Graz University of Technology

Master Thesis - Telematics

RTBlocks: A Cross-Platform Algorithm Design Framework for Real-Time Audio Processing on Android

conducted at the
Signal Processing and Speech Communications Laboratory
Graz University of Technology, Austria

by
Christoph Klug

Supervisors:
Dipl.-Ing. Dr.techn. Martin Hagmüller
Assoc.Prof. Dipl.-Ing. Dr.mont. Franz Pernkopf

Graz, October 1, 2012

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

date	(signature)

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am	(Unterschrift)

Abstract

In this treatise the new data flow programming framework *RTBlocks* which supports short-term algorithm development for mobile devices is introduced. Currently the framework exists in two programming languages: MATLAB and Java. A simple workflow for developing algorithms in MATLAB and targeting the Android OS is provided. For proof-of-concept two fundamental frequency (F0) estimation algorithms and two voice activity detection (VAD) algorithms are implemented. The real practical value of the framework is demonstrated by integrating third party libraries which are written in C or C++.

Kurzfassung

Im Zuge dieser Arbeit wird das neu entwickelte datenfluss-orientierte Programmierframework *RTBlocks* vorgestellt, welches den Entwicklungsprozess von Kurzzeit-Algorithmen für mobile Plattformen unterstützt. Das Framework ist zur Zeit in zwei verschiedenen Programmiersprachen verfügbar: MATLAB und Java. Weiters wird ein kompletter Entwicklungsprozess vorgestellt, beginnend beim Designen von Algorithmen mit MATLAB bis hin zur Implementierung der Algorithmen für Android OS. Für die Evaluierung des Frameworks werden zwei Algorithmen zur Bestimmung der Grundfrequenz und zwei Algorithmen zur Sprechpausenerkennung implementiert. Weiters wird die Flexibilität des Frameworks durch die Integrierung von C und C++ - Softwarepaketen von Drittanbietern gezeigt.

Acknowledgment

I would like to thank my advisor Dipl.-Ing. Dr.techn. Martin Hagmüller who has supported me during the whole project.

I would also like to show my gratitude to my whole family and my friends for their great assistance during writing this work. Special thanks to my mother, my father and my brother who have given me the possibility to reach my goals.

Especially I am grateful to my life partner Karina who has been very patient with me during these busy days. She is my inspiration and always brings the best of me to the surface.

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

ACF Auto-Correlation Function xi, xii, xxxvii, xliv, lxvi–lxx, lxxii–lxxviii, lxxx–lxxxii, lxxxiv, lxxxix, 24, 29, 30, 32

ADK Accessory Development Kit 4

ADT Android Developer Tools 4, 5

AGC Automatic Gain Control lxiv, 25

AM Average Magnitude Function xxxvi

AMDF Average Magnitude Difference Function xxxvii, lxviii

API Application Programming Interface 30, 47

ARM Advanced RISC Machine 32, 47

ASR Automatic Speech Recognition xxviii, xxxii, lxiv

CPU Central Processing Unit 31, 32, 47

DF Distance Function lxxx–lxxxii, 30

DF1 Direct Form I 19

DF2 Direct Form II xi, 19–21

DFT Discrete Fourier Transform 4, 26

DSSI Disposable Soft Synth Interface 3

DTX Discontinuous Transmission Mode xii, xxxiv

ELAN European Distributed Corpora Project Linguistic Annotator xxix

F0 Fundamental Frequency xi, xiii, xxvii, xxviii, xlv, lxv, lxx, lxxii–lxxxii, lxxxiv, lxxxviii, xc, 3, 7, 8, 28–30, 44–50

FFT Fast Fourier Transform xi, xliv, li, lxxiv, lxxv, lxxviii, 4, 5, 24, 26, 32

FIFO First In First Out 10, 15–18, 31, 32

FIR Finite Impulse Response xi, 18, 19, 49

FPU Floating Point Unit 32, 47

GPL GNU General Public License 4

GPU Graphics Processing Unit lxxxix, xc, 31, 32, 47

HMM Hidden Markov Model lxxxix, 4

HNR Harmonic Noise Ratio xliv, lxxiv, lxxv

HOS High Order Statistic xii, xlvi, xlvii, lvi

IIR Infinite Impulse Response xi, 20, 49

JNI Java Native Interface 7, 9, 28, 30, 39

JVM Java Virtual Machine 8, 9, 32, 37, 47

LADSPA Linux Audio Developer's Simple Plugin API 3, 5

LGPL GNU Lesser General Public License 3, 4

LP Linear Prediction xlviii, xlix, lxxi, 28

LPC Linear Prediction Coding lxxx

LRT Statistical Likelihood Ratio Test xii, xlix, l, lvi, lxxxviii

LTSD Long-term Spectral Divergence xii, xxxviii–xl, xlii, xliii, lvi

LTSE Long-term Spectral Envelope xxxviii, xlii, xliii

LV2 LADSPA version 2 3

MCR MATLAB Compiler Runtime 30, 32

MFB Mel Filter Bank li, lii

NCCF Normalized Cross -Correlation Function xii, lxxvii–lxxix, lxxxiv

NDK Native Development Kit 4, 5, 7, 9

NLP Natural Language Processing 4

OSF Order Statistic Filter lxxv

PAR Periodic -Component to Aperiodic -Component Ratio xliv

PARADE Periodic -Component to Aperiodic -Component Ratio Based Activity Detection lvi

PCB Printed Circuit Board xxiii

PD Pure Data 5

PDA Pitch Determination Algorithm xii, xxxii, xlv, lxv, lxvi, lxix, lxxi–lxxiii, lxxvii, lxxix, lxxxii–lxxxvii, xc

PID Project Initiation Form viii, xviii

PMF Probability Mass Function li, lii

PRAAT PRAAT - Doing Phonetics by Computer Boersma and Weenink [2011] lxxv, lxxxiv, lxxxviii

PTDB-TUG Pitch Tracking Database from Graz University of Technology xxxii, lxxxii, lxxxiv

RAPF Rhetorical Acoustic Phonetic Feedback Device xvi, xviii, xxvi, 1, 29, 30, 39–42, 44, 45, 50

RAPT Robust Algorithm for Pitch Tracking lxxix, lxxx, lxxxiv

ROC Receiver Operating Characteristic lix, lxiv

RTBlocks Real-Time Block Processing Framework viii, xi, 1, 3–5, 7–10, 14, 18, 19, 26, 28, 30, 31, 33–35, 39–42, 44–50

SDK Software Development Kit 4, 9

SNR Signal to Noise Ratio xii, xlix, l, lvii, lviii, lxiv, lxxxiv

SOS Second Order Section xi, 20, 21

SPSC Lab Signal Processing and Speech Communication Laboratory of Graz University of Technology xxxii

SR Speaking Rate xxxii

SSNR Segmental Signal to Noise Ratio xii, lvii, lviii, lxii, lxiii, lxxxvi, lxxxvii

STFT Short -Term Fourier Transform xlvii

T0 Fundamental Period lxv, lxxi, lxxiii–lxxv, lxxix, lxxxii

TIMIT Acoustic-Phonetic Continuous Speech Corpus xxix, xxxii, liv, lvi, lxiii, lxiv

UML Unified Modeling Language xi, 13, 14

VAD Voice Activity Detection xi–xiii, xxvii, xxxii, xxxiii, xxxv, xxxvii, xxxviii, xli–xliii, xlvi–lv, lix–lxiv, lxxxviii, xc, 3, 28, 29, 39–43, 47–50

VBO Vertex Buffer Object 31, 32

VM Virtual Machine 3, 37, 47

ZCR Zero -Crossing Rate xxxvi, xxxvii

1 Introduction

1.1 Motivation

Voice and speech education have a long history. Many tools where developed to increase the learning curve and support both, trainers and trainees. Most of these tools require sufficient education in linguistics and phonetics.

The project *RAPF* targets the development of speech analysis and training tools for everyone (see appendix section II). The main goal of this project is a wearable device which provides live feedback for different speech features. One major topic of the project is converting a MATLAB high level algorithm design into device specific code for mobile platforms. This can be a time-consuming and error-prone procedure. The new real-time block processing framework (RTBlocks) simplifies this step by hiding complexity, providing the same code base for different target platforms and unifying the implementation procedure for different programming languages. The main goal of the framework is to run signal processing algorithms on the Android OS (Google [2012]) as well as on all other targets which can execute Java byte code.

1.2 Overview

This document is structured as follows: In the current chapter a short overview of speech analysis algorithms, the prototype's target platform and related libraries and applications is given. The design and implementation of RTBlocks are discussed in chapters 2 and 3. Details about the framework usage are outlined in chapter 4. The result of the work is summarized in chapter 6.

1.3 Speech Analysis

In Rabiner [2007, page 3] a complete speech processing chain is described, from speaker's message formulation to message understanding by a listener. Depending on the type of application, digital speech processing can support any of those steps. The complete speech processing chain is shown in figure 1.1, application types for digital speech processing are shown in figure 1.2.

Speech analysis is one of the key tasks for many digital speech processing applications. In Rabiner [2007, pages 19 - 22] a simple speech production *source/system* model for sampled speech signals is explained. It mainly consists of an excitation generator and a linear system. The source/system model is shown in figure 1.3. More advanced speech models are described in Furui [2001, page 28] and Marshall and Sicuranza [2006].

Short-term speech analysis assumes that the speech model parameters change slowly and can be assumed as constant within short time intervals.

Many applications require *real-time processing*: The system must guarantee response within a

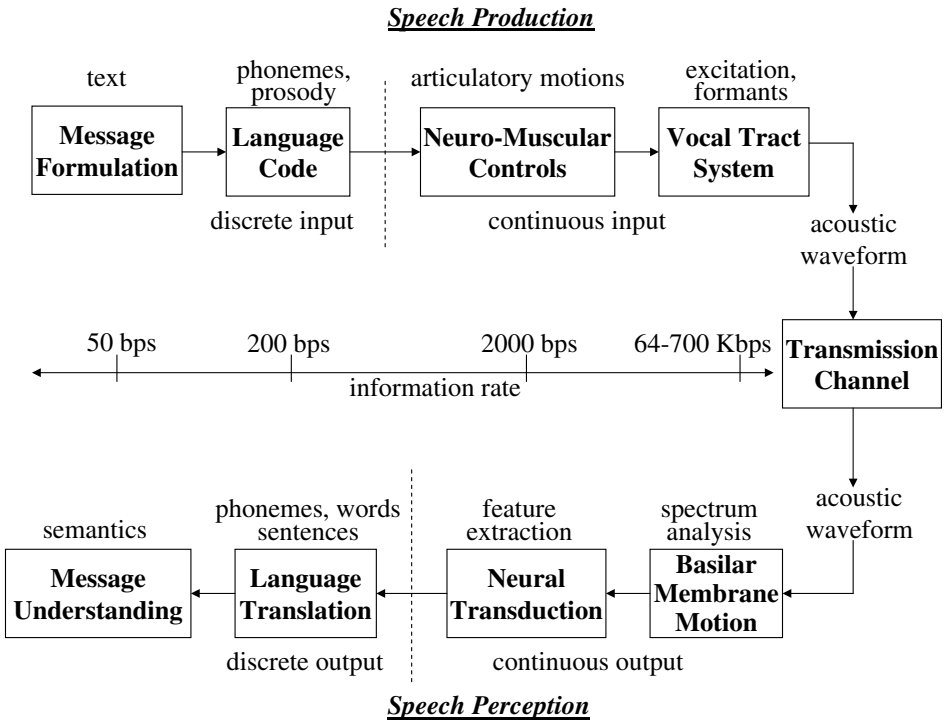


Figure 1.1: The image shows a complete speech chain as described in Rabiner [2007, page 3].It includes all important parts between producing and perceiving speech, from the message formulation in the brain of the talker to the creation of speech to the message understanding by the listener. It also includes hints about the required band width required for transmitting the information.

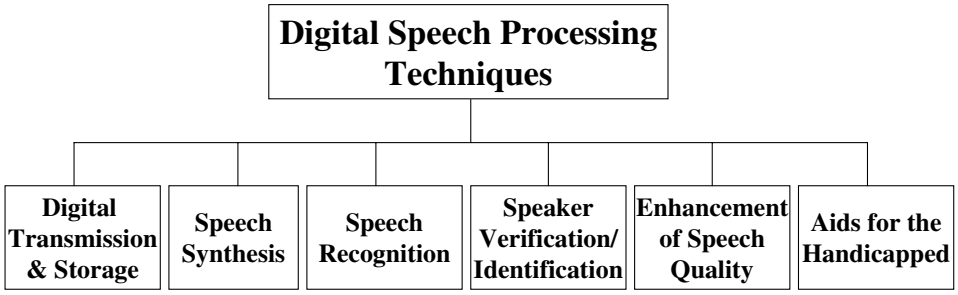


Figure 1.2: The image (taken from Rabiner [2007, page 13]) shows different types of speech applications. It includes areas of speech transmission and storage, speech synthesis, speech recognition, speaker verification and identification, speech enhancement and aids for handicapped.

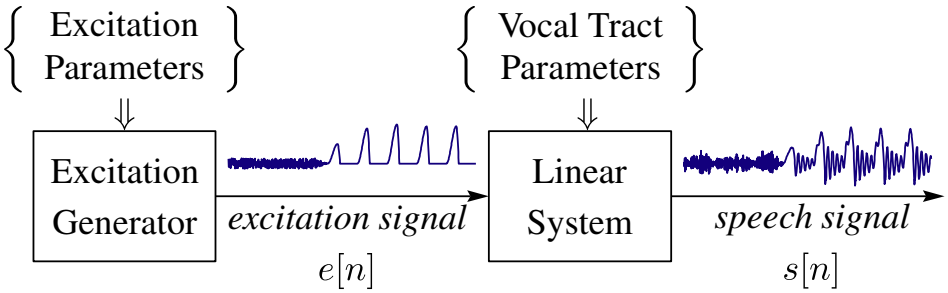


Figure 1.3: This image (taken from Rabiner [2007, page 21]) shows a simple speech production model. The excitation generator models the different sound generation modes of the vocal tract. The model of the vocal tract tube simulates the signal shaping by a discrete-time time-varying linear system. For small time intervals the vocal tract model can be approximated by a time-invariant system function. See Rabiner [2007, page 21 ff.] for further details.

certain time period (Ben-Ari [1990]). Global estimation approaches are not possible for this kind of applications.

Mobile speech analysis and *embedded speech analysis* add further constraints, most often related to performance, energy limitations and difficult environmental conditions. Basic considerations for embedded speech analysis are listed in Sinha [2010].

Nowadays, the advance of smartphones offers rich and cheep development and prototyping environments for mobile speech analysis. Real-time problems for the Android OS on mobile devices are evaluated in Mongia and Madisetti [2010].

The main algorithm design environment for the current project is MATLAB. An introduction to speech processing with MATLAB and many practical examples are given in McLoughlin [2009]. The target platform for the current project is the *Samsung Galaxy Nexus (GT-I9250)*, which uses Google's Android operation system and the Dalvik virtual machine (VM) (Google [2012]). This implies Java as main programming language and C / C++ for special purposes. Android's high speed computation engine *Renderscript* (Guihot [2012]) is not used to keep the code operation system independent.

1.4 Related Work

In this section, related libraries and applications are noted without any claim to completeness. Many ideas and concepts of these software packages served as a first draft for RTBlocks. To demonstrate the flexibility of the framework, some parts of the introduced software packages are directly integrated in RTBlocks.

1.4.1 Acoustic Phonetics Laboratory Software

Transcriber is used to create orthographic transcriptions of audio records. It supports hierarchical annotation layers, speaker lists and much more. Transcriber is written in Tcl/Tk with C extensions and is open source (Audiotranskription Transcriber [2011]). Orthographic transcriptions can serve as reference signals for voice activity detection (VAD).

PRAAT is designed especially for phonetic speech analysis. It is open source, written in C / C++ and includes numerous analysis algorithms (Boersma and Weenink [2011]).

WaveSurfer is an audio editor for acoustic phonetic researcher. It is written in Tcl/Tk and open source (Sjőlander and Beskow [2010]).

1.4.2 Sound Processing Standard and API for Plugin Development

Linux audio developer's simple plugin api (LADSPA) is a linux standard for handling audio filters and effects (Furse et al. [2000]). It is written in C, cross platform and licensed under the GNU Lesser General Public License (LGPL).
Disposable soft synth interface (DSSI) extends the plugin system by instrument bindings (Cannam et al. [2010]).
The successor of these two standards is LADSPA version 2 (LV2) (Harris et al. [2009]). A competitor standard is *Steinberg's Virtual Studio Technology (VST)* which mainly has the same features.

1.4.3 Frameworks and Libraries

MATLAB is one of the most popular numerical computing environments for audio research purposes. It is cross-platform, has numerous toolboxes and has been developed by MathWorks. An important extension for MATLAB is Simulink, which is used for modeling and analyzing dynamical systems (MathWorks [2011a]). For automatic code generation the toolboxes *Simulink Coder*^{TM}, *MATLAB Coder*^{TM} or the *Embedded Coder*$^{®}$ can be used. Open source alternatives to MATLAB are Octave (Eaton et al. [2012]) and Python with special extension packages like *SciPy* (Jones et al. [2012]), *NumPy* (Oliphant [2007]) and *matplotlib* (Hunter [2012]).Well known speech and audio processing toolboxes for MATLAB are the VOICEBOX (Brookers [2011]), *PsySound3* (Cabrera et al. [2007]), *MIRtoolbox* (Lartillot et al. [2008]), *COLEA* (Loizou [2003]) and *GLOttal Analysis Toolbox (GLOAT)* (Drugman [2012]).

With the Android Developer Tools (ADT) *Google* provides a complete framework for Android software development. The Android Software Development Kit (SDK) is the core part of the framework (Android SDK [2012]). It includes all required parts for developing and debugging Java applications for Android. The Android Native Development Kit (NDK) provides an mechanism for integrating C and C++ code within applications (Android NDK [2012]). The Android Accessory Development Kit (ADK) is designed for building and integrating hardware accessories for Android (Android ADK [2012]).

CMU Sphinx is a collection of speech recognition and modeling libraries developed at Carnegie Mellon University (Walker et al. [2004]). Besides real-time speech recognition it includes several speech analysis algorithms, acoustic hidden markov models (HMMs), n-gram statistical models and others. Current releases include a full featured version, written in Java, and a mobile version, written in C.

FFTW is a fast and often used Discrete Fourier Transform (DFT) library, developed at the Massachusetts Institute of Technology (MIT) (Frigo and Johnson [2005]). It is written in C and released under the GNU General Public License (GPL). The latest versions include various extensions for co-processors and special processor architectures as well as great cross compiling support. Another DFT library which is written in pure Java is *JTransforms* (Wendykier [2011]). It is open-source, supports multi-threading and implements one, two and tree-dimensional fast fourier transform (FFT) calculations.

FFmpeg is an audio and video processing library collection, licensed under the LGPL and the GPL. It includes the audio/video codec library *libavcodec* and is designed for cross-platform usage (FFmpeg [2012]). A fork library collection of FFmpeg is *Libav*, which has similar features (Libav [2012]).

Modular Audio Recognition Framework (MARF) is a collection of natural language processing (NLP) algorithms. It supports many audio formats and implements several algorithms for filtering, speech feature extraction and classification. MARF is highly modular and extensible, written in Java and open source (Mokhov [2008]).

Sound Object (SndObj) is an object oriented framework for audio processing. It includes many algorithms for audio production and analysis. SndObj is open source, cross-platform and written in C++ (Lazzarini [2000]).

Sound eXchange (SoX) is a command line tool for audio conversion and manipulation. It is cross-platform, written in C and open source. SoX provides many features for recording, editing

and analyzing audio and speech sounds (Bagwell and Norskog [2012]).

Pure data (PD) is a graphical data flow programming language for real-time data processing applications. It is cross-platform, written in C and open source (Puckette [1996]). *Functional AUdio STream* (FAUST) is a signal programming language especially useful for generating audio plugins for e.g. LADSPA or PD (Orlarey et al. [2012]).

1.4.4 Android Applications for Speech Analysis

Google Play is the main web store for android applications. Several developers offer software packages via it's free web site `https://play.google.com/store`. *Amazon* is an alternative supplier for Android applications. A few Android applications for speech analysis are mentioned in this section, without any claim to completeness.

The user *idroidbot* offers several microphone signal analysis applications, like real-time FFT (*iSpectral*) and real-time spectrogram (*iGram*). Another supplier for real-time audio recording and analysis applications is *NAND MOBILE* who also provides limit free versions. The applications *Spectrogram* from the user *CactusMonkey*, *SpectrogramOne* from *Simplecode.com* and *SimpleSpectrogram* from *MusicalSoundLab* are other spectrograph applications which mainly provide the same features.

Mixed Bit provides an application called *Speech Trainer* which immediately replays recored audio parts. It is designed for supplying pronunciation training.

Peter Meijer's application *The vOICe for Android* converts live video streams into audio streams which model the surrounding area. The main idea is to replace the visual sense by the sense of hearing, which might be useful for visually challenged people.

1.5 Conclusion

With MATLAB a sufficient high level development environment for speech analysis algorithms is provided. Application development for Android using Java or C++ is supported by ADT. With extension libraries like MATLAB Coder C++ reference implementations of algorithm parts can be directly extracted from MATLAB scripts. The extracted code can be integrated within a Java application skeleton using the Android NDK.

MATLAB scripts which should be automatically converted to C++ code must be carefully designed. The redesign of complex MATLAB algorithms for automatic code generation could require more effort than manual implementation may do.

To ease the implementation and verification of short-term audio processing algorithms for Android the new framework RTBlocks combines several concepts and supports different methods for converting MATLAB code to Android applications. The current technical possibilities are extended by the new framework to speed-up and improve these time-consuming and error-prone steps.

2 RTBlocks Design

The main purpose of this section is to answer the question: "Why is there another framework for audio processing required?". The motivation is outlined in section 2.1, features are described in section 2.2, programming languages, target platforms and supported algorithm types are discussed in the sections 2.3, 2.4 and 2.5.

2.1 Motivation

The design of speech analysis algorithms usually starts with a mathematical description. If the theoretical evaluation leads to sufficient results, a high level implementation of the algorithms will be used to verify the initial assumptions. If the expectations are not fulfilled, the process will start all over again. These algorithms may include global estimation approaches and will be based on assumptions or limitations regarding the signal's characteristics. MATLAB is a widely used scripting environment for modeling and testing of speech algorithms (McLoughlin [2009]).

In most cases, the algorithm design and the related high level implementation must be adjusted and re-evaluated to deal with constant audio streams. A common representation of such streaming applications is the *graphical data flow block processing diagram.*

MATLAB can generate standalone executables, so one may stop the design process here. The design of embedded systems, wearable devices, advanced systems, and similar applications requires further implementation steps. These steps usually include the conversion to a compiled language like C, C++ or Java, or even to a data flow language like VHDL and Verilog. Complex systems are difficult to convert using automatic conversion tools (Muelleger [2008], Vikstroem [2009]). Even if the tools are getting better and better (Nyrenius and Ramstroem [2011]), this final implementation step requires great effort and is error-prone. Often MATLAB scripts must be adapted for automatic conversion because not all programming constructs are allowed and not all MATLAB functions are supported by the conversion tools yet. Manual conversion of specific parts or even of whole systems is still required.

RTBlocks supports converting the high level design into a data flow block design by hiding the data stream functionality, block connections and data pagesize differences from the programmer. To ease the final implementation step there are a high level version in MATLAB's scripting language and a compilable version in Java available for all important framework parts. Both, the MATLAB and the Java version of a certain implementation can be tested in an unique MATLAB test bench. The integration of automatic generated C or C++ code from MATLAB scripts is supported by using the Android NDK and the Java native interface (JNI).

Figure 2.1 shows a common fundamental frequency (F0) estimation algorithm block diagram, the structure of the MATLAB high level design and the related implementation using RTBlocks. Converting the block diagram to the MATLAB high level design is an intermediate step which can be skipped in most cases.

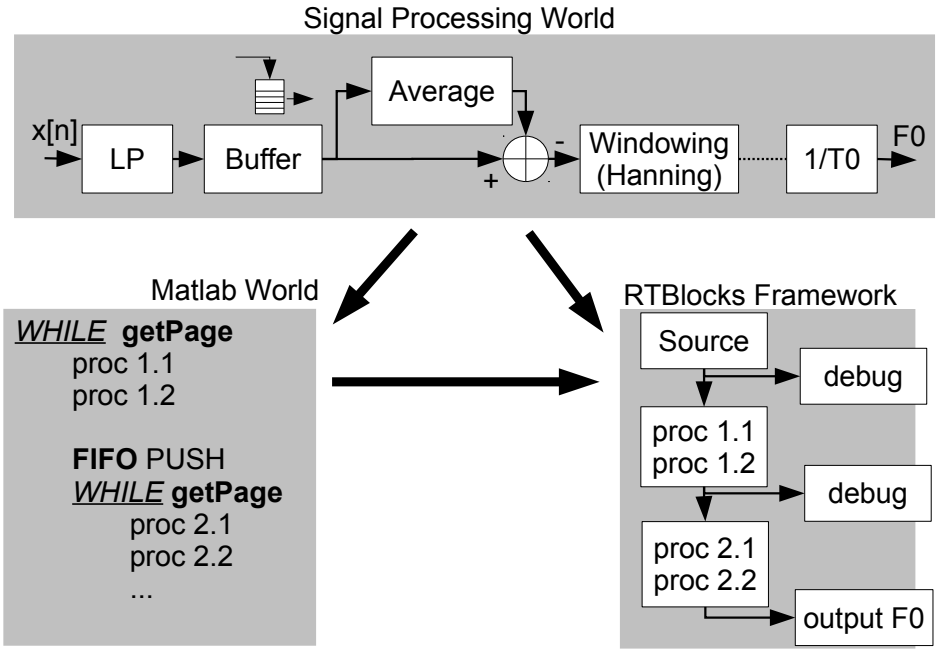


Figure 2.1: Three different representations of the same algorithm: The "top level represenation" of a common F0 estimation algorithm is shown as graphical block diagram (top part of the image). Such block processing systems can be implemented by nested loops as outlined in the left lower part of the image. High level implementations based on nested loops or even graphical block diagrams itself can be implemented in MATLAB or Java using the RTBlocks framework (lower right part of the image). This "low level implementation" directly reflects the graphical block diagram if the algorithms and implementation steps are well designed.

2.2 Features

The main features of the RTBlocks framework are:

- Cross platform algorithm implementation support; tested for MATLAB, OpenJDK Java virtual machine (JVM), Dalvik JVM
- Short framework training period, easy to use
- Direct mapping between graphical data flow block processing diagrams and implementation code
- Support for connecting blocks with different page sizes
- Multi-threading support
- 32 bit and 64 bit data width

The current project includes a smartphone prototype with Android OS and Dalvik JVM. Since Java is the recommended programming language for most Android applications (Ableson et al. [2012]), it is chosen as the main target for the RTBlock framework. Furthermore Java is known by the slogan "write once, run anywhere" (Sun [2012]), so all platforms which can execute the JVM can be targeted.

2.3 Programming Languages

Three different programming languages are used:

MATLAB is used for algorithm design and the block processing representation. A RTBlocks version for MATLAB exists.

Java is used as implementation language, which is recommended from *Google* as main programming language for android application development (Ableson et al. [2012]). A RTBlocks version and Android specific extensions exit. Android offers the Android SDK for application development and verification.

C/C++ is used by many third party libraries and for algorithms with excessive pointer arithmetic. For the MATLAB version of RTBlocks simply MEX files and MEX build scripts are used. For the Java version of RTBlocks the C/C++ integration is done by the JNI. Android offers additional build scripts and cross-compiling tools for Dalvik JVM. These tools are bundled together to the Android NDK.

2.4 Target Platforms

Both, non-mobile and mobile target platforms for prototypes are evaluated. The main differences are discussed in this section.

Non-Mobile High Performance Platforms: The main representatives of this targets are common desktop PCs. Power or performance issues usually do not appear. These platforms are also suitable for algorithm development and verification. Common integrated development environments (IDEs) like like MATLAB and Eclipse for Java can be used.

Mobile and Low Performance Platforms: Wearable platforms, smartphones and embedded systems are better suitable for applications used in everyday life. Most of this platforms have significant limitations compared to non-mobile platforms. These limitations have to be considered during the whole algorithm design process. Furthermore, many mobile platforms lack in debugging capabilities. By using the Java version of RTBlocks, mobile and non-mobile platforms share the same code base and therefore can be tested in common Java IDEs until platform-specific parts are added. This reduces development time and error-proneness of the implementation step.
In this context, laptops have similar features compared to common desktop PCs so they are not viewed as mobile platforms.

2.5 Supported Data Types

RTBlocks supports all short-term analysis algorithms. Multi-dimensional data arrays must be linearized. All data is stored in 32-bit single-precision or 64-bit double-precision IEEE 754 floating point format(IEEE 754-2008 [2012]). This ensures close coupling between Java and MATLAB due MATLAB is based on the same numerical format.
The major drawback of Java is that Java neither supports pre-processor macros nor generic primitive arrays for adjusting the bit size for different target platforms. To overcome this restriction the Java core implementation of all blocks uses double-precision. A Python script is provided to extract the single-precision version of the framework.

2.6 Source-Filter-Sink Concept

RTBlocks defines three structural elements, namely *source blocks*, *filter blocks* and *sink blocks*. All elements are derived from a common *base block*. Each block represents a processing step of an algorithm or a block of a graphical data flow block processing diagram. Blocks of the framework are implemented by classes.

2.6.1 Block Categories

Base Blocks: Short-term processing blocks behave more or less all the same way: A data page with a certain page size is taken from the input port, is buffered, modified and the result is supplied to the following processing blocks. The execution of the algorithms is triggered by the input data: If data is available, the block is executed. Class names of this block type have the prefix *RTBlock*.

Source Blocks: Blocks which provide data from another source than the input port are called *source blocks*. By definition these blocks define a blocking main execution method named *run_(...)*. Class names of this block type have the prefix *RTSource*. If the execution is triggered by the input port rather than by the *run_(...)* method, the block is not a source block even if data is obtained from a foreign source.

Filter Blocks: Filters are simply blocks which have the same input and output page size. Class names of this block type have the prefix *RTFilter*.

Sink Blocks: Sinks does not need to have the same input and output page size. It is also possible that blocks of this kind have others blocks connected to their output port. Class names of this block type have the prefix *RTSink*.

2.6.2 Block Ports

In graphical block designs, blocks are connected by directed lines. Data is only transferred from the *starting point* to the *endpoint* of the connection. Each block defines a single connection endpoint, but several connection starting points.

In RTBlocks connections are not separate data channel classes: They are represented by the elements' interconnections. Each block defines a method called *addSink(...)* for connecting one or several following elements to the current block. All blocks connected by this method are processed in parallel.

Blocks can have connected sub-blocks. By default sub-blocks are called before the data is processed by the current block, but this predefined behavior can be altered individually. Each block defines a method called *addFilter(...)* for connecting one or several sub-elements to the current block. All blocks connected by this method are processed in series.

2.7 Data Processing Flow

Data in RTBlocks is processed per page. If a source block has enough data, the connected sub-blocks and following blocks (sinks) are triggered. Each triggered block stores the data page in an internal first in first out (FIFO) buffer. When enough data is collected in the internal buffer, the

block itself calls it's sub-blocks, processes the data and triggers it's connected following blocks. Figure 2.2 shows a simplified data flow chart of a single block processing step.

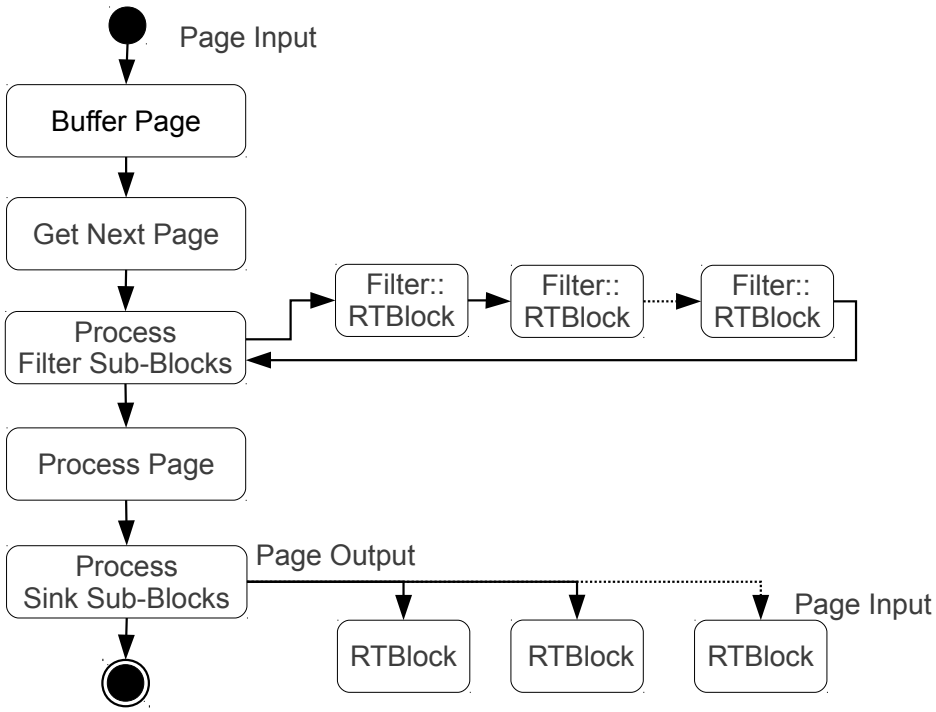


Figure 2.2: Data processing steps for each block: If enough data is available, the block reads out a single data page from the input buffer and triggers connected blocks. Connected filter blocks are processed in series, the result is returned to the block. When all filter blocks are processed, the block's own processing routine is executed. Then the result is passed to all connected sink blocks.

2.8 Graph Representation

Simple connected block structures can be represented by hierarchical tree structures with linked nodes. If more than one source is provided or blocks with more than one parent are used, the structure will be represented by directed acyclic graphs (Thulasiraman and Swamy [1992]).

2.8.1 Traversals

Traversal functions describe the way of accessing and processing each graph node. Currently the *traversal functions* are included in the nodes and triggered by parent nodes. The initial trigger is generated by the source blocks. Several traversal exits:

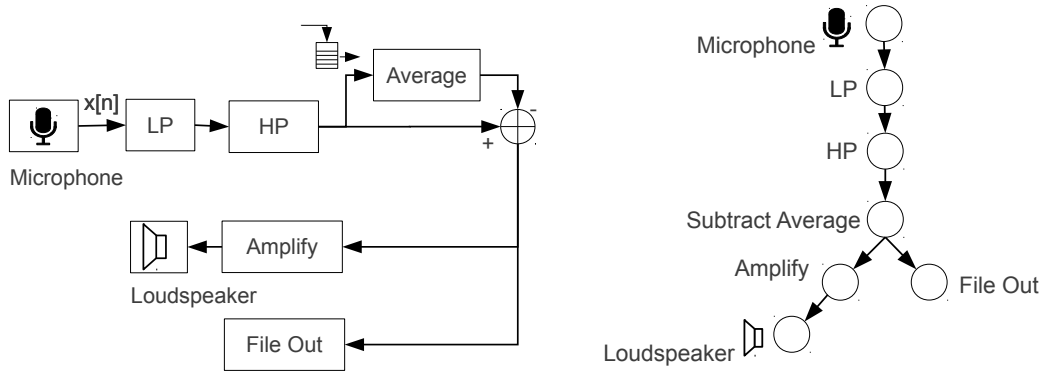
Initialization: All blocks need to be initialized before any processing can be done. If the root is initialized, all children will also be initialized.

Finalization: When all data is processed cleanup routines are called.

Processing: If enough data is available, the sources will start the processing traversals. The traversals stop at blocks which do not have enough available output data.

The applied traversal starts at the top node, recursively moves down the tree or graph structure and renders the left nodes before the right ones. Left nodes are added before right ones. Figure

2.3 shows a simple graphical block design of an audio streaming application and related graph structure.



(a) Block Diagram (b) Graph Structure (Tree)

Figure 2.3: The image outlines two different graphical representations of a simple amplification algorithm: On the left side the algorithm is shown as block diagram, on the right side the same algorithm is shown as tree or graph structure. When thinking in terms of trees or graphs rather than in terms of block processing diagrams algorithm implementations can be generalized using basic concepts of graph theory (Thulasiraman and Swamy [1992]).

3 RTBlocks Implementation

This section describes main implementation considerations for the framework, important blocks and the mathematical background of the implemented algorithms.

3.1 Base Class

All implemented processing blocks are derived from one single base class named *RTBlock*. This class is a ready-to-use filter or sink block and an abstract class for source blocks. Several programming hooks are implemented - methods, which are called on important data flow positions. The class *RTBlock* includes the base functionality of all processing blocks:

- Buffering input data
- Calling connected blocks in the initialization phase, processing phase and finalization phase
- Setting and obtaining block parameters like page sizes or block names
- Applying simple consistence and error checks

Functionality is added by extending the base block using class derivation. Methods which are provided for extending the base functionality are called *hooks*. Any of these hooks as well as the base processing methods can be extended or replaced. An reduced unified modeling language (UML) diagram of the base class is shown in figure 3.1. The call sequence of the programming hooks is shown in figure 3.2: When a block executes all attached filter or sink blocks it triggers the main *process* method of these blocks. When a block executes it's initialization, data path flushing or finalization method, it also triggers the related methods of the attached filter or sink blocks.

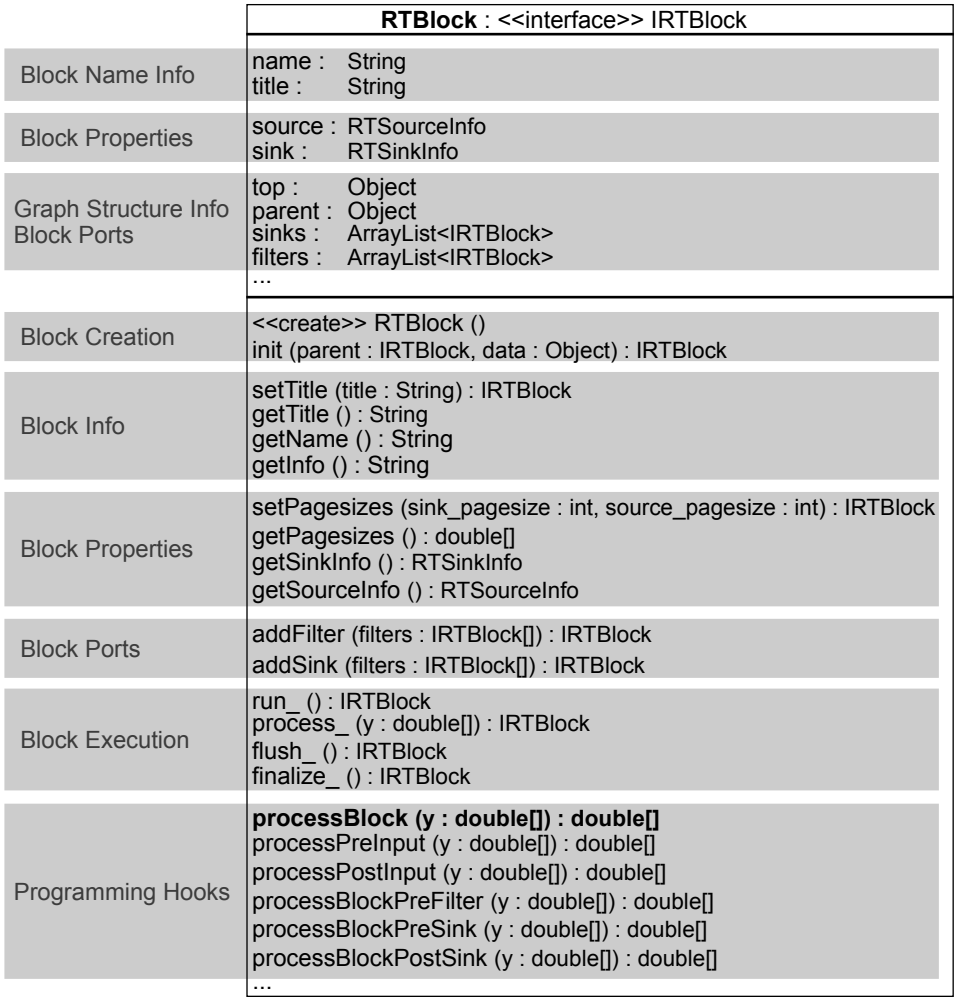


Figure 3.1: The image shows the most important parts of the base class for the RTBlocks framework in an UML class diagram. The class properties contain several categories: basic block informations like the block name and block title (Block Name Info); common block properties like source page size and sink page size (Block Properties); structural information like the parent block, the top most block and attached filters and sinks (Graph Structure Info/Block Ports); Each category contains methods for setting and retrieving the related data. Furthermore methods for block creation and initialization (Block Creation), for block processing (Block Execution) and for extending the block's functionality (Programming Hooks) are provided. The main method for adding functionality is processBlock(...).

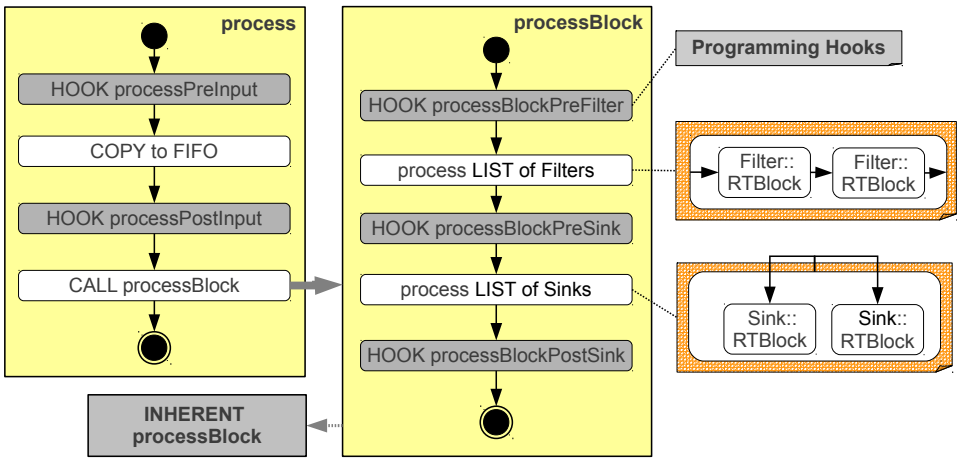


*Figure 3.2: The dataflow digram of a block for a single processing step is split into three parts: The main part **process** is the top routine for data processing. It is invoked by the parent block, copies data into the internal FIFO buffer, calls hooks and the core execution routine for page processing (**processBlock**). The page processing routine itself contains again several hooks for extending the base functionality of blocks. It calls all attached filter blocks in series and all attached sinks in parallel.*

3.1.1 Basic Processing Methods

The base class defines *five main methods* which are used to add functionality:

init(...): Blocks need to be initialized before any data processing is done. If blocks are interconnected only the initialization method of the root block will be called manually. The initialization methods of all connected child blocks are called automatically.

run_(...): This method is implemented by source blocks. Data processing is started by invoking the run method of the root block.

process(...): This method is triggered by the parent of a block or by the run method. The main purpose of this method is to push input pages to the internal FIFO buffer and to call the programming hooks. If enough data is available, the internal data manipulation method *processBlock(...)* will be called.

processBlock(...): This is the main data manipulation and data distribution method. The base implementation of this method simply passes input pages to the attached filters and sinks. Most blocks implement their data processing algorithms by extending this method.

finalize_(...): Cleanup functionality goes here. If a block is finalized, the finalization method of all connected blocks will be called recursively. Unlike the Java build-in finalization method this method must be manually invoked.

3.1.2 Programming Hooks

The base class defines *several programming hooks* for adding functionality or altering the base behavior of the block. These programming hooks are:

processPreInput(...): This method is called before the input data page is pushed to the block's FIFO buffer. All page size checks are done afterwards. The output page size of this method must be the same as the specified input page size of the block but the input page size can differ from it. This programming hook should only be used if none of the others are sufficient.

processPostInput(...): This method is called right after the input data page is pushed to the block's FIFO buffer. The main purpose of this method is program debugging and data verification.

processBlockPreFilter(...): This method is called before the attached filters are processed. The method must not change the data page size. The page size of the processed data is the same as the output page size of the block.

processBlockPreSink(...): This method is called before the attached sinks are processed. The method must not change the data page size. The page size of the processed page is the same as the output page size.

processBlockPostSink(...): This method is called after all attached sinks are processed. It is mainly used for debugging and performance measurements.

The call sequence of the programming hooks is shown in figure 3.2.

3.2 Block Overview

Currently about 40 different blocks are implemented and tested, ranging from simple filters to complex speech analysis algorithms. This section outlines block instantiation and important blocks, categorized by their functionality.

3.2.1 Block Instantiation

An *abstract block factory* is automatically generated by a Python script (see abstract factory design pattern in Gamma et al. [1995]). By using the factory pattern platform specific blocks as well as different implementations of the same functionality can easily be instantiated according to the different target systems without changing the core implementation of the algorithms.

3.2.2 Data Stream Manipulation

Complex algorithms often use different data page sizes or different data step sizes. Furthermore the block structure may change during runtime. In this section special block purposes for these problems are outlined.

Page size changes will easily be done by the base block *RTBlock* if different input and output page sizes are used. Input data is pushed in the internal FIFO buffer. If enough samples are available output pages are popped and processed until too few samples are in the buffer. Beside the base block most derived blocks also allow page size changes.

Port extensions are used for complex systems. This can be done simply by inserting the base block *RTBlock* at a certain data flow position. The passing data can be modified by the filter port and measured by the sink port of the inserted base block.

Placeholder are often used if algorithms are modified during runtime or certain block implementations are not currently available on specific platforms. By inserting the base block *RTBlock* the system implementation can be completed even if not all required blocks are implemented or tested.

Overlapped pages are used to change the default step size of one page. This functionality is implemented by *RTSink_OverlappedFrames*.

Upsampling is often used to reduce the analytical error of algorithms. This functionality is implemented by *RTSink_Upsampling*.

3.2.3 Data Sources

Source blocks provide streaming data from data sources which are foreign to the system. All these blocks implement the outer processing loop method called *run_(...)*.

Reading data from a microphone is implemented by the *RTSource_Microphone* block. Microphones are one of the most important data sources for speech processing applications. The block reads a data page from the microphone interface and triggers all connected processing blocks. When all blocks are finished, the source block waits until the next page is provided by the microphone interface. If the processing step takes too long input data will be lost.

Reading data from a file is implemented by the blocks *RTSource_Wavread* and *RTSource_Pcmread*. Currently only 8 kHz sampling frequency, single channel, 16 bit short value samples, and the byte order "big endian" is supported.

Reading data from a buffer is implemented by *RTSource_FromBuffer*. The data buffer must be filled before the *run_(...)* method of this source can be executed .

3.2.4 Data Sinks

Generally speaking, data sinks are used for communicating signal processing results. Currently RTBlocks implements methods for saving data to volatile and non-volatile storage, for visualizing and for replaying audio.

Replaying sound is implemented by the *RTSink_Loudspeaker* block. Java does not have a great native audio input/output support which means that this block is platform-dependent.

Writing data to a file is implemented by *RTSink_ToWavFile* and *RTSink_ToPcmFile*. Currently only 8 kHz sampling frequency, single channel, 16 bit short value samples and the byte order "big endian" is supported.

Writing data to a buffer is implemented by *RTSink_ToBuffer*. The data is stored in an internal buffer and can be read back during or after executing the system. This block may cause an "out of memory" exception if the algorithm tries to store too much data in the buffer.

Displaying data is implemented by *RTSink_Plot*. The data is stored in a FIFO buffer and plotted as a *x-t diagram*. MATLAB already contains several functions for plotting. RTBlocks for Java provides two different plotting methods, one based on *JFrame* and one on *OpenGL*. JFrame is not designed for Android, so RTBlocks for Android adds another plotting class based on *AndroidPlot* (AndroidPlot [2011]). The OpenGL version is designed to run on both, non-mobile and mobile devices.

