

Technische Universität Dresden
Fakultät Elektrotechnik und Informationstechnik
Institut für Akustik und Sprachkommunikation

Diplomarbeit

Prosodisch-phonetische Segmentierung und Annotierung von deutschen bzw. englischen Sprachkorpora

Michael Hofmann

mh21@gmx.net

8. März 2006

Verantwortlicher Hochschullehrer: Prof. Dr. R. Hoffmann

Betreuer: Dipl.-Ing. Oliver Jokisch

Technische Universität Dresden

Fakultät Elektrotechnik und Informationstechnik

Aufgabenstellung für die Diplomarbeit

für Herrn

Michael Hofmann

Thema

Prosodisch-phonetische Segmentierung und Annotierung von deutschen bzw. englischen Sprachkorpora

Zielsetzung:

Die Sprachsynthesequalität wird stark durch die zur Verfügung stehenden Sprachdatenbasen bestimmt. Sowohl bei der Bausteinauswahl in der Korpusssynthese als auch für Training bzw. gezielte Generierung von Prosodieverläufen werden umfangreiche, prosodisch und phonetisch annotierte Sprachkorpora benötigt. Die Prozesse der Segmentierung und Annotierung werden aus Aufwandsgründen zunehmend automatisiert, wobei eine phonetische Annotierung (Laute, Lautgrenzen, ggf. Akzente) heutiger Stand der Technik ist. Ansätze zur prosodischen Annotierung scheinen bisher weniger robust zu sein. Im Rahmen des EU-Forschungsprojektes Technology and Corpora for Speech to Speech Translation (TC-STAR) arbeitet die TU Dresden an entsprechend annotierten Sprachdatenbasen in UK English. Die Diplomarbeit soll vorhandene Segmentierungs- und Annotierungsalgorithmen (auf phonetischer Ebene) evaluieren bzw. anpassen, in einem Tool integrieren und die prosodische Analyse ergänzen. Referenzweise soll das Tool auch mit deutschen Sprachdaten der TU Dresden getestet werden.

Folgende Teilaufgaben sind im Rahmen der Diplomarbeit vorgesehen:

1. Literaturrecherche zu sogenannten Phoneme Labellers bzw. prosodischen Annotierern
2. Analyse der TC-STAR-Spezifikation und Grobkonzeption eines Label Tools
3. Evaluierung verschiedener Label Tools bzw. Daten für UK English bzw. Deutsch, z. B.:
 - a) Phoneme Labeller G. Strecha,
 - b) LAIP Phoneme Labeller (COST 258, Lausanne),
 - c) Fujisakiparameterbestimmung (Mixdorff, Kruschke) als Korrelat zu pot. Akzenten,
 - d) Testung der Phrasierungs- bzw. Wortakzentmarkierung von Siemens CT.
4. Implementierung ausgewählter Algorithmen und Ergänzung von Analysemerkmalen:
 - I Anpassung der Laut- bzw. Pitch Mark-Annotierung für UK English
 - II einfache prosodische Markierung (gemäß Spezifikation TC-STAR) für UK English
 - III Prosodische Zusatzmerkmale soweit möglich (z. B. Emotionsklassen, etc.)
5. Evaluation des integrierten Tool gegen manuelle Referenzdaten (English, Deutsch)
6. Dokumentation

Betreuer: Dipl.-Ing. O. Jokisch

Ausgehändigt am: 16.07.2005

Einzureichen bis: 15.01.2006

Prof. Dr.-Ing. M. Liese
Vorsitzender des
Prüfungsausschusses

Prof. Dr.-Ing. habil. R. Hoffmann
verantwortlicher Hochschullehrer

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tage eingereichte Diplomarbeit zum Thema

Prosodisch-phonetische Segmentierung und Annotierung von deutschen bzw.
englischen Sprachkorpora

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 8. März 2006

Dresden University of Technology
Department of Electrical Engineering and Information Technology
Laboratory of Acoustics and Speech Communication

Diplomarbeit

Prosodic and Phonetic Segmentation and Annotation of German and English Speech Corpora

Michael Hofmann

mh21@gmx.net

March 8, 2006

Supervisor: Prof. Dr. R. Hoffmann
Faculty Advisor: Dipl.-Ing. Oliver Jokisch

Abstract

With the development of advanced methods for speech synthesis and recognition and the availability of cheap processing power and storage solutions, the use of large speech corpora gains increasing consideration.

The elaborate annotation with phonetic, prosodic and linguistic features of such corpora can not be done manually and has to be supported by automatic means. Although the results of computer based annotation methods are still inferior to the ones obtained from the manual labeling by human experts, they can nevertheless significantly reduce the processing time and amount of correction necessary.

This thesis presents an annotation framework that combines a modular and easily extensible structure with the ability to process large amounts of data fully automatically. It implements basic reoccurring functionality like automatic wave file and phone set conversions, data storage and multi-level annotation management to ease the burden on the developer of annotation algorithms.

All necessary modules for the forced phoneme alignment of UK English voices are implemented, as are annotation modules for pitchmark determination from audio and electroglottograph signals, extraction of f0 contours, Fujisaki parameter calculation, word accent and phrase break level estimation, external lexicon lookups and others.

The used algorithms are tested and their performance is evaluated. The tests show that the results of the framework are suitable for the automatic annotation of large corpora. They nevertheless still require an human expert for quality control, parameter tuning and manual correction of the obtained results.

Contents

1	Introduction	1
1.1	Speech Production	2
1.1.1	Electroglottograph	4
1.1.2	Source Signal	5
1.2	Fujisaki Model	6
2	Signal Processing and Optimization Methods	9
2.1	Signal Processing	9
2.1.1	Windowing	9
2.1.2	Preemphasis	9
2.1.3	Linear Regression	11
2.1.4	Wavelet Transform	12
2.2	Evolutionary Optimization	16
2.2.1	Initialization	17
2.2.2	Parent Selection	17
2.2.3	Recombination	17
2.2.4	Mutation	17
2.2.5	Fitness Evaluation	18
2.2.6	Survivor Selection	18
2.2.7	Strategies	19
2.3	Artificial Neural Networks	19
2.3.1	Transfer Functions	20
2.3.2	Multilayer Feedforward Neural Networks	20
2.3.3	Training of a Neural Network	21
2.4	Dynamic Time Warping	23
3	An Automatic Annotation System	25
3.1	Requirements	25
3.2	Prerequisites	26
3.3	System Structure	26
3.4	Grapheme Phoneme Conversion	27
3.4.1	LaipTTS	28
3.4.2	Festival	29
3.4.3	Alternatives	30
3.5	Synthesis	30

3.6	Aligner Preprocessor	31
3.7	Phoneme Aligner	32
3.7.1	Supported Phoneme Aligner Programs	33
3.7.2	Performance Evaluation	33
3.8	Linguistic Annotation	34
3.8.1	Label Merger	34
3.8.2	Syllables	34
3.9	Prosodic Annotation	35
3.9.1	Rule-based Break Estimation	37
3.9.2	Derivation from Fujisaki Parameters	37
3.9.3	Neural Network Optimization	38
3.10	Polarization	40
3.11	Pitchmark Extraction from EGG Signals	42
3.11.1	Noise Filtering	43
3.11.2	Combining Pitchmarks Algorithm	44
3.12	Data Storage	48
4	Results	51
4.1	Used Databases	51
4.2	Plosive Splitter	51
4.3	Phoneme Aligner	52
4.3.1	Precision Errors	52
4.3.2	Transcription Errors	55
4.4	Prosodic Annotation	58
4.4.1	Rule-based Break Estimation	58
4.4.2	Derivation from Fujisaki Parameters	59
4.4.3	Neural Network Optimization	61
5	Discussion and Conclusions	64
A	File Formats	65
A.1	Utterance File Formats	65
A.1.1	Festival Utterance Format	65
A.1.2	XML Utterance Format	65
A.2	Segmental Information	68
A.2.1	Mbrola File Format	68
A.2.2	PhoneDat File Format	68
A.2.3	Xlabel File format	68
A.2.4	Wavesurfer File Format	69
A.2.5	.Seg File Format	69
A.3	Other Formats	70
A.3.1	.PAC File Format	70
A.3.2	PM Files	70
A.3.3	Pitchmark Files	71

A.3.4	Entropic Fundamental Frequency Contour Files	72
A.3.5	TC-STAR Phonetic Format	73
A.3.6	TC-STAR Prosodic Format	73
B	How to . . .	74
B.1	. . . Annotate a Corpus	74
B.1.1	General Notes	74
B.1.2	Wave File Preparation	74
B.1.3	Text Preparation	75
B.1.4	Forced Phoneme Alignment	75
B.1.5	Correcting Linguistic and Phonetic Tracks	77
B.1.6	Annotation	79
B.1.7	Bulk Processing	81
B.2	. . . Add a New Phone Set	84
B.3	. . . Add a New GPC, Synthesis, Processor, Aligner or Annotator Module	85
B.4	. . . Add Support for a New File Format	87
C	Programs	89
C.1	Gcida Aligner Fixes	89
C.2	Wavesurfer Pitchmark Plugin	91
C.3	Festival Utterance Viewer	93
C.4	Phone File Conversion	93
D	Tables	95
D.1	POS Tags	95
D.2	International Phonetic Alphabet	97
D.3	Spline Filter Coefficient Calculation	99
D.4	Default DTW Distance Function	100
D.5	Neural Network Training Results	100
E	License for the Developed Programs	105
E.1	GNU General Public License	105
E.1.1	Preamble	105
E.1.2	Terms And Conditions For Copying, Distribution And Modification	106
E.1.3	No Warranty	110
E.1.4	How to Apply These Terms to Your New Programs . . .	110
F	Modules and Attributes	112
F.1	Forced Alignment	112
F.1.1	Pipe.Modules	112
F.1.2	Gpcs.Type	112
F.1.3	Synthesizers.Type	113

Contents

F.1.4	Processors.Type	113
F.1.5	Aligners.Type	114
F.1.6	Evaluations.Type	114
F.2	Annotation System	114
F.2.1	Annotators.Type	114
F.3	Helper Modules	119
F.3.1	ConversionProviders.Type	119
F.3.2	PhoneFiles.Type	119
F.3.3	Text.Type	120
F.3.4	Window.Type	120
F.3.5	PhoneTransformer.Name	121
F.4	Evolutionary Optimization	121
F.4.1	AbstractBooleanGene.BooleanCombiners	121
F.4.2	AbstractBooleanGene.BooleanInitializers	121
F.4.3	AbstractBooleanGene.BooleanMutators	122
F.4.4	AbstractDoubleGene.DoubleCombiners	122
F.4.5	AbstractDoubleGene.DoubleInitializers	122
F.4.6	AbstractDoubleGene.DoubleMutators	122
F.5	Neural Networks	122
F.5.1	FieldExtractors.Type	122
G	CD-ROM Contents	125
G.1	Directory Structure	125
G.2	Java Package Structure	126

List of Tables

2.1	Window functions	10
3.1	Frequency of plosives and affricates in the <i>rob200</i> corpus	31
3.2	SNR and average pitchmark deviation for low pass filtered EGG signals	44
3.3	Commonly used relations	49
3.4	Feature types	49
3.5	Commonly used features	50
4.1	Median segmentation boundary errors for different scaling factors and time shifts	56
4.2	Number of manually labeled segment boundaries within a certain maximum deviation	57
4.3	Comparison of the mean phoneme durations of the <i>kate</i> voice and the synthetic reference for different scaling factors	57
4.4	Segmentation errors	57
4.5	Recognition rates with 25 % test set and separate break classes .	62
4.6	Recognition rates with 25 % test set and merged intermediate and major break classes	62
4.7	Recognition rates with 25 % and 75 % test sets and merged intermediate and major break classes	63
4.8	Recognition rates of the <i>kate</i> corpus for a neural network with 15 and 10 neurons in the hidden layers and a context size of ± 2 words	63
A.1	PAC file format	70
A.2	PM file format	71
B.1	Input and output parameters for the modules of the forced alignment modules	76
B.2	The most important attributes of a forced alignment process . .	78
B.3	The most important attributes for pitch and prosodic annotation	80
D.1	Alphabetical list of POS tags	95
D.2	International Phonetic Alphabet	97
D.3	Default DTW distance function for labels	100
D.4	Recognition rates for the <i>rob200</i> corpus	100
D.5	Recognition rates for the <i>kate</i> corpus	101

List of Tables

D.6	Comparison of the recognition rates of the <i>rob200</i> corpus for 25 % and 75 % test sets	101
D.7	Confusion matrix and recognition rates for stress and phrase break estimation for the <i>rob200</i> corpus and an ANN with a 25 % test set	102
D.8	Confusion matrix and recognition rates for stress and phrase break estimation for the <i>rob200</i> corpus and an ANN with a 75 % test set	103
D.9	Confusion matrix and recognition rates for stress and phrase break estimation for the <i>kate</i> corpus with an ANN	104

List of Figures

1.1	Midsagittal view of the vocal tract [VHH98] and sections of the human larynx [Fuj05]	3
1.2	EKG device and electrical field	4
1.3	Glottis, vocal fold position, air flow and EKG signal	5
1.4	Glottal width, EKG and audio signal for a male and a female voice [BLM83]	6
1.5	Fujisaki model	6
1.6	The different degrees of freedom for the movement of the thyroid cartilage	7
2.1	Phreemphasis filter amplitude response	11
2.2	Quadratic and cubic spline function and derivative	13
2.3	Multiresolution pyramid for the spline-based wavelet	14
2.4	Normalization coefficients for quadratic and cubic spline wavelets and the resulting step response for a cubic wavelet filter	15
2.5	Typical Structure of an Evolutionary Algorithm	16
2.6	Crossover with two crossover points	17
2.7	Pareto Front	18
2.8	$(\mu + \lambda)$ Evolutionary Strategy	19
2.9	Three-layer fully connected ANN	20
2.10	$f_{log}(x)$ log sigmoid and $f_{tanh}(x)$ hyperbolic tangent sigmoid transfer function	21
2.11	Area that is accepted by a single-layer ANN	22
2.12	Area that is accepted by a two-layer ANN	22
2.13	Area that is accepted by a three-layer ANN	22
2.14	Simplified error during the training of an ANN	23
2.15	DTW example for a label sequence	24
3.1	System structure	27
3.2	Phoneme Aligner Structure	28
3.3	Time and frequency domain properties for plosives of an example utterance	32
3.4	Calculating the boundary between pause and burst for plosive segments	33
3.5	Merging external syllable information	35
3.6	Fujisaki parameter example	37

List of Figures

3.7	Features used to describe base frequency and power contours . .	40
3.8	Comparison of two EGG signals of two different speakers	42
3.9	FFT spectra of the EGG signals for two different speakers . . .	43
3.10	Pitchmark extraction from an EGG signal with and without low pass filtering	44
3.11	EGG signal with missing GCIs between vowels	46
3.12	EGG signal with missing GCIs before vowels	46
3.13	EGG signal with missing GCIs after vowels	46
3.14	Example distribution of the distances between EGG and time- domain markers and linear regression results for a female speaker	47
3.15	Utterance structure	48
3.16	Example utterance	50
4.1	Histogram and plot of the pause-burst boundary errors for plo- sives and affricates	52
4.2	Histogram and plot of the segmentation boundary errors for an unshifted reference	53
4.3	Histogram and plot of the segmentation boundary errors for a shifted reference	53
4.4	Histogram and plot of the positive and negative errors for the aligner of Guntram Strecha for labels shifted by 4 ms	54
4.5	Histogram and plot of the positive and negative errors for the aligner of Karlheinz Stöber for labels shifted by 8 ms	54
4.6	Histogram and plot of segmentation boundary errors for shifted and scaled references	56
4.7	Confusion matrix and recognition rates for Festival phrase break estimation	58
4.8	Confusion matrix and recognition rates for stress and phrase break estimation for the <i>rob200</i> corpus	60
4.9	Confusion matrix and recognition rates for stress and phrase break estimation for the <i>kate</i> corpus	60
B.1	Relationship between the most used annotators. For a given an- notator, one or any of the preceding annotators are required for proper function.	81
C.1	Wavesurfer pitchmark plugin	91
C.2	Pitchmark plugin context menu and status line	92
C.3	Pitchmark plugin properties page	93
C.4	Festival Utterance Viewer example	94

Listings

2.1	Mallat algorithm	15
3.1	Festival wrapper scheme file to synthesize input from stdin and to store it in Festival utterances	30
3.2	Speech signal polarization determination	41
3.3	Determination of GCIs from an EGG signal	43
A.1	Festival utterance example	66
A.2	DTD for the XML utterance format	66
A.3	XML utterance example	67
A.4	Mbrola label file example	68
A.5	PhonDat label file example	68
A.6	XWaves label file example	69
A.7	Wavesurfer label file example	69
A.8	.Seg label file example	70
A.9	PAC Fujisaki parameter file example	71
A.10	PM pitchmark file example	72
A.11	Comparison of a multicolumn pitchmark file including confidence scores and a threshold with a file using negative values for un- voiced pitchmarks	72
A.12	Entropic fundamental frequency contour file	73
A.13	TC-STAR phonetic annotation file	73
A.14	TC-STAR prosodic annotation file	73
B.1	Commands to detect and correct common text problems	76
B.2	Example script for forced phoneme alignment	82
B.3	Helper functions to aid in bulk processing of large corpora	83
B.4	Two example phone sets for Dutch which can be converted be- tween each other	84
B.5	Example implementation of the <code>testExecutable</code> method	86
C.1	Patch to fix gcida WAVE file problems	89

Acronyms

ANN	Artificial Neural Network
ASCII	American Standard Code for Information Interchange
CSTR	Centre for Speech Technology Research
CART	Classification and Regression Tree
DRESS	Dresdener Sprachsynthese-System
CR	Carriage Return (0x0D)
DP	Dynamic Programming
DTD	XML Document Type Definition
DTW	Dynamic Time Warping
GA	Genetic Algorithm
GCI	Glottal Closure Instant
GP	Genetic Programming
GPC	Grapheme Phoneme Conversion
EA	Evolutionary Algorithm
EGG	Electroglottography
EO	Evolutionary Optimization
EP	Evolutionary Programming
ES	Evolution Strategies
FFT	Fast Fourier Transform
HMM	Hidden Markov Model
IPA	International Phonetic Alphabet
JavaNNS	Java Neural Network Simulator
LF	Line Feed (0x0A)
LPC	Linear Prediction Coding
MFC	Mel Frequency Cepstrum
MFN	Multilayer Feedforward Neural Network
MRPA	Machine Readable Phonetic Alphabet
POS	Part of Speech

PSOLA	Pitch-Synchronous Overlap and Add
RMS	Root Mean Square
SAMPA	Speech Assessment Methods Phonetic Alphabet
SNNS	Stuttgart Neural Network Simulator
SPEA	Strength Pareto Evolutionary Algorithm
SNR	Signal to Noise Ratio
STFT	Short Term Fourier Transform
TC-STAR	Technology and Corpora for Speech to Speech Translation
TTS	Text to Speech
UCS	Universal Character Set
UTF-8	UCS Transformation Format, 8-bit form
XML	Extensible Markup Language

1. Introduction

“There was once upon a time an old goat who had seven little kids, and loved them . . .” is the beginning of another well known fairy tale by the Grimm Brothers [Gri89]. The capability of the one reading to fully captivate the attention of little kids decides over the success of the story.

Now, automatic speech synthesis has still a long way to go until a computer has the necessary proficiency to generate speech able to achieve these effects. Nevertheless, automatic speech processing is improving steadily and gaining increasing acceptance in consumer devices and communication systems.

With the advancing development of computer and storage solutions, the use of very large databases in speech synthesis and recognition becomes feasible. Alternative approaches to natural sounding speech synthesis like corpus-based methods need large speech corpora that have to be extensively annotated. Because of the sheer amount, this work can not be done manually and has to be supported by automatic means. Although computer based annotation may not fulfill the accuracy requirements for such a task, it can significantly reduce the processing time and amount of correction necessary.

The automatic speech annotation system presented in this thesis tries to simplify and automate most of the tasks required for the initial processing of a corpus. It implements functionality like automatic wave file and phone set conversions, data storage and multi-level annotation management while providing a modular framework for phoneme-level, prosodic and linguistic annotation algorithms.

The remaining sections of this chapter give an introduction into the fundamentals of speech production and prosody modeling with the Fujisaki model. In chapter 2, necessary mathematical and algorithmic methods for signal processing and feature optimization are provided. The developed annotation framework together with the implemented algorithms is presented in chapter 3 and the annotation results are evaluated in chapter 4. Finally, chapter 5 summarizes the achieved results and outlines some prospects for future research.

The appendix contains extensive documentation of the developed framework. It provides information about the file formats used for data storage and communication with the various external programs, many examples that illustrate the use of the framework for the fully automatic annotation of a newly recorded corpus and documentation of additional programs developed to visualize annotation results.

1.1. Speech Production

Speech is used to convey information between humans, both linguistic and non-linguistic. The linguistic part includes semantic information represented by the segmental structure of vowels and consonants as well as suprasegmental features like pitch and duration. Non-linguistic features give information about the attitudinal and emotional state, the social and geographic background, the sex and the age of a speaker [Sai92].

Speech can be divided in the following units [Mar97]:

Phonetic features form the parameter set that can describe differences between phonemes of a language. They may be defined in terms of articulatory, acoustic and perceptual properties.

Segments are short stretches of speech in the range of 30 to 300 ms that have relatively constant features.

Phrases are sequences of words that are separated by prosodic means.

Utterances describe portions of speech that are confined by silence and have no pauses in-between, often corresponding to written sentences.

Speaking turns are the successive parts in a conversation that are spoken by a single person.

A complex collaboration of several speech organs is necessary to produce speech (figure 1.1):

- The respiratory system consisting of the lung and the trachea supplies the airflow that is later on converted to sound energy.
- Depending on the voice quality, the vocal folds of the larynx are open, closed or vibrating, modulating the air flow.
- The source signal created by the glottis is shaped by the vocal tract to produce distinctive sounds. Vowels and some consonants are formed by the resonance frequencies of the vocal tract whereas consonants like fricatives, stops and affricates are caused by obstruction of the air flow [LC97].

The process of transforming the air flow into audible sounds at the larynx is called *phonation*. Depending on the movements of the vocal folds, the following different types are distinguished [Mar97]:

Nil phonation is caused by stationary vocal folds. Glottal stops are produced if the glottis is opened after it was blocking to airstream completely, whereas a narrow opening or high flow speeds cause turbulent airflow that can be heard as glottal fricatives or breath.

Whisper intonation is created by non-vibrating vocal folds that have very low tension and are only partially opened.

Modal voiced phonation is regarded as the neutral mode of phonation. Tension of the vocal folds is at an average, and the glottis alternates between fully opened and closed states.

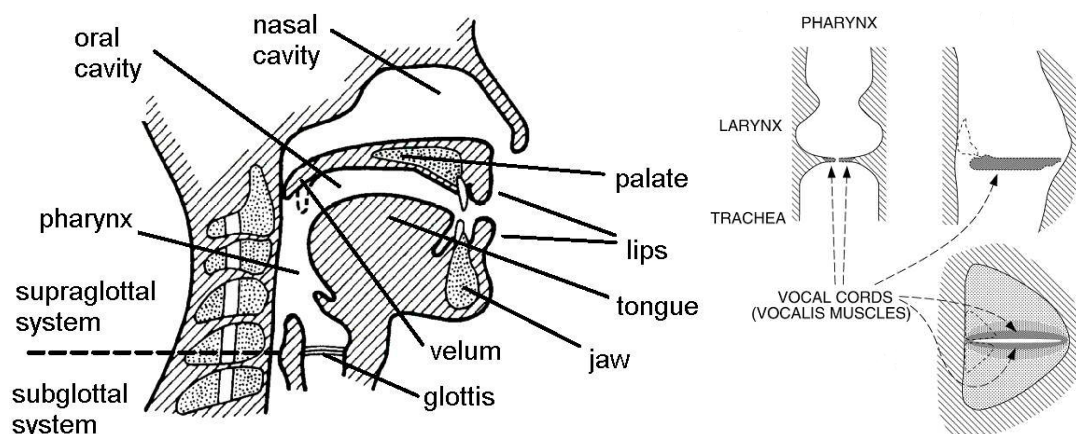


Figure 1.1: Midsagittal view of the vocal tract [VHH98] and sections of the human larynx [Fuj05]

Creak phonation or vocal fry is characterized by stiffened vocal folds with low tension that vibrate at low frequencies.

Breathy voice is a mixed phonation type comparable to voiced whisper. The vocal folds close only incompletely, allowing stationary airflow that causes audible friction noise.

Harsh voice is caused by high tension of all the muscles in the entire vocal tract, creating an irregular cycle duration and amplitude.

Falsetto phonation is due to relatively thin stretched vocal folds, resulting in a higher generated frequency.

The signal generated by the vocal folds is the basis for speech synthesis based on algorithms like Pitch-Synchronous Overlap and Add (PSOLA) that require marked pitch periods of the input units. Methods like Linear Prediction Coding (LPC) can be used to recreate the source signal from a recorded audio signal, making it possible to determine pitch markers with autocorrelation-based techniques.

Another possible way to obtain the source signal would be the measurement of the glottis opening as it is proportional to the acoustic pressure of the air flow. Although high speed and stroboscope camera recordings of the vocal cords have been made as early as in the 1940s [HF38], it still remains a rather invasive and uncomfortable method that can't be used for large-scale recordings.

To overcome these problems, various methods have been proposed that measure the glottal opening in an indirect way from the outside of the neck:

- Electromagnetic sensors are based on the different permittivity of air and body tissue [Pel04, BHN02]. During voiced segments of speech, the compound relative permittivity oscillates at the same frequency as the glottis which can be observed by capacitive sensors. The needed equipment however is not readily available and expensive.

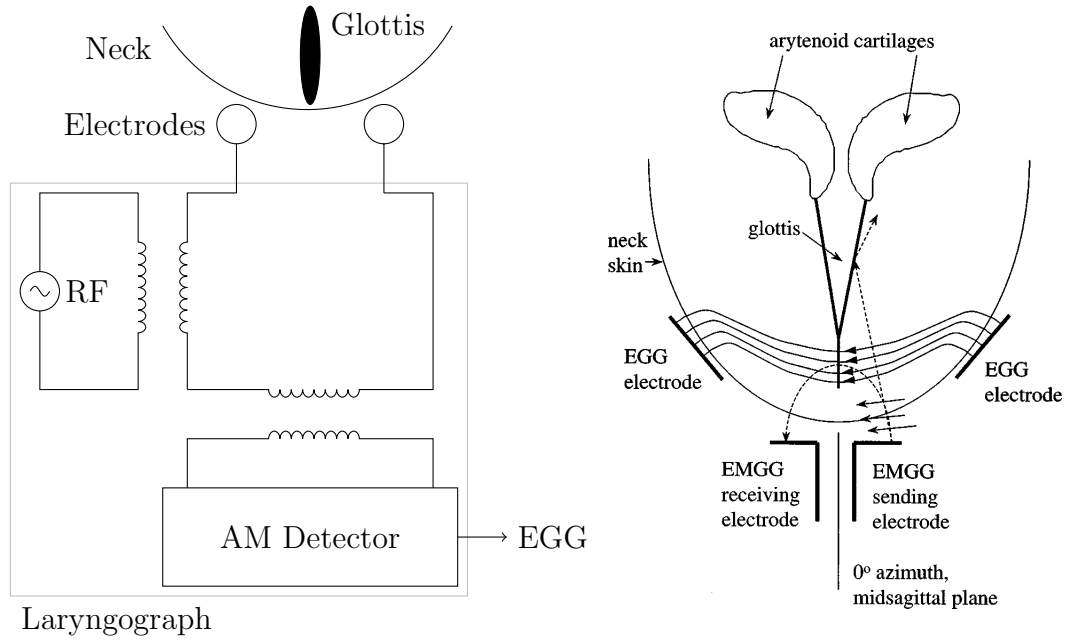


Figure 1.2: EGG device (after [CK85]) and electrical field created by the electrodes [TSB⁺00]

- Electroglottography (EGG) uses the electrical impedance to measure the opening of the glottis [KLKP00, Rot90, MPK01].

1.1.1. Electroglottograph

An Electroglottograph (figure 1.2) is connected to two electrodes that are attached to both sides of the thyroid cartilage with an elastic band. Sometimes a third electrode is used to gain an impedance reference. This attachment does not affect the speakers ability to talk, breathe and swallow therefore making it possible to wear for a prolonged period of time. A radio frequency sinus generator in the range of 300 kHz to 5MHz supplies a current of less than 10 mA to the electrodes, capacitively bypassing the skin layer [Mar97]. Additional conductive paste may also be used.

The RF signal is modulated by the varying impedance of the vibrating vocal folds. Because admittance is proportional to the conducting area, the electrical resistance of the larynx changes with the glottis size and maximum resistance can be observed with a totally open glottis. The RF frequency part is removed in the AM detector and the digitized demodulated result stored in a computer.

Although this methods can give excellent results, [KC86] warns that “*the reader is cautioned that we do not consider the EGG signal as a cure-all for solving pitch estimation problems.*” Various error sources complicate the work with an Electroglottograph [Eng03]:

- The plate electrodes have to be positioned correctly in relation to the

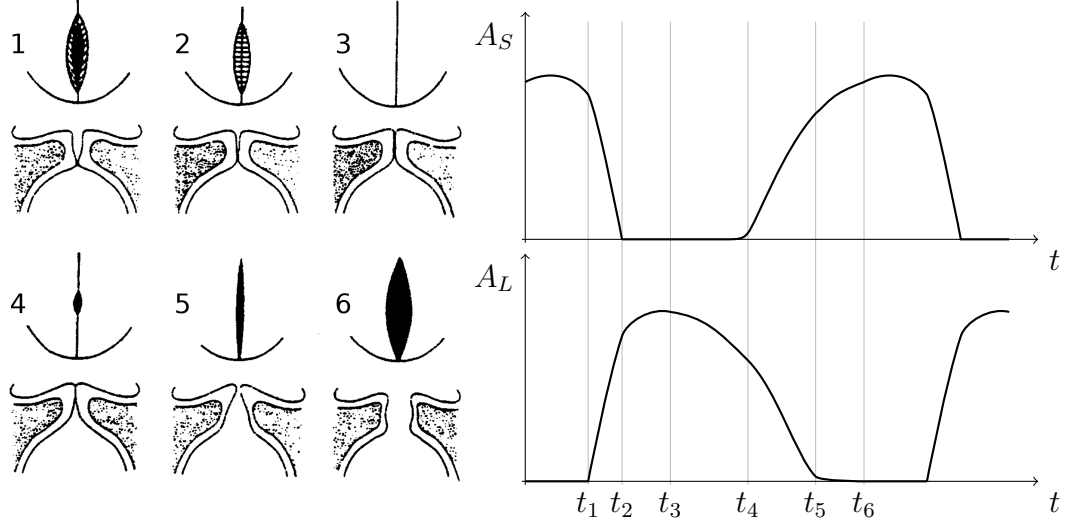


Figure 1.3: Glottis, vocal fold position, air flow A_S and EGG signal A_L [Hes83]: (1) glottis closes (2) glottis is closed completely (3) glottis is closed maximally (4) glottis starts to open (5) glottis is completely open (6) glottis is maximally open (after [Eng03])

larynx independently of the anatomy of the speaker.

- Hairiness, draining of the conductive paste, individual anatomy and movement of the larynx during speaking can cause limited or lost contact between the electrodes resulting in a signal that has a low Signal to Noise Ratio (SNR) or is even incomplete.
- EGG signal quality and shape depend on individual and sex.
- Speakers change the position of their larynx while speaking, causing a variation in the measured impedance and therefore a DC shift of the EGG signal that needs to be removed.
- The recorded signal is not identical with the wanted source signal and has to be post-processed to retrieve pitch markers.
- The EGG signal is distorted by random noise caused by the RF part of the AM detector and voice-synchronous noise because of tissue vibration [Rot92].

1.1.2. Source Signal

Figure 1.3 shows the glottis, the vocal folds, the resulting air stream and a simplified EGG signal that is proportional to the admittance of the glottis.

The Glottal Closure Instant (GCI) is defined here as the moment when the vocal folds start to close and is characterized by a steep increase in the EGG signal that nearly coincides with the maximum in the differentiated EGG. The instant of glottal opening occurs very close to the minimum of the differentiated EGG [KC86]. The distance between a GCI and the maximum is about $0.2 \text{ ms} \pm$

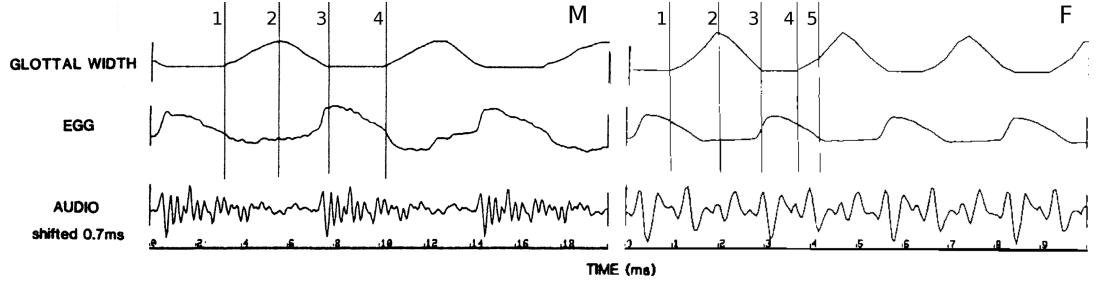


Figure 1.4: Glottal width, EGG and audio signal for a male (M) and a female voice (F): (1) glottis opens (2) glottis is opened maximally (3) glottis is closed (4) glottis opens again (5) glottis is completely open [BLM83]

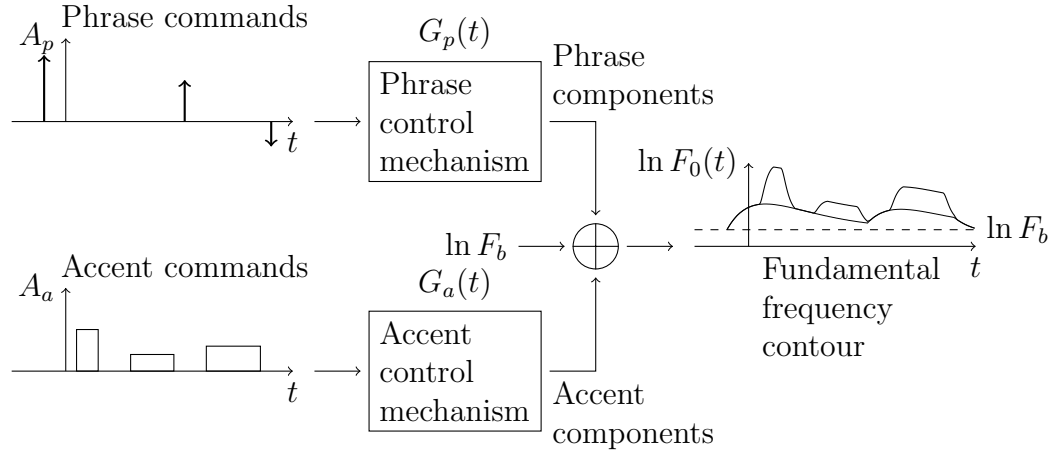


Figure 1.5: Fujisaki model (after [Fuj05])

0.1 ms, whereas the difference of the glottal opening and the minimum is slightly more variable with about $0.3 \text{ ms} \pm 0.2 \text{ ms}$. Similar values are mentioned in [Mic99] with differences between GCIs and maximums of 0 ms to 0.3 ms (figure 1.4).

1.2. Fujisaki Model

The determination of pitchmark positions allows the calculation of base frequency contours. The well-known Fujisaki model for parametrizing f_0 contours was developed in 1984 by Fujisaki and Hirose (figure 1.5). In contrast to other approaches to the description of base frequency contours, it tries to incorporate physical as well as physiological explanations.

