

Amplitude normalisation and intonation continuity modelling for unit selection

Diplomarbeit Nr. 50 im Fach Computerlinguistik

Andreas Madsack

Prüfer: PD Dr. Bernd Möbius
Betreuer: Martin Barbisch
PD Dr. Bernd Möbius

Begin: April 01, 2006
End: September 18, 2006

Universität Stuttgart
Institut für Maschinelle Sprachverarbeitung
Azenbergstraße 12
D-70174 Stuttgart
Germany

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text.

September 18, 2006 / Stuttgart

(Andreas Madsack)

Abstract

This thesis shows two attempts to improve synthesis quality in unit selection.

The first attempt is to normalise the amplitude of the wave output by using signal processing. For validation of the amplitude normalisation with 35 subjects, a perception experiment was conducted. No significant positive effect for using amplitude normalisation was found. Most subjects in the experiment only heard the few signal processing failures, but in most cases they cannot distinguish between normalised and unchanged stimuli.

The second attempt of improvement is to add a pitch cost function for concatenation. The results were tested with three evaluation ideas. All three verify that the pitch cost function has an impact on the synthesized speech. But without a perception experiment, which was not made due to time constraints, it is not possible to judge whether the changes lead to a perceivable improvement.

Acknowledgements:

At first I want to thank my supervisors Martin Barbisch and Bernd Möbius for their helpful advice.

I am also thankful to Antje Schweitzer for answering questions on Festival.

Moreover, I want to thank especially Ines Flechsig and Vera Demberg for proof-reading and answering numerous questions on English grammar. Additionally, my thanks go to Ralf Jankowitsch for his help with some L^AT_EX quirks.

Last, I want to thank Lenchen for always being there.

Contents

1. Introduction	1
1.1. Intonation	1
1.2. Festival	2
2. Amplitude Normalisation	4
2.1. Introduction	4
2.2. Energy Data Generation	4
2.3. Use in Festival	7
2.4. Perception Experiment	8
2.5. Results	11
2.6. Conclusion	14
3. F0 cost function	16
3.1. Introduction	16
3.2. Optimizing f0	16
3.3. Cost function	17
3.3.1. Implementation in Festival	17
3.3.2. Calculating <code>f0WeightFactor</code> for a unit selection voice	18
3.4. Creating a testbench using a brute force approach	19
3.5. Evaluating the cost function	20
3.5.1. Comparing f0 with PaIntE	20
3.5.2. Results of the comparison between f0 and PaIntE	22
3.5.3. F0 curve smoothness	26
3.5.4. Results of the f0 curve smoothness calculations	26
3.5.5. Calculating differences in psm results	28
3.5.6. Results of the differences in psm results	28
3.6. Conclusions	29
4. Summary	30
4.1. Amplitude normalisation	30
4.2. F0 cost function	30
Bibliography	31
List of Figures	33
List of Tables	34

A. Appendix	35
A.1. Plottings of energy values	36
A.1.1. The male speaker of SmartWeb	36
A.1.2. The female speaker of SmartWeb	37
A.2. Scripts	38
A.2.1. playnow.sh (German texts)	38
A.2.2. getpower.py	41
A.2.3. genmeanfile.py	44
A.2.4. genexp.sh	45
A.2.5. gen_bench.py	47
A.2.6. genR.py	51
A.2.7. evaluation.py	53
A.2.8. genAMP-graphics.py	57
A.2.9. genf0-graphics.py	58
A.3. Code	61
A.3.1. amplitude normalisation (part of ims_post_sigv.cc)	61
A.3.2. framedistance (part of ims_psm_tools.cc)	67
A.3.3. psm drop candidate code (part of ims_post_sigv.cc)	68

1. Introduction

Speech synthesis has highly evolved in the last decades. From rule based systems like formant synthesis in the 80s over diphone synthesis in the 90s to unit selection today. But the main problem persisted the same over all the years: the lack of naturalness. An important part for natural sounding speech is intonation.

The task for this work was to test improvements in the intonation part of the unit selection in Festival (cf. section 1.2). One of the tested improvements was adding an amplitude normalisation after synthesis. The amplitude normalisation was achieved by using signal processing on the resulting wave. The necessary steps to achieve this goal are described in chapter 2. Later, in section 2.4, a perception experiment is described which was used to evaluate the amplitude normalisation modification.

A second task was to improve the pitch continuity in unit selection. Chapter 3 explains how the author added a f_0 cost function to Festival. Additionally three ideas for evaluation of the new cost function were tried.

1.1. Intonation

Intonation is a major part of speech. Without intonation, speech would only be a chain of sounds. But speech can be whispered, creaky or can start in a quiet voice and can get louder towards the end of the utterances.

The main features of intonation which make it possible to control and modulate the utterance, are *pitch*, *length*, and *loudness* (cf. Cruttenden 1986). Of these three features, *pitch* is the most important. With pitch, a sentence transports meaning beyond its semantic content (Kamp & Reyle, 1993). From the human perspective *pitch* is produced in the larynx. The produced signal directly correlates with the fundamental frequency (f_0), which is between 100Hz and 220Hz for male speakers and 200Hz to 450Hz for female speakers (cf. Hakkarainen 1995, 33). In speech synthesis and especially in unit selection, the target is to generate or select a natural sounding fundamental frequency.

The intonation feature *length* seems simple, but is in some way tricky. The problem is to decide what the *length* of a phoneme, a syllable or even a full sentence is. For example, is the lengthening of a vowel the distinction between the shorter and the longer version of a vowel or is the lengthening part of the intonation? There are statistics and models for this problem, but it remains nevertheless difficult. Even in unit selection, where the units are chosen and the length should not be a problem, not all durations are perfect.

The third feature of intonation is *loudness*. *Loudness* is controlled by the amount of air that flows from the lungs through the larynx. It is difficult to say how important

loudness is for the perception of intonation or even for the perceptual impression of the quality of speech synthesis.

1.2. Festival

Festival (Black et al., 1999; Taylor et al., 1998) is a state of the art speech synthesis system developed at the University of Edinburgh and first introduced in the mid 90s. It is mainly used in research for example at University of Edinburgh, University of Stuttgart (Institut für Maschinelle Sprachverarbeitung (IMS)), Carnegie Mellon University (CMU) and others. Festival is open source and has a very modular architecture.

These modules are executed successively. Figure 1.1 shows the main steps in speech synthesis. The important parts for this thesis are the last two steps.

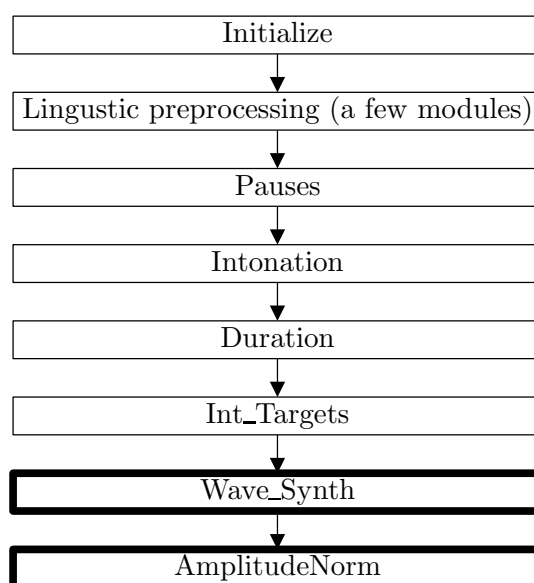


Figure 1.1.: Part of the synthesis sequence in Festival

The current project using Festival at the IMS is SmartWeb. SmartWeb uses an improved version of the phonological structure matching algorithm (psm) (Taylor, 2000; Taylor & Black, 1999; Black & Taylor, 1997) and the clustering algorithm, developed for SmartKom (Schweitzer et al., 2003; Klankert, 2003). The advantages of phonological-structure-matching are good results in limited domain use and the possibility to be open domain using the clustering algorithm. The results for open domain use are significant of a lesser quality than the limited domain use. The main part of the psm and cluster algorithm take place in the `Wave_Synth` step (cf. figure 1.1) which is normally the last step in the Festival synthesis system. The important element of the psm algorithm is a tree search function, which searches for candidates in the corpus. First for phrases, second

for words, then syllables and at last for segments. When enough e.g. words are found, no syllables are added to the candidate list. This candidate list is normally pruned for a specified number of items. Using the A*-algorithm and an optimal coupling calculation the best candidates are chosen.

2. Amplitude Normalisation

2.1. Introduction

Some phonemes are unnaturally loud or too quiet in the unit selection voice used in the SmartWeb project. SmartWeb is a research project sponsored by the German Federal Ministry of Education and Research. The Institut für Maschinelle Sprachverarbeitung at the University of Stuttgart is part of this project and responsible for the speech synthesis part of the project.

To reach a more naturally sounding loudness the amplitudes of the synthesised speech were normalised. The new amplitudes were calculated by using the means from a large corpus. For normalisation, after synthesis in Festival, at first energy data was extracted from all data of this speaker. A new routine is added to Festival which normalises every synthesised output. Finally, the resulting data was evaluated in a perception experiment.

2.2. Energy Data Generation

SmartWeb uses two unit selection voices: a male and a female one. For these two voices the loudness values vary because of the recording modulation. After a loudness change of the voice a new energy data calculation is required. Seeing that all speakers vary in their energy values and energy values were extracted for each speaker separately. The data used in Festival was generated by means of a python script (cf. A.2.2). This script extracts the energy curve from every wave in the corpus of the male SmartWeb voice using `sig2fv` from the `speech-tools` (Taylor et al., 1999). In this case, the energy curve is normalised using a Hanning window of 0.03 seconds length every millisecond and using root mean square error (RMSE). RMSE is the square root of the MSE, where the latter one is a good estimator for variance of values. The formular for RMSE is

$$x_{rms} = \sqrt{\frac{1}{N} \sum_{i=1}^N x_i^2}.$$

For every processed wave file, the matching label file is used to read the phoneme names and their start and end position into the wave file. The format of the label files is the xwaves semi-standard, which saves the ending time-stamp and the phoneme-code in one line each, for example `1.14557 121 a`. The number in the middle is a colour code, which is only used in xwaves (tool from Entropic). With the information from the label files, the generating script extracts the corresponding energy stream data from `sig2fv` for every phoneme in the wave. After this split, five points in recording time from each

phoneme segment are selected: at 10%, 25%, 50%, 75% and 90% of duration (cf. 2.1).

The attained energy values are written to a datafile for each phoneme together with a comment that indicates the wave filename from which the respective phoneme originates. A line in such a file may look like as follows

```
684.98761 855.60809 960.57623 929.98248 821.60553 # f068.wav.
```

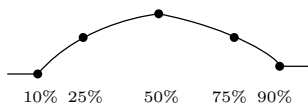


Figure 2.1.: Five point time plot

Another script calculates the mean value of all five values for each phoneme (cf. A.2.3). This calculation is a simple mean, i.e. every column is added to a column total and divided by the column count ($\frac{\sum v_i}{|z|}$). The result for every phoneme is written in Scheme syntax. This file which contains five values per phoneme, has to be readable in Festival and has to be added to Festival's lib directory. The energy data is placed in the appropriate voice definition in `ims_german_voices.scm`.

A further part of this investigation is the distribution of these five values over all phonemes. Figure 2.2 presents a few values from selected phonemes. The labels in the plots are in Sampa¹ except for one label which starts with a Q. The Q denotes a glottal stop. The plots show the five different possible values. The dimension for loudness in this plot is between 0 and 7277. Every box has 50% of the values, and both terminated lines span 90% of the values. The small circles are outliers. The line in the box is the median of the values, not the mean which was used for calculation of the five points by the python script (cf. A.2.3).

For example, for the phoneme /a/ at the upper left in the plot, most of the values are between 3000 and 4000. The median is between 3100.07 for the last box and 3435.0 for the second box. In the second plot the vowel /a/ is preceded by a glottal stop. The glottal stop is not as loud as the /a/ which is expected. The third plot shows a diphthong which consists of the vowel /a/ and the vowel /i/. For the first one most values are between 2000 and 3000 and for the second one the values are a lot more varied. The big variation may be the result of the loudness of the following phonemes. The fourth plot in the first row is the phoneme /u/. In contrast to the vowel /a/ in the first plot the values for the vowel /u/ do not vary this much. In the second row a few fricatives are shown. The plots indicate that fricatives are not as loud as vowels. In contrast to the third row which consists of three nasals and the phoneme /h/. Each nasal phoneme has five different median values. The last plot in figure 2.2 shows the variety of the the phoneme /h/. The phoneme /h/ is produced by evoking air turbulence at the glottis which undergoes changes from organs like tongue or lips (cf. Kohler 1977, p. 160). The

¹<http://www.phon.ucl.ac.uk/home/sampa/>

/h/ in the last plot is affected by formant transitions that pass over to the /h/ and show the high context-sensitivity of /h/.

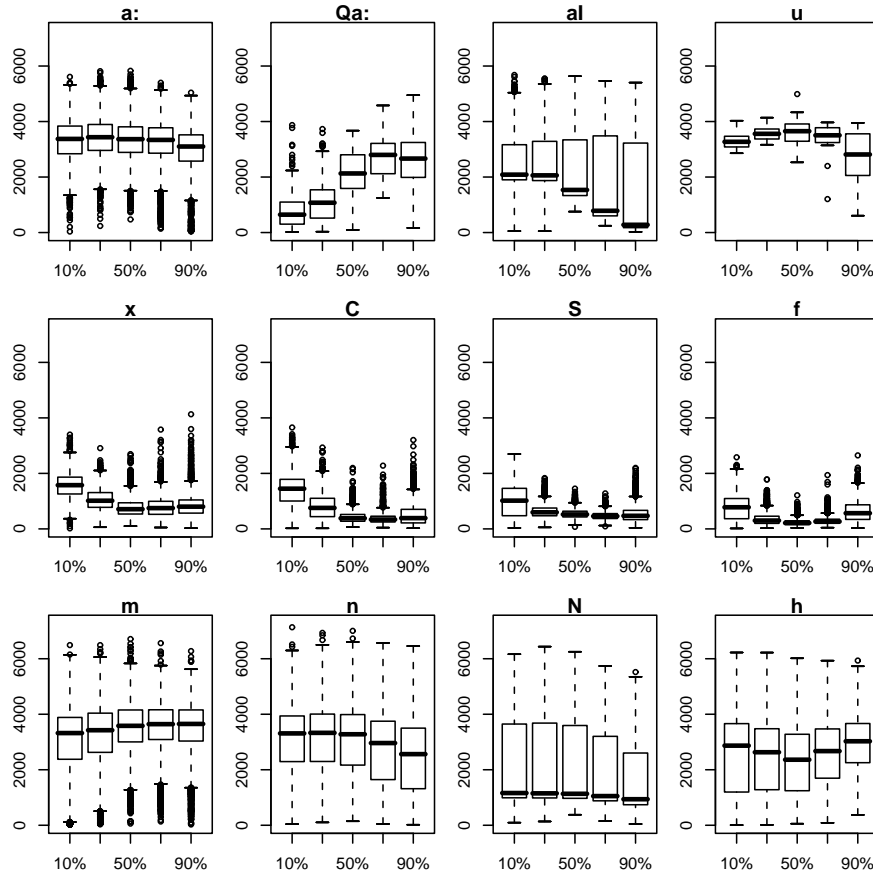


Figure 2.2.: Boxplots of energy values for selected phonemes

In appendix A.1 on page 36 the most important phonemes are plotted. Figure A.1 shows the plots for the male SmartWeb voice and figure A.2 for the female SmartWeb voice. The amplitude values of the female voice are much lower than of the male voice because the waves of the male voice are louder. The maximum of all energy values for both voices were calculated and used for the vertical maximum of the plots to make them comparable. The maximum for the female voice is 3367 and for the male voice as listed above 7277.

2.3. Use in Festival

The five data means which are generated by the python script (cf. chapter 2.2) were used in Festival. The amplitude normalisation function is written in C++ (cf. code in A.3.1) like most of the Festival code.

Amplitude normalisation took place after *Wave_Synth*, which is normally the last step in speech synthesis, as shown in figure 1.1 on page 2. It had to take place at such a late state of the process because the unit selection part needs to be finished before normalization takes place. Thus the normalisation manipulates the wave that has been entirely processed by Festival. A factor for each sample of the final wave is calculated by the amplitude normalisation function. For pauses and plosives this factor is 1.0, since a change of the amplitude of these two groups is not necessary. Pauses have no energy and it would be problematic to calculate an energy value for them. Plosives on the other hand are hard to normalise, because they consist of different parts, which are a pause part, a burst and friction. Normalisation would not improve the quality of plosives.

For all other phonemes, the factor is calculated using the five interval values. The calculated value is divided by the measured value of the wave. All values between the calculation points (10%, 25%, 50%, 75% and 90%) are linear interpolated. At the end of a phoneme (90%–100%) the energy value of the following phoneme is chosen to accomplish the interpolation. For the beginning of a phoneme (0%–10%) the energy value of the preceding phoneme is used respectively. The last segment of the phoneme preceding the final pause was not resampled and was assigned the factor 1.0. A resampling of the last phoneme would result in an unnatural amplitude at the end of the phrase.