3.2.5 Data Filtering

Several different signal filter algorithms are implemented, ranging from linear and nonlinear filters to statistical analysis filters. This section provides a short filter block overview.

A low pass finite impulse response (FIR) filter is implemented by the *RTFilter_LowpassFir* block. This block realizes a direct form FIR structure (see Oppenheim [1999, page 367]). The example implemenation uses a cutoff frequency of 500 Hz and a sampling frequency of 8 kHz. The filter output is calculated by

$$y[n] = \sum_{k=0}^{K} b_k \cdot x[n-k] \tag{3.1}$$

where $x[n]$ is the input at sample position n, K the filter order, $y[n]$ the filtered signal and b_k the k^{th} filter coefficient. Figure 3.3 shows the signal flow diagram of the implemented FIR structure.

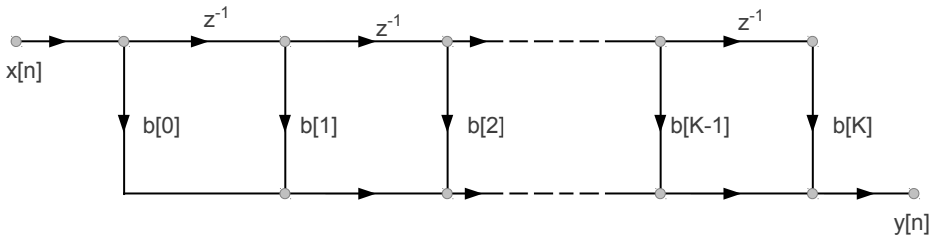


Figure 3.3: The images shows the implemented FIR filter structure as signal flow diagram. The filter coefficients b[k] for specific filters are calculated with MATLAB. The RTBlocks test bench ensures the correct implementation of the filter structure.

The filter coefficients b_k are calculated by

$$b_k = b[k] = h[k] \cdot w[k] \tag{3.2}$$

$$h[k] = \frac{sin(2\pi \frac{f_c}{f_s} k)}{\pi k} = 2\frac{f_c}{f_s} \cdot sinc(2\frac{f_c}{f_s} k) \tag{3.3}$$

$$w[k] = 0.62 - 0.48 \cdot \left| \frac{k}{K} - 0.5 \right| + 0.38 \cdot cos\left(2\pi \left(\frac{k}{K} - 0.5\right)\right) \quad 0 \leq k \leq K \tag{3.4}$$

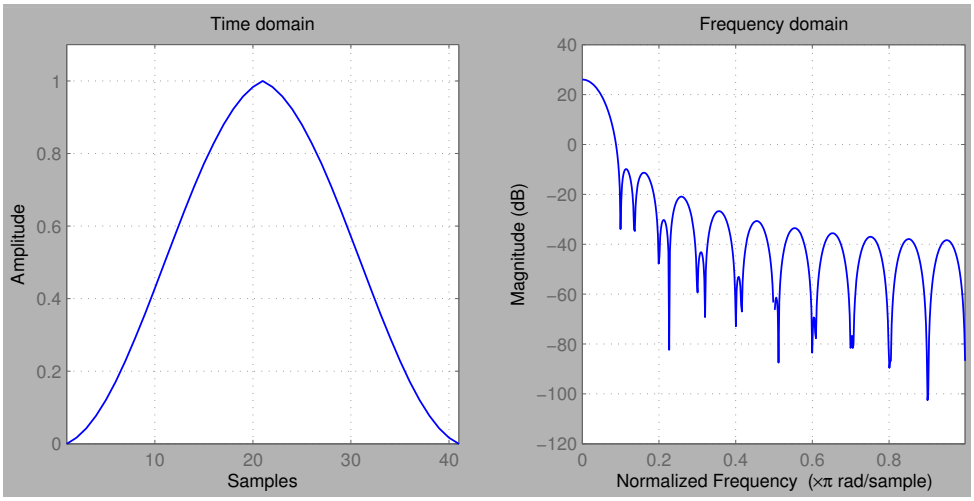
where $h[k]$ is the impulse response of the ideal time-discrete filter at sample position k, f_c the desired cutoff frequency of the low pass filter, and f_s the sampling frequency. To convert the ideal filter to an FIR filter the impulse response is windowed by a *Modified Bartlett-Hanning window function* with finite impulse response (Oppenheim [1999]): w is the time-discrete window function, and K is the filter order. The FIR filter coefficients are stored in $b[k]$. The window coefficients and the FIR filter coefficients are calculated by the MATLAB scripts *barthannwin.m* and *fir1.m*. The direct form I (DF1) and the direct form II (DF2) of FIR filters lead to the same result (Oppenheim [1999]). Figure 3.4 shows the window function and the designed filter.

High Pass infinite impulse response (IIR) filtering is implemented by the block *RTFilter_HighpassIir*. It uses a transposed DF2 second order section (SOS) structure, a sampling frequency of 8 kHz, a stop band frequency of 15 Hz, a passband frequency of 25 Hz, a stop band attenuation of 80 dB and an allowed passband ripple of 1 dB. The frequency bands for the filter design are shown in figure 3.5, the signal flow graph of the implemented filter is shown in figure 3.6. The filter coefficients are calculated with *MATLAB's Filter Design Toolbox* by the scripts *fdesign.m*, *design.m* and *dfilt.df2tsos* (MathWorks [2011b, Signal Processing Toolbox, fdesign.highpass, dfilt.df2tsos]) which apply the bilinear transform and the *Butterworth* approximation and the conversion to the transposed SOS filter structure (Oppenheim [1999, page 439 ff.]). The basic structures for IIR filters are described in Oppenheim [1999, page 354 ff.].

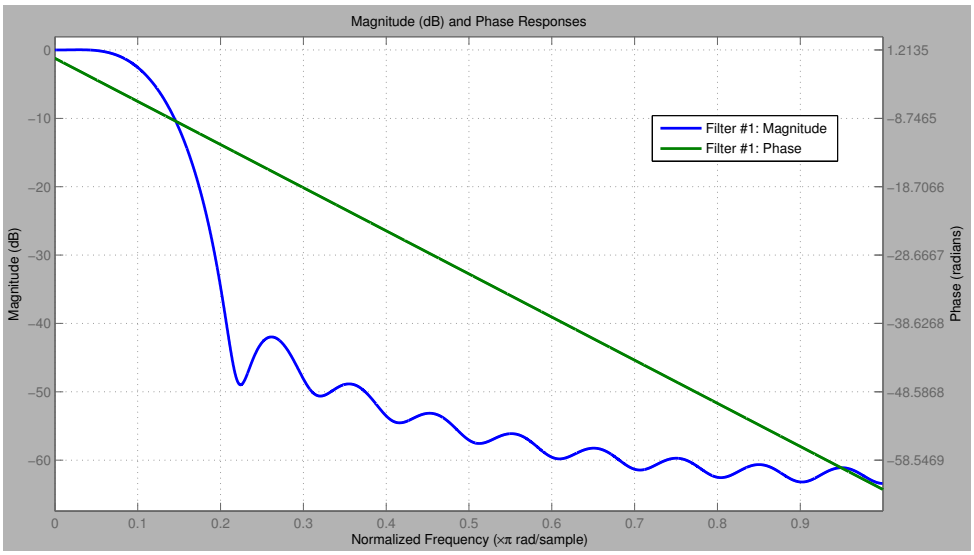
Hanning windowing (Oppenheim [1999, page 468]) is implemented by the block *RTFilter_Hanning*. The data page is multiplied by the window function $w[n]$ to reduce the windowing influence and to put more weight on samples which are located around the page center. The window function $w[n]$ is defined by:

$$w[n] = \begin{cases} 0.5 \cdot \left[1 - cos\left(\frac{2\pi n}{N-1}\right)\right] & 0 \leq n \leq (N-1) \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

where N is the data page size.



(a) Modified Bartlett-Hanning Window Function

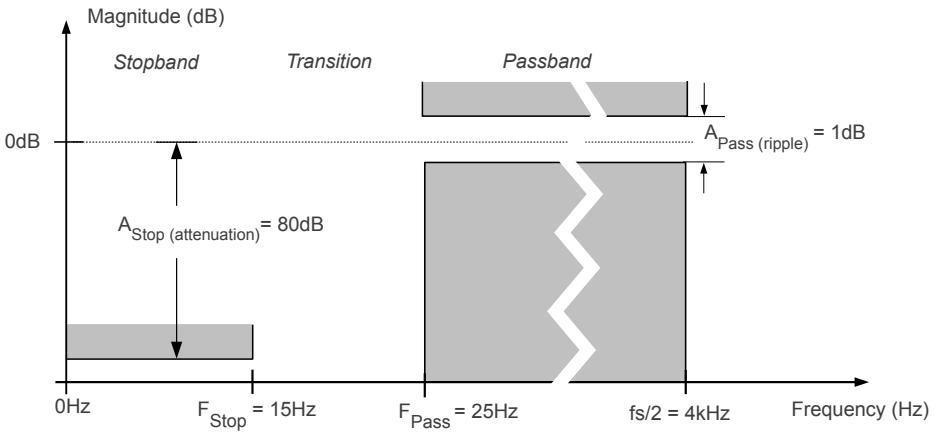


(b) Magnitude and Phase Response of the Designed Low Pass Filter

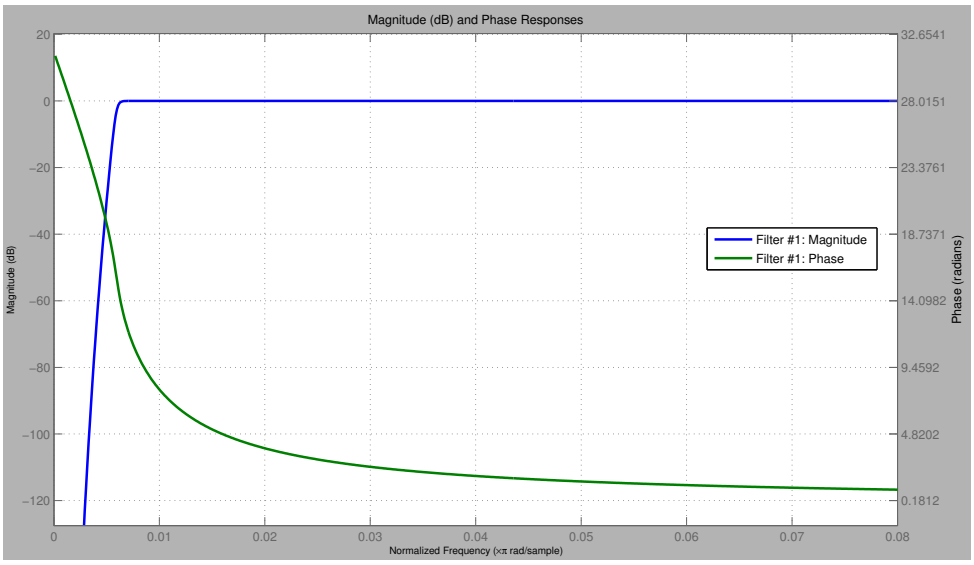
Figure 3.4: The image shows the impulse response of the "Modified Bartlett-Hanning window function" with finite impulse response (left upper corner), its magnitude squared Fourier transform (right upper corner), and the magnitude and phase response of the designed low pass filter. The filter is designed with an order of 40 or 41 coefficients, a sampling frequency of 8000 Hz, and a cutoff frequency of 500 Hz. The design leads to a linear phase filter and a constant group delay of $\frac{N-1}{2Fs} = 2.5ms$.

Median filtering (Pitas and Venetsanopoulos [1990, page 63 ff.]) is implemented by the *RTFilter_Median* block. The median filter simply returns the observation which separates the higher half from the lower half ordered data samples. For odd window length N the result is equals the k^{th} order statistic where $k = N/2 + 0.5$. This filter is especially useful for noise reduction and outlier removal. Further details about median filtering of speech signals are described in Rabiner and Schafer [1978, page 158 - 161].

Center clipping (Rabiner and Schafer [1978, page 151 ff.]) is provided by the *RTFilter_CenterClip*



(a) Frequency Response Specification for the High Pass Filter (Stopband, Transition, Passband)



(b) Magnitude and Phase Response for the High Pass Filter

Figure 3.5: The image shows the filter design specifications in an frequency response diagram (a) and the filter design result as magnitude and phase response diagram (b) for a high pass filter with a stop band frequency of 15 Hz, a passband frequency of 25 Hz, a stop band attenuation of 80 dB, an allowed passband ripple of 1 dB and the sampling frequency of 8000 Hz.

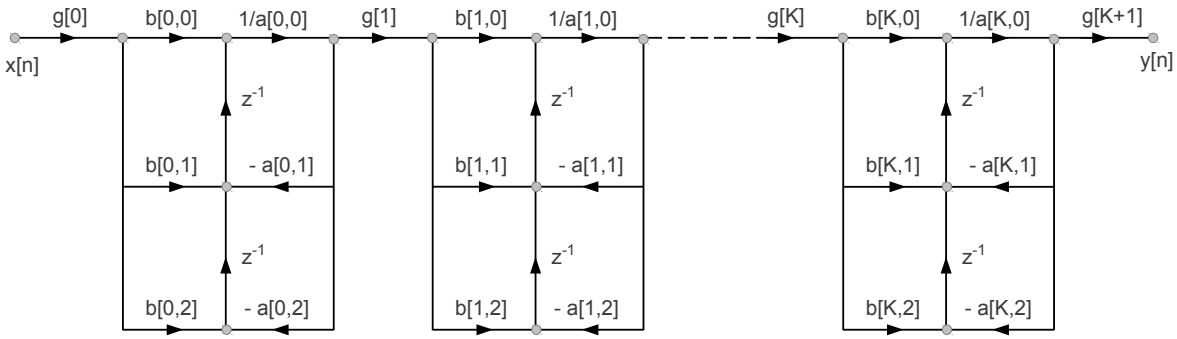


Figure 3.6: The image shows the transposed SOS filter structure as described in Oppenheim [1999, pages 365 - 366]. If all sections of the cascade are stable, the whole filter is stable. The transposed DF2 form implements the zeros first and then the poles which leads to less quantization errors in finite-precision digital implementations.

block. All samples below a certain limit are suppressed:

$$y[n] = \begin{cases} x[n] & |x[n]| > k_l \cdot max(|x[i]|) \quad 0 \leq i < K, \quad 0 < k_l \leq 1 \\ k_s \cdot x[n] & \text{otherwise} \end{cases} \tag{3.6}$$

where $x[n]$ are the input samples, $y[n]$ are the filtered samples, and K is the data page size. The current center clipping block uses a limit gain k_l of 0.5 and a suppression factor k_s of $\frac{1}{5}$. In figure 3.7 the transfer function of the block and an example center clipping step is shown.

Auto-correlation function (ACF) is implemented by the *RTFilter_Xcorr* block. The block computes the *unbiased* ACF as described in MathWorks [2011b, section xcorr: cross correlation]:

$$r_{xx}[m] = \begin{cases} \frac{1}{N-|m|} \cdot \sum_{n=0}^{N-m-1} x_{n+m} \cdot x_n^* & m \geq 0 \\ r_{xx}[-m] & \text{otherwise} \end{cases} \tag{3.7}$$

The lower symmetrical part of the correlation is discarded to provide same input and output page size. For fast calculation the FFT based algorithm described in Oppenheim [1999, page 746] is used.

Bilateral filtering is a non-linear data-dependent filter method which uses two Gaussian filter kernels. The filter was originally designed for edge preventing denoising, filtering and joining images. A detailed description is given in Paris et al. [2009]. The filter was also successfully tested for sound denoising Matsumoto and Hashimoto [2009]. Figure 3.8 shows a common audio filtering process using bilateral filtering. The block *RTFilter_Bilateral* implements a brute force bilateral filter algorithm for speech enhancement:

$$y[n] = k[n] \cdot \sum_{i=n-\lfloor M/2 \rfloor}^{n+\lceil M/2 \rceil} x[i] \cdot c(|n-i|) \cdot g(|x[n]-x[i]|, n) \tag{3.8}$$

$$k[n] = \left(\sum_{i=n-\lfloor M/2 \rfloor}^{n+\lceil M/2 \rceil} c(|n-i|) \cdot g(|x[n]-x[i]|, n) \right)^{-1} \tag{3.9}$$

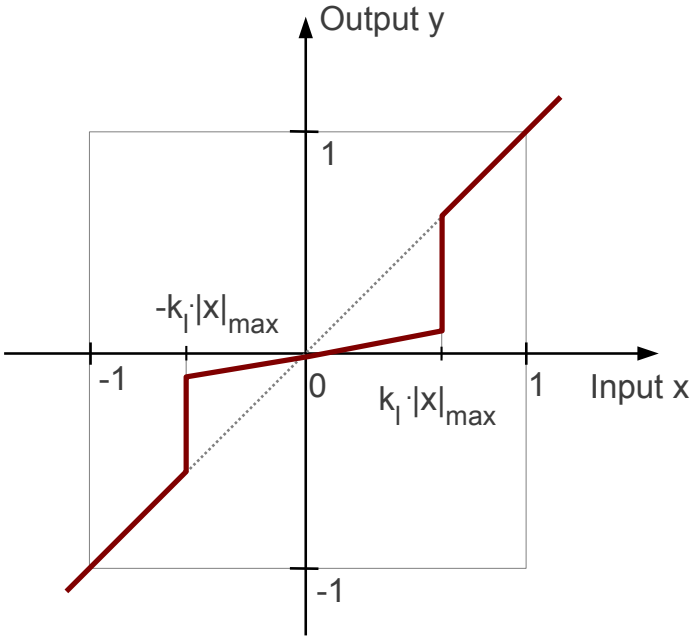
$$c[d_n] = \mathrm{e}^{-\frac{d_n^2}{2 \cdot \sigma_n^2}} \tag{3.10}$$

$$g[d_x, n] = \mathrm{e}^{-\frac{(k_x[n] \cdot d_x)^2}{2 \cdot \sigma_x^2}} \tag{3.11}$$

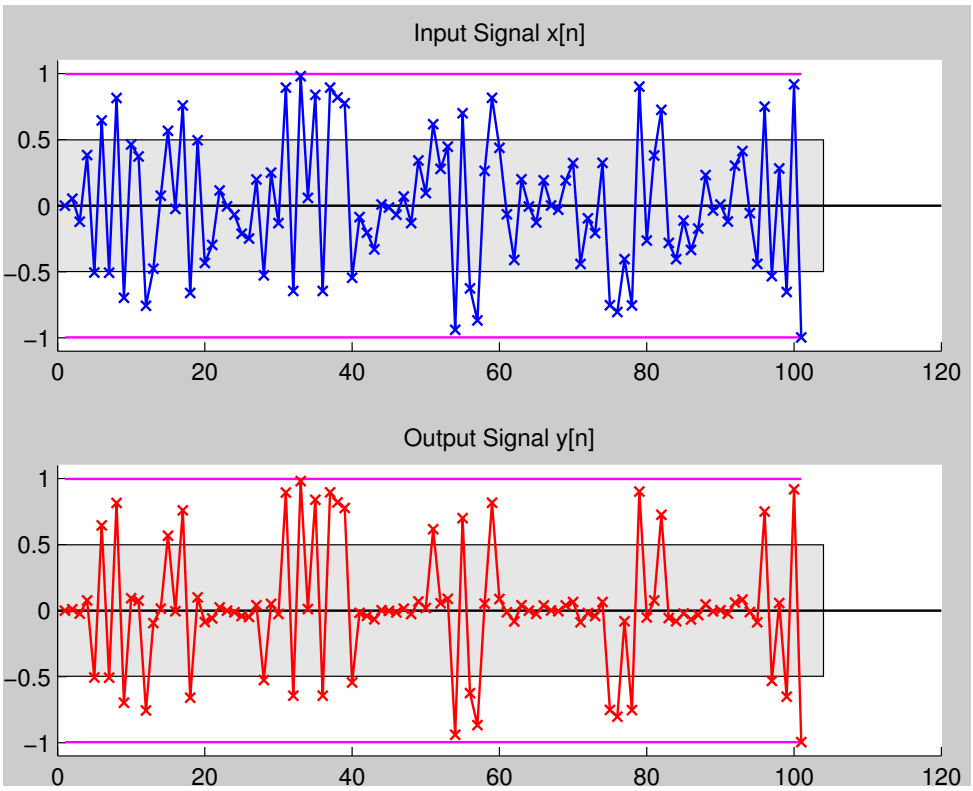
$$k_x[n] = (max(x[i]) \cdot M)^{-1} \qquad n - \lfloor M/2 \rfloor \leq i \leq n + \lceil M/2 \rceil \tag{3.12}$$

where $x[n]$ is the speech input at position n, M the spatial filter size, $k[n]$ the filter normalization factor, $c[d_n]$ the Gaussian weight function for spatial distances d_n, $g[d_x, n]$ the Gaussian weight function for the amplitude distance d_x and $k_x[n]$ the amplitude normalization factor for the filter kernel calculation.

To avoid critical scaling of the Gaussian weight function for spatial distances a dynamic a



(a) Transfer Function of the *RTFilter_CenterClip* Block.



(b) Applied Center Clipping (Input and Output Signals).

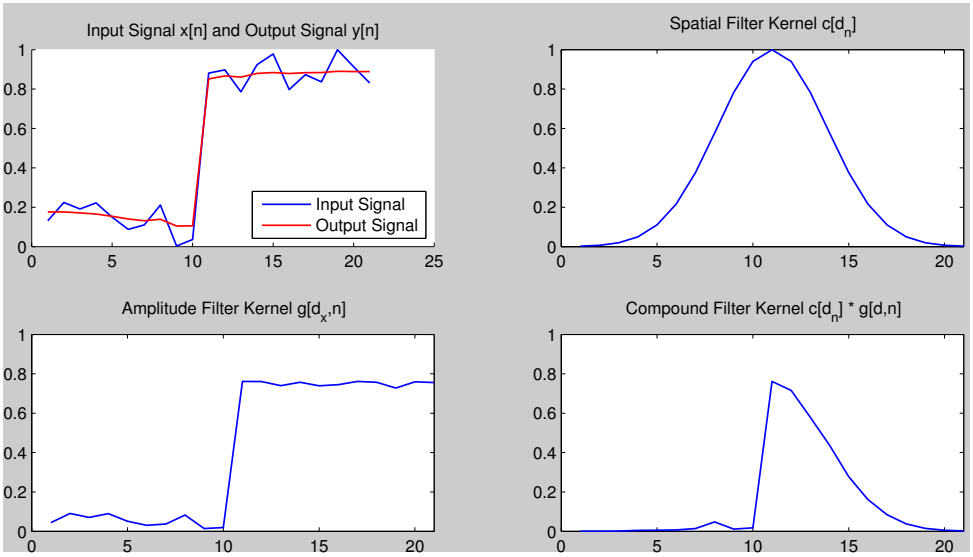
Figure 3.7: The block RTFilter_CenterClip implements the center clipping transfer function shown in (a). Signal parts with an amplitude lower than a certain limit are attenuated but still available at the output. The impact of this nonlinear filter is shown in (b).

range compression function can be added:

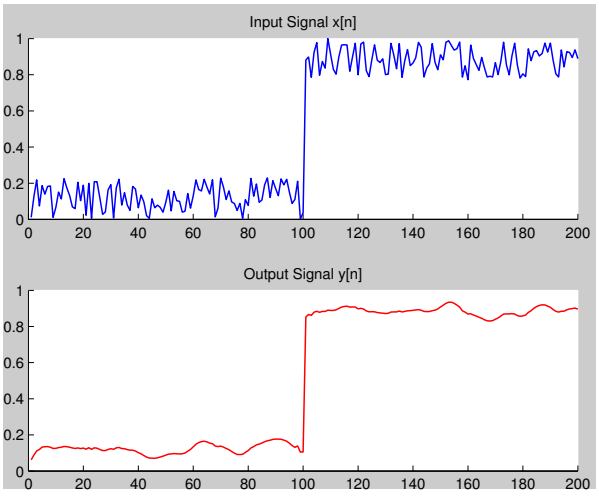
$$g[d_x, n]' = tanh(\mathrm{e}^{-\frac{(\cdot d_x)^2}{2\cdot\sigma_x^2}}) \tag{3.13}$$

$$tanh(x) = \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}} \tag{3.14}$$

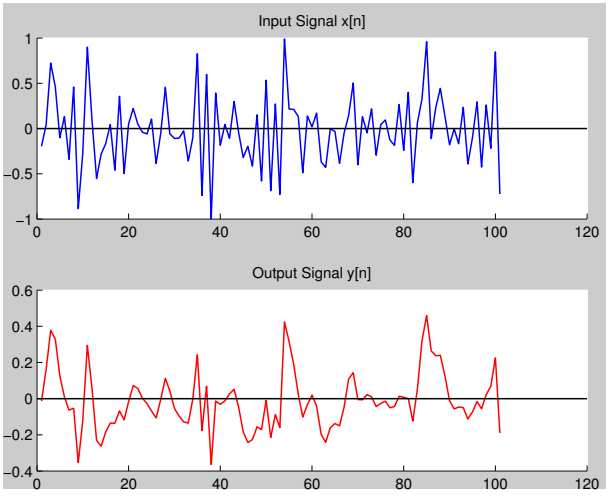
where the tangens hyperbolicus function *tanh* is used to compress the dynamic range of the Gaussian weight function $g[d_x, n]'$. See appendix sections K.2.2 and K.3.7 for more details.



(a) Filter Kernels for a Single Filtering Step



(b) Filtering A Noisy Step Signal



(c) Filtering Compound Noisy Sinus Signals

Figure 3.8: The image shows an example for bilateral filtering of time series. Small signal derivations are smoothed, huge signal edges are preserved, as shown in (b). The filter kernel must be re-calculated for each input signal which leads to significant computational effort. To overcome the problem of too high signal delay introduced by this calculation the filter can be parallelized.

Automatic gain control (AGC) is an important filter for real environment audio applications.

A very simple version of AGC is implemented by the block *RTFilter_AutomaticGainControl*:

$$y[n] = x[n] \cdot k[n] \tag{3.15}$$

$$m[n] = max(|x[n-m]|) \quad 0 \leq m < M \tag{3.16}$$

$$k[n] = \begin{cases} m_{max}^{-1} & m[n] > m_{max} \\ m[n]^{-1} & m_{min} \leq m[n] \leq m_{max} \\ m_{min}^{-1} & m[n] < m_{min} \end{cases} \tag{3.17}$$

where $x[n]$ is the input and $y[n]$ the output at sample position n, $k[n]$ is the gain factor, and $m[n]$ the moving absolute maximum value for a certain window M. The gain limits are specified by m_{min}^{-1} and m_{max}^{-1}.

Amplitude spectrum: Returns the absolute values of the FFT spectrum calculated over one data page. This block is implemented by *RTFilter_AbsFFT*. Figure 3.9 demonstrates the implemented block behavior.

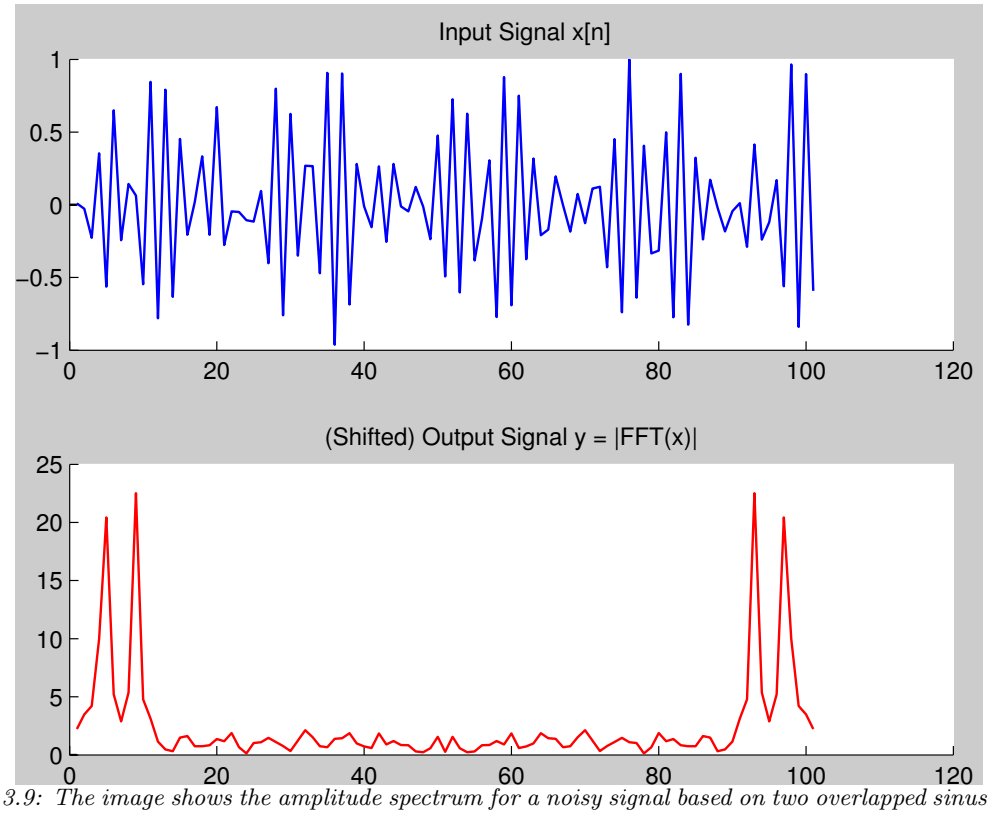


Figure 3.9: The image shows the amplitude spectrum for a noisy signal based on two overlapped sinus signals, calculated by the block RTFilter_AbsFFT. RTBlocks for MATLAB uses the MATLAB's build-in FFT algorithm, RTBlocks for Java uses the pure Java DFT library JTransforms (Wendykier [2011]), and RTBlocks for Android uses FFTW (Frigo and Johnson [2005]), cross-compiled and optimized for Samsung Galaxy Nexus GT-I9250.

Mean subtraction: Subtracts the mean value, calculated over one data page:

$$y[n] = x[n] - \overline{x} \tag{3.18}$$

$$\overline{x} = \frac{\sum_{i=0}^{N} x[i]}{N} \tag{3.19}$$

where $x[n]$ is the input signal of the current page for sample position n, $\overline{x}$ the average of the current page, N the page size and $y[n]$ is the related output signal.

This block is implemented by *RTFilter_SubtractMean*.

Normalization: Divides all samples by the maximum of the current data page.

$$y[n] = \frac{x[n]}{x_{max}} \tag{3.20}$$

$$x_{max} = max(|x[i]|) \qquad 0 \leq i < N \tag{3.21}$$

where $x[n]$ is the input signal of the current page for sample position n, x_{max} the maximum amplitude value of the current page, N the page size and $y[n]$ is the related output signal.

This block is implemented by *RTFilter_Normalize*.

Moving average: Returns the moving average of the data samples. The data page size of the block does not influence the results.

$$y[n] = \overline{x[n]} \tag{3.22}$$

$$\overline{x[n]} = \frac{\sum_{i=n}^{n-M} x[i]}{M} \tag{3.23}$$

where $x[n]$ is the input signal of the current page for sample position n, $\overline{x[n]}$ the moving average value, M the average window size and $y[n]$ is the related output signal. To simplify the implementation the average window is not centered around the current sample position n.

This block is implemented by *RTFilter_MovingAverage*.

Signal delay: Delays the input data for certain samples:

$$y[n] = x[n - n_d] \tag{3.24}$$

where $x[n]$ is the input at sample position n, n_d is the integer delay value and $y[n]$ is the related output sample.

This block is implemented by *RTFilter_Delay*.

Signal gain: Multiplies the input samples by a certain constant gain:

$$y[n] = k \cdot x[n] \tag{3.25}$$

where $x[n]$ is the input at sample position n, k is the constant gain factor and $y[n]$ is the related output sample. This block is implemented by *RTFilter_Gain*.

Signal's dynamic range compression: Limits the input signal to ± 1 by a certain transfer func-

tion. Currently four different transfer functions are available:

$$y_{clamp} = \begin{cases} -1 & x \leq -1 \\ x & -1 < x < 1 \\ 1 & x \geq 1 \end{cases} \tag{3.26}$$

$$y_{tanh} = tanh(x) \tag{3.27}$$

$$y_{tanh\_tanh} = k_0 \cdot tanh(tanh(x)) = \frac{1}{tanh(\infty)} \cdot tanh(tanh(x)) \tag{3.28}$$

$$y_{atan} = k_1 \cdot atan(x) = \frac{1}{atan(\infty)} \cdot atan(x) \tag{3.29}$$

where x is the input signal and y_{clamp}, y_{tanh}, y_{tanh_tanh} and y_{atan} are the output signals for the selected transfer function. The different transfer functions are shown in figure 3.10. The block implements a *memoryless system* as described in Oppenheim [1999, page 18].

This block is implemented by *RTFilter_DynamicCompression*.

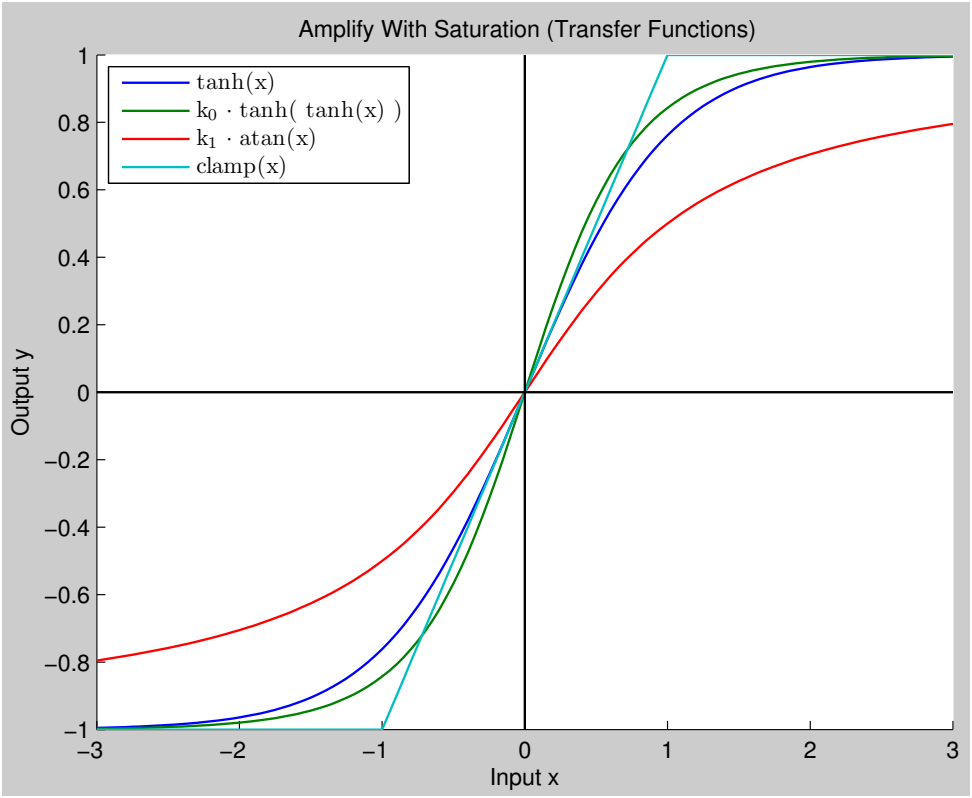


Figure 3.10: The image shows the different transfer functions for dynamic range compression which are included in RTBlocks. The implemented block is a memoryless system which means it does not introduce any signal delay.

Absolute value calculation: Returns the absolute values of the input samples:

$$y_[n] = |x[n]| \tag{3.30}$$

where $x[n]$ is the input signal at sample position n and $y[n]$ the related output signal. This block is implemented by *RTFilter_AbsoluteValue*.

3.2.6 Speech Analysis

The main purpose of the RTBlocks dataflow programming framework is the implementation of speech analysis algorithms. Currently two VAD and two F0 algorithms are implemented, each within its own block. Standard filter implementations as described in section 3.2.5 are reused, algorithm specific parts are added. Each speech analysis algorithm groups a complex RTBlocks networks together to a single *RTBlock* class. A short description of the blocks is provided in this section. Further details about the algorithms are explained in the appendix section III.

Multi-feature pattern matching VAD as described in Benyassine et al. [1997] is in implemented by *RTSink_Rapf_VadG729B*. The base VAD step calculates four speech features at each frame: Linear prediction (LP) spectrum, full-band energy, low-band energy and zero-crossing rate. Long-term energy and global threshold concepts are used to avoid possible algorithm deadlocks. The reference implementation proposed in Benyassine et al. [1997] is embedded within RTBlocks by using the JNI extension. Figure 3.11 shows a simplified block digram of the VAD block.

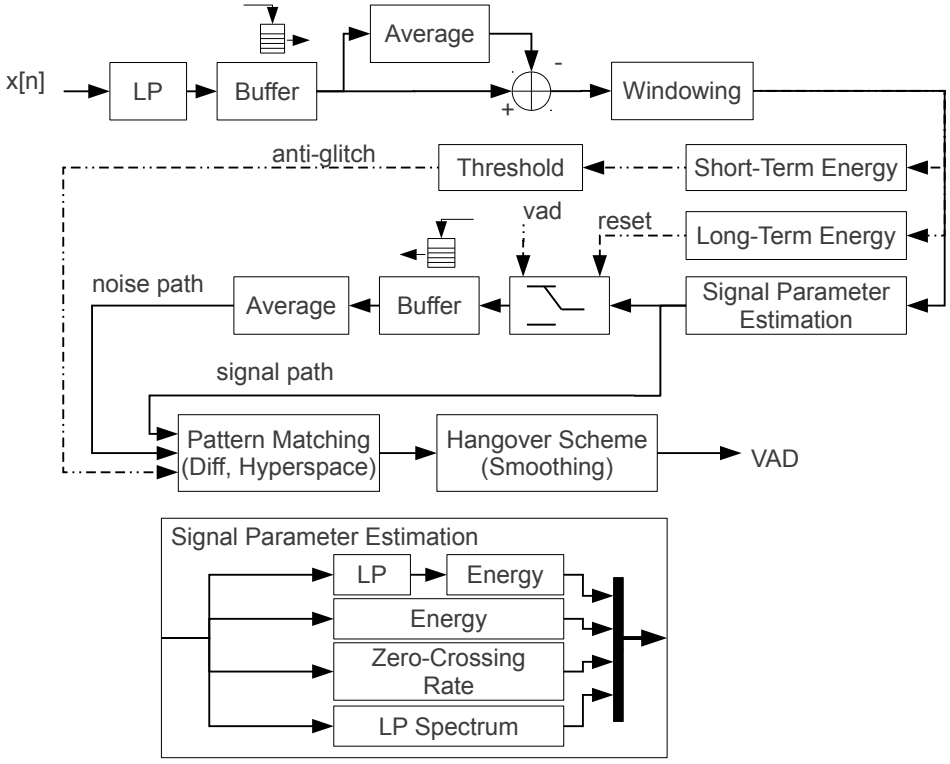


Figure 3.11: The image is taken from K.3.4 where a detailed review of the VAD algorithm introduced in Benyassine et al. [1997] is provided. The algorithm is based on multi-feature extraction, pattern matching and a hangover scheme for smoothed VAD results.

Variance based VAD as described in the appendix section K.3.7 is implemented by *RTSink_VadFromBilateral* in a simplified and real-time capable version. The core algorithm calculates the amplitude variance for a single frame. A simple VAD decision can be formed by applying a constant or variable threshold. More advanced concepts may include the signal curve of the amplitude variance, hangover schemes or plausibility checks. The core implementation of the algorithm is shown in figure 3.12.

A simple ACF based F0 is implemented by *RTSink_Rapf_F0Xcorr01*. The algorithm searches

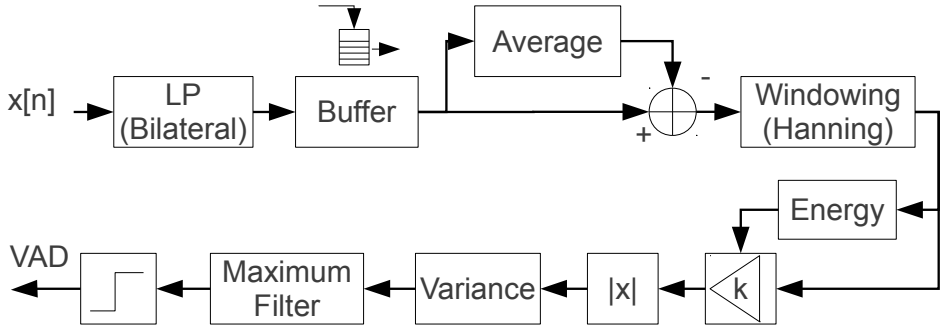


Figure 3.12: The image shows the simplified version of the variance based algorithm introduced in "RAPF Preliminary Algorithm Study" (see appendix section K.3.7). Energy based post-processing is omitted to decrease the computational effort and to calculate the result in real-time. The core feature of the algorithm is the bilateral pre-processing step which also is the most complex part of the implementation. Parallel implementation is possible to speedup the calculation.

the ACF for strong peaks next to the main correlation peak. A simplified version of the algorithm is shown in figure 3.13. A detailed description of this algorithm is provided in L.3.1.

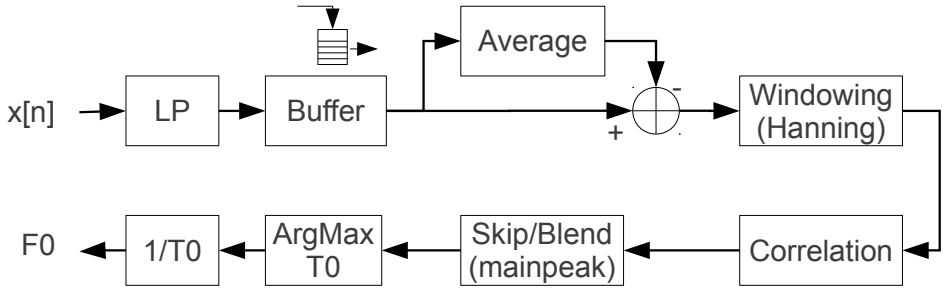


Figure 3.13: The image is taken from "RAPF Preliminary Algorithm Study" (see appendix section L.3.1). It shows a simple ACF based F0 estimation algorithm. Complex post-processing methods are omitted to keep the implementation simple and fast. If the F0 is within a certain range the algorithm leads to good estimation results, outside this range the estimation tends to fail. See appendix section L.4 for further details.

Robust F0 estimation as described in de Cheveigné and Kawahara [2002] is implemented by *RTSink_Rapf_F0Yin*. Several extensions to the ACF and distance function (DF) lead to a more robust F0 estimation. The algorithm's C implementation of *CMU Sphinx* is used by RTBlocks through the JNI interface. A detailed review of the algorithm is provided in the appendix section L.3.6.

3.3 Code Generation for Different Target Platforms

Platform specific code can be generated automatically or implemented manually. Automatic code generation with MATLAB is discussed in section 3.3.1, Java code generation is described in section 3.3.3.

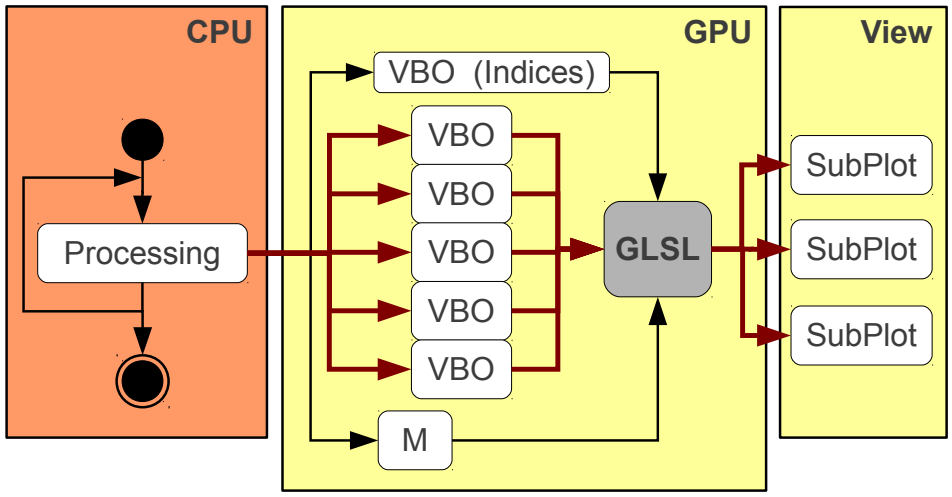
3.3.1 Automatic Code Generation

Most high level implementations of speech algorithms are done with MATLAB. *MATLAB Simulink* models, *Embedded MATLAB* code or even MATLAB scripts can be converted to C code, C++ code or program libraries. Therefore *Simulink Coder*^{TM}, *MATLAB Coder*^{TM} or the *Embedded Coder*® can be used. The convesion results do not require the matlab compiler runtime (MCR) on the target platform (MathWorks [2011b]). Automatic code generation is discussed in e.g. Nyrenius and Ramstroem [2011] and Vikstroem [2009], MATLAB compilers for C code generation are reviewed in Muelleger [2008].
Generated code may be used directly, as code base or as reference implementation. Code generation for Android is not directly supported. Nevertheless C or C++ libraries can be compiled using Android's native development kit (NDK) and accessed through the JNI.

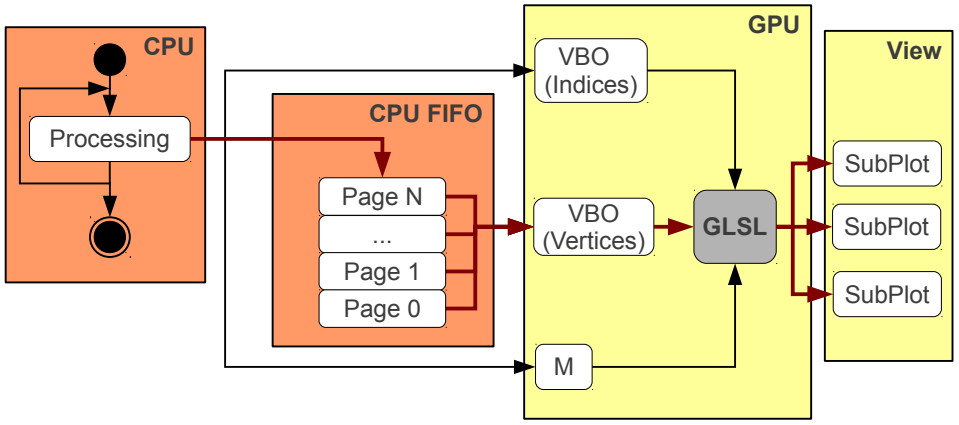
3.3.2 Cross Platform Real-Time Plotting

MATLAB integrates most important plotting functionalities which are used for RTBlocks. Java provides the application programming interface (API) *Swing* for graphical user interactions and visualizations (Hoy et al. [2003]). The core functionality of MATLAB for plotting one-dimensional time series is re-implemented in RTBlocks, based on Swing. Because Swing is not available on Android, RTBlocks for Android provides a separate plotting package based on the library *androidplot* (AndroidPlot [2011]).

The Swing and androidplot versions for plotting time series are not optimized for performance. RTBlocks provides another cross-platform plotting package which is optimized for fast time series plotting. It has limited user interaction functionality and is based on OpenGL ES (Khronos Group [2012]). The consecutively numbered time values are stored in a single vertex buffer object (VBO), the amplitude values are stored in one or more VBOs. Besides the standard plotting functionality the package provides continuous plotting with specified FIFO buffer length, multi-series plots and multi-axes plots. The FIFO buffer for the amplitude values can be placed in the graphics processing unit (GPU) memory to support modern smartphones with lower central processing unit (CPU) performance. In this case most controlling and processing parts for the FIFO buffer are done by the implemented *Shader Programs*. A general description of OpenGL is given in Wright et al. [2011], a detailed description of the OpenGL Shader Language (GLSL) is provided in Rost [2006]. The block diagram in figure 3.14 outlines the distribution of the FIFO buffer between the CPU and the GPU.



(a) OpenGL based Plotting with GPU FIFO Buffer.



(b) OpenGl based Plotting with CPU FIFO Buffer.