The Fujisaki model is based on the physiological structure of the larynx [Fuj05]. Two different mechanisms for the movement of the thyroid cartilage correspond to the two available degrees of freedom in the generation of base frequency contours (figure 1.6): global phrase commands (impulse functions)

1. Introduction

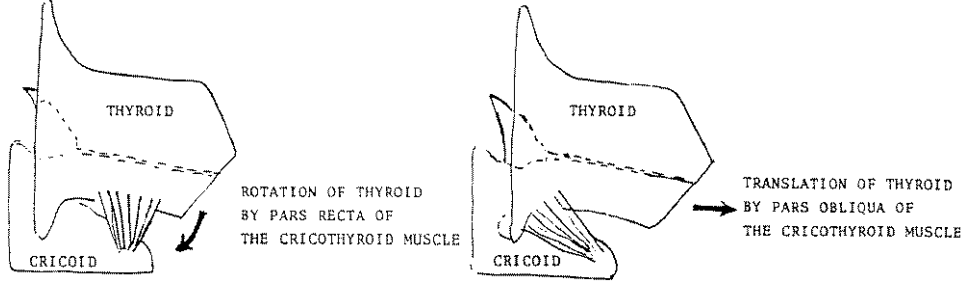


Figure 1.6: The different degrees of freedom for the movement of the thyroid cartilage (after [Fuj05])

and local accent commands (step functions). Together with the asymptotic minimum of the base frequency F_b the logarithmic base frequency contour is calculated (equation 1.1).

The model was originally developed for Japanese and has since been adapted to a wide range of languages, for example English [FO95], Swedish [FLM93] and German [Mix97].

$$\ln F_0(t) = \ln F_b + \sum_{i=1}^I A_{pi} G_p(t - T_{0i}) + \sum_{j=1}^J A_{aj} [G_a(t - T_{1j}) - G_a(t - T_{2j})] \quad (1.1)$$

$$G_p(t) = \begin{cases} \alpha^2 t \exp(-\alpha t), & t \geq 0 \\ 0, & t < 0 \end{cases} \quad (1.2)$$

$$G_a(t) = \begin{cases} \min[1 - (1 + \beta t) \exp(-\beta t), \gamma], & t \geq 0 \\ 0, & t < 0 \end{cases} \quad (1.3)$$

Phrase component $G_p(t)$ (equation 1.2) represents the basic contour over a complete phrase. An impulse signal that marks the phrase start at T_{0i} creates the impulse response $G_p(t)$. The time constant α determines the decay of the contour and is set to about 1.0/s. It is regarded as constant for the whole utterance. The magnitude of the impulse response A_{pi} defines the intensity of the phrase, i.e. the change in the logarithm of F_0 .

Accent component $G_a(t)$ (equation 1.3) describes local accents for syllables and words. A stepwise function with an onset time of T_{1j} and an end time of T_{2j} causes a step response from the accent control unit. The constant β determines the shape of the signal and is about 20.0/s. It is also considered constant for the utterance. The upper bound γ (normally $\gamma = 0.9$) limits the time until the ceiling value is reached, the constant A_{aj} determines the magnitude of the accent commands.

1. Introduction

F_b contains the baseline value of the fundamental frequency, i.e. the asymptotic value in the absence of any accent or phrase commands. The parameters A_p , T_0 and α for the phrase control unit and A_a , T_1 , T_2 and β for the accent control unit are called *Fujisaki parameters*.

2. Signal Processing and Optimization Methods

The following sections provide some mathematical and algorithmic background to the used signal processing and optimization methods. If you are familiar with the here discussed algorithms, you may want to skip this chapter.

2.1. Signal Processing

2.1.1. Windowing

Windowing is the process of cutting time frames of a certain length out of a longer signal. To avoid artifacts while doing short-time signal processing on these frames, window functions are employed to smooth the signal both in the time and spectral domain. They are used in the calculation of a variety of parameters, e. g. in the Fast Fourier Transform (FFT) or in the determination of power contours. The most common windows besides the rectangular window are the *Hann*, *Hamming*, *Bartlett* and *Blackman* windows (table 2.1).

Discrete window functions for the window length N can be calculated by substituting $\tau = \frac{1}{2}(N - 1)$ and $t = k - \frac{1}{2}N$ in the equations. Use $\tau = \frac{1}{2}N$ to get cyclic instead of symmetric windows as required by short-time analysis.

2.1.2. Preemphasis

To compensate the frequency spectrum of the human voice that has much of its energy in the higher bands, a preemphasis filter is used. It is mainly employed in the calculation of signal characteristics like power contours to get more meaningful results. The most common realization is

$$y(k) = y(k + 1) - py(k) \quad (2.1)$$

with its transfer function

$$G(z) = 1 - pz^{-1} \quad (2.2)$$

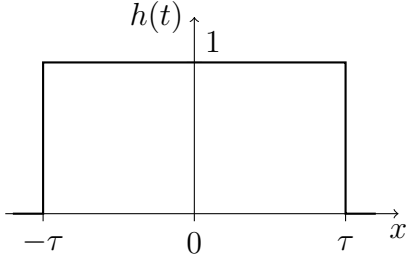
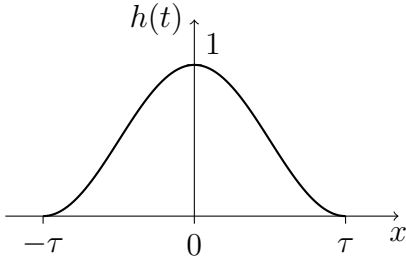
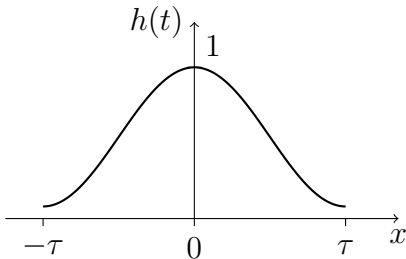
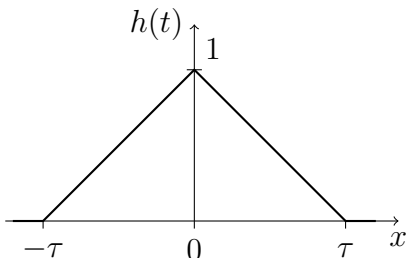
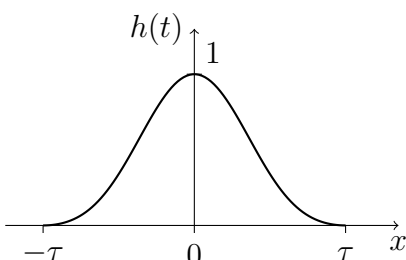
Name	Function
Rectangular window	$h(t) = \begin{cases} 1, & t < \tau \\ 0, & t \geq \tau \end{cases}$ 
Hann window	$h(t) = \begin{cases} \frac{1}{2}(1 + \cos(\pi \frac{t}{\tau})), & t < \tau \\ 0, & t \geq \tau \end{cases}$ 
Hamming window	$h(t) = \begin{cases} 0.54 + 0.46 \cos(\pi \frac{t}{\tau}), & t < \tau \\ 0, & t \geq \tau \end{cases}$ 
Bartlett window	$h(t) = \begin{cases} 1 - \frac{t}{\tau} , & t < \tau \\ 0, & t \geq \tau \end{cases}$ 
Blackman window	$h(t) = \begin{cases} 0.42 + 0.5 \cos(\pi \frac{t}{\tau}) + 0.08 \cos(2\pi \frac{t}{\tau}), & t < \tau \\ 0, & t \geq \tau \end{cases}$ 

Table 2.1: Window functions

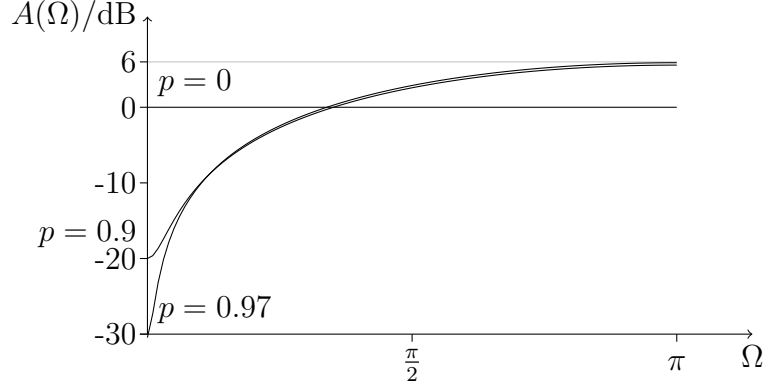


Figure 2.1: Phreemphasis filter amplitude response for different values of p

This results in an amplitude response of

$$A(\Omega) = |G(e^{j\Omega})| = \sqrt{1 + p^2 - 2p \cos(\Omega)} \quad (2.3)$$

$$A(0) = 1 - p \quad (2.4)$$

$$A(\pi) = 1 + p \quad (2.5)$$

Usual values for the coefficient p are 0.9 to 1.0 (figure 2.1).

2.1.3. Linear Regression

Regression analysis allows to calculate the coefficients for a specific function so that a certain number of data points is modeled optimally. More specifically, linear regression analysis tries to fit the equation

$$y(x) = mx + n \quad (2.6)$$

to N data points (x_i, y_i) so that the sum of the squares R of the distances between y_i and $y(x_i)$ is minimized:

$$R = \sum_{i=1}^N (y_i - y(x_i))^2 \quad (2.7)$$

With the correlations

$$S_{xx} = \sum_{i=1}^N x_i^2 - \frac{1}{N} \left(\sum_{i=1}^N x_i \right)^2 \quad S_{xy} = \sum_{i=1}^N x_i y_i - \frac{1}{N} \sum_{i=1}^N x_i \sum_{i=1}^N y_i \quad (2.8)$$

the coefficients m and n can be calculated by

$$m = \frac{S_{xx}}{S_{xy}} \quad n = \frac{1}{N} \sum_{i=1}^N y_i - \frac{m}{N} \sum_{i=1}^N x_i \quad (2.9)$$

The variation explained by the regression equation is expressed by the coefficient of determination r^2

$$r^2 = \frac{S_{xy}^2}{(S_{xx}S_{yy})^2} \quad (2.10)$$

The so determined coefficients can be used to analyze the general trend of a given sequence of data points or to evaluate assumptions about the data distribution.

2.1.4. Wavelet Transform

Wavelet analysis overcomes some of the limitations of the Short Term Fourier Transform (STFT) and Fourier Transform by allowing to choose different basis functions and flexible window lengths that depend on the analyzed frequency band.

The *continuous wavelet transform* is defined by the coefficients $\gamma(s, \tau)$ [Val04]

$$\gamma(s, \tau) = \int f(t) \psi_{s,\tau}^*(t) dt \quad (2.11)$$

for the analyzed function $f(t)$, the basis functions $\psi_{s,\tau}^*(t)$, the scale s and the translation τ . Each basis function can be derived from a *mother wavelet* $\psi(t)$

$$\psi_{s,\tau}(t) = \frac{1}{\sqrt{s}} \psi\left(\frac{t - \tau}{s}\right) \quad (2.12)$$

These so called *wavelets* need to fulfill the *admissibility condition*

$$\int \frac{|\Psi(\omega)|^2}{|\omega|} d\omega < +\infty \quad (2.13)$$

with $\Psi(\omega)$ being the Fourier transform of $\psi(t)$ which means that $\Psi(\omega)$ vanishes at the zero frequency

$$|\Psi(\omega)|^2 \big|_{\omega=0} = 0 \quad (2.14)$$

and wavelets have to be a wave

$$\int \psi(t) dt = 0 \quad (2.15)$$

A discrete wavelet can be defined by

$$\psi_{j,n}(t) = \frac{1}{\sqrt{s_0^j}} \psi\left(\frac{t - n\tau_0 s_0^j}{s_0^j}\right) \quad (2.16)$$

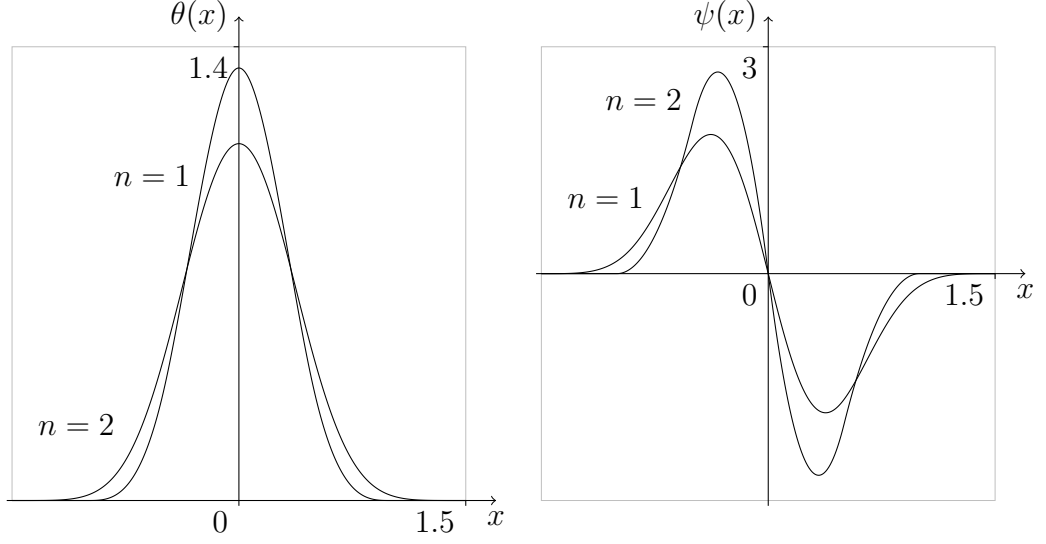


Figure 2.2: Quadratic ($n = 1$) and cubic ($n = 2$) spline function $\theta(x)$ and derivative wavelet function $\psi(x)$

with the integers j and n , the dilation step s_0 and the translation factor τ . We further assume $s_0 = 2$ and $\tau_0 = 1$ which gives dyadic sampling of the time and frequency axis.

A family of spline-based wavelets that can be used for edge detection is presented in [MZ92] (figure 2.2). The Fourier transform $\Theta(\omega)$ of the spline primitive $\theta(x)$ is defined by

$$\Theta(\omega) = \left(\frac{\sin(\omega/4)}{\omega/4} \right)^{2n+2} \quad (2.17)$$

with the derivative of the primitive used as wavelet that is given by its Fourier transform $\Psi(\omega)$

$$\Psi(\omega) = i\omega \left(\frac{\sin(\omega/4)}{\omega/4} \right)^{2n+2} \quad (2.18)$$

and the Fourier transform $\Phi(\omega)$ of the *scaling function* $\phi(x)$

$$\Phi(\omega) = i\omega \left(\frac{\sin(\omega/2)}{\omega/2} \right)^{2n+1} \quad (2.19)$$

The Mallat algorithm [Hof98, Dau91] can be used to calculate the desired wavelet coefficients with a multiresolution pyramid that is specified by the re-

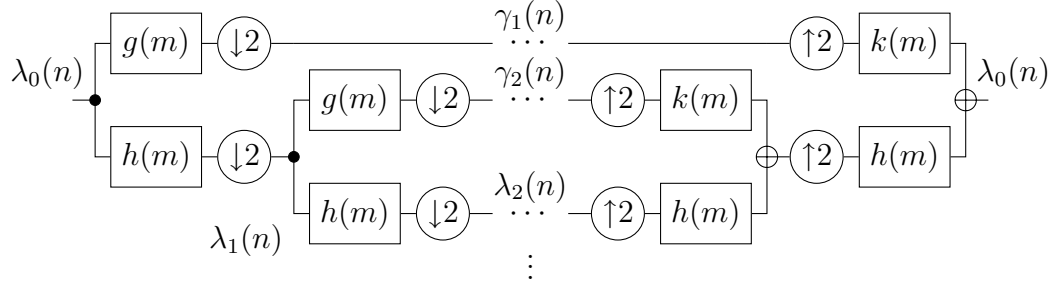


Figure 2.3: Multiresolution pyramid for the spline-based wavelet

cursive equations for the coefficient calculation

$$\lambda_{j+1}(n) = \sum_m h(m - 2n) \lambda_j(m) \quad (2.20)$$

$$\gamma_{j+1}(n) = \sum_m g(m - 2n) \lambda_j(m) \quad (2.21)$$

and for the signal reconstruction by

$$\lambda_{j-1}(n) = \sum_m k(m - 2n) \gamma_j(m) + \sum_m h(m - 2n) \lambda_j(m) \quad (2.22)$$

The calculated coefficients $\gamma_j(n)$ are the high pass part of a multiresolution pyramid (figure 2.3) and correspond to the wavelet coefficients of scale 2^j , the coefficients $\lambda_j(n)$ are the low pass result. The signal $f(n)$ can be regarded as the result of a (imaginary) initial filter step and equals $\lambda_0(n)$. The *scaling filter* $h(n)$, the *wavelet filter* $g(n)$ and the *reconstruction filter* $k(n)$ coefficients are specified by their Fourier transforms [MZ92]

$$H(\omega) = e^{i\omega/2} (\cos(\omega/2))^{2n+1} \quad (2.23)$$

$$G(\omega) = 4ie^{i\omega/2} \sin(\omega/2) \quad (2.24)$$

$$K(\omega) = \frac{1 - |H(\omega)|^2}{G(\omega)} \quad (2.25)$$

The calculation of the filter coefficients is shown in appendix D.3. Variable filters $H_j(\omega)$, $G_j(\omega)$ and $K_j(\omega)$ for each calculation step that have $2^j - 1$ zeros inserted between the filter coefficients may be used instead of the downsampling and upsampling steps in the multiresolution pyramid, the resulting algorithm for wavelet transform and reconstruction can be seen in listing 2.1.

The normalization coefficients p_j are required to achieve the same amplitude independent on the scale for a step edge (figure 2.4).

```

for (j = 0; j <= J; ++j) {
    Gfilter = InsertZeros (Gmother, (1 << j) - 1)
    Hfilter = InsertZeros (Hmother, (1 << j) - 1)
    Gamma[j] = 1 / p[j] * Convolve (Lambda, Gfilter)
    Lambda = Convolve (Lambda, Hfilter)
}

// Process wavelet coefficients ...

for (j = J; j >= 0; --j) {
    Kfilter = InsertZeros (Kmother, (1 << j) - 1)
    Hfilter = InsertZeros (Hmother, (1 << j) - 1)
    Lambda = Convolve (Lambda, Hfilter) +
              p[j] * Convolve (Gamma[j], Kfilter)
}

```

Listing 2.1: Mallat algorithm (after [MZ92])

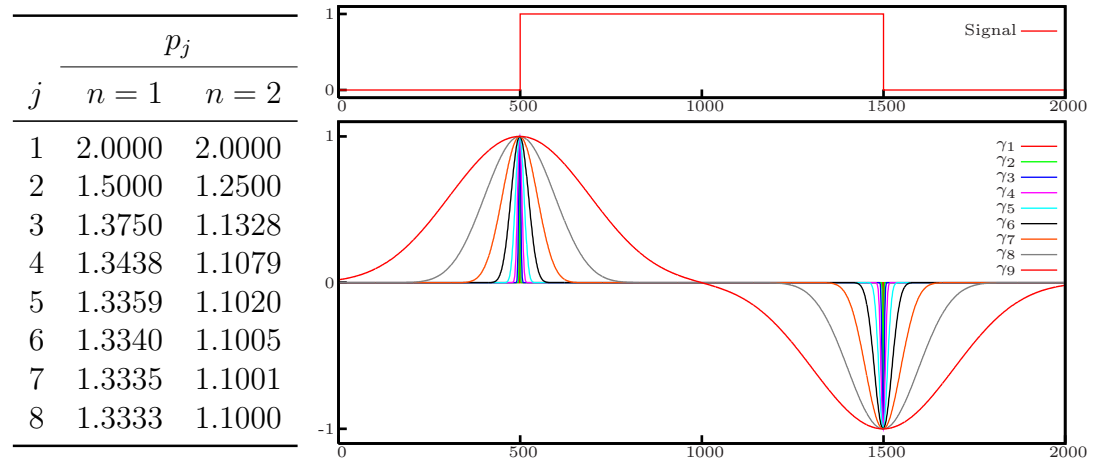


Figure 2.4: Normalization coefficients p_j for the quadratic ($n = 1$) and cubic ($n = 2$) spline wavelets and the resulting step response for a cubic wavelet filter

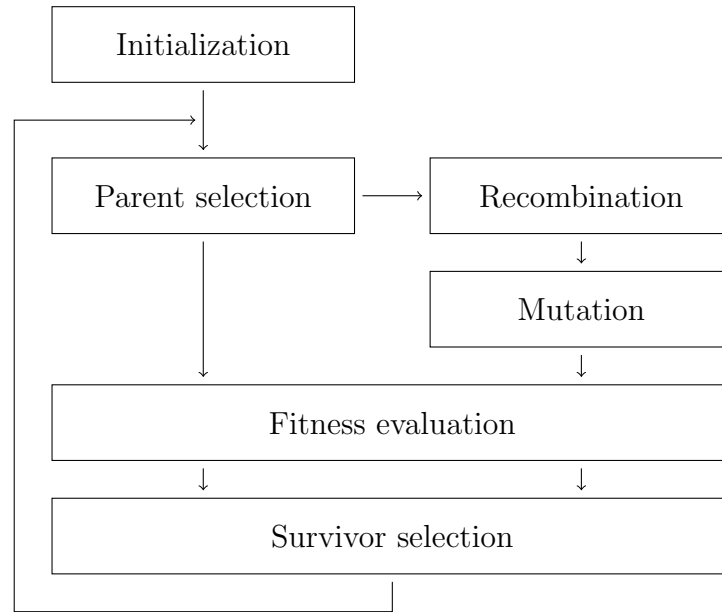


Figure 2.5: Typical Structure of an EA

2.2. Evolutionary Optimization

Evolutionary Algorithms (EAs) are ways to solve a computational problem by using methods that are similar to known mechanisms of evolution. They are typically employed if a direct connection between the parameters to optimize and the results of the optimization problem can not easily be made. For an extensive and humorous introduction, see [HB01]. Several different algorithms are available, among them

- Genetic Algorithms (GAs),
- Evolutionary Programming (EP),
- Evolution Strategies (ES),
- Genetic Programming (GP).

The typical structure of such an algorithm can be seen in figure 2.5. An EA operates on *populations* of *individuals* represented by a *genome* that encodes parameters for the problem to optimize. Each genome has a certain *fitness* and can be *mutated* or *recombined* with other genomes to create offsprings.

Two ways exist to represent the possible solutions of an EA: GAs use a separate encoding in chromosome-like structures to represent the parameters, whereas ESs do not demand a special description. For such genomes consisting of boolean or floating point *genes* of a certain range, the basic operations described in the next sections need to be defined to be able to solve Evolutionary Optimization (EO) problems [Kos01].

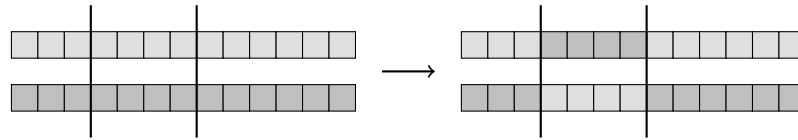


Figure 2.6: Crossover with two crossover points

2.2.1. Initialization

The first population is initialized with individuals chosen randomly from the available parameter range or set to a specific prototype.

Fixed All genes are initialized to fixed values.

Random The gene value is selected randomly from the valid range for the given parameter.

2.2.2. Parent Selection

From all available individuals, a subpopulation is selected for offspring production.

Fitness The best individuals according to their fitness score are selected.

Tournament The selection is obtained in rounds. Two individuals at a time are chosen from the population and the one with the higher fitness is promoted to the next round. The winner of the last round is selected.

2.2.3. Recombination

The parents' genomes are recombined to generate the offspring population.

Average The parents' genes are averaged sequentially with each other and the resulting value is used for the offspring. When applied to boolean values, the result will be random if the original values were different.

Crossover One or more random points on the genome are picked and the resulting portions exchanged between the parents (figure 2.6).

2.2.4. Mutation

The individuals of the offspring population are mutated, i. e. their genomes are altered.

Boolean The new boolean values is determined by chance.

Random A random offset within a certain range from a uniform distribution is added to the previous value.

Gaussian A Gaussian distributed offset is added to the previous value.

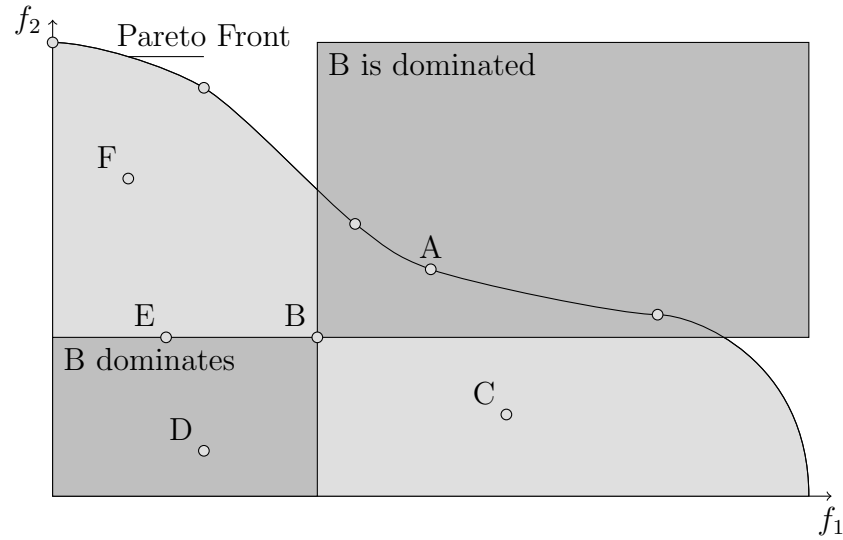


Figure 2.7: Pareto Front

2.2.5. Fitness Evaluation

All available individuals are rated regarding the fitness for the particular problem to solve. While it is easy to rate the outcome of a problem that contains only one objective, most real-world problems contain more than one goal. Optimizing such a problem causes an infinite number of optimal solutions, generally known as Pareto-optimal solutions. This set of solutions is also called the Pareto Front.

The concept of Pareto Dominance describes relationships between different solutions. One solution dominates another one, if exceeding it at least concerning a single parameter while no other parameter is worse. In figure 2.7, solutions in the dark gray rectangle (bottom left) are dominated by B . But B itself is dominated by the rectangle upper right. Solutions F or C are neither dominated by B nor dominate B , although they are not optimal, as well. Solutions like A , along the Pareto Front, are optimal. To approximate the Pareto Front, different algorithms like Strength Pareto Evolutionary Algorithm (SPEA) have been developed [Zit99].

2.2.6. Survivor Selection

Based on the fitness values, the survivors of both parent and offspring population are selected.

Kill worst The individuals with the worst scores are removed.

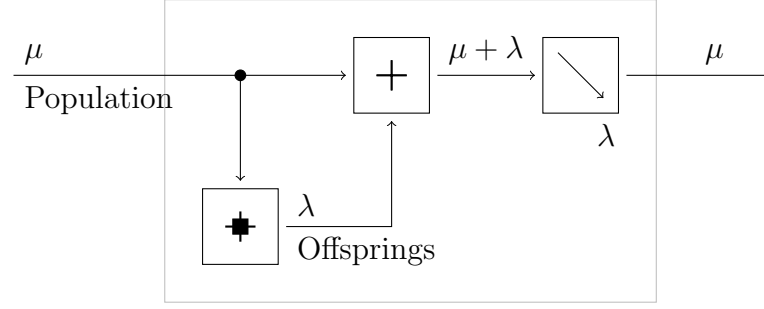


Figure 2.8: $(\mu + \lambda)$ Evolutionary Strategy (after [Kos01])

2.2.7. Strategies

Strategies define the exact sequence and parameters of then mentioned modules.

$(\mu + \lambda)$ All μ individuals from the parent population are used to create λ offsprings. Afterwards, the worst λ individuals from both parent and offspring population are removed (figure 2.8).

2.3. Artificial Neural Networks

Artificial Neural Networks (ANNs) are information processing dynamic systems that consist of a large number of simple units that influence each other [Zel94].

One important property of ANNs is their ability to learn unattended from training samples without the need of explicit programming. Additionally, the output quality of an ANN degrades gracefully with increased input noise.

Neural networks consist of:

- Units (neurons). they have the following constituents:
 - Activation state $a_j(t)$. It specifies the level of activation of the neuron j .
 - Bias (threshold) θ_j
 - Activation function f_{act} . Calculates the new activation value $a_j(t+1)$ from the old activation $a_j(t)$, the weighted output of the preceding units $net_j(t)$ and the bias of the neuron θ_j .
 - output function f_{out} . The output functions determine the output of the neurons from the activation $a_j(t)$.
- Connections (links) between the units. The weights w_{ij} for a connection from unit i to unit j and the output from the preceding neurons result in the net input for the succeeding neuron.
- Learning algorithm. Determines the way the ANN learns to produce the correct output for a given input by adjusting the weights of the links and the thresholds of the units.

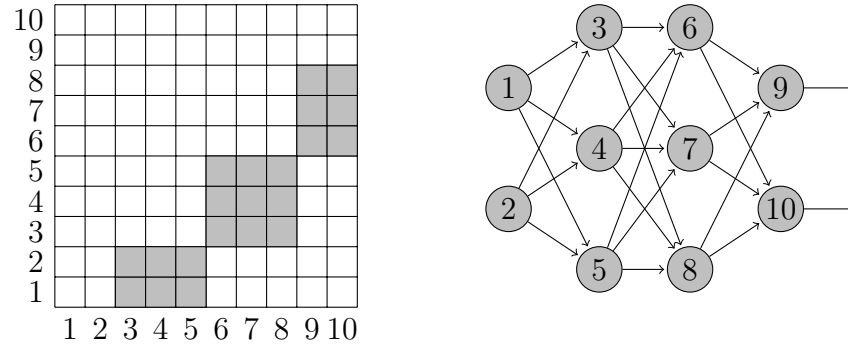


Figure 2.9: Example for a three-layer fully connected ANN and its connection or weight matrix

Depending on the task different structures for a ANN are possible. A commonly used type is the Multilayer Feedforward Neural Network (MFN) in which neurons are arranged in multiple layers. Units of different layers are connected by unidirectional links in the direction from the input to the output neurons, units within one layer are not connected to each other.

Connections that skip layers are called *shortcut connections*. A fully connected network has all the possible connections between adjacent layers (figure 2.9).

2.3.1. Transfer Functions

Often the activation function f_{act} is combined with the output function f_{out} to the transfer function f . Sometimes, this function is again called activation function.

Transfer functions can be divided into several groups:

- *Linear transfer functions* are only useful for single-layer networks. All networks of higher order can be reduced to such a network.
- The *step transfer function* is mainly used in binary perceptrons. The input value is compared with an internal threshold and switches the binary output.
- *Sigmoid transfer functions* are the most commonly used transfer functions. Two typical examples are the log sigmoid transfer function and the function $\tanh(x)$ (figure 2.10).

2.3.2. Multilayer Feedforward Neural Networks

The ability of the network to realize a given function is mainly determined by the number of layers of trainable links. The following figures are based on the use of a step transfer function, but also apply analogously to other transfer functions.

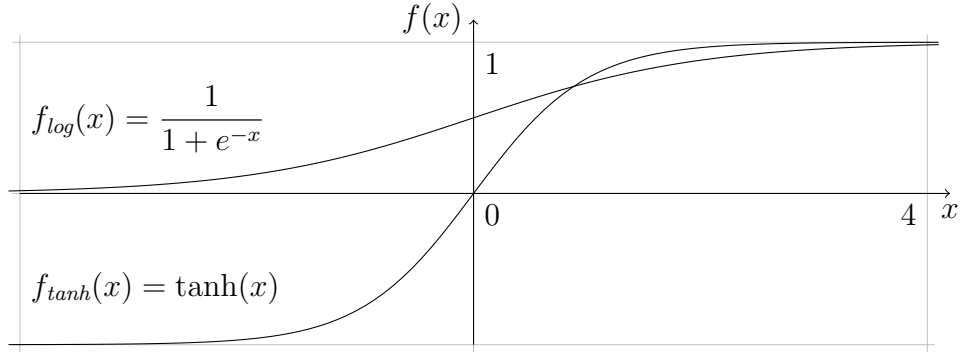


Figure 2.10: $f_{log}(x)$ log sigmoid and $f_{tanh}(x)$ hyperbolic tangent sigmoid transfer function

Figure 2.11 shows a single-layer network and a function that can be realized. This classic example requires linear separability of the request function with a hyperplane. Other functions like the XOR problem are not representable with this structure.

A two-layer neural network like the one in figure 2.12 can reproduce every logical combination of half-planes, i. e. convex polygon structures. The displayed function of an AND combination can be realized by a binary step transfer function at the output neuron that has a threshold slightly lower than the sum of the weights of the connections to this neuron.

The three-layer network (figure 2.13) can realize every possible functions through the logical combination of convex polygons. MFNs of higher order with more layers have no additional capabilities.

2.3.3. Training of a Neural Network

The training of ANNs is done in three steps and requires two or three disjunct groups of patterns:

1. The *training pattern set* serves for the supervised training of the network until convergence of the output error is reached. The most commonly used learning algorithm is *backpropagation*.
2. To select the best possible network state, the *validation pattern set* is used. These patterns are not included in the network training and determine when to stop the training to avoid *overfitting* (figure 2.14).
3. The result of the training is calculated with the help of the *test pattern set*.

Often test and validation sets are combined into only one set of patterns that fulfills both tasks, although this will result in a network that is not independent of its training process. The initial weights of the network are set to random values in the range of -1 to 1.

2. Signal Processing and Optimization Methods

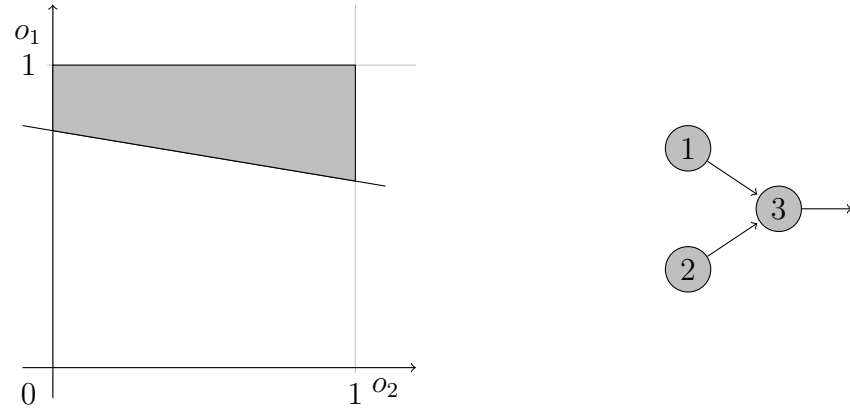


Figure 2.11: Area that is accepted by a single-layer ANN (after [Zel94])

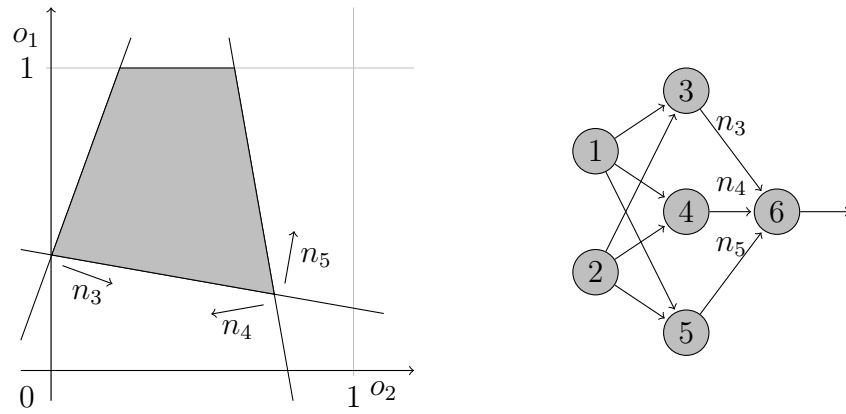


Figure 2.12: Area that is accepted by a two-layer ANN

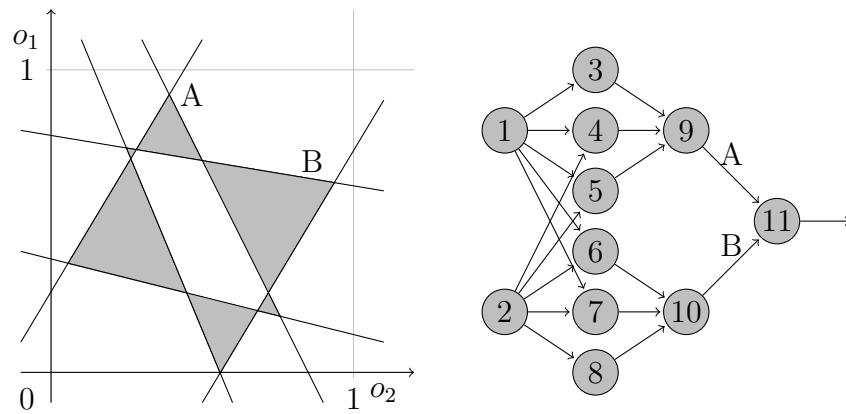


Figure 2.13: Area that is accepted by a three-layer ANN

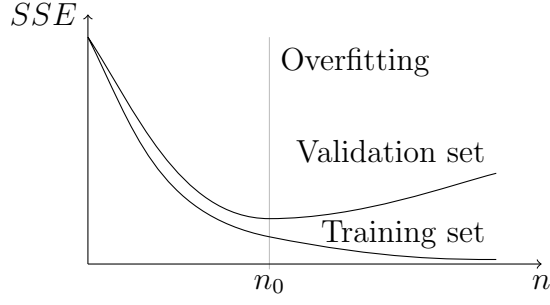


Figure 2.14: Simplified error during the training of an ANN

2.4. Dynamic Time Warping

A slightly modified Dynamic Time Warping (DTW) [Hof98] algorithm is used to align sequences of discrete elements. The goal is to find the optimal match of the two sequences, i.e. the alignment with the least number of insertions, deletions and substitutions.