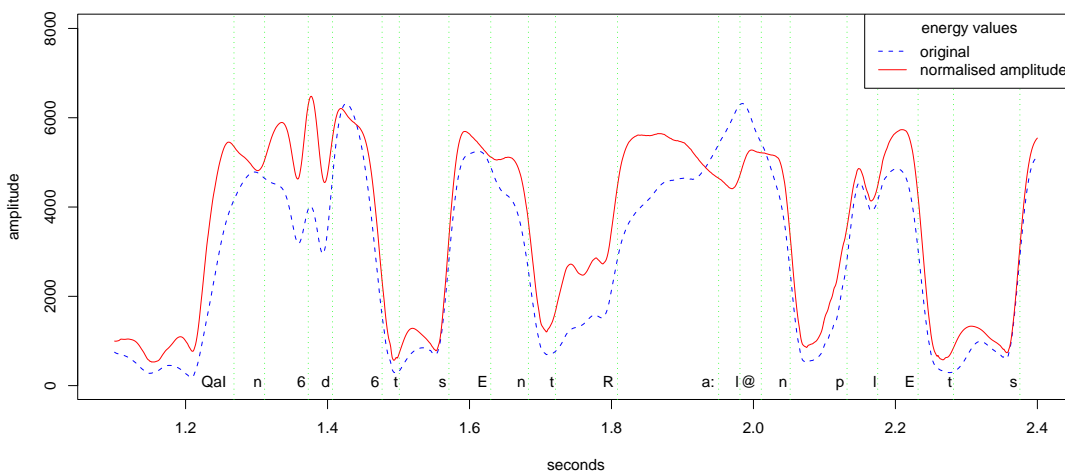


Figure 2.3.: Energy plot of beginning of sentence 17

The whole normalisation is scaled to the maximum amplitude of the original, unmodified wave. This means that the amplitude maximums of the normalised and the original wave are identical. A further advantage of this is that both waves are comparable by a subject in a perception experiment without noticing a completely different loudness.

See figure 2.3 for an example.

The amplitude normalisation function can be activated using the Festival script language by calling (*setq ampnorm_activate t*) in the voice definition and adding a statement in the definition of *deftype Text* after (*Wave_Synth utt*) which is (*AmplitudeNorm utt*).

2.4. Perception Experiment

For a perception experiment 19 sentences and six words were chosen. The six words are concatenated of different syllables, i.e. only the syllables but not the whole words are present in the corpus.

Three of the sentences are from classic German literature. The first sentence is from Goethe's *Der Erlkönig* (English: *Elfin King*), the second from Schiller's *Die Glocke* (English: *The bell*) and the third one from Brother Grimm's *Jorinde und Joringel*. The remaining 16 sentences are contained in the unit selection corpus, but only one (sentence 21) was selected as a whole by the psm-algorithm (cf. section 1.2). The words and sentences are listed in table 2.1.

```
Tastenkommados
-----
| ESC   Programm beenden
| 1     Datei 1 abspielen
| 2     Datei 2 abspielen
| n     naechste Wav
| --
| Bewerten:
| j     Datei 1 klingt besser als Datei 2
| k     Datei 2 klingt besser als Datei 1
| l     Datei 1 klingt gleich gut wie Datei 2
-----
aktuelle Wav: 018233472611a4d352b18f577626366dX
Wav 1 von 50
■
```

Figure 2.4.: Screenshot of the experiment (German)

Every word or sentence is used twice in the experiment. The first and the second test with 25 sentences each are not mixed with each other. The experiment results in 50 trial pairs, from whom the subjects have to choose. The variation of the two stimuli was different which means one was not modified and one was amplitude normalised or both were the same. In the experiment, which was generated by a script (cf. A.2.4) a random function was used to generate the test sequence. Whereas normalised versus original and

original versus normalised pairs were two times more frequent than normalised versus normalised and original versus original pairs. Although all sentences appeared two times in the experiment, they did not in the same test condition (normalised versus original, original versus normalised, normalised versus normalised or original versus original). The pairs with identical stimuli were used as control sentences.

For the experiment, a bash script was used that could be run on Windows and Linux equally. This script was not accessible via a website. It had to be executed in a terminal (cf. screenshot in figure 2.4).

The subjects had the choice of three conditions: Either wave no. 1 was better than wave no. 2 or wave no. 2 was better than wave no. 1 or both wave files seem identical. The subject could play the two waves as often as required to come to a decision. All interaction was recorded automatically into a log file, which was e-mailed by each subject to the author.

Subjects have been only instructed to choose pairwise between two stimuli with respect to decide on the best if a distinction was perceptible. The subjects have not been instructed that signal processing errors may be present, e.g. cracks, or to rate sounds that were equally good as equal. The result was a complete difference in the rating method of the subjects: One group rated a stimuli pair as equal when the stimuli seemed to have a similar quality and no signal processing errors were perceptible. The other group was more critical in the way that they seemed to have tried hard to find any perceptible difference in a stimuli pair. They seemed to only have rated equality when they were convinced that indeed no difference was present. The subjects were not told that the stimuli in this experiment differed in that one stimuli group was amplitude normalised whereas the other was not. The aim of the experiment was to show whether the amplitude normalised or the unmodified wave were perceived. Another aim was to see whether the signal modifications in the amplitude normalised wave were perceived.

The experiment consisted of 2 cycles each comprising the 25 stimuli pairs in random order. Moreover, the pairs themselves were newly shuffled for each of the two cycles. Due to a scripting error, which was fixed as soon as it was detected, some subjects conducted a unfixed version of the experiment, that is the second cycle of a stimuli pair followed directly after the first cycle. However, the error is not supposed to affect the perception of the subjects significantly. Another problem was, that two stimuli pairs (no. 12 and no. 18) were tested only once by some subjects. This results in the lower play counts for these two sentences.

- words
 1. Salihamidzic
 2. Schwanenkostüms
 3. Waffengewalt
 4. Fensterputzer
 5. Flaschenabfüllung
 6. Erschießungskommando
- longer sentences
 7. Fest gemauert in der Erden \ steht die Form, aus Lehm gebrannt.
Heute muß die Glocke werden. \ Frisch, Gesellen! seid zur Hand.
Von der Stirne heiß \ Rinnen muß der Schweiß.
 8. Wer reitet so spät durch Nacht und Wind?
Es ist der Vater mit seinem Kind.
Er hat den Knaben wohl in dem Arm,
Er faßt ihn sicher, er hält ihn warm.
 9. Es war einmal ein altes Schloß mitten in einem großen dicken Wald, darinnen
wohnte eine alte Frau ganz allein, das war eine Erzzauberin.
- corpus
 10. Das Zentrum blieb zumeist Cardoso vorbehalten, weil Spörl sich deutlich nach
rechts orientierte.
 11. Dabei hatte Störzenhofecker gegen Chapuisat große Probleme, während Van
Eck Salou weitgehend beherrschte.
 12. Lichtblicke gab es durchaus der Düsseldorfer Humangenetiker Majewski aller-
dings gehörte nicht zu ihnen.
 13. Zunächst kam ein 43jähriger Autofahrer mit seinem Personenwagen auf der
Brücke ins Schleudern.
 14. Von den vier Stürmern der Anfangsformationen konnte lediglich Mpenza
Akzente setzen.
 15. Hier standen die renommiertesten Hotels und die feinsten Gasthäuser.
 16. Der Brunnen am Mainzer Tor wurde 1987 von Gernot Rumpf geschaffen.
 17. Die Hauptwache ist einer der zentralen Plätze der Mainmetropole.
 18. Um etwa alle Patienten eines Arztes zu finden, sucht man zuerst den Arzt
über einen Primärschlüssel.
 19. Die Gastgeber kombinierten nach Belieben, gewannen die Zweikämpfe und
waren auch spielerisch deutlich überlegen.
 20. Dort lebten damals viele Arme unter denkbar schlechten Bedingungen.
 21. Hier wohnten früher die Kaufleute und Handwerker.
 22. Goethe hat in seinem Leben viele Frauen geliebt.
 23. Legion prüft daher jedes Angebot, bevor es on air geschaltet wird soviel die
Theorie.
 24. Die Bremer Defensive wirkte anfangs noch etwas unsortiert, agierte aber viel-
beinig, machte die Räume eng.
 25. Todt, zuletzt in der Halbposition, übernahm die rechte Außenbahn, Trares
(gegen Lautern Libero), die linke Seite.

Table 2.1.: All words and sentences occurring in the perception experiment.

2.5. Results

35 people participated in the experiment. Nine of them were female and 26 were male. Some of them work at the institute and are experienced with phonetics. The experiment was not actively supervised. The log file every subject mailed did not only contain the judgement of each pair, but also the play count and the time used for every utterance. Figure 2.5 shows how often each utterance was played. The minimum number of play counts is two because sentences pair no. 12 and sentence pair no. 18 were only tested once (cf. section 2.4) for some subjects in the experiment, which was fixed after a few subjects participated. The minimum play count for the other sentences is 4.0. On the average, sentences have been played 8.14 times by subjects before a decision was made. This means that in average every stimuli is played twice by the subjects. Without sentence no. 12 and sentence no. 18 the *mean* is somewhat higher with a play count of 8.31.

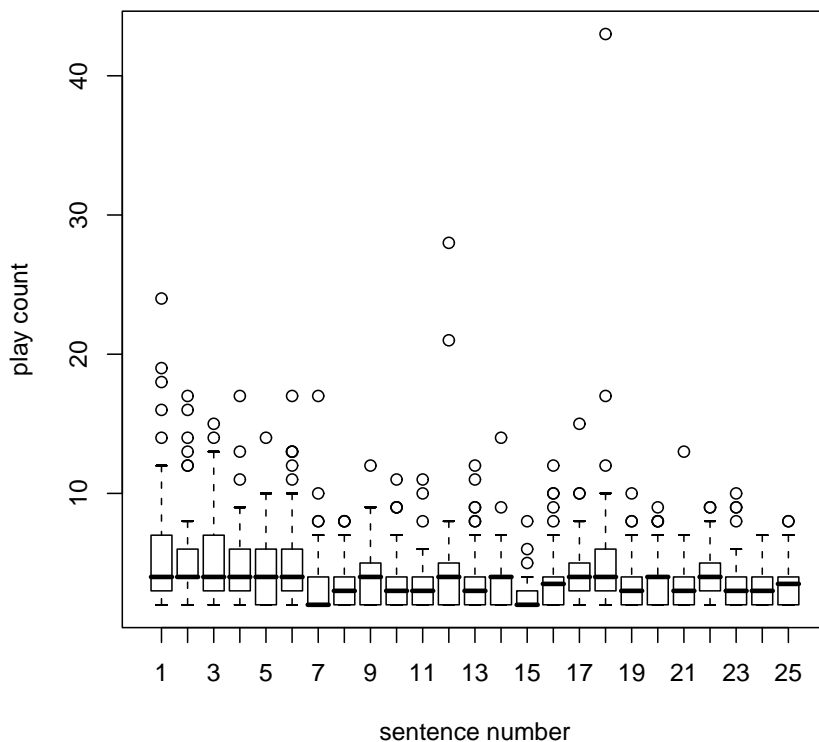


Figure 2.5.: Plays per sentence

Table 2.2 shows all subjects and their scores. The scores in the table are amplitude

normalisation (abbreviated by **a**), identical (abbreviated by **i**) or original (abbreviated by **o**). The **+** in the scores denotes that the rating is possible and the perception was correct. For example an **a+** rating means, that amplitude normalisation was preferred and the two stimuli were different. If **a-** is present, the two stimuli were the same. For original (**o+** and **o-**) the declaration is analogue. **i+** and **i-** are slightly different. For these two values it is possible to say whether they are correct. Either the two stimuli were the same, or not. In the last column, the values for **a-** and **o-** are summarised. These are false alarms, for the sentences were identical and the subject voted for difference. The other minus value (**i-**) is not as problematic because in that case the subject rated the stimuli as identical but the stimuli were different.

For illustration:

pairs	utterance 1 is better	utterance 2 is better	both are equal
a - a	a-	a-	i+
o - o	o-	o-	i+
a - o	a+	o+	i-
o - a	o+	a+	i-

Different sentences rated identical shows that the modification has neither positive nor negative effect. Consider person D1: D1 voted only nonidentical, when he perceived obvious quality differences. Instead, most other subjects seem to have searched for errors. The mistake from the author was that no instructions were given on how to approach this experiment. This caused the high counts of **a-** and **o-**. The opposite of D1 seems to be M5: M5 perceived in nearly all stimuli a difference and rated only five pairs as identical. The figure 2.6 shows the variance of the six experiment variables (**a+**, **a-**, **i+**, **i-**, **o+**, **o-**). It seems obvious that most subjects preferred the original, i.e. the unmodified, wave over the amplitude normalised wave.

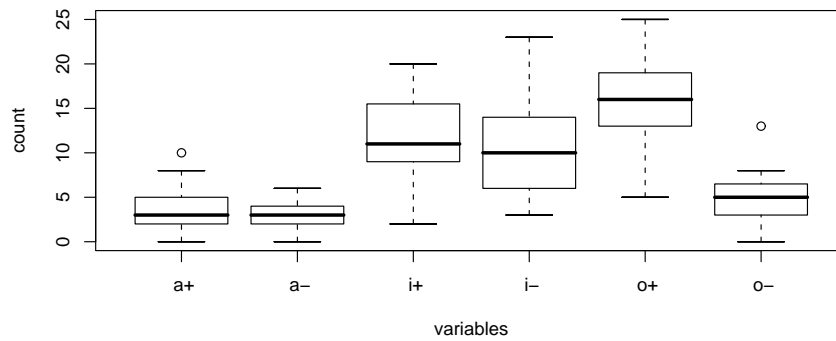


Figure 2.6.: Variance of the six experiment variables according to table 2.2

	User	a+	a-	i+	i-	o+	o-	a-&o-
1	A1	3	2	17	14	13	1	3
2	A3	2	1	19	19	9	0	1
3	B1	3	3	12	6	21	5	8
4	B2	3	4	9	14	13	7	11
5	C1	4	3	9	11	15	8	11
6	C2	5	2	15	11	14	3	5
7	D1	0	0	20	21	8	0	0
8	D2	3	2	12	15	12	6	8
9	D3	0	4	11	5	25	5	9
10	H1	3	3	9	6	21	8	11
11	I1	1	3	13	6	22	4	7
12	J1	8	3	11	9	13	6	9
13	K1	5	5	8	8	17	7	12
14	K2	5	4	11	3	21	5	9
15	M1	1	1	16	18	10	3	4
16	M2	1	1	19	23	6	0	1
17	M3	4	4	9	10	16	7	11
18	M4	4	4	11	6	20	5	9
19	M5	10	5	2	3	16	13	18
20	M6	6	6	9	5	19	5	11
21	M7	2	4	9	14	14	7	11
22	M8	2	3	12	9	19	5	8
23	M9	3	0	18	21	5	2	2
24	N1	2	0	19	16	11	1	1
25	N2	3	4	10	7	20	6	10
26	R1	7	6	9	5	18	5	11
27	R2	5	5	10	8	17	5	10
28	S1	0	1	16	16	13	3	4
29	S2	5	1	14	10	15	5	6
30	S3	5	5	9	4	21	6	11
31	T1	6	4	8	5	19	8	12
32	T2	2	4	13	14	13	3	7
33	U1	3	2	13	11	16	5	7
34	V1	7	3	10	5	18	7	10
35	V2	2	2	16	14	13	2	4

Table 2.2.: Results of the amplitude normalization perception experiment per subject

Table 2.3 shows the experiment itemised by sentences. The experiment was generated using the script in section A.2.4. The results of the generation script were not modified, but should have. There are sentences where both stimuli pairs were identical or both stimuli pairs were different. For example sentence no. 9 in table 2.3 which is in both test pairs identical. If they were different, they were permuted as well as the identical ones. For the evaluation these sentences nonetheless yield interesting results. Focusing on the sentences no. 9 and no. 25: Both sentences were present only as 70 identical stimuli pairs in the experiment, but both have 30 a-/o- and 33 respectively. So nearly half of these 70 sentence pairs each were perceived as non identical.