Figure 3.14: The image shows the distribution of the FIFO buffer between the CPU and the GPU for continuous time series plots. In (a) the FIFO buffer is placed on the GPU memory. It consists of indexable VBOs which are rendered as a single circular buffer and concatenated by a single vertex for each VBO transition. The transformation matrix M controls the rendering of the data onto a two dimensional plane. The buffer update routines and the rendering routines must be synchronized to avoid losing data. In (b) the FIFO buffer is placed in the system's memory and controlled by the CPU. Only one VBO is used for passing the amplitude data to the rendering pipeline. In both cases a separate VBO is used to store the time values. This buffer is increased as needed.

3.3.3 Java Code Generation

MATLAB does not generate Java code without MCR, which is not designed for embedded or mobile devices. Therefore all implemented Blocks exist in both, MATLAB and Java programming language. Both uses 32 bit or 64 bit double precision floating point calculations. Further extensions are added to support Android specific elements: Especially input and output blocks like recording from microphone and data plotting differs from the standard JVM versions. Due to CPU limitations the fast FFTW library (Frigo and Johnson [2005]) is used to speedup FFT and ACF calculations. The library is cross-compiled for Samsung Galaxy Nexus which has an advanced risc machine (ARM) CPU, a floating point unit (FPU) and single processing multi data (SIMD) capabilities. By cross-compiling the used algorithms can be processed in real-time.

3.3.4 Conclusion

To provide a good base for fast algorithm development the framework includes many different source, filter and sink blocks for both, MATLAB and Java. The blocks are functional identical which supports implementing MATLAB designs for Android. By enclosing processing networks into a single block complex parts can be reused like any other block. C and C++ can be integrated by using the JNI. With this method the framework can benefit from the strengths of different programming languages as well as from MATLAB's automatic code generation.

4 RTBlocks Usage

The basic design flow from algorithm design to mobile implementation using RTBlocks is explained in this chapter.

4.1 Algorithm Block Structures

The *basic structure* consists of only one single data source or several synchronized data sources. This is a common scenario for streaming applications and sufficient in most cases. If no multi-threading block except from microphone input and loudspeaker output is required, the high level design can be easily implemented in MATLAB.

The *advanced structure* enables using more than one non-synchronized data sources. Multi-threading, data joining and similar concepts are included. This structure is hard to debug and not easy to design and integrate within MATLAB.

4.2 Algorithm Design Flow

A simple and straight-forward design flow for RTBlocks based algorithm design and evaluation is shown in figure 4.1. In this section the design phases are explained by the step-by-step implementation of a *frame based mean subtraction algorithm*.

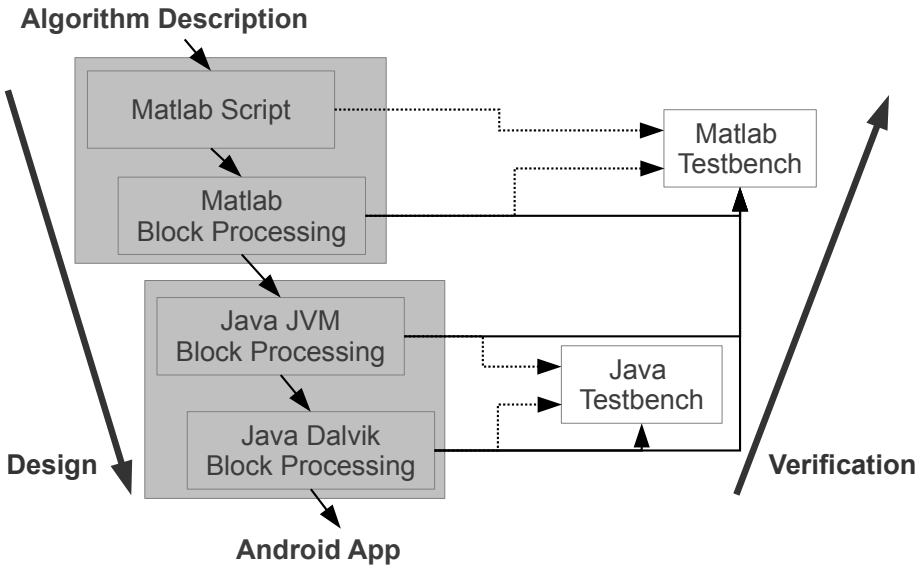


Figure 4.1: The image shows a simple design flow for RTBlocks based algorithms. Starting at the algorithm description, a MATLAB high level implementation is required. The algorithm needs to be converted to a block processing structure. Therefore the RTBlocks for MATLAB is implemented. From the high level implementation and the bock processing implemenation a test bench which is the base test bench for all further implementation steps can be extracted. Then the Java version of the algorithm needs to be implemented. Therefore the RTBlocks for Java is provided. The implemented blocks can be individually tested and compared against the MATLAB blocks. Finally Android specific parts are added. Many Android specific blocks are already provided by RTBlocks for Android.

4.3 Example Implementations

4.3.1 Example: Mean Subtraction Algorithm

An algorithm should be implemented which reads data from an audio file, splits the data into non-overlapped frames, subtracts the amplitude's mean value for each page and stores the result in another audio file. The dataflow diagram of this task is shown in figure 4.2.

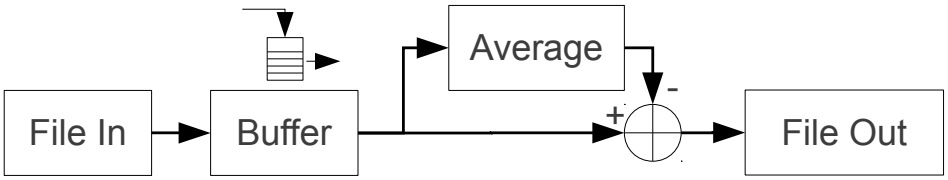


Figure 4.2: The image shows a simple mean subtraction system, which contains the data source, the buffer which converts continuous data samples into data pages, the mean subtraction stage and the data sink.

The main algorithm stage is the subtraction of the mean value, which should fulfill following equation:

$$y[n,m] = x[n,m] - \overline{x_m} \qquad 0 \leq n < N \tag{4.1}$$

$$\overline{x_m} = \frac{\sum_{n=0}^{N-1} x[n,m]}{N} \tag{4.2}$$

where $x[n, m]$ is the input sample at page m and sample offset n, N is the page size, and $\overline{x_m}$ is the mean amplitude value for page m.

A possible high level implementation for MATLAB is shown in the following listing:

```
% read data from file
[ x , fs ] = wavread( filename_in );

% split data into non-overlapped frames
x = buffer( x , framesize );

% for each frame: subtract average
av = sum( x , 1 ) / framesize;
y = x - repmat( av , size( x , 1 ) , 1 );
%...alternative: y = bsxfun( @minus , x , av );

% save data to file
wavwrite ( y( : ) , fs, filename_out );
```

The scripts *wavread.m* and *wavwrite.m* are used for reading and writing audio data. For splitting data into non-overlapped frames the script *buffer.m* is used.

A slower, but more Java-like implementation uses a loop to calculate and subtract the average for each data page:

```
% read data from file
[ x , fs ] = wavread( filename_in );

% split data into non-overlapped frames
x = buffer( x , framesize );

% for each frame: subtract average
y = zeros ( size ( x ) );
for i = 1 : size( y , 2 )
    page = x( : , i );
    av = sum ( page ) / framesize;
    y( : , i ) = page - av;
end

% save data to file
wavwrite ( y( : ) , fs, filename_out );
```

The next step is converting the MATLAB implementation into a streaming application, using RTBlocks. The MATLAB version for the task might look as follows:

```
%block for reading data from file
source = RTSource_Wavread();
source.setFilename( filename_in );
source.setPagesizes( framesize_in , framesize_out );

%block for average subtraction
filter = RTFilter_SubtractMean01();

%block for saving data to file
sink = RTSink_ToWavFile01();
sink.setFilename( filename_out );
```

```
12
13 %interconnect
14 source.addFilter( filter );
15 source.addSink( sink );
16
17 %process data
18 source.init().run_().flush_().finalize_();
```

The blocks *RTSource_Wavread* and *RTSink_ToWavFile01* are used for reading and writing audio data. Mean subtraction is implemented by the block *RTFilter_SubtractMean01*. The block is derived from *RTBlock* and overwrites it's processing method *processBlock(...)*:

```
1 function y = processBlock( obj , y )
2   % average subtraction
3   % av = sum( y ) / obj.sink.pagesize;
4   av = sum( y ) / length( y );
5   y = y - av;
6
7   % process connected filters and sinks
8   y = obj.processBlock@RTBlock( y );
9 end
```

The Java version of this application looks very similar to the MATLAB implementation:

```
1  // block for reading data from file
2  IRTSource_Wavread source = new RTSource_Wavread();
3  source.setFilename( filename_in );
4  source.setPagesizes( framesize_in , framesize_out );
5
6  // block for average subtraction
7  IRTBlock filter = new RTFilter_SubtractMean01();
8
9  // block for saving data to file
10 IRTSink_ToWavFile sink = new RTSink_ToWavFile01();
11 sink.setFilename( filename_out );
12
13 // interconnect
14 source.addFilter( filter );
15 source.addSink( sink );
16
17 // process data
18 source.init().run_().flush_().finalize_();
```

The Java version of mean subtraction uses special array utilities which reduce the gap between MATLAB vector operation functions and Java vector operations functions. The overwritten processing method *processBlock(...)* is shown in the following listing:

```
1 @Override
2 protected double[] processBlock( double[] y ) {
3   // average subtraction
4   // double av = ArrayUtils.sum( y ) / obj.sink.pagesize;
5   double av = ArrayUtils.sum( y ) / y.length;
```

```
6     y = ArrayUtils.SubtractInplace( y , av );
7
8     //process connected filters and sinks
9     y = super.processBlock( y );
10    return y;
11  }
```

If different target platforms or different block versions are expected, blocks should be instantiated by the factory rather by using *new*:

```
1  // create bock factory ('new' is only used to simplify the demonstation code)
2  IRTBlockFactory fact = new RTBlockFactory();
3
4  // block for reading data from file
5  IRTSource_Wavread source = fact.createStandardRTSource_Wavread();
6  source.setFilename( filename_in );
7  source.setPagesizes( framesize_in , framesize_out );
8
9  // block for average subtraction
10 IRTBlock filter = fact.createStandardRTFilter_SubtractMean();
11
12 // block for saving data to file
13 IRTSink_ToWavFile sink = fact.createStandardRTSink_ToWavFile();
14 sink.setFilename( filename_out );
15
16 // interconnect
17 source.addFilter( filter );
18 source.addSink( sink );
19
20 // process data
21 source.init().run_().flush_().finalize_();
```

The MATLAB version serves as a reference implementation for the Java version. The verification of the Java implementation can be done by comparing the output files or by direct instantiation of the Java blocks within MATLAB scipts. The Java implementation shown above can be used for JVM or Android's Dalvik VM.

4.3.2 Example: Simple ACF based F0 Algorithm

Complex algorithms can be constructed by interconnecting basic blocks rather than implementing all parts from scratch. If the block networks are enclosed within basic block classes and all ports are connected appropriately, the new processing systems can be used like any other blocks. This step is called *grouping* and is recommended for most algorithm implementations.

Figure 4.3 outlines one of many possible RTBlocks realizations for the ACF based F0 estimation algorithm shown in figure 3.13. The input page size and the step size of the system can be adjusted by modifying the enclosing block. Page size changes are forwarded to the top most block. If the algorithm is carefully designed the changes are propagated to all related children. The output page size is fixed to one sample per calculation iteration. Step size changes are directly handled by *RTSink_OverlappedFrames*. Decreasing the step size leads to more calculation iterations per input page.

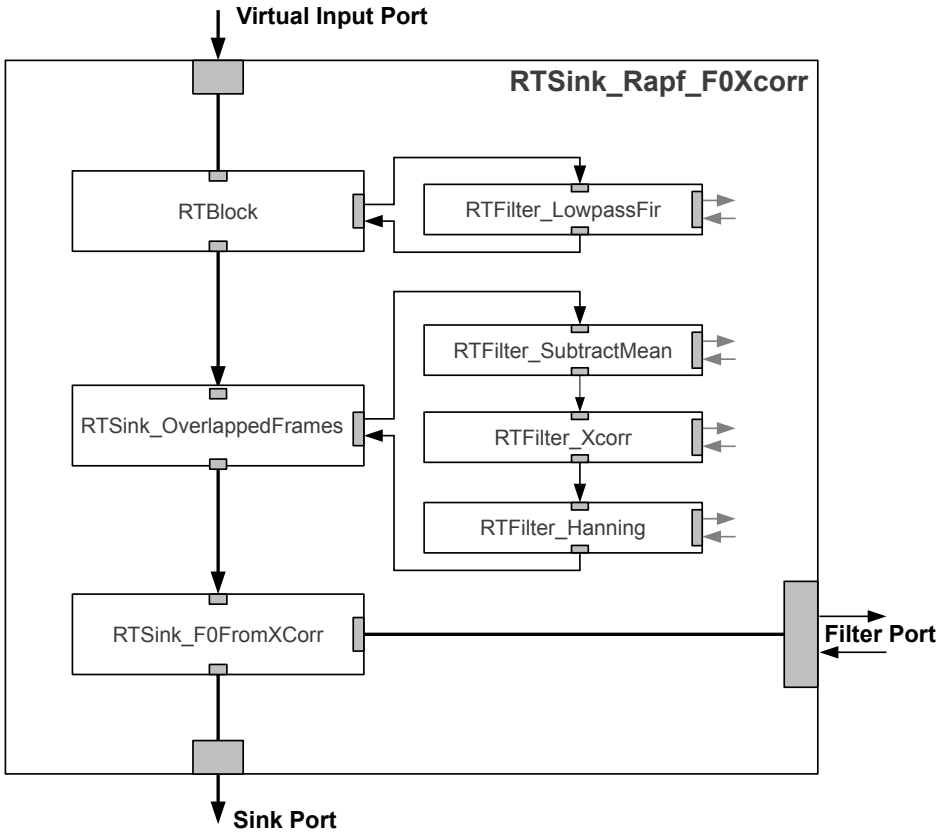


Figure 4.3: The image shows the RTBlocks implementation of the simple ACF based F0 estimation algorithm. It implements the algorithm shown in block diagram L.6 (see section 3.2.6). The implementation is formed by simply interconnecting standard filters and sinks, adding an algorithm specific F0 extraction block (RTSink_F0FromXCorr) and grouping the network together to the new block RTSink_Rapf_F0Xcorr.

4.3.3 Example: Speech Analysis System

A complete speech analysis system exists of at least one audio source, several feature extraction parts and a one or several feature representations. Figure 4.4 shows a very simple speech analysis system: Audio is recorded from a microphone and amplified by a simple automatic gain control block. VAD, F0 estimation and input data storage are connected in parallel to the audio source. Input data and estimated features are plotted as time series. By adding or removing blocks the influence of specific processing steps can easily be evaluated.

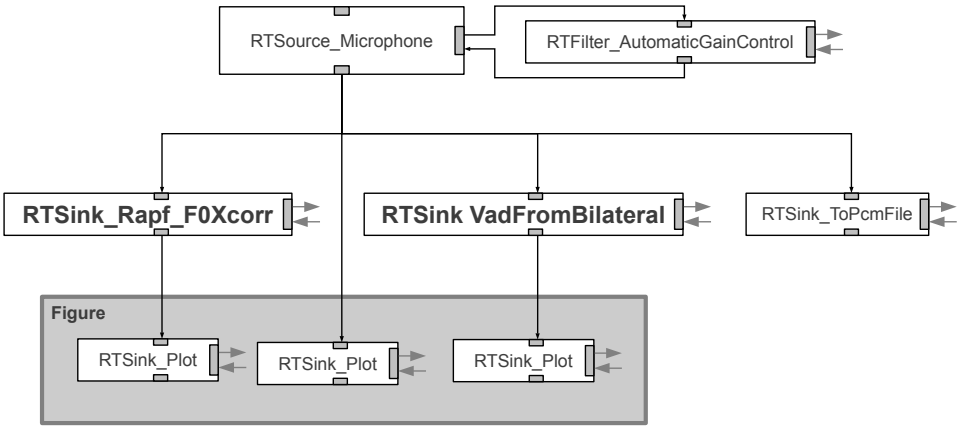


Figure 4.4: The image shows the RTBlocks implementation of a simple speech feature extraction system. Core blocks of the framework connected together, different page sizes can be used. By adding or removing blocks the influence of specific processing steps can be evaluated.

4.4 Conclusion

The framework supports the design of simple and complex audio processing systems. By dividing complex algorithms into smaller blocks or grouping networks together to single blocks implemented parts can easily be verified, tested and reused. Once the system architecture is settled, computational expensive parts can be encapsulated and replaced by faster implementations.

5 RTBlocks Verification

Unit tests for all blocks which are described in chapter 3 guarantee the correct behavior of the framework. For each RTBlocks version an own test bench is provided. Most MATLAB and Java parts can be used within the MATLAB test bench. The Java versions of blocks can also be used within the Java test bench. An Android test bench is provided which deals with all smartphone specific parts. Additionally the GUI based remote control for the Android test bench can be integrated in MATLAB.

In this section test results are discussed for the speech analysis algorithms of section 3.2.6. The discussion includes several plots of test runs which are executed within the different test benches.

5.1 Overview

In *RAPF Preliminary Algorithm Study* (see appendix section III) several speech analysis algorithms are reviewed, implemented and tested. RTBlocks includes four very different implementations to cover a large area of possible implementation methods. Extensive preprocessing and post-processing methods are switched off to simplify the algorithms and to ease the real-time implementation and verification. Global estimations like signal amplitude normalization are avoided. The algorithms computational effort can be lowered by increasing the step sizes of the algorithms according to the target performance capabilities. If the step sizes are changed, the affected algorithms must be re-evaluated to ensure their correct behavior.

5.2 G729B Based VAD Algorithm

The reference implementation of the G729B as described in Benyassine et al. [1997] is used to extract a VAD signal. This algorithm is wrapped by the block *RTSink_Rapf_VadG729B* as discussed in section 3.2.6. The ANSI-C code is accessed from MATLAB by using it's mex extension and by Java by using the JNI. By definition the page size is fixed to 80 samples. A step size equals $^1/_3$ of the page size leads to reasonable results. The test signal is formed two concatenated sentences of two different male speakers, taken from the TIMIT database (Garofolo et al. [2011]): *"If necessary to replace both halves on grill, sear cuts and allot extra time"* and *"Don't ask me to carry an oily rag like that"*. The MATLAB high level implementation as discussed in the appendix section K.3.4 is used as reference signal. Both, the MATLAB and the Java version of the RTBlocks implementation lead to equal VAD signals compared to the reference signal. The resulting VAD plots are shown in figure 5.1. The related MATLAB script for the unit test is provided in the appendix section A. The same test executed as Java standalone application and embedded within the Android test bench is shown in figure 5.2 and figure 5.3. The Android test bench can be directly controlled using the touch screen of the smartphones. Another method which is used for testing is the remote control of the test bench from a connected computer. The control software on the computer is based on a Python script which can unlock the smartphone, install, start and stop applications, simulate graphical user inputs and take screen snapshots of the smartphone (see appendix section E). The control script

can be used from within MATLAB if the scripts are compiled to Java byte code with the Jython compiler or the Jython Interpreter is used from within MATLAB (Jython [2012]).

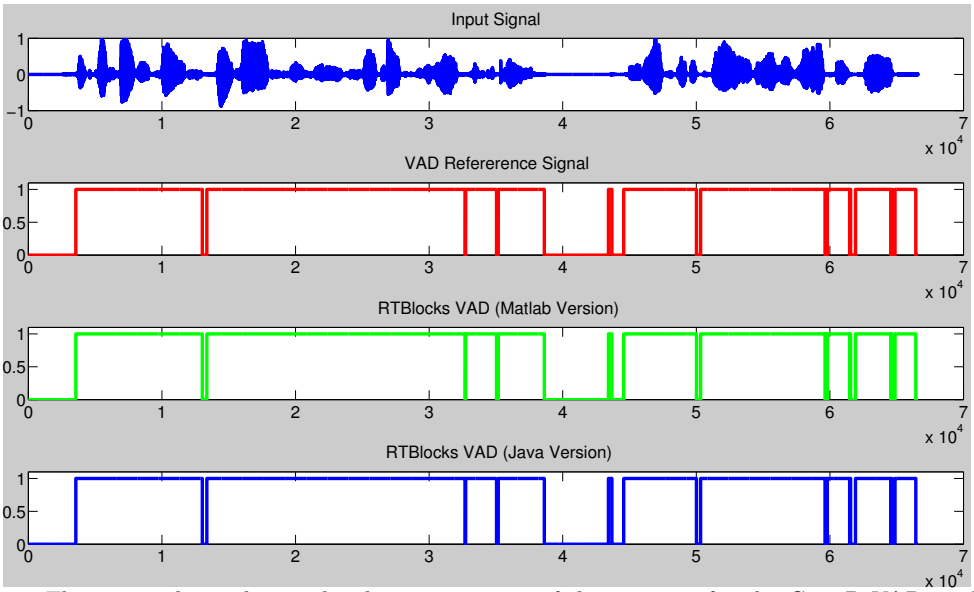


Figure 5.1: The image shows the graphical representation of the unit test for the G729B VAD implementation. The top most plot is the input signal which is formed of the TIMIT sentences "If necessary to replace both halves on grill, sear cuts and allot extra time" and "Don't ask me to carry an oily rag like that" (Garofolo et al. [2011]). The reference signal (second top most plot) is calculated by the reference implementation based on the results of the "RAPF Preliminary Algorithm Study" (see appendix section A). The implementations based on RTBlocks for MATLAB and on RTBlocks for Java are shown in the plots at the lower end of the image. All algorithms are invoked from within the MATLAB test bench.

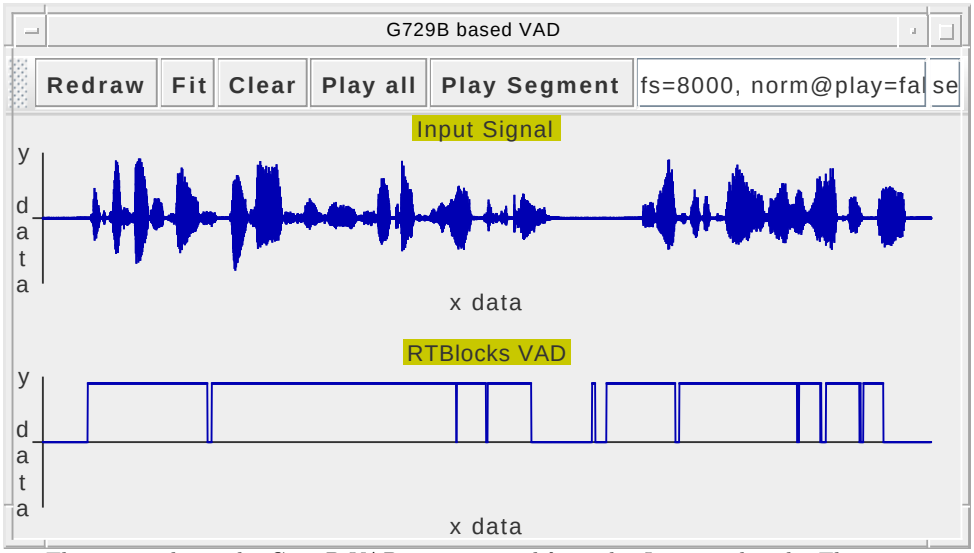


Figure 5.2: The image shows the G729B VAD test executed from the Java test bench. The top most plot is the input signal, the plot on the lower end is the VAD result. RTBlocks provides simple interactive plot analyzing tools like replaying the whole signal or just a signal segment, scaling the y axes, searching and marking plotted points and displaying information about selected points. For the current work the plot library is extended by a PDF export tool based on the open source library iText (iText Software Corp and 1T3XT BVBA [2012]).

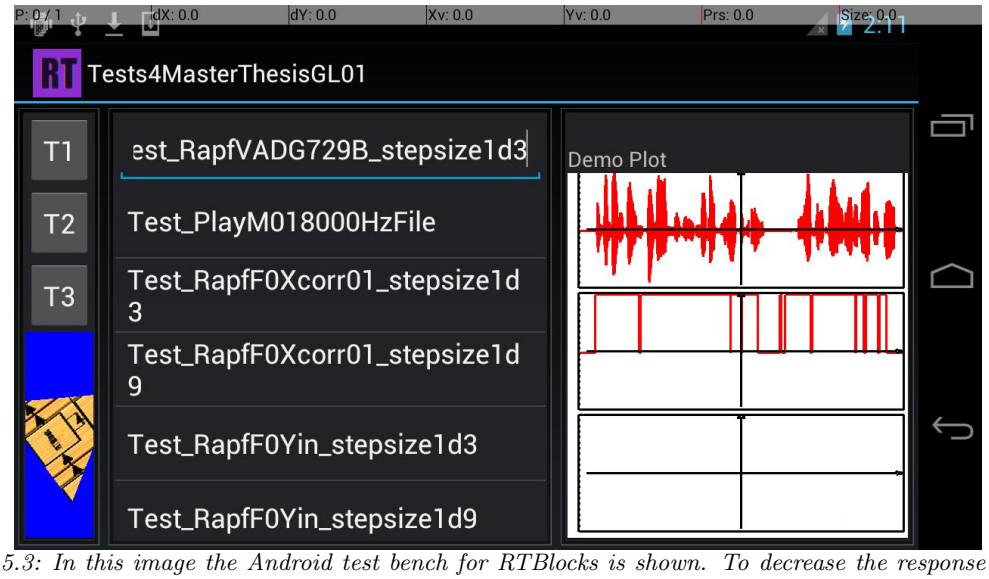


Figure 5.3: In this image the Android test bench for RTBlocks is shown. To decrease the response time of the application the test bench is started with the OpenGL based plot library (right frame). The top most plot shows the input signal, the second plot shows the VAD result. The plot at the lower end of the frame is not used. the left frame contains the toolbox buttons for controlling tests. Buttons can be added, removed or re-assigned as needed. The lower part of the toolbox contains a rotating OpenGL triangle which indicates when the system works as expected. The middle frame contains a scrollable list box where all implemented tests are listed. The top most line of the frame is a text input field. Tests are invoked by clicking on the related list item or by typing the test name. The latter method is used from the remote Python script to invoke specific tests.

5.3 Variance Based VAD Algorithm

A pure Java VAD is the variance based algorithm introduced in the *RAPF Preliminary Algorithm Study* (see appendix section K.3.7) and reviewed in section 3.2.6. The algorithm uses a page size equals 500 samples and a step size equals $^1/_{16}$ of the page size. The test signal is the same as described in section 5.2. The reference signal is formed by the MATLAB high level implementation and compared against the RTBlocks versions for MATLAB and Java. Both versions lead to the same VAD signals compared to the reference implementation. The results are shown in figure 5.4, the related unit test scripts are provided in the appendix section B. he same test executed as Java standalone application and embedded within the Android test bench is shown in figure 5.5 and figure 5.6.

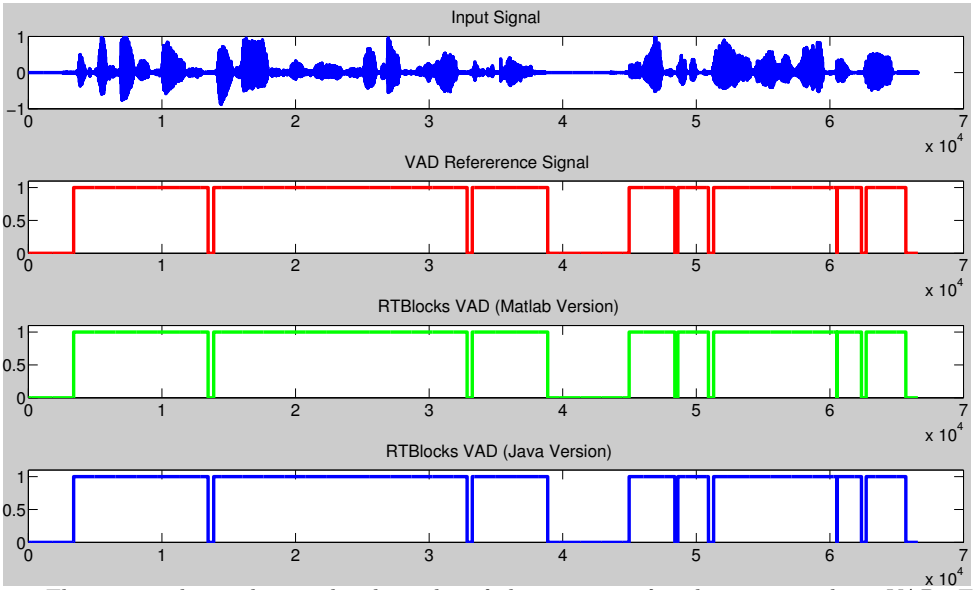


Figure 5.4: The image shows the graphical results of the unit test for the variance base VAD. The input signal is shown at the top most plot, the reference signal is shown in the second plot. The algorithm for the reference signal is proposed in the "RAPF Preliminary Algorithm Study" (see appendix section B). The third and fourth plots are the RTBlocks implementations for MATLAB and Java.

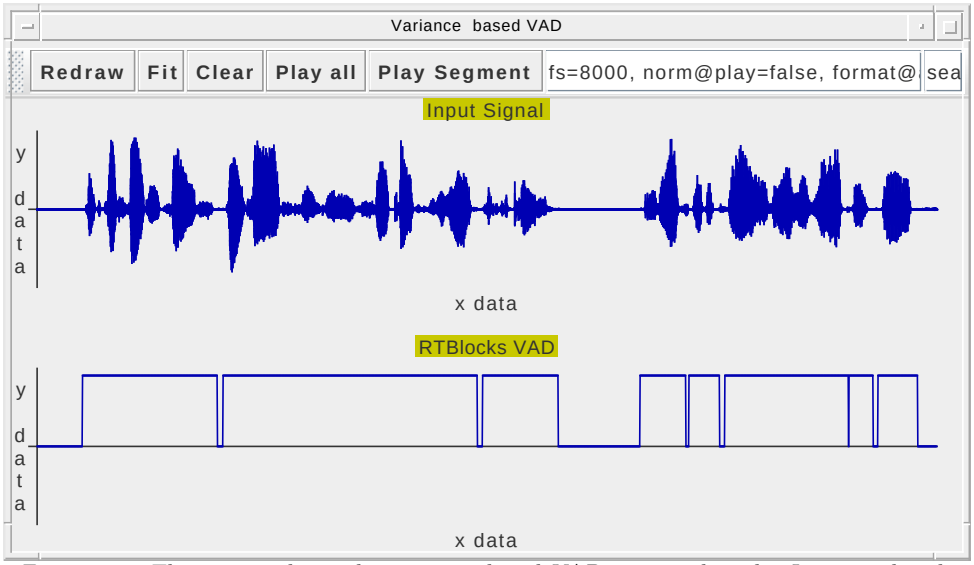


Figure 5.5: This image shows the variance based VAD, executed in the Java test bench.

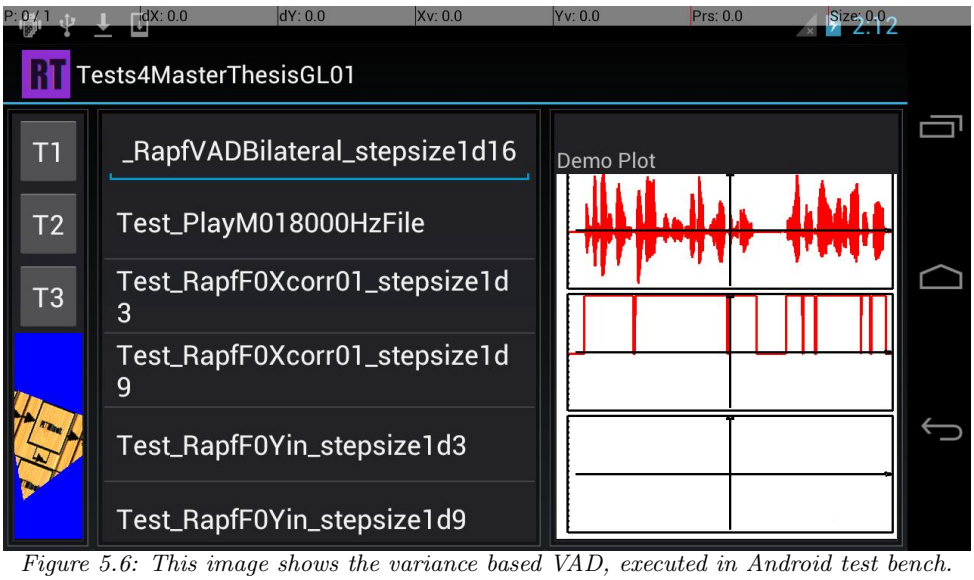


Figure 5.6: This image shows the variance based VAD, executed in Android test bench.

5.4 Xcorr Based F0 Estimation Algorithm

The correlation based F0 estimation algorithm described in section 3.2.6 is wrapped by the RTBlocks class *RTSink_Rapf_F0Xcorr01*. The block used a page size of 1600 samples and a step size equals $^1/_9$ of the page size. The implementations are tested with the down-sampled phonetically balanced sentence *"The source of the huge river is the clear spring"* (IEEE Recommendation [1969]). Neither for the F0 result of the RTBlocks MATLAB version nor of the RTBlocks Java version significant numerical differences were observed. The F0 results of the implementations are shown in figure 5.7, the unit test script is provided in the appendix section C. The same test executed as Java standalone application and embedded within the Android test bench is shown in figure 5.8 and figure 5.9.

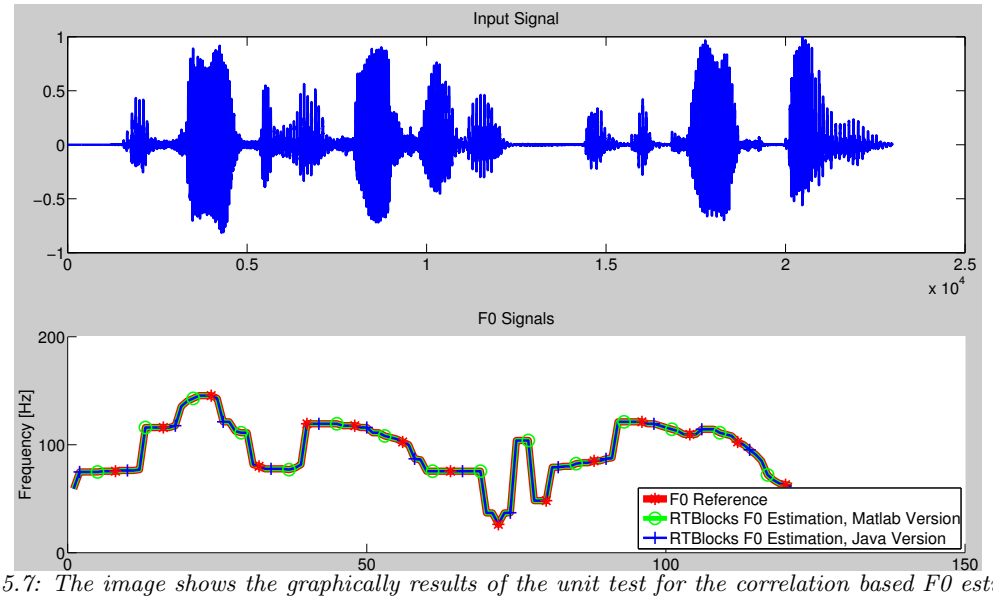


Figure 5.7: The image shows the graphically results of the unit test for the correlation based F0 estimation algorithm. The reference signal is calculated by the algorithm proposed in "RAPF Preliminary Algorithm Study" (see appendix section L.3.1). The first plot shows the test signal, which is the phonetically balanced sentence "The source of the huge river is the clear spring" (IEEE Recommendation [1969]). The second plot includes the reference F0 estimation result and the RTBlocks implementations for MATLAB and Java.

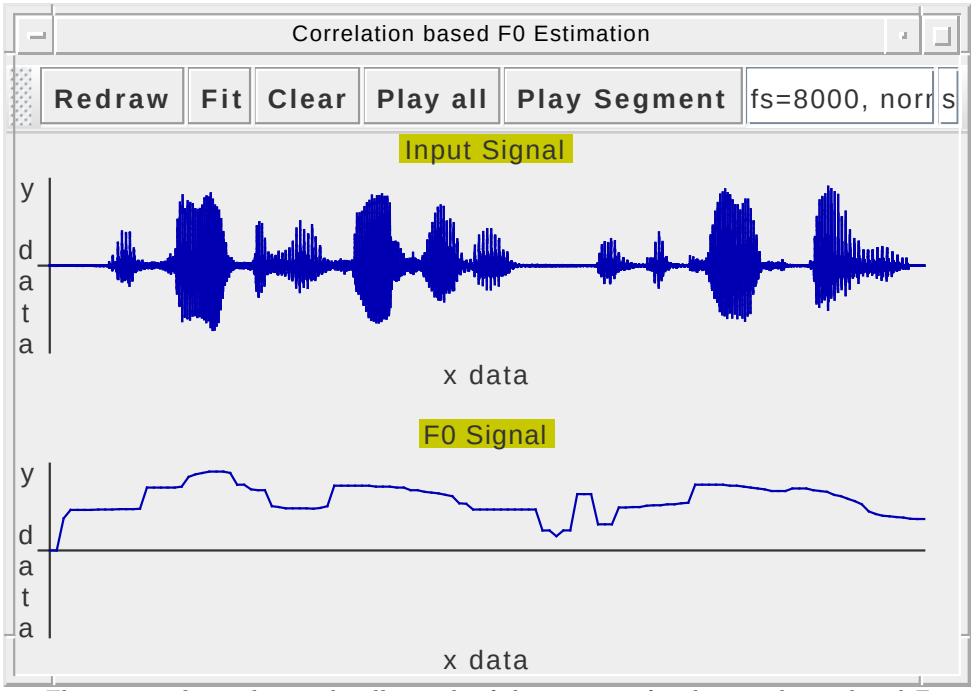


Figure 5.8: This image shows the graphically result of the unit test for the correlation based F0 estimation algorithm, executed in the Java test bench.

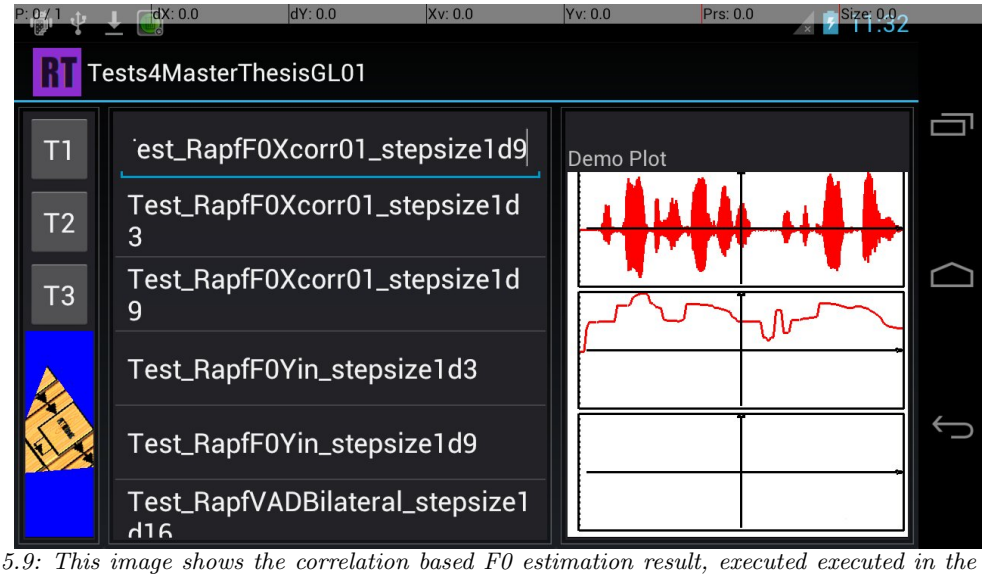


Figure 5.9: This image shows the correlation based F0 estimation result, executed executed in the Android test bench.

5.5 Yin F0 Estimation Algorithm

The *CMU Sphinx* implementation includes the F0 estimation algorithm described in de Cheveigné and Kawahara [2002]. This algorithm is wrapped by the block *RTSink_Rapf_F0Yin* as discussed in section 3.2.6. The block uses a page size of 1600 samples and a step size equals $^1/_9$ of the page size. The reference signal is formed by the MATLAB implementation of the *RAPF Preliminary Algorithm Study* (see appendix section L.3.6). The test signal is the same as described in section 5.4. The reference implementation, the RTBlocks MATLAB implementation and the RTBlocks Java version produce the same F0 results, no significant numerical differences were observed. The resulting F0 plots are shown in figure 5.10. The related MATLAB script for the

the unit test is provided in the appendix section D. The same test executed as Java standalone application and embedded within the Android test bench is shown in figure 5.11 and figure 5.12, respectively.

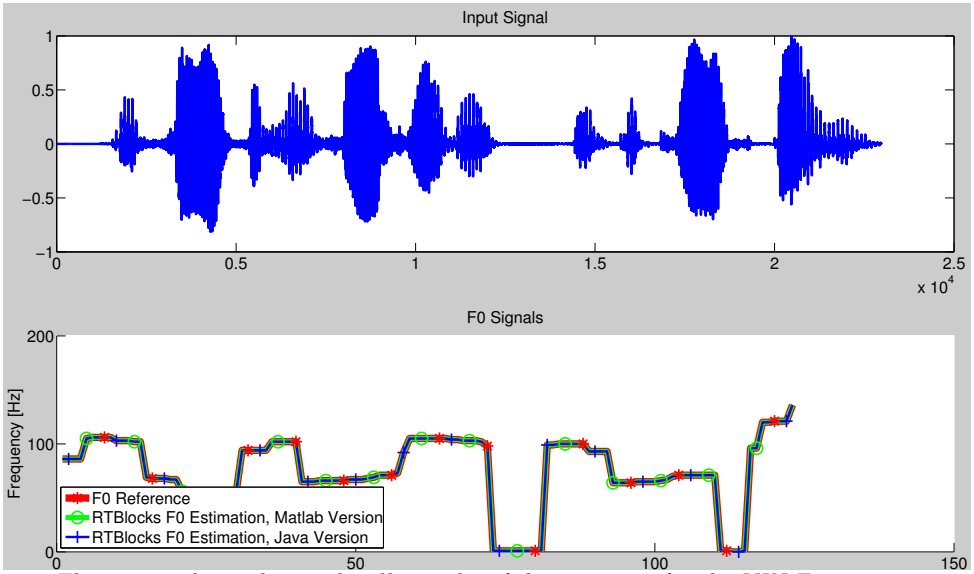


Figure 5.10: This image shows the graphically results of the unit tests for the YIN F0 estimation implementations. The first plot shows the test signal, the second plot shows the F0 estimations.

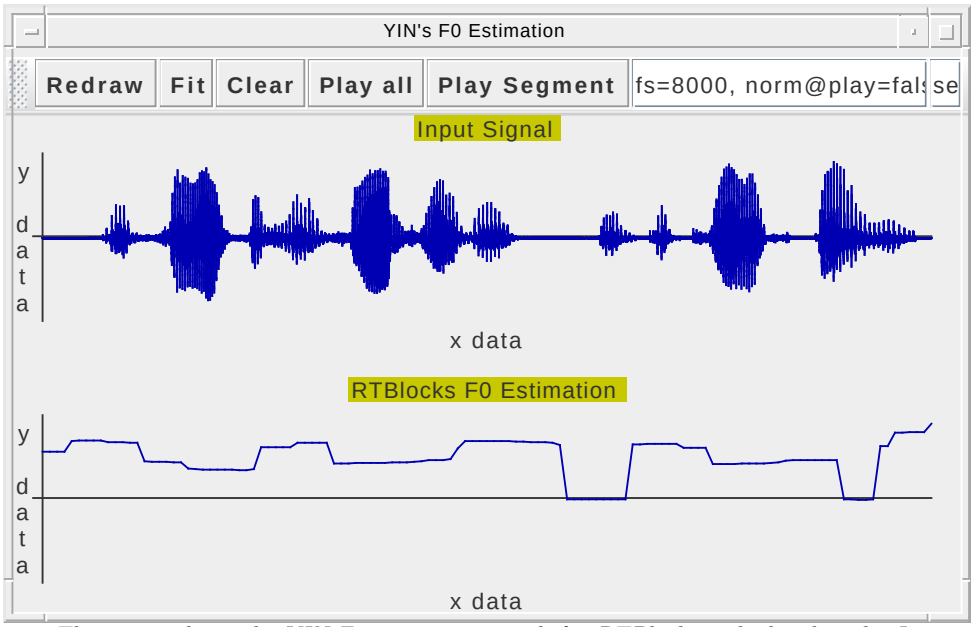


Figure 5.11: The image shows the YIN F0 estimation result for RTBlocks, calculated in the Java test bench.

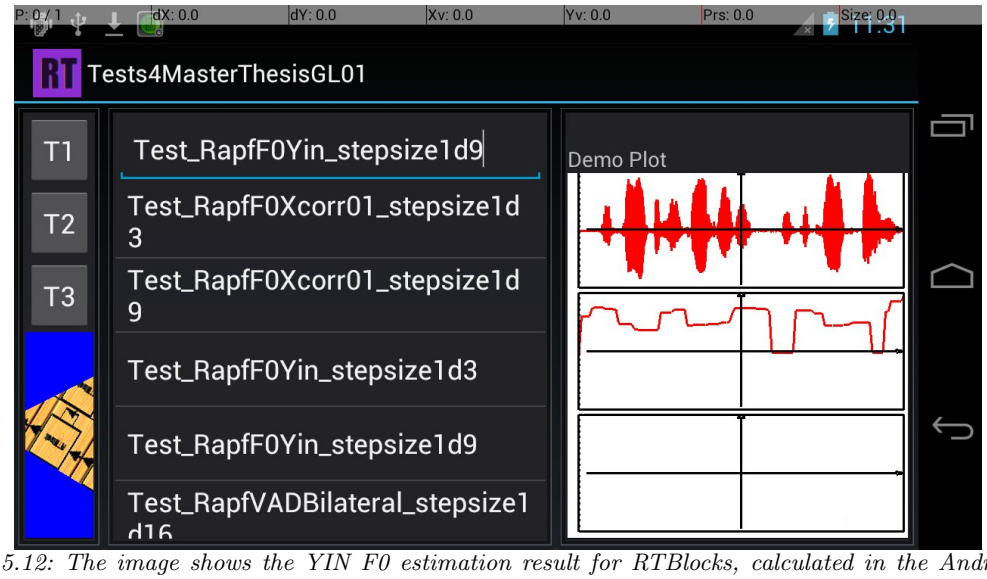


Figure 5.12: The image shows the YIN F0 estimation result for RTBlocks, calculated in the Android test bench.

5.6 Raspberry Pi Test Platform

RTBlocks is designed for cross platform usage. Beside the JVM of a full featured desktop PC and the Dalvik VM of a Smartphone with Android operation system the *Raspberry PI Model B* platform is used to test the framework (Raspberry Pi Foundation [2012]). The single-board computer has 2 USB ports, an Ethernet port, a 700 MHz ARM CPU core, a *Broadcom VideoCore IV* GPU and 256 megabyte RAM, which are shared between the CPU and the GPU. The energy consumption is about 700 mA or 3.5 Watt. It features the OpenJDK version 6 (Oracle Corporation [2012]), the OpenGL ES 2.0 API (Khronos Group [2012]) and contains a FPU.

The RTBlocks Java version is used to test the framework on the Raspberry Pi hardware. Figure 5.13 shows the test arrangement, the command line test bench, the variance based VAD algorithm and the correlation based F0 estimation algorithm (see section 3.2.6 for algorithm details). Without further optimizations real-time processing of those algorithms is not possible on low-performance systems like this.