More mathematically, given the feature vector \bar{X}_n for the n -th element, we are trying to find the path

$$u(n) = (u_1(n), u_2(n)) \quad (2.26)$$

so that the similarity measure D is minimized:

$$D^*(\bar{X}_{\text{ref}}, \bar{X}_{\text{usr}}) = \min_{u(n)} [D(\bar{X}_{\text{ref}}, \bar{X}_{\text{usr}}, u(n))] \quad (2.27)$$

$$(2.28)$$

With the accumulated distance function $d(\bar{X}_{\text{ref},i}, \bar{X}_{\text{usr},j})$ between the i -th reference element and the j -th user element, Dynamic Programming (DP) results in the iteratively calculated minimum distance

$$D^*(0, 0) = 0 \quad (2.29)$$

$$D^*(i, 0) = D^*(i-1, 0) + d(\bar{X}_{\text{ref},i}, *) \quad (i = 1, \dots, I) \quad (2.30)$$

$$D^*(0, j) = D^*(0, j-1) + d(*, \bar{X}_{\text{usr},j}) \quad (j = 1, \dots, J) \quad (2.31)$$

$$D^*(i, j) = \min \left[\begin{aligned} &D^*(i-1, j-1) + d(\bar{X}_{\text{ref},i}, \bar{X}_{\text{usr},j}), & (i = 1, \dots, I) \\ &D^*(i-1, j) + d(\bar{X}_{\text{ref},i}, *), & (j = 1, \dots, J) \\ &D^*(i, j-1) + d(*, \bar{X}_{\text{usr},j}) \end{aligned} \right] \quad (2.32)$$

and it is possible to find the minimizing path $u^*(n)$ by backtracking through

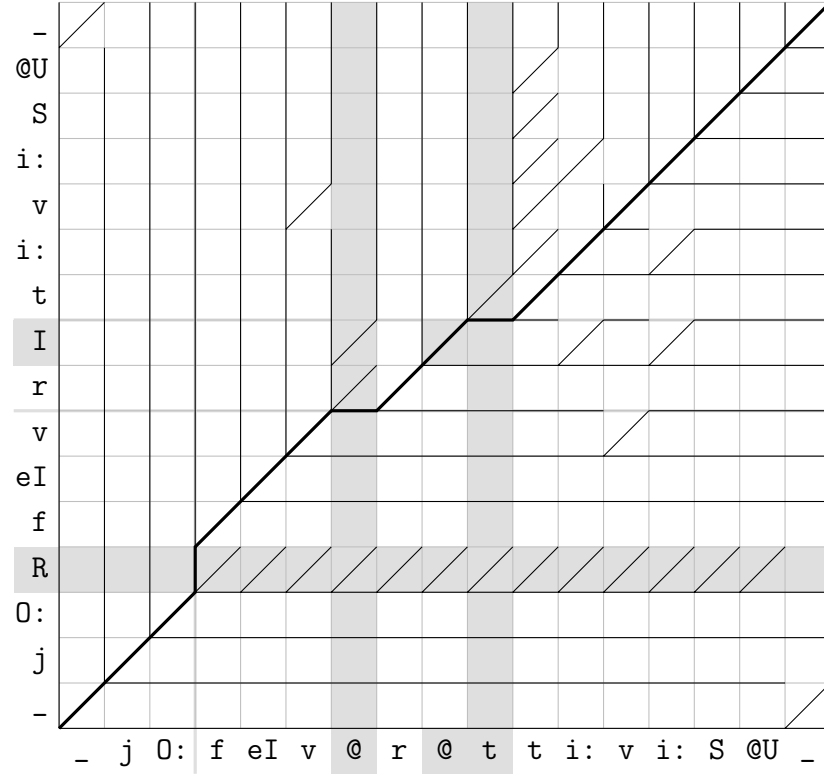


Figure 2.15: DTW for the label sequence of *Your favorite TV show*. One label is replaced, two labels exist only in the reference utterance at the bottom and one only in the user utterance to the left.

the distance matrix

$$u^*(N) = (I, J) \quad (2.33)$$

$$u^*(n) = \arg \min \begin{bmatrix} D^*(u_1^*(n+1) - 1, u_2^*(n+1) - 1), \\ D^*(u_1^*(n+1) - 1, u_2^*(n+1)), \\ D^*(u_1^*(n+1), u_2^*(n+1) - 1) \end{bmatrix} \quad (n = 1, \dots, N-1) \quad (2.34)$$

The special distance function values $d(\overline{X}_{\text{ref},i}, *)$ and $d(*, \overline{X}_{\text{user},j})$ denote the costs for deletion and insertion of elements respectively. Figure 2.15 shows an example of the sequence alignment.

3. An Automatic Annotation System

3.1. Requirements

A mostly automatic annotation system that is supposed to interoperate with external tools and their file and data formats and that can be used with arbitrary annotation tasks needs to fulfill all of the following requirements:

Platform independence is necessary to use the provided toolset on various current and future operating systems. It should be possible to

- copy the complete tool tree between different operating systems without or with only small changes,
- use one unique tree for shared network access,
- transfer the generated data files between platforms with different word size, byte order and encoding settings.

File format conversion must be provided to convert between the different formats used by external tools. Especially it must be possible to

- convert wave files between different file types, sample rates, channel numbers and sample sizes,
- transform label files between different formats and phone sets,
- read and write special formats used to store pitchmarks, f0 contours and Fujisaki parameters etc.

Unattended operation simplifies operation on large corpora. The system must

- fail gracefully in the event of an unexpected condition,
- provide meaningful error messages about the cause and code position of errors,
- be highly configurable and scriptable to the required task.

Easy maintenance is essential. It has to

- allow the easy correction of bugs,

- have all public interfaces and building blocks as well as the internal code structures documented,
- reuse available functionality internally as well as where provided by external programs to minimize redundancy,
- be structured in a modular way to allow extension with additional external programs and new modules.

Manual intervention should be possible to correct automatically calculated features. Especially

- phoneme-level labeling,
- syllabic structure and
- prosodic features

should allow easy modification.

3.2. Prerequisites

Automatic prosodic labeling of a corpus requires phone, syllable, word and sentence level features to be already present in the corpus annotation.

To automatically derive these features, forced alignment with the output from a Text to Speech (TTS) system should be used. This requires only the text that was presented to the speaker and the recorded signal to be available.

The following steps are therefore necessary to automatically calculate the basic phone level information:

- Grapheme Phoneme Conversion (GPC) of the text prompt
- Synthesis of the phonemes to get a synthesized speech signal
- Forced alignment of the synthesized and natural speech signal and derivation of the phoneme segmentation of the recorded signal

Furthermore, additional structural information available from the GPC and external dictionaries should be used to derive syllable, word and sentence level features.

3.3. System Structure

Figure 3.1 gives an overview of the proposed system structure. Each box represents a module slot that can be filled by a selectable implementation for that particular task. Shown are the required input and output files for each block.

Grapheme Phoneme Conversion The term GPC as used here comprises all steps necessary to convert the input text and convert it into phonemes. The module has to cope with multiple sentences, various text encodings

3. An Automatic Annotation System

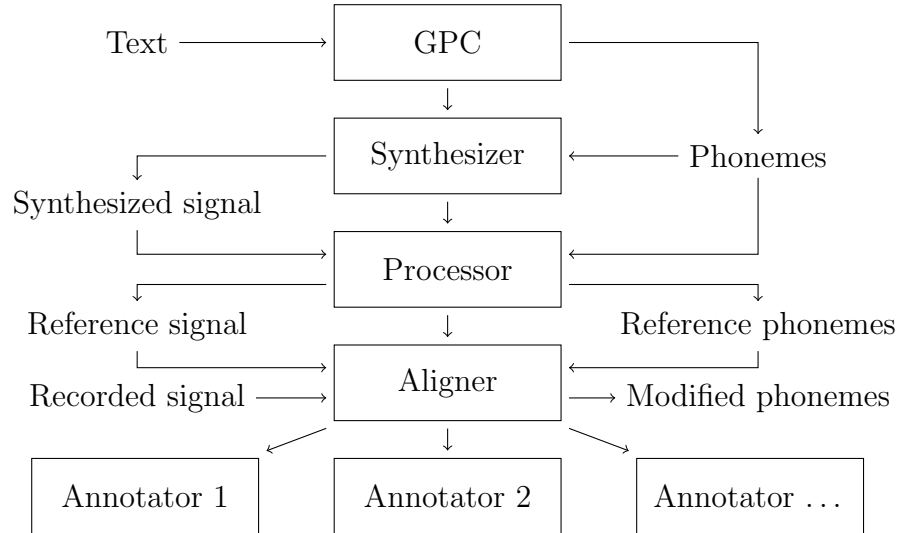


Figure 3.1: System structure

(especially umlauts), should generate additional structural information on syllable, word and sentence level and may provide predicted prosodic features like base frequency.

Synthesizer The synthesizer processes the phoneme and prosodic information from the GPC and produces audible output.

Aligner Preprocessor This step prepares the generated phonemes and the synthetic reference signal for the phoneme alignment.

Phoneme Aligner Now forced alignment between the reference and recorded signal is performed. Figure 3.2 shows the basic structure of a phoneme aligner that uses Dynamic Time Warping (DTW) for matching.

Certain features are calculated from the two signals which are then compared with DTW. Afterwards the labels of the reference signal are mapped to the most likely positions in the recorded signal.

Annotators The mapped label positions, the recorded signal and the linguistic data generated throughout the alignment process are used to automatically annotate the database with additional features.

3.4. Grapheme Phoneme Conversion

A GPC consists of the following steps [BTC02]:

Tokenization The input text is decomposed into tokens and utterance breaks are detected. Sentence breaks and abbreviations have to be distinguished.

Token to word conversation The tokens are analyzed and converted to words. Special care has to be taken for numbers, which may represent quantities, ordinals, dates or money amounts. Abbreviations and single letters are expanded.

3. An Automatic Annotation System

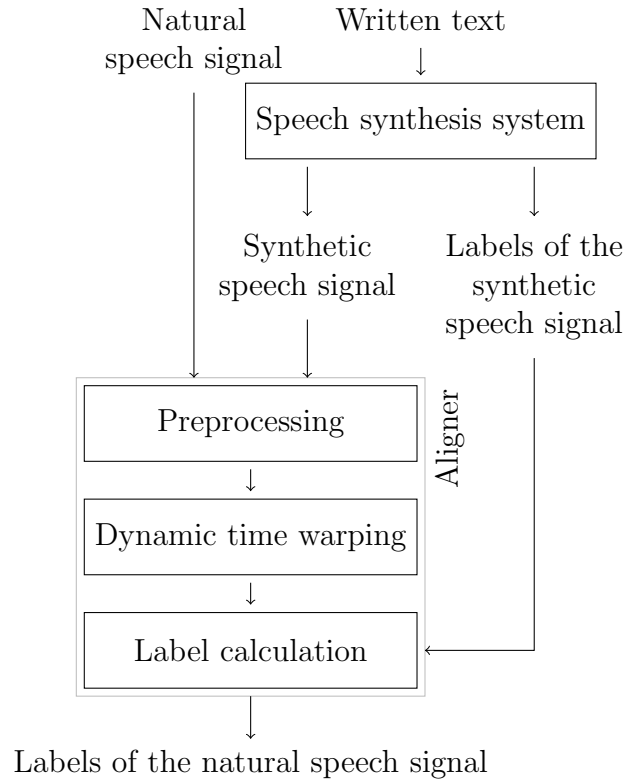


Figure 3.2: Phoneme Aligner structure (after [Str00])

POS tagging Each token is assigned a Part of Speech (POS) tag. Often HMM-based taggers are used that assign the tags based on the tag probability distribution for a word and ngram models.

Phrase break estimation Phrase breaks are determined that split the utterance into prosodic phrases. Different approaches exist that utilize Classification and Regression Trees (CARTs) or probabilistic models for phrase breaks depending on the neighboring words.

Lexicon lookup The lexicon provides pronunciation and phoneme information about a word. If a word can not be found, generic letter-to-sound rules are used to derive the phonemes.

Duration prediction The segmental duration for each phoneme is calculated. Different methods like CARTs or models depending on e.g. minimal and inherent durations can be used.

3.4.1. LaipTTS

The LaipTTS [KZ96] GPC was developed at the University of Lausanne and contains modules for French and German. Because of the unavailability of the source code, only basic support for German as provided by the underlying Java classes is supported. The GPC contains only incomplete handling of out-of-

vocabulary words and has no interface to obtain further linguistic information.

3.4.2. Festival

The Festival speech synthesis system [BTC02] provides multilingual synthesis and is developed at the Centre for Speech Technology Research (CSTR) at the University of Edinburgh. It is released under a non-copyleft license similar to the 3-clause BSD license.

It features voices for American and British English as well as Spanish and can be extended by third-party modules to gain support for additional languages, e.g. Dutch with the NeXTeNS project [KM02] and German with IMS Festival [Möh99]. Integrated are also several speech utilities that can be used to perform signal processing operations, training of language models or grammar generation. All synthesis modules can be used independently which allows the extraction of the results after the different steps of the GPC.

Although the system is written in C++, an integrated Scheme interpreter allows it to be programmed and extended during runtime. Scheme is a list-based language from the Lisp family and therefore a bit unusual-looking to somebody used to languages like Pascal or C. See [Sit04] for an introduction.

Festival is invoked with

```
sh> festival [--script FILE]
```

and presents a Scheme command line interface. Now it is possible to obtain help

```
festival> (help)
festival> (doc utt.save)
```

to select a voice for synthesis

```
festival> (voice_rab_diphone)
```

and to synthesize text with

```
festival> (SayText "Hi there")
```

Listing 3.1 shows a simplified example program that is used to synthesize multiple sentences from standard input and store them in one utterance file. It can be called with

```
sh> festival --script festivalwrap.scm voice_rab_diphone \
temp.est target.est
```

The first parameter is evaluated as a Scheme command (line 21) and used to set the required voice. The next command hooks the default Festival synthesis command chain to use `output-utterance` (line 15) to output each utterance to the temporary file name and append the contents to the result file (line 8). This file contains all the results from Festival's synthesis modules except the synthesized waveform.

```

(define (copy-file source-file target-file)
  (set! buffer (fread 8192 source-file))
  (if buffer
    (begin
      5      (fwrite buffer target-file)
      (copy source-file target-file))))

(define (append-file source-name target-name)
  (set! target (fopen target-name "a"))
  10  (set! source (fopen source-name "r"))
  (copy-file source target)
  (fclose source)
  (fclose target))

15 (define (output-utterance utt)
  "(output-utterance UTT)
  Output segment information in raw Festival format."
  (utt.save utt (cadr argv))
  (append-file (cadr argv) (caddr argv)))
20 ((eval (car argv)))
  (set! tts_hooks (list utt.synth output-utterance))
  (tts_file "-"))

```

Listing 3.1: Festival wrapper scheme file to synthesize input from stdin and to store it in Festival utterances

3.4.3. Alternatives

Other GPC programs are available that are not included in the framework at the moment. The DRESS system developed at the Dresden University of Technology provides multilingual speech synthesis for e. g. German, Italian and Chinese. For each language, several different speakers are available. It includes a GPC module as well diphone-based synthesis and can provide linguistic information together with the phonetic data. GPC for German is also provided by HADIFIX [PSP⁺92].

3.5. Synthesis

The synthesis module has to convert the phonemes into audible speech. Depending on the method used, this consists of steps like unit selection, segment concatenation and signal-based modification of prosodic parameters.

	n	n/N
b	152	1.72 %
p	175	1.98 %
d	277	3.14 %
t	528	5.99 %
g	73	0.82 %
k	280	3.18 %
tS	65	0.73 %
dZ	53	0.60 %
plosives	1603	18.20 %
total	8804	

Table 3.1: Frequency of plosives and affricates in the *rob200* corpus

Mbrola The most widely used synthesis package is Mbrola [DPP⁺96]. It is freely available for non-commercial purposes and is employed by the aforementioned HADIFIX, NeXTeNS and LaipTTS GPCs. Mbrola is based on a concatenative diphone engine and produces speech with a piecewise linear pitch contour. It uses an algorithm of the PSOLA family (Multi Band Resynthesis OverLap Add) for the concatenation that allows the smoothing of spectral discontinuities. Voices are available for a multitude of languages ranging from French to Indonesian.

3.6. Aligner Preprocessor

To prepare the generated phonemes and the synthesized signal for the forced alignment, an additional step provides the possibility of modifications before both reach the aligner. At the moment it is only used to prepare the labeling of plosives for alignment by splitting them into pause and burst.

Plosive Splitter Most available GPC systems do not generate a phoneme sequence that has plosives split into pause and burst. Nevertheless it would be attractive to have separate pause and burst labels for the forced alignment available to ease the burden on the human expert who has to post-process the automatic labeling results. Table 3.1 shows the frequency of occurrence of plosives and affricates for the *rob200* corpus.

To determine the boundary between pause and burst, several possible properties of the signal in the time domain or in the frequency domain could be evaluated. Because of the good signal quality and high SNR of synthesized speech, a straight forward approach in the time domain that is based on the energy signal is used. It will be shown that the presented algorithm provides more than adequate results for the given task (figure 3.3).

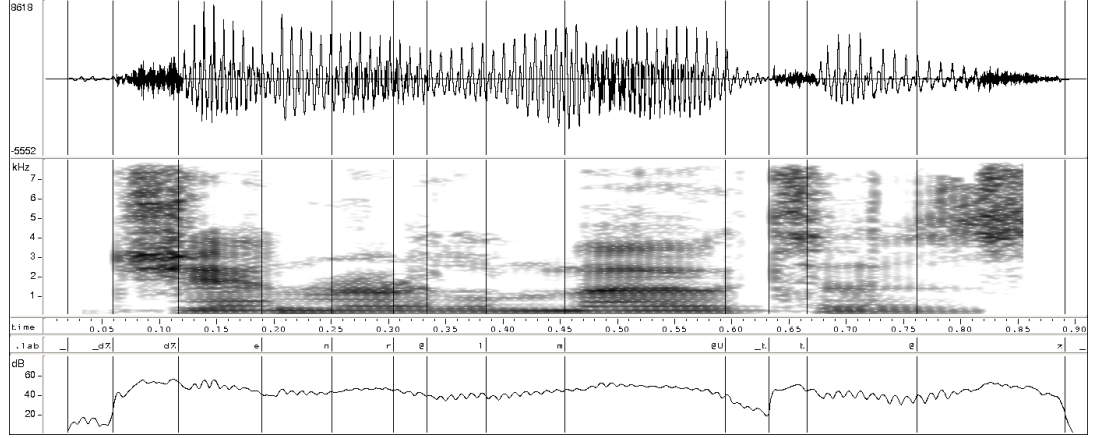


Figure 3.3: Time and frequency domain properties for plosives of an example utterance

For nearly all plosives the start of the burst correlates with a steep rise in the energy signal. The following parameters are used for the calculation of the short-time energy:

- Hamming window,
- window length of 10 ms,
- preemphasis of 0.97,
- frame interval of 1 ms.

For a given plosive segment, the longest monotonous increase in the energy signal from t_B to t_E is found. Two different approaches for the derivation of the pause-burst boundary t_b have been evaluated (figure 3.4):

- Relative position

$$t_b = t_\delta = t_B + \delta(t_E - t_B)$$

- Maximum slope

$$t_b = t_s = \arg \max (\Delta E(t))$$

3.7. Phoneme Aligner

Three different aligners are supported. All use forced alignment with different kinds of Dynamic Time Warping (DTW) to align a reference signal with the input signal and derive the phoneme segmentation.

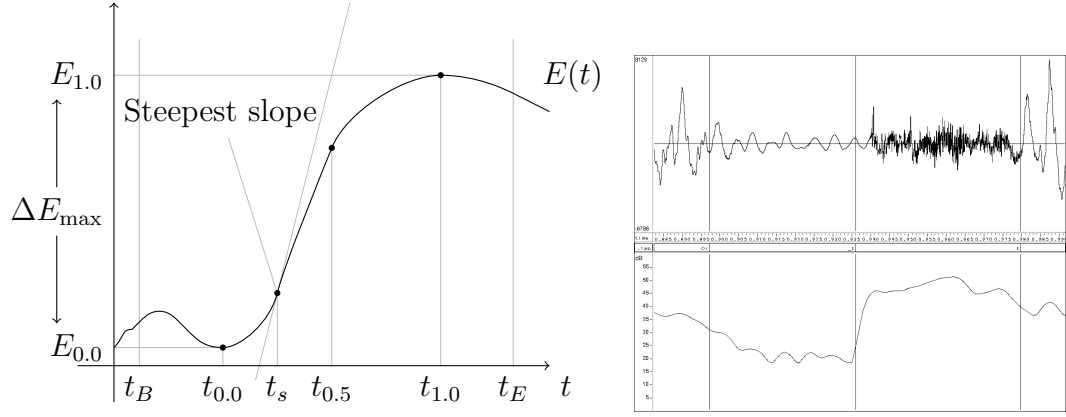


Figure 3.4: Calculating the boundary between pause and burst for plosive segments

3.7.1. Supported Phoneme Aligner Programs

Institute of Informatics Problems, Minsk This experimental aligner is developed by Andrej Davydov at the Institute of Informatics Problems, Minsk [Dav04]. Its main purpose is the alignment of phoneme segments for voice cloning [LT04] and is not directed at the alignment of longer phrases. Nevertheless it uses an interesting modified DTW algorithm and is therefore included in the comparison.

University of Bonn The *dsp_dtvalign* aligner was developed by Karlheinz Stöber at the IKP of the University of Bonn [Stö97]. It is a command line-base executable compiled for Cygwin/Windows [Cyg05], but can be successfully used under i386-based Linux operating systems with *Wine* [Win05].

Dresden University of Technology This aligner was developed by Guntrum Strecha at the Dresden University of Technology during his masters thesis [Str00]. It is highly configurable and has a user defined feature set. The aligner is used here with the recommended parameters from the provided evolutionary optimized settings file. The input signal is normalized, preemphasized and processed to derive the following features:

- Root Mean Square (RMS) energy of the signal and two derivatives
- zero crossing density and two derivatives
- 30 Mel Frequency Cepstrum (MFC) coefficients and one derivative

3.7.2. Performance Evaluation

There exist two major kinds of labeling errors:

- precision errors, i. e. how close do the predicted labeling positions match the ones of a human labeler

- transcription errors such as insertions, deletions and replacements of labels.

Both error types need to be evaluated separately to get meaningful results. In the following equations, N denotes the total number of manually labeled boundaries used for testing.

To estimate the precision of the aligner, the deviation of labels is rated by the *quality coefficient* Q_ρ [Hus99]

$$Q_\rho = \frac{n_\rho}{N}$$

where n_ρ is the number of labels that are less than ρ ms from the manually labeled position. In following [Str00], further evaluation is done on the basis of the *quality coefficients* Q_ρ for $\rho = \{10, 20, 30, 40, 50\}$.

The transcription error is calculated with the *accuracy measure* A [KWS97]

$$A = \frac{N - S - D - I}{N}$$

with the number of substitutions S , insertions I and deletions D .

3.8. Linguistic Annotation

3.8.1. Label Merger

To be able to compare alignment results and to modify labels for the further annotation process that have been manually corrected by a human expert, DTW with the feature vector

$$\overline{X}_n = \{N_n\}$$

with N_n being the name of the n -th label is performed. See appendix D.4 for the default distance function used in the framework.

For simplicity the comparison is solely performed on the label level with the names as parameters. It would also be possible to use additional features such as duration although this is not pursued further here.

3.8.2. Syllables

Because most dictionaries used for synthesis have syllable boundaries that are marked automatically and employed only for the stress determination of vowels, it is necessary to provide means for correction using an external dictionary for the target language. Because dictionaries differ in the phonetic description of words, a fault-tolerant way of merging syllable boundaries is needed.

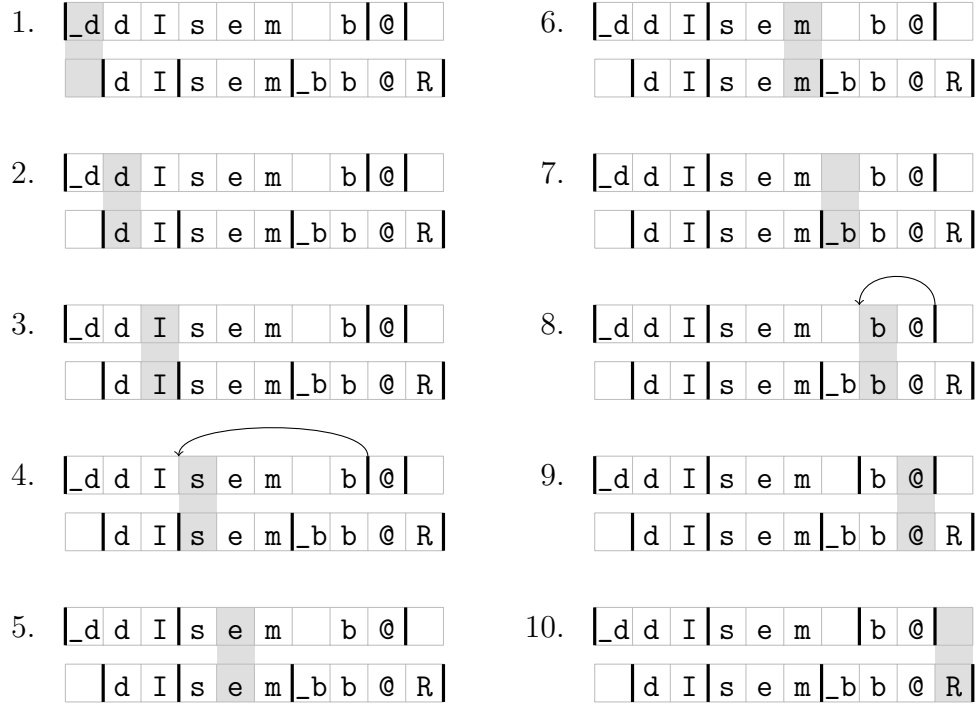


Figure 3.5: Merging external syllable information for the word *December*: the first realization is a recorded example, the second from an external dictionary

Similar to the approach for the integration of external labels, DTW is applied to the label sequence. After corresponding phonemes have been identified, the syllable boundaries are shifted to match the target syllable segmentation.

This process can be seen in figure 3.5. The first realization of the word *December* stems from the Festival dictionary for British English and has wrong syllable delimiters. Its labels have been manually corrected to match the actual recorded signal. The second realization is taken from an external dictionary with correct syllable segmentation. Step by step, the algorithm iterates over the phonemes, shifting and adding syllable boundaries as necessary.

3.9. Prosodic Annotation

The Technology and Corpora for Speech to Speech Translation (TC-STAR) project aims at producing speech corpora that can be used for building advanced state-of-the-art TTS systems as well as for intralingual and interlingual research on voice conversion and expressive speech [BHT⁺04]. It is going to provide high-quality language resources for UK English, Spanish and Mandarin. The voices are sampled at 96 kHz and with 24 bit precision and each voice is recorded in several corpora: Novels and short stories are included as well as expressive speech.

The required volume of 10 h of speech per voice corresponds to about 90 000 running words that need to be extensively annotated. Because of the sheer volume of data most of this task has to be performed automatically. The speech data analyzed here is build from single sentences and short paragraphs read by different male and female UK English voices. This corpus solely consists of well-defined read speech and does not contain spontaneous utterances.

The TC-STAR project requires the following prosodic annotation [BHT⁺04]:

Two-level phrase break annotation with distinction between *minor* (intermediate intonational phrases) and *major* breaks (full intonational phrases).
Intonational prominence annotated using two levels: *normal* and *emphatic*.

Several different approaches for the annotation of these features are presented in the following sections. The results can be found in chapter 4.

Besides for these features, no further research has been done to analyze other prosodic characteristics such as loudness, tempo, rhythm and emotional state as all of the used corpora consisted of planned speech.

For each prosodic feature, the possible values are divided into distinct classes and a confusion matrix between the reference and recognized classes is calculated. The following definitions are used (similar to [BKK⁺98]):

- Number of classes

$$n$$

- Number of elements in the reference class i that are recognized as class j

$$N_{i,j}$$

- Number of elements in the reference class i

$$N_i = \sum_j^n N_{i,j}$$

- Recognition rate for class i

$$RR_i = \frac{N_{i,i}}{N_i}$$

- Overall recognition rate

$$RR = \frac{\sum_i^n N_{i,i}}{\sum_i^n N_i}$$

- Average recognition rate

$$\overline{RR} = \frac{1}{n} \sum_i^n RR_i$$

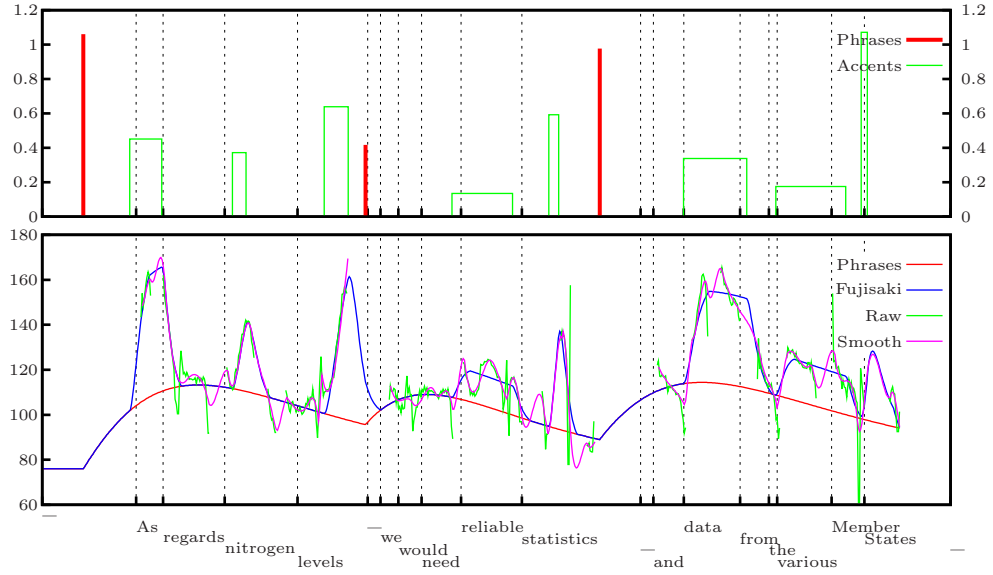


Figure 3.6: Fujisaki parameter example: *As regards nitrogen levels, we would need reliable statistics and data from the various Member States.*

3.9.1. Rule-based Break Estimation

The Festival system predicts phrase breaks with two different methods: by a CART or with a probabilistic model that is enabled for the American and British English voices that come with Festival [BTC02]. It uses the probabilities of breaks based on the POSs of the previous and following words of a break combined with an ngram model of the break distribution to optimize the phrasing. These automatically determined breaks could correspond to the breaks made by humans.

3.9.2. Derivation from Fujisaki Parameters

Automatically extracted Fujisaki parameters from the f0 contour [KL03] should correspond to accents and phrase breaks sensed by a human listener (3.6). To test this hypothesis, several assumptions are made:

- Fujisaki phrases correspond to phrases perceived by a listener,
- the strength of an accent is proportional to the Fujisaki accent amplitude and the accent length,
- the observed accent strength may depend on the word length,
- an accent spanning multiple words may emphasize only the main or all words,
- the accent strength is perceived relative to the strongest accent in the phrase.

These lead to the following implementation for the derivation algorithm:

1. The utterance is split into phrases according to Fujisaki phrase commands. A phrase command within a word leads to a phrase starting with the next word, as does a phrase command within silence.
2. The accents are assigned to words. Multiple accents for one word are added. If only one word per accent is permitted, the accent area that falls between the word limits has to be more than 50 %.
3. All accumulated accent scores are divided by the word length if requested.
4. The maximum accent per phrase is calculated.
5. A word is marked as accented if the accent score crosses a certain threshold.

3.9.3. Neural Network Optimization

A different approach that uses more input data than only the Fujisaki parameters can be implemented by machine learning techniques. A well known method for supervised learning is the use of ANNs.

The network simulation is done by the well known Stuttgart Neural Network Simulator (SNNS). It was developed by Andreas Zell and others at the University of Stuttgart [Zel95] and forms the basis of Java Neural Network Simulator (JavaNNS), an interactive Java based simulation environment created at the University of Tübingen [FHBZ02]. The annotation framework employs some of the classes developed in the JavaNNS project to interface the SNNS kernel for simulation.

Experience has shown that no determinate rule exist that can be used to find the optimal network structure, number of hidden layers, transfer functions etc. for a particular task [JH04]. The here employed neural network structure is derived from similar networks used for pattern recognition and prediction and has given good results for a variety of problems.

The structure chosen is a multilayer feed-forward network with a variable number of neurons in the two hidden layers. The input unit number is equal to the size of the feature set, and one output unit for the requested parameter is used. All neurons are realized by a sigmoid activation function and are trained with backpropagation.

The output value for each pattern is set to a value between zero and one, depending on the number of classes for the training. The training pattern set is grouped by category and patterns are duplicated as necessary to get groups of equal strength.

A large number of input features for the prediction of prosodic parameters have been proposed that model base frequency, duration, energy and linguistic information on the syllable and word level [BBH⁺99], especially

- f0 onset, offset and linear regression coefficients,
- absolute and normalized duration,
- energy contour linear regression coefficients,

3. An Automatic Annotation System

- POS tags [BBH⁺00],
- rule-based boundary labels [BWN⁺98].

Based on these features, the here implemented solution is additionally based on the following assumptions:

1. Only word-level features are used, as syllable-level features do not provide additional performance according to [BBNW00].
2. A configurable context window provides the network with information of previous and following words. The best window size has to be determined by optimization.
3. If possible, both normalized and absolute values are provided. Although normalized variables seem to serve intuitively better as network input, they may be seriously outperformed by absolute values in some cases [BNB⁺01].