The results in the other sentences show a clear preference for the unmodified (o+)

wave, not for a+, especially the sentences no. 15, no. 20 and no. 21. The sentences no. 15 and no. 21 start with an initial /h/, which is too loud. Figure 2.7 illustrates exactly this problem. One of the waves was normalised and the other one was not modified. Sentence no. 20 got corrupted by signal processing for the normalisation of the sentence: a short crack was inserted by accident.

	a+	a-	i+	i-	o+	o-	a-&o-	identical	different
1	2	0	25	23	10	10	10	35	35
2	2	3	32	24	9	0	3	35	35
3	2	0	31	20	13	4	4	35	35
4	6	0	28	23	6	7	7	35	35
5	1	0	23	15	19	12	12	35	35
6	2	0	25	13	20	10	10	35	35
7	5	0	13	5	25	22	22	35	35
8	14	0	0	27	29	0	0	0	70
9	0	18	37	0	0	15	33	70	0
10	3	15	20	7	25	0	15	35	35
11	5	0	20	8	22	15	15	35	35
12	10	0	0	30	27	0	0	0	67
13	6	21	14	16	13	0	21	35	35
14	4	0	22	9	22	13	13	35	35
15	3	0	0	1	66	0	0	0	70
16	4	0	14	19	12	21	21	35	35
17	11	0	0	25	34	0	0	0	70
18	17	0	0	23	23	0	0	0	63
19	3	0	14	6	26	21	21	35	35
20	8	0	0	19	43	0	0	0	70
21	0	0	30	3	32	5	5	35	35
22	1	15	20	20	14	0	15	35	35
23	10	0	0	31	29	0	0	0	70
24	6	15	20	5	24	0	15	35	35
25	0	17	40	0	0	13	30	70	0

Table 2.3.: Results of the amplitude normalization perception experiment per sentence

2.6. Conclusion

Some minor errors were made by the author regarding the perception experiment. The distribution of control sentences and test cases should have been equal for each stimuli pair, viz. there should have been the same amount of control sentences as there were test cases. In fact, this distribution varied a lot because the amount relation was generated by random, i.e. some experiments consisted only of identical stimuli (sentences no. 9 and no. 25) and some experiments only consisted of distinct stimuli (sentences no. 8, no. 12, no. 15, no. 17, no. 18, no. 20 and no. 23).

This means one control sentence versus one test case. The other minor mistakes which were fixed as soon as noticed should not have an impact on the results.

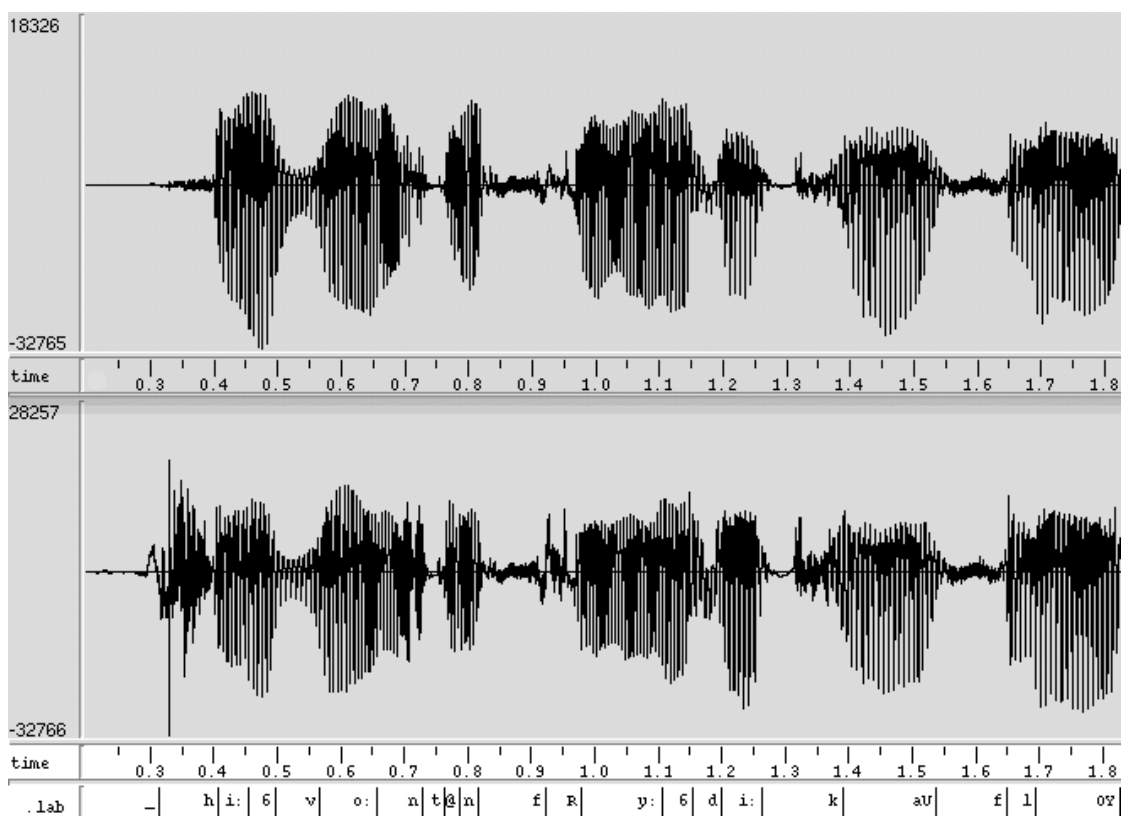


Figure 2.7.: Not modified (top) versus normalised wave for sentence 21.

The result of the perception experiment is that amplitude normalisation is not helpful. The minimal impact of amplitude normalisation and the problems signal processing causes, do not legitimate the use of amplitude normalisation. Most subjects perceived the signal processing problems like the /h/ or the one short crack in sentence no. 8. Maybe an amplitude normalisation without signal processing problems would improve the perceptive quality of speech synthesis. But nearly always signal processing results in quality reduction. For solving the problem with the phoneme /h/ an experiment with amplitude normalised diphones or amplitude normalised syllables would maybe be an interesting option for further studies.

3. F0 cost function

3.1. Introduction

The second part of this thesis is the attempt to improve the Festival synthesis system. This improvement should be realised with an addition of a f0 cost function. A part of this idea came from the work of Nukaga et al. (2004). They optimised the selection of units in their unit selection system using a pitch synchronous cross correlation cost function. Additionally they applied signal processing (psola, Moulines & Charpentier 1990) on the final wave data.

A pitch cost function has been added by the author to the psm algorithm. Later this cost function was evaluated by comparing it to an ideal pitch curve from PaIntE (cf. Möhler 1998), by calculating the smoothness of the pitch curve and by comparing the chosen psm units. The idea used in this work can be considered as a brute force approach (cf. Díaz & Banga 2005). With the brute force approach, all possible unit concatenations of the preselected psm units were tried and the best one was chosen by using the above mentioned pitch curve comparison and the smoothness of the pitch curve. The brute force attempt was evaluated offline, i.e. not at speech synthesis runtime.

3.2. Optimizing f0

The first idea was to use PaIntE which was developed by Gregor Möhler in his dissertation (cf. Möhler 1998). PaIntE which means "Parametric Representation of Intonation Events" approximates the pitch contour by a phonetically motivated model function. This model was trained by Gregor Möhler for a diphone voice. PaIntE is triggered by ToBI-rules (cf. Silverman et al. 1992) which are generated by previous synthesis steps.

The use of PaIntE as a cost function was difficult. PaIntE was developed for the diphone synthesis system in Festival which means that it uses a model for duration calculation. In unit selection, the duration is calculated by the length of the units. Normally, PaIntE runs in the `Int_Targets`-Phase of the syntheses (cf. figure 1.1). In this phase, the candidates for the unit selection have not been chosen yet, so there exists only the duration information from the diphone synthesis model which is often longer. When PaIntE is started at a later point of time, after the `Wave_Synth` phase, the duration of units are correct and can be saved in an f0-file. The resulting f0 from PaIntE is not the really expected f0 curve. The expected f0 curve is more fluctuating and natural than the f0 curve from PaIntE. But the PaIntE generated f0 curve is the only calculated reference available for comparison. This f0 curve from PaIntE is used to calculate the difference between it and the unit selection f0 curve (cf. chapter 3.5).

In order to improve the f0 smoothness, an additional cost function was added in the phase of the selection of the best unit for concatenation. This cost function is discussed in the next section.

3.3. Cost function

3.3.1. Implementation in Festival

A new cost function must interact and be comparable to the join weights cost function already used within the psm algorithm. Join weights base on the mel frequency cepstral coefficients (MEL-cepstrum) which are calculated beforehand which means not at runtime of Festival, but on voice generation via `sig2fv`. Using join weights, the best concatenation unit is calculated. The join weights are set to 1.0 the moment because it is an unsolved problem to balance these weights.

The join weight calculation is placed in the `framedistance`-function which calculated the distance of two units. Units can be phrases, words, syllables or segments. For the f0 cost function, two values are needed: `f0Weight` and `f0WeightFactor`. `F0weight` is comparable to the join weight values, which means an `f0Weight` of 1.0 should be comparable to 1.0 join weights. The `f0WeightFactor` is calculated for every unit selection voice and helps adapting it to the join weight state. Section 3.3.2 illustrated calculating `f0WeightFactor`.

See figure 3.1 for pseudo code of the `framedistance`-function and chapter A.3.2 for complete C++ code.

```

if f0Weight > 0 and f0WeightFactor > 0:
    cost = f0Weight * f0WeightFactor * abs(thisunit.f0(start)
                                           - prevunit.f0(end))
else:
    cost = 0

foreach c in channels:
    if joinweight(c) not 0:
        diff = thisunit(prevunitstart,c) - prevunit(prevunitend,c)
        diff = diff * joinweight(c)
        cost = cost + diff * diff

    if cost >= currentBestSqr:
        return currentBest

return sqrt(cost)

```

Figure 3.1.: Main part of `framedistance`-function in pseudo code

When `f0Weight` and `f0WeightFactor` are greater than 0, the starting cost is calculated

by multiplying `f0Weight`, `f0WeightFactor` and the f0 difference between the beginning of the current unit and the end of the last unit. After the initialization of the variable `cost`, the join weights were used within a loop to calculate the cost value for each of them. When the actual cost is larger than the actual best score, the function terminates and the calling function, which is `optimal_coupling`, uses the old current best score.

`f0Weight` and `f0WeightFactor` were set in the psm voice definition, which is located in `ims_psm_params.scm`. See the excerpt from `ims_psm_params.scm` to define the values for the male and the female voice of SmartWeb (see figure 3.2).

```
(set! psm_params_sw_ms
 '(
   ...
   ; f0Weight is used in optimal_coupling/framedistance
   (f0Weight 1.0)
   (f0WeightFactor 1165.591)
 )
)

(set! psm_params_sw_rk
 '(
   ...
   ; f0Weight is used in optimal_coupling/framedistance
   (f0Weight 1.0)
   (f0WeightFactor 4349.799)
 )
)
```

Figure 3.2.: Setting `f0Weight` and `f0WeightFactor` for the psm voice definitions

3.3.2. Calculating `f0WeightFactor` for a unit selection voice

This section explains the calculation of `f0WeightFactor` used in the previous section. The value `f0WeightFactor` is needed to balance `f0Weight` in the f0 cost function versus the join weights that are already in use. This section presents the calculation used for this factor in this work. For other voices or after voice changes a new calculation may be required.

To calculate the `f0WeightFactor`, a variation of the `framedistance`-function is needed. First a temporary variable is used for `cost` to save the value of the f0 cost calculation until the end of the function. Right before the return-statement the following line was added:

```
cout << "f0cost: " << savedF0cost << " | complete cost: " << cost << endl;
```

For this calculation, the sentences from the amplitude experiment and some additional sentences were used. All these sentences were processed by Festival with `f0Weight` and

`f0WeightFactor` set to 1.0. Afterwards the log was searched for `f0cost:` and the two numeric values were extracted.

```
cat logfile | grep "f0cost: " | cut -d' ' -f2,6 > cost.data
```

The resulting data file was processed with R (cf. R Development Core Team 2006). From all `f0` cost values and from the difference between full cost values and `f0` cost values, the mean was calculated. The mean of the difference was divided by the `f0` cost mean. The following R-code was used:

```
a <- read.table("cost.data")
a$V3 = a$V2-a$V1
mean(a$V3)/mean(a$V1)
```

The result is the `f0WeightFactor` for this voice. This may not be the perfect value, but it is an approach to get a reasonably usable value for `f0Weight`. Maybe a better `f0WeightFactor` can be calculated with more sentences.

3.4. Creating a testbench using a brute force approach

Nukaga et al. (2004) used a brute force approach to find the best `f0` by comparing each `f0` to the best `f0`. Unfortunately, there is no information in the paper on what the best units are or even what the best `f0` is.

This chapter shows an attempt where all curves were compared to the PaIntE-generated `f0` curve (cf. section 3.5). Using such a brute force approach results in the possibility to test all unit combinations the psm algorithm selects and not only those that are assumed to be the best ones. All unit combinations were synthesized and compared offline. For comparison the methods from chapter 3.5 were used. The basic idea was to run one synthesis with psm tree search and to leave every unit out once in a loop. This results in as many sentences as units found by the tree search. An important issue is that only units are left out when other candidates for this segment are possible. Scheme code to achieve this is as follows:

```
(set! utt1 (utt.synth (Utterance Text
  "Hier wohnten früher die Kaufleute und Handwerker.")))
(save_tts_output_all utt1)
(Param.set 'Synth_Method 'psm2_plus)
(set! i 0)
(while (> (count_cands utt1) i)
  (set! i (+ i 1))
  (Wave_Synth utt1)
  (set! tts_filename (string-append "sentence21" "-" i))
  (save_tts_output_all utt1))
(Param.set 'Synth_Method 'psm2)
```

In the synthesis method `psm2_plus` no `tree_search` function is called. Instead a function named `drop_psm_candidate` is called. This function and `count_cands`, which only returns a candidate count for termination of the while loop, were programmed in C++. All C++ code used for this part can be found in appendix (A.3.3). Within `drop_psm_candidate`, every unit independent from being a phrase, word, syllable or segment, will be dropped one time. The only exception are stand-alone pauses, mostly at the beginning or ending. They have no effect on the measured pitch values. For remembering which units have already been dropped, the variables `bruteforce_cnt`, `unit_cands_orig` and `unit_cands_orig_cnt` were used. Another approach would be to drop one candidate from the list, but that is not intended with the `EST_VTCandidate` class. Each time a new list has to be created by using the new function `remove_candidate`. Freeing this memory is a big problem and at the moment unsolved. The problem is, that nearly everything in a `EST_VTCandidate` variable is a pointer and should not be freed because the data the pointer refers to are used by other data structures. A few tests with freeing only the `EST_VTCandidate` memory were made, but it did not work as expected. The amount of memory used and not freed is minimal compared to the memory used by the rest of Festival.

The evaluation of the data generated using the brute force approach is dealt with in the corresponding evaluation part in the next section.

3.5. Evaluating the cost function

Three methods were used to evaluate the cost function.

3.5.1. Comparing f0 with PaIntE

One method is a comparison of the f0 curve to the output of PaIntE (cf. chapter 3.2). The idea is to calculate the area between the two curves.

When the area decreases, the f0 curve should consist of better candidates. Curve one is the PaIntE output and curve two is the output from a simple tcl script (cf. figure 3.3) using wavesurfer (cf. Sjölander & Beskow 2000). For calculating the area, the value from the wavesurfer-generated curve is subtracted from the PaIntE value for every time index.

An overall value for the full f0 plot is calculated using RMSE. The formula used is

$$curve_{RMSE} = \sqrt{\frac{\sum_{i=0}^{length(wave)} (f0_{ws}(i) - f0_{PaIntE})^2}{length(wave)}}.$$

The smaller the RMSE value the better the f0 curve is in comparison to the PaIntE curve.

```
#!/bin/sh
# the next line restarts using wish \
exec wish8.4 "$0" "$@"

package require snack
snack::sound s

# list of waves / generated using ls -l
set f [open list.txt]
set list [read $f]
close $f

foreach file $list {
  s read $file

  set fd [open [file rootname $file].wsurf.f0 w]
  puts $fd [join [s pitch -method esp] \n]
  close $fd
}
```

Figure 3.3.: Tcl code for generating f0 using wavesurfer

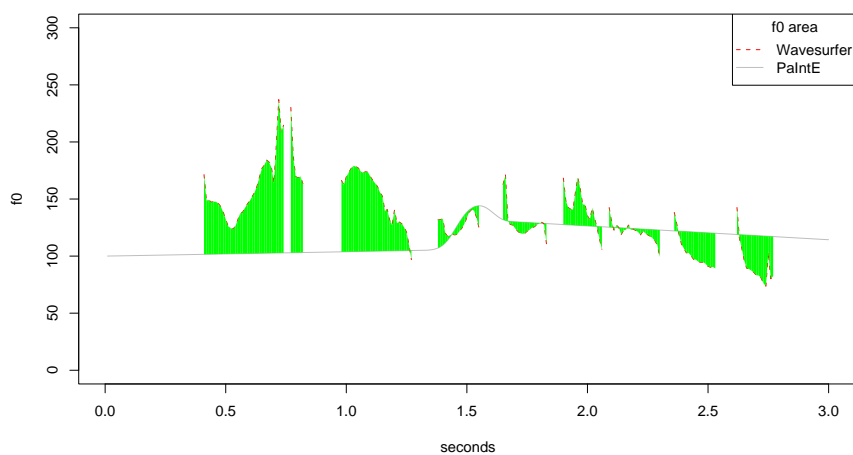


Figure 3.4.: f0 area example for sentence 21

3.5.2. Results of the comparison between f0 and PaIntE

This section presents a comparison between the PaIntE curve and all sentences that had been generated by brute force. The brute force calculations were made with `f0Weight` set to 0.0, 1.0, 2.0, 3.0 and 5.0. The RMSE is calculated as shown in section 3.5.1.