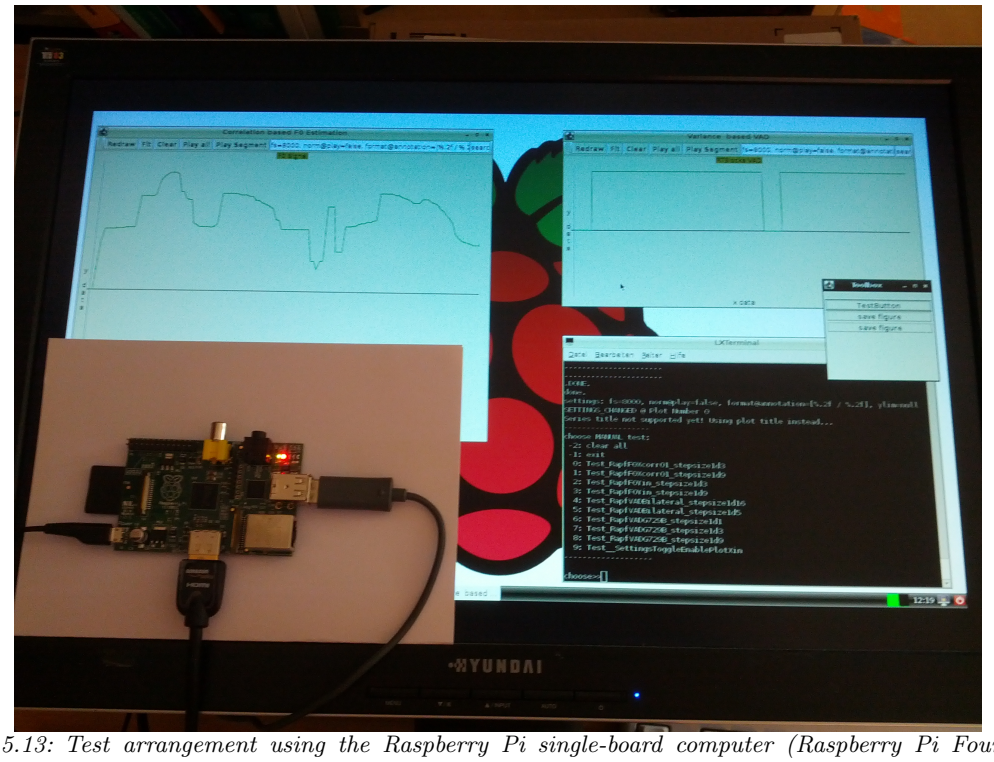


Figure 5.13: Test arrangement using the Raspberry Pi single-board computer (Raspberry Pi Foundation [2012]): The computer in the left lower corner is placed on the monitor surface and connected to the monitor by a high definition multimedia interface (HDMI) connector. The left upper corner shows the correlation based F0 estimation result, the right upper corner shows the variance based VAD result. The command line test bench and the export toolbox for the plots are shown in the right lower corner. The RTBlocks Java version is used which does not have any hardware optimizations, but is highly portable. Due to the lack of hardware optimized algorithm parts and the low performance processor of the Raspberry Pi real-time computation of the algorithms is not possible.

5.7 Conclusion

The different test benches provided by the framework make the evaluation of block implementations very easy. Test results of the Android test bench can be read out by MATLAB or Java by pulling files from the device or by communicating via a TCP port. When the same code base for the JVM and the Android Dalvik VM is used, tests must be applied only once.

The different implementations all leads to the same calculation results. Global estimations must be avoided or approximated for real-time applications.

It was shown that speech analysis is principally possible on modern smartphones with Android. Complex algorithms tend to require high computational effort and often consume very much energy. In real environments these are critical points for continuous operation applications.

6 Discussion and Conclusion

This chapter provides a short summary of the work, reviews the evaluation and gives an outlook for further steps.

6.1 RTBlocks for Matlab, Java and Android

The new framework RTBlocks supports the design and implementation of short term speech analysis algorithms for both, mobile and non-mobile target platforms. Currently about 40 different blocks are implemented and tested, from basic input and output blocks to simple FIR and IIR filters to ready-to-use speech analysis systems. RTBlocks eases the conversion of MATLAB high level algorithm implementations to target specific code. Android was chosen as primary operation system for mobile targets, so RTBlocks is available for MATLAB, Java and *Java for Android*. The latter adds Android specific parts to the base Java implementation. A MATLAB test bench is provided which can be used to verify any implemented block, regardless whether blocks are implemented in MATLAB or Java or both. This reduces possible error sources. Further test benches are provided by RTBlocks for Java and RTBlocks for Android.

Different use cases of the framework are verified, including MATLAB and Java real-time implementations of simple and complex algorithms, implementation and verification of speech analysis algorithms for smartphones, usage of third party libraries and algorithms written in C or C++, hardware accelerated processing and cross-compilation of third party libraries, fast OpenGL based plotting of time series signals on both, non-mobile and mobile devices and many more.

RTBlocks for Java and RTBlocks for Android share the same code base which obviates the need for unit tests for Android block implementations in most cases. Nevertheless, a test bench for Android is provided and tested on the *Samsung Galaxy Nexus (GT-I9250)* smartphone. The test bench can be controlled by interacting with the simple graphical user interface manually on the smartphone or remotely from a connected host computer.

The implementation and verification of complex VAD and F0 algorithms based on RTBlocks shows the benefits of the framework: Implementation time and error-proneness is reduced significantly. For intelligent designs the implemented blocks directly reflect the underlying graphical block designs of algorithms. The design flow for RTBlocks based implementations is intended to stop at any phase if the requirements are fulfilled: Many aspects of algorithms can be evaluated based on MATLAB high level implementations. To test real-time capabilities of algorithms the usage RTBlocks for MATLAB is sufficient. Standalone systems for full featured platforms can be directly extracted from MATLAB. For systems with limited processor performance the Java version of RTBlocks provides a counter part to each MATLAB block. New blocks are derived from a single base block which implements important core functionality for creating and executing short term analysis systems. Java utilities for vector math, signal processing and plotting are implemented to reduce the gap between MATLAB source code and Java source code. The Android extensions for RTBlocks provide a good working base for implementing speech analysis algorithms on smartphones.

6.2 Further Work

Currently, the framework only supports targets which execute Java byte code. The most important mobile devices, which can execute Java byte code, are smartphones with the Android operation system. To cover a larger field of platforms a C++ version of RTBlocks will be implemented.

The framework is designed as a part of the RAPF project which includes estimating and reporting important speech features in real-time in everyday life (see appendix section II). At the moment only a few basic speech features are calculated (VAD, F0 estimation). One of the next steps of the project will be the integration of more basic features like speaking rate, relative loudness and others. Besides basic features the final application should include features of a higher level of abstraction, like articulation and speaker intelligibility.

Currently, only methods for reporting results are implemented, like visual feedback (plotting) and audio feedback (replaying results). Especially for real-time reporting during conversations, more advanced feedback methods are required which do not distract the conversation. A possible method is haptic feedback, like controlling the vibration of a smartphone based on selected feature limits.

Bibliography

Ableson et al. [2012] W. Frank Ableson, Robi Sen, Chris King, and C. Enrique Oritz. *Android in Action*. Manning, Greenwich, Conn, 2012. ISBN 9781617290503. [Pages 8, 9].

Anderson [1977] Virgil Antris Anderson. *Training the Speaking Voice*. Oxford University Press, USA, 3 edition, 3 1977. ISBN 9780195021509. [Page xxvii].

Android ADK [2012] Android accessory development kit (adk). (android toolset revision 19), access date aug 2012. url: `http://developer.android.com/tools/adk`. [Page 4].

Android NDK [2012] Android native development kit (ndk). (android toolset revision 19), access date aug 2012. url: `http://developer.android.com/tools/sdk/ndk`. [Page 4].

Android SDK [2012] Android software development kit (sdk). (android toolset revision 19), access date aug 2012. url: `http://developer.android.com/sdk`. [Page 4].

AndroidPlot [2011] AndroidPlot (android software library version 0.5.0), access date dec 2011. url: `http://androidplot.com/`. [Pages 18, 30].

Audiotranskription Transcriber [2011] Audiotranskription.de/transcriber (computer program), access date aug 2011. url: `http://www.audiotranskription.de/transkription/weitere-transkriptionssoftware/transcriber/transcriber.html`. [Pages 3, xxix].

Audiotranskription F4 [2011] Audiotranskription.de/f4 (computer program), access date aug 2011. url: `http://www.audiotranskription.de/f4.htm`. [Page xxix].

Austrian Parliament [2011] Austrian parliament - online resource, access date aug 2011. url: `http://www.parlament.gv.at/PAKT/PLENAR/VIDEO/index.shtml`. [Page xxxi].

Babu et al. [2009] C. Ganesh Babu, Dr. P. T. Vanathi, R. Ramachandran, M. Senthil Rajaa, and R. Vengatesh. A comprehensive analysis of voice activity detection algorithms for robust speech recognition system under different noisy environment. *Int. J. Comp. Network. Secur.*, 1, 2009. [Page xliii].

Bagwell and Norskog [2012] Chris Bagwell and Lance Norskog. Sound eXchange (sox) (computer program version 14.4.0), 2012. url: `http://sox.sourceforge.net/`. [Page 5].

Bellman [1957] Richard Bellman. Dynamic programming. *Princeton University Press*, 1957. [Page lxxx].

Bellman [2003] Richard Bellman. *Dynamic programming*. Dover Publications, Mineola, N.Y, 2003. ISBN 0486428095. [Page lxxx].

Ben-Ari [1990] M. Ben-Ari. *Principles of concurrent and distributed programming*. Prentice Hall, New York, 1990. ISBN 013711821X. [Page 3].

Benesty and Huang [2008] Y. Benesty and J. Chen Huang. Wiener and adaptive filters. In Jacob Benesty, M. Mohan Sondhi, and Yiteng Huang, editors, *Springer Handbook of Speech Processing*. Springer-Verlag Berlin Heidelberg, 2008. [Page l].

Benesty [2004] Jacob Benesty. *Audio Signal Processing for Next-Generation Multimedia Communication Systems*. Springer, Berlin, 2004. ISBN 1402077688. [Page lxxiv].

Benyassine et al. [1997] A. Benyassine, E. Shlomot, H.-Y. Su, D. Massaloux, C. Lamblin, and J.-P. Petit. ITU-T recommendation G.729 annex B: A silence compression scheme for use with G.729 optimized for V.70 digital simultaneous voice and data applications. *IEEE Commun. Mag.*, 35(9):64–73, 1997. doi: 10.1109/35.620527. [Pages 28, 41, xxxiv, xlviii, xlix].

Boersma and Weenink [2011] Praat: doing phonetics by computer (computer program version 5.2.25), access date jun 2011. url: `http://www.praat.org/`. [Pages xvi, 3, lxxv].

Boersma [1993] Paul Boersma. Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound. In *Proc. Inst. Phonetic Sci. Amsterdam*, volume 17, pages 97–110, 1993. [Pages xii, xxxvi, xliv, lxxiv, lxxv].

Boll [1979] S. Boll. A spectral subtraction algorithm for suppression of acoustic noise in speech. In *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, volume 4, pages 200–203, 1979. doi: 10.1109/ICASSP.1979.1170696. [Page xxxiv].

Brookers [2011] Voicebox: Speech processing toolbox for matlab (matlab script package), access date may 2011. url: `http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html`. [Pages 4, xliii, xlix, l, lxxix, lxxx].

Brown and Puckette [1993] J. C. Brown and M. Puckette. A high-resolution fundamental frequency determination based on phase changes of the fourier transform. *J. Acoust. Soc. Am.*, 94/2:662–667, 1993. [Page lxx].

Cabrera et al. [2007] D. Cabrera, S. Ferguson, and E. Schubert. Psysound3: Software for acoustical and psychoacoustical analysis of sound recordings. pages 356–363, Montreal, Canada, 2007. Schulich School of Music, McGill University. [Page 4].

Cannam et al. [2010] DSSI: Disposable soft synth interface, access date aug 2010. url: `http://dssi.sourceforge.net/`. [Page 3].

Che and Miao [2010] Dandan Che and Zhenjiang Miao. Real-time cartoon style video generation. In *Int. Conf. Image Anal. Sig. Process.*, pages 640–643, 2010. doi: 10.1109/IASP.2010.5476191. [Page xli].

de Cheveigné and Kawahara [2002] Alain de Cheveigné and Hideki Kawahara. YIN, a fundamental frequency estimator for speech and music. *J. Acoust. Soc. Am.*, 111(4):1917–1930, 2002. [Pages xii, 29, 47, liv, lxvi, lxviii, lxxx, lxxxi, lxxxiii, lxxxix, xc].

Dixon et al. [2009] Paul R. Dixon, Tasuku Oonishi, and Sadaoki Furui. Fast acoustic computations using graphics processors. In *Proc. IEEE Int. Conf. Acoust. Speech Signal Process*, volume 1- 8, 2009. ISBN 978-1-4244-2353-8. [Page xxviii].

Drugman [2012] GLOAT: Glottal analysis toolbox (software), access date jun 2012. url: `http://tcts.fpms.ac.be/~drugman/Toolbox/`. [Page 4].

Eaton et al. [2012] GNU Octave (computer program version 3.6.2), access date jun 2012. url: `http://www.gnu.org/software/octave/`. [Page 4].

Ephraim and Malah [1984] Yariv Ephraim and David Malah. Speech enhancement using a minimum mean-square error short-time spectral ampliude estimator. *IEEE T. Acoust. Speech.*, 32(6):1109–1121, 1984. [Pages xliii, l].

Espi et al. [2010] M. Espi, S. Miyabe, T. Nishimoto, N. Ono, and S. Sagayama. Analysis on speech characteristics for robust voice activity detection. In *IEEE Workshop Spoken Language Technol.*, pages 151 –156, dec. 2010. doi: 10.1109/SLT.2010.5700838. [Page xlii].

ETSI 301 708 [1999] ETSI EN 301 708 V7.1.1: Voice activity detector (VAD) for adaptive multi-rate (AMR) speech traffic channels, access date 12 1999. url: `http://www.etsi.org`. [Page xxxiv].

FFmpeg [2012] FFmpeg (software library version 0.11.1), access date jan 2012. url: `http://ffmpeg.org`. [Page 4].

Frigo and Johnson [2005] M. Frigo and S. G. Johnson. The design and implementation of fftw3. *IEEE Proc.*, 93(2):216–231, February 2005. doi: 10.1109/JPROC.2004.840301. [Pages 4, 25, 31].

Fujimoto et al. [2008] M. Fujimoto, K. Ishizuka, and T. Nakatani. A voice activity detection based on the adaptive integration of multiple speech features and a signal decision scheme. In *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, pages 4441 –4444, march 2008. doi: 10.1109/ICASSP.2008.4518641. [Page lxiv].

Furse et al. [2000] Linux audio developer's simple plugin API (LADSPA) (software API), access date apr 2000. url: `http://www.ladspa.org/`. [Page 3].

Furui [2001] Sadaoki Furui. *Digital speech processing, synthesis, and recognition.* Marcel Dekker, New York, 2001. ISBN 0824704525. [Page 1].

Gamma et al. [1995] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: Elements of reusable object-oriented software.* Addison-Wesley, Reading, Mass, 1995. ISBN 0201633612. [Page 17].

Garofolo et al. [2011] DARPA TIMIT: Acoustic phonetic continuous speech corpus, access date jul 2011. [Pages 41, 42, xxix, xxxii].

Gerhard [2003] David Gerhard. Pitch extraction and fundamental frequency: History and current techniques. Technical report, Department of Computer Science - University of Regina, 2003. [Page lxv].

German Parliament [2011] German parliament - online resource, access date aug 2011. url: `http://www.bundestag.de/bundestag/parlamentsfernsehen`. [Page xxxi].

Google [2012] "philosophy and goals". android open source project., access date jul 2012. [Pages 1, 3].

Grimm and Kroschel [2007] Michael Grimm and Kristian Kroschel. *Robust Speech Recognition and Understanding.* I-Tech Education and Publishing, 2007. [Pages xxxiii, xxxv].

Guarnieri et al. [2006] G. Guarnieri, S. Marsi, and G. Ramponi. Fast bilateral filter for edge-preserving smoothing. *Electron. Let.*, 42(7):396 – 397, march 2006. doi: 10.1049/el:20064369. [Page xli].

Guihot [2012] Guihot. *Pro Android apps performance optimization.* Apress distributed to the book trade by Springer, Berkeley, CA New York, 2012. ISBN 9781430240006. [Page 3].

Gunturk [2011] B. K. Gunturk. Fast bilateral filter with arbitrary range and domain kernels. *IEEE T. Image. Process.*, 20(9):2690–2696, 2011. doi: 10.1109/TIP.2011.2126585. [Page xli].

Hall [2000] T. Allan Hall. *Phonologie: Eine Einfuehrung*. Walter De Gruyter Inc, City, 2000. ISBN 3110156415. [Page xxix].

Harris et al. [2009] LV2: Linux audio developer's simple plugin API version 2, access date June 2009. url: `http://lv2plug.in/`. [Page 3].

Hartung [2006] Arnulf Deppermann; Martin Hartung. Gespraechsforschung Online-Zeitschrift zur verbalen Interaktion. volume 7. Der Verlag fuer Gespraechsforschung, 2006. [Page xxix].

Heckmann et al. [2011] Martin Heckmann, Xavier Domont, Frank Joublin, and Christian Goerick. A hierarchical framework for spectro-temporal feature extraction. *Speech. Commun.*, 53(5):736 – 752, 2011. doi: 10.1016/j.specom.2010.08.006. [Page xxviii].

Hess [1976] W. Hess. An algorithm for digital time-domain pitch period determination of speech signals and its application to detect f0 dynamics in VCV utterances. In *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, volume 1, pages 322–325, 1976. doi: 10.1109/ICASSP.1976.1170073. [Page lxxiii].

Hess [2007] Wolfgang Hess. Pitch and voicing determination of speech with an extension toward music signals. In *Springer Handbook of Speech Processing*. Springer, 2007. [Pages lxv, lxvi, lxix, lxx, lxxi, lxxii].

Hong and Yusoff [2008] Tang Wei Hong and M.A. Yusoff. Speech processing in FPGA with C-to-RTL compiler technology. In *Int. Conf. Electron. Des.*, pages 1 –6, dec. 2008. doi: 10.1109/ICED.2008.4786677. [Page xxviii].

Howard et al. [2005] D.M. Howard, A. Jackson, and A.W. Howard. On the development of singing real-time visual displays for voice therapy. In *IEEE Int. Seminar Med. Applicat. Signal Proc.*, pages 141 – 146, nov. 2005. [Page xxviii].

Hoy et al. [2003] Marc Hoy, Robert Eckstein, and Dave Wood. *Java Swing*. O'Reilly, Sebastopol, CA, 2003. ISBN 0596004087. [Page 30].

Huang et al. [2001] Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing*. Prentice Hall PTR, Upper Saddle River, 2001. [Pages xii, lxxiii, lxxiv, lxxv, lxxvii, lxxviii].

Hunter [2012] Matplotlib: A 2D graphics environment, access date aug 2012. url: `http://matplotlib.sourceforge.net/`. [Page 4].

IEEE 754-2008 [2012] IEEE standard for floating-point arithmetic, access date feb 2012. url: `http://dx.doi.org/10.1109/IEEESTD.2008.4610935`. [Page 9].

IEEE Recommendation [1969] IEEE Recommendation. IEEE recommended practice for speech quality measurements. *IEEE T. Acoust. Speech.*, 17:225–246, 1969. [Page 46].

Igarashi et al. [2010] M. Igarashi, M. Ikebe, S. Shimoyama, K. Yamano, and J. Motohisa. O(1) bilateral filtering with low memory usage. In *IEEE Proc. Conf. Int. Image Process.*, pages 3301 –3304, sept. 2010. doi: 10.1109/ICIP.2010.5652046. [Page xli].

International Phonetic Association [2011] International phonetic association - online resource, access date aug 2011. url: `http://www.langsci.ucl.ac.uk/ipa`. [Page xxix].

Ishizuka et al. [2010] Kentaro Ishizuka, Tomohiro Nakatani, Masakiyo Fujimoto, and Noboru Miyazaki. Noise robust voice activity detection based on periodic to aperiodic component ratio. *Speech. Commun.*, 52(1):41 – 60, 2010. doi: 10.1016/j.specom.2009.08.003. [Pages xii, xliv, xlv, xlvi].

Itakura [1975] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE T. Acoust. Speech.*, 23(1):67 – 72, feb 1975. doi: 10.1109/TASSP.1975.1162641. [Page lxxx].

iText Software Corp and 1T3XT BVBA [2012] itext pdf library release notes (computer library version 155), access date jan 2012. url: `http://www.itextpdf.com/`. [Page 42].

Jawale [2009] Pranav Shriram Jawale. A study of voice activity detection techniques for cellphone speech. Technical report, Department of Electrical Engineering, IIT Bombay, 2009. [Pages xxxvi, xxxvii].

Johnson and Shami [1993] D.H. Johnson and P.N. Shami. The signal processing information base (SPIB). *IEEE Signal Process. Mag.*, 10(4):36, 38 –42, oct 1993. doi: 10.1109/79.248556. [Pages xxxii, lvii].

Jones et al. [2012] SciPy: Open source scientific tools for Python, access date jul 2012. url: `http://www.scipy.org/`. [Page 4].

Juang and Chen [1998] B.H. Juang and Tsuhan Chen. The past, present, and future of speech processing. *IEEE Signal Process. Mag.*, 15(3):24 –48, may 1998. doi: 10.1109/79.671130. [Page xxvii].

Jung et al. [2010] Junhee Jung, Seunghun Jin, Dongkyun Kim, Hyung Soon Kim, Jong Suk Choi, and Jae Wook Jeon. A voice activity detection system based on FPGA. In *Int. Conf. Control Autom. Syst. Eng.*, pages 2304 –2308, oct. 2010. [Page xxviii].

Jython [2012] Jython (software version 2.5.3), access date jul 2012. url: `http://www.jython.org/`. [Page 42].

K. et al. [2011] Chaudhury K., Sage D., and Unser M. Fast O(1) bilateral filtering using trigonometric range kernels. *IEEE T. Image. Process.*, (99), 2011. doi: 10.1109/TIP.2011.2159234. [Page xli].

Kay Elemetrics Corp. [1994] Kay Elemetrics Corp. Multi dimensional voice program (MDVP), operations manual, 1994. url: `www.kayelemetrics.com`. [Page xxviii].

Kempelen [1791] Farkas Kempelen. Wolfgangs von Kempelen k.k. wirklichen Hofraths Mechanismus der menschlichen Sprache nebst der Beschreibung seiner sprechenden Maschine. 1791. [Page xxvii].

Khronos Group [2012] Opengl for embedded systems (openGL ES) (graphical application programming interface version 2), access date jan 2012. url: `http://www.khronos.org/opengles`. [Pages 30, 49].

Lahat et al. [1987] M. Lahat, R. Niederjohn, and D. Krubsack. A spectral autocorrelation method for measurement of the fundamental frequency of noise-corrupted speech. *IEEE T. Acoust. Speech.*, 35(6):741 – 750, June 1987. doi: 10.1109/TASSP.1987.1165224. [Page lxx].

Lartillot et al. [2008] Olivier Lartillot, Petri Toiviainen, and Tuomas Eerola. A matlab toolbox for music information retrieval. pages 261–268. 2008. doi: 10.1007/978-3-540-78246-9_31. [Page 4].

Lazzarini [2000] Victor E. P. Lazzarini. The sndobj sound object library. *Org. Sound*, 5 (1):35–49, April 2000. doi: 10.1017/S1355771800001060. [Page 4].

Lee [1983] Jong-sen Lee. Digital image smoothing and the sigma filter. *Graphical Models /graphical Models and Image Processing /computer Vision, Graphics, and Image Processing*, 24:255–269, 1983. doi: 10.1016/0734-189X(83)90047-6. [Page xli].

Lei et al. [2009] Jianjun Lei, Jiachen Yang, Jian Wang, and Zhen Yang. A robust voice activity detection algorithm in nonstationary noise. In *Int. Conf. Indust. Inform. Syst.*, pages 195 –198, 24-25 2009. doi: 10.1109/IIS.2009.110. [Pages xii, l, li, lii].

Li et al. [2005] K. Li, M.N.S. Swamy, and M.O. Ahmad. An improved voice activity detection using higher order statistics. *IEEE T. Speech. Audi. P.*, 13(5):965 – 974, sept. 2005. doi: 10.1109/TSA.2005.851955. [Page xlvii].

Libav [2012] Libav (software library version 0.8.3), access date jan 2012. url: `http://libav.org/`. [Page 4].

Lim and Oppenheim [1979] J.S. Lim and A.V. Oppenheim. Enhancement and bandwidth compression of noisy speech. *IEEE T. Acoust. Speech.*, 67(12):1586 – 1604, dec. 1979. doi: 10.1109/PROC.1979.11540. [Page l].

Loizou [2003] Philip Loizou. Colea: A matlab software tool for speech analysis. 2003. [Page 4].

Luo et al. [2003] Huiyu Luo, Rick Lan, Joshua Cantrell, and Nima Kazemi. EE214A project: Pitch, formant tracking systems. Technical report, Department of Electrical Engineering, UCLA, March 2003. [Page lxxx].

Manfredi et al. [2008] Claudia Manfredi, Tommaso Bruschi, Alessandro Dallai, Alessandro Ferri, Piero Tortoli, and Marcello Calisti. Voice quality monitoring: A portable device prototype. In *Conf. Proc. IEEE Eng. Med. Biol. Soc.*, pages 997 –1000, aug 2008. doi: 10.1109/IEMBS.2008.4649323. [Page xxviii].

Manfredi et al. [2009] Claudia Manfredi, Leonardo Bocchi, and Giovanna Cantarella. A multipurpose user-friendly tool for voice analysis: Application to pathological adult voices. *Biomed. Signal Process Control*, 4(3):212 – 220, 2009. doi: 10.1016/j.bspc.2008.11.006. [Page xxviii].

Markel [1972] J. Markel. The sift algorithm for fundamental frequency estimation. *IEEE T. Acoust. Speech.*, 20(5):367 – 377, December 1972. doi: 10.1109/TAU.1972.1162410. [Page lxix].

Marshall and Sicuranza [2006] Stephen Marshall and Giovanni L. Sicuranza, editors. *Advances in nonlinear signal and image processing.* Hindawi Pub, New York, NY, 2006. ISBN 9775945372. [Page 1].

Martin [1982] Philippe Martin. Comparison of pitch detection by cepstrum and spectral comb analysis. In *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, volume 7, pages 180 – 183, May 1982. doi: 10.1109/ICASSP.1982.1171720. [Pages lxx, lxxi].

Martin [2001] R. Martin. Noise power spectral density estimation based on optimal smoothing and minimum statistics. *IEEE T. Speech. Audi. P.*, 9(5):504–512, 2001. doi: 10.1109/89.928915. [Page lviii].

MathWorks [2011a] Simulink (computer program, matlab extension package version 7.5 (r2010a)), access date jan2011. url: `http://www.mathworks.de/products/matlab/`. [Page 4].

MathWorks [2011b] Matlab (computer program, version 7.10.0 (r2010a)), access date jan 2011. url: `http://www.mathworks.de/products/matlab/`. [Pages 19, 22, 30].

Matsumoto and Hashimoto [2009] M. Matsumoto and S. Hashimoto. Bilateral sound denoising. In *Int. Conf. Inform. Manage. Eng.*, pages 119 –123, april 2009. doi: 10.1109/ICIME.2009.20. [Pages 22, xli, liii].

Mattoccia et al. [2011] S. Mattoccia, M. Viti, and F. Ries. Near real-time fast bilateral stereo on the gpu. In *IEEE Workshop Comput. Vis. Pattern. Recognit.*, pages 136 –143, june 2011. doi: 10.1109/CVPRW.2011.5981835. [Page xli].

Max-Planck-Institute [2011] Elan linguistic annotator (computer software). online, access date may 2011. url: `http://www.latmpi.eu/tools/elan/`. [Page xxix].

McLoughlin [2009] Ian McLoughlin. *Applied speech and audio processing: With Matlab examples*. Cambridge University Press, Cambridge New York, 2009. ISBN 0521519543. [Pages 3, 7].

Mendel [1999] J.M. Mendel. Estimation theory and algorithms: From Gauss to Wiener to Kalman. In Vijay K. Madisetti and Douglas B. Williams, editors, *Digital Signal Processing Handbook*. Boca Raton: CRC Press LLC, 1999. [Page l].

Min [2009] Yang Min. Design of portable hearing aid based on FPGA. In *IEEE Conf. Ind. Electron. Applicat.*, pages 1895 –1898, may 2009. doi: 10.1109/ICIEA.2009.5138532. [Page xxviii].

Mokhov [2008] Serguei A. Mokhov. Introducing MARF: A modular audio recognition framework and its applications for scientific and software engineering research. pages 473–478. 2008. doi: 10.1007/978-1-4020-8741-7_84. [Page 4].

Mongia and Madisetti [2010] Bhupinder S. Mongia and Vijay K. Madisetti. Reliable real-time applications on Android OS. *Georgia Tech Atlanta, GA 30332*, 2010. [Page 3].

Moreno and Fonollosa [1992] A. Moreno and J. A. R. Fonollosa. Pitch determination of noisy speech using higher order statistics. In *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, volume 1, pages 133–136, 1992. doi: 10.1109/ICASSP.1992.225954. [Page lxxii].

MPlayer [2011] Mplayer (computer software version 1.0), access date jan 2011. url: `http://www.mplayerhq.hu`. [Pages xxxi, lv, lvii, lxxxiii].

Muelleger [2008] Markus Muelleger. Evaluation of compilers for matlab- to c-code translation. *Master's Thesis in Computer Systems Engineering, School of Information Science, Computer and Electrical Engineering (Halmstad University)*, 2008. [Pages 7, 30].

Nakatani and Irino [2004] T. Nakatani and T. Irino. Robust and accurate fundamental frequency estimation based on dominant harmonic components. *J. Acoust. Soc. Am.*, 116: 3690–3700, 2004. [Page xlv].

Nakatani et al. [2008] Tomohiro Nakatani, Shigeaki Amano, Toshio Irino, Kentaro Ishizuka, and Tadahisa Kondo. A method for fundamental frequency estimation and voicing decision: Application to infant utterances recorded in real acoustical environments. *Speech. Commun.*, 50(3):203–214, MAR 2008. doi: 10.1016/j.specom.2007.09.003. [Pages xlv, lxxxi].

Noll [1967] A. Michael Noll. Cepstrum pitch determination. *J. Acoust. Soc. Am.*, 41(2): 293–309, 1967. doi: 10.1121/1.1910339. [Pages lxix, lxxv, lxxx].

Nordholm et al. [2005] Togneri Sven Nordholm, Aik Ming, and Toh Roberto. Spectral entropy as speech features for speech recognition. In *Proc. Postgraduate Electron. Eng. Comput. Symp. Australia*, pages pp. 22–25, 2005. [Page li].

Nyrenius and Ramstroem [2011] Mats Nyrenius and David Ramstroem. Generating embedded c code for digital signal processing. *Master of Science Thesis in Computer Science - Algorithms, Languages and Logic. Chalmers University of Technology (Goetenborg, Schweden)*, 2011. [Pages 7, 30].

Oliphant [2007] Travis E. Oliphant. NumPy: Python for scientific computing (python package), 2007. url: `http://www.numpy.scipy.org/`. [Page 4].

Oppenheim and Schafer [2004] A.V. Oppenheim and R.W. Schafer. From frequency to quefrency: a history of the cepstrum. *IEEE Signal Process. Mag.*, 21(5):95 – 106, 2004. doi: 10.1109/MSP.2004.1328092. [Page lxix].

Oppenheim [1999] Alan Oppenheim. *Discrete-time signal processing*. Prentice Hall, Upper Saddle River, N.J, 1999. ISBN 0130834432. [Pages 18, 19, 21, 22, 27].

Oracle Corporation [2012] Open java development kit (OpenJDK), access date oct 2012. url: `http://openjdk.java.net`. [Page 49].

Orlarey et al. [2012] Functional audio stream (FAUST) (programming language version 0.9.46), access date oct 2012. url: `http://faust.grame.fr/`. [Page 5].

Paris et al. [2009] Sylvain Paris, Pierre Kornprobst, Jack Tumblin, and Fredo Durand. *Bilateral filtering theory and applications*. Now, Hanover, Mass, 2009. ISBN 160198250X. [Page 22].

Pernía et al. [2007] O. Pernía, J. M. Górriz, J. Ramírez, C. G. Puntonet, and I. Turias. An efficient VAD based on a generalized gaussian PDF. In *Proc. Int. Conf. Nonlinear Process.*, NOLISP'07, pages 246–254, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-77346-0, 978-3-540-77346-7. [Page lxiv].

Pham and van Vliet [2005] T. Q. Pham and L. J. van Vliet. Separable bilateral filtering for fast video preprocessing. In *IEEE Int. Conf. Multimedia Expo*, 2005. doi: 10.1109/ICME.2005.1521458. [Page xli].

Pirker et al. [2011] Gregor Pirker, Michael Wohlmayr, Stefan Petrik, and Franz Pernkopf. A pitch tracking corpus with evaluation on multipitch tracking scenario. In *Int. Conf. Speech Communication and Technol.*, 2011. [Page xxxii].

Pitas and Venetsanopoulos [1990] Ioannis Pitas and Anastasios N. Venetsanopoulos. *Nonlinear digital filters: principles and applications*. Kluwer Academic Publishers, Boston, 1990. ISBN 0792390490. [Page 20].

Porikli [2008] F. Porikli. Constant time o(1) bilateral filtering. In *Proc. IEEE. Comput. Soc. Conf. Comput. Vis. Pattern. Recognit.*, pages 1 –8, june 2008. doi: 10.1109/CVPR.2008.4587843. [Page xli].

Praat [2011] Praat (computer software), access date jan 2011. url: `http://www.fon.hum.uva.nl/praat/`. [Page xxix].

Puckette [1996] Miller Puckette. Pure data: another integrated computer music environment. In *Proc. Internat. Comput. Music Conf.*, pages 37–41, 1996. [Page 5].

Pullum [1996] Geoffrey Pullum. *Phonetic Symbol Guide*. University of Chicago Press, Chicago, 1996. ISBN 0226685365. [Page xxix].

Rabiner and Schafer [1978] Lawrence R. Rabiner and Ronald W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall, Englewood Cliffs, 1978. ISBN 0132136031. [Pages 20, lxxv, lxxvii].

Rabiner [1977] L. Rabiner. On the use of autocorrelation analysis for pitch detection. *IEEE T. Acoust. Speech.*, 25(1):24 – 33, February 1977. doi: 10.1109/TASSP.1977.1162905. [Page lxix].

Rabiner [2007] Lawrence Rabiner. *Introduction to digital speech processing*. Now, Boston, Mass, 2007. ISBN 1601980701. [Pages 1, 2].

Ramirez et al. [2004a] J. Ramirez, J. C. Segura, C. Benirez, A. de la Torre, and A. Rubio. A new voice activity detector using subband order-statistics filters for robust speech recognition. In *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, volume 1,2004. doi: 10.1109/ICASSP.2004.1326119. [Pages xxxviii, xxxix, xl, lxiv].

Ramirez et al. [2004b] J Ramirez, C. Angel de la Torre Segura, J. C. Benitez, and Antonio Rubio. Efficient voice activity detection algorithms using long-term speech information. *Speech. Commun.*, 42(3-4):271 – 287, 2004. doi: 10.1016/j.specom.2003.10.002. [Pages xlii, xliii, lxiv].

Raspberry Pi Foundation [2012] Raspberry pi (single-board computer), access date feb 2012. url: `http://www.raspberrypi.org/`. [Pages 49, 50].

Rost [2006] Randi J. Rost. *OpenGL shading language*. Addison-Wesley, Upper Saddle River, NJ, 2006. ISBN 0321334892. [Page 30].

Schroeder and Rossing [2007] M. Schroeder and Thomas D. Rossing. *Springer Handbook of Acoustics*. Springer, 2007. [Page xxviii].

Schroeder [1968] M. R. Schroeder. Period histogram and product spectrum: New methods for fundamental-frequency measurement. *J. Acoust. Soc. Am.*, 43(4):829–834, 1968. doi: 10.1121/1.1910902. [Page lxx].

Shimamura and Kobayashi [2001] T. Shimamura and H. Kobayashi. Weighted autocorrelation for pitch extraction of noisy speech. *IEEE T. Speech. Audi. P.*, 9(7):727–730, 2001. doi: 10.1109/89.952490. [Page lxviii].

Shuyin et al. [2009] Zhang Shuyin, Guo Ying, and Zhang Qun. Robust voice activity detection feature design based on spectral kurtosis. In *Int. Workshop Educ. Technol. Comput. Sci.*, volume 3, pages 269 –272, 7-8 2009. doi: 10.1109/ETCS.2009.587. [Pages xlvi, xlvii, xlviii].

Sinha [2010] Priyabrata Sinha. *Speech processing in embedded systems*. Springer, New York London, 2010. ISBN 9780387755809. [Page 3].

Sjőlander and Beskow [2010] Wavesurfer (computer program version 1.8.8), access date apr 2010. url: `http://www.speech.kth.se/wavesurfer`. [Page 3].

Sloetjes and Wittenburg [2008] H. Sloetjes and P. Wittenburg. Annotation by category - elan and iso dcr. In *Int. Conf. Lang. Resour. Eval.*, 2008. [Page xxix].

Smith [2007] Julius Smith. *Introduction to digital filters: with audio applications*. W3K Publishing, Stanford, 2007. ISBN 9780974560717. [Pages xlv, xlviii].

Sohn et al. [1999] Jongseo Sohn, Nam Soo Kim, and Wonyong Sung. A statistical model-based voice activity detection. *IEEE Signal Process. Let.*, 6(1):1–3, 1999. doi: 10.1109/97.736233. [Pages xxxviii, xlix, l, lxiv, lxxxix].

Sondhi [1968] M. Sondhi. New methods of pitch extraction. *IEEE T. Acoust. Speech.*, 16 (2):262 – 266, June 1968. doi: 10.1109/TAU.1968.1161986. [Page lxix].

Sreenivas [1984] T. V. Sreenivas. Pitch estimation of aperiodic and noisy speech signals. *Speech. Commun.*, 3(2):171, 1984. [Page lxix].

Staworko and Rawski [2010] M. Staworko and M. Rawski. FPGA implementation of feature extraction algorithm for speaker verification. In *IEEE Int. Conf. Mixed Des. Integr. Circuits Syst.*, pages 557 –561, june 2010. [Page xxviii].

Sun [2012] Sun Microsystems (company), access date jul 2012. url: `http://www.sun.com`. [Page 8].

Talkin [1995] D. Talkin. A robust algorithm for pitch tracking (RAPT). *Speech Coding and Synthesis*, 1995. [Pages xii, lxxix, lxxx].

Thulasiraman and Swamy [1992] K. Thulasiraman and M. N. S. Swamy. *Graphs: Theory and algorithms*. Wiley, New York, 1992. ISBN 9780471513568. [Pages 11, 12].

Tomasi and Manduchi [1998] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Int. Conf. Comp. Vis.*, pages 839 –846, jan 1998. doi: 10.1109/ICCV.1998.710815. [Pages xli, xlii].

Vary [2006] Peter Vary. *Digital speech transmission: Enhancement, coding and error concealment*. John Wiley, Chichester, England Hoboken, NJ, 2006. ISBN 0471560189. [Page lvii].

Vikstroem [2009] Alexander Vikstroem. A study of automatic translation of MATLAB code to C code using software from the mathworks. *Master's Thesis at Lulea University of Technology, Department of Computer Science and Electrical Engineering*, 2009. [Pages 7, 30].

Villalpando et al. [2011] C. Y. Villalpando, A. Morfopolous, L. Matthies, and S. Goldberg. FPGA implementation of stereo disparity with high throughput for mobility applications. In *IEEE Conf. Aerosp.*, pages 1–10, 2011. doi: 10.1109/AERO.2011.5747269. [Page xli].

Walker et al. [2004] Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel. Sphinx-4: A flexible open source framework for speech recognition. *Sun Microsystems Technical Report*, (TR-2004-139), November 2004. [Page 4].

Weiss et al. [1966] Mark R. Weiss, Reinhold P. Vogel, and Cyril M. Harris. Implementation of a pitch extractor of the double-spectrum-analysis type. *J. Acoust. Soc. Am.*, 40(3):657–662, 1966. doi: 10.1121/1.1910131. [Page lxix].

Wells [1985] B. Wells. Voiced/unvoiced decision based on the bispectrum. In *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, volume 10, pages 1589–1592, 1985. doi: 10.1109/ICASSP.1985.1168198. [Page lxxii].

Wendykier [2011] Jtransforms (software library version 2.4), access date jul 2011. url: `http://sourceforge.net/projects/jtransforms/`. [Pages 4, 25].

Wright et al. [2011] Richard S. Wright, Nicholas Haemel, Graham Sellers, and Benjamin Lipchak. *OpenGL Superbible: Comprehensive Tutorial and Reference*. Addison-Wesley, Upper Saddle River, NJ, 2011. ISBN 0321712617. [Page 30].

Wu and Zhang [2011] Ji Wu and Xiao-Lei Zhang. An efficient voice activity detection algorithm by combining statistical model and energy detection. *EURASIP J. Adv. Signal. Process.*, 2011(1):18, 2011. doi: 10.1186/1687-6180-2011-18. [Page lxiv].

Yang et al. [2009] Jie Yang, Sheng Sheng Yu, and Jingli Zhou. The implementation and optimization of amr on mobile device. *Journal of Sofware*, 4, 2009. [Page xli].

Yang et al. [2010a] Ke-Hu Yang, Si-Yao Fu, and Ying-Gui Shi. Other approaches to obtain the fast and piecewise-linear bilateral filter in 3D space. In *Proc. Int. Network. Sensing Control Conf.*, pages 133–137, 2010. doi: 10.1109/ICNSC.2010.5461521. [Pages xxxiii, xxxvi, xxxvii].

Yang et al. [2010b] Xiaoling Yang, Baohua Tan, Jiehua Ding, Jinye Zhang, and Jiaoli Gong. Comparative study on voice activity detection algorithm. In *Int. Conf. Elect. Control Eng.*, pages 599–602, 2010. doi: 10.1109/iCECE.2010.153. [Page xli].

Yu et al. [2010] Jing Yu, Chuangbai Xiao, and Dapeng Li. Physics-based fast single image fog removal. In *Proc. IEEE 10th Int Signal Processing (ICSP) Conf*, pages 1048–1052, 2010. doi: 10.1109/ICOSP.2010.5655901. [Page xli].

