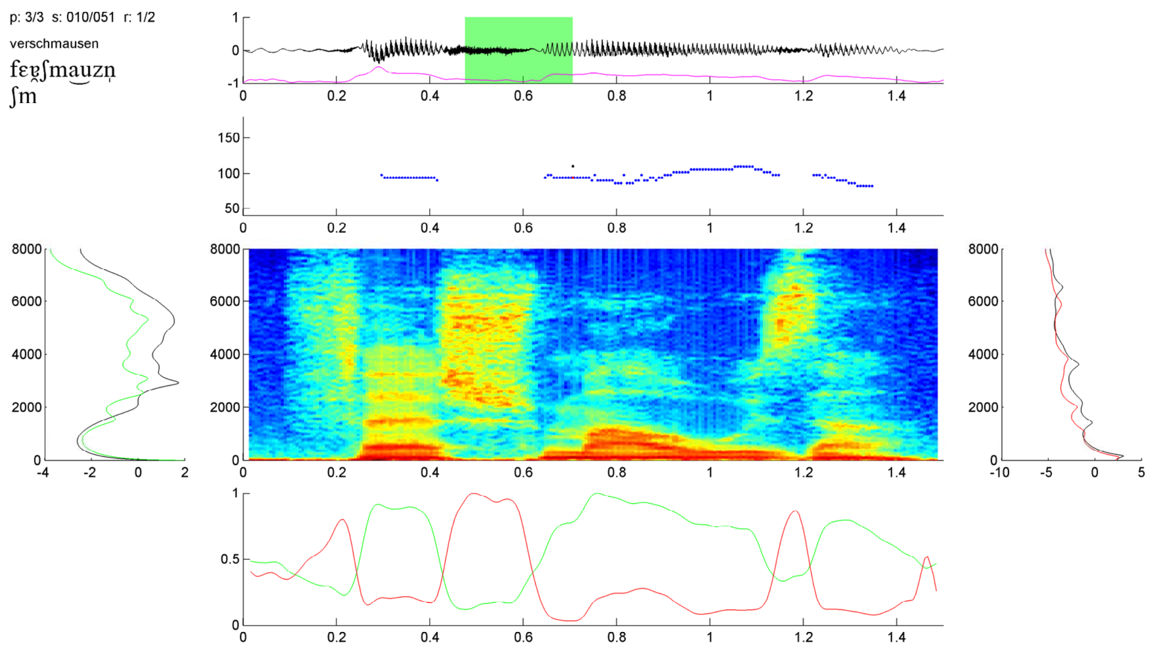
[WM98] J. Wouters and M. Macon. Perceptual evaluation of distance measures for concatenative speech synthesis. In *International Conference on Spoken Language Processing ICSLP 98*, pages 2747–2750, 1998.

Appendix

A Matlab Screen Shots

Two sample screen shots for phase 2 and 3:





B Menu Map

This section lists the major “menus” through which the user can navigate. A start (`*`) indicates the default choice which can be selected alternatively by pressing the return key. words in triangular braces (< and >) are variables which evaluate for the current position into appropriate numbers and strings.

```
Home:
   o * open project
   n - new project
   a - application settings
   q - quit the application

Project:
   r * go to record
   v - go to view
   s - go to simulate
   p - edit project settings
   t - tools
   c - close project

Record>Prepare:(<phase>/<total phases>.<step>/<total steps>.↩
     ↪<repetition>/<total repetitions>)”<graphemic>”/<phonemic>/↩
     ↪<phone or diphone>
   r * record
   s - feedback to speaker
   > - step forward
   < - step backward
   p - phase advance
   e - enter step manually
   q - quit

Record>Decide>(<phase>/<total phases>.<step>/<total steps>.↩
     ↪<repetition>/<total repetitions>)”<graphemic>”/<phonemic>/↩
     ↪<phone or diphone>
   g * go on and keep
   r - repeat and keep
   d - repeat and discard
   x - go on and discard
   s - feedback to speaker
   o - feedback to operator
   c - calculate again

View:(<phase>/<total phases>.<step>/<total steps>.<repetition>↩
     ↪/<total repetitions>)”<graphemic>”/<phonemic>/<phone or↩
     ↪diphone>
   > * next
```

```
  < - previous
  p - choose position
  c - calculate again
  q - quit

Project>Tools:
  c * concatenate word
  d - delete output
  q - quit

Simulate:
  e * from external input (removes existing simulation)
  s - from existing simulation
  p - from previous input (removes existing simulation)
  q - quit

Simulate:<starting simulation step>-<ending simulation step>
  s * simulate
  b - choose start
  e - choose end
  m - change simulation mode
  q - quit

Simulate>Decide:(<phase>/<total phases>.<step>/<total steps>.↩
     ↪<repetition>/<total repetitions>)"<graphemic>"/<phonemic>/↩
     ↪<phone or diphone>
  g * go on and keep
  r - repeat and keep
  d - repeat and discard
  x - go on and discard
  s - feedback to speaker
  o - feedback to operator
  c - calculate again
```