The results are shown in table 3.1. The headings in the following tables for RMSE are RMSE-# to denote the corresponding `f0Weight` value, i.e. for RMSE-2 `f0Weight` is 2.0. The voice used was the male SmartWeb voice with a `f0WeightFactor` set to 1165.591 which was calculated using the procedure described in section 3.3.2. The sentence numbers correlate to the sentences in table 2.1. Additionally, for all `f0Weights` a column with the occurring zeros in the f0 curve is listed. Zeros are the value of 0.0 in a f0 curve. This is important because the 0.0-values were not used in the calculation of the area RMSE. The zeros are named Z-#, i.e. Z-1 for `f0Weight` equal to 1.0.

	RMSE-0	RMSE-1	RMSE-2	RMSE-3	RMSE-5	Z-0	Z-1	Z-2	Z-3	Z-5
1	9.494173	10.240459	10.240459	9.559289	9.559289	106	105	105	106	110
2	14.026548	14.553622	14.553622	14.553622	14.553622	137	131	131	131	131
3	10.657527	10.657527	10.657527	10.657527	10.657527	89	89	89	89	89
4	8.158036	6.707045	6.707045	6.707045	6.707045	164	165	165	165	165
5	7.084494	7.153891	7.153891	7.153891	6.256744	136	138	138	138	138
6	7.553122	7.553122	7.553122	7.553122	7.496007	162	162	162	162	161
7	12.958902	13.127166	13.364115	13.173789	13.271254	431	421	427	417	417
8	13.317821	10.771456	10.322275	10.795495	12.049977	445	470	453	451	466
9	15.868860	16.446665	15.443363	15.443363	16.209701	281	269	271	271	259
10	11.774635	10.915285	10.915285	11.686443	11.269642	275	281	281	281	277
11	9.740616	9.317185	9.317185	9.317185	9.547906	271	271	271	271	266
12	10.667482	11.830677	11.502112	11.502112	11.124996	251	248	250	250	244
13	13.811605	13.790071	13.790071	13.790071	13.790071	221	218	218	218	218
14	12.230948	12.310486	12.310486	12.170036	12.170036	263	264	264	264	264
15	15.480062	15.835578	15.835578	15.835578	15.835578	194	190	190	190	190
16	11.680901	11.551691	11.566570	11.430233	11.430233	222	218	218	221	221
17	13.912797	13.912797	13.912797	13.912797	13.912797	187	187	187	187	187
18	13.495673	13.855253	13.750831	13.750831	13.035929	291	286	288	288	286
19	14.239654	13.812584	13.999325	13.999325	13.999325	232	234	234	234	234
20	14.918362	14.998431	14.998431	14.998431	14.998431	174	174	174	174	174
21	18.342743	17.150827	17.150827	17.150827	17.150827	162	157	157	157	157
22	11.897885	9.540581	9.540581	9.540581	9.540581	143	146	146	146	146
23	15.458915	14.964852	8.772951	8.772951	9.475283	242	244	244	244	246
24	11.506588	11.335677	11.183291	11.841330	12.084612	336	333	330	330	333
25	10.568789	14.896946	14.254650	13.901204	13.901204	311	311	315	313	313

Table 3.1.: RMSE and zero values for all synthesized sentences

The result are three groups of sentences: First, the ones that remain largely unchanged (sentences 3, 6, 17, 20). This is mainly because of a low number of candidates. Second, the ones which get clearly better. These are the sentences 4, 8, 10, 11, 13, 16, 19, 21, 22, 23 and 24. Third, the sentences which seem to get worse, but the zero value mostly decreases. This means, that there are more points to calculate an area, thus the RMSE is bigger. More points on the curve mean that points *now* are not 0.0 anymore but that

they may have been 0.0 when another `f0Weight` was applied. This can result in a better `f0`. RMSE-1 to RMSE-5 can be better than RMSE-0, but this cannot be shown using this test. Another problem is for example sentence 25. In this sentence Z-0 and Z-1 are identical, but are these zeros at the same time position?

For testing this, the 0.0 values in the both `f0`-files were compared. At 258 milliseconds both `f0`-files were 0.0, but at 106 milliseconds, one was 0.0 and the other was not. This is distributed exactly even-numbered, i.e. 53 0.0s each. So Z-0 and Z-1 seem identical but some of the zeros are at other positions.

Four sentences do not change, and most of the other ones get an decreased RMSE or decreased zeros. The conclusion from this observations is that the `f0` cost function works and increases quality.

Table 3.2 and table 3.3 show values where the brute force approach was used. The table 3.2 shows the best RMSE for a sentence and the occurring zeros. Table 3.3 shows the number of sentences that were synthesised and which number from the brute force run produced this result.

	RMSE-0	RMSE-1	RMSE-2	RMSE-3	RMSE-5	Z-0	Z-1	Z-2	Z-3	Z-5
1	6.764008	6.293054	6.293054	6.648047	7.758018	105	104	104	105	109
2	12.055558	11.751003	11.751003	11.751003	11.751003	135	130	130	130	129
3	9.484915	9.484915	9.484915	9.484915	9.188909	89	89	89	89	89
4	4.407645	3.314062	5.084101	5.359328	5.359328	159	160	160	160	160
5	6.767989	6.853690	6.853690	6.853690	5.355427	130	134	134	134	132
6	4.030137	4.845488	4.845488	4.845488	4.828476	162	162	162	162	161
7	11.400123	11.949901	11.682431	11.587415	11.817099	420	413	408	409	408
8	12.007310	9.183425	8.818972	9.434711	10.156553	423	436	420	421	437
9	14.733990	14.469578	14.476202	14.476202	15.026073	267	261	261	261	257
10	10.137733	9.195698	9.195698	9.282288	9.427088	268	269	269	269	267
11	9.116480	8.948360	8.948360	8.913797	8.831736	267	267	267	267	264
12	9.918522	9.674006	9.267711	9.267711	9.190921	241	237	237	237	236
13	12.050209	12.127920	12.127920	12.127920	11.492262	212	210	210	210	210
14	11.374048	11.903765	11.978060	11.599331	11.599331	252	259	259	260	260
15	13.216516	13.294504	13.294504	13.294504	13.294504	182	181	184	184	186
16	11.361576	11.287837	11.249193	11.183916	11.183916	213	215	215	215	215
17	11.496561	11.496561	11.496561	11.496561	11.496561	182	182	182	182	182
18	11.492703	11.286972	10.721538	10.721538	11.393423	280	279	282	282	282
19	12.854810	12.406581	12.616838	12.616838	12.616838	232	230	228	228	229
20	13.028471	13.336029	13.336029	13.336029	13.336029	172	170	170	170	170
21	18.342743	17.150827	17.150827	17.150827	17.150827	162	157	157	157	157
22	9.579492	9.540581	9.540581	7.959834	7.959834	143	143	144	144	144
23	8.671835	8.593316	7.036812	7.036812	7.219561	232	236	239	239	239
24	9.635696	10.347103	9.557252	10.326493	10.370974	327	328	327	328	331
25	9.337296	13.499182	13.000770	13.030472	12.952018	305	308	305	311	311

Table 3.2.: RMSE and zero values for the “best” synthesized sentences

	count	BEST-0	BEST-1	BEST-2	BEST-3	BEST-5
1	179	8	8	8	8	8
2	296	203	203	203	203	203
3	140	49	49	49	49	49
4	219	19	19	168	19	19
5	170	113	1	1	1	27
6	41	6	6	6	6	6
7	1284	483	483	483	483	483
8	952	315	84	84	84	84
9	630	35	25	25	25	25
10	189	35	35	35	35	35
11	220	198	170	170	22	22
12	551	407	407	407	407	86
13	429	29	29	29	29	29
14	277	20	188	34	250	250
15	321	19	19	19	19	19
16	130	53	107	9	53	53
17	143	12	12	12	12	12
18	715	24	24	33	24	24
19	142	51	51	51	51	51
20	130	11	11	11	11	11
21	12	0	0	0	0	0
22	67	35	0	0	2	2
23	485	310	340	296	296	296
24	308	45	6	41	45	45
25	526	88	462	103	105	105

Table 3.3.: Synthesized sentences with best RMSE and quantity of synthesized sentences

Table 3.2, the brute force table, is very similar to table 3.1. Similarly most RMSE values decrease or the corresponding zero values lower.

The interesting issue on table 3.3 is, that half of the best sentences remain constant (sentences 1, 2, 3, 6, 7, 10, 13, 15, 17, 19, 20, 21). Only one sentence (no. 22) changes to 0 as best sentence.

The zeros are interesting because they are varying and change the RMSE values. Figure 3.5 shows the distribution of zeros over all brute force synthesized sentences for each `f0Weight` value. In most sentences, the zero count increased on increasing `f0Weight` (for example sentence 1, 3, 6, 9 and others). Minimums and maximums are shown in table 3.4 for reference.

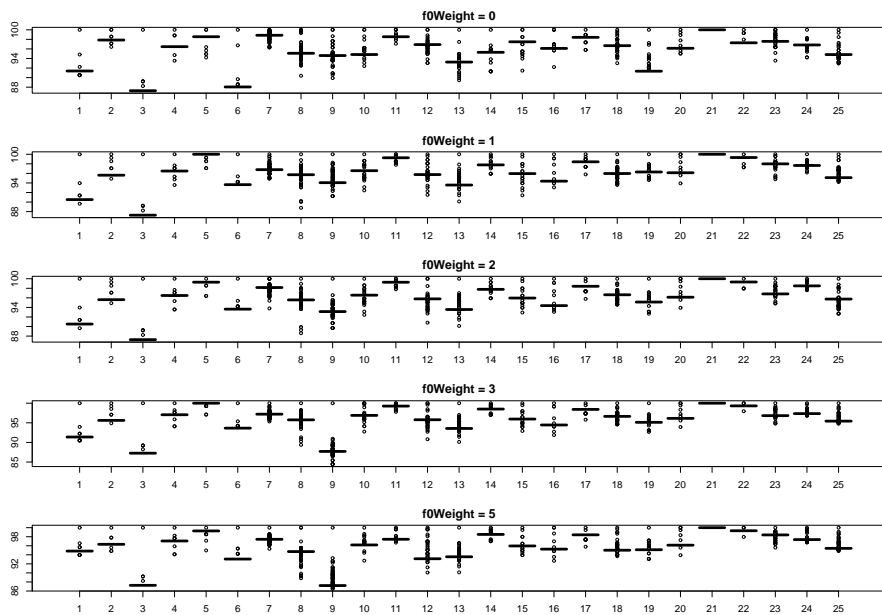


Figure 3.5.: Distribution of zero points on the pitch curve normalised in percent

	MIN-0	MIN-1	MIN-2	MIN-3	MIN-5	MAX-0	MAX-1	MAX-2	MAX-3	MAX-5
1	105	104	104	105	109	116	116	116	116	116
2	135	130	130	130	129	140	137	137	137	136
3	89	89	89	89	89	102	102	102	102	102
4	159	160	160	160	160	170	171	171	170	170
5	130	134	134	134	132	138	138	139	138	139
6	162	162	162	162	161	184	173	173	173	173
7	420	413	408	409	408	436	435	435	429	428
8	423	436	420	421	437	468	491	474	471	492
9	267	261	261	261	257	297	286	291	309	297
10	268	269	269	269	267	290	291	291	290	288
11	267	267	267	267	264	275	273	273	273	273
12	241	237	237	237	236	259	259	261	261	262
13	212	210	210	210	210	237	233	233	233	233
14	252	259	259	260	260	276	270	270	268	268
15	182	181	184	184	186	199	198	198	198	198
16	213	215	215	215	215	231	231	231	234	232
17	182	182	182	182	182	190	190	190	190	190
18	280	279	282	282	282	301	298	298	298	301
19	232	230	228	228	229	254	243	246	246	246
20	172	170	170	170	170	181	181	181	181	181
21	162	157	157	157	157	162	157	157	157	157
22	143	143	144	144	144	147	147	147	147	147
23	232	236	239	239	239	248	249	252	252	250
24	327	328	327	328	331	347	341	335	339	342
25	305	308	305	311	311	328	327	329	328	328

Table 3.4.: Maximum and minimum of zero counts

3.5.3. F0 curve smoothness

Another approach to evaluate the f0 curve was to calculate the smoothness of the curve. An increased smoothness is a good evidence for fewer steps in the f0 curve. This was achieved by adding the absolute difference between all consecutive points. The mathematical formula for this calculation is $\sum_{i=0}^{n-1} |x_{i+1} - i_i|$. For example a few values: 0, 0, 10, 11, 34, 0, 12, 4. The computation of these values is $(0 - 0) + (10 - 0) + (11 - 10) + (34 - 11) + \text{abs}(0 - 34) + (12 - 0) + \text{abs}(4 - 12) = 88$.

Zeros are not a problem because the difference addition is used on every f0 value pair.

3.5.4. Results of the f0 curve smoothness calculations

In table 3.5 and table 3.6 below the sums of every f0 pair difference is listed for f0Weight being 0.0, 1.0, 2.0, 3.0 and 5.0. The male SmartWeb voice was used for this experiments, too. Also the sentence numbers can be compared with table 2.1 on page 10. The expectation for both tables is, that the smoothness sum decreases with increasing f0Weight.

	S-0	S-1	S-2	S-3	S-5
1	869.0537	844.1460	844.1460	944.0207	944.0207
2	1300.0061	1222.7211	1222.7211	1222.7211	1222.7211
3	920.3887	920.3887	920.3887	920.3887	920.3887
4	1421.4055	1369.5170	1369.5170	1369.5170	1369.5170
5	925.7586	954.9979	954.9979	954.9979	1041.5160
6	1758.6144	1758.6144	1758.6144	1758.6144	1622.5162
7	9834.5950	10372.9539	10786.8381	10243.3817	10217.2811
8	10151.3434	9640.7697	9714.3353	9657.0340	9373.3342
9	10584.0522	9860.3523	9598.1125	9598.1125	10057.8102
10	5702.9350	5778.9656	5778.9656	5759.4510	5679.6300
11	7172.4119	7289.3867	7289.3867	7289.3867	7051.1797
12	6630.2317	6391.2175	7954.8624	7954.8624	7817.8195
13	6416.8020	6213.8972	6213.8972	6213.8972	6213.8972
14	6128.2533	6083.8751	6083.8751	6003.1436	6003.1436
15	4726.3576	4853.0448	4853.0448	4853.0448	4853.0448
16	6216.4504	5716.6732	5867.5191	5664.2491	5664.2491
17	3842.4980	3842.4980	3842.4980	3842.4980	3842.4980
18	8769.9849	8844.6793	8822.3397	8822.3397	8511.9881
19	7226.5524	7322.9901	7356.3546	7356.3546	7356.3546
20	4009.1062	3960.9468	3960.9468	3960.9468	3960.9468
21	3523.2109	3633.0447	3633.0447	3633.0447	3633.0447
22	2231.9649	2179.3294	2179.3294	2179.3294	2179.3294
23	6254.8593	6449.0515	5750.9664	5750.9664	5694.5087
24	7415.6391	7324.7212	7492.2125	6937.7133	7063.3659
25	5829.0783	6377.4482	7046.0934	6972.1185	6972.1185

Table 3.5.: Smoothness values for synthesized sentences

In table 3.5 the smoothness of most sentences decreases with `f0Weight` ≥ 1.0 , for some others the smoothness increases. Twelve sentences have a decreasing smoothness value whereas ten have an increasing smoothness value. For some sentences, the smoothing count does not change (sentences 3, 6 and 17). Such variation is no evidence for an improvement of the f0 cost function.

Table 3.6 shows almost the same smoothness scores as table 3.5. Decreasing and increasing smooth values balance each other.

Consequently, it can be stated that the smoothness calculations are no good benchmark for improvement by the f0 cost function.

	S-0	S-1	S-2	S-3	S-5
1	869.0537	844.1460	844.1460	944.0207	944.0207
2	1300.0061	1222.7211	1222.7211	1222.7211	1222.7211
3	920.3887	920.3887	920.3887	920.3887	920.3887
4	1421.4055	1369.5170	1369.5170	1369.5170	1369.5170
5	925.7586	954.9979	954.9979	954.9979	1041.5160
6	1758.6144	1758.6144	1758.6144	1758.6144	1622.5162
7	9834.5950	10372.9539	10786.8381	10243.3817	10217.2811
8	10151.3434	9640.7697	9714.3353	9657.0340	9373.3342
9	10584.0522	9860.3523	9598.1125	9598.1125	10057.8102
10	5702.9350	5778.9656	5778.9656	5759.4510	5679.6300
11	7172.4119	7289.3867	7289.3867	7289.3867	7051.1797
12	6630.2317	6391.2175	7954.8624	7954.8624	7817.8195
13	6416.8020	6213.8972	6213.8972	6213.8972	6213.8972
14	6128.2533	6083.8751	6083.8751	6003.1436	6003.1436
15	4726.3576	4853.0448	4853.0448	4853.0448	4853.0448
16	6216.4504	5716.6732	5867.5191	5664.2491	5664.2491
17	3842.4980	3842.4980	3842.4980	3842.4980	3842.4980
18	8769.9849	8844.6793	8822.3397	8822.3397	8511.9881
19	7226.5524	7322.9901	7356.3546	7356.3546	7356.3546
20	4009.1062	3960.9468	3960.9468	3960.9468	3960.9468
21	3523.2109	3633.0447	3633.0447	3633.0447	3633.0447
22	2231.9649	2179.3294	2179.3294	2179.3294	2179.3294
23	6254.8593	6449.0515	5750.9664	5750.9664	5694.5087
24	7415.6391	7324.7212	7492.2125	6937.7133	7063.3659
25	5829.0783	6377.4482	7046.0934	6972.1185	6972.1185

Table 3.6.: Smoothness values for best brute force synthesized sentences

3.5.5. Calculating differences in psm results

Finally, the third attempt shows how many units have changed by modifications in the speech synthesis process. The psm algorithm (cf. section 1.2) has a function that returns the chosen candidates. This means not only the ones emerging as the best in the end, but also that the units selected by the tree search function were written to the psm results file. By comparing the output for increasing `f0Weight` it is possible to show how many other candidates were chosen.