The following input fields are provided:

Duration Several different features describing the duration of the current word are provided:

- Absolute word duration
- Normalized word duration as calculated from the mean phoneme durations of the whole corpus
- Scale factor comparing absolute and normalized word durations

Linguistic information The output from the speech synthesis system is leveraged to provide information about POS tags and possible phrase structure:

- The POS calculated by the synthesis HMM tagger following the *Penn Treebank* notation with 45 classes.
- A derived POS tag that uses only 16 simplified classes.
- Phrase break information from the probabilistic Festival tagger.

Base frequency and power contour The logarithmic f0 and power contours are described with the following features (figure 3.7). They are calculated for raw as well as for smoothed contours where applicable.

- First signal value A_{on} . For f0 contours, the first value of the first voiced segment is used.
- Last signal value A_{off} . For f0 contours, the last value of the last voiced segment is used.
- Maximum and minimum value A_{min} and A_{max} .
- Mean value A_{mean} , for f0 contours only voiced segments are included.
- Absolute and relative position of the maximum and minimum value t_{min} and t_{max} .
- Linear regression coefficients m and n and residual sum of squares R for the contour. For f0 contours, only voiced segments are considered.

Fujisaki parameters Analogous to the EO, the following input fields are calculated for Fujisaki parameters:

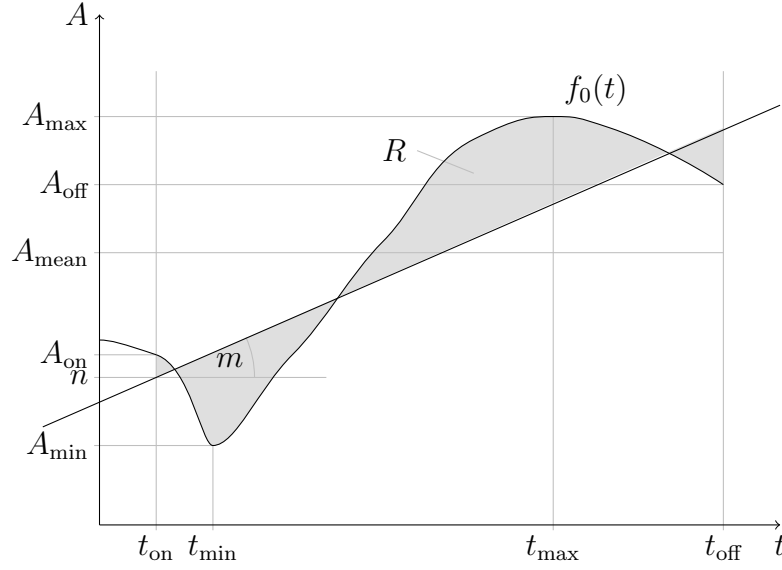


Figure 3.7: Features used to describe base frequency and power contours

- Accumulated accent area (score) per word for all Fujisaki accent commands.
- Phrase command amplitude if there is a phrase command in the current word or in a possibly following silence.
- Maximum accent score per Fujisaki phrase as derived from the Fujisaki phrase commands.

Structural information Certain additional positional and structural fields are provided. Phrase segmentation for these features is obtained from the phrase break annotation of the Festival synthesis.

- Number of syllables in the current word, current phrase and previous phrase.
- Index of the current word in the phrase and utterance.
- Index of the first syllable of the current word in the phrase and utterance.
- Start and end time of the current word and phrase.

3.10. Polarization

The phonation process leaves traces in the speech signal that make it possible to determine the polarization of an audio signal required for pitchmark algorithms based on inverse LPC filtering. The following algorithm can also be used to determine the polarization of an EGG signal. It is based on the fact that the negative Bernoulli pressure caused by the air flow modifies the speed of movement of the vocal folds. The otherwise symmetric oscillation is shifted towards

```

X = X - mean (X)
X = -accumulate (X)
X = highpass (X, 90)
max = 0
min = 0
for (j = scale (800); j <= scale (50); ++j) {
    Gamma = transform (X, j)
    maxmean = mean (maxima (Gamma, 1/800))
    minmean = mean (minima (Gamma, 1/800))
    max += mean (only (maxima (Gamma), > 0.5 * maxmean))
    min += mean (only (minima (Gamma), < 0.5 * minmean))
}
score = (max + min) / (max - min)

```

Listing 3.2: Speech signal polarization determination

a faster closing movement that creates steeper transients in the EGG and source signals. A signal is regarded to have positive polarization if the negative transients are stronger (e.g. the source signal, where the closing of the vocal folds causes an abrupt reduction of the air stream) and to have negative polarization with larger positive transients (e.g. an EGG signal that is proportional to the admittance of the vocal folds which increases on closing).

The used algorithm is almost identical with the one described in [Eng03] (listing 3.2):

1. The DC part of the signal is removed, creating a signal that is zero at the beginning and the end after the next step.
2. The negative accumulated signal is calculated

$$x_a(k) = \sum_{n=0}^{k-1} x(n)$$

3. An FFT high pass filter with 90 Hz cut-off frequency is used to remove low-frequency noise caused by e.g. head movements.
4. The coefficients $\gamma_j(k)$ of the cubic B-spline wavelet are calculated for corresponding scales j from 50 Hz to 800 Hz.
5. The averages of all local maxima and minima of the wavelet coefficients for each scale are used as a threshold to calculate filtered averages without low-amplitude extrema.
6. The filtered averages are accumulated for all scales, the polarity of the sum determines the polarization of the signal.

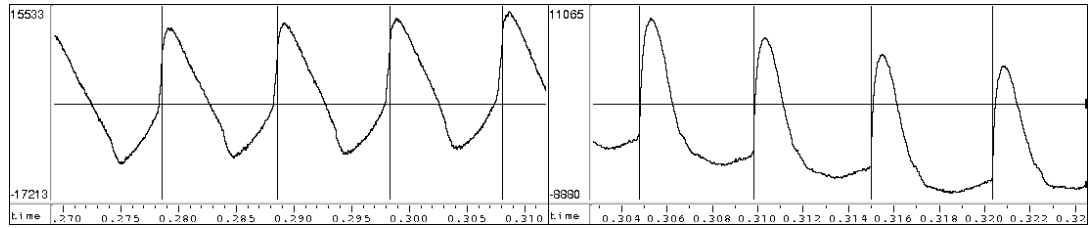


Figure 3.8: Comparison of two EGG signals of two different speakers. In contrast to the right example, the left signal has no easily detectable GCIs with distinct inflection points.

3.11. Pitchmark Extraction from EGG Signals

Pitchmarks are required to enable synthesis methods like PSOLA to manipulate the base units in pitch and length. To concatenate different units, the pitch periods additionally have to be marked in a consistent way to achieve smooth transitions. Different methods for the placement of markers in a pitch period are in use:

- Maximum or minimum peak in the audio signal
- Zero crossing preceding or following an extremum
- Maximum amplitude of the EGG signal
- Start of the glottal closure as observable in the EGG signal
- Maximum slope of the EGG signal

The selection of a suitable position is further complicated by the fact that it should be determinable both from the EGG and audio signal. The analysis in [Kot05] has shown that only the point of the beginning of glottal closure (defined here as the GCI) in the EGG signal can be correlated to a time-domain feature. GCIs marked in this manner and corrected for a nearly constant time delay coincide with the most negative-going peak of the audio signal.

The GCI is easily observable in most EGG signals. It is characterized by a sharp bend after a steady signal part that is followed by the steepest edge in a period (figure 3.8). Because it may be impossible to detect the inflection point at the GCI, the following algorithm (similar to the approach in [HI84]) extends the one developed by Toni Engel and Hans Kruschke [Eng03] and tries to determine the position of the steepest slope as the GCI (listing 3.3):

1. The polarization of the signal is tested. Signals with positive polarization are inverted.
2. An FFT band pass filter is used to remove low-frequency noise caused by e. g. larynx movements and high-frequency noise from the detector circuit.
3. The resulting signal is normalized.
4. Offsets at the beginning and the end of the signal are removed by addition of a linear function.

```

X = X - linear (X(0), X(length))
X = bandpass (X, 40, 2000)
Gamma = 0
for (j = 0; j <= 8; ++j) {
    Gamma += normalize (transform (X, j))
}
pitchmarks = threshold (maxima (Gamma, 1/800), 9 * 0.1)

```

Listing 3.3: Determination of GCIs from an EGG signal

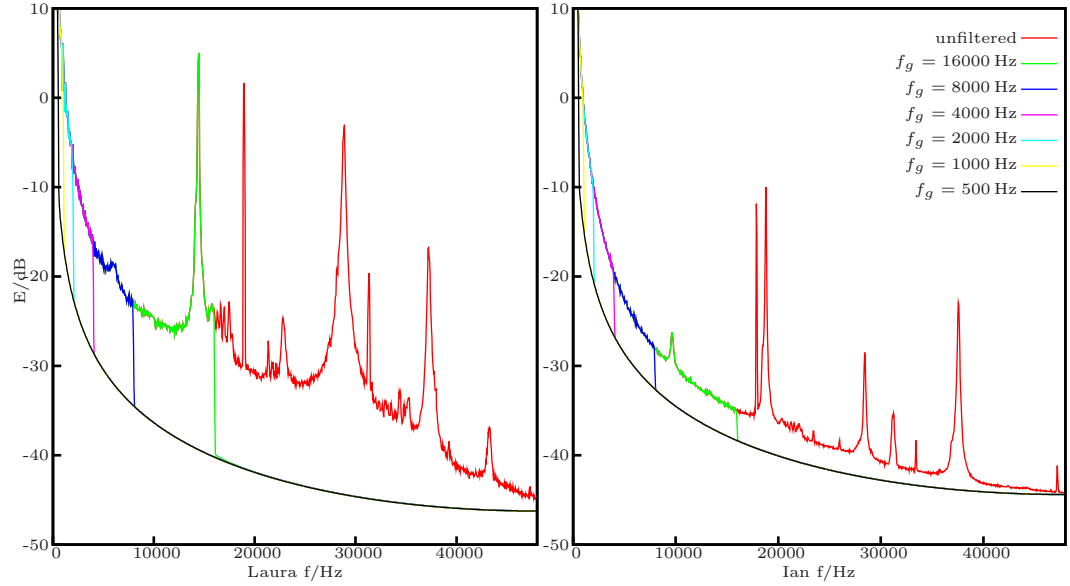


Figure 3.9: FFT spectra of the EGG signals for two different speakers before and after low pass filtering with different cutoff frequencies

5. The coefficients $\gamma_j(k)$ of the cubic B-spline wavelet are calculated for scales j from 0 to 8, normalized and accumulated.
6. The positions of the local maxima larger than a certain threshold correspond to the GCIs.

3.11.1. Noise Filtering

Despite perfect recording conditions [Eng03], some speakers' EGG signal may exhibit a low SNR that complicates the pitchmark extraction. Low amplitude thresholds for the EGG extraction will then result in noise being mistakenly marked as pitchmarks, whereas high thresholds cause low-amplitude GCIs to not be detected. Figure 3.9 shows the average FFT spectra for two recordings of a male and a female speaker. Quite visible are the high frequency noise spikes,

f_g/Hz	<i>laura</i>					<i>ian</i>	
	SNR/dB	$ \overline{\Delta t} /\text{ms}$	n_c	n_t	n_e	SNR/dB	$ \overline{\Delta t} /\text{ms}$
unfiltered	24		447	60	18	37	
16000	29	0.031	447	60	18	42	0.014
8000	39	0.032	447	60	18	44	0.017
4000	40	0.039	446	61	18	46	0.033
2000	42	0.062	446	61	18	48	0.058
1000	44	0.095	446	61	19	49	0.136
500	47	0.105	450	57	21	50	0.225

Table 3.2: SNR, average pitchmark deviation, number of coinciding pitchmarks n_c , pitchmarks derived only from time-domain characteristics n_t and from the EGG signal n_e for a low pass filtered EGG signal of a female speaker. The two rightmost columns show SNR and deviation for an example of a male speaker.

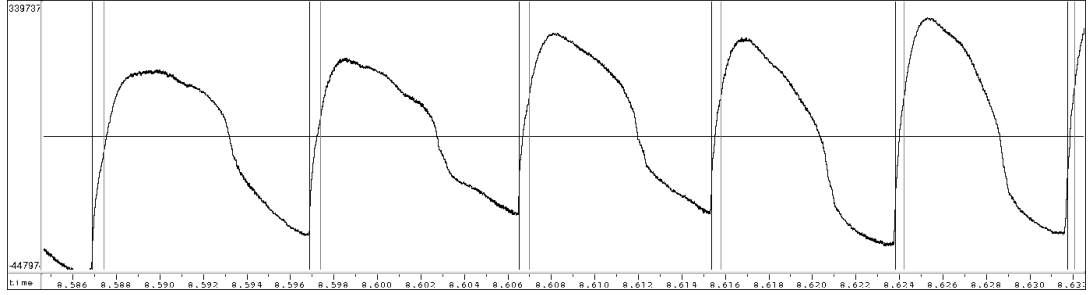


Figure 3.10: Pitchmark extraction from an EGG signal without (black) and with (gray) low pass filtering ($f_g = 500 \text{ Hz}$)

e.g. at 18 kHz, 28 kHz and 37 kHz for both recordings and the higher overall noise level for the *laura* speaker.

An FFT low pass filter can be used to suppress these frequency ranges. For different cutoff frequencies, table 3.2 shows the achievable SNRs of the two speakers with and without filtering.

The filtering alters the signal shape and therefore influences the pitchmark extraction. Figure 3.10 shows an example of the shift that is introduced for an aggressive low pass filter with $f_g = 500 \text{ Hz}$. More moderate cutoff frequencies cause much lower deviations as can be seen in table 3.2.

3.11.2. Combining Pitchmarks Algorithm

Physical properties of the phonation process inhibit the detection of pitchmarks from the EGG signal in certain transitional states of the vocal tract. Weak phonation as present during the beginning and the end of a vowel does not cause the vocal folds to open completely, therefore making it hard or impossible

3. An Automatic Annotation System

to observe the abrupt decrease in impedance when the vocal folds start to close again. Three examples of such problematic segments can be seen in the figures 3.11 to 3.13.

On the other hand, time-domain based pitchmark algorithms make assumptions about allowed pitch periods that create problems with e.g. laryngealization [BBJK93]. To keep the advantages of both methods and to overcome some of their problems, it would be favorable to merge EGG and time-domain derived pitchmarks.

The following assumptions are made:

1. The time-domain pitchmarks coincide with the GCIs as observable in the time-domain signal. This requires a derivation based on an algorithm like inverse LPC filtering as implemented in [Eng03].
2. The steep decrease in impedance of the EGG signal marks the start of the closing of the vocal folds and occurs before the actual GCI. This interval t_d is considered nearly constant.
3. The distances between vocal folds and lips t_v as well as lips and microphone t_a do not change significantly. The length of the vocal tract depends on the age and sex of the speaker, the here used values were determined in [FG99] for male and female speakers in the age of 18 to 25. The studio settings are similar like the ones in [Eng03] with a distance between lips and studio microphone of about 20 cm.

More formally, the interval between the i -th decrease in the EGG signal t_E and the GCI t_G as observed in the time-domain signal can be described with

$$\Delta t_i = t_{E,i} - t_{G,i} = \Delta t_{d,i} + \Delta t_{v,i} + \Delta t_{a,i} \quad (3.1)$$

and

$$\Delta t_d = (0.2 \pm 0.1) \text{ ms} \quad (3.2)$$

$$\Delta t_v = \frac{(0.15 \pm 0.01) \text{ m}}{343 \text{ m s}^{-1}} \quad (3.3)$$

$$\Delta t_a = \frac{(0.21 \pm 0.02) \text{ m}}{343 \text{ m s}^{-1}} \quad (3.4)$$

which results in an average delay of

$$\Delta t = (1.25 \pm 0.42) \text{ ms} \quad (3.5)$$

With those assumptions, these relations for the distance shift between EGG and time-domain markers should also be true:

- The standard deviation of all distances between EGG and time-domain markers should be lower than the one for the distances between time-domain and the following EGG markers.

$$\sigma(t_{G,i} - t_{E,i}) < \sigma(t_{E,i} - t_{G,i+1})$$

3. An Automatic Annotation System

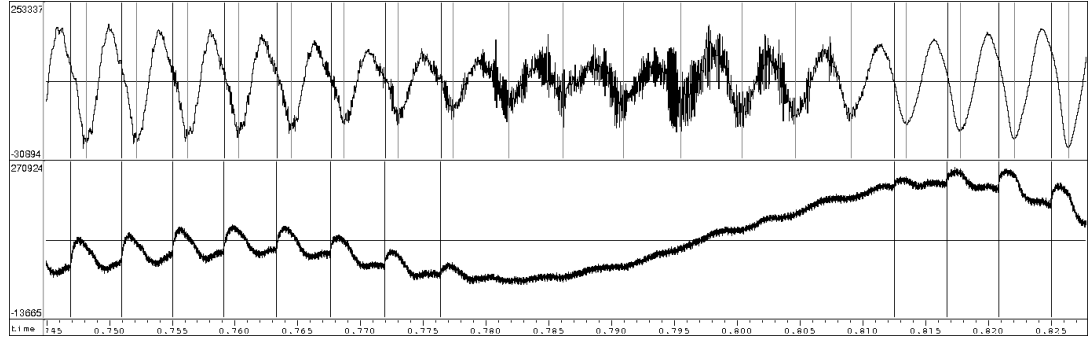


Figure 3.11: EGG signal with reduced phonation and missing GCIs between vowels (black pitchmarks extracted from the EGG signal, gray markers from the audio signal)

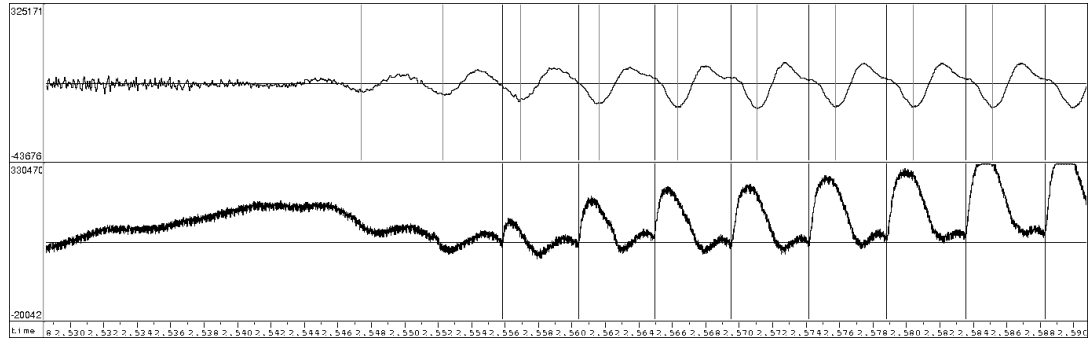


Figure 3.12: EGG signal with reduced phonation and missing GCIs before a vowel (black pitchmarks extracted from the EGG signal, gray markers from the audio signal)

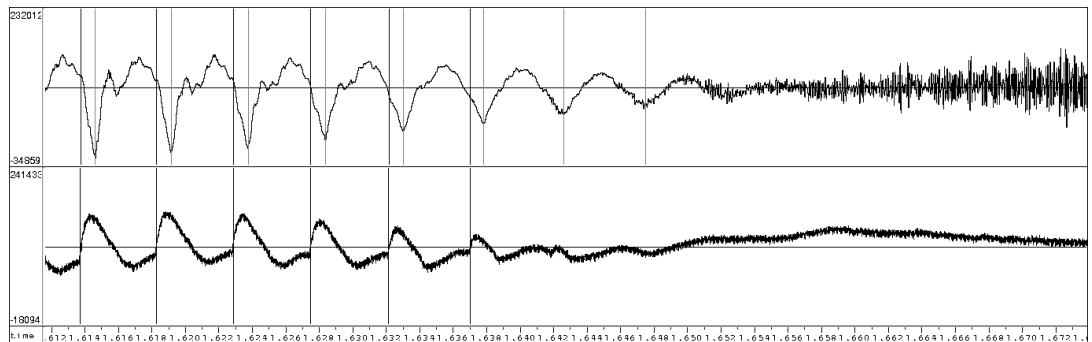


Figure 3.13: EGG signal with reduced phonation and missing GCIs after a vowel (black pitchmarks extracted from the EGG signal, gray markers from the audio signal)

3. An Automatic Annotation System

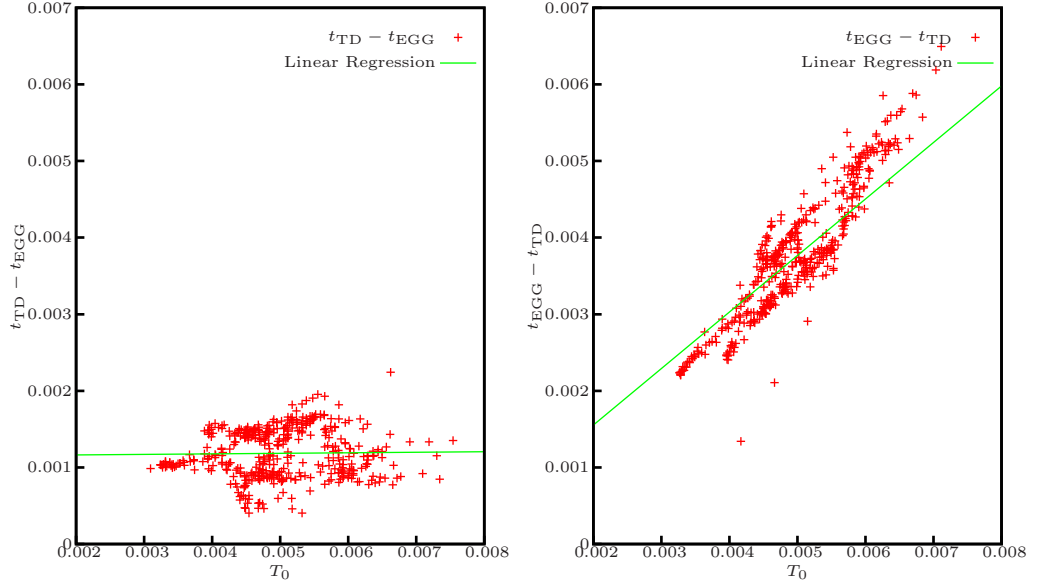


Figure 3.14: Example distribution of the distances between EGG and time-domain markers and linear regression results for a female speaker depending on the pitch interval T_0

- For the relation between pitch interval and distance between EGG and time-domain markers, the coefficient m and the determination of linear regression analysis should be lower for the distances between EGG and time-domain markers than for the ones between time-domain and the following EGG markers.

$$m(t_{G,i} - t_{E,i}) < m(t_{E,i} - t_{G,i+1})$$

$$r^2(t_{G,i} - t_{E,i}) < r^2(t_{E,i} - t_{G,i+1})$$

Figure 3.14 shows the distribution of such distances for an example utterance of a female speaker where these relations are true. They can be used to determine whether the given pitchmark data can be merged successfully.

The combination of the two pitchmark tracks is therefore done in three steps:

1. Validation of the above mentioned criteria that the pitchmarks are in the right order and can be merged.
2. Determination of pitchmarks that have been detected from both the EGG and audio signal.
3. Interpolation of new positions for pitchmarks that are only available from either the EGG or audio signal.

The last step can either be done by the interpolation of the f_0 contour in the segment that is supposed to have voiced pitchmarks [FKLL04] or by the direct calculation of pitchmark positions from the other track. This implementation uses the latter approach to transfer pitchmarks from one track to another:

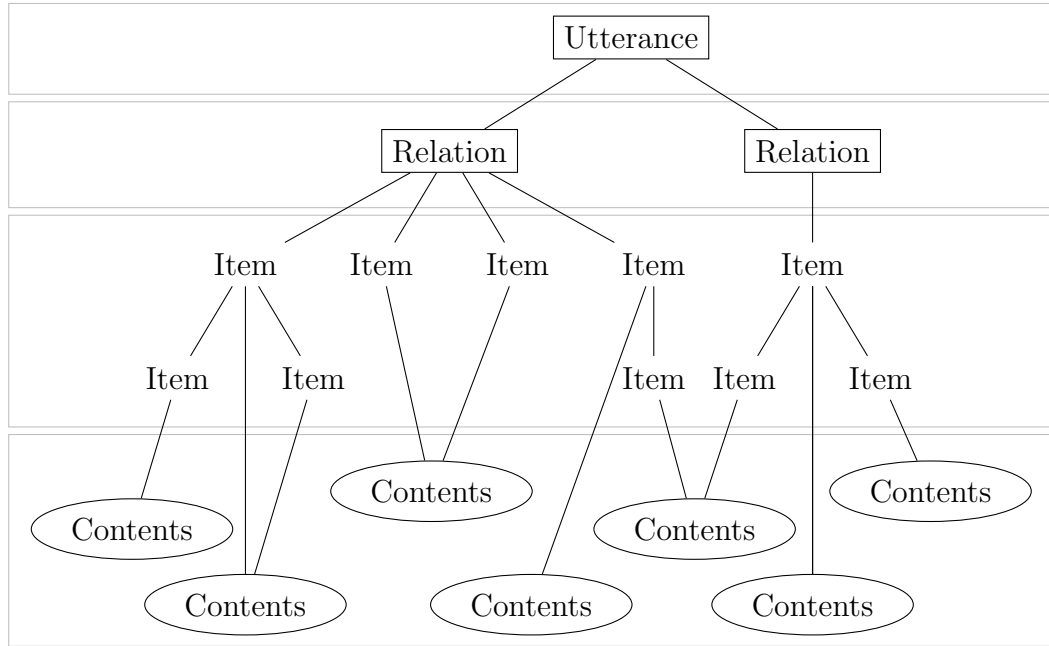


Figure 3.15: Utterance structure (framed nodes can contain features)

- Pitchmarks that are not preceded or followed by another pitchmark that can be considered adjacent (i. e. the distance is less than the maximum pitch period) are copied to the other track with a shift that is equal to the median shift of all pitchmarks that are available in both tracks.
- Pitchmarks that have either following or preceding adjacent pitchmarks that lead to a pitchmark that is available in both tracks (i. e. pitchmarks at the beginning or end of voiced sections) are copied with a shift equal to the one of the pitchmark available in both tracks.
- Pitchmarks that have both following or preceding adjacent pitchmarks that are linked to the other track get copied with a linearly interpolated shift of the ones from the two linked pitchmarks.

External components of the delay could be isolated by the comparison of the audio signal of the studio microphone with that of a headset microphone, although this is not pursued further here.

3.12. Data Storage

The main storage format for the framework is derived from the utterance structures that are used by Festival [BTC02]. It allows easy storage of interconnected data in different formats.

An *utterance* is a data structure that can hold information in one or more so called *relations* (figure 3.15). A relation is a tree-like structure of items that

3. An Automatic Annotation System

Relation	Description
Token	The token sequence from the tokenizer. After token to word conversation, each token contains the generated words and punctuation marks as offsprings.
Word	List of words as generated from the token to word conversation. Each entry in this relation has children in the <i>SylStructure</i> relation.
Phrase	Words divided into phrases. Each item denotes a phrase and has the words within it as leaf nodes.
SylStructure	Contains all words and punctuation marks from the <i>Token</i> relation. For a word, the daughters are its syllables which in turn contain their segments.
Syllable	List of all syllables in the utterance.
Segment	Contains the phones for the utterance. All but the silences are also in the <i>SylStructure</i> relation as children of the corresponding syllable.
Target	List of all segments that have predicted f0 values as daughters.

Table 3.3: Commonly used relations

Type	Description
String S	Text-based values like phoneme names as well as enumerated values for e. g. POS
Integer I	Boolean values and integral numbers
Double F	Floating point values, e. g. the results from signal processing operations

Table 3.4: Feature types

link to the actual content objects. This format is capable of both storing highly interconnected knowledge about e. g. the syntactic structure of a text as well as list data from signal processing operations like base frequency estimation in one common structure.

The most important relations from Festival are shown in table 3.3. Figure 3.16 shows the contents of these relations for a sample utterance. More relations that are related to the internal workings of Festival’s synthesis exist, see [BTC02] for an exhaustive list.

The actual information in an utterance is contained in the so called *features* that are available for content objects, relations and the utterance itself. The possible feature types are shown in table 3.4, the most common features for the relations given above in table 3.5.

See appendix C.3 for the description of a tool to explore Festival utterances.

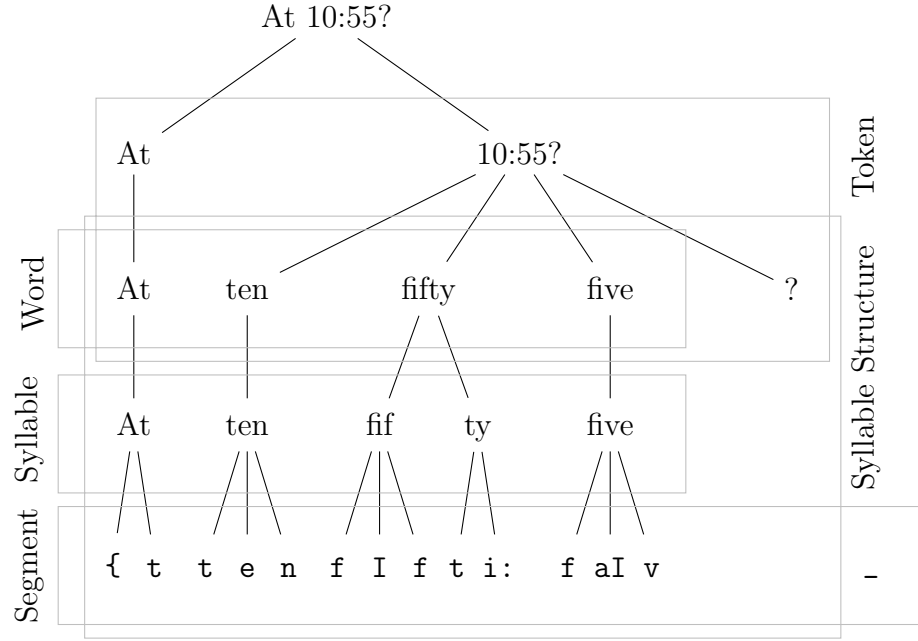


Figure 3.16: Example utterance (boxes denote relations)

Relation	Feature	Type	Description
Token	name	S	Token contents that has been stripped from punctuation symbols
	whitespace	S	Whitespace before the token
	prepunctuation	S	Punctuation preceding the token, e.g. quotation marks
	punc	S	Punctuation succeeding the token, e.g. question marks or commas
Word	name	S	Word name
	pos	S	POS class of this word
	phr_pos	S	Simplified POS class
	pbreak	S	phrase break type after this word: NB denotes no break, B or BB intermediate or full phrase break
Phrase	blevel	I	phrase break level
	name	S	phrase break type
Segment	name	S	Phoneme name
	end	F	End time of the segment
Target	f0	F	F0 value
	pos	F	time of the f0 value

Table 3.5: Commonly used features

4. Results

4.1. Used Databases

Several male and female speakers for the TC-STAR project have been used to obtain the following results. An additional external database was employed to estimate the applicability of an algorithm for inter-corpus prediction. The *kate* corpus from a female speaker consists of 1197 short phrases that contain mainly names, dates, numbers and web addresses. Manual phoneme labeling exist for 40 sentences with a total of 726 phonemes.

4.2. Plosive Splitter

A comparison of the results of the plosive splitter for different settings can be seen in figure 4.1. The used data consisted of 40 sentences extracted from the *kate* corpus that were synthesized with the Festival *rab* voice. The pause-burst boundaries of all plosives were manually corrected to be within a 1 ms distance before the burst.

Various values for the parameter δ as well as positioning based on the point of the steepest slope were used. The results show that different parameters may be appropriate. If the labeling task requires low overall errors, parameter values like $\delta = 0.5$ may be appropriate although this will cause more than 25 % of all labels to be shifted towards the burst section. That amount can be decreased with $\delta = \{0.0, 0.3\}$ which in turn give slightly higher overall errors. A good compromise denotes the selection of the position of the steepest slope.

These results should be readily applicable to other voices or synthesis methods. The main error source are plosives that have no noticable pause before the burst or no burst at all. It may be appropriate to exclude certain plosives and affricates like **b** from the splitter or to investigate methods of automatic identification for such irregular plosives.

A multi-step algorithm could increase the precision of the determined position of the boundary. A first analysis with a larger window for the power calculation would identify the rough position, followed by passes with reduced window sizes to find the exact position.

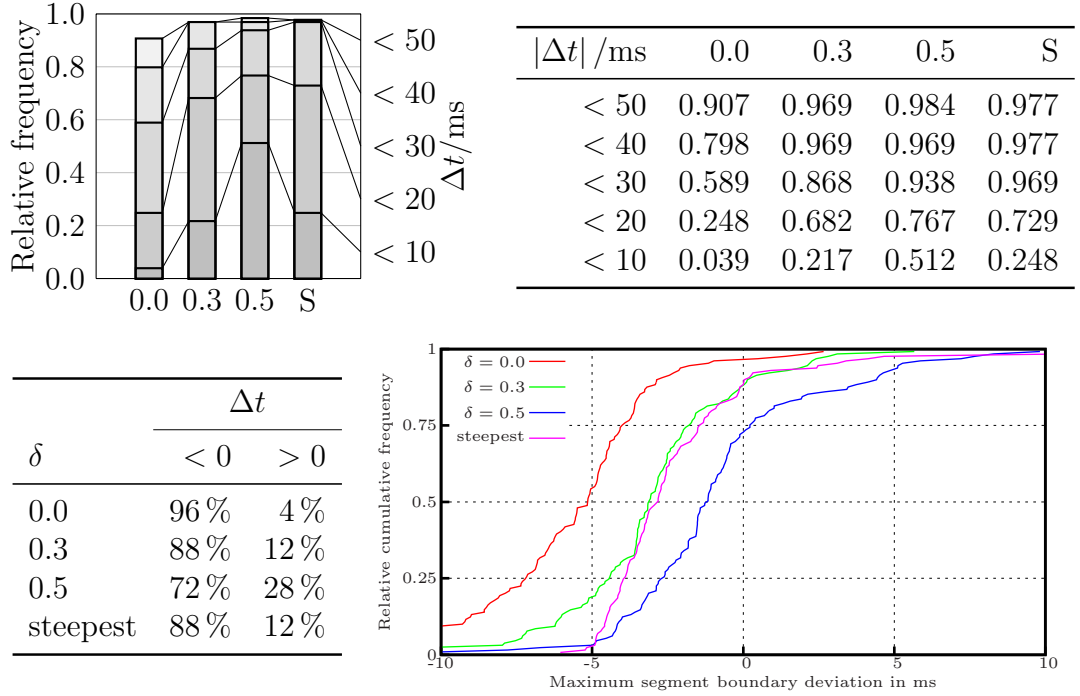


Figure 4.1: Histogram and plot of the pause-burst boundary errors for plosives and affricates of the segmentally labeled part of the *kate* corpus: relative position with $\delta = \{0, 0.3, 0.5, 0.7\}$ and (S)teeppest slope

4.3. Phoneme Aligner

4.3.1. Precision Errors

To estimate the precision errors that occur with the mentioned aligners, the segmentally labeled sentences from the UK English *kate* corpus are used.

A plain comparison of the three available phoneme aligners is shown in figure 4.2. Displayed are the percentage of automatically detected phoneme boundaries that deviate less than $|\Delta t| = \{10, 20, 30, 40, 50\}$ ms from the manually labeled boundaries and the error distribution. Negative error values denote automatic labels that have been placed earlier than the corresponding manual ones.

The aligner of Guntram Strecha has about 47 % of all labels within a 10 ms range from the correct position, followed by the aligner of Karlheinz Stöber. Oddly enough the plot of the relative cumulative frequency for the range from -100 ms to 100 ms shows that for no aligner the curve crosses the 50 % mark at a deviation of 0 ms.