Appendices

Part I

Verification Scripts

A G729B Based VAD

```
%% read input data
filename = '/mnt/sdcard/m01_8000Hz.wav';
[x, fs] = wavread(filename);

%normalize input data
x=x/max(abs(x));

%% get reference signal
ENABLE.DENOISE_ITERATIONS = 0;
ENABLE.USE_POSTPROCESSING = 0;
vad_ref = getVAD( [x;zeros(80*3,1)], fs , 'g729b' , ENABLE );

%% matlab rtblock version
%...rtblock source
source=RTSource_FromBuffer01();
source.setBuffer(x);

%...rtblock vad extractor
vad_extractor_matlab=RTSink_Rapf_VadG729B01();
vad_extractor_matlab.setStepsizeInSamples(80/3);

%...rtblock buffer sink
sink=RTSink_ToBuffer01();

%...interconnect
source.addSink(vad_extractor_matlab);
vad_extractor_matlab.addSink(sink);

%...process
source.init().run_().flush_().finalize_();

%...get result
vad_est_matlab = sink.getBuffer();

%% java rtblock version
%...rtblock source
source=RTBlocks.Standard.RTSource_FromBuffer01();
source.setBuffer(x);

%...rtblock vad extractor
vad_extractor_java=RTBlocks.Rapf.RTSink_Rapf_VadG729B01();
vad_extractor_java.setStepsizeInSamples(80/3);

%...rtblock buffer sink
sink=RTBlocks.Standard.RTSink_ToBuffer01();

%...interconnect
source.addSink(vad_extractor_java);
vad_extractor_java.addSink(sink);
```

```
50
51 %...process
52 source.init().run_().flush_().finalize_();
53
54 %...get result
55 vad_est_java = sink.getBuffer();
56
57 %% crop following zeros
58 N=length(vad_ref);
59 vad_est_matlab = vad_est_matlab(1:N);
60 vad_est_java = vad_est_java(1:N);
61
62 %% get diff
63 ndiff_matlab = sum(abs(vad_ref - vad_est_matlab'));
64 ndiff_java = sum(abs(vad_ref - vad_est_java'));
65 fprintf('differences: ref vs. rtblocks@matlab: %d samples\n',ndiff_matlab);
66 fprintf('differences: ref vs. rtblocks@java: %d samples\n',ndiff_java);
67
68 %% plot result
69 figure('Name','G729B based VAD (Reference Implementation vs. RTBlocks
                                Version)');
70 subplot(4,1,1); plot(x); title 'Input Signal';
71 subplot(4,1,2); plot(resample2unique(vad_ref,x),'r');
72 title 'VAD Refererence Signal'; ylim([0,1.1]);
73 subplot(4,1,3); plot(resample2unique(vad_est_matlab,x),'g');
74 title 'RTBlocks VAD (Matlab Version)'; ylim([0,1.1]);
75 subplot(4,1,4); plot(resample2unique(vad_est_java,x),'b');
76 title 'RTBlocks VAD (Java Version)'; ylim([0,1.1]);
77
78 %% store
79 [~,fname]=fileattrib('~/temp/evalRTBlockG729B01.pdf');
80 format_fig(gcf,'LineWidth',4,'FontSize',20,'Position',[0 0 1 1]);
81 export_fig(fname.Name, gcf);
```

B Variance Based VAD

```
%% read input data
filename = '/mnt/sdcard/m01_8000Hz.wav';
[x, fs] = wavread(filename);

%normalize input data
x=x/max(abs(x));

%% get reference signal
ENABLE.USE_NORM=0;
ENABLE.USE_POSTPROCESSING = 0;
ENABLE.DENOISE_ITERATIONS = 0;
SETTINGS.BILAT_N = 1;
vad_ref = getVAD( x, fs , 'bilat' , ENABLE , SETTINGS);

%% matlab rtblock version
%...rtblock source
source=RTSource_FromBuffer01();
source.setBuffer(x).setPagesizes(512,512);

%...rtblock f0 extractor
vad_extractor=RTSink_Rapf_VadBilateralV01();
vad_extractor.setStepsizeInFrames(1/16);

%...rtblock buffer sink
sink=RTSink_ToBuffer01();

%...interconnect
source.addSink(vad_extractor);
vad_extractor.addSink(sink);

%...process
source.init().run_().flush_().finalize_();

%...get result
vad_est_matlab = sink.getBuffer();

%% java rtblock version
%...rtblock source
source=RTBlocks.Standard.RTSource_FromBuffer01();
source.setBuffer(x).setPagesizes(512,512);

%...rtblock f0 extractor
vad_extractor_java=RTBlocks.Rapf.RTSink_Rapf_VadBilateralV01();
vad_extractor_java.setStepsizeInFrames(1/16);

%...rtblock buffer sink
sink=RTBlocks.Standard.RTSink_ToBuffer01();

%...interconnect
```

```
50 source.addSink(vad_extractor_java);
51 vad_extractor_java.addSink(sink);
52
53 %...process
54 source.init().run_().flush_().finalize_();
55
56 %...get result
57 vad_est_java = sink.getBuffer();
58
59 %% crop following zeros
60 N=length(vad_ref);
61 vad_est_matlab = vad_est_matlab(1:N);
62 vad_est_java = vad_est_java(1:N);
63
64 %% get diff
65 ndiff_matlab = sum(abs(vad_ref - vad_est_matlab'));
66 ndiff_java = sum(abs(vad_ref - vad_est_java'));
67 fprintf('differences: ref vs. rtblocks@matlab: %d samples\n',ndiff_matlab);
68 fprintf('differences: ref vs. rtblocks@java: %d samples\n',ndiff_java);
69
70 %% plot result
71 figure('Name','Variance based VAD (Reference Implementation vs. RTBlocks
                                Version)');
72 subplot(4,1,1); plot(x); title 'Input Signal';
73 subplot(4,1,2); plot(resample2unique(vad_ref,x),'r');
74 title 'VAD Refererence Signal'; ylim([0,1.1]);
75 subplot(4,1,3); plot(resample2unique(vad_est_matlab,x),'g');
76 title 'RTBlocks VAD (Matlab Version)'; ylim([0,1.1]);
77 subplot(4,1,4); plot(resample2unique(vad_est_java,x),'b');
78 title 'RTBlocks VAD (Java Version)'; ylim([0,1.1]);
79
80 %% store
81 [~,fname]=fileattrib('~/temp/evalRTBlockVadBilateral01.pdf');
82 format_fig(gcf,'LineWidth',4,'FontSize',20,'Position',[0 0 1 1]);
83 export_fig(fname.Name, gcf);
```

C ACF Based F0

```
%% read input data
filename = '/mnt/sdcard/m116_8000Hz.wav';
[x, fs] = wavread(filename);

%normalize input data
x=x/max(abs(x));

%% get reference signal
ENABLE.DENOISE_ITERATIONS = 0;
f0_ref = getF0( x, fs , 'xcorr1' , ENABLE );

%% matlab rtblock version
%...rtblock source
source=RTSource_FromBuffer01();
source.setBuffer(x).setPagesizes(1600,1600);

%...rtblock f0 extractor
f0_extractor_matlab=RTSink_Rapf_F0Xcorr01V01();
f0_extractor_matlab.setStepsizeInFrames(1/9);

%...rtblock buffer sink
sink=RTSink_ToBuffer01();

%...interconnect
source.addSink(f0_extractor_matlab);
f0_extractor_matlab.addSink(sink);

%...process
source.init().run_().flush_().finalize_();

%...get result
f0_est_matlab = sink.getBuffer();
f0_est_matlab = f0_est_matlab(3:end);

%% java rtblock version
%...rtblock source
source=RTBlocks.Standard.RTSource_FromBuffer01();
source.setBuffer(x).setPagesizes(1600,1600);

%...rtblock f0 extractor
f0_extractor_java=RTBlocks.Rapf.RTSink_Rapf_F0Xcorr01V01();
f0_extractor_java.setStepsizeInFrames(1/9);

%...rtblock buffer sink
sink=RTBlocks.Standard.RTSink_ToBuffer01();

%...interconnect
source.addSink(f0_extractor_java);
f0_extractor_java.addSink(sink);
```

```
50
51 %...process
52 source.init().run_().flush_().finalize_();
53
54 %...get result
55 f0_est_java = sink.getBuffer();
56 f0_est_java= f0_est_java(3:end);
57
58 %% crop following estimation
59 N=length(f0_ref);
60 f0_est_matlab=f0_est_matlab(1:N);
61 f0_est_java=f0_est_java(1:N);
62
63 %% get diff
64 max_diff_matlab = max(abs(f0_ref' - f0_est_matlab));
65 max_diff_java = max(abs(f0_ref' - f0_est_java));
66 fprintf('differences: ref vs. rtblocks@matlab: %f\n',max_diff_matlab);
67 fprintf('differences: ref vs. rtblocks@java: %f\n',max_diff_java);
68
69 %% plot result
70
71 lw=3;
72 figure('Name','Correlation based F0 Estimation: (Reference Implementation vs.
                                RTBlocks Version)');
73 subplot(2,1,1); plot(x); title 'Input Signal';
74 h2=subplot(2,1,2); hold on; title 'F0 Signals';
75 set(findall(gcf,'Type','line'),'LineWidth',lw)
76 h1=plot(f0_ref,'r','LineWidth',lw*3);
77 plot(f0_est_matlab,'b','LineWidth',lw*2);
78 plot(f0_est_java,'Color',[0 0.5 0],'LineWidth',lw*1);
79 ylabel('Frequency [Hz]');
80 legend('F0 Reference', 'RTBlocks F0 Estimation, Matlab Version',...
81     'RTBlocks F0 Estimation, Java Version', 'Location', 'SouthEast');
82
83 %% store
84 [~,fname]=fileattrib('~/temp/evalRTBlockF0Xcorr01.pdf');
85 format_fig(gcf,'FontSize',20,'Position',[0 0 1 1]);
86 format_fig(h2,'MarkerSize',15,'MarkerLineWidth',2, ...
87     'MarkerSpacing', 5);
88 export_fig(fname.Name, gcf);
```

D YIN's F0

```
%% read input data
filename = '/mnt/sdcard/m116_8000Hz.wav';
[x, fs] = wavread(filename);

%normalize input data
x=x/max(abs(x));

%% get reference signal
ENABLE.DENOISE_ITERATIONS = 0;
f0_ref = getF0( [x ; zeros(1600,1)], fs , 'yinSphinx' , ENABLE );
f0_ref = f0_ref(3:end);

%% matlab rtblock version
%...rtblock source
source=RTSource_FromBuffer01();
source.setBuffer(x).setPagesizes(1600,1600);

%...rtblock f0 extractor
f0_extractor_matlab=RTSink_Rapf_F0YinV01();
f0_extractor_matlab.setStepsizeInFrames(1/9);

%...rtblock buffer sink
sink=RTSink_ToBuffer01();

%...interconnect
source.addSink(f0_extractor_matlab);
f0_extractor_matlab.addSink(sink);

%...process
source.init().run_().flush_().finalize_();

%...get result
f0_est_matlab = sink.getBuffer();
f0_est_matlab = f0_est_matlab(5:end);

%% java rtblock version
%...rtblock source
source=RTBlocks.Standard.RTSource_FromBuffer01();
source.setBuffer(x).setPagesizes(1600,1600);

%...rtblock f0 extractor
f0_extractor_java=RTBlocks.Rapf.RTSink_Rapf_F0Yin01();
f0_extractor_java.setStepsizeInFrames(1/9);

%...rtblock buffer sink
sink=RTBlocks.Standard.RTSink_ToBuffer01();

%...interconnect
source.addSink(f0_extractor_java);
```

```
50 f0_extractor_java.addSink(sink);
51
52 %...process
53 source.init().run_().flush_().finalize_();
54
55 %...get result
56 f0_est_java = sink.getBuffer();
57 f0_est_java= f0_est_java(5:end);
58
59 %% crop following estimation
60 N=min([length(f0_ref), length(f0_est_matlab), length(f0_est_matlab)]);
61 f0_ref=f0_ref(1:N);
62 f0_est_matlab=f0_est_matlab(1:N);
63 f0_est_java=f0_est_java(1:N);
64
65 %% get diff
66 max_diff_matlab = max(abs(f0_ref' - f0_est_matlab));
67 max_diff_java = max(abs(f0_ref' - f0_est_java));
68 fprintf('differences: ref vs. rtblocks@matlab: %f\n',max_diff_matlab);
69 fprintf('differences: ref vs. rtblocks@java: %f\n',max_diff_java);
70
71 %% plot result
72
73 lw=3;
74 figure('Name','Correlation based F0 Estimation: (Reference Implementation vs.
                              RTBlocks Version)');
75 subplot(2,1,1); plot(x); title 'Input Signal';
76 h2=subplot(2,1,2); hold on; title 'F0 Signals';
77 set(findall(gcf,'Type','line'),'LineWidth',lw)
78 h1=plot(f0_ref,'r','LineWidth',lw*3);
79 plot(f0_est_matlab,'g','LineWidth',lw*2);
80 plot(f0_est_java,'b','LineWidth',lw*1);
81 ylabel('Frequency [Hz]');
82 legend('F0 Reference', 'RTBlocks F0 Estimation, Matlab Version',...
83     'RTBlocks F0 Estimation, Java Version', 'Location', 'SouthWest');
84
85 %% store
86  [~,fname]=fileattrib('~/temp/evalRTBlockF0Yin01.pdf');
87  format_fig(gcf,'FontSize',20,'Position',[0 0 1 1]);
88  format_fig(h2,'MarkerSize',15,'MarkerLineWidth',2, ...
89      'MarkerSpacing', 5);
90  export_fig(fname.Name, gcf);
```

E Python Control Script for Android Testbench

```
'''
Created on Aug 29, 2012

@author: chklug
'''
from rtblocksControl import RTBlocksControl
import sys

def startActivity(control=None):
    #start activity
    activity = ".Test.Activities.Tests4MasterThesisGL01"
    print "starting activity..."
    if not control:
        control = RTBlocksControl()
        control.connectDevice()
    control.unlockDevice()
    control.startActivity(None, activity)
    print "activity started."
    return control

def stopActivity(control):
    #stop activity
    control.stopActivity()
    print "done."

def startTest(control, testname):
    print "starting test %s" % testname
    control.startTest(testname)

def main(args=None):
    #android RTBlocks testbench contrl
    control = None

    #RTBlocks tests
    testnames = ("Test_RapfVADG729B_stepsize1d3",
                 "Test_RapfVADBilateral_stepsize1d16",
                 "Test_RapfF0Yin_stepsize1d9",
                 "Test_RapfF0Xcorr01_stepsize1d9")

    #RTBlocks test execution time (sec, upper limit)
    delays = (10, 50, 5, 5)

    #run tests
    for test_idx, testname in enumerate(testnames):
        filename = "./generated/%s.png" % testname
        my_delay = delays[test_idx]

        #start activity
        control = startActivity(control)
```

```
50
51          #start test
52          startTest(control, testname)
53
54          #delay
55          print "sleeping for %d seconds" % my_delay
56          control.sleep(my_delay)
57          control.screenShot(filename)
58
59          #stop activity
60          stopActivity(control)
61
62      #done
63      print "done"
64
65  if __name__ == "__main__":
66      main(sys.argv)
```

Part II

Project Design Document (Project RAPF)

Project Design Documents

Rhetorical Acoustic Phonetic Feedback Device (RAPF)

Christoph Klug

Graz, October 1, 2012

The goal of this project is a wearable device which should support people with language difficulties during conversations, phone talks or speakings. A seamless integrated device can provide additional feedback about speech properties.

Another goal of this work is the vision that voice education can be seamlessly integrated into daily routines. Nowadays human supported lessons as well as special computer programs are required to train the required skills of e.g. hearing impaired people, so that they are able to tune their voice in an accurate way. The training has to be repeated regularity. The proposed wearable acoustic phonetic feedback device can act as a bridge between two training sessions. I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

F Introduction

In this chapter the motivation for the project is described.

F.1 Motivation

Hearing impaired people often have problems with controlling the pitch and volume of their voice. This problems can lead to feelings like insecurity, unhappiness and otherness. Other language difficulties lead to similar problems.

With computer programs and human support they can train to tune their voice in an accurate way. The training has to be repeated regularity.

The proposed device can act as a bridge between two training sessions.

G Design

In this chapter both, the project management and the technical project design are described.

G.1 PID

Project Initiation Form (PID)	
Project Title:	RAPF
Project Background:	Hearing-impaired persons may have problems with keeping their voices in an accurate pitch and volume. Other language difficulties are also very common. This device gives such people feedback about the acoustic phonetics of their voices.
Project Benefits:	A wearable real-time feedback device will help hearing impaired persons and persons with language difficulties to deal with their problems during conversations. This device also can support common speech training sessions. NOTE: None-profit-project!
Project Objectives:	A mechanical, visual and/or acoustical feedback should be provided when the voice of the device wearer doesn't match the acoustic phonetic constraints. The device should be seamless integrated into a common outfit. An alternative would be the implementation as smartphone application.
Project Deliverables:	A prototype for the feedback device.
This project will include: • Design, simulations and a prototypes. • The exploration and implementation of the acoustic phonetic parameter extraction algorithms are the central tasks. • Simple context recognition, echo canceler and voice-to-speaker affinity recognition may be added, if required.	

This project will not include: • No extensive software/driver bundle would be designed. • Hardware implementations of the algorithms are restricted to the basic phonetic speech features. Additional evaluation remains in software. • Hardware may be only explored within Matlab context. The standalone prototype is optional. • The product itself also is an optional part.	
Success Criteria:	A robust concept and a working prototype (laboratory environment).
Constraints:	Real-time, noisy environments, acoustic speech features, linguistic speech features, feedback channel
Key Assumptions:	The focus is on low-cost, low-power consumption and wearability (⇒usage of simple algorithms).
Project Manager:	Christoph Klug
Project Sponsor:	Christoph Klug

Table G.1: Project Definition Form [PID]

G.2 Basic Design

G.3 Application Specific Hardware

Audio data is captured and filtered by a wearable device. From the data both, the context and the acoustic parameters are derived. If the acoustic parameters don't match the parameter set related to the determined context, feedback is provided (e.g. vibrator as mechanic feedback, sound as acoustic feedback or special visual feedback).
The wearable device should be able to integrate seamless in a common outfit. If it is possible, it would be integrated in a necklet (figure G.2) or a bracelet.
Therefore both, the amount of sensors and the size of the control unit have to be minimized. Further the overall power consumption and the source voltage are basic parameters which have to be considered for a real product.
A rude block-diagram of the product is given in figure G.3.

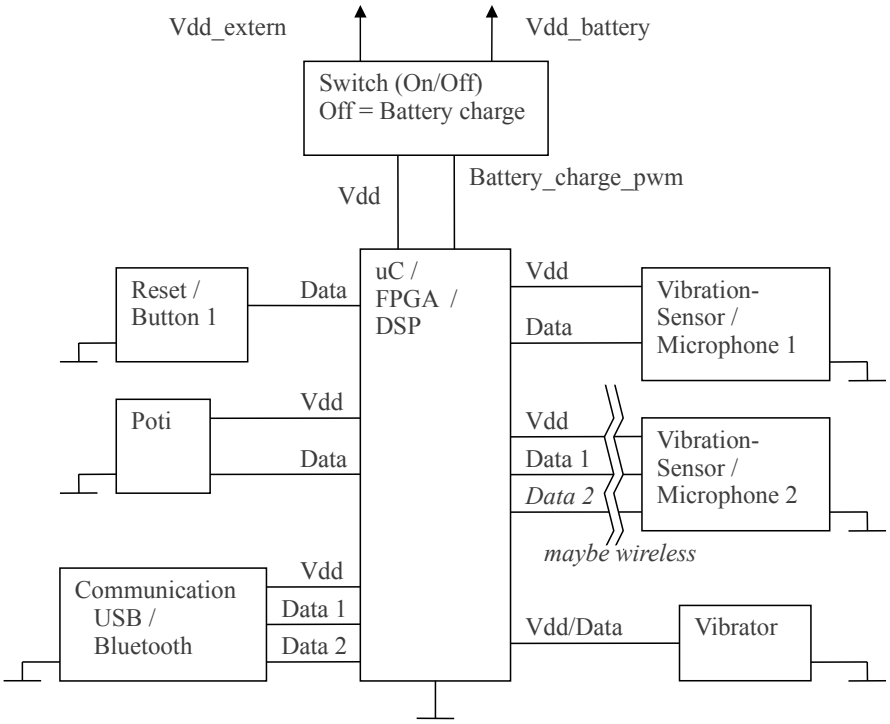


Figure G.1: Design Phase: Not more than 50 to 75 percent of the control unit's resources should be allocated. We have 13 IO pins in the base design, so three IO ports would be required ($3 \cdot 8 = 24$ pins). Such a device is way too huge for a product so significant reductions have to be considered. The goal is to use just eight pins for the control unit.

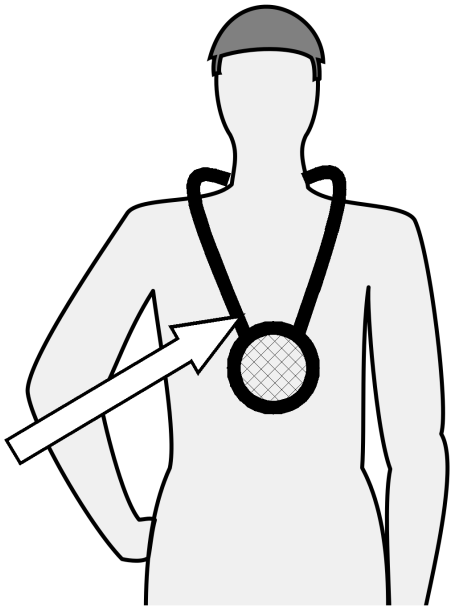


Figure G.2: A possible package for the device could be a necklet or a bracelet. Seamless integration is very important to get acceptance for the product. (The body silhouette is taken from `http://www.openclipart.org`)

G.4 Data Acquisition

The main idea is to capture audio data from two microphones: One microphone measure the reflected audio signal (figure G.3), the second one should measure the voice from the bone conduction (body vibrations). With this data the acoustic phonetic parameters of the speaker's voice should be determined. Adaptive filters are probably required to extract the speech data from the noisy signals (figure G.4).

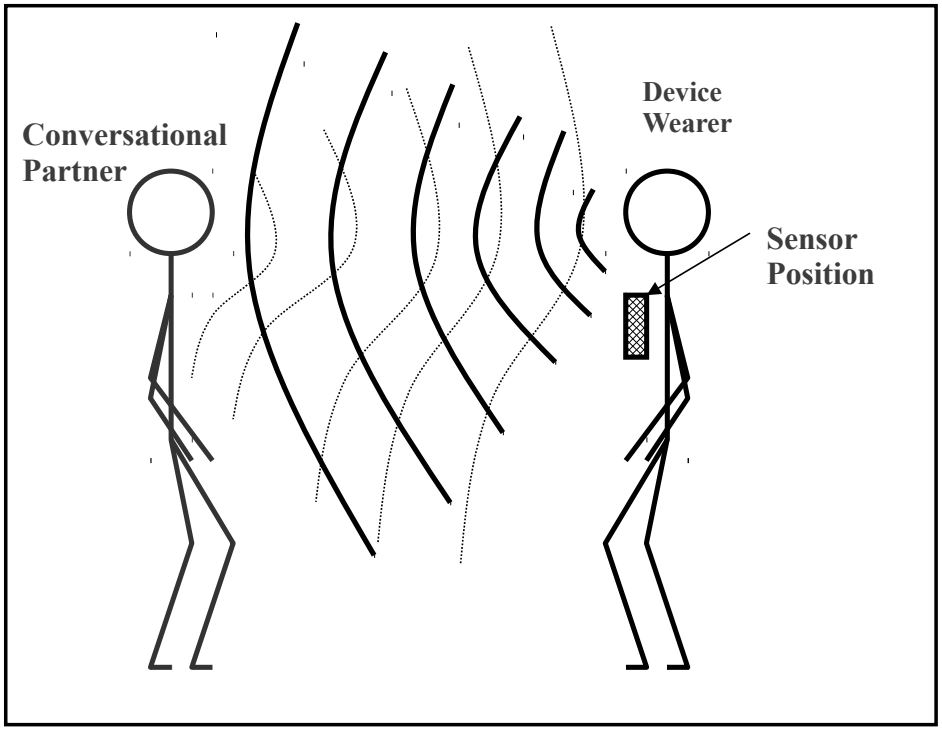


Figure G.3: The main idea is to capture the reflected audio signal. Audio-channel estimations shouldn't be necessary in this step. Nevertheless by adding a single actor (simple SMD piezo buzzer) these estimations can be done. Also the feedback vibrator itself maybe produces a silent, but suitable sound for channel estimations.

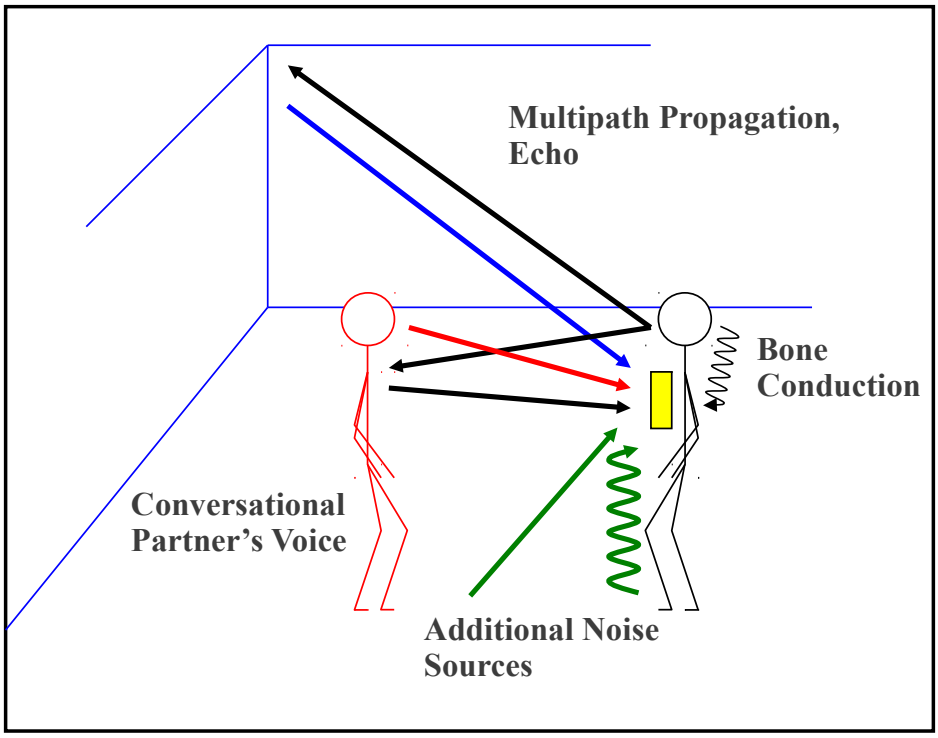


Figure G.4: The main challenge of the project is to estimate the correct audio signal. Adaptive filters and at least two sensors are required for this. In the worst case scenario the second microphone (bone conduction measurement) needs to be placed at the back of the neck.

G.5 Smartphone Device

When using a common smartphone, all required peripheral equipment is provided. The final product could be delivered as smartphone application.

G.6 Feedback Mechanism

Modern technology offers several kinds of feedback-mechanism. One possibility is a visual feedback, using VGA-integrated glasses. Another method is mechanic feedback by a vibrator like used in mobile phones. In respect of cost constraints (low-cost product) the vibrator feedback would be preferred. Additional acoustic feedback provides a huge range of possibilities, from simple replay of recorded audio signals (sound signals or spoken information) towards to the replay of the current voice with exaggerated changes of the actual critical parameters.

G.7 Context Recognition

Depending on the actual situation, the proper acoustic phonetics differs in several parameters. Simple context recognition based on given sensor data is required:

Single person context: The device has do guess location, activity and/or actual rule of the device's wearer.

- Examples for activities: Sing (drop data), sleep (drop data), phone (accept data), converse (accept data)

- Examples for location: Home, sleeping room, cinema, discotheque
- Examples for wearer's rule: Group member/speaker (presentation parameter set), group member/listener (drop data/whisper parameter set), ...

Two person's context: The device has to guess who is speaking and what kind of conversation it is.

- Possible speaker: device wearer (accept data), conversational partner (drop data), both (drop data)
- Kind of conversation: talk in a whisper (whisper parameter set), normal conversation (conversation parameter set), quarrel (conversation parameter set)

G.8 Product Components

G.8.1 Smartphone Application

When targeting smartphones only software needs to be delivered. Additional input/output extensions like a neck-microphone or professional haptic feedback devices are optional.

G.8.2 Application Specific Hardware

The whole product should fit onto a tiny Printed Circuit Board (PCB) with very few input/output connectors. Figure G.5 shows a possible PCB composition.

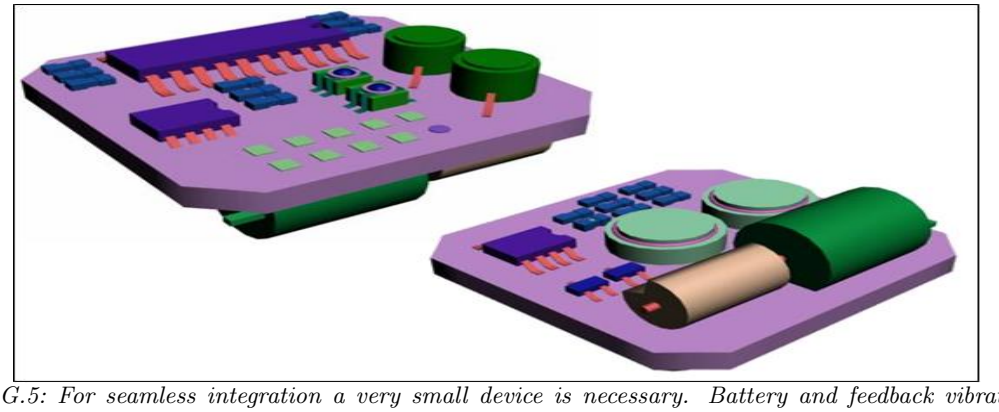


Figure G.5: For seamless integration a very small device is necessary. Battery and feedback vibrator are very big components and will lead to serious space problems. Also the control device would need some external parts (filters and amplifiers).

G.8.3 Product Packages

The device should be wearable without attracting much attention. When targeting smartphones no additional packing is required. Figure G.6 shows a possible package for an application specific hardware.

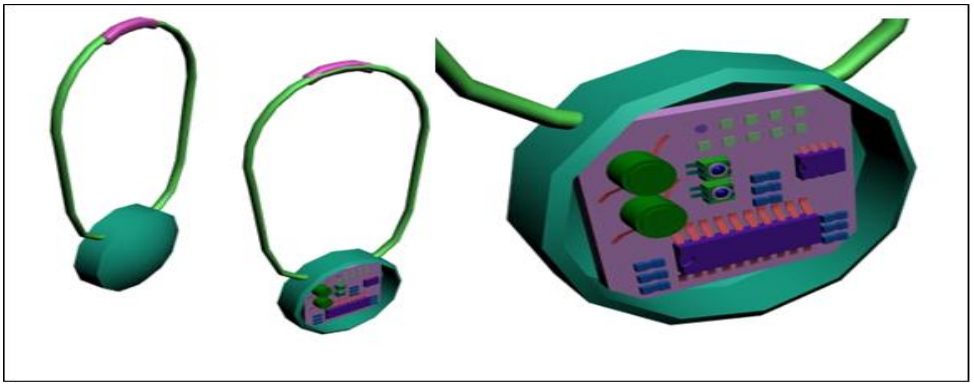


Figure G.6: The preferred package considers the case if a microphone at the back of the neck is required. An alternative package would be a bracelet with two microphones for the air-transmitted audio signal and one microphone for the bone-transmitted signal.

Part III

Preliminary Algorithm Study (Master Practical, Project RAPF)

Graz University of Technology

Master Project - Telematics

RAPF

Preliminary Algorithm Study

conducted at the
Signal Processing and Speech Communications Laboratory
Graz University of Technology, Austria

by
Christoph Klug

Supervisors:
Dipl.-Ing. Dr.techn. Martin Hagmüller
Assoc.Prof. Dipl.-Ing. Dr.mont. Franz Pernkopf

Graz, October 1, 2012

H Introduction

H.1 Motivation

Voice training became a major problem in voice and speech research disciplines. Medical purposes due to voice disorders, linguistic usage due to public speaking training or educational training are only a few examples where voice analysis and voice training are essential parts. Education, training and analysis of someone's voice is nowadays supported by assessment sheets, audio or video recordings or even voice measurement and visualization tools. Most of these tools require a sufficient education in linguistics and phonetics and therefore are restricted in their application fields. Reliable voice measurement and feedback tools tend to be very expensive because they are not designed for home-usage. This work is based on the vision that voice education can be seamlessly integrated into daily routines.

To come one step closer to this dream, reliable real-time speech analysis as well as linguistic speech feature extraction and evaluation need to be automatized. In Anderson [1977] both, a linguistic and a technical point of view are provided.

The chosen starting point for solving the technical tasks of the problem is the discussion of different speech analysis algorithms for two important speech features: Fundamental frequency (F0) estimation and voice activity detection (VAD). The results are summarized in this work.

H.2 Overview

This document is structured as follows: In section H.3, a brief summary of the history of speech processing as well as related current technological achievement are given. Speech data gathering, storage and suitable preparation for further analysis and used databases are discussed in chapter I and chapter J. Algorithm comparison and evaluation are provided in chapter K and chapter L. Finally, the results of the work are summarized in chapter M.

H.3 Related Work

Acoustics, the science of sound, has a long history. A specialized technical area of this huge research field deals with speech acoustics and speech processing. In this section some important historical experiments and some current research results are mentioned.

H.3.1 History

One of the first motivation for doing research in speech processing and communication was the goal of building mechanical models which emulate human verbal communication capabilities (Juang and Chen [1998]). An early attempt of this type is proposed in Kempelen [1791].

Among others, Wheatstone, Faber, Bell, Paget and Riesz followed this early approach of physically based source-tract modeling for speech production.

Helmholtz, Miller, Koenig and others invented an alternative approach in the 20^{th} century, synthesizing vowels by mathematical sinusoidal models (Schroeder and Rossing [2007]).

For more than 100 years, research in speech signal processing was founded on these two models. Today, research in speech processing has gone far beyond the simple models due to the advances in mathematical tools, computers and application fields.

A particularly important instrument for speech analysis is the *sound spectrograph*, originally developed at the Bell Telephone Laboratories around 1945. Nowadays the *speech spectrogram* is one of the most important analysis and visualization tools in speech processing.

H.3.2 State of the Art

Mainstream products like smartphones, navigation systems, hearing aids, digital cameras and lots of others require signal processing capabilities. Requirements like low price, energy efficiency, robustness and high-speed or high-bandwidth capabilities boosted the development of digital signal processors (DSPs) and field programmable gate arrays (FPGAs), which generated huge fields of applications in the last decade.

In Manfredi et al. [2008] the authors acknowledged the importance of the new possibilities. They implemented a portable voice-monitoring device on a digital signal processor (DSP) platform under a clinical perspective. The analysis of important voice quality features (e.g. F0, jitter...) should support patients carryover therapy goals outside the clinical environment. The real-time feedback is highly limited, which made off-line analysis necessary.

An earlier study (Howard et al. [2005]) evaluated the benefits of a real-time display providing more features (e.g. spectrogram, spectrum, spectral ratio, acoustic pressure waveform...). Experiments are designed for two singing teachers, aiming benefits for learning vocal skills. They introduced a real-time software system called *WinSingad*, which cannot only support singing education, but is also useful for clinical purposes.

In Heckmann et al. [2011] the authors are discussed the benefits using auditory and visual processing instead of traditional speech features extraction. The usage of image processing techniques are mentioned to solve common speech processing problems, like speech vs. non-speech discrimination, source separation or automatic speech recognition (ASR).

A user-friendly voice analysis tool called *BioVoice* is evaluated for clinical purposes in Manfredi et al. [2009]. The system estimated only a few features, focusing robustness and automatic evaluation with very view user settings. The results are compared to a common commercial voice analysis tool which performed *quantitative acoustic assessment of voice quality* (Kay Elemetrics Corp. [1994]).

Jung et al. [2010], Staworko and Rawski [2010], Min [2009] and Hong and Yusoff [2008] focused on speech feature extraction algorithms designed for *field programmable gate arrays (FPGAs)*. Several performance improvements are reported, which is a great step towards real-time capability of resource-intensive algorithms. Another, conceptually similar approach is presented in Dixon et al. [2009],

I Transcription and Annotation of Speech

I.1 Introduction

For evaluation of speech analysis methods more information is required than spoken language can provide. Orthographic and phonetic transcriptions of spoken language are basic steps towards enhanced speech analysis. Popular phonetic datasets like the *acoustic-phonetic continuous speech corpus (TIMIT)* provides both, phonetically and lexically transcribed speech (Garofolo et al. [2011]).

I.2 Speech Transcription and Speech Annotation

Orthographic notation is based on grammatic orthography rules of a certain language. Orthographic transcription is a way less time-consuming than phonetic transcription. Unambiguous assignments of speech sounds to related orthographic signs are often not possible (Hall [2000, page 2]).

Phonetic notation is the visual representation of speech sounds. The most common used phonetic alphabet is the International Phonetic Alphabet (International Phonetic Association [2011]). An huge overview of phonetic symbols and phonetic notation systems is given in Pullum [1996].

I.2.1 Transcription and Annotation Software

Many software tools exist for supporting speech annotation, orthographic and/or phonetic transcription and acoustic-phonetic analysis. Based on Hartung [2006] the following tool set is proved to cover the linguistic requirements of this work:

- Transcription and annotation software: F4, Transcriber (Audiotranskription F4 [2011], Audiotranskription Transcriber [2011])
- Audio meta-data database software: European distributed corpora project linguistic annotator (ELAN) (Max-Planck-Institute [2011], Sloetjes and Wittenburg [2008])
- Acoustic phonetic analysis software: PRAAT (Praat [2011])

Some Matlab and Java based extensions of ELAN enable additional search-and-extract capabilities as well as special import and export features.

I.3 Discussion and Conclusion

Speech database creation and maintenance can be well-supported by suitable software tools. As it is expected that non-technical researchers and end-users make use of the database, convenient

handling is important. Providing information about speech samples and corresponding meta-data via a single graphical user interface, using a well-defined human readable database format and sticking to well-established software will meet this requirement, which will be considered in the follow-on project.

J Speech Data

J.1 Introduction

Widespread types of speech records are very important for the development and test of robust speech feature algorithms. This requires recording and conditioning of field specific speech samples as well as the usage of common acoustic phonetic speech databases. On the other hand, gathering and conditioning speech acts is highly labour-intensive, so restriction to special speech acts are necessary.

J.2 Political Speech Databases

Political speech acts seem to be well-suited for field tests of many algorithms for speech analysis. The audio records are available online so no significant legally restricts are expected. Many textual transcripts are provided which support the conditioning. The algorithms can be tested with the voice of men and women, using different speech acts, with different background noise and with different emotional influences.

J.2.1 Online Resources

Austrian parliament live streams are recorded from Austrian Parliament [2011], German parliament live streams are recorded from German Parliament [2011]. Germany provides a well-organized video archive where speeches can be reviewed.

J.2.2 Recording Audio Streams

Audio tracks are extracted from recorded video streams. Mplayer (MPlayer [2011]) is used for both, recording and extraction.

J.2.3 Discussion and Conclusion

Live streams don't support very high speech quality, especially if the speech samples are audio tracks of online video streams. Low quality and low sampling rates are essential problems. Another drawback is the lack of the absolute speech energy information. This feature is only reliable if the whole recording path is well-known. Background noise and speech interjections are very common during public speeches. Orthographic transcriptions, if available, are provided in a "cleaned" form: no interjections are documented, slips of the speaker's tongue are dropped and malpositions of words are corrected. Therefore fully automated conditioning of the audio data with the corresponding orthographic information is not part of this work.

Even if field tests of the algorithm are very important, the development and evaluation of speech analysis algorithm require a ground truth signal for the different speech features, which is not available from live stream recordings. Thus, additional speech databases are used.

J.3 PTDB

The Signal Processing and Speech Communication Laboratory of Graz University of Technology (SPSC Lab) provides different speech corpora free of charge for research purposes. Laryngograph signals are extremely useful for evaluation of pitch determination algorithms (PDAs), VAD algorithms and speaking rate (SR) algorithms. The pitch tracking database from Graz University of Technology (PTDB-TUG) provides audio recordings from 20 English native speakers, reading sentences from the existing TIMIT database (see section J.4). Beside the laryngograph signals extracted pitch trajectories are provided as reference (Pirker et al. [2011]).

J.4 TIMIT

The TIMIT database (Garofolo et al. [2011]) is a hand-verified collection of phonetically and lexically transcribed sentences, spoken from American English speakers, including different sexes and dialects. This database of read speech is especially designed for acoustic phonetic studies and for ASR tasks.

J.5 Noise Database

Tests in realistic environments usually are time-consuming or cost expensive. To simulate realistic environments in the laboratory, background noise can be added to clean speech audio tracks. One source of different noise records is proposed in Johnson and Shami [1993], where natural and synthetic noise types are available for free.

K Voice Activity Detection

K.1 Introduction

Many speech analysis techniques need the knowledge about the boundaries where speech is present or no speech is present. This information can be provided from external sources or can be estimated beside base calculations within an algorithm. Misjudgments of the speech/non-speech parts usually introduce errors and lead to incorrect feature calculation results, which makes the speech/non-speech detection itself a very important feature for speech analysis and for further speech processing methods.

K.1.1 Problem Definition

As present in Grimm and Kroschel [2007], speech/non-speech detection is an unsolved problem in speech processing affecting numerous applications. Grimm and Kroschel defines the VAD problem as *detecting the presence of speech in a (noisy) signal*. Assuming additive noise, VAD algorithms have to distinguish between two cases:

$$H_1 : \mathbf{x} = \mathbf{n} + \mathbf{s} \tag{K.1}$$

$$H_0 : \mathbf{x} = \mathbf{n} \tag{K.2}$$

where the hypotheses H_1 and H_0 indicate if speech is present or not, n is the noise, s the speech signal itself, and x the noisy speech signal assuming only additive noise is present.
Usually voiced speech sounds and noise can be distinguished according to their different signal characteristics. Unfortunately, this is not always the case for unvoiced speech sounds, which may have time-domain and frequency-domain characteristics similar to white noise. As a result of this similarity in many cases only voiced/unvoiced sound detection is applied (Yang et al. [2010a]).
Most VAD algorithms fail in noisy environments. Recent researches address this weakness by analyzing noise-robust features.

K.1.2 Applications

Speech coding, *speech enhancement* and *speech recognition* are listed in Grimm and Kroschel [2007, page 2] as the most important VAD applications in speech processing. A brief description of these applications is presented in this section.

Speech Coding

The compression of digital speech data is called *speech coding*, VAD is required for high compression efficiency. The concept of *dual-mode speech coding* splits the signal in speech and non-speech parts. Non-speech parts are compressed with another codec which usually allows significant decrease of the required bit-rate. Standardized examples using dual-mode speech coding are the

ITU-T Recommendation G.729 Annex B (Benyassine et al. [1997]) and the *ETSI AMR speech coder* (ETSI 301 708 [1999]). Figure K.1 shows the block diagram of a dual-mode speech codec.

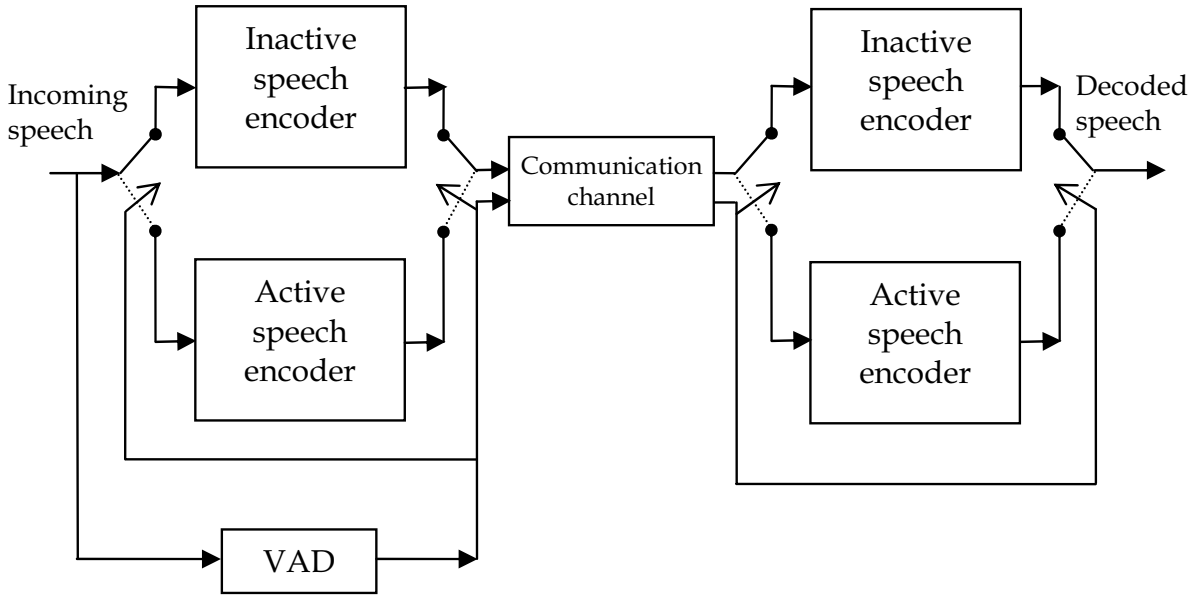


Figure K.1: Dual mode speech coding for DTX, presented in Benyassine et al. [1997].

Speech Enhancement

Speech enhancement mainly focuses on suppression of background noise, where it is assumed that only additive noise is present. In Boll [1979] the author describes an algorithm based on spectral subtraction: During non-speech periods, the noise-spectrum is estimated, which is subtracted from the spectrum of the noisy speech signal frames:

$$|S(f)| = |X(f)| - |N(f)| \tag{K.3}$$

where $S(f)$ is the current frame of the de-noised spectrum of the speech signal, $X(f)$ is the noisy signal spectrum and $N(f)$ the estimated noise-spectrum.

Speech Recognition

The automatic conversion of spoken words to a written lexical representation is called *speech recognition*. The performance of the conversation is highly influenced by the quality of the speech signals and how good the training environment matches the test conditions (if a learning technique is applied). Pre-processing steps which drop non-speech frames or apply speech enhancement methods lead to an essential increase of the recognizer's performance. Figure K.2 shows an example of the feature extraction part of a typical speech recognition system, using spectral noise reduction and non-speech frame-dropping.

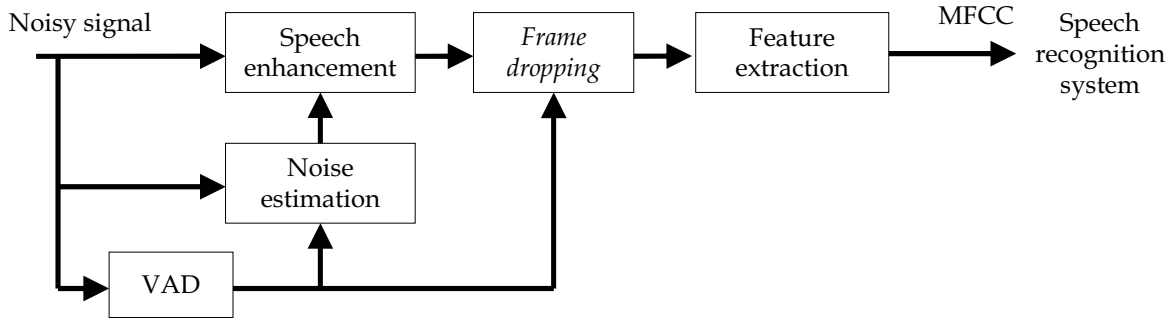


Figure K.2: Typical feature extraction system for speech recognition in noisy environments, proposed in Grimm and Kroschel [2007, page 5].

K.2 Algorithm Overview

This section presents a overview of a few VAD methods. The block diagram in figure K.3 describes the three main steps of common VAD algorithms:

1. Pre-processor (feature extraction)
2. Basic extractor (decision module)
3. Post-processor (decision smoothing)

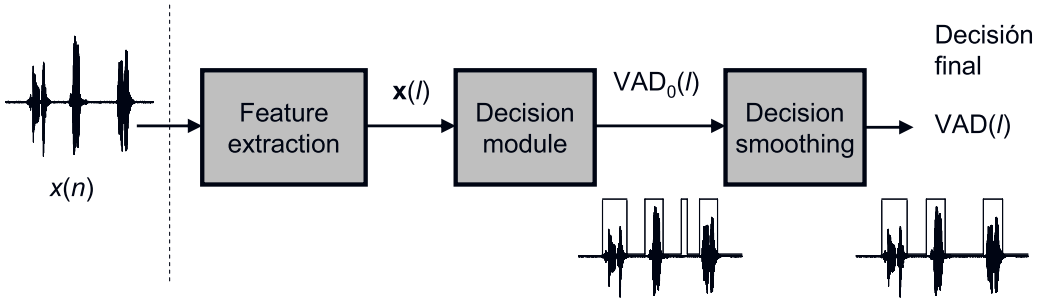


Figure K.3: Common VAD structure as present in Grimm and Kroschel [2007, page 8].

According to their application requirements, many different approaches of VAD algorithms exists. Three classes according to their type of application can be distinguished:

1. Basic VAD algorithms, mainly designed for simple and fast implementation.
2. VAD algorithms which are designed for highly noisy environments.
3. Compound VAD algorithms which are reliable for a wide range of environmental conditions.

K.2.1 Basic VAD Algorithms

The algorithms described in this section extract different signal features from short-time signal frames. Applying a rectangular window $w(m)$, the short-time signal $x_n(m)$ for frame n can be

obtained in a pre-processing step:

$$x_n[m] = x[m] \cdot w_{rectangular}[n-m] \qquad 0 \leq m \leq N-1 \tag{K.4}$$

$$w_{rectangular}[m] = \begin{cases} 1 & 0 < m < (N-1) \\ 0 & \text{otherwise} \end{cases} \tag{K.5}$$

Usually overlapped frames with fixed frame-size and fixed step size are used, so for each frame $n = \{0, 1T, 2T, ...\}$ the step size can be defined as $1 \leq T < N,\ T = \text{const}$. Within a single frame the signal parameters are assumed to be constant.
Phase effects, injected by rectangular windowing, can be avoided using other window functions. Common speech window functions (beside the *rectangular window* $w_{rectangular}$) are the *Hamming window* $w_{Hamming}$ and the *Hanning window* $w_{Hanning}$:

$$w_{Hamming}[n] = \begin{cases} 0.54 + 0.64 \cdot cos\left\{\left[\frac{2n}{N-1} - 1\right]\pi\right\} & 0 \leq n \leq (N-1) \\ 0 & \text{otherwise} \end{cases} \tag{K.6}$$

$$w_{Hanning}[n] = \begin{cases} 0.5 \cdot \left[1 - cos\left(\frac{2\pi n}{N-1}\right)\right] & 0 \leq n \leq (N-1) \\ 0 & \text{otherwise} \end{cases} \tag{K.7}$$

Advantages and disadvantages of the different window functions are discussed in Boersma [1993].

VAD Based on Short-Time AMs

The voiced/unvoiced decision is based on the short-term energy of each frame:

$$E_n = \sum_{m=0}^{N-1} x_n^2[m] \tag{K.8}$$

Problems with huge window length and high amplitudes are observed, so the Average Magnitude Functions (AMs) are frequently used to characterize the short-term energy (Yang et al. [2010a]):

$$AM_n = \sum_{m=0}^{N-1} |x_n[m]| \tag{K.9}$$

Voiced/Unvoiced decision H_n in Jawale [2009] is based on comparison of the energy related value (E_n or AM_n) against a single, constant threshold:

$$H_n = \begin{cases} 1 & AM_n > \eta \\ 0 & \text{otherwise} \end{cases} \tag{K.10}$$

$$min(AM_{voiced}) \leq \eta \leq max(AM_{noise}) \tag{K.11}$$

where $H_n = 1$ indicates the presence and $H_n = 0$ the absence of voice for frame n. Overlapping areas between voiced speech band energy and noise speech band energy make a satisfying choice of the constant threshold value η very difficult.

VAD Based on Short-Time Average ZCR

The number of times a signal crosses the horizontal axis (zero level) is denoted as *zero -crossing rate (ZCR)* . The average short-term ZCR (Z_n) can be calculated by counting the amount of

times the sign of two consecutive signals changes:

$$Z_n = \frac{1}{2}\sum_{m=0}^{N-1} |sgn[x_n[m]] - sgn[x_n[m-1]]| \tag{K.12}$$

$$sgn[x] = \begin{cases} 1 & x \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{K.13}$$

In cases of voiced speech sounds signal energy is concentrated in a low frequency region. The energies of unvoiced and noisy frames tend to distribute over the whole frequency range or even to be concentrated in higher frequency regions. So low ZCRs indicate the presence of voiced speech sounds, whereby unvoiced parts may have higher ZCR values.
In Jawale [2009] a double threshold method is applied, which is an algorithm based on markers where different thresholds for start points and stop points of voiced speech portions and backward - searching are used to determine the speech bounds. Usually additional hangover schemes can increase the reliability of this technique.