C File Definitions

In this section the file formats of all the ASCII text files involved are documented. They all have in common, that a hash (#) at the beginning of a line indicates a comment which is ignored when the files are parsed. All variable parameters are written in the form `<name(type)>`. Further parameter descriptions might be given if needed. The symbol "␣" means any numbers of (but at least one) space and/or tab characters.

C.1 Phone List

The phone list contains lines which all have the following form:

```
<number(integer)>␣<class(integer)>␣$␣$␣.␣<phone(ETHPAstring)>␣$␣↩
   ↪\<language(character)>\␣<charactercodes(string)>
```

The parameter `charactercodes` is a string consisting of numbers with underscores in between. They are used as character codes to display the phonetic symbol by means some IPA font. A free IPA font can be downloaded from [ipa].

The phones should be listed in such an order that if two phone strings don't differ by the first character, then the longer comes before the shorter as can be observed in the example below. This is because the parsing algorithm when used on a long string of ETHPA phones lined up does not know the borders between the phones. As the searching begins at the top of the list, the shorter ETHPA phones need to be last lest the algorithm believes having found a short ETHPA when in reality a longer one is meant.

The following shows an example excerpt from a phone list:

```
# number
# |  class
# |  |  $
# |  |  | $
# |  |  | | .
# |  |  | | |      phone
# |  |  | | |      |   $
# |  |  | | |      |   |   language (\g\: german)
# |  |  | | |      |   |   |       indices into font
# |  |  | | |      |   |   |       |

  01 01 $ $ .      a:  $   \g\     097_249
  02 21 $ $ .      a_i $   \g\     097_237_105
  03 21 $ $ .      a_u $   \g\     097_237_117
  04 02 $ $ .      a   $   \g\     097
  05 02 $ $ .      ^6  $   \g\     140_057
  06 02 $ $ .      6   $   \g\     140
```

C.2 Word List for Phase 1

The word list for phase 1 contains lines which all have the following form:

```
<number(integer)>␣<targrep(integer)>␣$␣$␣.␣$␣$␣\<language(↩
   ↪character)>\␣<graphemic(string)>␣<phonemic(ETHPAstring)>
```

The parameter `targrep` is the target repetition for this entry. The character `$` can be given instead of an integer in which case the target repetition is set to `1`. The entry is completely defined by the graphemic string `graphemic`. The phonemic ETHPA string `phonemic` is optional. If two entries are found to be the same while parsing, then they are identified with the same step resulting in the target repetition for this step to be the total of all `targrep` values given for this step. The order however is preserved in the record mode.

The following shows an example excerpt from a word list for phase 1:

```
# number
# | target repetition (1 if $)
# | | $ $ .    $ $
# | | | | |    | |     language (\g\: german)
# | | | | |    | |     |       graphemic word
# | | | | |    | |     |       |               phonemic word
# | | | | |    | |     |       |               |

  1 3 $ $ .    $ $     \g\     abgeben         <?apge:b=n>
  2 $ $ $ .    $ $     \g\     Ursache         <?u:^6sax@>
  3 $ $ $ .    $ $     \g\     vernichten      <fE^6nICt=n>
```

C.3 Word List for Phase 2

The word list for phase 2 contains lines which all have the following form:

```
<number(integer)>␣<targrep(integer)>␣$␣$␣.␣<phone(ETHPAstring)>␣↩
  ↪$␣\<language(character)>\␣<graphemic(string)>␣↩
  ↪<phonemic(ETHPAstring)>
```

The parameter `targrep` is the target repetition for this entry and behaves like the in the word list for phase 1. `phone` identifies the phone whose features are to be initialized. The entry is completely defined by `phone`, `graphemic` and `phonemic`.