Depending on the aligner, a certain shift seems to be necessary to equalize the plot. This amounts to about 4 ms for Strecha’s aligner and 8 ms for the one of Stöber. Although these errors may be caused by wrong labeling of the synthesized reference as obtained from the speech synthesizer, it seems not clear why such a systematic error would depend on the type of aligner used.

4. Results

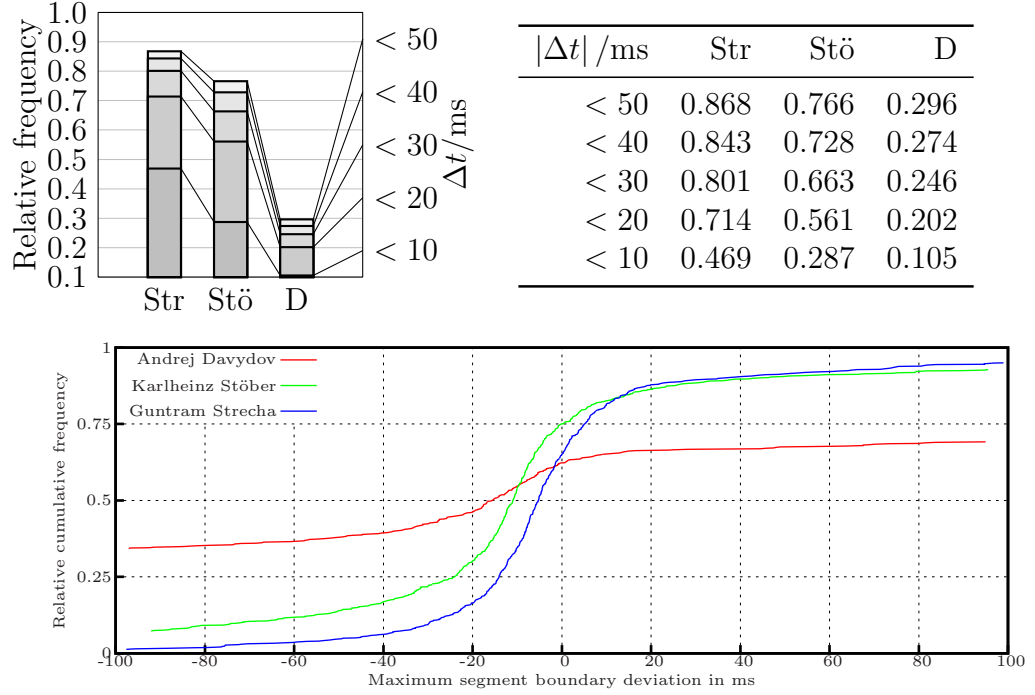


Figure 4.2: Histogram and plot of the segmentation boundary errors for an unshifted reference: (Str)echa, (Stö)ber, (D)avydov

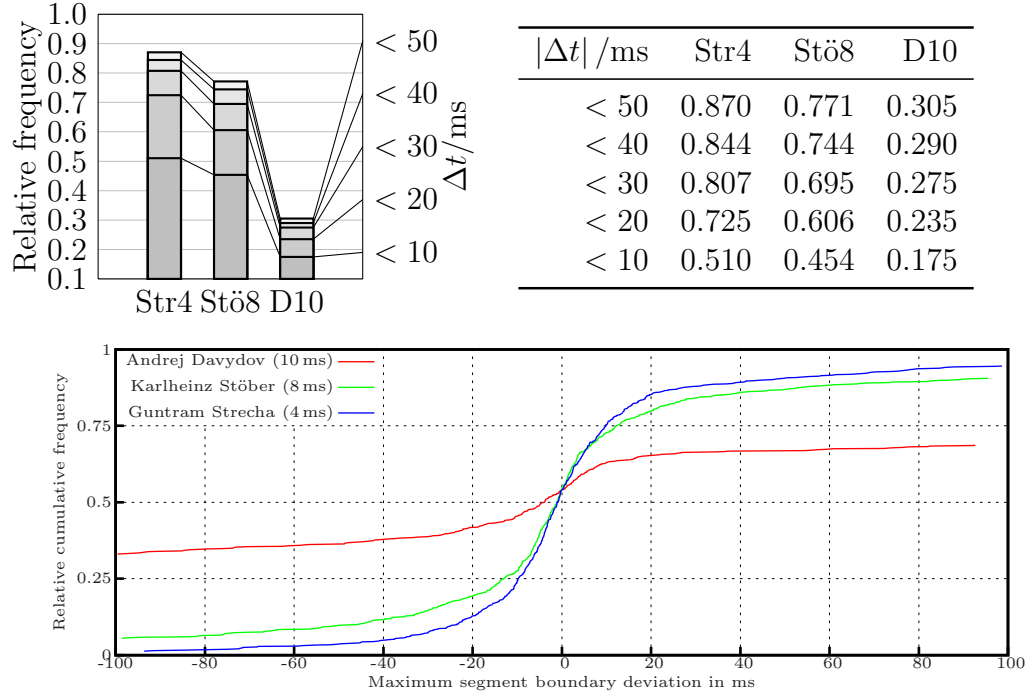


Figure 4.3: Histogram and plot of the boundary errors for a shifted reference: (Str)echa 4 ms, (Stö)ber 8 ms, (D)avydov 10 ms

4. Results

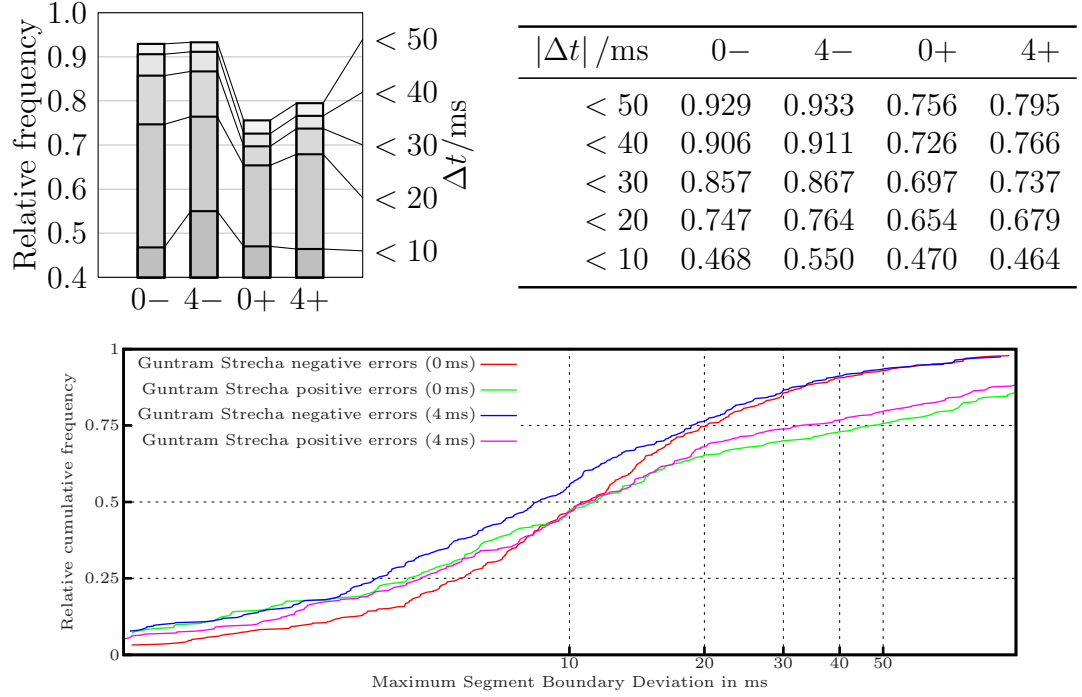


Figure 4.4: Histogram and plot of the positive and negative errors for the aligner of Guntram Strecha for labels shifted by 4 ms

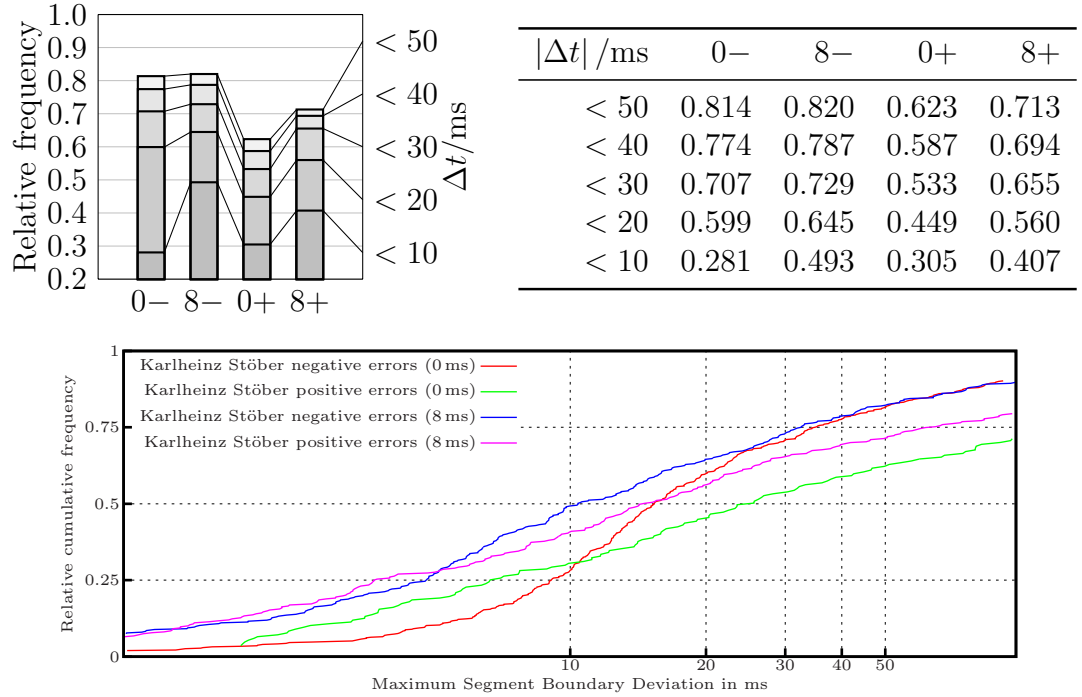


Figure 4.5: Histogram and plot of the positive and negative errors for the aligner of Karlheinz Stöber for labels shifted by 8 ms

To compensate for this difference, the labels for the reference signals are shifted by the required amount and the alignment is performed again. The results can be seen in figure 4.3. The errors improved considerably, especially for the aligner of Karlheinz Stöber. The distribution plot shows that the remaining differences between the aligners are not caused by translational effects anymore.

The figures 4.4 and 4.5 show the effect of the shifted labels on the error distributions divided into negative and positive errors. There seem to be more problems of labels places too late than too early. The delayed labels actually improve these cases slightly, especially for Stöber’s aligner.

Further it is analyzed whether stretching of the synthesized signals improves the alignment result. The duration of the phonemes from the GPC is scaled and the changed sequence is used for synthesis. Table 4.1 shows the median segmentation boundary errors that result for different scaling factors γ and time shifts τ . The aligner of Karlheinz Stöber shows worse results for all scaling factors, the one of Guntram Strecha has slight improvements for a scaling factor of 1.3, only the aligner of Andrej Davydov performs much better with a scaling factor of 1.5. This scaling factor effectively corresponds to the ratio of the mean phoneme durations of the synthetic reference voice *rab* and the voice *kate* (table 4.3) and hints that this aligner makes some assumptions about the minimum and maximum time scale allowed.

An updated comparison of all aligners with the optimized settings can be found in figure 4.6. The improvement for labels within a 10 ms interval from the correct position ranges from 5 % for the aligner of Strecha to about 20 % for the one of Davydov. Because of the high robustness of the results of Strecha’s aligner, it is used for all following experiments.

According to [WK96], the segmentation boundaries of corpora manually labeled by different human experts agree with the accuracy that can be seen in table 4.2. Compared to the automatic results, they show quite a bit of room for improvement for phoneme labeling programs. Maybe other approaches based on Hidden Markov Model (HMM) speech recognition engines could provide more precise results if combined with a DTW based forced alignment.

4.3.2. Transcription Errors

The transcription generated from the speech synthesis system is rated by a comparison with the manually labeled part of the *kate* corpus and 158 manually labeled sentences from the *rob200* corpus. Plosives separated into pause and burst are merged into one phoneme prior to evaluation.

The phonemes from the *kate* corpus have been labeled independently from a synthetic reference, whereas the *rob* data was corrected by a different human expert based on the automatic phonemes generated by the speech synthesis.

The results are shown in table 4.4. The *kate* sentences have more replacements of phonemes caused by the different lexicons used by the manual and the automatic labeling. It also shows a larger number of manual labels that have no

4. Results

Aligner	τ/ms	median($\Delta t/\text{ms}$) for $\gamma =$				
		1.0	1.3	1.5	1.7	2.0
Karlheinz Stöber	0	16.343	17.116	16.460	17.975	18.843
	4	13.479	14.167	13.524	16.592	17.708
	6	12.287	13.481	13.139	15.938	17.755
	8	12.634	12.995	12.827	15.879	17.333
	10	12.379	13.291	13.068	15.791	17.600
Guntram Strecha	0	11.016	11.032	12.369	14.006	14.123
	4	9.819	9.520	10.693	11.892	12.032
	6	9.746	9.395	9.930	11.160	11.552
	8	10.709	9.693	9.749	10.826	11.156
	10	12.188	9.770	9.412	11.177	10.923
Andrej Davydov	0	215.839	137.292	38.064	66.372	147.560
	4	216.275	136.351	36.635	67.773	148.921
	6	216.425	136.351	36.336	68.838	148.921
	8	215.839	135.258	35.219	69.236	151.538
	10	213.914	136.448	36.375	74.359	151.538

Table 4.1: Median segmentation boundary errors for different scaling factors γ and time shifts τ

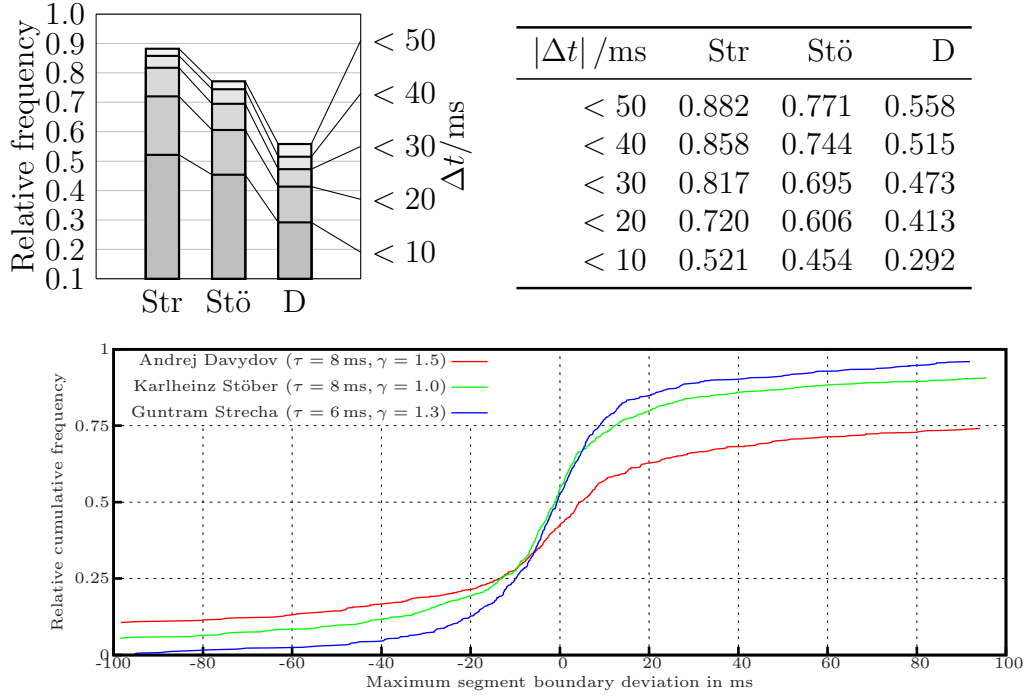


Figure 4.6: Histogram and plot of the segmentation boundary errors for shifted and scaled references: (Str)echa ($\tau = 6 \text{ ms}, \gamma = 1.3$), (Stö)ber ($\tau = 8 \text{ ms}, \gamma = 1.0$), (D)avydov ($\tau = 8 \text{ ms}, \gamma = 1.5$)

4. Results

$\Delta t/\text{ms}$	N
< 5	73 %
< 10	87 %
< 15	93 %
< 20	96 %
< 32	99 %
< 64	100 %

Table 4.2: Number of manually labeled segment boundaries within a certain maximum deviation (from [WK96])

Voice	γ	d_{mean}
<i>rab</i>	1.0	66.15
	1.3	85.89
	1.5	99.23
	1.7	112.32
	2.0	132.14
<i>kate</i>		100.23

Table 4.3: Comparison of the mean phoneme durations d_{mean} of the *kate* voice and the synthetic reference for different scaling factors γ

	<i>kate</i>		<i>rob</i>	
	n	n/N	n	n/N
Only in manual labels (D)	58	7.98 %	78	0.88 %
Only in manual labels, no silence	6	0.82 %	0	0.00 %
Same	599	82.50 %	8676	98.54 %
Replaced (S)	69	9.50 %	50	0.56 %
Not in manual labels (I)	6	0.82 %	367	4.16 %
Not in manual labels, no silence	0	0.00 %	230	2.61 %
Number of manual labels (N)	726		8804	
Total	732		9171	
Accuracy measure (A)	81.68 %		94.37 %	

Table 4.4: Segmentation errors

automatic equivalent in contrast to the *rob* voice that features more phonemes that are only present in the automatic segmentation. These characteristics of the phonetic labeling despite the use of the same synthetic reference voice and lexicon may be caused either by different speaking styles or by peculiarities of the experts' labeling.

The difference in accuracy of about 13 % is mainly caused by the different lexicons that were used for the labeling of the *kate* voice. Most of these errors seem to be vague cases that have no definite correct labeling. The *rob* results give a good impression of the required manual work by a human expert necessary for correction. It is advantageous to use the same dictionary for all tasks concerning a given corpus to avoid differences caused by vague labeling.

Label	<i>rob200</i>	Recognition in %			<i>kate</i>	Recognition in %		
	#	NB	B BB		#	NB	B BB	
NB	1930	92.6	7.4		4040	97.3	2.7	
B BB	414	16.7	83.3		2773	47.8	52.2	
		NB	B	BB		NB	B	BB
NB	1930	92.6	7.2	0.2	4040	97.3	2.7	0.0
B	231	27.7	64.1	8.2	1458	83.1	16.9	0.0
BB	183	2.7	30.0	67.2	1315	8.6	2.8	88.6

Figure 4.7: Confusion matrix and recognition rates for Festival phrase break estimation

4.4. Prosodic Annotation

The training data consists of 158 sentences with 2344 words containing 3596 syllables from the *rob200* corpus. Because the training and evaluation data consists of well planned read speech, no accents have been marked as *emphatic* and only one stress level is included in the following evaluations.

The *kate* corpus is employed to evaluate the algorithms for a different kind of break and stress annotation. Because of the very short utterances in this corpus, the marked phrases are also much shorter and contain exactly one accent per phrase.

4.4.1. Rule-based Break Estimation

The probabilistic model for phrase segmentation of synthesized utterances that is used in Festival for the UK English voices is compared with the manually labeled phrase breaks (figure 4.7). It can be seen that the Festival algorithm can predict phrase breaks for the *rob200* corpus with about 85 % precision if the level of phrase break does not matter. With a distinction between intermediate and final break, the accuracy drops to about 65 %.

The agreement of the automatic annotation with the manual labels from the *kate* corpus is much worse. The phrase break annotation from Festival creates larger phrases than in the corpus, resulting in a lot of intermediate breaks not recognized. The precision is about 50 % for merged phrase breaks and splits unevenly between the classes if intermediate and final breaks are distinguished.

For longer utterances, there is a surprisingly high correlation between the breaks marked with the probabilistic Festival tagger and the manual labels. Additionally, the resulting breaks are at linguistically correct positions that should not generate prosody that will be sensed as incorrect when synthesized.

4.4.2. Derivation from Fujisaki Parameters

In contrast to the Festival probabilistic model, Fujisaki parameters can be used to predict both phrase breaks and word accents. The parameters

- **Limit:** Threshold relative to the phrase maximum for the accent score
- **Split:** accents are split between words, otherwise only the word which contains most of the accents area is used.
- **Relative:** Accents are rated relative to the word length, i. e. longer words get lower scores.

are calculated by EO with

- Initialization: random values
- Parent selection: selection by fitness
- Recombination: average of parents
- Mutation: random for the boolean, Gaussian-distributed for the floating point parameters
- Fitness evaluation: single objective
- Survivor selection: kill worst
- Strategy: $(\mu + \lambda)$

For simplicity a single objective optimization of the overall recognition rate is used. Figures 4.8 and 4.9 show the optimized recognition rates of manually labeled word accents and phrase breaks for the *rob200* and *kate* corpora, respectively.

Similar to the rule-based phrase break estimation, the results for the *kate* corpus are worse. Nevertheless, the achieved recognition rates are comparable to the ones of the Festival method for both corpora.

The stress prediction works slightly better for the *kate* corpus with about 70 % recognition rate and about 65 % for the *rob* corpus. Compared to human labelers that agree with about 87 % on the presence of an accent [GRB⁺96], these results are promising.

A more sophisticated processing of the Fujisaki phrase commands may improve the phrase break estimation. The current method creates a high amount of predicted breaks that are just one word before or after the manually marked break. In combination with the probabilistic tagger, some of these cases could be corrected if information about break probability would be included in the algorithm.

Similarly, the stress detection could be modified to ignore certain POS classes such as pronouns etc. that are very unlikely to be accented. Furthermore, detailed analysis of the position and size of multi-word Fujisaki accent commands and the sensed stress position by a human listener may provide further possibilities for optimization.

4. Results

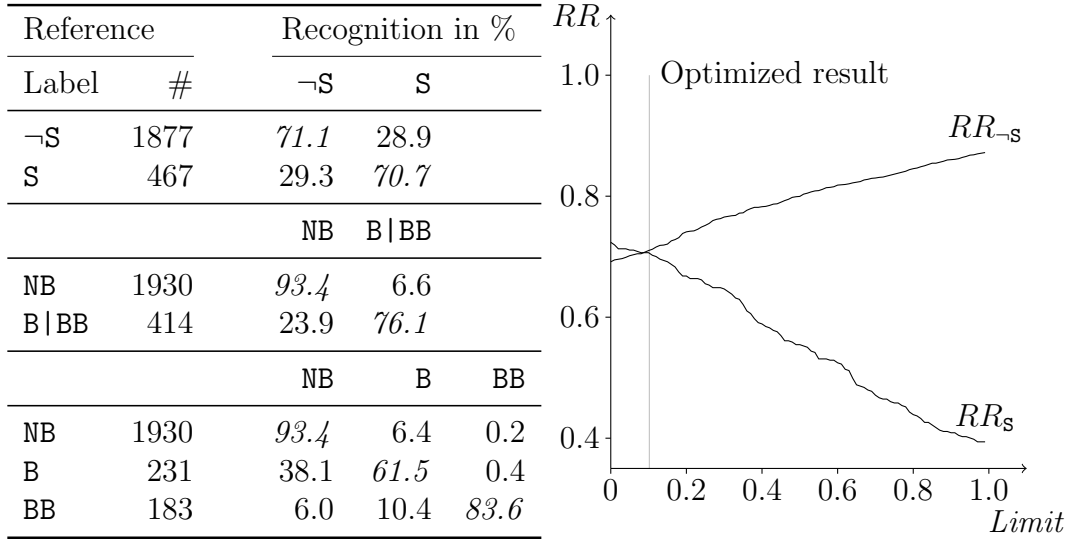


Figure 4.8: Confusion matrix and recognition rates for stress and phrase break estimation with `Split=false`, `Relative=false`, `Limit=0.102` for the *rob200* corpus

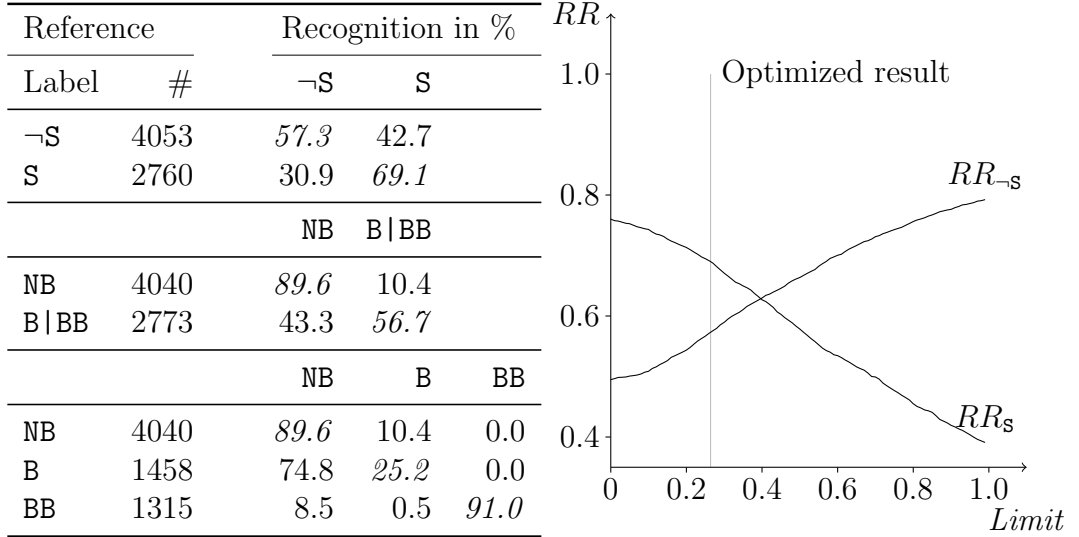


Figure 4.9: Confusion matrix and recognition rates for stress and phrase break estimation with `Split=false`, `Relative=false`, `Limit=0.264` for the *kate* corpus

4.4.3. Neural Network Optimization

The training is done with the previously described parameter set and the following settings [Hof04]:

- Backpropagation learning algorithm
- 25 % combined validation and test set
- Fully connected two layer MFN without shortcut connections
- Training patterns grouped by output field value and duplicated as needed to get groups of equal strength

The tables D.7 to D.9 in appendix D.5 show the results for three different training setups and various network structures with the *rob200* and *kate* corpora. The context size in these tables represents the number of previous and following words that are additionally used as input for the ANN. Additionally, the appendix contains some tables which compare the overall recognition rates for word accents and phrase breaks for the different speakers and network structures.

In the following, an extensive analysis of the phrase break annotations performed by the various ANNs is carried out. Similar results have been obtained for the word stress levels assigned by the network, the necessary data can be found in appendix D.5.

Analysis goals Different network structures and training scenarios have been evaluated to determine the performance of the described approach for phrase break annotations. The analysis concentrates on the following questions:

- How well can phrase breaks be recognized? Does the result improve if the phrase break categories for intermediate and final breaks are combined into one class?
- Is the approach suited for predictive use, i. e. can the labeling be inferred by a network that is only trained on a small part of the whole corpus?
- Are the results of a network independent from the specific database and speaker used for training?

Phrase break recognition The resulting recognition rates for the *rob200* corpus with a 25 % test set of all patterns and separate as well as merged intermediate and major break categories can be seen in table 4.5 and 4.6. The columns RR_{NB} , $RR_{B|BB}$, RR_B and RR_{BB} contain the recognition rates for words followed by no break, any break, an intermediate break or a major phrase boundary, respectively. RR denotes the overall recognition rate independent from the break category and \overline{RR} the arithmetic mean of all recognition rates for the single classes.

Two network structures with different number of neurons N_1 and N_2 in the two hidden layers and various context sizes C are compared. Because of the

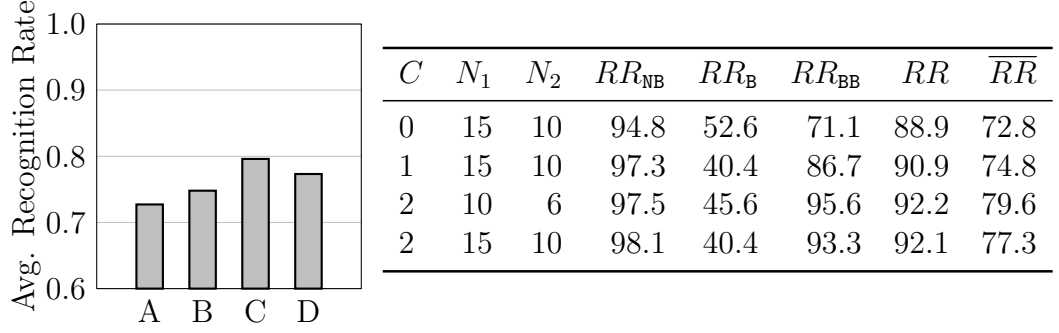


Table 4.5: Recognition rates with 25 % test set and separate break classes (C : context size, N_1 , N_2 : number of neurons in the hidden layers, columns $\{A, B, C, D\}$ in the diagram correspond to lines in the table)

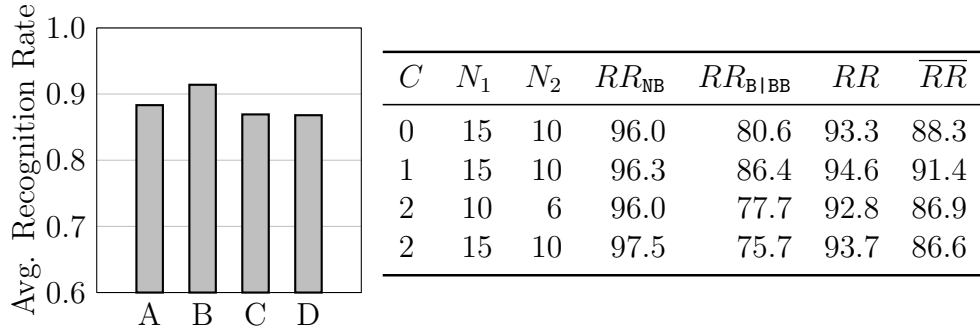


Table 4.6: Recognition rates with 25 % test set and merged intermediate and major break classes (C : context size, N_1 , N_2 : number of neurons in the hidden layers, columns $\{A, B, C, D\}$ in the diagram correspond to lines in the table)

varying recognition results of different classes for one network, \overline{RR} is always lower than RR . Improved context knowledge as well as an increased number of neurons generally leads to better recognition rates. The largest network with ± 2 words context size seems to generalize worse, fitting more of the noise in the data set than the smaller networks. The overall recognition rate for merged categories is about 2 %, the average class recognition rate about 12 % better than with separate classes.

Phrase break prediction Table 4.7 compares the results of the training with a 25 % test set to one with a test set of 75 % of all patterns, thus modeling a partially hand-labeled corpus where the rest should be annotated automatically. Interestingly, the results are only slightly worse for the 75 % test set. The network seems to be able to derive all the specific prosodic features influencing the phrase segmentation for a given speaker from a small part of the whole corpus (25 % corresponds to about 40 sentences).

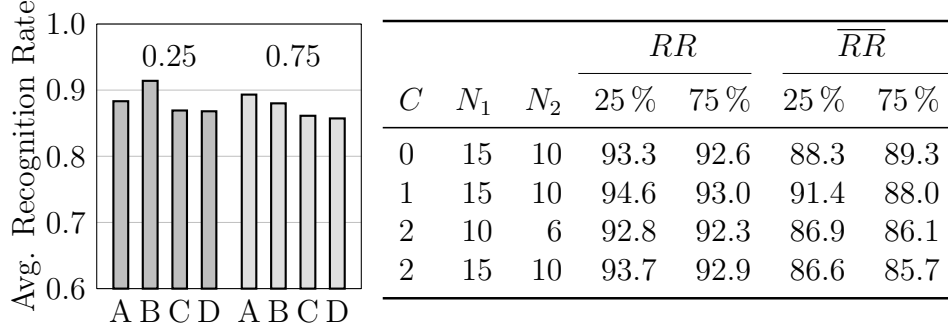


Table 4.7: Recognition rates with 25 % and 75 % test sets and merged intermediate and major break classes (C : context size, N_1 , N_2 : number of neurons in the hidden layers, columns $\{A, B, C, D\}$ in the diagram correspond to lines in the table)

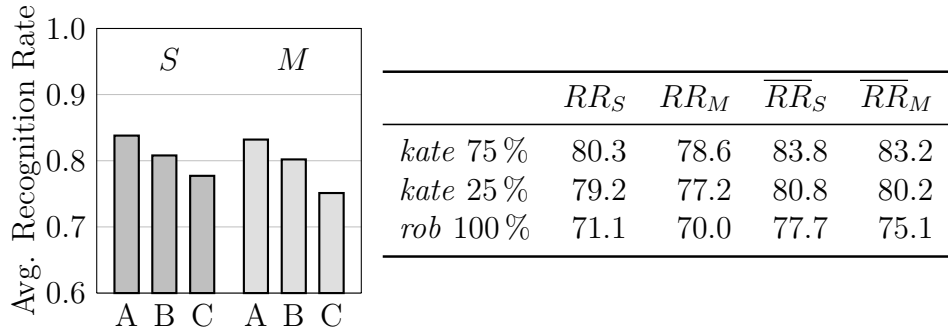


Table 4.8: Recognition rates of the *kate* corpus for a neural network with 15 and 10 neurons in the hidden layers and a context size of ± 2 words, shown are the used training sets for split and merged break classes (the test patterns were always different from the training patterns, columns $\{A, B, C, D\}$ in the diagram correspond to lines in the table)

Speaker independence To test the speaker independence of a so trained network, a totally different database is used for comparison. Table 4.8 shows a summary of the results for the *rob* and *kate* corpora. The average recognition rate for a network that is trained on the *rob* and used for the *kate* corpus is about 3 % to 6 % worse than the one trained on the *kate* corpus itself.

Overall Performance The results of the feature prediction with ANNs make them a good solution for tasks where a part of the corpus is manually annotated and can be used for training. The provided input features for the network should be analysed for their influence on the performance of the ANN and complemented with additional features if necessary.

5. Discussion and Conclusions

This thesis developed an extensible framework for the automatic annotation of speech corpora. It integrates a multitude of external tools already available such as synthesis engines, phoneme aligners and annotation programs.

It has been successfully used for the annotation of recorded speech in the TC-STAR project and has proven suitable for the processing of large amounts of data that have to be extensively annotated.

The presented framework provides modules for the generation of various phonetic, prosodic and linguistic features solely from a recorded audio signal and the transcribed speaker prompt. Several improved or new algorithms for pitchmark derivation from EGG signals, pitchmark correction and word stress and phrase break prediction have been implemented and evaluated with manual reference data for UK English from different corpora.

Several phoneme aligners based on DTW have been used and evaluated for the automatic phoneme alignment. A combination with HMM-based speech recognition may improve the precision of the marked boundaries.

The pitchmark extraction from EGG signals and the correction with an additional pitchmark track derived from time-domain features results in the detection of reliable pitch periods suitable for synthesis. A further analysis of the EGG signal could provide means to detect isolated GCIs that are not relevant to speech synthesis or pitch determination and could therefore be omitted from the merging process as well as additional error measures that allow the correction of erroneously detected pitchmarks. Because the strength of voicing for a glottal oscillation cycle is available after the pitchmark extraction, laryngealization can be easily detected and analyzed.

The implemented prosodic annotation modules for phrase break and word stress detection provide a good estimation of these features in the recorded speech. The still existing gap between the labeling precision of human experts and the presented automatic algorithms may be further closed by various approaches that have been outlined in the corresponding sections. Because no linguistic information could be obtained from the used LaipTTS synthesis, many of the developed annotation modules do not work yet for German. An integration of the DRESS speech synthesis system could improve on this situation.

Although the framework makes it possible to automatically process a complete corpus without manual interaction, it still requires the expert for quality control and parameter tuning. It would be interesting to increase the robustness of the implemented algorithms by a more refined automatic parameter detection, e. g. for minimum and maximum f_0 values or the adjustment of threshold values.