3.5.6. Results of the differences in psm results

For this evaluation, the psm results of every synthesised sentence is used. In the psm results every candidate is listed.

A script counted the units that changed for the brute force run. The resulting table 3.7 lists the minimums and maximums of the comparison between the listed `f0Weight` and a `f0Weight` equal to 0.0 (i.e. MIN-2 is the minimum of the differences between `f0Weight` equals 0.0 and `f0Weight` = 2.0).

	MIN-1	MIN-2	MIN-3	MIN-5	MAX-1	MAX-2	MAX-3	MAX-5
1	0	0	1	2	3	3	3	5
2	1	1	1	1	6	6	6	6
3	0	0	0	0	0	0	0	4
4	1	1	1	1	4	4	4	4
5	0	0	0	2	4	4	4	4
6	0	0	0	1	2	2	2	3
7	3	7	9	11	8	13	15	17
8	1	5	8	11	7	12	15	18
9	5	6	6	8	10	11	11	13
10	0	0	1	1	3	3	6	7
11	1	1	1	2	6	6	7	6
12	2	4	5	5	5	9	9	9
13	0	0	0	0	2	3	3	3
14	1	1	2	2	6	6	8	8
15	0	0	0	0	3	5	5	5
16	0	0	0	1	3	3	3	3
17	0	0	0	0	1	1	1	1
18	1	1	1	2	6	7	7	8
19	0	0	0	0	3	4	4	4
20	0	0	0	0	0	0	0	2
21	1	1	1	1	1	1	1	1
22	0	0	0	0	2	2	3	3
23	0	0	0	1	3	3	3	4
24	1	3	3	5	4	6	9	11
25	7	7	10	10	12	12	16	16

Table 3.7.: Minimum and maximum of every psm changes for every candidate combination of the sentences

The table shows that for most sentences the maximum and for some the minimum of unit changes increases. The result can not contribute anything to the question whether this changes are positive or not.

3.6. Conclusions

In the first evaluation method the area between the f0 curve and a idealistic curve (PaIntE) was tested. The result is that the area gets smaller with higher f0Weight or the amount of zero points on the f0 curve decreases. If there are less zero points on the curve the area automatically increases, but the f0 can be better. The second evaluation method was the calculation of the smoothness of the f0 curves. This method is no good benchmark for evaluating the improvement of the f0 cost function because the values balance each other. The third evaluation method shows only the changes in the selection algorithm and does not help in rating the f0 cost function. So every evaluation method shows that the f0 cost function modifies the resulting synthesised speech. But none of the evaluation methods can prove the usefulness of the changes that have been applied. A perception experiment may be convenient to evaluate the usefulness of the f0 cost function and can help to improve the f0WeightFactor.

4. Summary

4.1. Amplitude normalisation

The results of the amplitude normalisation are disillusioning. Modifying loudness was found to result only in a minor effect on perception. The subjects seem to perceive the signal processing problems only. These losses through signal processing do not seem to legitimate the use of such an amplitude modification. The problems are on the one hand the loudness regulation of the phoneme /h/ which seems to be extremely context sensitive and on the other hand, short cracks which were added by mistake. Even though the signal processing works perfect and the problems with the /h/ are solved, the use of amplitude normalisation remains questionable. The perception experiment does not imply that amplitude normalisation is worthless. But the results do also not implicate that amplitude normalisation is helpful even if no quality reduction is perceivable.

An interesting option for further studies would be maybe an amplitude normalisation for diphones or better for syllables. This may solve the problem with the phoneme /h/.

4.2. F0 cost function

All three evaluation approaches show that there are modifications by the new cost function. The problem is, that they cannot prove whether these changes are positive, unneeded or even effectless.

Maybe a perception experiment would help to improve the cost function values `f0Weight` and `f0WeightFactor`. The problem here would be to find an overall solution for all cases, which seems impossible at the moment. There was no time to run such a perception experiment but it would be interesting to see whether the changes made were audible.

Bibliography

- Black, A. W., Taylor, P., 1997. Automatically clustering similar units for unit selection in speech synthesis. In: Proceedings of the European Conference on Speech Communication and Technology (Rhodos, Greece). Vol. 2. 601–604.
- Black, A. W., Taylor, P., Caley, R., 1999. *The Festival speech synthesis system—system documentation*. CSTR Edinburgh.
URL <http://www.cstr.ed.ac.uk/projects/festival>
- Cruttenden, A., 1986. *Intonation*. Chambridge University Press.
- Díaz, F. C., Banga, E. R., 2005. A method for combining intonation modelling and speech unit selection in corpus-based speech synthesis systems. *Speech Communication* **48** (8), 941–956.
- Hakkarainen, H. J., 1995. *Phonetik des Deutschen*. Wilhelm Fink Verlag.
- Kamp, H., Reyle, U., 1993. *From Discourse to Logic*. Kluwer Academic, Dordrecht, The Netherlands.
- Klankert, B., 2003. *Hybride Unit Selection für ein Sprachsynthesesystem*. Master Thesis.
- Kohler, K. J., 1977. *Einführung in die Phonetik des Deutschen*. Erich Schmidt Verlag.
- Möhler, G., 1998. *Theoriebasierte Modellierung der deutschen Intonation für die Sprachsynthese*. AIMS.
- Moulines, E., Charpentier, F., 1990. Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. *Speech Commun.* **9** (5-6), 453–467, beschaffen!!
- Nukaga, N., Kamoshida, R., Nagamatsu, K., 2004. Unit selection using pitch synchronous cross correlation for Japanese concatenative speech synthesis. In: Proceedings of the 5th ISCA Speech Synthesis Workshop (Pittsburgh, PA).
- R Development Core Team, 2006. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.
URL <http://www.R-project.org>
- Schweitzer, A., Braunschweiler, N., Klankert, T., Möbius, B., Säuberlich, B., 2003. Restricted unlimited domain synthesis. In: Proceedings of Eurospeech-2003 (Geneva). 1321–1324.

- Silverman, K., Beckman, M., Pitrelli, J., Ostendorf, M., Wightman, C., Price, P., Pierrehumbert, J., Hirschberg, J., 1992. ToBI: A standard for labelling English prosody. In: Proceedings of the International Conference on Spoken Language Processing (Banff, Alberta). Vol. 2. 867–870.
- Sjölander, K., Beskow, J., 2000. Wavesurfer – an open source speech tool. In: ICSLP. Proc of ICSLP, Beijing, 4:464–467.
- Taylor, P., 2000. *Concept-to-Speech synthesis by phonological structure matching*.
- Taylor, P., Black, A. W., 1999. Speech synthesis by phonological structure matching. In: Proceedings of the European Conference on Speech Communication and Technology (Budapest, Hungary). Vol. 2. 623–626.
- Taylor, P., Black, A. W., Caley, R., 1998. The architecture of the festival speech synthesis system. In: In The Third ESCA Workshop in Speech Synthesis (Jenolan Caves, Australia, 1998). 147–151.
- Taylor, P., Caley, R., Black, A., King, S., 1999. *The Edinburgh speech tools library: System Documentation*. The Centre for Speech Technology Research, University of Edinburgh.
URL http://www.cstr.ed.ac.uk/projects/speech_tools/

List of Figures

1.1. Part of the synthesis sequence in Festival	2
2.1. Five point time plot	5
2.2. Boxplots of energy values for selected phonemes	6
2.3. Energy plot of beginning of sentence 17	7
2.4. Screenshot of the experiment (German)	8
2.5. Plays per sentence	11
2.6. Variance of the six experiment variables according to table 2.2	12
2.7. Not modified (top) versus normalised wave for sentence 21.	15
3.1. Main part of <code>framedistance</code> -function in pseudo code	17
3.2. Setting <code>f0Weight</code> and <code>f0WeightFactor</code> for the psm voice definitions . . .	18
3.3. Tcl code for generating <code>f0</code> using <code>wavesurfer</code>	21
3.4. <code>f0</code> area example for sentence 21	21
3.5. Distribution of zero points on the pitch curve normalised in percent . . .	25
A.1. Boxplots of energy values for most relevant phonemes – male speaker . . .	36
A.2. Boxplots of energy values for most relevant phonemes – female speaker . .	37

List of Tables

2.1.	All words and sentences occurring in the perception experiment.	10
2.2.	Results of the amplitude normalization perception exp. per subject	13
2.3.	Results of the amplitude normalization perception exp. per sentence . . .	14
3.1.	RMSE and zero values for all synthesized sentences	22
3.2.	RMSE and zero values for the “best” synthesized sentences	23
3.3.	Sentences with best RMSE and quantity of synth. sentences	24
3.4.	Maximum and minimum of zero counts	26
3.5.	Smoothness values for synthesized sentences	27
3.6.	Smoothness values for best brute force synthesized sentences	28
3.7.	Min. and max. of every change per candidate combination of the sentences	29

A. Appendix

A.1. Plottings of energy values

A.1.1. The male speaker of SmartWeb

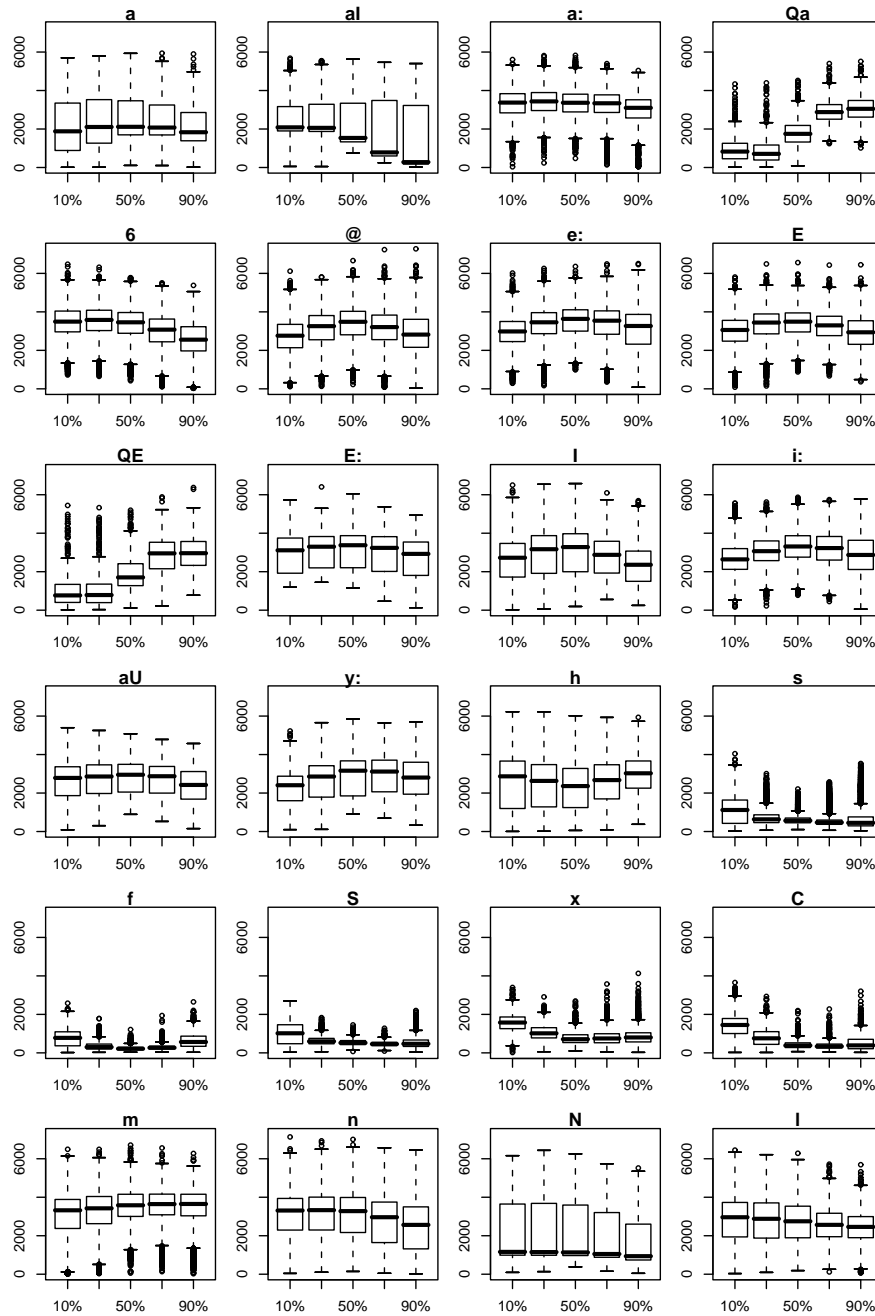


Figure A.1.: Boxplots of energy values for most relevant phonemes by the male speaker

A.1.2. The female speaker of SmartWeb

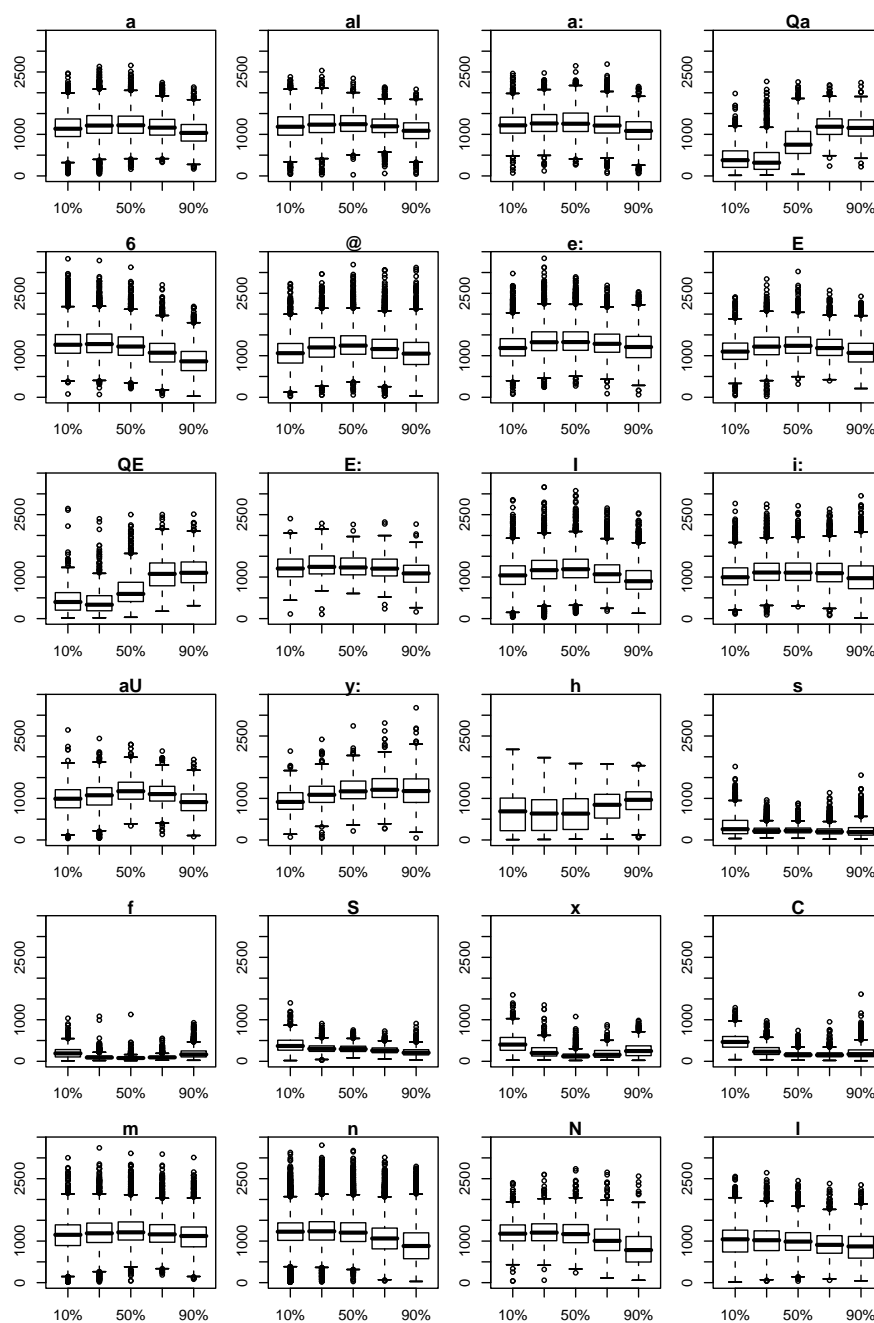


Figure A.2.: Boxplots of energy values for most relevant phonemes by the female speaker