VAD Based on Short-Time AMDF

Simple VAD algorithms are very often based on calculation of short-term auto-correlation function (ACF) values (Yang et al. [2010a]). To decrease the computational effort, the average magnitude difference function (AMDF) can be calculated instead:

$$AMDF_n[k] = \sum_{m=0}^{N-1-k} |x_n[m] - x_n[m+k]| \tag{K.14}$$

$$x_n[m] = x[n+m]w_{rectangular}[n] \tag{K.15}$$

where $x_n[m]$ is the speech signal and $w_{rectangular}[n]$ a rectangular window function according to equation K.5. For strictly periodic signals with period N_p, the difference $d_m = x[m] - x[m+k]$ becomes zero for $k = 0, \pm N_p, \pm 2N_p ....$ In case of voiced speech signals a significant minimum can be observed at the corresponding points. To remove the influence of the lag k, the window length for $w[n]$ can be adapted and the AMDF rewritten as (Yang et al. [2010a]):

$$AMDF'_n[k] = \sum_{m=0}^{N-1} |x'_n[m] - x'_n[m+k]| \tag{K.16}$$

$$x'_n[m] = x[n+m]w'[n] \tag{K.17}$$

$$w'[n] = \begin{cases} 1 & 0 \leq n \leq N+k-1 \\ 0 & \text{otherwise} \end{cases} \tag{K.18}$$

K.2.2 VADs for Noisy Environments

Claiming reliable results even within noisy environments require algorithms based on more robust mathematical methods. Typical approaches of algorithms designed for such conditions are presented here.

VAD Based on LRT

A two-hypothesis test using the Bayes Classifier for selection of the class with the largest posterior probability $P(H_i|\mathbf{x})$ is presented in Sohn et al. [1999]:

$$P(H_1|\mathbf{x}) \underset{H_0}{\overset{H_1}{\gtrless}} P(H_0|\mathbf{x}) \tag{K.19}$$

$$\frac{P(H_1|\mathbf{x})}{P(H_0|\mathbf{x})} \underset{H_0}{\overset{H_1}{\gtrless}} \frac{P(H_0)P(H_1|\mathbf{x})}{P(H_1)P(H_0|\mathbf{x})} = \eta \tag{K.20}$$

where $\mathbf{x}$ is the feature vector of the noisy speech signal and η defines a decision threshold. Different calculation principles of $P(H_i|\mathbf{x})$ defines a huge set of VAD algorithms.

VAD Based on LTSD

Ramirez et al. assume in Ramirez et al. [2004a] that important information for speech/non-speech discrimination can be obtained from the signal spectrum magnitude. The long-term spectral envelope (LTSE) estimates a spectral envelope from long-term speech windows:

$$LTSE_N(k,l) = max\left\{X(k,l+j)\right\}_{j=-N}^{j=+N} \tag{K.21}$$

where $x(n)$ is the noisy signal and $X(k,l)$ is the k-th amplitude spectrum bin for frame l. N defines the order of the LTSE. Calculating the so called long-term spectral divergence (LTSD), a decision rule can be formulated related to the spectral divergence of speech and noise:

$$LTSD_N(l) = 10 \cdot log_{10}\left(\frac{1}{NFFT}\sum_{k=0}^{NFFT-1}\frac{LTSE^2(k,l)}{N^2(k)}\right) \underset{H_1}{\overset{H_0}{\lessgtr}} \eta \tag{K.22}$$

where $N(k)$ is the average noise spectrum magnitude for band k.

In figure K.4 the effect of the long-term window length N is explained, using an 8 kHz input signal, decomposed into overlapping frames with a time-shift of 10 ms. The optimum window length according to the classification error is shown in figure K.5. Finally in the paper a sub-optimal LTSD window length of N=12 at NFFT=256 is used.

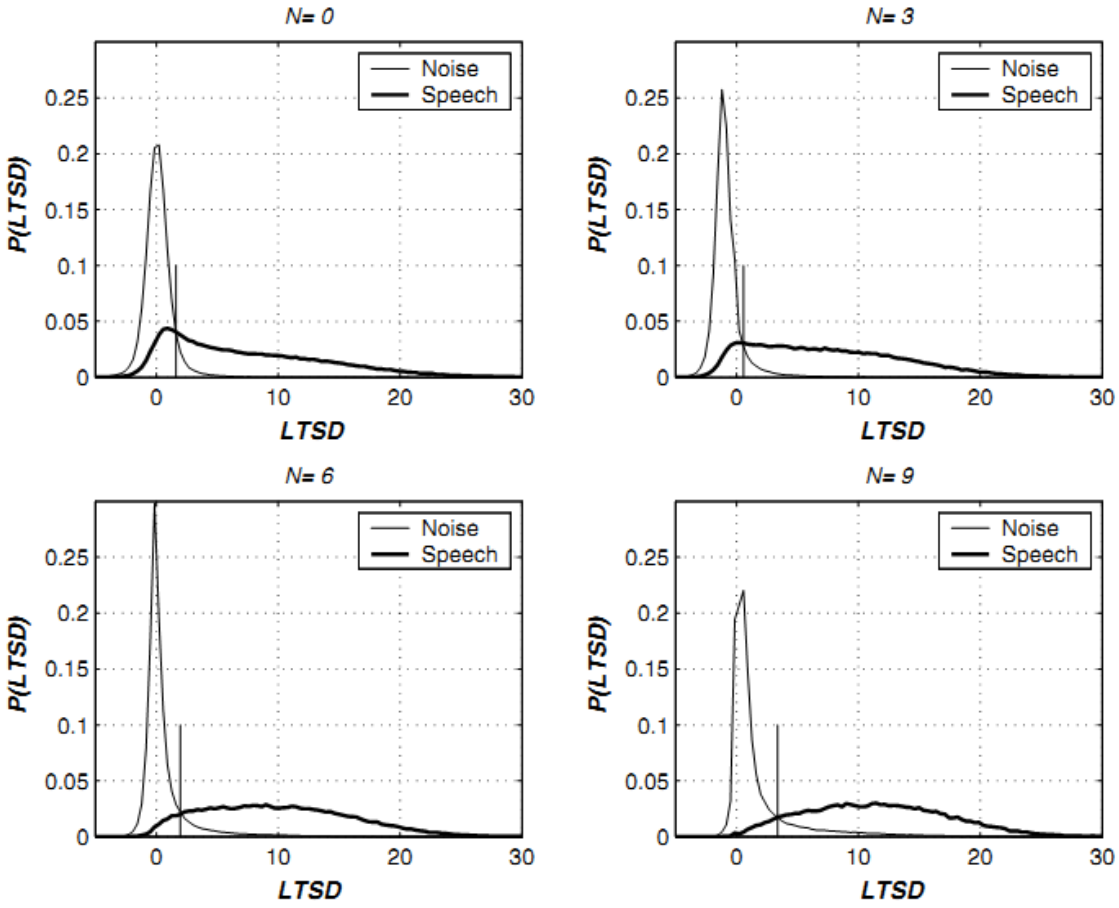


Figure K.4: The influence of the LTSD window length, visualized using speech / non-speech LTSD probability distribution (Ramirez et al. [2004a]).

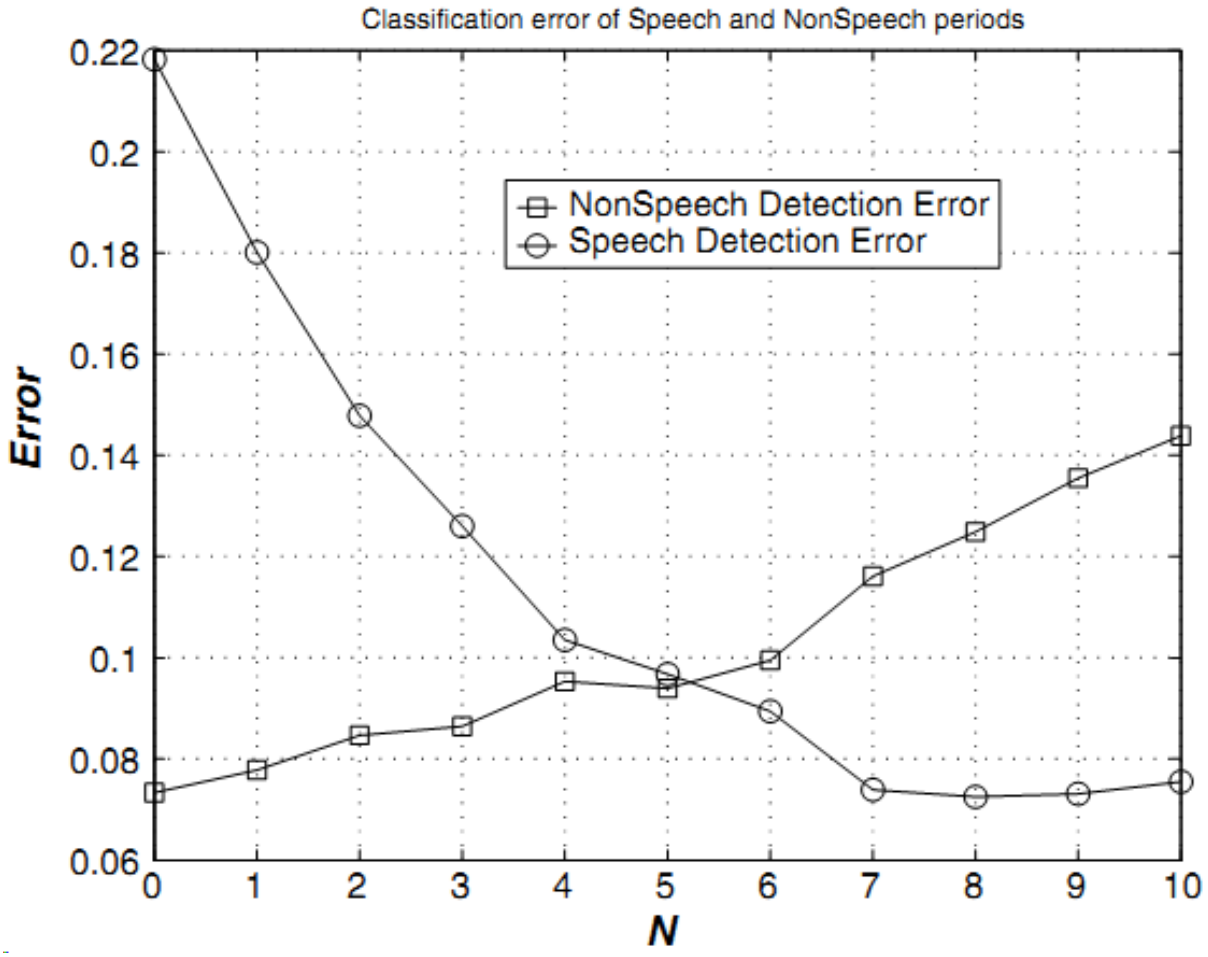


Figure K.5: The optimum of the LTSD window length can be estimated using speech and non-speech detection error, plotted as a function of the LTSD window length N (Ramirez et al. [2004a]).

VAD Based on Bilateral Filtering

Moving computational complexity from the basic extractor to the preprocessing step can be very helpful as many VAD techniques are only useful if specific signal-conditions are fulfilled. Instead of dealing with difficult conditions in the main feature extraction step, the signal may be translated to representations which are easier to analyze. Regarding to the mean square error criterion, a linear filter may be applied if the noise is stationary, additive and Gaussian (optimum filter). Realistic environments differ from this assumption quite often, which makes the usage of non-linear filtering very useful.

Here an edge-preserving low-pass filter is applied, which leads to a de-noised, non-blurred signal. In image-processing, this idea follows the concept of filtering the spacial domain and the tonal domain with a single filter, as proposed in Lee [1983] (sigma-filter) and in Tomasi and Manduchi [1998] (bilateral filter). Many investigations are made to make the non-separable 2-D filter more efficient (Che and Miao [2010]; Guarnieri et al. [2006]; Gunturk [2011]; Igarashi et al. [2010]; K. et al. [2011]; Mattoccia et al. [2011]; Pham and van Vliet [2005]; Porikli [2008]; Villalpando et al. [2011]; Yang et al. [2009, 2010b]; Yu et al. [2010]).
In Matsumoto and Hashimoto [2009] the filter is analyzed for sound-denoising.

For image processing, the bilateral filter result $y(\mathbf{r})$ at pixel $\mathbf{r}$ is defined as (Tomasi and Manduchi [1998], Matsumoto and Hashimoto [2009]):

$$y(\mathbf{r}) = \frac{1}{k(\mathbf{r})} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x(\mathbf{u}) \cdot c(\mathbf{r}, \mathbf{u}) \cdot s(x(\mathbf{r}), x(\mathbf{u}))\ du \tag{K.23}$$

$$k(\mathbf{r}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\mathbf{r}, \mathbf{u}) \cdot s(x(\mathbf{r}), x(\mathbf{u}))\ du \tag{K.24}$$

where $k(\mathbf{r})$ is the weight factor for the filter kernel, $\mathbf{r}$ and $\mathbf{u}$ are coordinates within the image, $x(\mathbf{r})$ and $x(\mathbf{u})$ are the photometric values at the related coordinates, $c(\mathbf{r}, \mathbf{u})$ is the geometric closeness between point $\mathbf{r}$ and point $\mathbf{u}$, and $s(x(\mathbf{r}), x(\mathbf{u}))$ is the photometric closeness between the pixel values (photometric values) $x(\mathbf{r})$ and $x(\mathbf{u})$.

Usually a Gaussian function is used for the geometric closeness $c(\mathbf{r}, \mathbf{u})$ as well as for the photometric closeness $s(x(\mathbf{r}), x(\mathbf{u}))$:

$$c(\mathbf{r}, \mathbf{u}) = e^{-\frac{1}{2}\left(\frac{d(\mathbf{r},\mathbf{u})}{\sigma_d}\right)^2} \tag{K.25}$$

$$s(\mathbf{a}, \mathbf{b}) = e^{-\frac{1}{2}\left(\frac{\delta(\mathbf{a},\mathbf{b})}{\sigma_\delta}\right)^2} \tag{K.26}$$

$$d(\mathbf{r}, \mathbf{u}) = d(\mathbf{r} - \mathbf{u}) = ||\mathbf{r} - \mathbf{u}|| \tag{K.27}$$

$$\delta(\mathbf{a}, \mathbf{b}) = \delta(\mathbf{a} - \mathbf{b}) = ||\mathbf{a} - \mathbf{b}|| \tag{K.28}$$

where $d(\mathbf{r}, \mathbf{u})$ is the Euclidean distance between $\mathbf{r}$ and $\mathbf{u}$, and $\delta(\mathbf{a}, \mathbf{b})$ a photometric distance measurement. Beside the size of the filter kernel, the behaviour of the filter is controlled by the geometric spread parameter σ_d and the photometric spread parameter σ_δ.
For visualization purposes a filtering-process of a gray-value image is shown in figure K.6.

For audio processing, the equations are reduced to 1-D space: The 2-D coordinates $\mathbf{r}$ and $\mathbf{u}$ are replaced by time-indices of the speech signal, $x(t)$ is the speech signal itself, and instead of the photometric distance measurement $\delta(\mathbf{a}, \mathbf{b})$ simply the difference between two signal amplitude

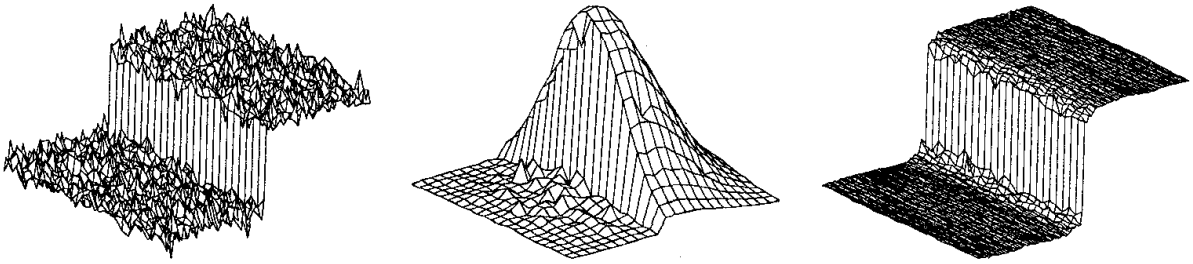


Figure K.6: Bilateral filtering, using Gaussian kernel functions (Tomasi and Manduchi [1998]): The left plot shows a part of a gray-value image represented in a Cartesian coordinate system where the pixel intensity is assigned to the height coordinate. The middle plot is a 23x23 pixel filter kernel, calculated for a center-pixel of the left plot. The right plot shows the filtered image.

values can be taken:

$$\tilde{\delta}(x(t_0), x(t_1)) = |x(t_0) - x(t_1)| \tag{K.29}$$

If sufficient speech-enhancement can be assumed, very simple VAD techniques may be applied, based on e.g. zero-crossing rate, signal energy or sub-band energy, or even on spectral features, statistical features, and many others. Another very common method is to analyze the speech characteristics itself and to use human speech specific features, as discussed in Espi et al. [2010]. Bilateral filtering, combined with spectral subtraction de-noising is assumed to improve most algorithm results, especially in noisy conditions.
Even if bilateral filtering is provided for all proposed algorithms of this paper by the evaluation-framework, only one VAD algorithm is designed and evaluated especially based on this speech enhancement method (see section K.3.7).

K.3 Implementation

Some VAD algorithms are implemented and evaluated with Matlab. The Matlab-scripts are well-structured and well-documented, so only the main steps of the algorithms provided here.

The algorithms are chosen according to aspects like real-time capabilities, level of awareness, implementation availability, complexity level, reliability of the results in noisy environments and if their usage is suitable.

K.3.1 LTSD Algorithm

The algorithm proposed here is related to the method presented in Ramirez et al. [2004b]. It is basically designed to detect speech within noisy environments. The basic steps of the algorithm's data stream are outlined in the pipeline diagram in figure K.7.

The basic components of the VAD are the N-order LTSE and the related LTSD, calculated according to equations K.21 and K.22. The noise spectrum $N(k)$ is calculated for each frame l as follows:

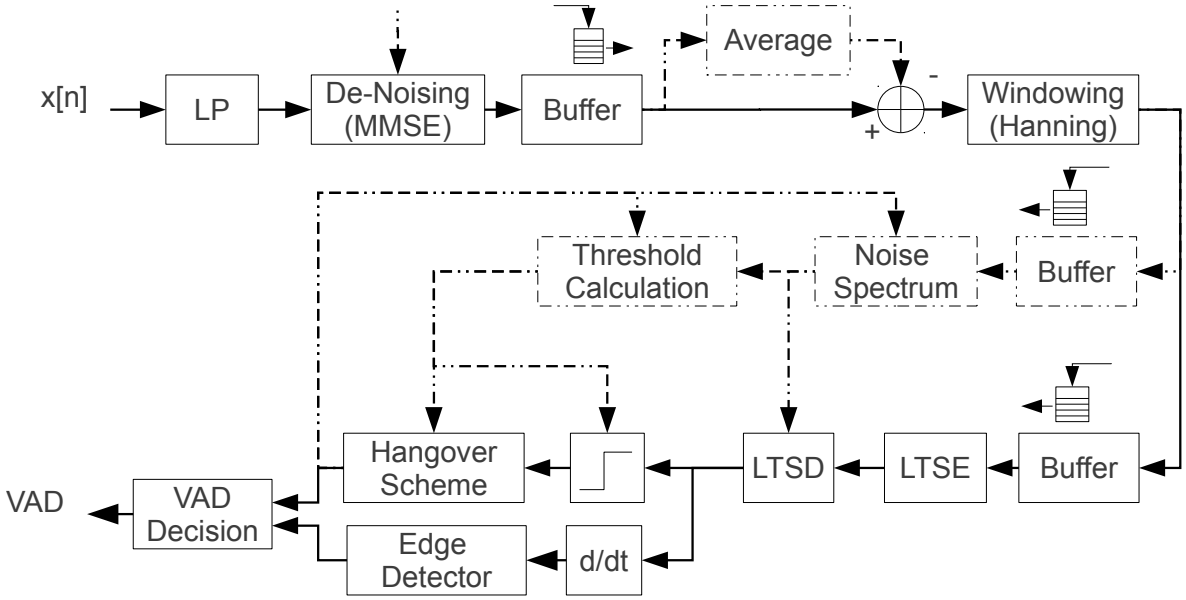


Figure K.7: Basic steps of the VAD related to Ramirez et al. [2004b] and Babu et al. [2009]. Pre-processing includes: Low pass filtering (LP), speech enhancement (De-Noising-MMSE using Brookers [2011] implementation of speech enhancement, proposed in Ephraim and Malah [1984]), splitting the signal into frames and applying a Hanning window function (Buffer, Windowing). Basic feature calculation is based on LTSE and LTSD. Post-processing involves: thresholding (⌈), applying a simple hangover scheme (Hangover Scheme), detecting changes in the LTSD course (d/dt, Edge Detector) and the final speech/non-speech decision (VAD-Decision).

$$N(k,l) = \begin{cases} \alpha \cdot N(k,l-1) + (1-\alpha) \cdot N_K(k) & \text{if speech pause is detected} \\ N(k,l-1) & \text{otherwise} \end{cases} \tag{K.30}$$

$$N_K(k) = \frac{1}{2K+1} \sum_{j=-K}^{K} X(k,l+j) \tag{K.31}$$

where $N_K(k)$ is the average spectrum magnitude at frame k, calculated over a K-frame neighborhood.

Post-Processing

The $LTSD_N(l)$ calculated in equation K.22 is compared to an adaptive threshold value γ:

$$\gamma = \begin{cases} \gamma_0 & E \leq E_0 \\ \frac{\gamma_0-\gamma_1}{E_0-E_1} E + \gamma_0 - \frac{\gamma_0-\gamma_1}{1-E_1/E_0} & E_0 < E < E_1 \\ \gamma_1 & E \geq E_1 \end{cases} \tag{K.32}$$

where E_0 and E_1 are the background noise energy values for clean and noisy conditions. The optimum thresholds for clean and noisy conditions (γ_0, γ_1) can be obtained experimentally.

To protect word-endings from misclassification as non-speech, a hangover scheme corrects the speech/non-speech classification. In low-noise environments ($LTSD_N(l) > LTSD_0$), the hangover-scheme is disabled to improve the decision accuracy. Additional improvements can be gained by analyzing the time derivation of the LTSD signal.

K.3.2 PARADE Algorithm

An algorithm especially designed for conditions where non-stationary noise is present is proposed in Ishizuka et al. [2010]. The periodic and aperiodic parts are extracted and related to each other. The speech/non-speech decision is based on the observation that the power of voiced speech is centered around their harmonic components. The main steps of the algorithm's data stream are outlined in the pipline diagram in figure K.8.

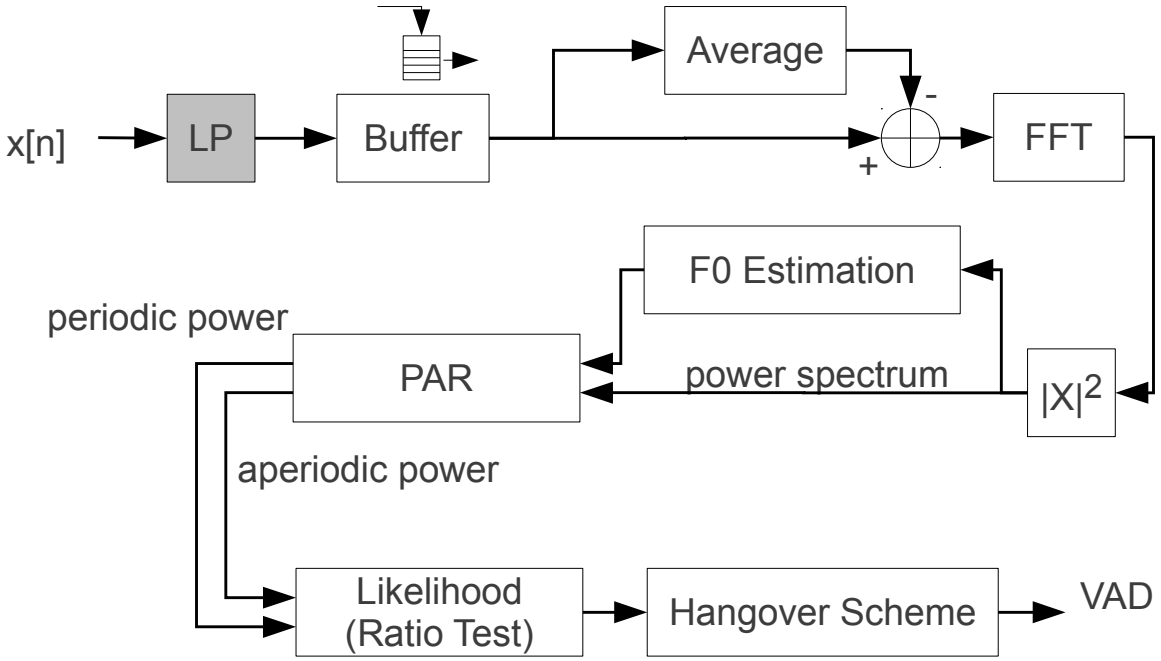


Figure K.8: The main steps of the algorithm proposed in Ishizuka et al. [2010] are: Signal conditioning and framing (LP, Buffer, Average), calculating the periodic and aperiodic power components by usage of FFT (FFT, $|x|^2$, F0 Estimation, PAR) and statistical analysis (Likelihood Ratio Test). Post-processing is formed by a hangover Scheme to deal with unvoiced speech components which are located near to voiced speech components (Hangover Scheme).

Basic Extractor

The basic feature extraction is related to the algorithm proposed in Boersma [1993] (see section L.3.2). The ACF used in the harmonic noise ratio (HNR) algorithm cancels the influence of additive white Gaussian noise. However, significant errors still may be introduced by colored noise. This weakness is addressed here by analyzing the power of the harmonic and non-harmonic components instead of the ACF signal itself.
It is assumed that the speech signal $x(n)$ can be split into periodic parts $x_p(n)$ and aperiodic parts $x_a(n)$:

$$x(n) = x_p(n) + x_a(n) \tag{K.33}$$

A further assumption includes the additivity of the power of periodic and aperiodic parts in the frequency domain:

$$|X(i,k)|^2 = |X_p(i,k)|^2 + |X_a(i,k)|^2 \tag{K.34}$$

where $|X(i,k)|^2$, $|X_p(i,k)|^2$ and $|X_a(i,k)|^2$ represent the short-term power of the signal $x(n)$, the periodic parts $x_p(n)$ and the aperiodic parts $x_a(n)$ at frame i for the frequency bin k, respectively. Substituting $p(i)$ for the short-term power, equation K.34 can be rewritten as

$$p(i) = p_p(i) + p_a(i) \tag{K.35}$$

$$p(i) = \frac{1}{NFFT} \sum_{k=0}^{NFFT-1} |X(i,k)|^2 \tag{K.36}$$

where $p_p(i)$ and $p_a(i)$ are the short-term power of the periodic and aperiodic part of the signal, respectively.

The calculation of the periodic energy $p_p(i)$ is calculated with an approximation similar to a comb filter (Smith [2007]):

$$p_p(i) = \sum_{m=1}^{v(i)} p_{p,m}(i) \tag{K.37}$$

$$p_{p,m}(i) = \eta \cdot |X_p(i, [m \cdot f_0(i)]|^2 \tag{K.38}$$

$$\tag{K.39}$$

where $f_0(i)$ is the F0 of the periodic component at frame i, $v(i)$ is the number of it's harmonics, $p_{p,m}(i)$ is defined as the power of a sinusoid corresponding to the m^{th} harmonic component and η is a constant form factor (see Ishizuka et al. [2010]).
The average power of aperiodic speech components are assumed to have uniform distribution over the whole frequency spectrum, which is also fulfilled at dominant frequencies:

$$p_a(i) = \frac{1}{NFFT} \sum_{k=0}^{NFFT-1} |X_a(i,k)|^2 = \frac{1}{v(i)} \sum_{m=1}^{v(i)} |X_a(i, [m \cdot f_0(i)])|^2 \tag{K.40}$$

Putting all together the speech energy can be written as

$$p(i) = \eta \sum_{m=1}^{v(i)} |X(i, [m \cdot f_0(i)])|^2 + (1 - \eta v(i)) p_a(i). \tag{K.41}$$

$$\hat{p}_a(i) = \frac{p(i) - \eta \sum_{m=1}^{v(i)} |X(i, [m \cdot f_0(i)])|^2}{1 - \eta v(i)} \tag{K.42}$$

$$\hat{p}_p(i) = p(i) - \hat{p}_a(i) = \eta \frac{\sum_{m=1}^{v(i)} |X(i, [m \cdot f_0(i)])|^2 - v(i) p(i)}{1 - \eta v(i)} \tag{K.43}$$

where $\hat{p}_a(i)$ and $\hat{p}_p(i)$ are the final estimations for the power of periodic and aperiodic components, respectively. The short-term power $p(i)$ is calculated according to equation K.36 whereby F0 can be calculated by any PDA as well as by using the numerator of equation K.43:

$$\hat{f}_0(i) = \arg\max_{f_0(i)} \left(\sum_{m=1}^{v(i)} |X(i, [m \cdot f_0(i)])|^2 - v(i) p(i) \right) \tag{K.44}$$

where $\hat{f}_0(i)$ is an estimation for F0 at frame i (Nakatani and Irino [2004], Nakatani et al. [2008]).

Post-Processing

The assumptions and approximations used in section K.3.2 may lead to systematical errors. To take these errors into account, the speech/non-speech decision is based on the likelihood derived from the error distributions of the periodic and aperiodic components.
A *joint probability density function* based on the segmental signal power $p(i)$, the estimated periodic power component $\hat{p}_p(i)$ and the estimated aperiodic power component $\hat{p}_a(i)$ defines the likelihood $\mathfrak{L}$ of a speech segment:

$$\mathfrak{L}(H_i) = \mathbf{p}(p(i), \hat{p}_a(i), \hat{p}_p(i)|H_i) = \mathbf{p}(\hat{p}_a(i), \hat{p}_p(i)|H_i, p(i)) \cdot \mathbf{p}(p(i)|H_i) \tag{K.45}$$

where H_i may be 0 or 1. Remark that $\mathbf{p}(p(i)|H_i)$ is a constant term and can be dropped for further calculations.
Non-speech segments (marked with $H_i = 0$) are assumed to have no periodic components ($p(i) = p_a(i)$). The error of this assumption can be calculated with

$$\epsilon_a(i) = p(i) - \hat{p}_a(i) = \hat{p}_p(i) \tag{K.46}$$

For speech segments (marked with $H_i = 1$) the calculation error can be approximated with

$$\epsilon_p(i) = p(i) - \hat{p}_p(i) = \hat{p}_a(i) \tag{K.47}$$

Assuming that the error distributions of $\epsilon_a(i)$ and $\epsilon_p(i)$ follows Gaussian distributions (mean and standard derivation are 0, $\alpha \cdot p_a(i) \approx \alpha \cdot \hat{p}_a(i),\ \alpha > 0$ and $\beta \cdot p_p(i) \approx \beta \cdot \hat{p}_p(i),\ \beta > 0$), the likelihood for a speech period can be formulated as:

$$\mathbf{p}(\hat{p}_a(i), \hat{p}_p(i)|H_i, p(i)) \approx \mathbf{p}(\epsilon_p(i)|H_i = 1, p(i) = p_p(i)) \tag{K.48}$$

$$\approx \frac{1}{\sqrt{2\pi}\beta\hat{p}_p(i)} e^{-\frac{1}{2\beta^2}\left(\frac{\hat{p}_a(i)}{\hat{p}_p(i)}\right)^2} \tag{K.49}$$

which is defined for all cases of H_i assuming no background noise (see Ishizuka et al. [2010]).
The VAD decision is based on the likelihood ratio $\Lambda(i)$

$$\Lambda(i) = \frac{\mathfrak{L}(H_i = 1)}{\mathfrak{L}(H_i = 0)} = \frac{\alpha}{\beta} \cdot \frac{1}{\mu(i)} e^{\frac{1}{2\alpha^2}\mu(i)^2 - \frac{1}{2\beta^2}\cdot\frac{1}{\mu(i)^2}} \tag{K.50}$$

$$\mu(i) = \frac{\hat{p}_p(i)}{\hat{p}_a(i)} \tag{K.51}$$

Using $\alpha = 1$ and $\beta = 1$, equation K.50 can be simplified to

$$\Lambda(i) = \frac{1}{2\mu(i)} e^{\frac{1}{2}(\mu(i)^2 - \frac{1}{\mu(i)^2})} \tag{K.52}$$

where $\Lambda(i) \geq Threshold$ describes frames containing speech components and $\Lambda(i) < Threshold$ describes non-speech frames.

K.3.3 HOS Algorithm

To avoid initial background noise guesses which are usually required for automatic threshold determination, the algorithm proposed in Shuyin et al. [2009] uses high order statistic (HOS). To separate speech from non-speech, the background noise is assumed to have almost Gaussian distribution, for which third and higher order statistics tend to vanish. In contrast, speech is

regarded as non-Gaussian distributed signal, which makes HOS a robust feature to distinguish between speech and non-speech components in both - time and frequency domain.
The algorithm is generalized to work in time-varying noisy environments where the noise may be described as white additive noise or as colored additive noise. The main steps of the algorithm's data stream are outlined in the pipline diagram in figure K.9.

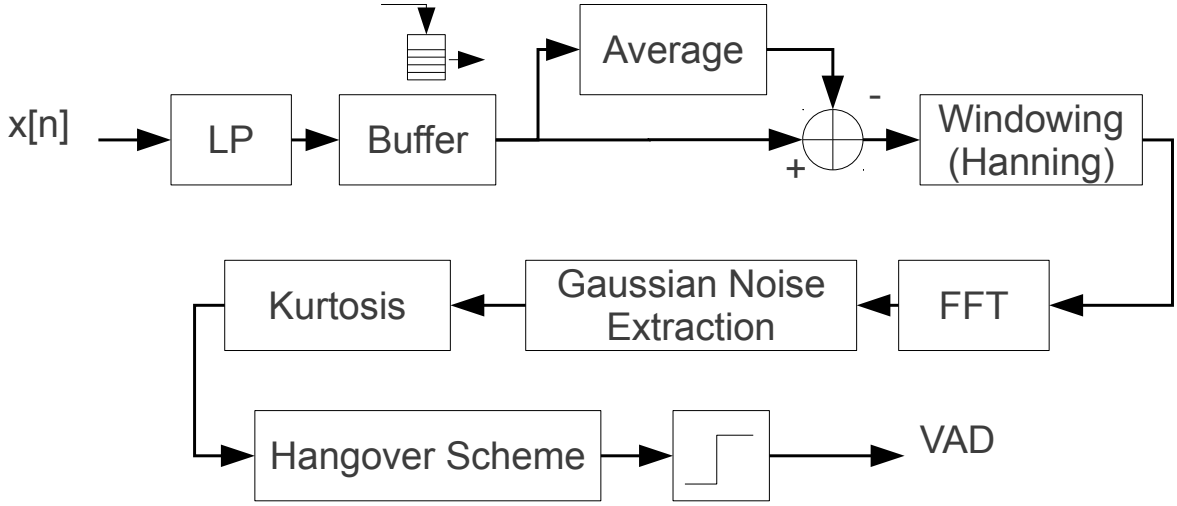


Figure K.9: The basic steps of the VAD algorithm proposed in Shuyin et al. [2009] include: Filtering, local average subtraction and windowing (LP,Buffer,Average, Windowing), STFT (FFT), extraction of zero-mean Gaussian distributed information from colored environmental noise (Gaussian Noise Extraction), calculation of the main decision feature (Kurtosis) and the speech/non-speech decision based on a simple hangover scheme and a fixed threshold value (Hangover Scheme, ⌠).

Basic Extractor

In Shuyin et al. [2009] speech signal is modeled by a zero-mean random variable. Furthermore only additive Gaussian noise with zero mean is assumed in the first step. The author mathematically proved that the noise signal is zero for all "*high-order cumulants*" (also known as *eigenfunctions*, see Li et al. [2005]), which means only speech information remains in cases where speech is superimposed with zero mean Gaussian noise. In the paper the 4^{th} order cumulant is approximated by the Spectral Domain Kurtosis $kurt(x)$:

$$kurt(x) = c_4^{(x)}(0,0,0) = E[x^4(n)] - 3E^2[x^2(n)] \tag{K.53}$$

where $E[X]$ is the expectation of the zero-mean random variable X. Applying a simple threshold method, the Spectral Domain Kurtosis can be used as voiced/unvoiced decision rule. The final calculation of the Spectral Domain Kurtosis may be formalized using short -term fourier transform (STFT) for the current frame l:

$$kurt_l(i) = \frac{1}{NFFT}\sum_{k=0}^{NFFT-1} X_l^4(k) - 3\cdot\left[\frac{1}{NFFT}\sum_{k=0}^{NFFT-1} X_l^2(k)\right]^2 \tag{K.54}$$

where $X_l(k)$ is the frequency domain representation of the speech signal at frame l.

Mathematical principles which are required for the derivation of the proposed formulas are given in Shuyin et al. [2009].

Generalizing to Colored Noise

Gaussian information can be extracted from colored noise, which is required to remain reliable speech/non-speech decision results when colored background noise is present:

$$X'_l(k) = X_l(k) - \hat{\mu}_l(k), \qquad k = 0, 1...NFFT - 1 \tag{K.55}$$

$$\hat{\mu}_l(k) = \frac{1}{M} \sum_{i=l-M+1}^{l} X_i(k) \tag{K.56}$$

where $\hat{\mu}_l(k)$ is the average spectral amplitude and $X'_l(k)$ the spectral amplitude sequence with zero-mean Gaussian distribution, l the frame index and M the average computation size. The Spectral Domain Kurtosis $kurt(x)$ is calculated according to equation K.54.

Post-Processing

To avoid misclassification of unvoiced sounds, a hangover scheme is required. In Shuyin et al. [2009] a very simple method similar to average filtering (Smith [2007]) is used:

$$kurt(i) = \alpha \cdot kurt(i-1) + (1-\alpha) \cdot kurt(i) \tag{K.57}$$

where α is known as forgetting factor and $kurt(i)$ is the feature used for speech/non-speech decision.

K.3.4 G729B VAD Algorithm

In Benyassine et al. [1997] a VAD algorithm is proposed which is based on multi-feature pattern matching, noise-estimation and a special hangover (smoothing) scheme. Basically four features are extracted for each frame (instantaneous pattern): linear prediction (LP) spectrum, full-band energy, low-band energy, and zero-crossing rate. Beside this features a long-term energy is calculated for a window length of about 1 secound to avoid a deadlock which may be injected by the algorithm's feedback loop. Additionally a global threshold for the short-term energy is used to avoid result glitches for very low signal amplitudes.

The primary VAD decision is based on pattern matching: The four features described above are bundled to a instantaneous signal pattern, which is calculated for each frame. Within speech absence, the background pattern is updated (which require a simple VAD in advance). The current signal pattern then is compared in the hyperspace. Time-varying influences are considered by the smoothing/hangover step, which forms the final VAD decision.

The main steps of the algorithm's data stream are outlined in the pipline diagram in figure K.10. As the algorithm is provided as source-code and the implementation is fairly simple, no further description is given here.

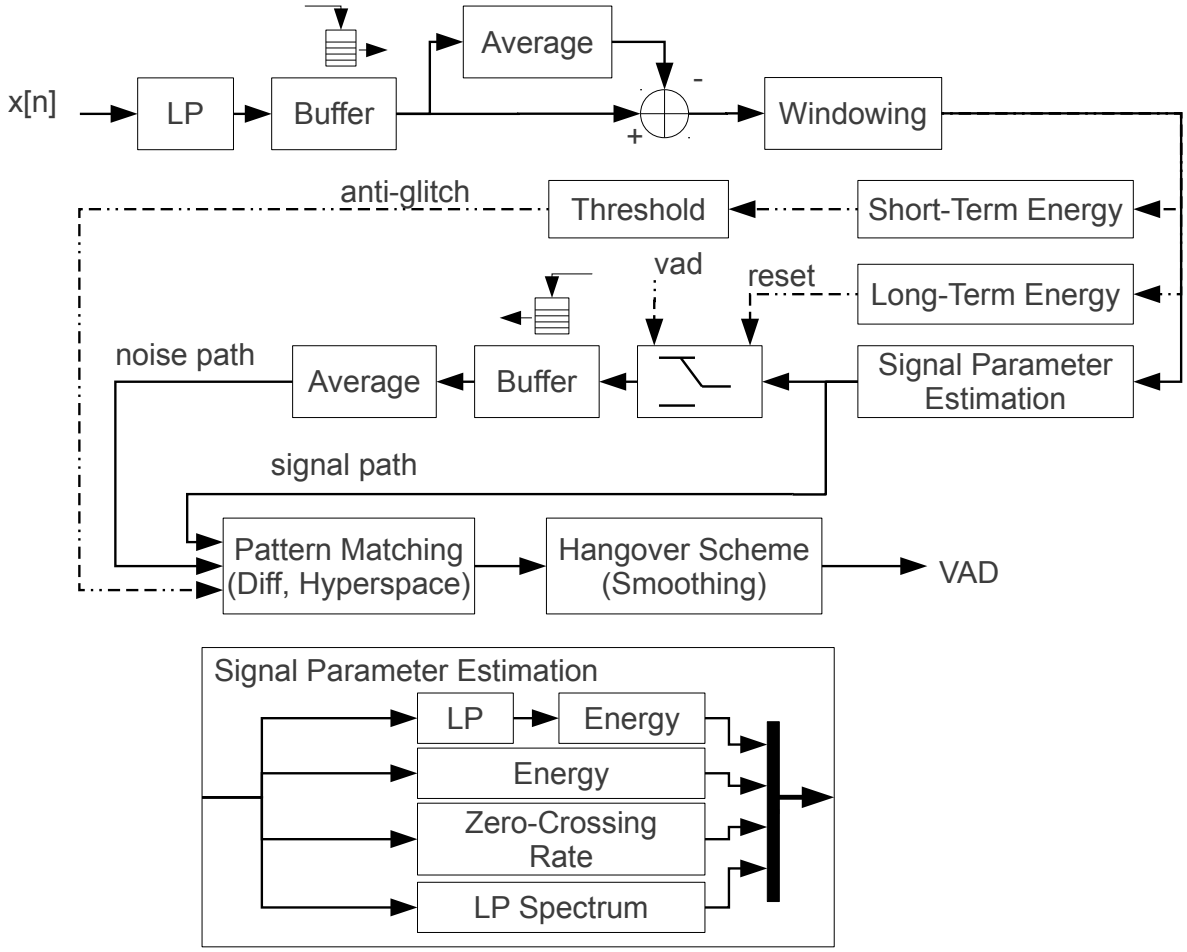


Figure K.10: The basic steps of the VAD algorithm proposed in Benyassine et al. [1997] include: Filtering, local average subtraction and windowing (LP,Buffer,Average, Windowing), extraction of the four main speech features used for pattern matching (Zero-Crossing Rate, Full-Band Energy, Low-Band Energy, LP Spectrum Parameters), background noise - pattern estimation, and the pattern matching itself. Short-term energy estimation is used to avoid result glitches, long-term energy estimation is used to avoid a possible deadlock due adverse conditions (hidden feedback loop). The final result is formed by a 4-state post-processing step.

K.3.5 LRT based Algorithm

An implementation according to K.2.2 is provided in Brookers [2011] , which mainly implements the algorithm proposed in Sohn et al. [1999]. The VAD algorithm estimates the background noise in the spectral domain, calculates the so-called *a priori signal to noise ratio (SNR)* and *a posteriori SNR* values and defines a related statistical likelihood ratio test (LRT). An sufficient hangover scheme forms the final VAD decision. The main steps of the algorithm's data stream are outlined in the pipeline diagram in figure K.11.
The algorithm is provided in source-code, so no further description is given here. Details about the algorithm and implementation are provided in Brookers [2011] and Sohn et al. [1999].

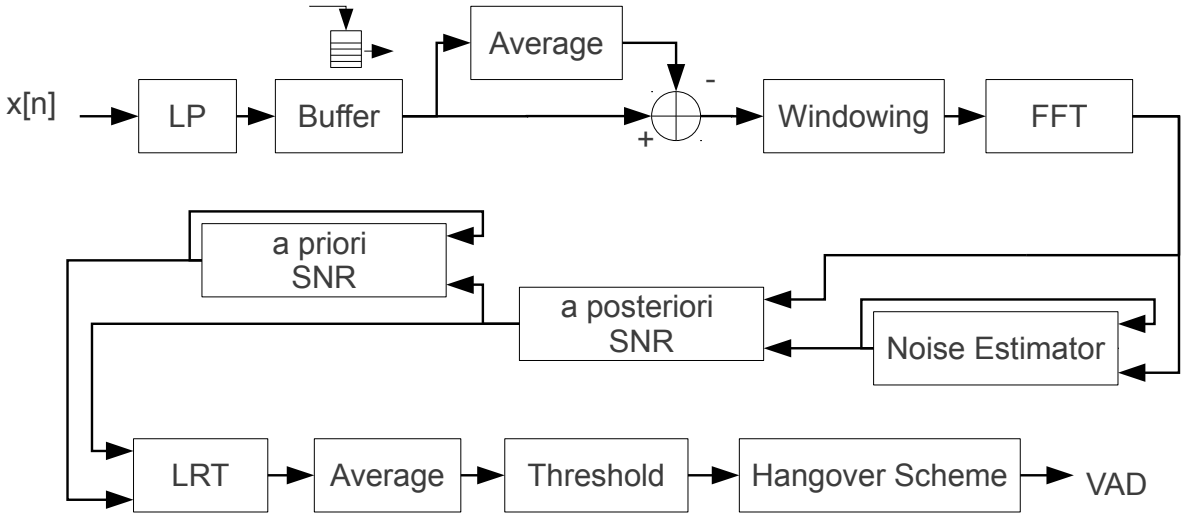


Figure K.11: The basic steps of the VAD algorithm proposed in Sohn et al. [1999] include: Filtering, local average subtraction and windowing (LP,Buffer,Average, Windowing). Furthermore noise estimation is done to estimate the a priori SNR and the a posteriori SNR . The initial VAD decision is based on the smoothed result of the LRT, the final decision rule is formed by a hangover scheme.

K.3.6 Entropy based Algorithm

In Lei et al. [2009] Wiener filtering (Lim and Oppenheim [1979], Benesty and Huang [2008], Mendel [1999]), a mel-filterbank and spectral entropy is used to improve the robustness of the VAD within noisy environments. In the current implementation Wiener filtering is replaced by the speech enhancement method of Brookers [2011] which generally implements the de-noising method proposed in Ephraim and Malah [1984]. The main steps of the algorithm's data stream are outlined in the pipeline diagram in figure K.12.

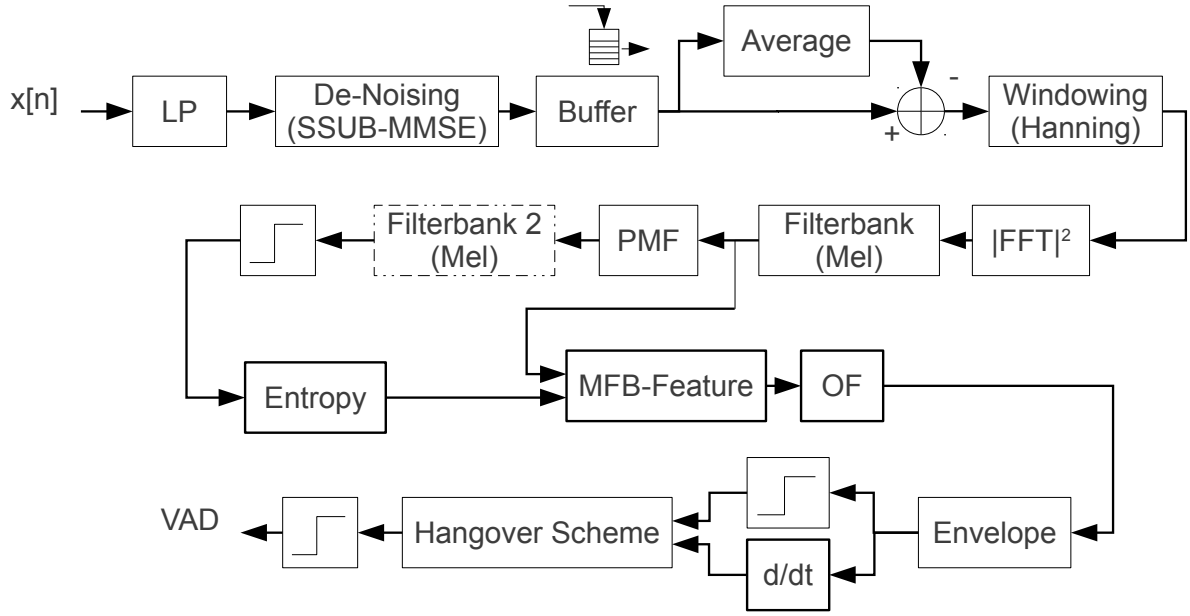


Figure K.12: The basic steps of the implemented algorithm related to Lei et al. [2009] and Nordholm et al. [2005] are: Signal conditioning (LP, De-Noising, Buffer, Average), applying a Hanning window function (Windowing), calculating the energy in the frequency space ($|FFT|^2$), calculating the PMF (Filterbank, PMF, ⌈); the main VAD feature is based on calculating the MFB frequency-space entropy (Entropy) and the MFB-Feature; post-processing includes ordered statistic filtering (median filter, OF), feature envelope approximation (Envelope), a hangover scheme and a VAD decision based on thresholding.