The following shows an example excerpt from a word list for phase 2:

```
# number
# | target repetition (1 if $)
# | | $ $ .
# | | | | |       phone
# | | | | |       |   $
# | | | | |       |   |   language (\g\: german)
# | | | | |       |   |   |       graphemic word
# | | | | |       |   |   |       |               phonemic word
# | | | | |       |   |   |       |               |

  1 2 $ $ .       a:  $   \g\     ah              <?a:>
  2 2 $ $ .       s   $   \g\     ss              <s>
  3 2 $ $ .       I   $   \g\     bist            <bIst>
```

C.4 Word List for Phase 3

The word list for phase 3 contains lines which either have the form:

```
<number(integer)>␣<targrep(integer)>␣$␣$␣.␣<leftphone(ETHPAstring)>␣↩
   ↪<rightphone(ETHPAstring)>␣\<language(character)>\␣↩
   ↪<graphemic(string)> <phonemic(ETHPAstring)>
```

or the form:

```
-1␣<leftphone(ETHPAstring)>␣<rightphone(ETHPAstring)>␣\<language(characte
```

The parameter `targrep` is the target repetition for this entry and behaves like the in the word list for phase 1. `leftphone` and `rightphone` identify the diphone to be acquired. The second form adds another diphone to the preceding carrier word and therefore cannot be the first line. The entry is completely defined by all diphones, `graphemic` and `phonemic`.

The following shows an example excerpt from a word list for phase 3:

```
# number
# |  target repetition (1 if $)
# |  | $ $ .
# |  | | | |       left half of diphone
# |  | | | |       |   right half of diphone
# |  | | | |       |   |   language (\g\: german)
# |  | | | |       |   |   |       graphemic word
# |  | | | |       |   |   |       |             phonemic word
# |  | | | |       |   |   |       |             |

  01 $ $ $ .       a:  m   \g\     die_Dame      <di:da:m@>
  02 $ $ $ .       n   e:  \g\     benehmen      <b@ne:m@n>
  -1               e:  m
  03 $ $ $ .       /   a:  \g\     ahnen         <a:n@n>
```

C.5 Project Creation File

The project creation file contains lines all of which have the following form (mark the delimiting "`;`" character):

```
<fieldname(string)>;␣<fieldvalue(string)>;␣<fieldformat(string)>
```

The file is used to initialize a structure with the fields `fieldname` having the values `fieldvalue` in the formats `fieldformat`. The last argument must be either "`%f`" which indicates a numerical value or "`%s`" which indicates a string. The following fields are mandatory since they are used to create the project directory and file structure:

`parentdir` This is the parent directory in which the project directory will be created. If a relative path is given it is interpreted relative to the Matlab working directory.

`name` This is the project name. The string should qualify as a directory and file name string.

`phonelist` A file name indicating the location and name of the phone list file to be used for this project.

`wordlist1` A file name indicating the location and name of the word list file for phase 1.

`wordlist2` A file name indicating the location and name of the word list file for phase 2.

`wordlist3` A file name indicating the location and name of the word list file for phase 3.

All lists will be copied to the projects `extinp` directory and renamed.

As optional fields, all fields of the project settings structure `Data.Set` can be given. (See section 5.3 for a description of this structure.) Default values will be assigned to fields which are not given. Here is an example of a project creation file:

```
# project creation information
parentdir;     ../projects/;                     %s
name;          test;                             %s
phonelist;     ../templatelists/phonelist.txt;   %s
wordlist1;     ../templatelists/wordlist1.txt;   %s
wordlist2;     ../templatelists/wordlist2.txt;   %s
wordlist3;     ../templatelists/wordlist3.txt;   %s

# project settings
fs;            16000;                            %f
bps;           16;                               %f
specMethod;    mfcc;                             %s
distance;      SKL;                              %s
```

C.6 Input Log File

The input log file contains any repetition of the following five lines:

```
<number>␣<phase>␣<step>
-1␣file␣<filename>
-1␣uttdet␣<uttdet>
-1␣tidet␣<tidet>
-1␣keep␣<keep>
```

In the first line, `number` gives a running number for the input and `phase` together with `step` identify the position. In the current implementation `filename` will have the form `<number>.wav` and `uttdet` as well as `tidet` are always `1`. `keep` can be `1` or `0` depending on whether the recording was kept or discarded. Comments about the session number and the date and time are interspersed as can be seen in the following excerpt of an input log file:

```
# Automatically generated input log file

# 25-Jun-2005 16:40:26 Project generation successfully terminated
# 25-Jun-2005 16:40:26 session number: 1
0001    1       0001
-1      file    0001.wav
-1      uttdet  1
```

```
-1      tidet   1
-1      keep    1
0002    2       0001
-1      file    0002.wav
-1      uttdet  1
-1      tidet   1
-1      keep    0
```