A.2. Scripts

A.2.1. playnow.sh (German texts)

```
#!/bin/bash

# (c) Andreas Madsack
#
# $Rev: 124 $
#

DATECMD='date'
WHOAMICMD='whoami'

## windows:

PLAYCMD='playwav.exe '
PLAYCMD2='playwav.exe '
AWKCMD='awk.exe'
PERZLOG="ampnorm-perz-`$WHOAMICMD`.log"

## others:

if [[ "`uname`" -eq "Linux" ]]; then
    PLAYCMD='aplay '
    PLAYCMD2='aplay -q '
    AWKCMD='awk'
    PERZLOG="/tmp/ampnorm-perz-`$WHOAMICMD`.log"
elif [[ "`uname -o`" -eq "Cygwin" ]]; then
    echo
elif [[ "`uname`" -eq "SunOS" ]]; then
    echo Sorry, bitte nur auf Linux-Rechnern ausfuehren!
    exit
elif [[ "`uname -o`" -eq "IRIX" ]]; then
    echo Sorry, kein Irix-support am IMS Aufgrund einer viel zu alten Bash!
    echo Bitte auf einem Linux-Rechner ausfuehren.
    exit
fi

## keys:
ESC='1b'
KESC='71'
ONE='31'
TWO='32'
SHOW='73'
NEXT='6e'
PREV='70'
VIEW='76'
EQ='3d'
TO1='3c'
TO2='3e'
KEQ='6c'
KTO1='6a'
KTO2='6b'
```

```

touch $PERZLOG
chmod 600 $PERZLOG

echo '$DATECMD' - Start of playnow!' >> $PERZLOG

j=0
for i in `ls -lt *-1.wav`; do
    WAVLISTE[$j]='basename $i -1.wav'
    let "j = j + 1"
done

let "MAX = j"

j=1

PLAYED=0
JUDGED=0

clear
while [ "$keyesc" != $ESC ]
do
    echo "Tastekommandos"
    echo "-----"
    echo "| ESC      Programm beenden"
    echo "| 1       Datei 1 abspielen"
    echo "| 2       Datei 2 abspielen"
    echo "| n       naechste Wav"
    echo "| --"
    echo "| Bewerten: "
    echo "| j       Datei 1 klingt besser als Datei 2"
    echo "| k       Datei 2 klingt besser als Datei 1"
    echo "| l       Datei 1 klingt gleich gut wie Datei 2"
    echo "-----"

    THISWAV=${WAVLISTE[j]}

    echo "aktuelle Wav: " $THISWAV
    echo "Wav $j von $MAX"

    read -n1 -s key          # Read 1 character.

    keyesc=$( echo $key | od -t x1 | $AWKCMD '{ print $2}' )

    clear

    case "$keyesc" in
        "$KESC" )
            keyesc=$ESC
            ;;
        "$ONE" )
            echo '$DATECMD' - Play: $THISWAV-1' >> $PERZLOG
            $PLAYCMD "$THISWAV-1.wav"
            if (( $PLAYED % 2 )); then

```

```

:
else
  let "PLAYED = $PLAYED + 1"
fi
clear
;;
"$TWO" )
echo '$DATECMD' - Play: $THISWAV-2" >> $PERZLOG
$PLAYCMD "$THISWAV-2.wav"
if (( $PLAYED < 2 )); then
  let "PLAYED = $PLAYED + 2"
fi
clear
;;
"$NEXT" )
if (( $JUDGED < 1 )); then
  echo ""
  echo "Sie haben den Satz noch nicht bewertet."
  echo ""
else
  if (( $j < $MAX )); then
    let "j = $j + 1"
    echo '$DATECMD' - next [$j]" >> $PERZLOG
    JUDGED=0
    PLAYED=0
  else
    echo ""
    echo "Die letzte Wav wurde erreicht. Beenden mit ESC oder q."
    echo ""
  fi
fi
;;
"$EQ" | "$KEQ" )
if (( $PLAYED < 3 )); then
  echo ""
  echo "Erstmal beide Dateien anhoeren!"
  echo ""
else
  echo '$DATECMD' - "$THISWAV": =" >> $PERZLOG
  echo "Bewertung >Beide Dateien klingen gleich.< wurde gespeichert."
  echo ""
  JUDGED=1
fi
;;
"$T01" | "$KT01" )
if (( $PLAYED < 3 )); then
  echo ""
  echo "Erstmal beide Dateien anhoeren!"
  echo ""
else
  echo '$DATECMD' - "$THISWAV": <" >> $PERZLOG
  echo "Bewertung >Datei1 klingt besser als Datei2.< wurde gespeichert."
  echo ""
  JUDGED=1

```

```

    fi
    ;;
    "$T02" | "$KT02" )
    if (( $PLAYED < 3 )); then
        echo ""
        echo "Erstmal beide Dateien anhoeren!"
        echo ""
    else
        echo "$DATECMD" - "$THISWAV": >" >> $PERZLOG
        echo "Bewertung >Datei2 klingt besser als Datei1.< wurde gespeichert."
        echo ""
        JUDGED=1
    fi
    ;;
esac

done

echo "$DATECMD" - End of playnow!" >> $PERZLOG
clear
echo ""
echo "WICHTIG:"
echo "Mailen Sie mir bitte die Datei $PERZLOG an "
echo " andreas.madsack@ims... ."
echo "Dieses Skript uebermittelt keine Daten automatisch!"
echo ""
echo "Danke fuer Ihre Mithilfe!"
echo ""
$PLAYCMD danke.wav

exit 0

```

A.2.2. getpower.py

```

#!/usr/bin/env python

"""From wav/label-files to powercurves"""

__revision__ = "$Rev: 119 $"

import sys, os, getopt
from myconfig import MyConfig

class PowerProfile(MyConfig):

    def iterateWavDirectory(self):

        def toolcmd():
            t = self.getcfgvalue(self.getsection(), "tool")
            if t in "sig2fv":
                cmd = self.getcfgvalue(self.getsection(), "sig2fv_location") \

```

```

    + "/sig2fv -coefs energy -window_type " \
    + self.getcfgvalue("defaults", "sig2fv-window") \
    + " -size " \
    + self.getcfgvalue("defaults", "sig2fv-window-size") \
    + " -shift 0.001" # value for every ms!
else:
    print "No tool defined!!"
    sys.exit(3)
return cmd

def loadsegmentfile(filename):
    f = open(self.getcfgvalue(self.getsection(), "segments") + "/" \
            + filename + ".Segment")
    a = []
    for i in f.readlines():
        if len(i.split(" "))>2:
            b = []
            b.append(i.split(" ")[0])
            b.append(i.split(" ")[2])
            a.append(b)
    f.close()
    return a[1:] # eliminate first element/line

def writeresults(phonem, values, filename):
    f = open(self.getcfgvalue(self.getsection(), "destination") \
            + "/" + phonem + ".energy", "a")
    for i in values:
        f.write(i)
        f.write(" ")
    if self.getcfgvalue("defaults","fileinfo") == "true":
        f.write(" # "+filename)
    f.write("\n")
    f.close()

def genresultlist(l):
    a = len(l)
    b = []
    b.append(tmplist[a/10])
    b.append(tmplist[a*25/100])
    b.append(tmplist[a/2])
    b.append(tmplist[a*75/100])
    b.append(tmplist[a*9/10])
    return b

cmd = toolcmd()
for f in os.listdir(self.getcfgvalue(self.getsection(), "wavs")):
    if f.endswith('wav') and "orig" not in f:
        print f
        cmdp = os.popen(cmd + " " + \
            self.getcfgvalue(self.getsection(), \
                "wavs") + "/" + f)
        # get values from pipe in ms
        values = []

```

```

    for i in cmdp.readlines():
        values.append(i[:-2])

    segmentlist = loadsegmentfile(f.split(".")[0])
    pre = segmentlist[0][0]
    for i, j in segmentlist[1:]:
        tmplist = []
        for k in range(int(float(pre)*1000), int(float(i)*1000)):
            tmplist.append(values[k])
        writeresults(j[:-1], genresultlist(tmplist), f)
        pre = i

def __init__(self, cfgfile, section):

    self.setsection(section)
    self.config = self.loadconfig(cfgfile)

def usage():
    print
    print "-h, --help      for help"
    print "-c=<fn.ini>      set config-file"
    print "-t=<section>      type [i.e. test, ims, notebook] " \
        + "(as described in config-file)"
    print "-f, --fileinfo    write fileinfo in datafiles"
    print "-v, --verbose     be more verbose"
    print

if __name__ == "__main__":
    try:
        opts, args = \
            getopt.getopt(sys.argv[1:], \
                "hc:t:vf", \
                ["help", "config=", "type=", "verbose", "fileinfo"])
    except getopt.GetoptError:
        # print help information and exit:
        usage()
        sys.exit(2)

    # defaults
    cfgfile = "config.ini"
    section = "test"

    for o, arg in opts:
        if o in ("-h", "--help"):
            usage()
            sys.exit()
        if o in ("-c", "--config"):
            cfgfile = arg[1:]
        if o in ("-t", "--type"):
            section = arg[1:]

    t = PowerProfile(cfgfile, section)

```

```

for o, arg in opts:
    if o in ("-v", "--verbose"):
        t.setcfgvalue("defaults", "verbose", "true")
    if o in ("-f", "--fileinfo"):
        t.setcfgvalue("defaults", "fileinfo", "true")

# iterate ...
t.iterateWavDirectory()

```

A.2.3. genmeanfile.py

```
#!/usr/bin/env python
```

```
"""generate from energy-result-files a scheme file"""
```

```
__revision__ = "$Rev: 119 $"
```

```
import sys, os
import getopt
```

```
from myconfig import MyConfig
```

```
class GenMean(MyConfig):
```

```
    def iterateResultDirectory(self):
```

```
        outp = open(self.getcfgvalue(self.getsection(), "destination") \
                    + "/" + "energy.scm", "w")
        outp.write("## This file contains energy-means for one voice!\n");

        outp.write("(set! energy '(\n")
```

```
        for f in os.listdir(self.getcfgvalue(self.getsection(), "destination")):
            if f.endswith('energy') and "_" not in f:
                fp = open(self.getcfgvalue(self.getsection(), "destination") \
                        + "/" + f)
                phonem = f.split(".")[0]
                mean = {}
                count = {}
                for line in fp.readlines():
                    for i, j in [(10, 0), (25, 1), (50, 2), (75, 3), (90, 4)]:
                        value = line.split(" ")[j]
                        if mean.has_key(i):
                            a = mean[i]
                            a += float(value)
                            mean[i] = a
                            count[i] = int(count[i]) + 1
                        else:
                            mean[i] = float(value)
                            count[i] = 1
                outp.write('(' + phonem + ' ')
```

```

        outp.write(" ".join([" %s" % \
                               (str(float(mean[x])/float(count[x]))) \
                               for x in [10, 25, 50, 75, 90]]) \
                    )
        outp.write(")\n")
    outp.write(")\n")
outp.close()

def __init__(self, cfgfile, section):

    self.setsection(section)
    self.config = self.loadconfig(cfgfile)

def usage():
    print
    print "-h, --help      for help"
    print "-c=<fn.ini>      set config-file"
    print "-t=<section>     type [i.e. test, ims, ...] (as described in config-file)"
    print

if __name__ == "__main__":
    try:
        opts, args = getopt.getopt(sys.argv[1:], "hc:t:", ["help", "config=", "type="])
    except getopt.GetoptError:
        # print help information and exit:
        usage()
        sys.exit(2)

    # defaults
    cfgfile = "config.ini"
    section = "test"

    for o, a in opts:
        if o in ("-h", "--help"):
            usage()
            sys.exit()
        if o in ("-c", "--config"):
            cfgfile = a
        if o in ("-t", "--type"):
            section = a[1:]

    t = GenMean(cfgfile, section)
    t.iterateResultDirectory()

```

A.2.4. genexp.sh

```

#!/bin/bash

# script for generating a new perception experiment
#
# Andreas Madsack <madsacas@ims...>

```

```

#
# $Rev: 125 $
#

EXPDIR='pwd'/exp
PERZLOG=$EXPDIR/perz.log

# test if exp already exists!
[-e $EXPDIR ] && echo "'$EXPDIR' already exists!!" && exit 1

mkdir exp

touch $PERZLOG
chmod 600 $PERZLOG

RANGE=6

for j in @$; do
(
  cd $j
  echo "-----" >> $PERZLOG
  echo "Generating perceptiontest files for $i" >> $PERZLOG

  for i in `find . -name \*-amp.wav`; do
    W=`basename $i -amp.wav`
    WX="$W"X

    number=$RANDOM
    let "number %= $RANGE"
    case "$number" in "4") let "number=0";; "5") let "number=1";;esac

    case "$number" in
      "0" )
        echo $W ": 1[amp] 2[orig] " >> $PERZLOG
        cp "`dirname $i'/$W-amp.wav" "$EXPDIR/$W-1.wav"
        cp "`dirname $i'/$W.wav" "$EXPDIR/$W-2.wav"
        ;;
      "1" )
        echo $W ": 1[orig] 2[amp] " >> $PERZLOG
        cp "`dirname $i'/$W.wav" "$EXPDIR/$W-1.wav"
        cp "`dirname $i'/$W-amp.wav" "$EXPDIR/$W-2.wav"
        ;;
      "2" )
        echo $W ": 1[amp] 2[amp] " >> $PERZLOG
        cp "`dirname $i'/$W-amp.wav" "$EXPDIR/$W-1.wav"
        cp "`dirname $i'/$W-amp.wav" "$EXPDIR/$W-2.wav"
        ;;
      "3" )
        echo $W ": 1[orig] 2[orig] " >> $PERZLOG
        cp "`dirname $i'/$W.wav" "$EXPDIR/$W-1.wav"
        cp "`dirname $i'/$W.wav" "$EXPDIR/$W-2.wav"
        ;;
    esac
  done
done

```

```

number2=$number
while [ "$number2" == "$number" ]
do
    number2=$RANDOM
    let "number2 %= $RANGE"
    case "$number2" in "4") let "number2=0";; "5") let "number2=1";;esac
done

case "$number2" in
"0" )
    echo $WX ": 1[amp] 2[orig] " >> $PERZLOG
    cp "dirname $i'/$W-amp.wav" "$EXPDIR/$WX-1.wav"
    cp "dirname $i'/$W.wav" "$EXPDIR/$WX-2.wav"
    ;;
"1" )
    echo $WX ": 1[orig] 2[amp] " >> $PERZLOG
    cp "dirname $i'/$W.wav" "$EXPDIR/$WX-1.wav"
    cp "dirname $i'/$W-amp.wav" "$EXPDIR/$WX-2.wav"
    ;;
"2" )
    echo $WX ": 1[amp] 2[amp] " >> $PERZLOG
    cp "dirname $i'/$W-amp.wav" "$EXPDIR/$WX-1.wav"
    cp "dirname $i'/$W-amp.wav" "$EXPDIR/$WX-2.wav"
    ;;
"3" )
    echo $WX ": 1[orig] 2[orig] " >> $PERZLOG
    cp "dirname $i'/$W.wav" "$EXPDIR/$WX-1.wav"
    cp "dirname $i'/$W.wav" "$EXPDIR/$WX-2.wav"
    ;;
esac

done

) # end of subshell
done

(
    cd $EXPDIR
    touch `find . -name \*-\.wav | grep -v X`
)

cp playnow.sh $EXPDIR
cp danke.wav $EXPDIR

```

A.2.5. gen_bench.py

```

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

"""Testbench for evaluation"""

```

```

__revision__ = "$Rev: 76 $"

import sys, re, os, md5, getopt

class TestBench:

    sable_readtemplate = []
    config = {}
    scmfilep = 0
    thisdir = ""
    key = ""
    bruteforce = True
    makedirs = True

    def enable_makedirs(self):
        # self.makedirs = True
        print "not supported ATM."

    def enable_bruteforce(self):
        self.bruteforce = True

    def disable_makedirs(self):
        self.makedirs = False

    def disable_bruteforce(self):
        self.bruteforce = False

    def setprefilename(self, pf):
        self.config["prefilename"] = pf

    def filename(self):
        if self.makedirs:
            return self.key + "/" + self.key
        else:
            return self.key

    def scheme_generate(self, line):
        def filter(str):
            for i,j in (( ' ', '-'), ('\n', ""), ('\r', "")):
                str = str.replace(i,j)
            return str

        self.key = md5.new(line).hexdigest()

        if self.makedirs:
            os.mkdir(self.thisdir + "/" + self.key)
            os.symlink(self.key, self.thisdir + "/" + filter(line))

        # start scheme gen:
        if self.makedirs:
            x = self.thisdir + "/" + self.key