Basic Extractor

In theory, spectral entropy is not very much influenced by the total energy if the signal distribution of the signal doesn't change much. Therefore different speech enhancement methods may be used without standing to loose significant information for the speech/non-speech decision. Using short-time fast fourier transform (FFT), a mel-scaled triangular filter-bank is applied in the frequency domain for each frame:

$$p'_i = \frac{Mel(f_i)}{\sum_{k=1}^{M} Mel(f_k)} \tag{K.58}$$

$$p_i = \begin{cases} 0 & p'_i > 0.9 \\ p'_i & \text{otherwise} \end{cases} \tag{K.59}$$

where M is the number of filters used, $Mel(f_i)$ is the filter-bank coefficient for component f_i and p'_i the corresponding probability density function. Applying equation K.59 leads to a more accurate speech/non-speech decision since noise-parts at specific frequency-bands are eliminated (for further information, see Lei et al. [2009]). A slightly different method is used in Nordholm et al. [2005], where a PMF is calculated by normalizing the spectrum in each sub-band with the total spectral energy:

$$p'_i = \frac{X_i}{\sum_{i=0}^{NFFT-1} X_i} \qquad i = 0, 1..NFFT-1 \tag{K.60}$$

where X_i is the energy of the i^{th} frequency bin and p'_i the corresponding PMF.

The mel filter-bank spectral entropy H at each frame is defined as

$$H = -\sum_{i=1}^{M} p_i \; log \; p_i \tag{K.61}$$

Energy based information is added by frame-wise calculation of the spectral energy

$$E_s = \sum_{i=0}^{NFFT-1} |\hat{X}_i|^2 \tag{K.62}$$

Finally both, the spectral entropy H and the spectral energy E_s are combined to the so-called MFB feature (Lei et al. [2009]):

$$MFB - feature = \sqrt{H.E_s} \tag{K.63}$$

Post-Processing

A long term envelope of the MFB feature is calculated which may be used immediately for a simple VAD-threshold method. Nevertheless, additional improvements (calculation and selection of sub-band based MFB-feature results, adding information of the feature's envelope - time derivation,...) are added before thresholding is applied to increase the reliability of the algorithm.
Finally, a hangover scheme would be required to form the final VAD decision.

K.3.7 Variance based Algorithm

A new method proposed in this project leads to a very robust VAD algorithm. If noise free sound is assumed and noise-like speech sounds can be eliminated, the data spread can be used as voiced/non-voiced decision criterion. Such spread can be measured using for example variance, standard derivation or absolute derivation.

Assuming that most syllables consist of a voiced core sound, usually embedded in non-voiced speech-sounds, and smooth transitions between speech sounds are common within connected speech, one may define a VAD algorithm using massive low-pass filtering, the spread-feature, and a kind of "time region markers".

A main part of this algorithm is the integration of the variance over time, which makes the algorithm very robust against outliers and bursts. A drawback is the delayed VAD-decision. When using this method in real-time scenarios, the maximum latency introduced by the integration may be set to a constant value by forcing the VAD-decision if a specified time-period has expired.

Low-pass filtering within the pre-processing step is another essential part of this algorithm. To prevent this step of blurring the audio signal, which can make the information unextractable, an edge preventing filter is used instead of a common low-pass filter. Additionally the speech signals may be locally amplified using an adaptive gain factors.

The main steps of such an algorithm's data stream are outlined in the pipeline diagram in figure K.13.

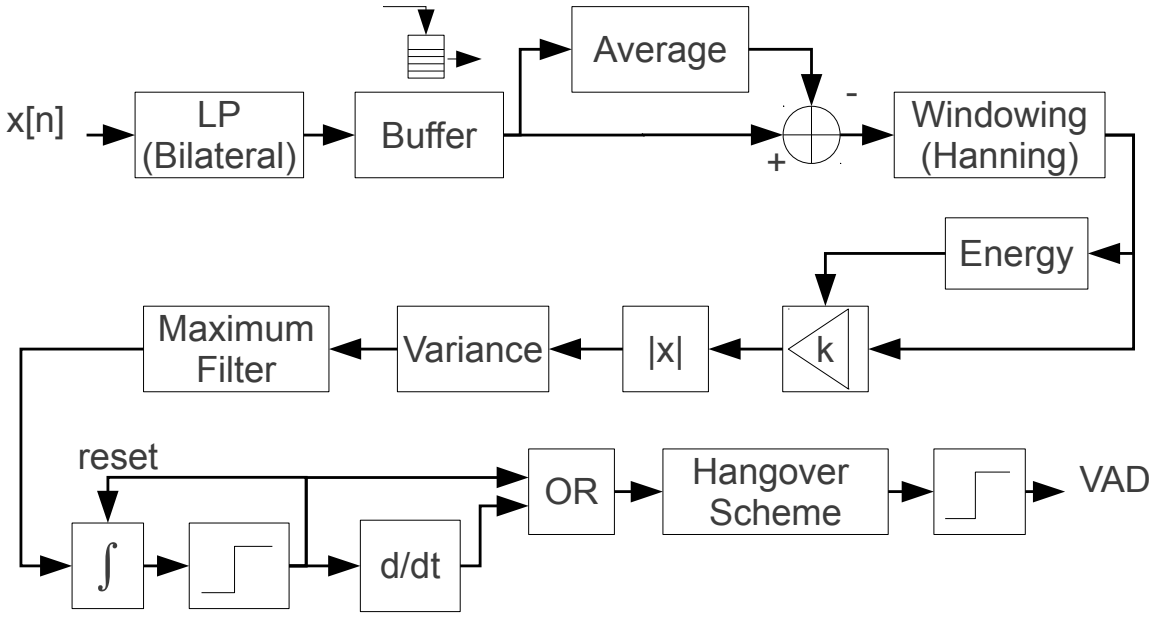


Figure K.13: The basic steps of the variance-based algorithm are: Signal conditioning using a bilateral filter as edge preventing low-pass filter (LP, De-Noising, Buffer, Average), applying a Hanning window function (Windowing), calculating the energy for adaptive (automatic) gain control; the main VAD feature is based on variance calculation, ordered-statistic filtering (maximum filter), and integration over time. Simple post-processing steps form the final VAD decision.

Pre-Processing

Beside the common pre-processing steps like windowing, local average subtraction or applying a Hanning window function, the low-pass filtering is a very important step for the current algorithm. Edge preventing low-pass filtering avoids signal-blur. Here a 1-D bilateral filter (as proposed in Matsumoto and Hashimoto [2009]) is used:

$$y[n] = k[n] \cdot \sum_{m=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} x[n] \cdot c[n,m] \cdot s(x[n], x[m]) \tag{K.64}$$

$$k[n] = \sum_{m=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} c[n,m] \cdot s(x[n], x[m]) \tag{K.65}$$

$$\tag{K.66}$$

whereby $y[n]$ is the filtered signal sample at position n, $k[n]$ the weight for sample position n, $c[n,m]$ the temporal closeness of sample positions n and m, and $s(x[n], x[m])$ the amplitude

value closeness. The closeness calculations are based on Gaussian functions:

$$c[n,m] = e^{\frac{1}{2}\left(\frac{d[n,m]}{\sigma_d}\right)^2} \tag{K.67}$$

$$s(a,b) = e^{\frac{1}{2}\left(\frac{\delta(a,b)}{\sigma_\delta}\right)^2} \tag{K.68}$$

$$d[n,m] = d[n-m] = |n-m| \tag{K.69}$$

$$\delta(x[n], x[m]) = |x[n] - x[m]| \tag{K.70}$$

with the temporal spread factor σ_d, the amplitude spread factor σ_δ, and the distance functions $d[n,m]$ and $\delta(x[n], x[m])$. In a straight forward implementation, the temporal closeness function can be precomputed and provided as lookup table, whereby the amplitude value closeness needs to be calculated for each sample separately. A common speedup technique is the quantization of the amplitude distance values. The amplitude value closeness may then be estimated by interpolation of the two closest precomputed values.

Basic Extractor

The basic extractor calculates the variance of the absolute sample values for a single frame. The variance of frame i is given by:

$$\sigma_y^2[i] = \frac{1}{M-1}\sum_{n=0}^{M-1}\left(|y[n]| - \frac{\sum_{m=0}^{M-1}|y[m]|}{M}\right)^2 \tag{K.71}$$

where $M > 1$ is the frame length, n the sample offset within frame i, and $y[n]$ the filtered speech sample at sample index $i*SZ+n$, using a constant step size SZ. For the current implementation a frame length of 512 samples (64 ms) and a step size of 33 samples (4 ms) is used.
The first VAD decision is based on comparison of the variance envelope against an constant threshold value. The envelope is computed over 51 samples (6.4 ms), using a maximum filter.

Post-Processing

A time marker is set where the basic extractor classifies a frame as voiced part. Until the next non-voiced part, the variance is summed up and compared against a certain threshold value. If the sum exceeds the threshold, the whole region is classified as speech. A new region starts if the basic extractor classifies a frame as non-speech.

K.4 Evaluation

The TIMIT database is used for the evaluation of the VAD algorithms. The VAD reference signal is extracted from the phonetic transcription files of the database. These audio samples are referred to as “single sentence tracks”. To balance the overall amount of speech parts and non-speech parts, concatenated speech samples, filled with silence of arbitrary length between the speech parts, are analyzed. These audio samples are referred to as “multi sentence tracks”.

The VAD estimation results are automatically shifted in time to get the minimum error between the reference signal and the estimated signal. This method is used for the evaluation-processes in de Cheveigné and Kawahara [2002] to ease the comparison between different algorithms, using

different databases where the total latency in the audio-recording-process is unknown.
Main advantage of this auto-alignment is the independence of the results from recording-delays, signal-shifts and result-alignment within the analyzing windows. On the other hand it has a negative impact on the results because incorrect alignments are possible. Nevertheless this errors are expected to occur seldom which would make their statistical influence negligible.

Even if some algorithms are used "as-is" (black-boxes), an uniform pre-processing and post-processing step is intended to make the results more comparable. Therefore the basic extraction algorithms are embedded in an uniform framework, which is shown in figure K.14.

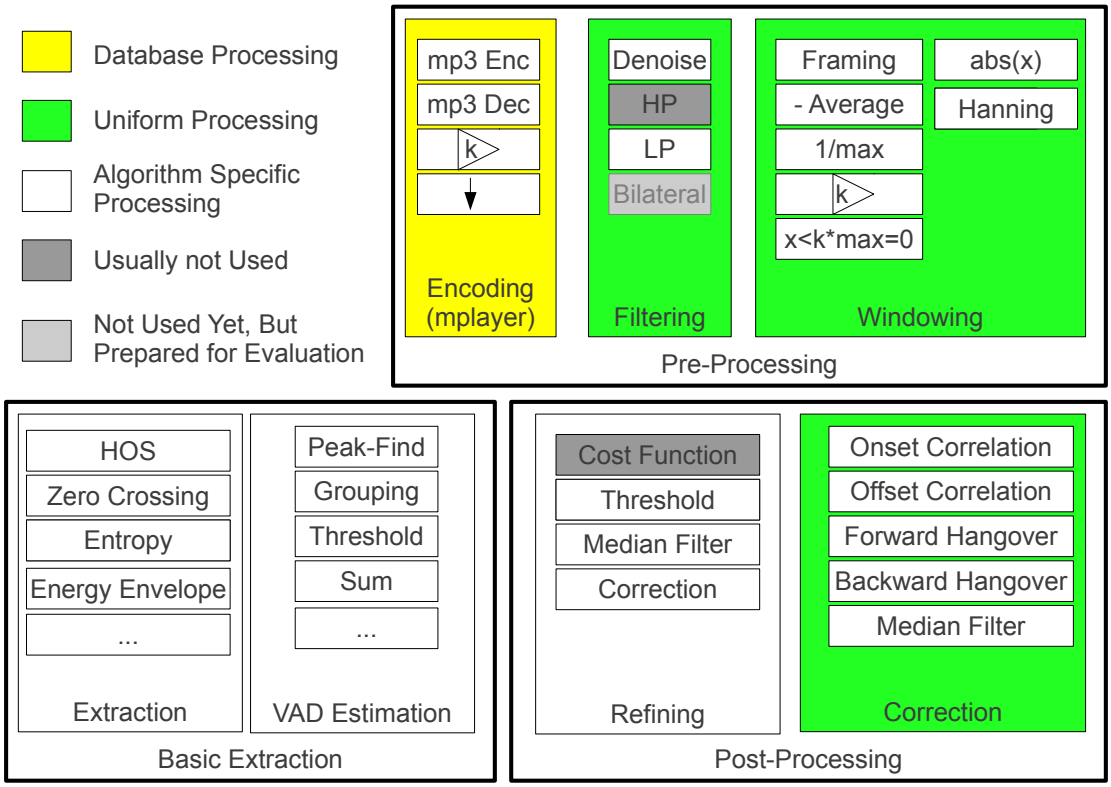


Figure K.14: Uniform VAD evaluation framework: Mp3-coding, mp3-decoding, amplifying and filtering/down-sampling is applied on the speech samples of the databases using Mplayer (MPlayer [2011]). The main filtering and windowing process is similar in all VAD algorithms. Special filter-operations like edge-preventing low-pass-filtering (e.g. bilateral filter), spectral whitening or window-related amplifying are implemented mainly for prospective purposes. The basic feature extraction step and parts of the post-processing step are formed by the individual algorithms. Finally uniform post-processing-steps like VAD onset and offset correction (using correlation around the boundaries), common hangover schemes and ordinary median filtering refine the VAD feature results.

Table K.1 includes the algorithms' name mapping between the specific names used in this report and the short labels used in the plots of the evaluational framework. Furthermore it is indicated if an algorithm was available in source-code or is re-implemented according to the related paper and the modifications stated in the corresponding section of this report.

Report Label	Short Label (Used for Plots)	Source-Code Freely Available
G729B VAD Algorithm, section K.3.4	ECSE412B ECSE	yes, not re-implemented
HOS Algorithm, section K.3.3	HOS	no, implemented as Matlab Script
Entropy based Algorithm, section K.3.6	ENTROPY	no, implemented as Matlab Script
LTSD Algorithm, section K.3.1	LTSD	no, implemented as Matlab Script
periodic -component to aperiodic -component ratio based activity detection (PARADE) Algorithm, section K.3.2	PARADE	no, implemented as Matlab Script
Variance based Algorithm, section K.3.7	BILAT	no, implemented as Matlab Script
LRT based Algorithm, section K.3.5	SOHN	yes, not re-implemented

Table K.1: Algorithms' name mapping between the specific names used in this report and the short labels used in the plots of the evaluational framework. Furthermore it is indicated if an algorithm was available in source-code or is re-implemented according to the related paper and the modifications stated in the corresponding section of this report.

K.4.1 Evaluation Databases

From the TIMIT database 1393 audio samples, spoken by men, and 758 audio samples, spoken by women are used. As all these audio samples contain silence within the spoken sequence, they are used twice: once as single sentence track where one file is exactly one speech sample and once as a fraction of the multi-sentence tracks.

The later one results in 464 (concatenated) audio samples, containing male voice, and 189 (concatenated) audio samples, containing female voice, whereby for each audio sample three base-audio tracks of the TIMIT database and five silence-parts (three in-between the audio samples, one at the beginning and one at the end) with arbitrary length between 0.05 seconds and 0.5 seconds are used.

The single sentence audio tracks have an average-length of 4.5 seconds, enclosed in leading and trailing silence with an average-length of about 0.1 seconds. the multi sentence audio tracks have an average-length of 11.7 seconds, enclosed in leading and trailing silence with an average length of about 0.4 seconds. Leading and trailing silence is assumed during the signal is lower than -60 dB related to the global maximum of the audio-file.

K.4.2 Evaluation Conditions

To test the algorithms in more realistic scenarios, background noise is added to the audio-tracks, leading to following evaluational conditions:

- Evaluation of male and female tracks without background noise

- Evaluation of male and female tracks with white noise, 10 dB and 20 dB SNR
- Evaluation of male and female tracks with babble noise, 10 dB and 20 dB SNR
- With and without mp3-coding and decoding (to simulate a common transmission- or storage-scenario)

Background noise is taken from the noise database proposed in Johnson and Shami [1993]. Using perceptive comparison (listening to audio tracks with noise which are generated in the laboratory and audio tracks from political speeches, see J.2) a realistic environment for speech analysis may be declared using babble noise and speech signals with 20 dB SNR. A worst-case scenario is declared using 10 dB SNR, where the algorithms still should lead to reliable results.

Mp3 coding and decoding is done using Mplayer (MPlayer [2011]). The mp3 coding step involves filtering and down-sampling to 16 kHz sampling frequency, the mp3 decoding step results in wav-formated audio files (raw) with 8 kHz sampling frequency. The two-step filtering and down-sampling is required because most audio-samples are recorded with 48 kHz sampling frequency, and the direct conversion from raw (wav) audio format, using 48 kHz sampling frequency, down to mp3 coded audio files with 8 kHz sampling frequency leads to many acoustic disturbances if Mplayer is used.

The final audio-tracks for the algorithm-evaluation are constructed by following equations:

$$x[q] = s[q] + \tilde{n}[q] \tag{K.72}$$

$$\tilde{n}[q] = k.\frac{max\left(|s[q]|\right))}{max\left(|n[q]|\right)} \cdot n[q] \tag{K.73}$$

$$k = 10^{-\frac{SNR_{dB,A}}{20}} \tag{K.74}$$

where $x[q]$ is the final audio track (test sample), $\tilde{n}$ is the additive noise, $s[q]$ a spoken sound record from the database (or a concatenated audio track), SNR_{dB} the SNR according the the predefined conditions and $n[q]$ the noise signal.

In equation K.74 the SNR is defined by maximum amplitude values calculated over a single file:

$$SNR_{dB,A} = 20 \cdot log\left(\frac{max(|s[q]|)}{max(|n[q]|)}\right) \tag{K.75}$$

where $SNR_{dB,A}$ is the SNR defined by maximum amplitude values, and can be interpreted as the "worst case scenario" for a given sound sample, as background noise is assumed to be almost uniform distributed and not to have significant bursts. One may use signal energy values for calculating the SNR, which would lead to a slightly different definition:

$$SNR_{dB,E} = 10 \cdot log\left(\frac{E_s}{E_n}\right) \qquad \text{(not used)} \tag{K.76}$$

where $SNR_{dB,E}$ is the SNR defined by energies, E_s is the total signal energy and E_n is the total noise energy.

To provide a better comparability to related papers, the evaluational framework also implements the so-called segmental signal to noise ratio (SSNR) method for the calculation of the background noise gain factor k, used in equation K.73. The SSNR is calculated according to Vary [2006,

page 226]:

$$SSNR_{dB,E}[l] = \begin{cases} 10 \cdot log\left(\frac{E_s[l]}{E_n[l]}\right) & E_n[l] \neq 0 \\ \infty & \text{otherwise} \end{cases} \tag{K.77}$$

where $E_s[l]$ is the speech energy, $E_n[l]$ is the noise energy and $SSNR_{dB,E}[l]$ is the SNR value at frame l. For the current implementation a frame size of 15 ms and a step size of half the frame size is used. For the final noise-gain factor, the average of $SSNR_{dB,E}[l]$ over a single file is used. The process how the different environment conditions are simulated in laboratory is shown in figure K.15.

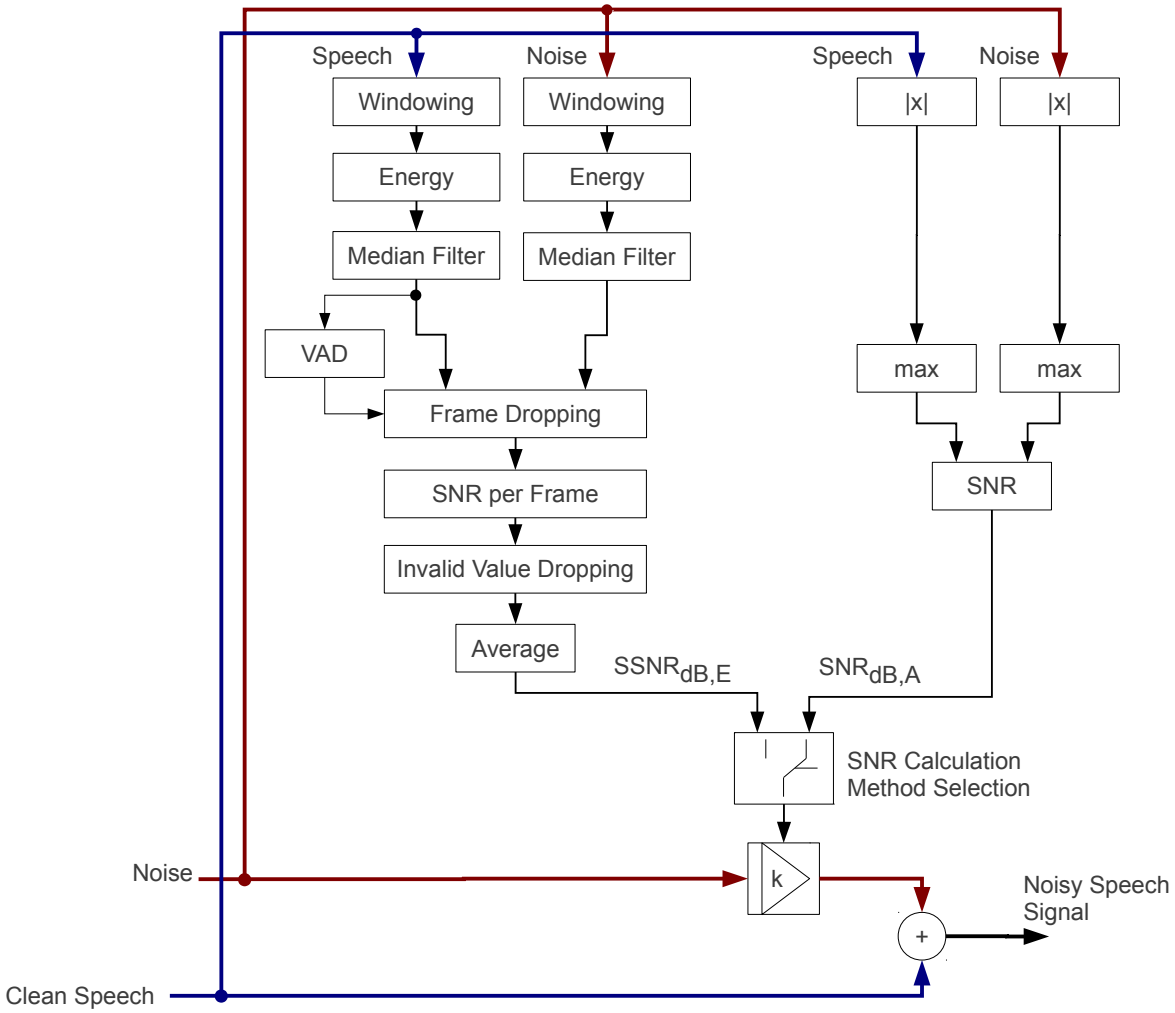


Figure K.15: For simulating different environmental conditions, the framework provides 2 different methods for calculating the SNR of the speech and the background noise: The first method uses the amplitude maxima of the related speech and noise files to calculate the SNR, which results in $SNR_{dB,A}$. As single signal bursts have strong influence on the SNR value, a second method is provided, which is based on the SSNR calculation, where a window length of 15 ms and a step size of half the window length is used.

Furthermore all algorithms are tested with and without denoising. For denoising, the spectral subtraction method from the VOICEBOX is used, which mainly implements the algorithm proposed in Martin [2001].

K.4.3 Results

Statistical comparison of the total error results within the different conditions are shown in figure K.16. The total error $E[alg_i]$ of a specific algorithm alg_i is calculated as follows:

$$E_j[alg_i] = \frac{EN1_j[alg_i] + EN0_j[alg_i]}{N} \tag{K.78}$$

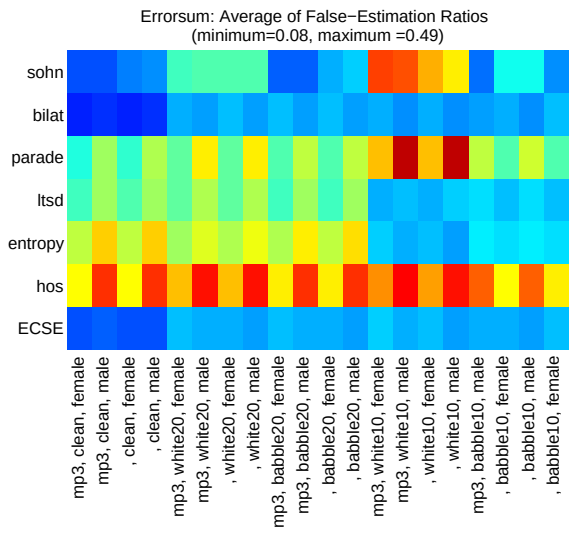
$$E[alg_i] = \frac{\sum_{j=0}^{M-1} E_j[alg_i]}{M} \tag{K.79}$$

where M is the total count of test-files, N is the number of samples within file j, $EN1_j[alg_i]$ is the count of silence-samples which the specific algorithm alg_i incorrectly detect as voiced parts, $EN0_j[alg_i]$ is the count of the voiced parts which the algorithm alg_i incorrectly detect as silence, and $E_j[alg_i]$ is the error ratio of file j. Not all algorithms are trimmed to result in the lowest possible total error. Adjusting the parameters so that the count of incorrect detected silence parts is low would increase the count of incorrect detected voice parts, and vice versa. This behavior is visualized in figure K.5 and also can be seen in table K.2, which is a small part of the VAD evaluation result.

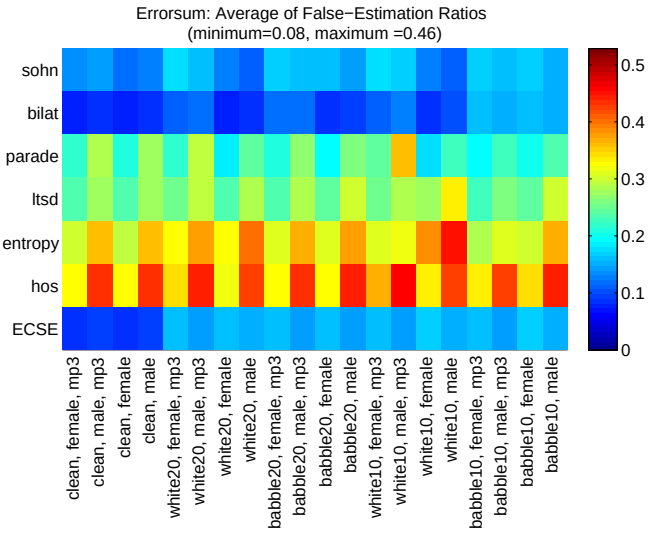
Algorithms:	**ECSE412B**		**HOS**		**ENTROPY**		**LTSD**		**PARADE**		**BILAT**		**SOHN**	
Gender: [m]ale, [f]emale	**m**	**f**	**m**	**f**	**m**	**f**	**m**	**f**	**m**	**f**	**m**	**f**	**m**	**f**
Database, Conditions	***TIMIT Database, Single Sentence Samples, Clean Speech Conditions, no Denoising***													
Total Error	0.11	0.10	0.44	0.33	0.35	0.29	0.28	0.23	0.28	0.21	0.09	0.08	0.11	0.10
Incorrect Detected Silence	0.00	0.00	0.44	0.32	0.35	0.29	0.27	0.22	0.27	0.20	**0.02**	**0.02**	0.02	0.02
Incorrect Detected Voice	0.11	0.10	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	**0.07**	**0.06**	0.09	0.09
Database, Conditions	***TIMIT Database, Multi Sentence Samples, Clean Speech Conditions, no Denoising***													
Total Error	0.10	0.10	0.39	0.30	0.32	0.27	0.25	0.21	0.24	0.19	0.07	0.07	0.12	0.11
Incorrect Detected Silence	0.00	0.00	0.39	0.29	0.32	0.27	0.25	0.21	0.23	0.18	**0.03**	**0.03**	0.01	0.01
Incorrect Detected Voice	0.10	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	**0.04**	**0.05**	0.11	0.10
Balanced (Silence vs. VAD)	no		no		no		no		no		yes		no	

Table K.2: Error-Estimations of the VAD algorithms, where the balance between voiced and unvoiced detection errors can be figured out. The total error is the sum of the incorrect detected voice ratio (incorrect detected voice samples related to the total sample count) and the incorrect detected silence ratio (incorrect detected silence samples related to the total sample count). If both ratios are similar, the parameters of the specific algorithm are trimmed to get the lowest total error. This behavior is known as ROC. For this table the $SNR_{dB,A}$ method is used.

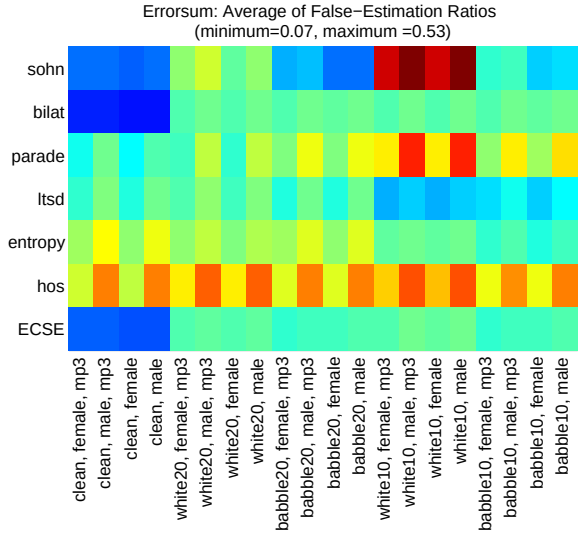
In figure K.17 all algorithms with a total estimation error lower than 25% in any condition (using denoised sound samples) are shown, where the $SNR_{dB,A}$ method is used.



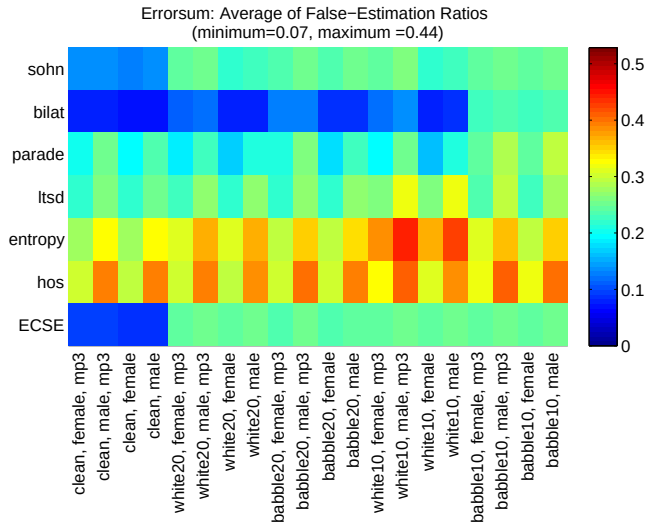
(a) No Denoising, Single Sentence Samples



(b) Denoised Input, Single Sentence Samples

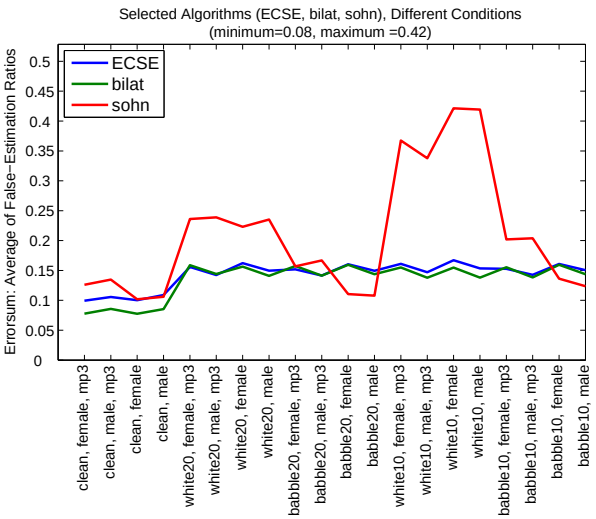


(c) No Denoising, Multi Sentence Samples

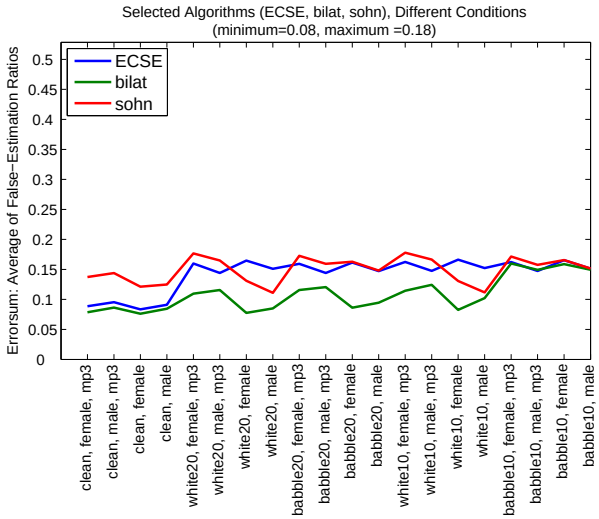


(d) Denoised Input, Multi Sentence Samples

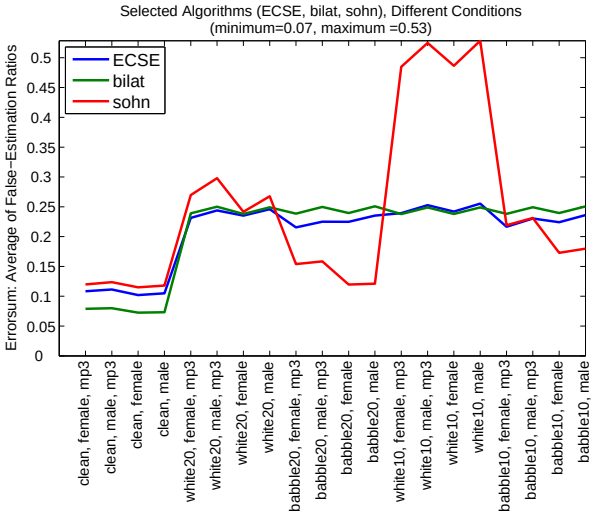
Figure K.16: Comparison of VAD algorithms, applied on different conditions (no noise, white noise, babble noise) and different genders (male, female); The algorithms are assigned to the ordinates, the conditions are shown at the abscissas.



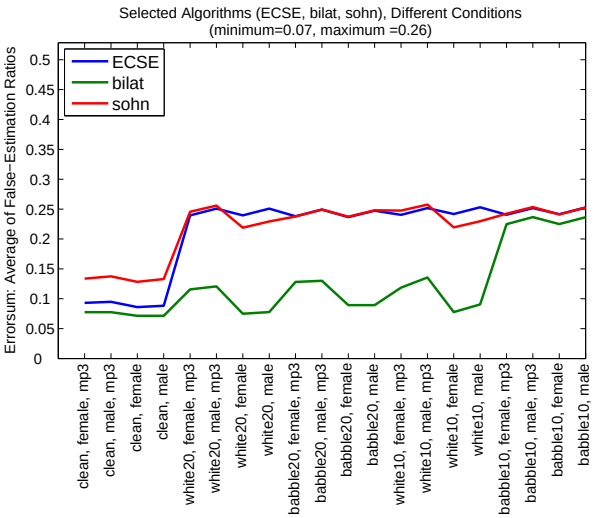
(a) No Denoising, Single Sentence Samples



(b) Denoised Input, Single Sentence Samples



(c) No Denoising, Multi Sentence Samples

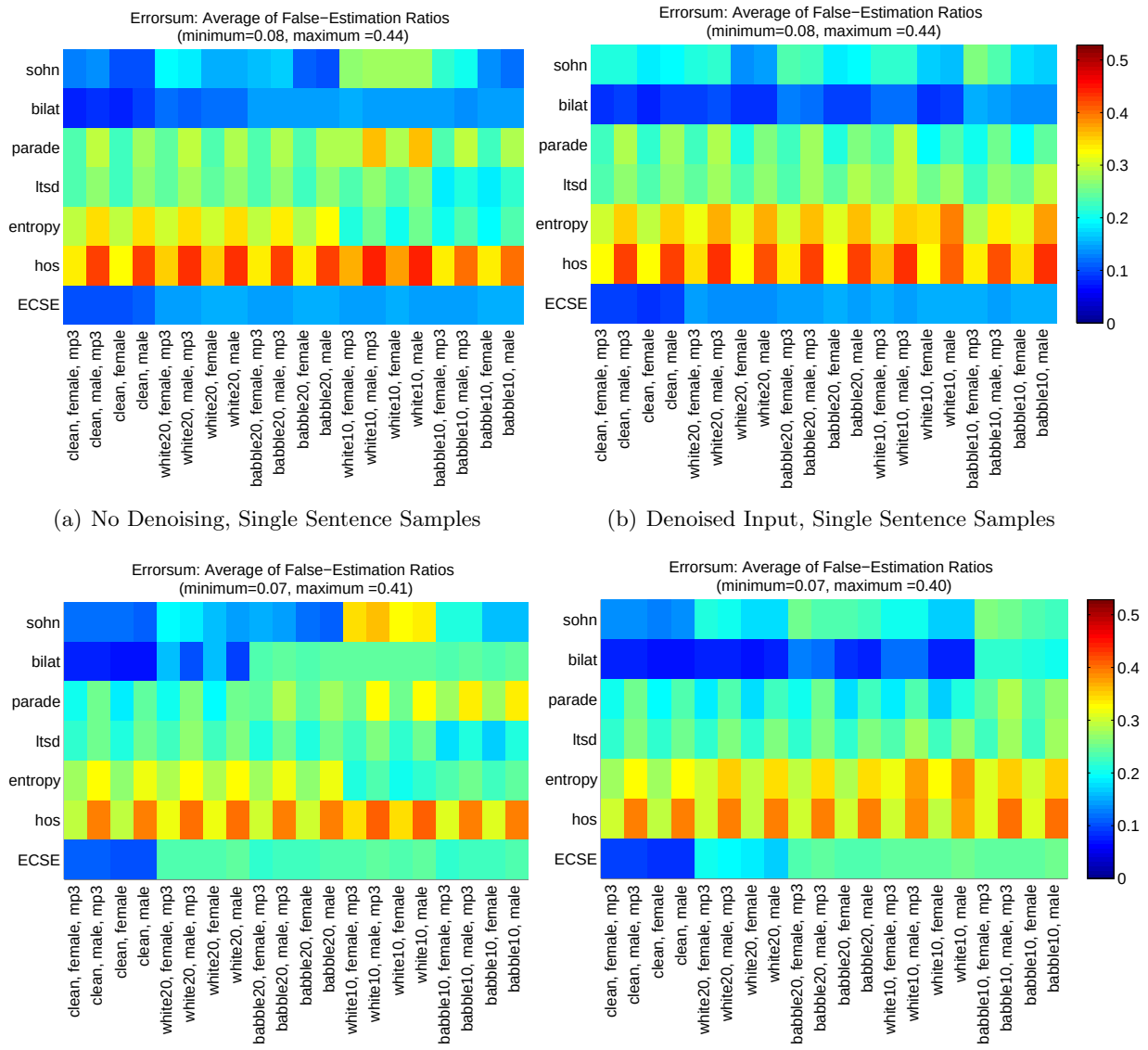


(d) Denoised Input, Multi Sentence Samples

Figure K.17: Comparison of selected VAD algorithms, applied on different conditions (no noise, white noise, babble noise) and different genders (male, female); The total error rates are are assigned to the ordinates, the conditions are shown at the abscissas. Here, the $SNR_{dB,A}$ method is used.

SSNR

The same evaluations are repeated, using $SSNR_{dB,E}$, but only 100 audio tracks for a single condition, which also should be enough for statistical evaluations. In figure K.18, the results of the evaluation are shown. The best three algorithms are extracted and shown in figure K.19.



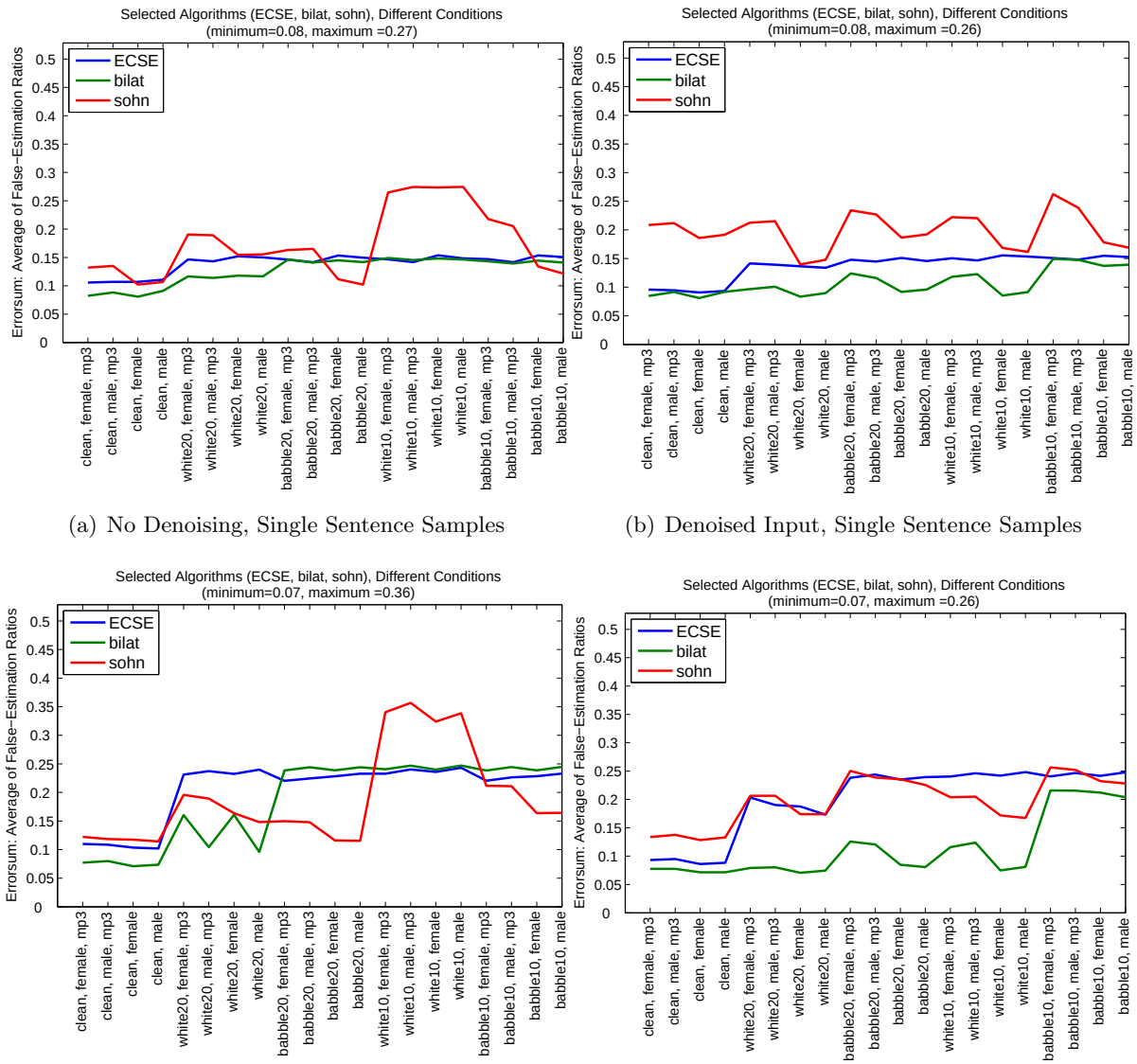
(a) No Denoising, Single Sentence Samples

(b) Denoised Input, Single Sentence Samples

(c) No Denoising, Multi Sentence Samples

(d) Denoised Input, Multi Sentence Samples

Figure K.18: Comparison of the VAD algorithms, applied on different conditions (no noise, white noise, babble noise) and different genders (male, female); The algorithms are assigned to the ordinates, the conditions are shown at the abscissas. Here, SSNR is used to calculate the noise levels, but only 100 audio files are analyzed.



(a) No Denoising, Single Sentence Samples

(b) Denoised Input, Single Sentence Samples

(c) No Denoising, Multi Sentence Samples

(d) Denoised Input, Multi Sentence Samples

Figure K.19: Comparison of selected VAD algorithms, applied on different conditions (no noise, white noise, babble noise) and different genders (male, female); The total error rates are are assigned to the ordinates, the conditions are shown at the abscissas. Here, SSNR is used to calculate the noise levels, but only 100 audio files are analyzed.

Bilateral Filter Influence

"BILAT" is designed to outsource the main computational complexity to the preprocessing step (bilateral filtering, denoising). Tests have shown that the core feature extraction also work well **without bilateral filtering**: Analyzing the total error rate of the algorithm, using the multi sentence track database (combined TIMIT samples), 20 dB $SNR_{dB,A}$ with white noise and applied denoising, and only male speakers, would lead to the total error of approximately 6%, which is almost the same result as if bilateral filtering would have been applied (see figure K.20).

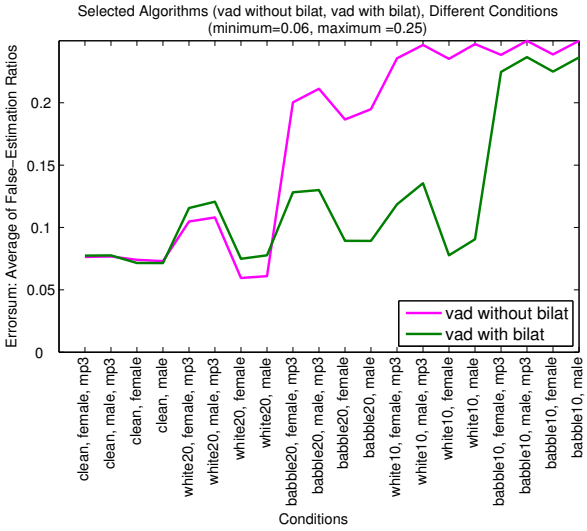


Figure K.20: Testing the algorithm proposed in section K.3.7 with and without bilateral filtering. In environments with 'harmless conditions' (only additive and white noise, high SNR, AGC of speech is applied,...) the algorithm doesn't require bilateral filtering as de-noising step. For more challenging conditions (babble noise, non-stationary environment, low SNR, no AGC ...) bilateral filtering has positive impact on the results.

K.4.4 Discussion

VAD Parameter Adjustments

The evaluation results of all algorithms are highly dependent on their parameter settings, on the preparation of the database and the error definitions. The results proposed in this paper are much worse compared to previous comparative publications like Wu and Zhang [2011], Ramirez et al. [2004a], Ramirez et al. [2004b], Fujimoto et al. [2008], especially when the error is calculated using $SNR_{dB,A}$. The reasons for this are:

- The VAD results proposed in this work are all interpolated and (thresholded to get binary results) to fit the reference signals, which introduce an additional error, mainly influenced by the algorithms' step sizes. This method avoids whitewashing the results: Bigger step sizes lead to bigger errors.
- As proposed in Wu and Zhang [2011], the TIMIT database is not well balanced according to speech and non-speech parts. Extending the non-speech part boundaries is a common way to deal with this problem, but would relax the algorithms' requirements. In this work this method is avoided to take the problems of short non-speech parts into account.
- When source code was available, the algorithms are mainly used with default parameters, which not always let to the minimum error for the evaluated conditions. An example for this is the algorithm proposed in Sohn et al. [1999]. It is designed to work well for speech enhancement, ASR or similar requirements. Adjusting the parameters in such way that the values for "incorrect detected voice" (false alarm rate) and "incorrect detected silence" are balanced would lower the total error approximately from 22% to 16% when analyzing the multi sentence track database, 20 dB SNR with white noise and applied denoising, and only male speakers. This behavior is expressed by the so called "ROC", as discussed in Pernía et al. [2007]).