C.7 Simulation Sequence File

Similar to the input log file, the simulation sequence file contains any repetition of the following lines:

```
<number>␣<phase>␣<step>
-1␣file␣<filename>
-1␣start␣<start>
-1␣end␣<end>
-1␣uttdet␣<uttdet>
-1␣tidet␣<tidet>
-1␣keep␣<keep>
```

However, the three last lines are optional, in which case `uttdet` is assumed to be `0`, `tidet` is assumed to be `0` and `keep` is assumed to be `1`. See section C.6 above for a description of these fields. If `uttdet` or `tidet` is `1` then the corresponding word label or centroid/diphone label must be present in a label file.

The parameters `start` and `end` indicate the starting and ending sample of this recording step. If they are not given and when simulating from external input, the simulation first tries to locate words in a corresponding label file and tries to match them. If this fails then the whole wave file is taken as input.

Also for all lines starting with `-1`, the order does not matter. The following shows an excerpt of a simulation sequence file:

```
0001    2       0006
-1      file    ger_dp_m_1.wav
-1      start   000465002
-1      end     000468480
0002    2       0006
-1      file    ger_dp_m_1.wav
-1      start   000493122
-1      end     000496880
0003    2       0002
-1      file    ger_dp_m_1.wav
-1      start   000675722
-1      end     000680440
0004    2       0002
-1      file    ger_dp_m_1.wav
```

```
-1      start   000703802
-1      end     000707400
```

D Thesis Specification by the Institute

Sommersemester 2005

SEMESTERARBEIT

für

Herrn Christoph Reller

Betreuer:	H. Romsdorfer	ETZ D97.5
Stellvertreter:	Dr. B. Pfister	ETZ D97.6

Ausgabe:	29. März 2005
Abgabe:	7. Juli 2005

Überwachungsprogramm für die Aufnahme von Diphonkorpora

Einleitung

Die Sprachsignalerzeugung in Sprachsynthesesystemen basierend auf dem Verkettungsansatz erfolgt durch Aneinanderfügen geeigneter Segmente, so genannter Grundelemente, aus vorhandenen, natürlichen Sprachsignalen. Der Vorteil dieses Ansatzes ist, dass man auf der Ebene dieser Signalelemente absolut natürliche Sprache hat, und somit einen wesentlichen Teil der Probleme anderer Sprachsignalproduktionsansätze vermeidet.

Um abrupte Lautübergänge zu vermeiden, müssen die Grundelemente alle möglichen Lautübergänge beinhalten. Sehr häufig werden Diphone als Grundelemente verwendet. Ein Diphonelement ist dabei ein Signalausschnitt, der in der Mitte der quasistationären Phase eines Lautes beginnt und in der Mitte des Folgelautes endet.

Die Qualität dieser Sprachsignalerzeugung hängt wesentlich von der lautlichen Qualität der einzelnen Diphonelemente und vom Mass der Übereinstimmung an den Grenzen aufeinander folgender Diphone ab. Da jede Modifikation der Diphone die resultierende Sprachqualität verschlechtert, wäre es am besten, direkt bei der Aufnahme der Diphone die gewünschte lautliche Qualität und das Übereinstimmungsmass automatisch kontrollieren und gegebenenfalls korrigieren zu lassen.

Problemstellung

Grundsätzlich stellen sich an ein Verfahren zur interaktiven Überwachung der Aufnahme von Grundelementen administrative und signaltechnische Anforderungen:

- Da die Grundelemente selbstverständlich alle möglichen Lautfolgen einer Sprache darstellen können sollen, ist die Anzahl der notwendigen Grundelemente grundsätzlich $(N+1)^2 - 1$ mit $N =$ Zahl der Laute.

  Da für jedes Grundelement ein geeignetes Trägerwort vorhanden sein muss, aus dem das entsprechende Segment ausgeschnitten werden kann, muss das Programm die Liste der Trägerwörter, die dazugehörenden Signalaufnahmen, die im Trägerwort vorhandenen Grundelemente, und die Frage, ob dieses Trägerwort eventuell nochmals wiederholt werden muss, geeignet verwalten können.

- Um für verschiedene Sprachen einsetzbar zu sein, muss die Liste der möglichen Laute bzw. Grundelemente, der Trägerwörter, usw. frei definierbar sein. Dazu ist eine strikte Trennung von Programmfunktionalität und Daten notwendig.