```

```

self.scmfilep = open(x + '/' + self.config["scmfilename"], "w")

# pre scheme
inp = open(self.config["prefilename"])
for i in inp.readlines():
    self.scmfilep.write(i)
inp.close()
self.scmfilep.write("\n")

self.scmfilep.write('(setq ampnorm_activate nil)' + "\n")
self.scmfilep.write('(set! tts_filename "' + self.filename() + '")' + "\n")
self.scmfilep.write('(set! utt1 (utt.synth (Utterance Text "'
    + line + '")))') + "\n")
self.scmfilep.write('(save_tts_output_all utt1)' + "\n")

self.scmfilep.write('(setq ampnorm_activate t)' + "\n")
self.scmfilep.write('(set! tts_filename "' + self.filename() + '-amp")' + "\n")
self.scmfilep.write('(set! utt1 (utt.synth (Utterance Text "'
    + line + '")))') + "\n")
self.scmfilep.write('(save_tts_output_all utt1)' + "\n")

if self.bruteforce:
    self.scmfilep.write('(print "--")' + "\n")
    self.scmfilep.write('(setq ampnorm_activate nil)' + "\n")
    self.scmfilep.write("(Param.set 'Synth_Method 'psm2_plus )" + "\n")
    self.scmfilep.write('(set! i 0)' + "\n")

    self.scmfilep.write('(while (> (count_cands utt1) i)' + "\n")
    self.scmfilep.write('(set! i (+ i 1))' + "\n")
    self.scmfilep.write('(Wave_Synth utt1)' + "\n")
    self.scmfilep.write('(set! tts_filename (string-append "'
        + self.filename() + '" "-" i))' + "\n")
    self.scmfilep.write('(save_tts_output_all utt1)' + "\n")
    self.scmfilep.write(')' + "\n")
    self.scmfilep.write("(Param.set 'Synth_Method 'psm2 )" + "\n")

# post scheme
self.scmfilep.write("\n")
inp = open(self.config["postfilename"])
for i in inp.readlines():
    self.scmfilep.write(i)
inp.close()

self.scmfilep.close()

def parse_sentencesfile(self, filename):
    type = "scheme" # default
    inp = open(filename)
    for line in inp.readlines():
        p = re.compile('^ \ *%\. *')
        if len(p.findall(line)) < 1: # no commented line
            if type == "scheme":
                self.scheme_generate(line)
            else:

```

```

        x = line.split(":")
        if "format" in x[0].lower():
            if "scheme" in x[1].lower():
                type = "scheme"
inp.close()

def directory_create(self):

    def directory_check():
        def isInt(s):
            try:
                i = int(s)
            except ValueError:
                i = None
            return i

        for root, dirs, files in os.walk('.'):
            try:
                x = []
                for i in dirs:
                    a = i.split("-")
                    if isInt(a[0]):
                        x.append(a[0])
                if int(max(x)):
                    return max(x)
            except:
                return 0

    x = self.config["sentencesfilename"].split(".")
    self.thisdir = '%03d-%s' % ((int(directory_check()+1), x[-2])
    os.mkdir(self.thisdir)

def __init__(self):

    self.config["prefilename"] = "pre.scm"
    self.config["postfilename"] = "post.scm"
    self.config["scmfilename"] = "foo.scm"

def run(self, filename):

    self.config["sentencesfilename"] = filename

    self.directory_create()

    type = self.parse_sentencesfile(self.config["sentencesfilename"])

    print "# Run with:"
    print "./festival.sh " + self.thisdir
    print "./f0.sh " + self.thisdir
    print "./amp.sh " + self.thisdir

```

```

if __name__ == "__main__":

    try:
        opts, args = \
            getopt.getopt(sys.argv[1:], \
                "hbdp=", \
                ["help", "bruteforce", "makedirs", "prefilename="])
    except getopt.GetoptError:
        print "TODO: write usage"
        sys.exit(2)

    t = TestBench()

    for o,a in opts:
        if o in ("--makedirs", "-d"):
            t.disable_makedirs()
        if o in ("--bruteforce", "-b"):
            t.disable_bruteforce()
        if o in ("--prefilename", "-p"):
            t.setprefilename(a)

    t.run(args[0])

```

A.2.6. genR.py

```

#!/usr/bin/env python

"""Convert energy-result-files in R-readable-format"""

__revision__ = "$Rev: 119 $"

import sys, os
import getopt

from myconfig import MyConfig

class GenR(MyConfig):

    def iterateResultDirectory(self):

        def isInt(s):
            try:
                i = int(s)
            except ValueError:
                i = None
            return i

        for f in os.listdir(self.getcfgvalue(self.getsection(), "destination")):
            if f.endswith('energy') and "_" not in f:
                fp = open(self.getcfgvalue(self.getsection(), "destination") \
                    + "/" + f)
                phonem = f.split(".")[0]

```

```

phonemorig = phonem
for i, j in [(": ", "D"), ("~", "T"), ("@", "AT")]:
    phonem = phonem.replace(i, j)
if isInt(phonem[0]):
    phonem = "X" + phonem
a = {}
for line in fp.readlines():
    for i, j in [(10, 0), (25, 1), (50, 2), (75, 3), (90, 4)]:
        value = line.split(" ")[j]
        if a.has_key(i):
            b = a[i]
            b.append(value)
            a[i] = b
        else:
            b = []
            b.append(value)
            a[i] = b

print 'png(file="' \
+ self.getcfgvalue(self.getsection(),"destination") + "/" \
+ phonemorig + '_bpx.png', height = 1500, width = 1500)'

for i in [10, 25, 50, 75, 90]:
    print phonem + "_" + str(i) + " = c(" ,
    print ", ".join(["%s" % (x) for x in a[i]]),
    print ") "

print 'P <- boxplot(' + phonem + '_10,' + phonem + '_25,' \
+ phonem + '_50,' + phonem + '_75,' \
+ phonem + '_90,plot=FALSE)'
print 'P$names <- c("10%", "25%", "50%", "75%", "90%")'
print 'bpx(P, main="' + phonemorig + '",ylim=c(0,8000))'
print 'dev.off()'

def __init__(self, cfgfile, section):

    self.setsection(section)
    self.config = self.loadconfig(cfgfile)

def usage():
    print
    print "-h, --help      for help"
    print "-c=<fn.ini>      set config-file"
    print "-t=<section>     type [i.e. test, ims, notebook] "\
    + "(as described in config-file)"
    print

if __name__ == "__main__":
    try:
        opts, args = getopt.getopt(sys.argv[1:], "hc:t:", \
            ["help", "config=", "type="])
    except getopt.GetoptError:
        # print help information and exit:
        usage()

```

```

    sys.exit(2)

# defaults
cfgfile = "config.ini"
section = "test"

for o, a in opts:
    if o in ("-h", "--help"):
        usage()
        sys.exit()
    if o in ("-c", "--config"):
        cfgfile = a
    if o in ("-t", "--type"):
        section = a[1:]

t = GenR(cfgfile, section)
t.iterateResultDirectory()

```

A.2.7. evaluation.py

```

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

__revision__ = "$Rev: 119 $"

import os, sys, md5

def evaluation(switcher):

    def readperzexp():
        f = open("perzexp.txt")
        i = 0
        lval = {}
        for line in f.readlines():
            if '%%' not in line:
                i += 1
                lval[md5.new(line).hexdigest()] = i
        return lval

    def gentable():
        f = open("perz.log")
        perz = {}
        for i in f.readlines():
            x = []
            j = i.split(" : ")
            if len(j)>1:
                key = j[0]
                j = j[1].split(" ")
                if "amp" in j[0]:
                    x.append("a")
                elif "orig" in j[0]:
                    x.append("o")

```

```

        if "amp" in j[1]:
            x.append("a")
        elif "orig" in j[1]:
            x.append("o")
        perz[key] = x
    f.close()
    return perz

def genid(name):
    x = name[0].upper()
    idvals = ids.items()
    used = []
    for i in idvals:
        used.append(i[1])
    i = 1
    while x+str(i) in used:
        i += 1
    return x+str(i)

ids = {}
md5table = readperzexp()
perz = gentable()
eval = {}
users = {}
for f in os.listdir("."):
    if f.endswith('.log') and "perz.log" not in f:
        thiseval = {}
        thisplay = {}
        fx = open(f)
        for i in fx.readlines():
            if "=" in i or "<" in i or ">" in i:
                j = i[:-1].split(" - ")
                j = j[1].split(": ")
                key = j[0]
                val = j[1][1:] # get only first character
                a = ""
                if val == "=":
                    if perz[key][0] == perz[key][1]:
                        a = "i+"
                    else:
                        a = "i-"
                elif val == "<":
                    if perz[key][0] != perz[key][1]:
                        a = perz[key][0] + "+"
                    else:
                        a = perz[key][0] + "-"
                elif val == ">":
                    if perz[key][0] != perz[key][1]:
                        a = perz[key][1] + "+"
                    else:
                        a = perz[key][1] + "-"
                else:
                    print "Error: no <, > or = found !!!"

```

```

        sys.exit(1)

        # save always last one!
        thiseval[key] = a
    if "Play" in i:
        j = i[:-1].split(" - ")
        j = j[1].split(": ")
        key = j[1].split("-")
        key = key[0]
        if len(key)>0:
            if thisplay.has_key(key):
                thisplay[key] = thisplay[key] + 1
            else:
                thisplay[key] = 1
    fx.close()
    # copy all values in thiseval to eval-list:
    uname = f.split(".")[0].split("-")[2]
    cnt = {}
    for i in ('a+', 'a-', 'o+', 'o-', 'i+', 'i-'):
        cnt[i]=0;
    for i in thiseval.keys():
        if not eval.has_key(i):
            eval[i] = []
            x = []
            x.append(uname)
            x.append(thiseval[i])
            if thisplay.has_key(i):
                x.append(thisplay[i])
            else:
                x.append(0)
            eval[i].append(x)
            cnt[thiseval[i]] = cnt[thiseval[i]] + 1
    users[uname] = cnt

b = users.keys()
b.sort()

for j in b:
    ids[j] = genid(j)

if "user" in switcher:
    print "nr.          name id  a+   a-   i+   i-   o+   o-   ao-   sum"
    print 77*"-"
    nr = 0
    for j in b:
        nr += 1
        print "%2d.%18s %2s  " % (nr, j, ids[j]),
        x = users[j]
        a = x.keys()
        a.sort()
        for i in a:

```

```

    print "%2i    " % (x[i]),
    print "    %2i    " % (x["o-"]+x["a-"]),
    print "    %2i    " % (x["o-"]+x["a-"]+x["i-"]+x["o+"]+x["a+"]+x["i+"]),
    print

```

elif "data" in switcher:

```

a = md5table.items()                # important!!
a.sort(lambda a, b: cmp( b[1], a[1] )) # sort by value
a.reverse()

```

for i in a:

```

    values = eval[i[0]] + eval[i[0]+"X"]

```

for (k,j,p) in values:

```

    print "%02d " % (md5table[i[0]]),
    print "%2s " % (ids[k]),
    print "%2s " % (j),
    for l in ('a+', 'a-', 'i+', 'i-', 'o+', 'o-'):
        if j in l:
            print "1 ",
        else:
            print "0 ",
    print "%02d " % p,
    print "%s" % (i[0]),
    print "%s" % (k),
    print

```

elif "sentences" in switcher:

```

print "nr.    a+    a-    i+    i-    o+    o-    ao-    sum    key"

```

```

a = md5table.items()                # important!!
a.sort(lambda a, b: cmp( b[1], a[1] )) # sort by value
a.reverse()

```

```

initcnt = {}

```

```

for j in ('a+', 'a-', 'o+', 'o-', 'i+', 'i-'):
    initcnt[j] = 0;

```

for i in a:

```

    print "%2d.    " % (md5table[i[0]]),
    values = eval[i[0]] + eval[i[0]+"X"]

```

```

    cnt = initcnt

```

```

    cnt = {}

```

```

    for j in ('a+', 'a-', 'o+', 'o-', 'i+', 'i-'):
        cnt[j] = 0;

```

for (k,j,p) in values:

```

    cnt[j] = cnt[j] + 1

```

```

b = cnt.keys()

```

```

    b.sort()
    for j in b:
        print "%3i  " % (cnt[j]),

    print "   %3i  " % (cnt["o-"]+cnt["a-"]),
    print "   %3i  " % (cnt["o-"]+cnt["a-"]+cnt["i-"]+cnt["o+"]+cnt["a+"]+cnt["i+"]),

    print "  %s" % (i[0]),
    print

elif "foo" in switcher:

    a = eval.keys()
    a.sort()
    cnt = {}
    for i in a:
        for j in eval[i]:
            if not cnt.has_key(j):
                cnt[j] = 0
            cnt[j] = cnt[j] + 1
    print "all: " + str(cnt)

if len(sys.argv) > 1:
    evaluation(sys.argv[1])
else:
    print "Commandline arguments: user or sentences"

```

A.2.8. genAMP-graphics.py

```

#!/usr/bin/env python

"""Generation of R-code"""

__revision__ = "$Rev: 70 $"

import sys, os, getopt

def genR(fn):

    def loadsegmentfile(filename):
        f = open(filename)
        a = []
        for i in f.readlines():
            if len(i.split(" "))>2:
                b = []
                b.append(i.split(" ")[0])
                b.append(i.split(" ")[2])
                a.append(b)
        f.close()
        return a[1:] # eliminate first element/line

```

```

def getdata(fn, n):
    inp = open(fn)
    a = "c" + str(n) + " = c("
    for line in inp.readlines():
        x = line.split(" ")
        if len(x[0]) == 0:
            x[0] = x[1]
        if float(x[0]) < 1:
            x[0] = "NA" # eliminate really small values
        a = a + x[0] + ", "
    a = a[:-1] + ")"
    return a

filep = open(fn[:-4] + ".R", "w")

# remove -amp from filename
filep.write(getdata(fn[:-4] + ".energy", 1) + "\n")
filep.write(getdata(fn + ".energy", 2) + "\n")

filep.write("b1 = seq (.01,length(c1)/100,0.01)\n")
filep.write("b2 = seq (.01,length(c2)/100,0.01)\n")

# calculate y-lim
filep.write("wx <- length(c1)*3+"\n");
filep.write("png(file=\" + fn[:-4] + ".amp.png\", height = 1500, width = wx)\n")
filep.write("plot(b1,c1,ylim=c(min(c(c1,c2)),max(c(c1,c2))),type='l',col='blue')\n")
filep.write("lines(b2,c2,type='l',col='red')\n")
filep.write('legend("topright",lty = 1,cex=2,c("original","ampnorm"),'\
            + 'col=c("blue","red"),title="Energy-values")'+ "\n")

segmentlist = loadsegmentfile(fn[:-4] + ".lab")
pre = segmentlist[0][0]
for i, j in segmentlist[1:]:
    filep.write('abline(v=' + str(i) + '*10,lty=3, col="green")'+ "\n")
    filep.write('text(' + str(i) + '*10,-40,"' + j + '", col="black",adj = c(1.3,0),cex=2)+' "\n")

filep.close()

if __name__ == "__main__":
    genR(sys.argv[1])

```

A.2.9. genf0-graphics.py

```

#!/usr/bin/env python

"""Generation of R-code"""

import sys, math

__revision__ = "$Rev: 84 $"

```

```

def genR(fn):
    def getdata(fn):
        inp = open(fn)
        a = []
        for line in inp.readlines():
            line = line.replace("\n","").replace("\r","")
            x = line.split(" ")
            if len(x[0]) == 0:
                x[0] = x[1]
            if float(x[0]) < 1:
                x[0] = "NA" # eliminate really small values
            a.append(x[0])
        return a

    def mkRlist(l,n):
        a = n + " = c("
        for i in l:
            a = a + i + ", "
        a = a[:-1] + ")"
        return a

    filep = open(fn + ".f0.R","w")

    origf0 = getdata(fn + ".f0")
    filep.write(mkRlist(origf0, "origf0") + "\n")

    pdaf0 = getdata(fn + ".pda.f0")
    filep.write(mkRlist(pdaf0, "pdaf0") + "\n")

    wsurff0 = getdata(fn + ".wsurf.f0")
    filep.write(mkRlist(wsurff0, "wsurff0") + "\n")

    festf0 = getdata(fn + ".fest.f0")
    filep.write(mkRlist(festf0, "festf0") + "\n")

    filep.write("b1 = seq (.01,length(origf0)/100,0.01)\n")
    filep.write("b2 = seq (.01,length(pdaf0)/100,0.01)\n")
    filep.write("b3 = seq (.01,length(wsurff0)/100,0.01)\n")
    filep.write("b4 = seq (.01,length(festf0)/100,0.01)\n")

    filep.write("wx <- max(length(origf0),length(pdaf0),length(wsurff0), " \
        + "length(festf0))*10" + "\n")

    # f0-plotting
    filep.write("png(file=\"\" + fn + ".f0.png\", height = 1500, width = wx)\n")

    filep.write("plot(b1,origf0,ylim=c(0,500),type='l',col='grey')\n")
    filep.write("lines(b2,pdaf0,type='l',col='blue')\n")
    filep.write("lines(b3,wsurff0,type='l',col='green')\n")
    filep.write("points(b4,festf0,type='o',col='red')\n")
    filep.write('legend("topright",cex=2,lty = c(0,1,1,1),pch=c(1,-1,-1,-1), ' \
        + 'c("utt-save","pda","wsurf","save.f0"), ' \
        + 'col=c("red","blue","green","grey"),title="F0")' + "\n")

```

```

filep.write("dev.off()\n")

# plot wsurf + fest with area between
filep.write("png(file=\"\" + fn + ".f0-dist.png\", height = 1500, width = wx)\n")
filep.write("plot(b1,origf0,ylim=c(0,500),type='l',col='grey')\n")
filep.write("lines(b3,wsurff0,type='l',col='green')\n")

filep.write("for(i in seq(2, length(b1)))"+"\\n")
filep.write("  if (!any(is.na(wsurff0[c(i-1, i)]))"+"\\n")
filep.write("    polygon(c(b1[i-1], b1[i], b3[i], b3[i-1]),"+"\\n")
filep.write("      c(origf0[i-1], origf0[i], wsurff0[i], wsurff0[i-1]),\\n")
filep.write("        col = \"green\", border = 0)'+\"\\n")

filep.write('legend("topright",cex=2,lty = c(1,1),pch=c(-1,-1),' \
+ 'c("wsurf","save.f0"),col=c("green","grey"),title="F0")' + "\\n")
filep.write("dev.off()")

filep.close()

# not the best solution!!
a = fn.split("-")
filep = open(a[0] + ".f0.data","a")

filep.write("# filename RMSE diff-mean diff-add F0-curve-variance count-NAs\\n")
filep.write(fn + " ")
sum = 0
sum2 = 0
for i in range(1,min(len(wsurff0),len(festf0))):
  if not ((wsurff0[i] in 'NA') or (festf0[i] in 'NA')):
    sum += abs(float(wsurff0[i]) - float(festf0[i]))
    sum2 += math.pow(float(wsurff0[i]) - float(festf0[i]),2)

filep.write(str(math.sqrt((sum2 / i))) + " ")
filep.write(str(sum / i) + " ")
filep.write(str(sum) + " ")

diff = 0
na = 0
pre = wsurff0[0]
if pre == 'NA':
  na += 1
for i in wsurff0[1:]:
  if i == 'NA':
    i = 0
    na += 1
  if pre == 'NA':
    pre = 0
  diff += math.fabs(float(pre) - float(i))
  pre = i

filep.write(str(diff) + " ")
filep.write(str(na) + " ")
filep.write("\\n")

```

```

filep.close()

if __name__ == "__main__":

    if len(sys.argv)>1:
        fn = sys.argv[1]
        genR(fn)
    else:
        print "Filename needed!! (without extension!)"