A. File Formats

A.1. Utterance File Formats

Various file formats are employed to represent data used for speech processing, e.g. segmental phoneme information, base frequency contour and pitchmark positions. This section presents several of the more common formats that are needed for data storage or to communicate with the various external tools of the framework.

A.1.1. Festival Utterance Format

The Festival utterance format is based on a non-mixed content model. The data itself and the structural relations between them are stored separately in a file.

Each ASCII utterance file starts with a header (lines 1 to 4 in listing A.1) and a line which describes the utterance's features.

The next section **Stream_Items** (lines 6 to 11) stores the actual data objects. An object identifier is followed a semicolon-separated list of features or () if no features are available.

The **Relations** section (lines 12 to 19) provides the structuring Festival relations. For each relation, multiple lines describe the data objects that make up the hierarchical structure. They consist of six numbers, denoting the item, the associated data object, the parent, daughter, next and previous items respectively. Missing links in the last four numbers are replaced by zero.

A.1.2. XML Utterance Format

Additionally, a self-descriptive Extensible Markup Language (XML) utterance format is supported. It can hold the same information as the Festival format and provides type-safe storage that can be processed by the many XML tools available.

The format features the same separation between content and structure as the Festival format. Because of the hierarchical nature, XML is well suited to describe the utterance structure in itself without the need to reference parent, daughter, previous and next items explicitly. The XML Document Type Definition (DTD) for the format can be seen in listing A.2, an example that describes the same utterance as in listing A.1 is shown in listing A.3.

```

EST_File utterance
DataType ascii
version 2
EST_Header_End
5 Features ()
Stream_Items
1 name Julia ; pos nnp ; pbreak NB ;
2 name syl ; stress 1 ;
3 name _dZ ; end 0.052 ;
10 4 name dZ ; end 0.116384 ;
End_of_Stream_Items
Relations
Relation SylStructure ; ()
3 3 2 0 4 0
15 4 4 0 0 0 3
2 2 1 3 0 0
1 1 0 2 0 0
End_of_Relation
End_of_Relations
20 End_of_Utterance

```

Listing A.1: Festival utterance example

```

<!ELEMENT utterance (features,items,relations)>
<!ELEMENT features (feature*)>
<!ELEMENT feature (#PCDATA)>
<!ELEMENT items (contents*)>
<!ELEMENT contents (features)>
<!ELEMENT relations (relation*)>
<!ELEMENT relation (features,item*)>
<!ELEMENT item (item*)>

<!--
<!ATTLIST feature
  type (int|string|float) #REQUIRED
  name CDATA #REQUIRED
-->
<!--
<!ATTLIST contents id ID #REQUIRED>
<!ATTLIST relation name CDATA #REQUIRED>
<!--
<!ATTLIST item contents IDREF #REQUIRED>

```

Listing A.2: DTD for the XML utterance format

```

<utterance>
  <features />
  <items>
    <contents id="_1">
      <features>
        <feature type="string" name="name">Julia</feature>
        <feature type="string" name="pos">nnp</feature>
        <feature type="string" name="pbreak">NB</feature>
      </features>
    </contents>
    <contents id="_2">
      <features>
        <feature type="string" name="name">syl</feature>
        <feature type="int" name="stress">1</feature>
      </features>
    </contents>
    <contents id="_3">
      <features>
        <feature type="string" name="name">_dZ</feature>
        <feature type="float" name="end">0.052</feature>
      </features>
    </contents>
    <contents id="_4">
      <features>
        <feature type="string" name="name">dZ</feature>
        <feature type="float" name="end">0.116384</feature>
      </features>
    </contents>
  </items>
  <relations>
    <relation name="SylStructure">
      <features />
      <item contents="_1">
        <item contents="_2">
          <item contents="_3" />
          <item contents="_4" />
        </item>
      </item>
    </relation>
  </relations>
</utterance>

```

Listing A.3: XML utterance example

```

_      517.189
_dZ    18.633
dZ     89.409 0 99.8
u:    95:395 50 110.0 100 105.0

```

Listing A.4: Mbrola label file example

```

hend
  8275 _dZ
  8612 dZ
10004 u:

```

Listing A.5: PhonDat label file example

A.2. Segmental Information

A.2.1. Mbrola File Format

The Mbrola file format is used by the Mbrola synthesizer to specify phonemes, durations and a piecewise linear pitch contour. Each line consists of the segment name, the phoneme duration in milliseconds and pitch contour points separated by whitespace. Each pitch contour point is specified as a tuple of the relative position in the phoneme in percent and the pitch value at this position in Hertz (listing A.4).

A.2.2. PhoneDat File Format

The German PhonDat project defined several different file formats to store segmentation results [Sch04]. A header that stores meta data about the segmentation file is followed by `hend` on a single line and the segmentation data. Each entry consists of the first sample position and the name of each segment delimited by whitespace (listing A.5). Because of its dependency on sample units the positions can't be interpreted without an external signal file or explicitly specified sample rate.

A.2.3. Xlabel File format

The *Xwaves* package by *Entropic Inc.* uses a special label file format for its *xlabel* tool. The format is described in detail in [Ent98].

An *Xlabel* file consists of a header section followed by the label entries. The header consists of keyword-value pairs delimited by spaces or tabs and ends with a line containing only a single `#` sign. See [Ent98] for a list of all possible header

```

signal 0001.manual
type 0
color 121
comment created using WaveSurfer wo nov 30 08:45:36 2005
font -misc-*-*-15-*-*-*-*
separator ;
nfields 1
#
    0.000000  121  _
    0.517189  121  _dZ
    0.535822  121  dZ
    0.625231  121  u:

```

Listing A.6: XWaves label file example

```

0.000000 0.517189  _
0.517189 0.535822  _dZ
0.535822 0.625231  dZ

```

Listing A.7: Wavesurfer label file example

items.

The label file body consists of lines with three values, separated by blanks. The first one specifies the tag time in seconds, the second the color used for displaying the label and the third the label text. Listing A.6 shows an example of such a label file. Because the label file contains only labeled time tags, segmental information can be stored in two slightly different ways: the tag time can either encode the end of a segment (Wavesurfer) or the beginning (most project specifications).

A.2.4. Wavesurfer File Format

This is the native transcription format of the Wavesurfer program developed by Kare Sjölander at the Centre for Speech Technology (CTT) of the Royal Institute of Technology (KTH) in Stockholm, Sweden [SB00]. Each line consists of three fields that determine the segment boundaries in seconds and the displayed label name. An example can be seen in listing A.7.

A.2.5. .Seg File Format

This kind of segment file consists of only one line with interleaved label names and boundary positions. The first label name is always set to **a**. For the bound-

```
a 8275 _dZ 8612 dZ 10004
```

Listing A.8: .Seg label file example

Line	Contents
1	PCM file name
2 to 6	reserved
7	number of f0 values
8	number of Fujisaki phrase commands I
9	number of Fujisaki accent commands J
10	base frequency value
11	time step used for f0 analysis
12 to 20	reserved
21	1st phrase command
$21 + I - 1$	I th phrase command
$21 + I$	1st accent command
$21 + I + J - 1$	J th accent command

Table A.1: PAC file format

ary positions sample units are used, making it impossible to interpret such a file without an associated signal file or sample rate. See [Kru03] for a more thorough description of the format. Listing A.8 shows an example.

A.3. Other Formats

A.3.1. .PAC File Format

PAC files are used to store Fujisaki parameters. The basic structure can be seen in table A.1. Phrase and accent command lines consist of four values delimited by whitespace. For a phrase command, the first value contains the onset time in seconds, the third the phrase command amplitude and the fourth the coefficient alpha, the second field is unused. For an accent command, the first two values are onset and offset time followed by the accent command amplitude and the coefficient beta. An example is shown in listing A.9, multiple reserved lines have been collapsed into one empty line.

A.3.2. PM Files

This is a binary format to represent pitchmark positions [Kru03]. A file consists of an eight byte header and four byte period marker chunks (table A.2). The

```

0001.wav

200
1
2
0.000000
0.010000

0.343412 0.000000 1.906760 0.750042
0.402377 0.718791 0.476134 18.820200
1.046570 1.470500 0.341375 23.212700

```

Listing A.9: PAC Fujisaki parameter file example (multiple empty lines collapsed into one line)

Header (8 bytes)		
0x0000	u16	file identification (set to 0x0002, used to detect byte order)
0x0002	u16	sampling frequency
0x0004	s32	reserved
Period markers (4 bytes)		
0x0000	u16	period length since last marker
0x0002	s8	voicing (0 means unvoiced, 1 voiced)
0x0003	s8	unused (0) or correction c

Table A.2: PM file format

sampling frequency value in the header can't be used for sample rates larger than 65535 Hz.

In [Eng03] an extension to the format to reduce quantization errors is introduced. The unused byte in the marker chunks is used to correct the stored period lengths between sample positions. The new pitchmark position is calculated from period lengths ΔT_j and the correction value c with

$$T_{i,\text{refined}} = \sum_{j=0}^{i-1} \Delta T_j + \frac{c}{256}$$

A.3.3. Pitchmark Files

Text based pitchmark files basically consist of one pitchmark per line, given by its position in seconds (listing A.11 right). Unvoiced pitchmarks are marked with negative positions.

```

0x0000  02 00 80 3E  # id, sample rate = 16 kHz
0x0004  00 00 00 00  # reserved
0x0008  4F 00 00 00  -79.00
0x000C  22 00 00 00  -113.00
0x0010  73 00 01 7A  228.48
0x0014  86 00 01 A3  361.64
0x0018  7E 00 01 66  488.40
0x001C  80 00 01 AF  615.68

```

Listing A.10: PM pitchmark file example

```

# 0.3
 79.00 0                                -79.00
113.00 0                                -113.00
228.48 1  0.121
361.64 1  0.767                        361.64
488.40 1 -1.000 0.212                  488.40
615.68 1  0.754                        615.68

```

Listing A.11: Comparison of a multicolumn pitchmark file including confidence scores and a threshold with a file using negative values for unvoiced pitchmarks

An extended variant is used to store more information in additional columns (listing A.11 left). The first line contains a comment with the threshold to be applied automatically to the confidence scores in the file. Values other than zero in the second column mark voiced pitchmarks, otherwise they are regarded as unvoiced. The third column holds the confidence score, with zero for disabled and -1 for unconditionally enabled pitchmarks. If the confidence value has been changed manually, the forth column contains the original value before the modification.

A.3.4. Entropic Fundamental Frequency Contour Files

The fundamental frequency files used by Entropic's tools are simple text files consisting of one pitch value per line with a frame step of 10 ms [Ent96]. Each line contains four fields that hold the fundamental frequency value in Hertz, the probability of voicing in the range of 0 to 1, the RMS value of a 30 ms Hann window and a peak normalized cross-correlation value for time-domain based algorithms (listing A.12).

```

0.0 0 1387.9 0
0.0 0 1965.2 0
237.7 1 2725.6 0
241.4 1 3216.2 0
247.9 1 3503.2 0

```

Listing A.12: Entropic fundamental frequency contour file

```

| - { z - | - r I - g A: d z - | - n aI - t r @ - dZ @ n - | -
l e - v @ l z - <pau> | - w i: - | - w U d - | - n i: d - | - r I
- l aI - b @ l - | - s t @ - t I - s t @ k s - <pau> | - { n -
| - d eI - t @ - | - f r Q m - | - D @ - | - v e@ - r I@ s - | -
m e m - b @ - | - s t eI t s - |

```

Listing A.13: TC-STAR phonetic annotation file

A.3.5. TC-STAR Phonetic Format

The file format for the phonetic transcription of the TC-STAR project includes word and syllable boundaries as well as pauses. Phonemes are delimited by whitespace, syllables by - and words by |. The outer syllable and word delimiters are not omitted, pauses are attached to the preceding word and marked with <PAU> (listing A.13).

A.3.6. TC-STAR Prosodic Format

This file format is used to store prosodic information for the TC-STAR project. It contains the plain text utterance enriched with markup for pitch accent and phrase structure. A word followed by # is marked as normally, one followed by ## as emphatically accented. Intermediate intonational phrases are delimited by , whereas major phrase breaks are marked with <BB>. An example can be seen in listing A.14.

```

As regards <b>
nitrogen# levels, <BB>
we would need reliable statistics# <b>
and data# from the various Member States#. <BB>

```

Listing A.14: TC-STAR prosodic annotation file

B. How to . . .

The following sections show some examples for the usage and extension of the framework.

B.1. . . . Annotate a Corpus

B.1.1. General Notes

The executable Java classes (programs) are called with

```
sh> java [Java params] mh21.progs.ClassName [program params]
```

The parameters of the Java interpreter can be obtained with `java -h`. Most of them are not necessary for the execution of the framework, although it may be necessary to increase the heap memory available to Java with e.g. `-Xmx512M`.

General parameters that are available for all programs are:

```
--help
    shows command line parameter documentation
--verbose
    increases the verbosity of messages
--quiet
    decreases the verbosity of messages
--dump
    shows the attributes of all selected modules and performs rudimen-
    tary tests of external programs
```

Parameters written in upper case reference constants that are described in appendix F. The allowed values can also be found in the command line documentation available with `--help`.

B.1.2. Wave File Preparation

The recorded continuous wave files have to be cut into utterances to be processed. Open the audio track in Wavesurfer and create a transcription file that marks utterance segments with e.g. U and silence with `_`. Move each wave file into a separate directory and execute

```
sh> java mh21.progs.RunCutter \
    --input input.lab --input-format FORMAT --wave input.wav
```

for each track with the following command line parameters:

```
-i, --input FILE
    manually labeled utterance segmentation for the continuous wave
    file
-I, --input-format FORMAT
    file format for the labels, see section F.3.2
-w, --wave FILE
    continuous wave file
-s, --silence
    creates additional wave files for the segments marked as silence
```

to generate numbered output wave files in the current directory for each marked segment. This should work independent of the encoding used for the wave file and provide track segments that match each other exactly.

B.1.3. Text Preparation

The text files used for the synthesis need to be syntactically correct for the target language. They should be provided in a supported character encoding (like UTF-8 or Latin 1/ISO-8559-1) and manually corrected for the most common mistakes. Listing B.1 shows some commands that can be used to detect and correct problematic expressions and characters:

- DOS/Windows CR/LF line endings should be replaced by Unix-style LFs.
- Line numbers at the beginning must be removed.
- Excessive whitespace before “,”, “.”, “;”, “:”, “!”, “?” and “)” or after “(” should be deleted.
- Umlauts like “ä”, “æ” or “ß” should be replaced by their ASCII “a”, “ae” or “ss” equivalents depending on the dictionary and synthesis engine used.
- Spacing of single and double quotation marks should be corrected to not include any space between an opening or closing quotation mark and the following or preceding word, respectively.
- Whitespace around apostrophes like in “boy ’ s” should be removed.

B.1.4. Forced Phoneme Alignment

The forced alignment interface in `mh21.progs.RunPipe` supports the following steps that can be selected with the parameters `--first` and `--last`:

1. `gpc`: A GPC reads a text file and produces a phoneme sequence with additional linguistic information.
2. `synthesizer`: The phoneme sequence is converted into a synthesized speech signal.

```
# Replace preceding spaces
sed -ri 's/ ([.,;?!:)%])/\1/g' *
# Delete line numbers (surrounded by whitespace) at the
# beginning of the line
sed -ri 's/^\s*[0-9]+\s*//' *
# Replace following spaces
sed -ri 's/\( /\)/g' *
# Replace DOS/Windows line endings
sed -ri 's/\r//g' *
# Search for files that have unusual characters in them
grep --color "[^A-Za-z0-9,.;\!?:]" *
# Search again, more tolerant
grep --color "[^A-Za-z0-9,.;\!?:'\\"%/ ]" *
```

Listing B.1: Commands to detect and correct common text problems

Module	Input parameters	Output parameters
gpc	--synth-phones	--synth-wave
synthesizer	--synth-phones	--synth-wave
processor	--synth-phones	--reference-phones
	--synth-wave	--reference-wave
aligner	--reference-phones	--user-phones
	--reference-wave	
	--user-wave	

Table B.1: Input and output parameters for the modules of the forced alignment modules

3. **processor**: The phoneme sequence and synthesized speech signal are modified to become the reference for the alignment.
4. **aligner**: The reference phoneme sequence and signal are aligned to the natural speech signal, creating the aligned phoneme sequence.

The required parameters for each module can be seen in table B.1. If several steps are executed at once, temporary files will be used for intermediate results that are generated in-between and that do not have a name specified on the command line. Most of these command line parameters can be complemented by additional options to specify file format and phone set.

To demonstrate the use of the alignment interface, the following example illustrates the necessary steps to align an utterance of a speaker of British English with the help of Festival, Mbrola and the DTW aligner of Guntram Strecha.

To get an overview of the available modules, we execute

```
sh> java mh21.progs.RunPipe --help
```

We select a Festival GPC (`festival`), the Mbrola synthesizer (`mbrola`), a processor that splits plosives into pause and burst (`plosive`) and the aligner of Guntram Strecha (`strecha`). The call of

```
sh> java mh21.progs.RunPipe --first gpc --last aligner -d \
    --gpc festival \
    --synthesizer mbrola \
    --processor plosive \
    --aligner strecha
```

provides a list of the available parameters for these modules and does a rudimentary test of the required external programs. It also checks for the availability of audio file converters like Sox and Ecasound.

The most common parameters for these modules can be found in table B.2. The selected British English *rab* voice in Festival creates phonemes in the *BritishMRPA* phone set whereas the Mbrola voice *en1* requires phonemes encoded in *BritishMbrola*. A look at the help page of the program shows that this conversation is included in the list of known phone set transformations that can be applied automatically.

With input text provided in UTF-8 encoding that should get any remaining accents removed before synthesis, the following command line will invoke the forced alignment with temporary intermediate files and a more verbose output containing a list of all external programs that are called in the process. It creates a Festival utterance file with all phonemes in *BritishDress* encoding.

```
sh> java mh21.progs.RunPipe --first gpc --last aligner -v \
    --gpc festival \
    --synthesizer mbrola \
    --processor plosive \
    --aligner strecha \
    --user-text 001.txt --user-text-format utf8-strip \
    --user-wave 001.wav \
    --user-phones 001.est --user-phones-set BritishDress \
    --user-phones-format festival
```

B.1.5. Correcting Linguistic and Phonetic Tracks

The utterance produced by the forced phoneme alignment can now be modified and annotated with various data tracks created by `mh21.progs.RunAnnotators`. The most important parameters are:

```
-i, --input FILE
    input file for the utterance that will be annotated
-o, --output FILE
    output file for the annotated utterance
```

Parameter	Modules	Description
Executable	festival mbrola strecha	Executable to use. If no path is supplied, the PATH environment variable is used to look for the executable.
Voices	festival mbrola	All voices known by the module. Each voice has an associated phone set that defines the encoding used for the generated phonemes.
Voice	festival mbrola	Currently selected voice.
Phonaset	festival mbrola	Phone set associated with the currently selected voice. This attribute is determined automatically depending on the selected voice and can't be set manually.
Converter	plosive strecha	Forces the use of a specific audio file conversion program as needed for this processor. An empty string means automatic selection depending on the known capabilities of the programs.
Plosives	plosive	Comma separated list of plosives that should be split into pause and burst. This should also contain africates like [tʃ] or [pf].
Position	plosive	Relative position to use for the pause-burst boundary on the longest monotonous increase in the signal energy. Can be in the range of 0 to 1 or steepest to take the point of the maximum slope.
ReferenceShift	strecha	Shifts the reference labels by a certain number of ms before the phoneme alignment.
Config	strecha	Pathname of the configuration file to use of the aligner.

Table B.2: The most important attributes of a forced alignment process using Festival, Mbrola, a plosive splitter and the aligner of Guntram Strecha

```
-w, --wave FILE
    wave file for the processing by annotation modules
-a, --annotator ANNOTATOR
    comma-delimited annotation modules, see section F.2.1
-A, --annotator-param ANNOTATOR,KEY=VALUE,...
    module parameters
```

To manually correct the produced phonetic segmentation, the phoneme labels can be exported in a format suitable for Wavesurfer or Xwaves:

```
sh> java mh21.progs.RunAnnotators -v \
    --input 001.est \
    --output 001.lab --output-format wavesurfer
```

Together with an external lexicon to correct the syllable boundaries generated by Festival, the manually modified phoneme boundaries can be merged back into to utterance. Phoneme insertion, deletion and modification will be handled in a graceful manner.

```
sh> java mh21.progs.RunAnnotators -v \
    --annotator label,syllable \
    --annotator-param label,LabelFile=001.man.lab \
    --annotator-param syllable,LocalLexiconFile=lex.txt \
    --input 001.est \
    --output 001.corrected.est
```

B.1.6. Annotation

The most commonly used annotators are shown in figure B.1, the full list of all available annotators with a description of their attributes can be found in appendix F.2.1.

The following example demonstrates the determination of pitchmarks from an EGG signal. The used threshold and the minimum and maximum base frequency values have to be determined by manual inspection of the acoustic and EGG tracks for some utterances by the speaker. Unvoiced pitchmarks are inserted in areas without voiced pitchmarks samples that are longer than a certain multiple of the pitch period:

```
sh> java mh21.progs.RunAnnotators -v \
    --annotator laryn,unvoiced \
    --annotator-param laryn,MaximumF0=200,MinimumF0=50 \
    --annotator-param unvoiced,MaximumF0=200,MinimumF0=50
    --annotator-param unvoiced,Threshold=0.2,MaxGap=4 \
    --input 001.corrected.est \
    --output 001.annotated.est \
    --wave 001.laryn.wav
```

Parameter	Modules	Description
MinimumF0	gcida laryn unvoiced pitch2f0 smoothf0	Minimum base frequency value expected by the module. This value should be determined manually from a sample of the recorded corpus. Lower f0 values than set by this limit may result in spurious inserted pitchmarks caused by noise. If this limit is too low, the insertions of random unvoiced pitchmarks may not work as expected.
MaximumF0	gcida laryn unvoiced pitch2f0 smoothf0	Maximum base frequency value expected by the module. This value should be determined manually from a sample of the recorded corpus. Higher f0 values than set by this limit may result in “holes” in the pitchmark sequence, whereas a limit too high may insert spurious pitchmarks because by noise.
InputRelation OutputRelation	unvoiced pitch2f0 smooth fuji fuji2f0 fuji2prosody	Input and output relations that are used to read the data to be processed and to store the results afterwards. If the output relation exists beforehand, it will be deleted and recreated right before the results are stored. The output relation can be the same as the input relation.
Threshold	unvoiced laryn	Confidence threshold in the range 0 to 1 to determine reliable pitchmarks from an EGG signal. All pitchmarks with lower confidence will be deleted prior to processing or storage.
LowPass HighPass	laryn	Filters to use for the EGG signal. The low pass filter suppresses slow changing signal parts caused by head or larynx movements. The high pass filter reduces the noise level but degrades the precision of the pitchmark positions.
Insert	unvoiced	Determines whether randomly generated unvoiced pitchmarks are inserted. The minimum and maximum base frequency values are respected.

Table B.3: The most important attributes for pitch and prosodic annotation

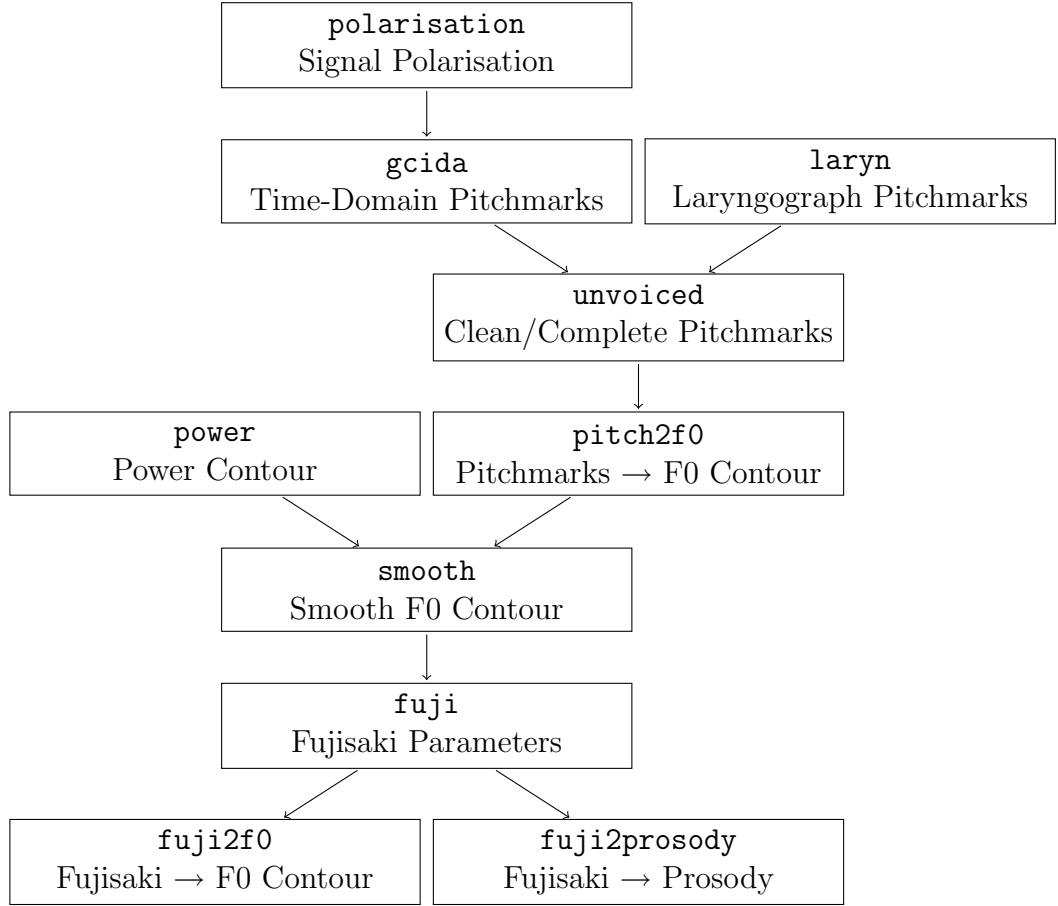


Figure B.1: Relationship between the most used annotators. For a given annotator, one or any of the preceding annotators are required for proper function.

B.1.7. Bulk Processing

To annotate large corpora, scripts to do automated bulk processing are necessary to generate consistent data and to be able to rerun certain annotation steps. The presented example scripts utilize the following directory structure:

corpus/ Main directory for the whole corpus.

java/ All Java classes of the framework, including files required during the execution of the framework like configuration settings or global lexicons.

scripts/ The here described scripts to automate the annotation.

data1/ Data directories for the corpus recordings.

track01/ Audio files for the first track and generated annotations.

track02/ Additional tracks.

...

...

```

#!/bin/bash
. $(dirname $([ -h $0 ] && readlink $0 || echo $0))/jp_default.sh
jp_init "$@"

5 for i in "${jp_files[@]"; do
    jp_file i

    echo $num
    java -Xmx512M mh21.progs.RunPipe -v \
10     --first gpc --last aligner \
        --gpc festival --synthesizer mbrola \
        --processor plosive --aligner strecha \
        --user-text "$num".txt --user-text-format utf8-strip \
        --synth-wave "$num".synth.wav \
15     --reference-phones "$num".synth.pho \
        --reference-phones-set BritishMbrola \
        --user-wave "$num".wav \
        --user-phones "$num".est --user-phones-format festival \
        --user-phones-set BritishDress
20 done

jp_done

```

Listing B.2: Example script for forced phoneme alignment (`jp_aligner.sh`)

To assure consistency, all scripts are only stored once in the designated script directory. For each recording, the directory of the first track should have symbolic links to the scripts created with e.g.

```
track01/wav_01> ln -s ../../scripts/*.sh .
```

and will be used for storage of all annotation results.

A simple example for a script to automate forced phoneme alignment can be seen in listing B.2. It uses some helper functions from listing B.3 that ease the command line parsing and allow to change to the Java root directory before the processing. All similarly structured scripts can be called without parameters and will process all files with a given extension (line 23) or may be limited to stripped filenames given on the command line (line 29). After a call to `jp_file` (line 6), the variables `short_num`, `num` and `path` can be used to refer to the filename without extension, the complete pathname without extension or only the path of the currently processed file, respectively.

```

#!/bin/bash
jp_stripest() {
    result=${!1}
    for ((i=3;i<=#;++i)); do
5       result=${result%${!i}}
    done
    export $2="$result"
}

10 jp_init() {
    DIR='pwd'
    pushd $FRAMEWORKROOT > /dev/null
    if [ $# -gt 0 ]; then
        jp_files=("$@")
15     # expand relative paths
        for ((i=0;i<${#jp_files[@]};++i)); do
            if [ "${jp_files[$i]:0:1}" != "/" ]; then
                jp_files[$i]="$DIR"/"${jp_files[$i]}"
            fi
20     done
    else
        shopt -s nullglob
        jp_files=("$DIR"/{??,???,????}.txt)
        fi
25 }

# Generates num, short_num and path variables from $1
jp_file() {
    jp_stripest $1 num .wav .est .txt .lab .pho .pm
30     short_num=${num##*/}
    path=${num%/*}
}

jp_done() {
35     popd > /dev/null
}

```

Listing B.3: Helper functions to aid in bulk processing of large corpora (jp_default.sh)

```

public final class PhoneTransformer
{
    /* ... */
    /** Convert Dutch SAMPA to Dutch Siemens. */
    public static final PhoneTransformer DUTCH_SAMPA_TO_SIEMENS
        = new PhoneTransformer (Name.DUTCH_SAMPA, Name.DUTCH_SCT,
            "}", "Y", "i", "i:", "y", "y:", "u", "u:", "2:", "|:",
            "Ei", "K", "9y", "L", "Au", "M", "O:", "<");

    public enum Name implements DescriptiveEnum<Name> {
        /** Dutch Siemens encoding */
        DUTCH_SCT ("DutchSiemens", "Dutch Siemens CT"),
        /** Dutch SAMPA encoding */
        DUTCH_SAMPA ("DutchSampa", "Dutch SAMPA"),
        /* ... */
    }
    /* ... */
}

```

Listing B.4: Two example phone sets for Dutch which can be converted between each other

B.2. ... Add a New Phone Set

The following two steps are necessary to define new phone sets that can be automatically converted between each other (listing B.4):

1. The phone set names have to be registered as additional enum values in `mh21.prosodic.PhoneTransformer.Name`. A short description must be provided that will be used to generate the help for command line arguments (lines 12 and 14).
2. A transformation table holds the phonemes that differ between the two phone sets. A static instance of `PhoneTransformer` takes the enum constants of the source and target phone sets together with the phoneme tuples to be converted as arguments (line 5).

Afterwards, the `PhoneTransformer` class will enable conversations between all phone sets that have transformations defined between them. This includes reverse operation where source and target phone set are exchanged as well as chained conversations that require multiple transformation tables.

B.3. How to Add a New GPC, Synthesis, Processor, Aligner or Annotator Module

Several steps are necessary to add a new class as a module for forced alignment or annotation which has attributes that are accessible from the command line. The following example demonstrates the glue code needed to integrate an external program *synth* as a GPC module:

1. Each module resides in a separate class that should be created in the appropriate package (e.g. `mh21.prosodic.gpc` for GPC modules).
2. The newly created class needs to implement the corresponding interface for the module type (`Gpc`, `Synthesizer`, `Processor`, `Aligner` or `Annotator`):

```
public class SynthGpc implements Gpc { /* ... */ }
```

3. The `testExecutable` method provides feedback of the status of the external program to the user. It is called before the actual processing takes place and should provide some meaningful program output such as version number or description on success. If the program is not available or non-functional, an exception of type `ExecutionFailedException` is thrown. An example implementation can be seen in listing B.5. The instance variable `executable` stores the pathname of the external program.
4. For GPC modules, the `convert(Text)` method does all the work. The source code of the already existing modules gives a good impression of the details necessary for implementation.
5. The `Attributable` subinterface which provides the command line access functionality for the module's parameters can be either implemented by deriving from `ReflectiveAttributable`

```
public class SynthGpc extends ReflectiveAttributable
    implements Gpc {
    /* ... */
}
```

or by using composition and delegates for the interface's methods

```
public class SynthGpc implements Gpc {
    private ReflectiveAttributable attributable =
        new ReflectiveAttributable (this);

    public String getAttribute (String key)
        throws IllegalArgumentException
    {
        return this.attributable.getAttribute (key);
    }
    /* ... */
}
```

```
private String executable = "synth";

public String testExecutable () throws ExecutionFailedException
{
    List<String> command = new ArrayList<String>();
    command.add (this.executable);
    Processbuilder builder = new ProcessBuilder (command);
    Process process = null;
    try {
        Globals.verbose ("Starting Synth: " + command);
        process = builder.start ();
    } catch (IOException e) {
        throw new ExecutionFailedException (
            "Was not able to start Synth", e);
    }
    BufferedReader reader = new BufferedReader (
        new InputStreamReader (process.getInputStream ()));
    String line = null;
    try {
        line = reader.readLine ();
    } catch (IOException e) {
        throw new ExecutionFailedException (
            "Was not able to read output", e);
    }
    return line;
}
```

Listing B.5: Example implementation of the `testExecutable` method

Now it is only necessary to provide appropriate getter and setter methods for each attribute that should be readable and/or writable from the command line and to use the `ReflectiveAttributable.Description` annotation for either the getter or the setter to make the attribute accessible:

```
@Description("pathname of Synth executable")
public String getExecutable ()
{
    return this.executable;
}
public void setExecutable (String executable)
{
    this.executable = executable;
}
```

6. Finally, the module has to be registered with the appropriate enum class:

```
public final class Gpcs
{
    public enum Type {
        SYNTH ("synth", SynthGpc.class, "Synth GPC"),
        /* ... */
    }
    /* ... */
}
```

B.4. How to Add Support for a New File Format

A file format is represented by a class that implements the `PhoneFile` interface. It provides methods to read and write data from a binary stream. A simple pitch file reader could be declared with

```
public class PitchFile implements PhoneFile
```

It needs to implement the `read(InputStream)` method that returns a newly created utterance filled with the read data. An example method that reads newline-delimited doubles into the pitchmark relation could look this:

```
public Utterance read (InputStream reader)
    throws IOException
{
    Utterance utterance = Utterance.create ();
    Relation relation = utterance.createRelation
        (Relation.PITCHMARK);
    BufferedReader buffered = new BufferedReader
        (new InputStreamReader (reader, "UTF-8"));
    Item tail = null;
    String line;
    while ((line = buffered.readLine ()) != null) {
        ItemContents contents = new ItemContents ();
        contents.setObject ("pos", Double.valueOf (line));
        tail = tail == null ? relation.appendItem (contents)
            : tail.appendItem (contents);
    }
    return utterance;
}
```

The `write(OutputStream,Utterance)` method can be implemented to save data in this format. If an operation is not supported for a particular file format, an instance of `ProsodicException` should be thrown:

```
public void write (OutputStream writer, Utterance
    utterance) throws ProsodicException
{
    throw new ProsodicException ("read-only");
}
}
```

To make the file format known to the input and output routines it has to be registered with the `PhoneFiles.Type` enum class:

```
public class PhoneFiles
{
    public enum Type {
        PITCH ("pitch", "", "Pitch data", PitchFile.class),
        /* ... */
    }
    /* ... */
}
```


C. Programs

C.1. Gcida Aligner Fixes

The aligner developed by Toni Engel during his thesis has bugs that prevents it to work with Microsoft wave files. A patch for these problems is shown in listing C.1.