- Die Schnittstellen der Grundelemente sollen grundsätzlich so festgelegt werden, dass bei einer späteren Verkettung die spektrale Diskontinuitäten minimal werden. Dazu wäre ein Verfahren zur globalen Minimierung der Diskontinuitäten an allen potentiellen Schnittstellen denkbar. Für ein Verfahren jedoch, dessen Ziel es ist, den Sprecher nach jedem gesprochenen Wort korrigieren zu können, ist eine globale Optimierung nicht möglich. Hier ist für Laute mit quasistationärer Phase der Einsatz von Lautzentroiden denkbar, wie er in [PHB05] näher erklärt wird. Für die übrigen Laute, wie z.B. Plosive, kann man dazu andere lautliche Eigenschaften verwenden (siehe auch [PHB05]). Das dafür notwendige Set von Lautzentroiden kann in einem ersten Aufnahmeschritt mittels spezieller Trägerwörter initialisiert werden, und eventuell im Laufe der Aufnahmen weiter optimiert werden.

- Um bei der Aufnahme der Trägerwörter die Verwendbarkeit des geraden aufgenommen Wortes beurteilen zu können, kann man z.B. die cepstrale Distanz zu den entsprechenden Lautzentroiden als Mass verwenden. Auch sollte die Grundfrequenz der Diphonelemente möglichst übereinstimmen und konstant sein. [KV01, WM98] vergleichen mehrere spektrale Distanzmasse zur Ermittlung von hörbaren Diskontinuitäten in der Diphonsynthese miteinander.

- Um dem Sprecher Hinweise zur Verbesserung der Aussprache geben zu können, könnte man z.B. über das LPC-Spektrum auf die Form des Vokaltraktes zu schliessen und daraus Anleitungen zur Aussprache formulieren.

Im Rahmen dieser Semesterarbeit soll nun ein Programm in MATLAB entwickelt werden, das Plattform unabhängig eine interaktive Aufnahme von Diphonelementen entsprechend den oben genannten Anforderungen ermöglicht.

Dabei ist ein Teil dieser Aufgabe die Erstellung eines Konzepts für die Implementierung des Aufnahme-Tools in MATLAB. Darin soll speziell auf die korrekte Trennung der Programmfunktionalität von Daten wie etwa Lautinventar, Diphonträgerwörterliste usw. geachtet werden.

Um die Funktionalität des Programms testen zu können, ist es erforderlich, die direkte Aufnahme von Sprachsignalen über das Mikrofon durch eine Simulation ersetzen zu können. Eine solche Simulation kann z.B. mittels einer frei definierbaren Sequenzliste von bereits aufgenommenen Sprachsignalen von Diphonträgerwörter erfolgen. Der Programmablauf sollte natürlich durch die Simulation nicht verändert werden.

Ein möglicher, grob schematisierter Programmablauf ist in folgender Liste skizziert:

- Einlesen und Validierung des Lautinventars, der Liste mit Diphonträgerwörter, Simulationssequenz, usw.
- Initialisierung eines vollständigen Sets von Lautzentroiden.
- Iterative Aufnahme der Trägerwörter:
  - Aufnahme des Sprachsignals eines Trägerwortes.
  - Bewertung der Qualität der an den Diphonen beteiligten Laute.
  - Rückmeldung bei qualitativ schlechtem oder fehlerhaften Diphon, und erneute Aufnahme des entsprechenden Trägerwortes.
- Festlegung der Lautmitten zum Ausschneiden der Diphone.
- Auswahl eines optimalen Sets von Diphonen aus allen aufgenommenen Varianten.

Aufgaben

1. Einarbeitung in die Literatur [PHB05, KV01, WM98].
2. Erstellung eines Grundkonzepts. Dabei sollen bereits Sprachunabhängigkeit und die Testbarkeit (mittels Simulation der Aufnahme) berücksichtigt werden.
3. Implementierung in MATLAB auf SUN Solaris.
4. Versuche mit eigenen Audioaufnahmen.
5. Simulation mit Audioaufnahmen eines professionellen Sprechers.
6. Die ausgeführten Arbeiten und die erhaltenen Resultate sind in einem Bericht zu dokumentieren, der in zwei Exemplaren abzugeben ist, wovon eines Eigentum des Instituts bleibt.

Zürich, den 24. März 2005

Prof. Dr. L. Thiele