```

A.3. Code

A.3.1. amplitude normalisation (part of ims_post_sigv.cc)

```

/**
 * Linear Interpolation
 *
 * @param x1 x of point 1
 * @param y1 y of point 1
 * @param x2 x of point 2
 * @param y2 y of point 2
 * @param x x of wanted y
 *
 * @return (int)
 */
int linear_interpolation(int x1,int y1,int x2,int y2,int x)
{
    int y;

    y = y1 + (((x - x1) * (y2 - y1))/(x2 - x1));

    return y;
}

/**
 * simple rescale-function
 *
 * At the moment only channel one is rescaled!!
 *
 * @param w pointer of wave
 * @param fc track with normalisation values
 *
 * @return 0 on success, error code otherwise
 */
int rescale(EST_Wave *w, EST_Track &fc)
{
    int nsmax=0, wavmax=0;
    float factor=1;

    if (fc.length() != (*w).num_samples())
    {

```

```

    cout << "length(Track) != length(Wave)" << endl;
    return 1;
}

// to prevent oversampling
for ( int k = 1; k<fc.length(); k++ )
{
    int a = (*w).a_no_check(k,0);
    int ns = (int)(a * fc.a(k,0));

    if(abs(ns) > nsmax)
        nsmax = abs(ns);
    if(abs(a) > wavmax)
        wavmax = abs(a);
}

if (nsmax > 32766)
{
    cout << "Warning: AmpNorm: nsmax > 32766" << endl;
    nsmax = 32766;
}

// factor for all samples to reach same loudness as original wave
factor = (float) wavmax / (float) nsmax;
*cdebug << "nsmax: " << nsmax << " -> factor: " << factor << endl;

for ( int k = 1; k<fc.length(); k++ )
{
    int ns = (int)((*w).a_no_check(k,0) * fc.a(k,0) * factor);

    if (ns < -32766)
        (*w).a_no_check(k,0) = -32766;
    else if (ns > 32766)
        (*w).a_no_check(k,0) = 32766;
    else
        (*w).a_no_check(k,0) = ns;
}

cout << "Rescale erfolgreich!" << endl;

return 0;
}

/**
 * filter for all phonemes we DON'T want to normalise
 * atm: pauses and all plosives
 *
 * @param s
 *
 * @return 0 or 1 to indicate a filtered character
 */
int charfilter(EST_String s)
{

```

```

if (!strcmp(s,"_") ||
    !strcmp(s,"p") || !strcmp(s,"b") ||
    !strcmp(s,"k") || !strcmp(s,"g") ||
    !strcmp(s,"t") || !strcmp(s,"d")
    )
    return 1;
else
    return 0;
}

/**
 * Amplitude Normalisation
 *
 * @param utt
 *
 * @return
 */
LISP ampnorm(LISP utt)
{
    EST_Utterance* u = get_c_utt(utt); // utterance from arg
    EST_Wave *w; // Wave (from utterance)
    LISP _energyparms = NIL; // Energy parameter set
    EST_Track fill; // energy-Track
    // my frame_factor_values (same for calculation of mean-values)
    float frame_factor = 2.0 * -1.0 * 0.03;
    EST_WindowFunc *wf; // for the hanning-window
    EST_StrList sl;

    EST_Track normalise; // normalise-value-Track
    float wlen; //length of wave (in seconds)
    int wsamples; // length of wave (in samples)
    int prex = -1; // previous x-value (start with -1)
    float prec = -1, prem = -1; // previous calculated and measured values
    float tmp;

    float cv[5]; // calculated RMS-values
    float mp[5] = {0.1,0.25,0.5,0.75,0.9}; // the 5 per cent points
    int sp[5]; // samples at position
    float mv[5]; // measured values
    float l; // length in seconds
    int ls; // length in samples
    int a,b; // which per cent point
    int x1,x2,y1,y2; // variables for linear interpolation

    LISP iii;

    float sampleRate;

    // test if ampnorm_activate is set:
    iii = rintern("ampnorm_activate");
    if(symbol_boundp(iii,current_env) == NIL)
    {
        *cdebug << "DEBUG: Amplitude normalisation deactivated! (1)" << endl;
    }
}

```

```

    return NIL;
}
if(symbol_value(iii,current_env) == NIL)
{
    *cdebug << "DEBUG: Amplitude normalisation deactivated! (2)" << endl;
    return NIL;
}

cout << "Amplitude normalisation" << endl;

// get energy-data
_energyparams = siod_get_lval("ampnorm_energy",
                             "AmpNorm: No energy-values set!");

if(u->relation_present("Wave") == true)
{
    w = get_utt_wave(u);

    // energy
    wf = EST_Window::creator("hanning");
    fill.resize((int)ceil(*w).end() / 0.001, 0);
    fill.fill_time(0.001);
    sl.append("energy");
    fill.resize(-1, sl);

    energy(*w, fill, frame_factor, wf);

    // normalise-init
    wlen = (*w).end();
    wsamples = (*w).num_samples();
    sampleRate = (float) (*w).sample_rate();
    normalise.resize(wsamples,1);
}
else
{
    cerr << "No wave present in ampnorm!!" << endl;
    return NIL;
}

for (EST_Item* seg = u->relation("Segment")->first();
     seg; seg = next(seg))
{
    float start = ffeature(seg,"segment_start").Float();
    int startsample = (int)( start * wsamples / wlen);
    float end = ffeature(seg,"segment_end" ).Float();
    int endsample = (int)( end * wsamples / wlen);

    l = end - start;
    ls = endsample - startsample;

    for(int k = 0; k < 5; k++)
    {
        cv[k] = atoi(get_c_string(siod_nth(
            k,car(cdr(siod_assoc_str(seg->name(),_energyparams))))));
    }
}

```

```

mv[k] = fill.a(int(ceil((1 * mp[k] + start ) * 1000)));
sp[k] = (int)(ls * mp[k] + startsample);
}

// from 90%-point to 10%-point
if(prex > 0)
{
  for(int k=prex;k<sp[0];k++)
  {
    x1 = prex;
    x2 = (int)(ls * mp[0] + startsample);

    y1 = (int)(prec / prem * 10000);
    y2 = (int)(cv[0] / mv[0] * 10000);

    // use charfilter and filter for last phonem
    if(charfilter(seg->name()) || seg->length() < 3)
    {
      if(k > startsample)
        y1 = 10000;
      else
        x2 = startsample;
      y2=10000;
      cv[4]=1; mv[4]=1;
    }

    tmp = (float)linear_interpolation(x1,y1,x2,y2,k);
    if(tmp != 1)
      tmp = tmp / 10000;
    normalise.a(k) = tmp;
  }
}

// from 10% to 90%-point!!
a = 0;
b = 0;
for(int k=startsample;k<sp[4];k++)
{
  if(a<4 && k == sp[b])
  {
    prex = -2;
    if(a != b)
      a++;
    b++;
  }

  x2 = (int)(ls * mp[b] + startsample);
  y2 = (int)(cv[b] * 10000 / mv[b] );

  if(prex == -1)
  {
    x1 = startsample;
    y1 = (int)(cv[a] * 10000 / fill.a(start * 1000) );
  }
}

```

```

else
{
  x1 = (int)(ls * mp[a] + startsample);
  y1 = (int)(cv[a] * 10000 / mv[a] );
}

// use charfilter and filter for last phonem
if(charfilter(seg->name()) || seg->length() < 3)
{
  y1=1; y2=1;
  cv[4]=1; mv[4]=1;
}

// preventing the 0%-10%-case not in first segment, but in all other:
if(prex < 0)
{
  tmp = (float)linear_interpolation(x1,y1,x2,y2,k);
  if(tmp != 1)
    tmp = tmp / 10000;
  normalise.a(k) = tmp;
}
}

prec = cv[4];
if(prec == 0)
  prec = 1;
prem = mv[4];
prex = sp[4];

if(u->relation("Segment")->last() == seg)
{
  // last 90%-endsample
  for(int k=prex;k<wsamples;k++)
  {
    x1 = prex;
    x2 = wsamples;

    y1 = (int)(prec / prem * 10000);
    y2 = y1;

    if(charfilter(seg->name()))
    {
      y1=1; y2=1;
    }

    tmp = (float)linear_interpolation(x1,y1,x2,y2,k);
    if(tmp != 1)
      tmp = tmp / 10000;
    normalise.a(k) = tmp;
  }
}
}

```

```

    rescale(w,normalise);

    return NIL;
}

```

A.3.2. framedistance (part of ims_psm_tools.cc)

```

/**
 * Frame distance. Differs from other version in that it stops the calculation
 * if the cost is already too high.
 * @param currentBest The currently best cost.
 * @param a
 * @param ai Index into track a.
 * @param b
 * @param bi Index into track b.
 * @param wghts A vector of floats containing weights.
 * @param f0Weight Importance of F0.
 * @return
 */
float frameDistance(float currentBest,
                   const EST_Track& a, int ai,
                   const EST_Track& b, int bi,
                   const EST_FVector& wghts,
                   float f0Weight)
{
    // added for f0WeightFactor: [AM]
    ims_psm ::PSMCorpus* currentCorpus =
        ims_psm ::PSMCorpusManager::getInstance()->getCurrentCorpus();

    if ((ai < 0) || (ai >= a.num_frames()) ||
        (bi < 0) || (bi >= b.num_frames()))
    {
        cerr << "frame_distance: frames out of range" << endl;
        festival_error();
    }

    float currentBestSqr = currentBest * currentBest;

    const float f0Factor = get_param_float("f0WeightFactor",
                                           currentCorpus->m_psmParams, 0);

    float cost = (f0Weight > 0 && f0Factor > 0)
        ? f0Weight * f0Factor * fabs((a.t(ai) - ((ai > 0) ? a.t(ai - 1) : 0)) -
                                     (b.t(bi) - ((bi > 0) ? b.t(bi - 1) : 0)))
        : 0.0;

    int nChannels = a.num_channels();
    for (int k = 0; k < nChannels; k++)
    {
        float weight = wghts.a_no_check(k);
        if (weight != 0.0)
        {

```

```

float diff = a.a_no_check(ai, k) - b.a_no_check(bi, k);
diff *= weight;
cost += diff * diff;

if (cost >= currentBestSqr) // is cost already too high?
{
    return currentBest; // then stop calculation
}
}
}
return sqrt(cost);
}

```

A.3.3. psm drop candidate code (part of ims_post_sigv.cc)

```

/**
 * counts all candidates in an Unit-relation
 *
 * @param utt
 *
 * @return sum of all candidates
 */
LISP count_psm_candidates(LISP utt)
{
    int cnt=0;
    EST_Utterance* u = get_c_utt(utt);

    for (EST_Item* unitItem = u->relation(REL_NAME)->head();
         unitItem; unitItem = next(unitItem))
    {
        if (unitItem->f_present(UNIT_CNT)) // "unit_cands"
        {
            cnt += unitItem->I(UNIT_CNT);
        }
    }

    return flocons((float) cnt);
}

/**
 * copy a EST_VTCandidate and return new one
 *
 * @param oc
 *
 * @return
 */
EST_VTCandidate* copy_cand(EST_VTCandidate *oc)
{
    if (oc == NULL)
    {
        return NULL;
    }
}

```

```

EST_VTCandidate* c = new EST_VTCandidate;
c->name = oc->name;
c->s    = oc->s;
c->score = oc->score;
c->next = oc->next;
return c;
}

// not used!! -> very problematic to free the memory
void free_cands(EST_Item *item)
{
  if (item->f_present(UNIT_VAL)) // "unit_cands"
  {
    EST_VTCandidate* candList = vtcand(item->f(UNIT_VAL));
    delete candList;
  }
}

/**
 * removes one candidate from list and returns pointer to start of new list
 *
 * @param item
 * @param cnt
 *
 * @return
 */
EST_VTCandidate* remove_candidate(EST_Item *item, int cnt)
{
  int i=0;

  EST_VTCandidate *prev = 0;
  EST_VTCandidate *copy = 0;
  EST_VTCandidate *start = 0;

  if (item->f_present(UNIT_VAL)) // "unit_cands"
  {
    // foreach candidate of current item
    for (EST_VTCandidate* candList = vtcand(item->f(UNIT_VAL));
         candList; candList = candList->next)
    {
      i++;
      copy = copy_cand(candList);

      if(i == 1)
      {
        start = copy;
      }
      if (i > 1)
      {
        prev->next = copy;
      }
    }
  }
}

```

```

    prev = copy;

}
copy->next = NULL;

i = 0;
// foreach candidate of copied list
for (EST_VTCandidate* candList = start;
     candList; candList = candList->next)
{
    i++;
    if(cnt == i)
    {
        if(i > 1)
        {
            prev->next = candList->next;
        }
        else
        {
            start = candList->next;
        }
    }
    prev = candList;
}
return start;
}
return NULL;
}

/**
 * drops one candidate from psm-candidate-list with control data
 *
 * @param utt
 *
 * @return
 */
LISP drop_psm_candidate(LISP utt)
{
    EST_Utterance* u = get_c_utt(utt); // utterance from arg

    int x;

    for (EST_Item* unitItem = u->relation(REL_NAME)->head();
         unitItem; unitItem = next(unitItem))
    {
        // original doesn't exist -> first call for this item
        if ( !unitItem->f_present("unit_cands_orig") )
        {
            unitItem->set_val("unit_cands_orig",unitItem->f(UNIT_VAL));
            unitItem->set_val("unit_cands_orig_cnt",unitItem->f(UNIT_CNT));
        }
    }
}

```

```

else
{
  // restore unit_cands
  unitItem->set_val(UNIT_VAL,unitItem->f("unit_cands_orig"));
  unitItem->set_val(UNIT_CNT,unitItem->f("unit_cands_orig_cnt"));
}

// increase correct bruteforce counter
if (unitItem->f_present("bruteforce_cnt"))
{
  if (unitItem->I("bruteforce_cnt") < unitItem->I(UNIT_CNT))
  {
    unitItem->set_val("bruteforce_cnt",unitItem->I("bruteforce_cnt")+1);

    // call helperfunction for removing correct element
    unitItem->set_val(UNIT_VAL,::est_val(
      remove_candidate(unitItem,unitItem->I("bruteforce_cnt"))));

    x = unitItem->I("unit_cands_orig_cnt")-1;
    unitItem->set_val(UNIT_CNT,x);
    break;
  }
}
else
{
  unitItem->set_val("bruteforce_cnt",1);
  if(!strcmp(unitItem->S("name"),"_")) // don't remove pauses
  {
    cout << "-" << unitItem->S("name") << endl;
    unitItem->set_val("bruteforce_cnt",unitItem->I(UNIT_CNT));
  }
  if (unitItem->I("bruteforce_cnt") < unitItem->I(UNIT_CNT))
  {
    // call helperfunction for removing correct element
    unitItem->set_val(UNIT_VAL,::est_val(
      remove_candidate(unitItem,unitItem->I("bruteforce_cnt"))));
    x = unitItem->I("unit_cands_orig_cnt")-1;
    unitItem->set_val(UNIT_CNT,x);
    break;
  }
}
}
}
}

```