- The file format determination uses a 4-byte buffer (line 8) that is filled with the potential RIFF signature of the input file. The following strcmp call (line 14) will overflow on comparison and inhibit the recognition as a wave file.
- The header processing uses unsigned long values to read fields that have a size of 4 bytes. This does not work for 64-bit systems which have 64-bit longs. The proposed fix uses ints (line 25) that are 32-bit on most architectures.

```
--- gcida.C      2006-02-03 14:39:01.000000000 +0100
+++ gcida.C      2006-02-03 14:41:37.000000000 +0100
@@ -187,13 +187,13 @@

5 // Signal einlesen
  // Auf *.WAV-Format pruefen
-   char riff[4] = {0,0,0,0};
+   char riff[5] = {0,0,0,0,0};
   for (l=0;l<4;l++) {
10      if (!feof(fidin)) {
          fread(&riff[l],sizeof(char),1,fidin);
      }
      if (!strcmp(riff,"RIFF"))
15      IsWavFile = TRUE;
      else
          IsWavFile = FALSE;
@@ -200,8 +200,8 @@

20 // WAV-Header auswerten
    if (IsWavFile && (header== -1)) {
-       unsigned long  nRiffSize;
```

```

-      fread(&nRiffSize ,sizeof(unsigned long),1,fidin);
+      unsigned int nRiffSize;
25 +      fread(&nRiffSize, sizeof(unsigned int), 1, fidin);
        if (feof(fidin)) {WAVERROR}

        fseek(fidin,20,SEEK_SET);
@@ -221,14 +221,14 @@
30      if (nChannels !=1) {WAVERROR}

        // Samples per second    .. fs
-      unsigned long  nSamplesPerSec;
-      fread(&nSamplesPerSec ,sizeof(unsigned long),1,fidin);
35 +      unsigned int nSamplesPerSec;
+      fread(&nSamplesPerSec, sizeof(unsigned int), 1, fidin);
        if (feof(fidin)) {WAVERROR}
        if (Para.swap_in) SWAP_LONG(nSamplesPerSec);
        fs = (long)nSamplesPerSec;

40      // Avg transfer rate    .. ignorieren
-      fread(&nSamplesPerSec ,sizeof(unsigned long),1,fidin);
+      fread(&nSamplesPerSec, sizeof(unsigned int), 1, fidin);
        if (feof(fidin)) {WAVERROR}

45      // Block alignment        .. ignorieren
@@ -242,12 +242,12 @@
        if (Para.swap_in) SWAP_SHORT(nBits);
        if (nBits !=16) {WAVERROR}

50      unsigned long nHeader;
-      fread(&nHeader ,sizeof(unsigned long),1,fidin);
+      unsigned int nHeader;
+      fread(&nHeader, sizeof(unsigned int), 1, fidin);
55      if (feof(fidin)) {WAVERROR}

-      unsigned long  SizeInBytes;
-      fread(&SizeInBytes ,sizeof(unsigned long),1,fidin);
+      unsigned int SizeInBytes;
60 +      fread(&SizeInBytes, sizeof(unsigned int), 1, fidin);
        if (feof(fidin)) {WAVERROR}
        if (Para.swap_in) SWAP_LONG(SizeInBytes);
        Lx = (long)((double)SizeInBytes*8.0f/(16.0f*1.0f));

```

Listing C.1: Patch to fix gcida WAVE file problems

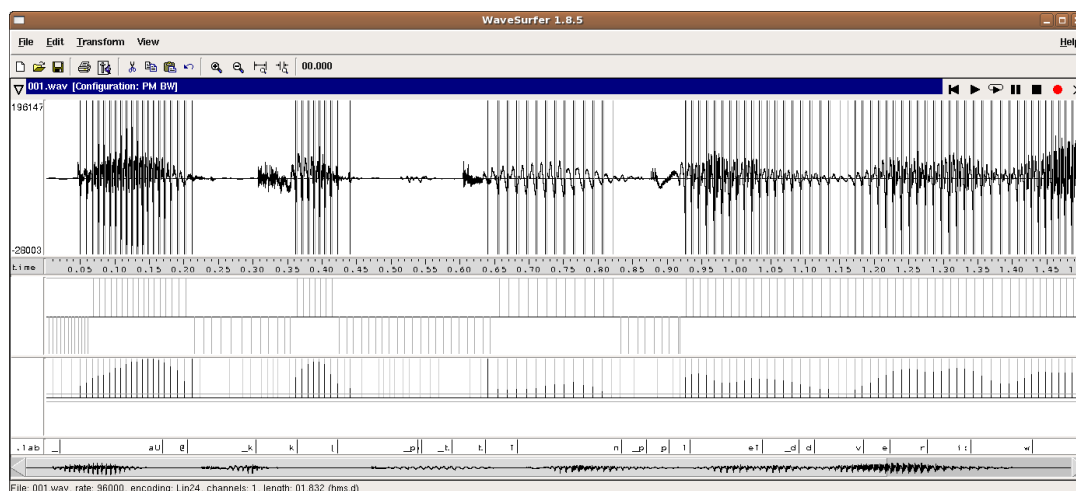


Figure C.1: Wavesurfer pitchmark plugin

C.2. Wavesurfer Pitchmark Plugin

An additional Wavesurfer plugin is provided to evaluate and modify pitchmarks (figure C.1). It has to be placed in the system or user Wavesurfer plugin directory, e.g. `/usr/lib/wsurf1.8/plugins`, `~/.wavesurfer/1.8/plugins` or a directory specified in the `WSPLUGINDIR` environment variable.

The pane is vertically divided into voiced and unvoiced pitchmarks. Automatically extracted pitchmarks can include a confidence score in the range of 0 to 1 that is shown as a split bar. A threshold can be used to select only pitchmarks with a higher confidence score. The status line shows the current position, the selected threshold value and a short description of the available mouse actions.

Pitchmarks can be inserted, deleted and modified. Most actions are accessible from the context menu (figure C.3) as well as with the mouse:

- Pitchmarks can be inserted by selecting *Insert Pitchmark* from the context menu or by double-clicking with the middle mouse button on an empty position of the pane.
- To delete pitchmarks, use *Delete Pitchmark* from the context menu or double-click with the middle mouse button on a specific pitchmark.
- Pitchmarks can be moved by dragging them with the middle mouse button.
- Selection of a pitchmark with the left mouse button shows information about the pitchmark position, voicing and the confidence score as long as the button is pressed.
- Drag a pitchmark between the upper and lower parts with the left mouse button or select *Change Voicing* from the context menu to switch between voiced and unvoiced.

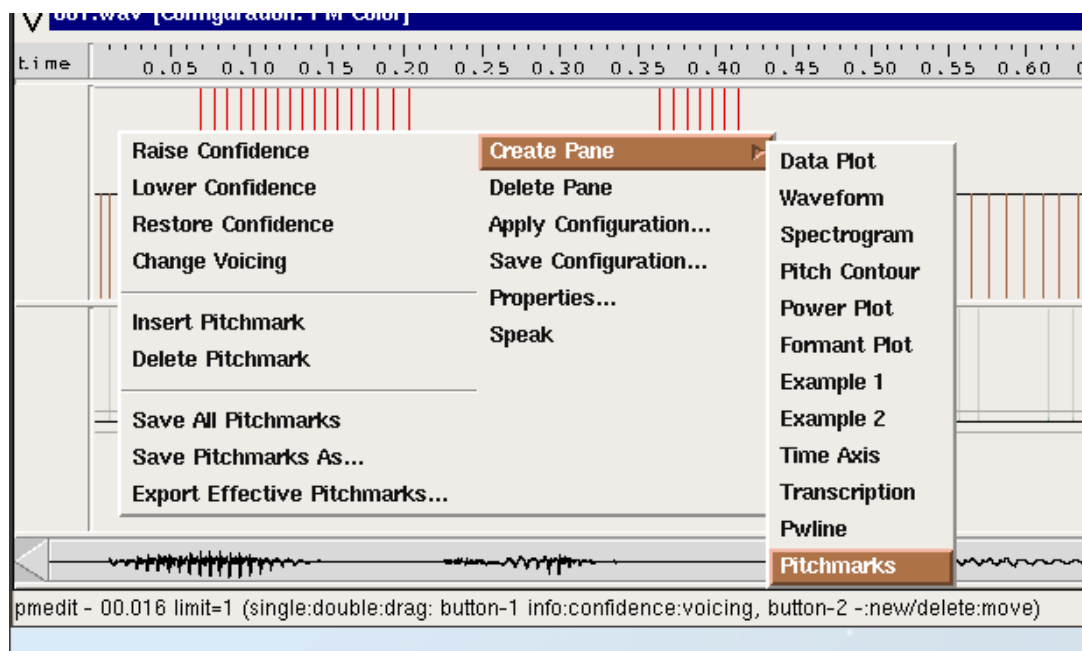


Figure C.2: Pitchmark plugin context menu and status line

- Double click with the left mouse button on a pitchmark to change its confidence value. A pitchmark can be disabled (confidence=0) or unconditionally enabled (confidence=-1). In addition to these options, the context menu provides the possibility to restore the confidence back to its original value.
- Use the scroll wheel on the mouse to switch between two customizable zoom levels.

The plugin can load and save simple and multi-column pitchmark files (see section A.3.3). All stored files will include the selected threshold as a comment in the first line and use the shortest format available for the following pitchmark entries. To export only pitchmarks that have a confidence value larger than the selected threshold in the simple pitchmark format, use the *Export Effective Pitchmarks* entry in the context menu.

The property page (figure C.3) of the plugin allows to modify some of its settings. The upper entries on the page can be used to adjust the appearance of the display elements such as the pitchmark bars for voiced, unvoiced and disabled pitchmarks. The filename extension is used to load a pitchmark file with this extensions if it is available in the same directory. A change to this setting causes the reload of the pitchmark file. The zoom level entries adjust the magnification levels for the scroll wheel actions and the bottom checkboxes allow the voiced and unvoiced bars to extend into waveform and frequency domain panes.

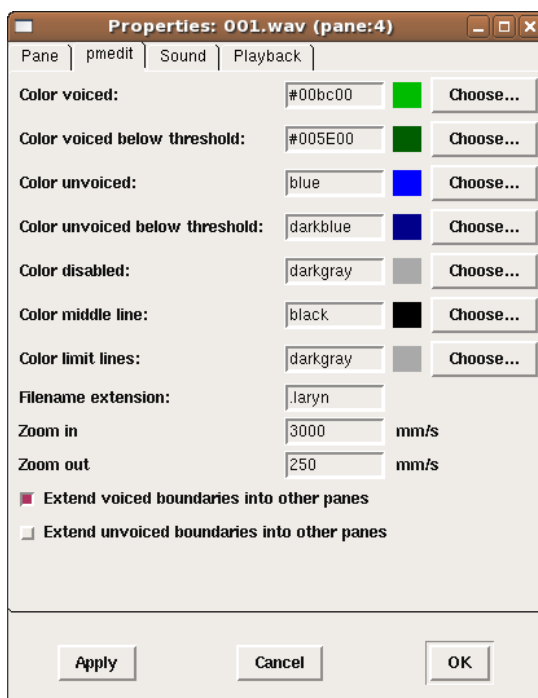


Figure C.3: Pitchmark plugin properties page

C.3. Festival Utterance Viewer

This program can be used to display and explore the contents of Festival utterance files in a concise way. It is invoked from the prompt of the command interpreter as follows:

```
sh> java mh21.progs.RunFestivalViewer [FILES...]
```

You can specify multiple files on the command line.

An example can be seen in figure C.4. The left list shows all available relations and the linked items within them, the right column the features for the selected relation or item, a preview of daughter items and a cross reference list for the current item. Relations that are not direct ancestors of an item are displayed in gray.

C.4. Phone File Conversion

To provide interoperability between the different external programs in the framework, the `RunAnnotators` program can be used to convert between different phone file formats and phone sets:

```
sh> java mh21.progs.RunAnnotators \
    -i input-file -o output-file [OPTIONS]
```

The following command line parameters are supported:

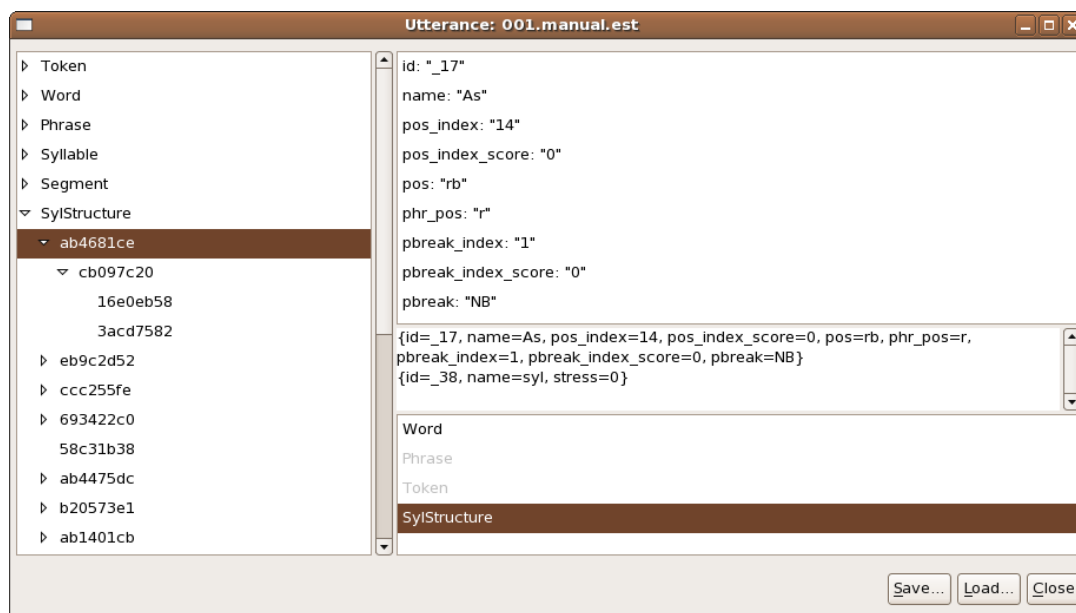


Figure C.4: Festival Utterance Viewer example

```

-i, --input FILE
    input phone file name
-I, --input-format PHONEFORMAT
    file format for the input phones, see section F.3.2
-input-phones-set PHONESET
    phone set for the input phones, see section F.3.5
-o, --output FILE
    output phones file name
-O, --output-format PHONEFORMAT
    file format for the output phones, see section F.3.2
--output-phones-set PHONESET
    phone set for the output phones, see section F.3.5

```

You must specify at least an input and an output file. For Festival and XML files, the phone set is autodetected if possible. Only these two formats are capable of storing interlinked data, all other conversations will result in loss of information.

D. Tables

D.1. POS tags

POS sets are used to sort words into classes that have a similar behavior. The minimal tag set consists of noun, verb, pronoun, preposition, adverb, conjunction, adjective and article, although most currently used sets are larger than that, e.g. *Penn Treebank* with 45 tags or *Susanne* with 353 tags [Ren05].

POS tags				
<i>pos</i>	<i>phr</i>	Penn	Description	Examples
cc	cc	CC	Coordinating conjunction	and both but either nor yet
cd	cd	CD	Cardinal number	'30s billion both i three
dt	dt	DT	Determiner	all an half that the these
ex	ex	EX	Existential <i>there</i>	there
fw	n	FW	Foreign word	deutsche esprit glasnost
in	in	IN	Preposition or subordinating conjunction	beneath from till whereas
jj	j	JJ	Adjective	gallant immense wounded
jjr	j	JJR	Adjective, comparative	better harder richer worse
jjs	j	JJS	Adjective, superlative	best eldest most richest
ls	n	LS	List item marker	a b c first second third
md	md	MD	Modal	can may ought will would
nn	n	NN	Noun, singular or mass	glee ketchup teddy urine
nns	n	NNS	Noun, plural	cans ears flames swings
nnp	n	NNP	Proper noun, singular	Christ Pluto Koenig Zhang
nnps	n	NNPS	Proper noun, plural	Afghans Hondas Vikings
of	of		<i>of</i>	of
pdt	pdt	PDT	Predeterminer	both many quite such
pos	pos	POS	Possessive ending	' 's
prp	prp	PRP	Personal pronoun	he me oneself us
prp	prp	PRP\$	Possessive pronoun	her ours their your
rb	r	RB	Adverb	perhaps tenfold wisely
rbr	r	RBR	Adverb, comparative	duller sooner wider

Table D.1: Alphabetical list of POS tags used for the *pos* and *pos_phr* features in the *Segment* relation and the equivalent name from the Penn Treebank Project (without punctuation) [San95]

POS tags			Description	Examples
<i>pos</i>	<i>phr</i>	Penn		
rbs	r	RBS	Adverb, superlative	best highest least
rp	r	RP	Particle	across away off upon
sym	n	SYM	Symbol	& = < >
to	to	TO	<i>to</i>	to
uh	uh	UH	Interjection	ah hey uh whoopee yeah
vb	v	VB	Verb, base form	bite load taste
vbd	v	VBD	Verb, past tense	cleaned lit meant
vbg	v	VBG	Verb, gerund or present participle	dumping reshaping working
vbn	v	VBN	Verb, past participle	jammed peeled won
vbp	v	VBP	Verb, non-3rd person singular present	bury jostle strive
vbz	v	VBZ	Verb, 3rd person singular present	comes flips needs visits
wdt	wdt	WDT	Wh-determiner	that what whichever
wp	wp	WP	Wh-pronoun	what whoever whom
wp	wp	WP\$	Possessive wh-pronoun	whose
wrb	wrb	WRB	Wh-adverb	how when wherein why

Table D.1: Alphabetical list of POS tags used (continued)

D.2. International Phonetic Alphabet

IPA Phone	SAMPA	German	SAMPA	DRESS	MRPA	British
Vowels						
[a:]	a:	Name				
[a]	a	Dach				
[e:]	e:	gehen				
[ɐ]	6	besser				
[o:]	o:	groß				
[ɔ]	0	offen				
[ɛ:]	E:	spät				
[œ:]	2:	schön				
[œ]	9	plötzlich				
[y:]	y:	süß				
[Y]	Y	hübsch				
[ɛ]	E	Gesetz	e	e	e	pet
[i:]	i:	Termin	i:	i:	ii	ease
[I]	I	in	I	I	i	pit
[u:]	u:	Schule	u:	u:	uu	lose
[ʊ]	U	Hummel	U	U	u	put
[ə]	@	bitte	@	@	@	another
[ɜ:]			3:	3:	@@	furs
[ʌ]			V	V	uh	cut
[ɔ:]			0:	0:	oo	cause
[ɒ]			Q	0	o	pot
[ɑ:]			A:	A:	aa	stars
[æ]			{	}	a	pat
Diphthongs						
[ɔY]	0Y	Kreuz				
[aI]	aI	Eis	aI	aI	ai	rise
[aʊ]	aU	Haus	aU	aU	au	now
[əʊ]			@U	@U	ou	no
[Iə]			I@	I@	i@	fears
[eə]			e@	e@	e@	stairs
[ʊə]			U@	U@	u@	cures
[ɔI]			0I	0I	oi	noise
[eI]			eI	eI	ei	raise
Plosives						
[p]	p	Pein	p	p	p	pin

Table D.2: International Phonetic Alphabet [Int96, Wel95]

D. Tables

IPA Phone	SAMPA	German	SAMPA	DRESS	MRPA	British
[b]	b	Bein	b	b	b	bin
[t]	t	Teich	t	t	t	tin
[d]	d	Deich	d	d	d	din
[k]	k	Kunst	k	k	k	kin
[g]	g	Gunst	g	g	g	give
[ʔ]	ʔ	Antritt	ʔ	ʔ	ʔ	bottle
Affricates						
[pf]	pf	Pfahl				
[ts]	ts	Zahl				
[tʃ]	tʃ	Dschungel				
[tʃ]	tʃ	Deutsch	tʃ	tʃ	ch	chin
[dʒ]			dʒ	dʒ	jh	gin
Fricatives						
[ç]	ç	sicher				
[f]	f	fast	f	f	f	fin
[v]	v	was	v	v	v	vin
[s]	s	Tasse	s	s	s	sin
[z]	z	Hase	z	z	z	zing
[ʃ]	ʃ	waschen	ʃ	ʃ	sh	shin
[ʒ]	ʒ	Genie	ʒ	ʒ	zh	measure
[x]	x	Buch	x	x	x	loch
[h]	h	Hand	h	h	h	hit
[θ]			θ	θ	th	thin
[ð]			ð	ð	dh	this
Approximants, Trills						
[r]	r	Frist				
[j]	j	Jahr	j	j	y	yacht
[l]	l	Land	l	l	l	like
[ɹ]			ɹ	ɹ	ɹ	run
[w]			w	w	w	wasp
Nasals						
[m]	m	mehr	m	m	m	man
[n]	n	nahmen	n	n	n	not
[ŋ]	ŋ	Ding	ŋ	ŋ	ng	long

Table D.2: International Phonetic Alphabet (continued)

D.3. Spline Filter Coefficient Calculation

$$H[n_ , \omega_] := e^{j\omega/2} \cos[\omega/2]^{2n+1};$$

$$G[\omega_] := 4je^{j\omega/2} \sin[\omega/2];$$

$$K[n_ , \omega_] := \text{Simplify} \left[\frac{1 - \text{Abs}[H[n, \omega]]^2}{G[\omega]}, \omega \in \text{Reals} \right]$$

$$\text{Table} \left[\text{InverseFourierTransform}[G[\omega], \right. \\ \left. \omega, t, \text{FourierParameters} \rightarrow \{1, -1\}], \{t, -1, 0\} \right] /. \text{DiracDelta}[0] \rightarrow 1 \\ \{2, -2\}$$

$$\text{Table} \left[\text{InverseFourierTransform}[H[1, \omega], \right. \\ \left. \omega, t, \text{FourierParameters} \rightarrow \{1, -1\}], \{t, -2, 1\} \right] /. \text{DiracDelta}[0] \rightarrow 1 \\ \left\{ \frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8} \right\}$$

$$\text{Table} \left[\text{InverseFourierTransform}[H[2, \omega], \right. \\ \left. \omega, t, \text{FourierParameters} \rightarrow \{1, -1\}], \{t, -3, 2\} \right] /. \text{DiracDelta}[0] \rightarrow 1 \\ \left\{ \frac{1}{32}, \frac{5}{32}, \frac{5}{16}, \frac{5}{16}, \frac{5}{32}, \frac{1}{32} \right\}$$

$$\text{Table} \left[\text{InverseFourierTransform}[\text{Simplify}[\text{TrigToExp}[K[1, \omega]]], \right. \\ \left. \omega, t, \text{FourierParameters} \rightarrow \{1, -1\}], \{t, -2, 3\} \right] /. \text{DiracDelta}[0] \rightarrow 1 \\ \left\{ -\frac{1}{128}, -\frac{7}{128}, -\frac{11}{64}, \frac{11}{64}, \frac{7}{128}, \frac{1}{128} \right\}$$

$$\text{Table} \left[\text{InverseFourierTransform}[\text{Simplify}[\text{Expand}[\text{TrigToExp}[K[2, \omega]]]], \right. \\ \left. \omega, t, \text{FourierParameters} \rightarrow \{1, -1\}], \{t, -4, 5\} \right] /. \text{DiracDelta}[0] \rightarrow 1 \\ \left\{ -\frac{1}{2048}, -\frac{11}{2048}, -\frac{7}{256}, -\frac{11}{128}, -\frac{193}{1024}, \frac{193}{1024}, \frac{11}{128}, \frac{7}{256}, \frac{11}{2048}, \frac{1}{2048} \right\}$$

D.4. Default DTW Distance Function

N_1	N_2	$d(\{N_1\}, \{N_2\})$	$d(\{N_2\}, \{N_1\})$
Default		10	10
*	-	9	9
I	i:	9	9
I	@	9	9
l=	l	9	9
n=	n	9	9

Table D.3: Default DTW distance function for labels (* denotes insertions and deletions)

D.5. Neural Network Training Results

Method	<i>rob200</i> recognition rate in %					
	Fest	Fuji	A	B	C	D
RR_{-s}		71.1	85.7	87.4	87.8	88.5
RR_s		70.7	53.4	52.6	54.3	52.6
RR		71.0	79.3	80.5	81.2	81.3
\overline{RR}		70.9	69.6	70.0	71.1	70.5
RR_{NB}	92.6	93.4	96.0	96.3	96.0	97.5
$RR_{B BB}$	83.3	76.1	80.6	86.4	77.7	75.7
RR	91.0	90.3	93.3	94.6	92.8	93.7
\overline{RR}	88.0	84.8	88.3	91.4	86.9	86.6
RR_{NB}	92.6	93.4	94.8	97.3	97.5	98.1
RR_B	64.1	61.5	52.6	40.4	45.6	40.4
RR_{BB}	67.2	83.6	71.1	86.7	95.6	93.3
RR	87.8	89.5	88.9	90.9	92.2	92.1
\overline{RR}	74.6	79.5	72.8	74.8	79.6	77.3

Table D.4: Recognition rates for the *rob200* corpus (Fest: Festival phrase break prediction; Fuji: Fujisaki parameter estimation; {A, B, C, D}: ANN with a 25 % test set and $\langle context\ size \rangle - \langle neurons\ first\ layer \rangle - \langle neurons\ second\ layer \rangle$ {0-15-10, 1-15-10, 2-10-06, 2-15-10})

Method	<i>kate</i> recognition rate in %					
	Fest	Fuji	A	B	C	D
RR_{-S}		57.3	87.2	73.1	69.0	71.4
RR_S		69.1	45.6	61.2	66.8	62.5
RR		62.1	78.9	68.2	68.1	67.8
\overline{RR}		63.2	66.4	67.1	67.9	66.9
RR_{NB}	97.3	89.6	84.3	87.3	86.4	87.6
$RR_{B BB}$	52.2	56.7	76.5	79.9	79.9	77.8
RR	78.9	76.2	81.1	84.3	83.8	83.6
\overline{RR}	74.8	73.2	80.4	83.6	83.2	82.7
RR_{NB}	97.3	89.6	75.3	82.1	84.0	81.7
RR_B	16.9	25.2	46.7	64.6	59.1	59.9
RR_{BB}	88.6	91.0	88.7	94.5	92.7	95.7
RR	78.4	76.1	71.8	80.7	80.3	79.7
\overline{RR}	67.6	68.6	70.3	80.4	78.6	79.1

Table D.5: Recognition rates for the *kate* corpus with a 25 % test set

Recognition rates in %			25 % test set				75 % test set			
Method	Fest	Fuji	A	B	C	D	A	B	C	D
RR_{-S}		71.1	85.7	87.4	87.8	88.5	70.1	86.9	86.8	85.4
RR_S		70.7	53.4	52.6	54.3	52.6	63.9	47.0	51.0	47.6
RR		71.0	79.3	80.5	81.2	81.3	67.6	79.0	79.6	77.7
\overline{RR}		70.9	69.6	70.0	71.1	70.5	67.0	66.9	68.9	66.1
RR_{NB}	92.6	93.4	96.0	96.3	96.0	97.5	94.4	95.7	95.7	96.8
$RR_{B BB}$	83.3	76.1	80.6	86.4	77.7	75.7	84.2	80.3	76.5	74.5
RR	91.0	90.3	93.3	94.6	92.8	93.7	92.6	93.0	92.3	92.9
\overline{RR}	88.0	84.8	88.3	91.4	86.9	86.6	89.3	88.0	86.1	85.7
RR_{NB}	92.6	93.4	94.8	97.3	97.5	98.1	96.5	98.3	96.7	97.2
RR_B	64.1	61.5	52.6	40.4	45.6	40.4	28.3	23.7	43.4	22.5
RR_{BB}	67.2	83.6	71.1	86.7	95.6	93.3	82.5	89.1	86.9	88.3
RR	87.8	89.5	88.9	90.9	92.2	92.1	88.7	90.2	90.6	89.1
\overline{RR}	74.6	79.5	72.8	74.8	79.6	77.3	69.1	70.4	75.6	69.3

Table D.6: Comparison of the recognition rates of the *rob200* corpus for 25 % and 75 % test sets

D. Tables

		Training Recognition in %			Test Recognition in %					
Network	Label	#	\neg S	S	#	\neg S	S			
0-15-10	\neg S	1404	97.7	2.3	468	85.7	14.3			
	S	350	1.7	98.3	116	46.6	53.4			
1-15-10			99.1	0.9	87.4			12.6		
			1.4	98.6	47.4			52.6		
2-10-06			99.3	0.7	87.8			12.2		
			0.6	99.4	45.7			54.3		
2-15-10			98.6	1.4	88.5			11.5		
			1.1	98.9	47.4			52.6		
			NB	B BB	NB			B BB		
0-15-10	NB	1443	99.3	0.7	481	96.0	4.0			
	B BB	311	1.0	99.0	103	19.4	80.6			
1-15-10			99.3	0.7	96.3			3.7		
			1.3	98.7	13.6			86.4		
2-10-06			99.6	0.4	96.0			4.0		
			0.6	99.4	22.3			77.7		
2-15-10			99.5	0.5	97.5			2.5		
			0.6	99.4	24.3			75.7		
			NB	B	BB	NB	B	BB		
0-15-10	NB	1443	99.9	0.0	0.1	481	94.8	4.2	1.0	
	B	174	0.0	100.0	0.0	57	29.8	52.6	17.5	
	BB	138	0.0	0.0	100.0	45	4.4	24.4	71.1	
1-15-10			100.0	0.0	0.0	97.3			2.3	0.4
			0.0	100.0	0.0	52.6			40.4	7.0
			0.0	0.0	100.0	2.2			11.1	86.7
2-10-06			100.0	0.0	0.0	97.5			1.7	0.8
			1.1	98.9	0.0	36.8			45.6	17.5
			0.0	0.0	100.0	4.4			0.0	95.6
2-15-10			100.0	0.0	0.0	98.1			1.5	0.4
			0.0	100.0	0.0	47.4			40.4	12.3
			0.0	0.0	100.0	0.0			6.7	93.3

Table D.7: Confusion matrix and recognition rates for stress and phrase break estimation optimized by an ANN with a 25% test set for the *rob200* corpus. Network names are in the format *<context size>-<neurons first layer>-<neurons second layer>*.

D. Tables

		Training Recognition in %			Test Recognition in %				
Network	Label	#	¬S	S	#	¬S	S		
0-15-10	¬S	468	97.9	2.1	1404	87.2	12.8		
	S	117	0.9	99.1	349	54.4	45.6		
1-15-10			99.4	0.6		86.9	13.1		
			0.0	100.0		53.0	47.0		
2-10-06			99.6	0.4		86.8	13.2		
			0.9	99.1		49.0	51.0		
2-15-10			99.1	0.9		85.4	14.6		
			0.0	100.0		53.3	46.7		
			NB	B BB		NB	B BB		
0-15-10	NB	481	98.8	1.2	1443	94.4	5.6		
	B BB	104	1.0	99.0	310	15.8	84.2		
1-15-10			99.8	0.2		95.7	4.3		
			1.0	99.0		19.7	80.3		
2-10-06			99.8	0.2		95.7	4.3		
			0.0	100.0		23.5	76.5		
2-15-10			100.0	0.0		96.8	3.2		
			1.0	99.0		25.5	74.5		
			NB	B	BB	NB	B	BB	
0-15-10	NB	481	100.0	0.0	0.0	1443	96.5	1.9	1.5
	B	58	0.0	100.0	0.0	173	44.5	28.3	27.2
	BB	46	0.0	0.0	100.0	137	5.8	11.7	82.5
1-15-10			100.0	0.0	0.0		98.3	1.1	0.6
			0.0	100.0	0.0		57.2	23.7	19.1
			0.0	0.0	100.0		6.6	4.4	89.1
2-10-06			100.0	0.0	0.0		96.7	2.8	0.6
			0.0	100.0	0.0		41.6	43.4	15.0
			0.0	0.0	100.0		6.6	6.6	86.9
2-15-10			100.0	0.0	0.0		97.2	1.9	0.9
			0.0	100.0	0.0		53.2	22.5	24.3
			0.0	0.0	100.0		7.3	4.4	88.3

Table D.8: Confusion matrix and recognition rates for stress and phrase break estimation optimized by an ANN with a 75% test set for the *rob200* corpus. Network names are in the format *<context size>-<neurons first layer>-<neurons second layer>*.

D. Tables

		Training Recognition in %			Test Recognition in %				
Network	Label	#	¬S	S	#	¬S	S		
0-15-10	¬S	3040	87.1	12.9	1013	70.1	29.9		
	S	2070	11.0	89.0	690	36.1	63.9		
1-15-10			93.8	6.3		7.31	26.9		
			2.9	97.1		38.8	61.2		
2-10-06			93.0	7.0		69.0	31.0		
			4.3	95.7		33.2	66.8		
2-15-10			95.7	4.3		71.4	28.6		
			2.3	97.7		37.5	62.5		
			NB	B BB		NB	B BB		
0-15-10	NB	3030	95.2	4.8	1010	84.3	15.7		
	B BB	2080	9.9	90.1	693	23.5	76.5		
1-15-10			98.8	1.2		87.3	12.7		
			2.5	97.5		20.1	79.9		
2-10-06			99.1	0.9		86.4	13.6		
			2.3	97.7		20.1	79.9		
2-15-10			98.8	1.2		87.6	12.4		
			2.0	98.0		22.2	77.8		
			NB	B	BB	NB	B	BB	
0-15-10	NB	3030	87.2	3.6	9.2	1010	75.3	10.6	14.1
	B	1094	3.3	80.8	15.9	364	36.0	46.7	17.3
	BB	987	3.1	0.9	95.0	328	7.6	3.7	88.7
1-15-10			98.2	1.6	0.2		82.1	13.7	4.3
			0.7	99.1	0.2		26.1	64.6	9.3
			0.6	0.3	99.1		4.3	1.2	94.5
2-10-06			86.6	2.7	10.7		84.0	9.9	6.1
			1.5	89.2	9.3		31.6	59.1	9.3
			0.9	0.3	98.8		5.5	1.8	92.7
2-15-10			97.7	1.6	0.7		81.7	13.7	4.7
			0.5	98.9	0.5		37.1	59.9	3.0
			0.5	0.1	99.4		3.7	0.6	95.7

Table D.9: Confusion matrix and recognition rates for stress and phrase break estimation optimized by an ANN test set for the *kate* corpus. Network names are in the format *<context size>-<neurons first layer>-<neurons second layer>*.

E. License for the Developed Programs

All programs developed for this thesis are free software; you can redistribute them and/or modify them under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

The programs are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

E.1. GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

E.1.1. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation’s software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

E.1.2. Terms and Conditions for Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appro-

E. License for the Developed Programs

privately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

E. License for the Developed Programs

- a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights

E. License for the Developed Programs

granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for per-

mission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

E.1.3. No Warranty

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

E.1.4. How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
<one line to give the program's name and what it does.>
```

E. License for the Developed Programs

Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail. If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type 'show w'. This is free software, and you are welcome to
redistribute it under certain conditions; type 'show c' for
details.
```

The hypothetical commands “show w” and “show c” should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than “show w” and “show c”; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in
the program 'Gnomovision' (which makes passes at compilers)
written by James Hacker.
```

```
<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

F. Modules and Attributes

Most components of the annotation system can be selected and configured with command line parameters. The following sections give an overview of the available modules of a certain type and the attributes that are available.

The description follows the following format:

module-name (JAVA_NAME)

Description: Module description

Creates a **JavaClassName** with the following attributes:

Attribute description

...

F.1. Forced Alignment

F.1.1. Pipe.Modules

aligner (ALIGNER)

Description: Phone level aligner

evaluation (EVALUATION)

Description: Result evaluation

gpc (GPC)

Description: Grapheme phoneme conversion

processor (PROCESSOR)

Description: Synthesis result processor

synthesizer (SYNTHESIZER)

Description: speech synthesizer

F.1.2. Gpcs.Type

festival (FESTIVAL)

Description: Synthesis with the Festival Speech Synthesis System

Creates a **FestivalGpc** with the following attributes:

Voice currently selected voice

Scale ratio for phone durations

Phonset phonset of the currently selected voice

Executable pathname of Festival executable
InsertSilence length of silence between words (ms)
Voices all known voices

laip ttsgerman (LAIP TTS GERMAN)

Description: LaipTTS_D of Beat Siebenhaar, Lausanne

Creates a **LaipTTSGermanGpc** with the following attributes:

PauseStrength pause strength
Style speaking style: ‘n’ for normal, ‘s’ for slow/explicit, ‘f’ for fast
Phonset phonset for the German synthesis
LinearSpeed linear speed
F0Variation f0 variation
F0Base f0 base frequency

F.1.3. Synthesizers.Type

mbrola (MBROLA)

Description: Speech synthesizer by Thierry Dutoit, Mons

Creates a **MbrolaSynthesizer** with the following attributes:

Voice currently selected voice
Parameters additional command line parameters seperated by semi-colon
Phonset phonset of the currently selected voice
DatabaseDir path of Mbrola voices
Executable pathname of Mbrola executable
Voices all known voices

F.1.4. Processors.Type

dummy (DUMMY)

Description: No processing

Creates a **DummyProcessor** with the following attributes:

Converter audio file format converter

plosive (PLOSIVE)

Description: Split plosives into pause and burst

Creates a **PlosiveProcessor** with the following attributes:

Plosives recognized plosives
Position steepest or 0.0 (minimum) to 1.0 (maximum)
Converter audio file format converter

F.1.5. Aligners.Type

belarus (BELARUS)

Description: Belarussian aligner of Andrew Davydov

Creates a **DavydovAligner** with the following attributes:

ReferenceShift time shift for reference phones (ms)

Executable pathname of aligner-belarus executable

Converter audio file format converter

dspdtw (DSPDTW)

Description: Dsp_dtwalign aligner of Karlheinz Stöber

Creates a **DspDtwAligner** with the following attributes:

Wine pathname of Wine executable

ReferenceShift time shift for reference phones (ms)

Executable pathname of Dspdtw executable

Converter audio file format converter

strecha (STRECHA)

Description: TUD aligner of Guntram Strecha

Creates a **StrechaAligner** with the following attributes:

ReferenceShift time shift for reference phones (ms)

Executable pathname of dtwlabel-strecha executable

Converter audio file format converter

Config pathname of config file

F.1.6. Evaluations.Type

quadratic (QUADRATIC)

Description: Calculate standard deviation etc.

Creates a **QuadraticEvaluation** with the following attributes:

Type analysis type: type, label, separate

Filename output filename

Compress compress silence and plosives

F.2. Annotation System

F.2.1. Annotators.Type

fuji (FUJI)

Description: Calculates Fujisaki parameters from f0 values with fujies

Creates a **FujisakiAnnotator** with the following attributes:

InputRelation input relation

OutputRelation output relation
ConfigFile pathname of wavelet config file
Executable pathname of Getfuji executable

fuji2f0 (FUJI_TO_F0)

Description: Calculates f0 values from Fujisaki parameters

Creates a **FujiToF0Annotator** with the following attributes:

Accents use accent information
InputRelation input relation
OutputRelation output relation
Phrases use phrase information

fuji2prosody (FUJI_TO_PROSODY)

Description: Calculates stress and phrase breaks from Fujisaki parameters

Creates a **FujiToProsodyAnnotator** with the following attributes:

InputRelation input relation
Split split accents between words
Relative calculate accent strength relative to word length
Phrases create phrases from Fujisaki phrase commands
Limit accent limit

polarisation (POLARISATION)

Description: Determines the signal polarisation

Creates a **PolarisationAnnotator** with the following attributes:

Converter audio file format converter

laryn (LARYNGOGRAPH)

Description: Calculates pitchmarks from a laryngograph signal

Creates a **LaryngographAnnotator** with the following attributes:

LowPass use a low pass filter
OutputRelation output relation
MinScale minimum scale for the Mallat wavelet
Normalize normalize Mallat results separately
MaxScale maximum scale for the Mallat wavelet
MinimumF0 minimum allowed f0 value
Converter audio file format converter
HighPass use a high pass filter
MaximumF0 maximum allowed f0 value
Threshold confidence threshold for pitchmarks

shift (PITCH_SHIFT)

Description: Shifts markers from a laryngograph signal to match signal GCIs

Creates a **PitchShiftAnnotator** with the following attributes:

InputRelation input relation
GciRelation GCI relation
OutputRelation output relation

unvoiced (UNVOICED)

Description: Fills gaps between voiced pitchmarks

Creates an **UnvoicedAnnotator** with the following attributes:

InputRelation input relation
OutputRelation output relation
Insert insert random unvoiced markers
MinimumF0 minimum allowed f0 value
MaxGap maximum gap between voiced pitchmarks (1/f_min units)
MaximumF0 maximum allowed f0 value
Threshold confidence threshold for voiced pitchmarks

gcida (GCIDA)

Description: Calculates pitchmarks with gcida

Creates a **GcidaAnnotator** with the following attributes:

SoxExecutable pathname of Sox executable
OutputRelation output relation
MinimumF0 minimum allowed f0 value
Executable pathname of Gcida executable
Converter audio file format converter
MaximumF0 maximum allowed f0 value

label (LABEL)

Description: Merges an external corrected label file

Creates a **LabelMergeAnnotator** with the following attributes:

PhoneSet phone set for the label file
LabelFormat label file format
LabelFile label file name

lexicon (LEXICON)

Description: Replaces phonemes with the ones from an external lexicon

Creates a **LexiconMergeAnnotator** with the following attributes:

LexiconFile lexicon file name
Plosives recognized plosives
LocalLexiconFile additional lexicon file name
PhoneSet phone set for the label file

pda (PDA)

Description: Calculates f0 values with pda

Creates a **PdaAnnotator** with the following attributes:

NoiseFloor noise floor (ADC units)

LowPassFiltering input signal is low pass filtered
AntiDoublingHalvingRatio anti pitch doubling/halving ratio
VoicedToUnvoiced voiced to unvoiced coefficient threshold
Executable pathname of Pda executable
MaximumF0 maximum allowed f0 value
PeakTracking peak tracking
FrameLength frame length in ms
DecimationFactor decimation factor for downsampling
OutputRelation output relation
WindowLength window length in ms
LpCutOff low pass cutoff frequency (Hz)
UnvoicedToVoiced unvoiced to voiced coefficient threshold
VoicedToUnvoicedRatio voiced to unvoiced coefficient threshold-ratio
MinimumF0 minimum allowed f0 value
Converter audio file format converter
LpOrder low pass filter order (odd value)

pitch2f0 (PITCH_TO_F0)

Description: Converts pitchmarks to f0 values

Creates a **PitchToF0Annotator** with the following attributes:

InputRelation input relation
OutputRelation output relation
MinimumF0 minimum allowed f0 value
FrameLength frame length in ms

pitch2segments (PITCH_TO_SEGMENTS)

Description: Converts pitchmarks to segment labels (highly destructive)

Creates a **PitchToSegmentsAnnotator** with the following attributes:

InputRelation input relation

words2segments (WORDS_TO_SEGMENTS)

Description: Converts words to segment labels (highly destructive)

Creates a **WordsToSegmentsAnnotator** with the following attributes:

Stress mark stressed words with #

power (POWER)

Description: Adds power values

Creates a **PowerAnnotator** with the following attributes:

WindowType window type
Scale scale
OutputRelation output relation
WindowLength window length
Converter audio file format converter

PreEmph preemph coefficient

FrameLength frame length in ms

smooth (SMOOTH)

Description: Smooths f0 values with smoothf0

Creates a **F0SmoothAnnotator** with the following attributes:

InputRelation input relation

Parameters additional command line parameters

OutputRelation output relation

Split split contour on pauses

PowerRelation input relation for power values

PowerParameters power command line parameters

MinimumF0 minimum allowed f0 value

Executable pathname of SmoothF0 executable

MaximumF0 maximum allowed f0 value

syllable (SYLLABLE)

Description: Corrects syllable boundaries

Creates a **SyllableMergeAnnotator** with the following attributes:

LexiconFile lexicon file name

Plosives recognized plosives

LocalLexiconFile additional lexicon file name

SplitPlosives split plosives in the dictionary into pause and burst

PhoneSet phone set for the label file

prosodic (PROSODIC)

Description: Imposes prosodic structures

Creates a **ProsodicMergeAnnotator** with the following attributes:

Stress Festival stress feature name

Format prosodic file format: tctstar

SplitStructure Split utterance in imposed phrases

PBreak Festival phrase break feature name

BLevel Festival break level feature name

File file name

vad (VAD)

Description: Splits a wave file into silence and utterances

Creates a **VadAnnotator** with the following attributes:

NoiseAdaptionDelta maximum delta from nominal noise threshold
for noise threshold adaption (dB)

InputRelation input relation

OutputRelation output relation

MinimalUtteranceDuration minimal utterance duration (s)

MinimalBreakDuration minimal pause duration (s)

NominalNoiseLevel start noise threshold (dB)

NoiseFloor minimum noise threshold for threshold adaption (dB)

UtteranceDelta minimum delta from threshold that leads to state change (dB)

NominalSignalLevel start signal threshold (dB)

NoiseAdaptionConstant weight of old noise threshold in noise threshold adaption per frame (0..1)

SignalAdaptionDelta maximum delta from nominal signal threshold for signal threshold adaption (dB)

SignalAdaptionConstant weight of old signal threshold in signal threshold adaption per frame (0..1)

relation (RELATION)

Description: Moves and deletes relations

Creates a **RelationAnnotator** with the following attributes:

Relations relation names e.g. delete=;move=newname

F.3. Helper Modules

F.3.1. ConversionProviders.Type

sox (SOX)

Description: SoX Sound eXchange executable by Chris Bagwell

Creates a **SoxConversionProvider** with the following attributes:

Executable pathname of Sox executable

ecasound (ECASOUND)

Description: Ecasound package by Kai Vehmanen

Creates an **EcasoundConversionProvider** with the following attributes:

Executable pathname of Ecasound executable

F.3.2. PhoneFiles.Type

mbrola (MBROLA)

Description: Mbrola synthesizer phone file

Creates a **MbrolaPhoneFile**

wavesurfer (WAVESURFER)

Description: Wavesurfer segment label file

Creates a **WavesurferPhoneFile**

xwaves (XWAVES)

Description: XWaves label file with labels at the beginning of a segment

Creates a **XWavesPhoneFile**

xwaves-late (XWAVES_LATE)

Description: XWaves label file with labels at the end of a segment

Creates a **XWavesLatePhoneFile**

festival (FESTIVAL)

Description: Edinburgh Speech tools utterance format used with Festival

Creates a **FestivalPhoneFile**

xml (FESTIVAL_XML)

Description: XML utterance format

Creates a **FestivalXmlPhoneFile**

phonetic (TC_STAR_PHONETIC)

Description: TC-Star phonetic format (write only)

Creates a **PhoneticPhoneFile**

prosodic (TC_STAR_PROSODIC)

Description: TC-Star prosodic format (write only)

Creates a **ProsodicPhoneFile**

pitch (PITCH)

Description: Text pitchmark file format

Creates a **TextPitchFile**

lexicon (LEXICON)

Description: Lexicon file format

Creates a **LexiconFile**

F.3.3. Text.Type

latin1 (LATIN1)

Description: Latin1 text

latin1-strip (LATIN1_STRIP)

Description: Latin1 text, non-ascii stripped

siemens (SIEMENS)

Description: Siemens phrase format

utf8 (UTF8)

Description: UTF-8 text

utf8-strip (UTF8_STRIP)

Description: UTF-8 text, non-ascii stripped

F.3.4. Window.Type

rect (RECT)

hanning (HANNING)
bartlett (BARTLETT)
blackman (BLACKMAN)
hamming (HAMMING)

F.3.5. PhoneTransformer.Name

BritishMRPA (BRITISH_MRPA)
Description: British English Festival (MRPA)
BritishMbrola (BRITISH_MBROLA)
Description: British English Mbrola (SAMPA)
BritishDress (BRITISH_DRESS)
Description: British English DRESS (modified SAMPA)
AmericanMRPA (AMERICAN_MRPA)
Description: American English Festival (MRPA)
AmericanMbrola (AMERICAN_MBROLA)
Description: American English Mbrola (SAMPA)
GermanMbrola (GERMAN_MBROLA)
Description: German Mbrola (SAMPA)

F.4. Evolutionary Optimization

F.4.1. AbstractBooleanGene.BooleanCombiners

average (AVERAGE)
Description: Averages parents' gene values
Creates an **AverageGeneCombiner**

F.4.2. AbstractBooleanGene.BooleanInitializers

fixed (FIXED)
Description: Fixed value
Creates a **FixedGeneInitializer** with the following attributes:
 Value value
random (RANDOM)
Description: Random selection
Creates a **RandomGeneInitializer**

F.4.3. AbstractBooleanGene.BooleanMutators

random (RANDOM)

Description: Random mutation

Creates a **RandomGeneMutator** with the following attributes:

RelativeRange probability for state change

F.4.4. AbstractDoubleGene.DoubleCombiners

average (AVERAGE)

Description: Averages parents' values

Creates an **AverageGeneCombiner**

F.4.5. AbstractDoubleGene.DoubleInitializers

fixed (FIXED)

Description: Fixed value

Creates a **FixedGeneInitializer** with the following attributes:

Value value

random (RANDOM)

Description: Random selection

Creates a **RandomGeneInitializer**

F.4.6. AbstractDoubleGene.DoubleMutators

gaussian (GAUSSIAN)

Description: Gaussian distributed random mutation

Creates a **GaussianGeneMutator** with the following attributes:

StandardDeviation relative standard deviation

random (RANDOM)

Description: Random mutation

Creates a **RandomGeneMutator** with the following attributes:

RelativeRange relative range

F.5. Neural Networks

F.5.1. FieldExtractors.Type

duration (DURATION)

Description: Phone durations

Creates a **DurFieldExtractor** with the following attributes:

Fields field names

Settings settings file name suffix, empty for automatic

fuji (FUJI)

Description: Fujisaki parameters

Creates a **FujiFieldExtractor** with the following attributes:

Fields field names

Settings settings file name suffix, empty for automatic

double (DOUBLE)

Description: Double feature

Creates a **DoubleFieldExtractor** with the following attributes:

Feature feature name

Fields field names

Settings settings file name suffix, empty for automatic

int (INT)

Description: Integer feature

Creates an **IntFieldExtractor** with the following attributes:

Feature feature name

Fields field names

Settings settings file name suffix, empty for automatic

string (STRING)

Description: String feature enumeration

Creates a **StringFieldExtractor** with the following attributes:

Feature feature name

Fields field names

Settings settings file name suffix, empty for automatic

syllables (SYLS)

Description: Number of syllables

Creates a **SyllablesFieldExtractor** with the following attributes:

Fields field names

Settings settings file name suffix, empty for automatic

time (TIME)

Description: Word start and end time

Creates a **TimeFieldExtractor** with the following attributes:

Fields field names

Settings settings file name suffix, empty for automatic

index (INDEX)

Description: Positional indices

Creates an **IndexFieldExtractor** with the following attributes:

Fields field names

Settings settings file name suffix, empty for automatic

series (SERIES)

Description: Time series

Creates a **TimeSeriesFieldExtractor** with the following attributes:

Feature feature name

Relation relation name

Fields field names

Logarithmic Use logarithmic scale

Settings settings file name suffix, empty for automatic

break (BREAK)

Description: Phrase breaks

Creates a **BreakFieldExtractor** with the following attributes:

Feature feature name

Fields field names

Settings settings file name suffix, empty for automatic

G. CD-ROM Contents

The CD-ROM contains all generated and used source code as well as the data produced during evaluation. For easy access, the thesis itself and all literature from the bibliography available in electronic form is also included.

G.1. Directory Structure

/Data Data generated during the evaluation of the framework. The original wave files had to be omitted to save space.

/Source Code Source code used or developed during the thesis.

/Wavesurfer Plugin Tcl/Tk Plugin to view and modify pitchmarks with Wavesurfer. Additionally, some wrapper scripts and a Wavesurfer configuration file is provided to act as a drop-in replacement for the korr script.

/Framework Framework root directory.

/Classes Java framework classes, see next section for details.

/Root Configuration files, external lexicons etc. that are needed during runtime.

/External Programs Source code for external programs that were used for this thesis if available. If some of the programs had patches applied to work correctly, these are also included.

/Binary Only Programs used that were only available in binary form. The windows executables can be executed on Linux operating systems with the Wine Windows API emulator.

/Aligner Stoeber Phoneme aligner of the University of Bonn.

/Smooth F0 Contour Smoothes f0 contours with splines and linear interpolation.

/Fujisaki Parameter Extraction Extracts Fujisaki parameters from f0 contours.

/Scripts Batch conversion script examples.

/Report Electronic version of the thesis. All graphics and the L^AT_EX sources to regenerate the PDF version are included. The bibliography of this version links to an electronic version of the literature if available.

/Paper SpeechProsody 2006 Paper sources for the Speech Prosody 2006 conference that details some of the prosodic prediction results.

/Literature Used literature if available in electronic form.

G.2. Java Package Structure

The framework consists of some external Java packages and additional 60 000 lines of documented Java source code which are split into the following packages:

- mh21** Class root of the annotation framework. Contains some classes that did not fit anywhere else.
- .prosodic** Core framework classes and package root for the alignment and annotation classes.
- .evolution** Evolutionary optimization of e. g. Fujisaki parameters.
- .phonefile** Classes for loading and storing of annotation information.
- .conversion** Wave file conversion.
- .aligner** Forced phoneme alignment.
- .processor** Aligner preprocessing, e. g. plosive splitting.
- .annotator** Annotation classes.
- .evaluation** Basic result evaluation.
- .synthesizer** Speech synthesis support.
- .gpc** Grapheme phoneme conversion.
- .nn** Neural network optimization of word-level features.
- .progs** Supported command line executables.
- .soundproc** Wave file reading and writing as well as various signal processing operations.
- .utterance** Festival utterance, relation and content classes as well as some utility functions.
- .evolution** Evolutionary optimization classes. Several interfaces are defined to provide the core functionality. The implementation is limited to the most common optimization methods.
- .nn** High-level interface to MFNs that allows to create, load, train and evaluate such networks.
 - .core** Core classes for the management of neural networks representing all necessary parts like neurons, links, patterns etc.
- .options** High-Level command line option parsing that generates help pages and automatically parses provided options.
- .attributable** Interfaces for command line accessible attributes and an implementation using reflection.
- .collection** Collection interfaces and classes for tree-like structures.
- .texdoclet** Generation of \LaTeX documentation from Java source code.
- .related** Unsupported executable classes for test programs and quick calculations.
- .tests** Over 110 tests to prevent regressions.
- laiptts** LaipTTS speech synthesis for German.
- org.apache.commons** Apache Commons utility functions.
- gnu.getopt** GNU getopt command line option parsing.

Bibliography

- [BBH⁺99] Batliner, Anton, Jan Buckow, Richard Huber, Volker Warnke, Elmar Nöth, and Heinrich Niemann: *Prosodic feature evaluation: brute force or welldesigned?* In Ohala, John J., Yoko Hasegawa, Manjari Ohala, Daniel Granville, and Ashlee C. Bailey (editors): *Proceedings of the 14th International Congress of Phonetic Sciences*, volume 3, pages 2315–2318, San Francisco, California, USA, 1999. [Local Copy](#).
- [BBH⁺00] Buckow, Jan, Anton Batliner, Richard Huber, Heinrich Niemann, Elmar Nöth, and Volker Warnke: *Detection of prosodic events using acoustic-prosodic features and part-of-speech tags*. In *Proceedings of the 5th International Workshop Speech and Computer (SPECOM)*, pages 63–66, St. Petersburg, Russia, 2000. [Local Copy](#).
- [BBJK93] Batliner, Anton, Susanne Burger, Birgit Johne, and Andreas Kießling: *MÜSLI: a classification scheme for laryngealizations*. In *Proceedings of the European Speech Communication Association (ESCA) Workshop on Prosody*, pages 176–179, Lund, Sweden, 1993. [Local Copy](#).
- [BBNW00] Batliner, Anton, Jan Buckow, Heinrich Niemann, and Volker Warnke: *The prosody module*. In Wahlster, Wolfgang (editor): *Verbmobil: foundations of speech-to-speech translation*, pages 106–121. Springer, 2000.
- [BHN02] Burnett, Greg C., John F. Holzrichter, and Lawrence C. Ng: *System and method for characterizing voiced excitations of speech and acoustic signals, removing acoustic noise from speech, and synthesizing speech*. US Patent No. 6,377,919, 2002. [Local Copy](#).
- [BHT⁺04] Bonafonte, Antonio, Harald Höge, Herbert S. Tropic, Asuncion Moreno, Henk van der Heuvel, David Sündermann, Ute Ziegenhain, Javier Pérez, Imre Kiss, and Oliver Jokisch: *TTS baselines and specifications*. Technology and Corpora for Speech to Speech Translation (TC-STAR) Deliverable D8, 2004. [Local Copy](#).
- [BI96] Bunnell, H. Timothy and William Idsardi (editors): *Proceedings of the 4th International Conference on Spoken Language Processing (ICSLP)*, Philadelphia, Pennsylvania, USA, 1996.

- [BKK⁺98] Batliner, Anton, Richard Kompe, Andreas Kießling, Marion Mast, Heinrich Niemann, and Elmar Nöth: *M = Syntax + Prosody: A syntactic-prosodic labelling scheme for large spontaneous speech databases*. Speech Communication, 25(4):193–222, 1998. [Local Copy](#).
- [BLM83] Baer, Thomas, Anders Löfqvist, and Nancy S. McGarr: *Laryngeal vibrations: A comparison between high-speed filming and glottographic techniques*. Journal of the Acoustic Society of America (JASA), 73(4):1304–1308, 1983. [Local Copy](#).
- [BNB⁺01] Batliner, Anton, Elmar Nöth, Jan Buckow, Richard Huber, Volker Warnke, and Heinrich Niemann: *Duration features in prosodic classification: why normalization comes second, and what they really encode*. In Bacchiani, Michiel, Julia Hirschberg, Diane Litman, and Mari Ostendorf (editors): *Proceedings of the International Speech Communication Association (ISCA) Tutorial and Research Workshop Workshop (ITRW) on Prosody in Speech Recognition and Understanding*, pages 23–28, Red Bank, New Jersey, USA, 2001. [Local Copy](#).
- [BTC02] Black, Alan W., Paul A. Taylor, and Richard Caley: *The Festival speech synthesis system*. University of Edinburgh, 2002. [Local Copy](#).
- [BWN⁺98] Batliner, Anton, Volker Warnke, Elmar Nöth, Jan Buckow, Richard Huber, and Matthias Nutt: *How to label accent position in spontaneous speech automatically with the help of syntactic-prosodic boundary labels*. Verbmobil-Report 228, 1998. [Local Copy](#).
- [CK85] Childers, Donald G. and Ashok K. Krishnamurthy: *A critical review of electroglottography*. Critical Reviews in Biomedical Engineering, 12(2):131–161, 1985.
- [Cyg05] Cygwin Project: *A Linux-like environment for Windows*. <http://www.cygwin.com/>, 2005.
- [Dau91] Daubechies, Ingrid: *Ten lectures on wavelets*. SIAM, 1991. [Local Copy](#).
- [Dav04] Давыдов, А. Г. and Б. М. Лобанов: *Сегментация речи на основе метода модифицированного непрерывного динамического программирования*. Technical report, Объединенный институт проблем информатики НАН Беларуси, 2004. [Local Copy](#).

- [DPP⁺96] Dutoit, Thierry, Vincent Pagel, Nicolas Pierret, François Bataille, and Olivier van der Vreken: *The Mbrola project: Towards a set of high-quality speech synthesizers free of use for non-commercial purposes*. In Bunnell, H. Timothy and William Idsardi [BI96], pages 1393–1397. [Local Copy](#).
- [Eng03] Engel, Toni: *Robuste Markierung von Grundfrequenzperioden*. Diplomarbeit, Technische Universität Dresden, 2003. [Local Copy](#).
- [Ent96] Entropic, Inc.: *Get_f0 manual*, 1996. [Local Copy](#).
- [Ent98] Entropic, Inc.: *Xlabel manual*, 1998. [Local Copy](#).
- [FG99] Fitch, W. Tecumseh and Jay Giedd: *Morphology and development of the human vocal tract: A study using magnetic resonance imaging*. Journal of the Acoustic Society of America (JASA), 106(3):1511–1522, 1999. [Local Copy](#).
- [FHBZ02] Fischer, Igor, Fabian Hennecke, Christian Bannes, and Andreas Zell: *JavaNNS - Java Neural Network Simulator*, 2002. [Local Copy](#).
- [FKLL04] Ferencz, Attila, Jeong-Su Kim, Yong-Beom Lee, and Jae-Won Lee: *Automatic pitch marking and reconstruction of glottal closure instants from noisy and deformed electro-glottograph signals*. In Soon Hyob Kim and Dae Hee Youn (editors): *Proceedings of the 8th International Conference on Spoken Language Processing (ICSLP)*, pages 2437–2440, Jeju Island, Korea, 2004. [Local Copy](#).
- [FLM93] Fujisaki, Hiroya, Mats Ljungqvist, and Hiroshi Murata: *Analysis and modeling of word accent and sentence intonation in Swedish*. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 211–214, Minneapolis, Minnesota, USA, 1993. [Local Copy](#).
- [FO95] Fujisaki, Hiroya and Sumio Ohno: *Analysis and modeling of fundamental frequency contours of english utterances*. In *Proceedings of the 4th European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 421–424, Madrid, Spain, 1995.
- [Fuj05] Fujisaki, Hiroya: *The interplay between physiology, physics and phonetics in the production of tonal features of speech of various languages*. In Kokkinakis, George, Nikos Fakotakis, Evangelos Dermatas, and Rodmonga Potapova (editors): *Proceedings of the 10th International Workshop Speech and Computer (SPECOM)*, volume 1, pages 39–48, Patras, Greece, 2005. [Local Copy](#).

- [GRB⁺96] Grice, Martine, Matthias Reyelt, Ralf Benzmlüller, Jsrg Mayer, and Anton Batliner: *Consistency in transcription and labelling of German intonation with GToBI*. In Bunnell, H. Timothy and William Idsardi [BI96], pages 1716–1719. [Local Copy](#).
- [Gri89] Grimm, Brüder: *Märchen*. Thienemann, 1989.
- [HB01] Heitkötter, Jörg and David Beasley: *The Hitch-Hiker’s Guide to Evolutionary Computation: A list of Frequently Asked Questions (FAQ)*. <http://surf.de.uu.net/encore/>, 2001. [Local Copy](#).
- [Hes83] Hess, Wolfgang J.: *Pitch determination of speech signals*. Springer, 1983.
- [HF38] Herriott, W. and D. W. Farnsworth: *High speed motion pictures of the vocal cords*. Journal of the Acoustical Society of America (JASA), 9(3):274, 1938.
- [HI84] Hess, Wolfgang J. and Helge Indefrey: *Accurate pitch determination of speech signals by means of a Laryngograph*. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 9, pages 73–76, San Diego, California, USA, 1984. [Local Copy](#).
- [Hof98] Hoffmann, Rüdiger: *Signalanalyse und Erkennung – Eine Einführung für Informationstechniker*. Springer, 1998.
- [Hof04] Hofmann, Michael: *Optimierung einer trainingsbasierten Prosodiegenerierung*. Studienarbeit, Technische Universität Dresden, 2004. [Local Copy](#).
- [Hus99] Husson, Jean-Luc: *Evaluation of a segmentation system based on multi-level lattices*. In *Proceedings of the 6th European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 471–474, Budapest, Hungary, 1999. [Local Copy](#).
- [Int96] International Phonetic Association: *International Phonetic Alphabet*. <http://www.arts.gla.ac.uk/IPA/>, 1996. [Local Copy](#).
- [JH04] Jokisch, Oliver and Michael Hofmann: *Evolutionary optimization of an adaptive prosody model*. In Soon Hyob Kim and Dae Hee Youn (editors): *Proceedings of the 8th International Conference on Spoken Language Processing*, pages 797–800, Jeju Island, Korea, 2004. [Local Copy](#).
- [KC86] Krishnamurthy, Ashok K. and Donald G. Childers: *Two-channel speech analysis*. In *Proceedings of the IEEE International*

- Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 34, pages 730–743, Tokyo, Japan, 1986. [Local Copy](#).
- [KL03] Kruschke, Hans and Michael Lenz: *Estimation of the parameters of the quantitative intonation model with continuous wavelet analysis*. In *Proceedings of the 8th European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 2881–2884, Geneva, Switzerland, 2003. [Local Copy](#).
- [KLKP00] Kang, Dong Gyu, Jung Chul Lee, Sang Hun Kim, and Jun Park: *Pitch modification method by glottal closure interval extrapolation*. US Patent No. 6,125,344, 2000. [Local Copy](#).
- [KM02] Kerkhoff, Joop and Erwin Marsi: *NeXTeNS: a new open source text-to-speech system for Dutch*. Presentation at the 13th meeting of Computational Linguistics in the Netherlands (CLIN), 2002. [Local Copy](#).
- [Kos01] Kossebau, Friedrich W. H.: *Anwendung von Methoden aus dem Gebiet Evolutionäres Rechnen zum optimierten Entwurf von integrierten intelligenten Systemen*. Studienarbeit, Technische Universität Dresden, 2001. [Local Copy](#).
- [Kot05] Kotnik, Bojan: *Determination of characteristic points inside the glottal cycle*. Technical report, University of Maribor, Faculty of Electrical Engineering and Computer Science, 2005. [Local Copy](#).
- [Kru03] Kruschke, Hans: *Dokumentation des Programms cutter*. Fakultät Elektrotechnik, Institut für Akustik und Sprachkommunikation, Technische Universität Dresden, 2003. [Local Copy](#).
- [KWS97] Kipp, Andreas, Maria-Barbara Wesenick, and Florian Schiel: *Pronunciation modeling applied to automatic segmentation of spontaneous speech*. In *Proceedings of the 5th European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 1023–1026, Rhodes, Greece, 1997. [Local Copy](#).
- [KZ96] Keller, Eric and Brigitte Zellner: *Output requirements for a high-quality speech synthesis system: The case of disambiguation*. In Boitet, Christian (editor): *Proceedings of the International Seminar on Multimodal Interactive Disambiguation (MIDDIM)*, pages 300–308, Grenoble, France, 1996. [Local Copy](#).
- [LC97] Lander, Terri and Tim Carmell: *Structure of spoken language: Spectrogram reading*. <http://speech.bme.ogi.edu/tutordemos/SpectrogramReading/>, 1997. [Local Copy](#).

- [LT04] Lobanov, Boris M. and Lilia I. Tsurulnik: *Phonetic-acoustical problems of personal voice cloning by TTS*. In Kokkinakis, George, Nikos Fakotakis, Evangelos Dermatas, and Rodmonga Potapova (editors): *Proceedings of the 9th International Workshop Speech and Computer (SPECOM)*, pages 17–21, St. Petersburg, Russia, 2004. [Local Copy](#).
- [Mar97] Marasek, Krzysztof: *EGG and voice quality*. <http://www.ims.uni-stuttgart.de/phonetik/EGG/>, 1997. [Local Copy](#).
- [Mic99] Michaelis, Dirk: *Das Göttinger Heiserkeits-Diagramm - Entwicklung und Prüfung eines akustischen Verfahrens zur objektiven Stimmgütebeurteilung pathologischer Stimmen*. Dissertation, Georg-August-Universität zu Göttingen, 1999. [Local Copy](#).
- [Mix97] Mixdorff, Hansjörg: *Intonation patterns of German - model-based quantitative analysis and synthesis of f0 contours*. PhD thesis, Dresden University of Technology, 1997. [Local Copy](#).
- [Möh99] Möhler, Gregor: *IMS Festival*. <http://www.ims.uni-stuttgart.de/phonetik/synthesis/index.html>, 1999.
- [MPK01] Mataušek, Jindrich, Josef Psutka, and Jiří Krůta: *design of speech corpus for text-to-speech synthesis*. In *Proceedings of the 7th European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 2047–2050, Aalborg, Denmark, 2001. [Local Copy](#).
- [MZ92] Mallat, Stephane G. and Sifen Zhong: *Characterization of signals from multiscale edges*. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(7):710–732, 1992. [Local Copy](#).
- [Pel04] Pelteku, Altin: *Development of an electromagnetic glottal waveform sensor for applications in high acoustic noise environments*. Master’s thesis, Worcester Polytechnic Institute, 2004. [Local Copy](#).
- [PSP⁺92] Portele, Thomas, B. Steffan, R. Preuss, Walter F. Sendlmeier, and Wolfgang Hess: *HADIFIX - a speech synthesis system for German*. In *Proceedings of the 2th International Conference on Spoken Language Processing (ICSLP)*, volume 2, pages 1227–1230, Banff, Alberta, Canada, 1992.

Bibliography

- [Ren05] Renals, Steve: *Part-of-speech tagging*. <http://www.inf.ed.ac.uk/teaching/courses/icl/lectures.html>, 2005. [Local Copy](#).
- [Rot90] Rothenberg, Martin: *Tracking multielectrode electroglottograph*. US Patent No. 4,909,261, 1990. [Local Copy](#).
- [Rot92] Rothenberg, Martin: *A multichannel electroglottograph*. Journal of Voice, 6(1):36–43, 1992. [Local Copy](#).
- [Sai92] Saito, Shuzo: *Speech Science and Technology*. Ohmsha, IOS Press, 1992.
- [San95] Santorini, Beatrice: *Part-of-speech tagging guidelines for the Penn Treebank Project*. <http://www.cis.upenn.edu/treebank/>, 1995. [Local Copy](#).
- [SB00] Sjölander, Kåre and Jonas Beskow: *Wavesurfer - an open source speech tool*. In Yuan, Baezon, Taiyi Huang, and Xiaofang Tang (editors): *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, volume 4, pages 464–467, Beijing, China, 2000. [Local Copy](#).
- [Sch04] Schiel, Florian: *Bavarian Archive for Speech Signals - file formats*. <http://www.phonetik.uni-muenchen.de/Bas/BasFormatseng.html>, 2004. [Local Copy](#).
- [Sit04] Sitaram, Dorai: *Teach yourself Scheme in fixnum days*. <http://www.ccs.neu.edu/home/dorai/>, 2004. [Local Copy](#).
- [Stö97] Stöber, Karlheinz: *Methoden der automatischen Lautsegmentierung unter Einsatz selbstorganisierender Karten und Dynamic Time Warping*. Diplomarbeit, Universität Bonn, 1997.
- [Str00] Strecha, Guntram: *Multilinguale Etikettierung natürlicher Sprachsignale auf Basis synthetischer Referenzsignale*. Diplomarbeit, Technische Universität Dresden, 2000. [Local Copy](#).
- [TSB⁺00] Titze, Ingo R., Brad. H. Story, Gregory. C. Burnett, John. F. Holzrichter, Lawrence C. Ng, and Wayne A. Lea: *Comparison between electroglottography and electromagnetic glottography*. Journal of the Acoustic Society of America (JASA), 107(1):581–588, 2000. [Local Copy](#).
- [Val04] Valens, Clemens: *A really friendly guide to wavelets*. <http://perso.wanadoo.fr/polyvalens/clemens/clemens.html>, 2004. [Local Copy](#).

Bibliography

- [VHH98] Vary, Peter, Ulrich Heute und Wolfgang J. Hess: *Digitale Sprachsignalverarbeitung*. Teubner, 1998.
- [Wel95] Wells, John C.: *Computer-coding the IPA: a proposed extension of SAMPA*. <http://www.phon.ucl.ac.uk/home/sampa/>, 1995. [Local Copy](#).
- [Win05] Wine Project: *An open source implementation of the Windows API on top of X and Unix*. <http://www.winehq.com/>, 2005.
- [WK96] Wesenick, Maria-Barbara and Andreas Kipp: *Estimating the quality of phonetic transcriptions and segmentations of speech signals*. In Bunnell, H. Timothy and William Idsardi [BI96], pages 129–132. [Local Copy](#).
- [Zel94] Zell, Andreas: *Simulation Neuronaler Netze*. Addison-Wesley, 1994.
- [Zel95] Zell, Andreas: *SNNS - Stuttgart Neural Network Simulator*, 1995. [Local Copy](#).
- [Zit99] Zitzler, Eckart: *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Technische Hochschule Zürich, 1999. [Local Copy](#).