L Fundamental Frequency Estimation

L.1 Introduction

A very important feature of speech signals is the perceptual feature *pitch level*. The related physical quality to pitch level is the F0 or even the corresponding fundamental period (T0). Usually, in literature no distinction between F0 and *pitch level* is made. Keep in mind that all methods referred in this work as PDAs rather calculate the physical features F0 or T0 than the perceptual feature *pitch level* itself. Depending on their calculation methods different definitions for pitch level, F0 and T0 are used. Common definitions are summarized in Hess [2007].

Using a laryngograph signal as pitch reference base, following definition may be used:

> "*T0 is defined as the elapsed time between two successive laryngeal pulses. Measurement starts at a well-specified point within the glottal cycle, preferably at the instant of glottal closure.*"

Without laboratory laryngograph signal measurement is not practicable. Nevertheless the laryngeal pulses can be reconstructed using advanced speech signal analysis techniques. The following definition of T0 covers this approach:

> "*T0 is defined as the elapsed time between two successive laryngeal pulses. Measurement starts at an arbitrary point within an excitation cycle. The choice of this point depends on the individual method, but for a given PDA it is always located at the same position within the cycle.*"

Even if laryngeal pulses are very useful for PDA evaluations, measurements in more realistic environments usually are based on other approaches. The algorithms proposed and implemented in this work follow the following definition:

> "*T0 is defined as the average length of several periods. The way in which averaging is performed, and how many periods are involved, is a matter of the individual algorithm.*"

A historical overview of PDAs is given in Gerhard [2003], whereby a more detailed and a more up to date comparison is provided in Hess [2007]. In this chapter only a subset of common concepts for pitch determination are reviewed.

L.2 Algorithm Overview

As presented in Hess [2007], each PDA can be structured in 3 parts:

1. Pre-processor
2. Basic extractor
3. Post-processor

Taking the *basic extractor* as criterion, 2 basic types of PDAs exist:

1. *Time domain PDAs* which share the same signal time base in all processing steps.
2. *Short-term PDAs* which translate the signals time base in the *pre-processor step*.

L.2.1 Short-Term Analysis PDAs

Speech is a non-stationary process. The *pre-processor step* of short-term based analysis algorithms split the speech signal into a series of overlapping frames. Within a single frame, the signal parameters are assumed to be quasi-stationary. Basically 3 pitch periods should fit into one frame. With this approach phase information as well as sensitivity to phase errors are lost. *Power spectrum methods* focus the information of one frame in a single sample. Related techniques have the same focusing effect. To this algorithm family belong *correlation techniques*, *frequency domain analysis*, *active modeling* and some *statistical approaches* (figure L.1).

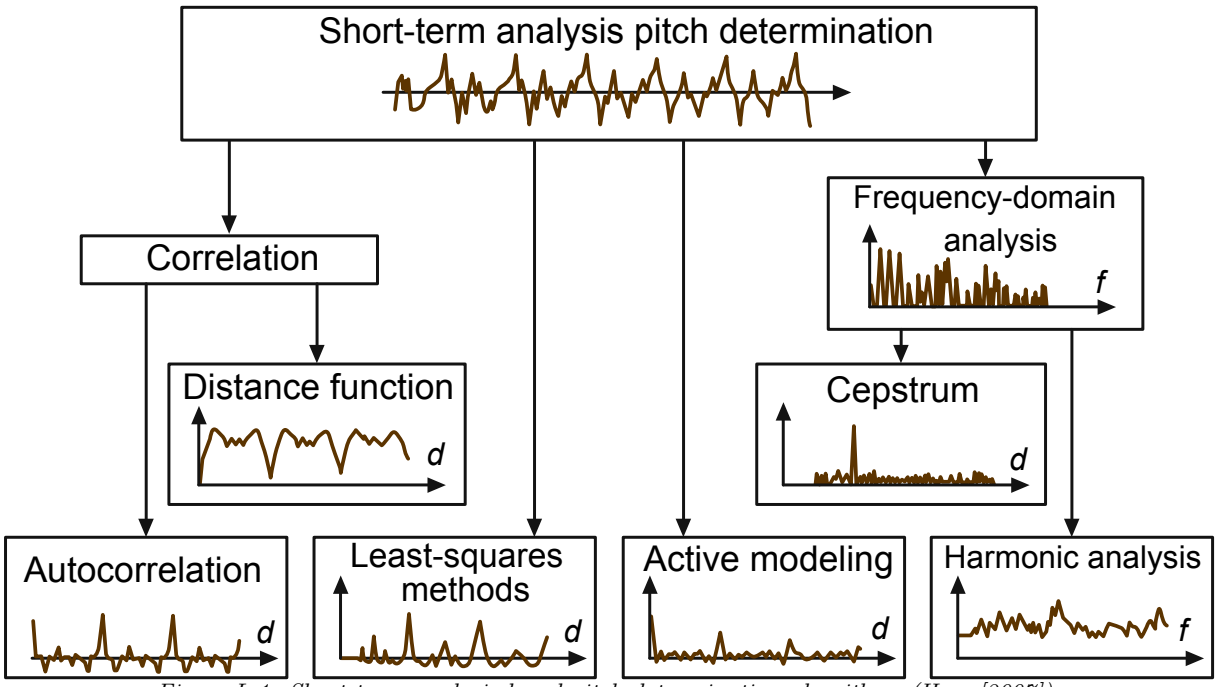


Figure L.1: Short-term analysis based pitch determination algorithms (Hess [2007]).

PDAs based on the Short-Term ACF

The discrete short-term ACF $r[d, q]$ for lag d at position q is usually given by

$$r[d,q] = \sum_{n=q}^{q+N-1} s[n]s[n+d] \tag{L.1}$$

where N denotes the integration window, d defines the correlation lag and $s[n]$ is the speech signal. A different definition for the short-term ACF is given by (de Cheveigné and Kawahara

[2002]):

$$r[d,q] = \sum_{n=q}^{q+N-d} s[n]s[n+d] \tag{L.2}$$

where the integration window shrinks with increasing values for the lag d. If the signal is zero outside $n = [q...q+N-1]$, both equations lead to the same result, but differ otherwise. The influence of the equations are shown in figure L.2. Other slightly different definitions may be used (non-biased ACF, centered results...). Details about the used ACF for a specific algorithm are provided in the related publications.

Usually windowed input signal is used, so the sample offset q is calculated by:

$$q = k \cdot N_{step} \tag{L.3}$$

where N_{step} is the step size of the framing procedure, given in samples, and k is the frame index.

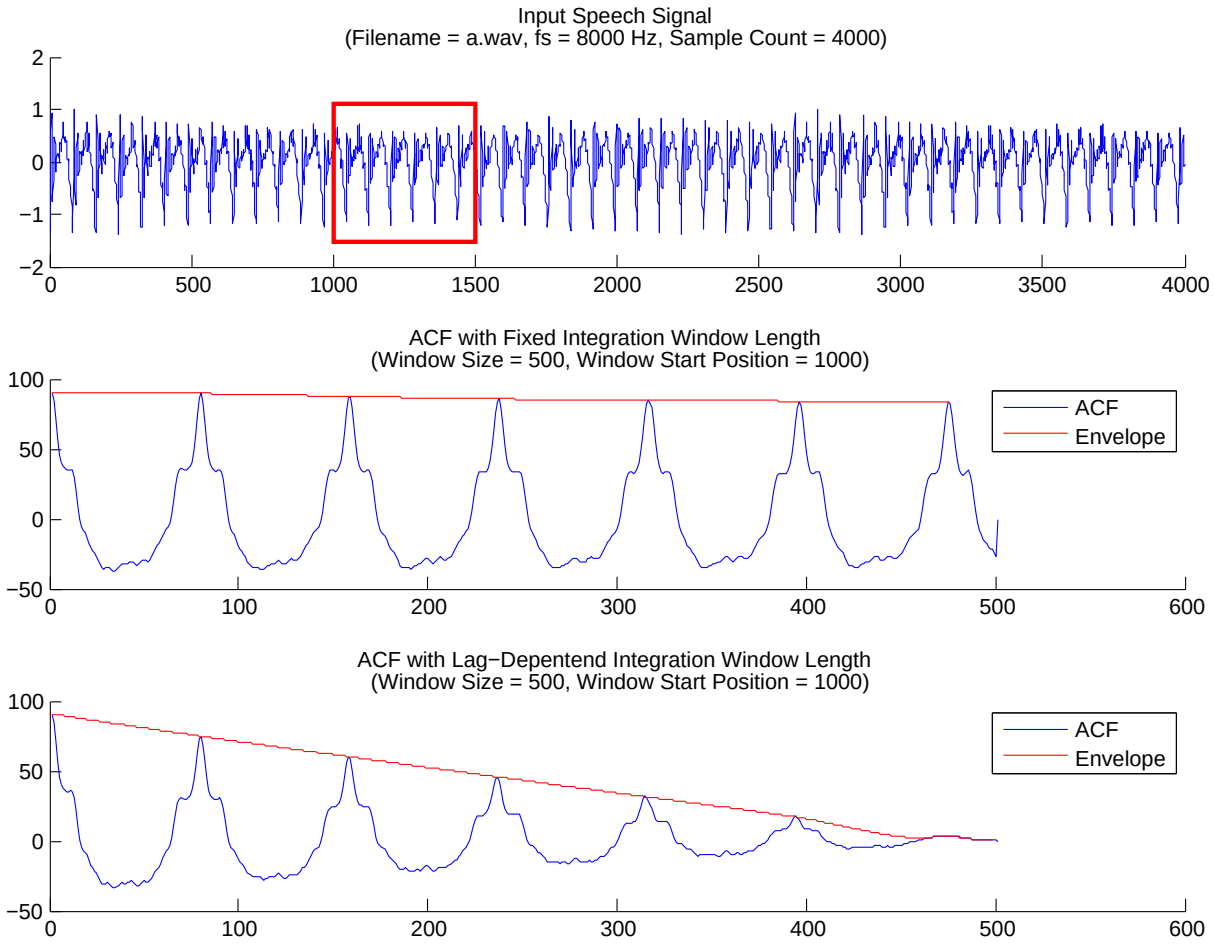


Figure L.2: Influence of different short-term ACFs. The audio signal is the vowel "a", spoken from a man, whereby the total duration is extended with Matlab. The methods used for the ACF calculations are equation L.1 and equation L.2, analyzed for the ACF lag range $\doteq$0...500. Note that the calculations results in a biased ACF and is not centered within the integration window.

When the signal is (quasi-) periodic, a strong maximum peak can be observed at lag d equals

the period T/T_0, T being the time-domain sampling interval of the speech signal. This method is often combined with *nonlinear pre-processing*.

PDAs based on Distance Functions

The AMDF as counterpart to the ACF if given by

$$AMDF[d,q] = \sum_{n=q}^{q+N-1} |s[n] - s[n+d]|. \tag{L.4}$$

where q is the sample offset, d the "distance function lag", N the analyze window and $s[n]$ the speech signal. When the signal is (quasi-) periodic, a strong minimum can be observed at lag $d = \frac{T}{T_0}$. This method doesn't require stationarity therefore small frame-length can be used.

PDAs based on ACF and Distance Function

Using the reciprocal of a distance function as weight for the ACF leads to robust pitch estimation algorithms for noisy environments (Shimamura and Kobayashi [2001], de Cheveigné and Kawahara [2002]). Normalized distance functions have increased values at lower lags:

$$DF[d,q] = \frac{\sum_{n=q}^{q+N-1} (s[n] - s[n+d])^2}{\frac{1}{d}\sum_{\sigma=1}^{d}\sum_{n=q}^{q+N-1} (s[n] - s[n+\sigma])^2}; \quad d > 0 \tag{L.5}$$

$$r[d,q] = \frac{\frac{1}{N}\sum_{n=q}^{q+N-d} s[n]s[n+d]}{DF[d,q] + k}; \quad k > 0. \tag{L.6}$$

When the signal is (quasi-) periodic, a strong maximum peak of $r[d,q]$ can be observed at lag $d = \frac{T}{T_0}$.

PDAs formulated as Optimization Problem Respected to the Analysis Window Length

Improper frame-length (too few or too many pitch periods within a single frame) introduces a significant error. Without adaption of the frame-length according to the pitch level the algorithms are non-optimal.
Assumed that the frame-length matches exactly 2 pitch periods, a similarity function expresses the relationship of two periods within a single frame:

$$s[n,q] = a \cdot s[n+p,q] + e[n,q]; \quad n = q, ..., q+p-1. \tag{L.7}$$

According to this equation an optimization problem with an error function J can be formulated and rewritten as *least square approach*:

$$\hat{p} = \underset{a,e}{argmin} \left(J = \frac{\sum_{n=q}^{q+p-1} (s[n] - as[n+p])^2}{\sum_{n=q}^{q+p-1} s^2[n]} \right) \tag{L.8}$$

The trial period $\hat{p}$ to obtain the frame-length and the speech pitch can be found by solving the optimization problem

$$\hat{p} = \underset{p}{argmax} \left(J = \frac{\left(\sum_{n=q}^{q+p-1} s[n]s[n+p] \right)^2}{\left(\sum_{n=q}^{q+p-1} s^2[n] \right) \left(\sum_{n=q}^{q+p-1} s^2[n+p] \right)} \right) \tag{L.9}$$

for an finite range $p_{min} < p < p_{max}$.

PDAs based on Double-Transform Methods

Spectral flatting suitable meet the problem of algorithms' sensitivity against strong first formants. PDAs including such a procedure are referred as *double-transform methods*. Common spectral flatting techniques are:

1. Time-Domain nonlinear distortion (e.g. center clipping; Rabiner [1977], Sondhi [1968])
2. Linear spectral distortion by inverse filtering (Markel [1972])
3. Frequency-Domain amplitude compression by nonlinear distortion of the spectrum (Hess [2007])

Frequency-Domain amplitude compression by nonlinear distortion of the spectrum is shown in figure L.3:

- Applying the *log-operator*, the well known *power-cepstrum* is calculated (Noll [1967], Oppenheim and Schafer [2004]). This PDA is insensitive against strong first formants, but sensitive against additive noise.
- Using the *squared-magnitude of the complex spectrum* as nonlinear distortion operator instead of the log-operator, the *ACF* is computed (Rabiner [1977]). As mentioned above, ACF based PDAs are insensitive against white noise, but sensitive against strong first formants.
- Using different nonlinear distortion operators is also suitable (Sreenivas [1984], Weiss et al. [1966]).

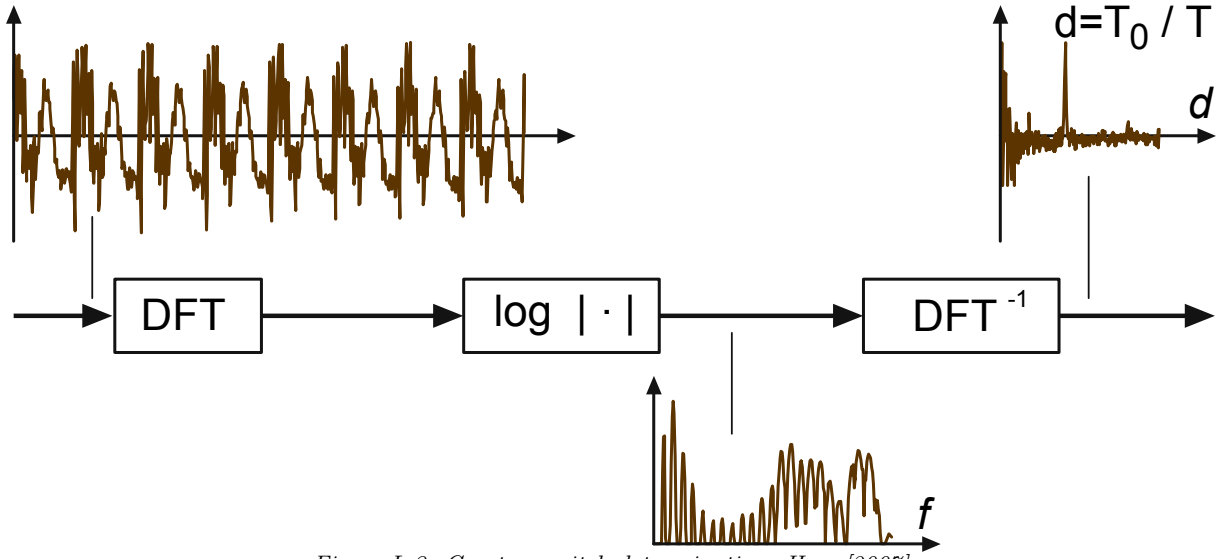


Figure L.3: Cepstrum pitch determination. Hess [2007]

PDAs based on Frequency Domain Methods

A better estimation of the F0 can be obtained if the harmonics of F0 contribute to the calculation. With harmonic pattern matching using compressed spectral signals the *greatest common divider* of the harmonics can be calculated, which is an estimation for F0 (figure L.4). Special mathematical methods decrease the required huge window length to make this technique applicable for speech signal processing (Martin [1982], Schroeder [1968], Brown and Puckette [1993]).
Applying the ACF to the filter-bank output of the amplitude spectrum is another way to estimate F0 of noise-corrupted speech (Lahat et al. [1987]).

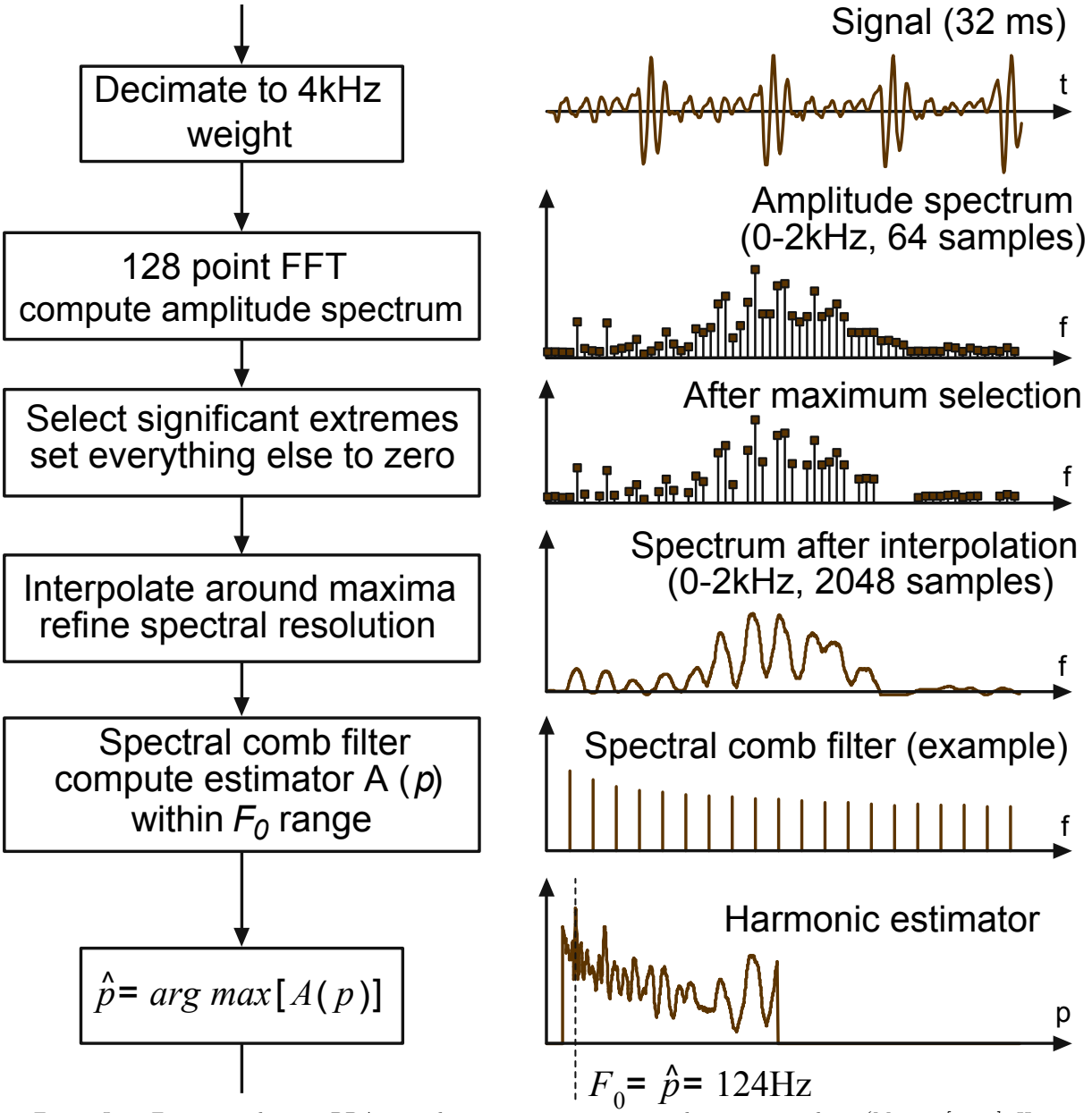


Figure L.4: Frequency-domain PDA using harmonic compression and pattern matching (Martin [1982], Hess [2007]).

PDAs based on Active Modeling

LP can be used to model the transfer function of the vocal tract or, if higher order filters are applied, to match the signal's harmonics. From the impulse response of the calculated linear prediction filter T0 can be obtained.

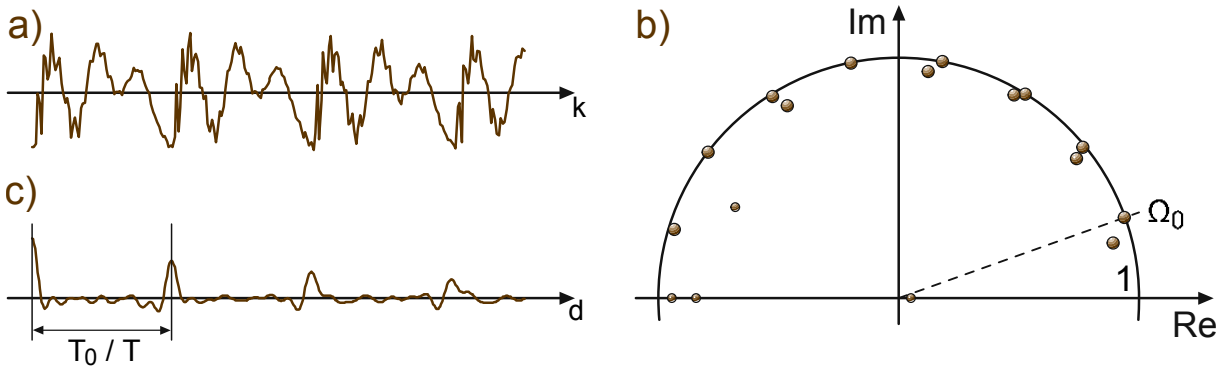


Figure L.5: PDA based on active modeling. In (a), the vowel [e] of a male voice is shown. After low-pass-filtering, the LP polynomial is calculated. In (b) the zeros of the polynomial are represented in the z-plane (upper half). (c) shows the reconstructed impulse response of the LP-filter. Taken from Hess [2007].

PDAs based on HOS

Moreno and Fonollosa Moreno and Fonollosa [1992] proposed and noise-robust PDA using a *third-order cumulant*

$$C[0,d] = \sum_n s[n]^2 s[n+d]. \tag{L.10}$$

Equation L.10 is used as ordinary input to common ACF based PDAs.
Wells [1985] presented a similar approach using the so-called *bispectrum*, which is defined as the Fourier transform of an third-order moment function.

L.2.2 Time-Domain Analysis PDAs

With *time-domain approaches*, the signal is tracked period by period which leads to so-called *pitch markers* (sequence of periodic boundaries). The underlying theory declares the pitch period as truncated response of the vocal tract to an individual glottal response. The vocal tract behaviour is approximated by a *lossy linear system*: The impulse response can be expressed as the sum of exponentially damped oscillations.

Local information on pitch is taken from each period individually which makes the algorithms sensitive to local signal degrations. On the other hand correct results can be expected even if the signal is aperiodic. Time-domain analysis PDAs are preferential used for high precision determination algorithms of the instant of glottal closure.

PDAs based on Fundamental Harmonic Processing

Applying extensive low-pass filtering in the pre-processor step, the F0 can be detected via the waveform of the fundamental harmonic. The analysis can be done with simple algorithms based on *zero-crossing*, *nonzero-threshold* or *threshold with hysteresis*.
A drawback of this algorithm is the required extensive low-pass filter.

PDAs based on Temporal Structure Simplification by Inverse Filtering

Suggesting that the temporal structure of the laryngeal excitation function is much simpler then the speech signal itself, signal simplification can be applied. *Inverse filters* try to cancel the

transfer function of the vocal tract and the laryngeal excitation function can be reconstructed. Such an algorithm is present in Hess [1976].
Problems occur if F0 comes near the first formant. Workarounds have to be applied to overcome this issue. This problems are not existent for methods based on glottal statistical properties.

L.3 Implementation

A small subset of the wide range of different PDAs types are implemented in Matlab. In this section, a more precise description of implemented algorithms is provided. The Matlab-scripts are well-structured and well-documented, so exact implementation details can be gained from the code itself. Existing implementations are not re-implemented at this state of the project.
Again, the algorithms are chosen according to aspects like real-time capabilities, level of awareness, implementation availability, complexity level, reliability of the results in noisy environments and if their usage is suitable.

L.3.1 ACF Algorithm

A PDA based on short-term analysis technique is present in Huang et al. [2001, page 321 ff.]. Generally expressed, the algorithm calculates T0 for each frame:

$$T0[k] = arg \max_{T} f(T|\mathbf{x}[k]) \tag{L.11}$$

where $\mathbf{x}[k]$ describes the windowed speech signal for frame k.
The main steps of the algorithm's data stream are implied in the pipeline diagram in figure L.6.

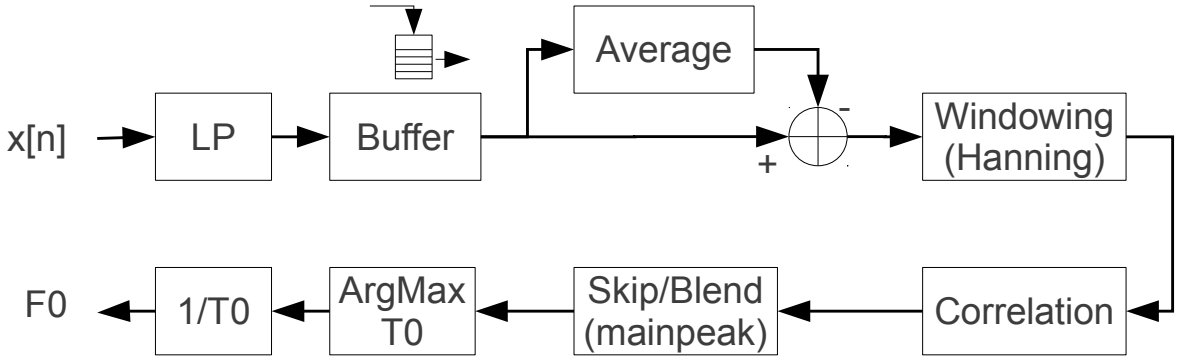


Figure L.6: The main steps of the algorithm proposed in Huang et al. [2001, page 321 ff.] are: Low-pass filtering (LP) within the pre-processor step, splitting the signal into frames (Buffer), local average subtraction, applying a Hanning window function (Hanning), ACF (Correlation), searching the main peaks whereby the signal energy peak at lag $d = 0$ is skipped (Skip/Blend and ArgMax) and result conditioning ($1/T0$). Remark that splitting the signal frames and applying the Hanning window can be combined to one step, subtracting the local average afterwards. For simplification, required delay-blocks are hidden.

Basic Processor

Interpreding speech signal as random process, where voiced signal is approximated with a sinusoidal random process and unvoiced signal is approximated with an zero mean Gaussian signal, candidates for T0 can be derived from the ACF as they result in significant maxima.

The statistical ACF may be approximated with the short-term ACF of equation L.1. To avoid biased results when applying the algorithm on speech signals, the local average of each frame is calculated and subtracted before the ACF is evaluated.

Using a rectangle window for the framing procedure leads to a problem known as *spectral leakage*. To decrease this influence, framing is done with an overlapping *Hanning window* function.

Window Size

This method requires a window size N which is at least $N \geq 2 \cdot T0$. For high values of T0 the stationary assumption fails. A possible solution for this problem is proposed in Huang et al. [2001, page 323], where different window sizes N for different lags d are recommended.

Strong First Formants

In Benesty [2004, page 186], algorithms based on ACF are reported to be error-prone when strong first formants exist.

L.3.2 PRAAT Related Algorithm

The algorithm presented in Boersma [1993] calculates the HNR in the ACF lag domain. If the value exceeds an certain threshold, the corresponding F0 candidate is trusted. The main steps of the algorithm's data stream are outlined in the pipeline diagram in figure L.7. To overcome problems introduced by windowing and to deal with the non-stationary nature of speech signals, normalized short-term ACF $r[d,q]'$ is used:

$$r[d,q]' = \frac{r[d,q]}{r[0,q]} \tag{L.12}$$

where $r[d,q]$ is the discrete short-term ACF for sample-offset q and lag d, according to equation L.2. Signals which have periodic parts show maxima at $d = n \cdot \frac{T0}{T_s}$, whereby T_s denotes the signal sampling interval. The ACF at lag $d = 0$ represents the signal energy, the normalized ACF at lag $d = \frac{T0}{T_s} = d_{max}$ represents the relative power of the periodic signal, called the *harmonic strength* R_0. The (logarithmic) HNR can be calculated with

$$HNR_{dB} = 10 \cdot log_{10} \left(\frac{r'_x[d_{max}]}{1 - r'_x[d_{max}]} \right) \tag{L.13}$$

where $r'_x[d_{max}]$ is the relative signal energy of an estimated T0 candidate. To make the results more accurate, interpolation around the T0 candidates is applied. As framed input signals are used, the mean of each frame is calculated and subtracted before ACF calculation is done. The effect of windowing, introduced by the framing process, is canceled by following approximation:

$$r[d,q] \approx \frac{r_x[d,q]'}{r_w[d,0]'} \tag{L.14}$$

whereby $r_x[d,q]'$ is the normalized short-term ACF for sample-offset q, and $r_w[d,0]'$ the ACF of the applied window. For performance reasons the ACF is calculated using the FFT. Remark that the equations in Boersma [1993] differ from the equations proposed here as the short-term ACF calculations in the paper are centered around the middle of the framing window function. Boersma compared different windowing functions (*Hanning window*, *rectangular window*, *Welch window* and *Hamming window*) and found out that the *Hanning window* outperforms the other

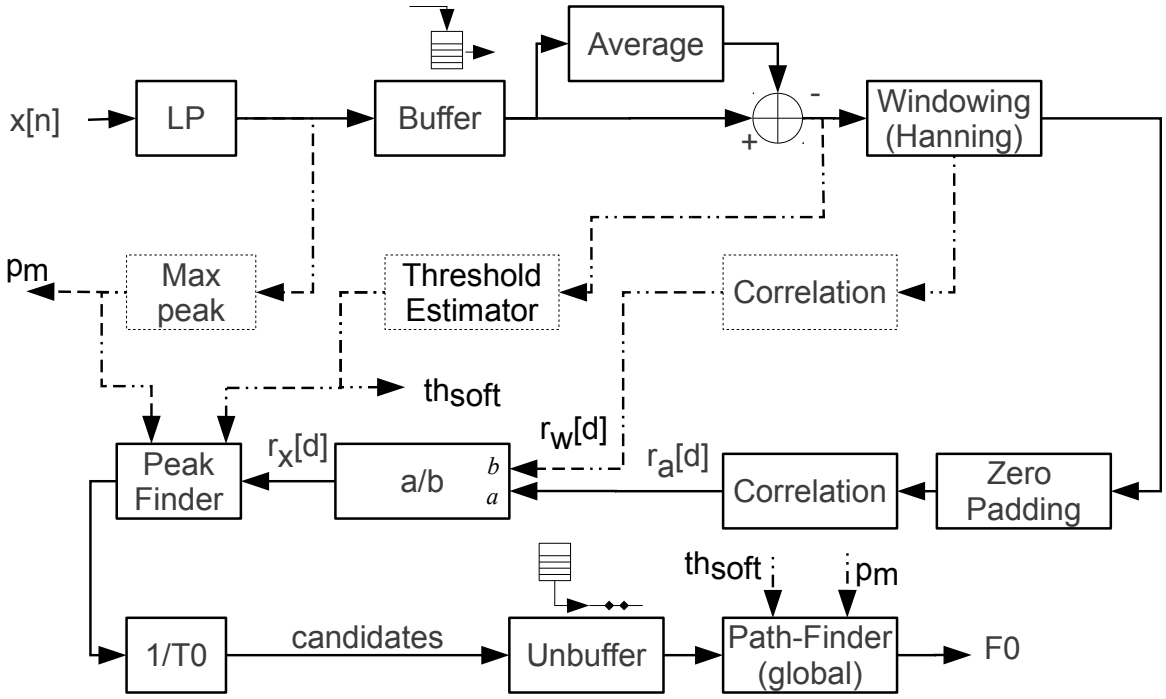


Figure L.7: The block diagram explains the main steps of the algorithm proposed in Boersma [1993]: Low-pass filtering done in the pre-processor step (LP), splitting the signal into frames (Buffer), subtraction of the local average, applying a Hanning window (Windowing), up-sampling and correlation, done by usage of FFT (Zero Padding, Correlation), canceling the errors introduced by windowing (a/b); The remaining steps calculate candidates for T0 which results in F0 candidates with associated harmonic strength values (Peak Finder, 1/T0) and estimate an global F0 course (Path-Finder). Global peak (Max peak) and automatic threshold selection (Threshold Estimator) increase the reliability of the T0 candidates and also are used to find the resulting F0 path. The correlation of the Hanning window, resulting in $r_w[d]$ is calculated once. For simplification, required delay blocks are hidden.

ones (Boersma [1993]). He also compared his algorithm among others against the *cepstrum* method (Noll [1967]) and noticed a more robust behavior of his algorithm within noisy environments.

Reference Implementation

The algorithm is implemented in PRAAT - doing phonetics by computer Boersma and Weenink [2011] (PRAAT). An global post-processing step based on is used to estimate the final F0 course. For real-time applications this post-processing step have to be approximated by a local approach.

L.3.3 Modified ACF Algorithm

A more robust algorithm can be designed by combining different approaches. This is done for the ACF algorithm proposed in Huang et al. [2001] (see section L.3.1) and the HNR based algorithm proposed in Boersma [1993] (see section L.3.2). An order statistic filter (OSF) (median-filter; discussed in Rabiner and Schafer [1978, page 158-161]), is used within the post-processor step. The main steps of the algorithm's data stream are implied in the pipeline-diagram in figure L.8.

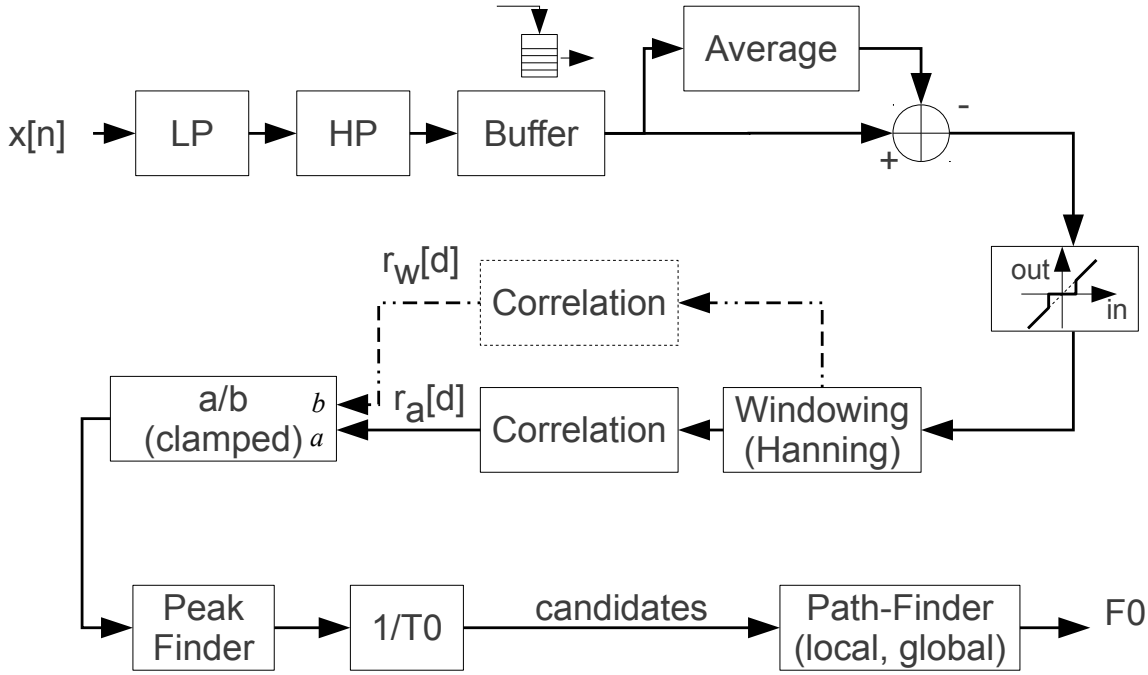


Figure L.8: The main steps of this compound algorithm are: Band-pass filtering (LP, HP), splitting the input signal into frames (Buffer), local average subtraction, center clipping, windowing with and Hanning window (Windowing) and applying normalized ACF (Correlation, a/b); The F0 course is estimated by an two pass path-finder (Peak-Finder, 1/T0, Path-Finder), which is a very course post-processing implementation and not prepared to use it within realistic conditions.

Pre-Processing

The pre-processing step consists of band-bass filtering (a low-pass filter followed by a high-pass filter), splitting the signal into frames and subtracting the local average which is done for each frame separately. Then the maximum signal amplitude peak within a single frame is obtained. To decrease the system's latency, this function can be combined with the subtraction of the local average.

Basic Extractor

A threshold function rejects too weak signal portions. The threshold is formed by the frame-wise calculated global maximum:

$$y[k \cdot N + n] = \begin{cases} x[k \cdot N + n] & |x[k \cdot N + n]| > K \cdot \max\limits_{i=0\ldots M-1}(|x[k \cdot N + i]|) \\ 0 & \text{otherwise} \end{cases} \tag{L.15}$$

where $x[n]$ is the pre-processed speech signal, k the actual frame index, N the step size of the framing procedure, M the related frame size and $y[n]$ the speech signal after the applied threshold function. With $0 < K < 1$ the (local) threshold valued is calculated for each single frame. It turned out that equation can be simplified to:

$$y[k \cdot N + n] = \begin{cases} 1 & |x[k \cdot N + n]| > K \cdot \max\limits_{i=0\ldots M-1}(|x[k \cdot N + i]|) \\ 0 & \text{otherwise} \end{cases} \tag{L.16}$$

Equations L.3.3 and L.16 are also known as center clipping (Rabiner and Schafer [1978, page 151 ff.]). The core calculation is based on normalized ACF as done in equation L.14. To avoid numerical errors, the windowed signal is clamped to zero near the frame boundaries.

Post-Processing

The final F0 course is estimated with two passes: The first pass collects the best local estimation for F0 by maximizing the harmonic strength value (see text passage L.3.2). The second pass forms a global average and estimates the final F0 course by picking the local F0 values. This is done by minimizing the distances function

$$F0[k] = \underset{f}{\operatorname{argmin}} \, |f_{average} - f_{candidates}[k]| \tag{L.17}$$

were k is the frame index, $f_{average}$ the global averaged course of F0 estimated in the first pass, $f_{candidates}[k]$ are the F0 candidates for the fundamental period at frame k, and f the selected frequency candidate.
The path-finder described here is a very rude method and not practicable in real environments. It is only implemented for *proof-of-concept* and to ease the comparison between the different algorithm results.

L.3.4 NCCF Algorithm

Algorithms based on ACF depends on the analyzed window size, which leads to biased results. Furthermore border problems and incorrect assumptions of stationarity for huge window sizes lead to another type of PDAs using normalized cross -correlation function (NCCF) (Huang et al. [2001, page 324 ff.]).
The main steps of the algorithms' data stream is outlined in the pipeline diagram in figure L.9. (Remark that it differs from the algorithm proposed in Huang et al. [2001] in some steps, especially the post-processing is not implemented according to the paper.)

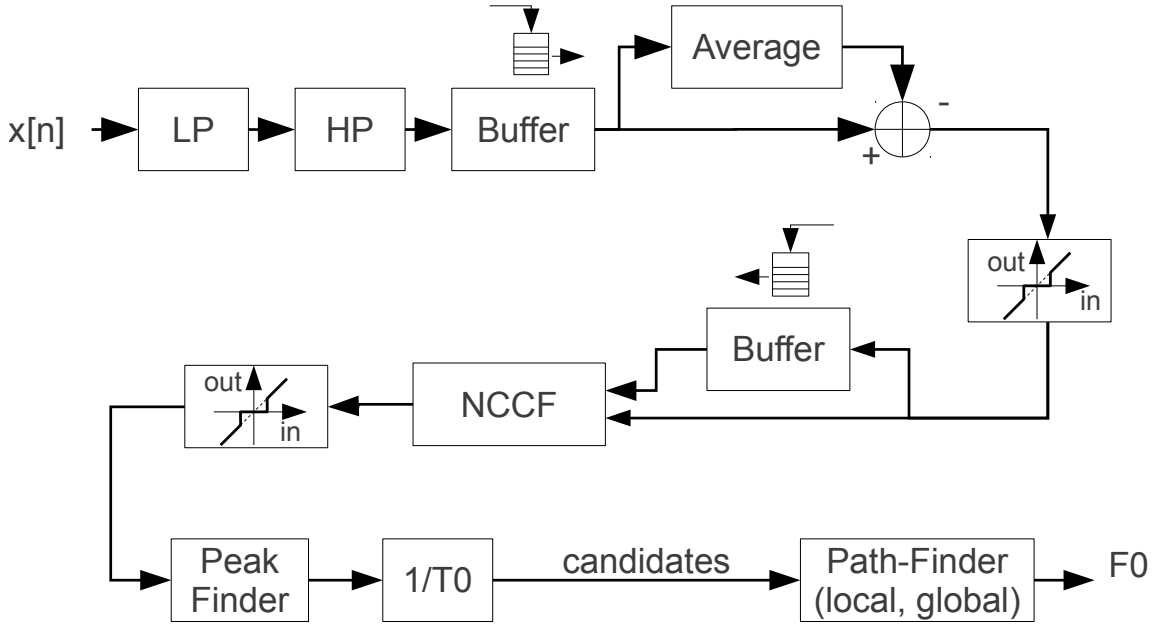


Figure L.9: The main steps of the implemented NCCF algorithm, related to Huang et al. [2001]: Band-pass filtering (LP,HP), splitting the input signal into frames (Buffer), local average subtraction, center clipping, NCCF (NCCF, Buffer), second center clipping operation, and a very coarse F0-path finder (Peak Finder, 1/T0, Path-Finder).

Basic Extractor

When interpreting the sampled and framed speech signal as data vectors, the NCCF can be formalized by the *inner product* of two vectors:

$$NCCF[d,q] = \frac{< x_q, x_{q-d} >}{|x_q||x_{q-d}|} \tag{L.18}$$

$$< x_n, y_l > = \sum_{m=0}^{N-1} x[n+m] \cdot y[l+m] \tag{L.19}$$

where x_q is th signal vector with N samples and $< x_n, y_l >$ is the inner product of two vectors. Using equations L.18 and L.19 leads to

$$NCCF[d,q] = \frac{\sum_{n=q}^{q+N-1} x[n] \cdot x[n-d]}{\sqrt{\sum_{n=q}^{N-1} x^2[n] \sum_{m=q}^{N-1} x^2[m+d]}} \tag{L.20}$$

Huang et al. noted that the constant number of samples involved in the calculation eliminates the bias-problem which is present when using ACF (see section L.3.1). The window size could be even lower than the pitch period which reduces the problem of non-stationarity signals.
A drawback is the increased computational effort compared to ACF since ACF can be efficiently calculated using FFT.

L.3.5 RAPT

Brookers [2011] provide a PDA called *FXRAPT* which implements the algorithm proposed in Talkin [1995]. The robust algorithm for pitch tracking (RAPT) uses NCCF and a global search function based on dynamic programming. The main steps of the algorithm's data stream are outlined in the pipeline diagram in figure L.10.

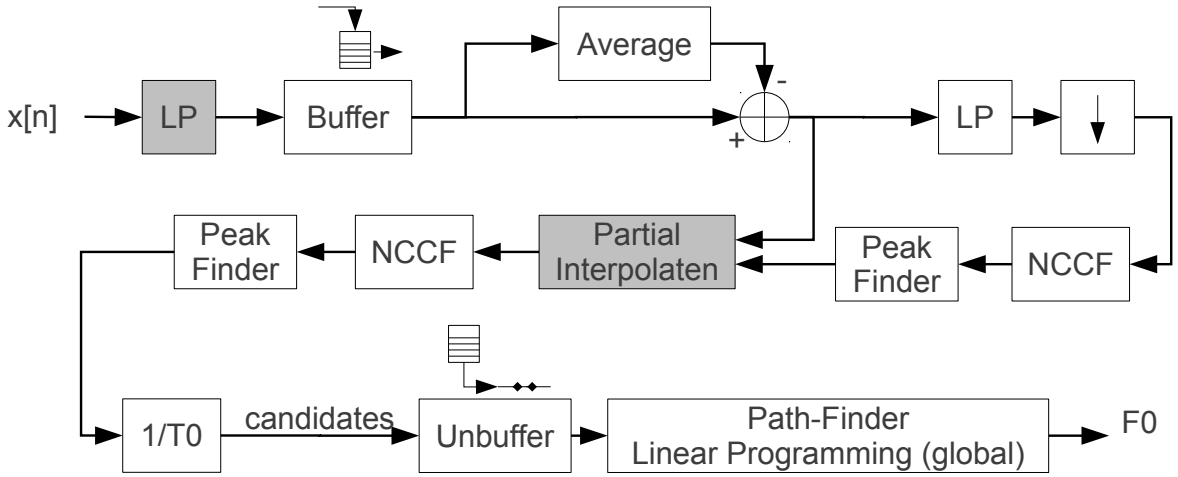


Figure L.10: The block diagram shows the main stepsof the algorithm proposed in Talkin [1995]: Optional low-pass filtering (LP), framing and local average subtraction (Buffer, Average, $\oplus$), down-sampling and a rude T0 estimation (LP, $\downarrow$, NCCF, Peak-Finder), parabolic interpolation around T0 candidates (Parabolic Interpolation), second pass NCCF and an accurate T0 candidate estimation applied on speech portions with a high sampling rate (NCCF, Peak-Finder), result conditioning ($1/T0$); The final F0 course is estimated with an global approach using linear programming (Unbuffer, Path-Finder). For simplification, required delay-blocks are hidden.

NCCF (Basic Extractor)

The NCCF is calculated at each frame k:

$$NCCF[d,q] = NCCF[d,k \cdot SZ] = \frac{\sum_{n=q}^{q+N+1} s[n]s[n+d]}{\sqrt{A\_FACT + e[q]e[q+d]}} \tag{L.21}$$

$$e[q] = \sum_{n=q}^{q+N-1} s^2[n] \tag{L.22}$$

$$d = 0...D-1, \quad q = k \cdot SZ \quad k = 0...K-1 \tag{L.23}$$

where $s[q]$ is the sampled signal *with zero mean* at sample-offset q, N is the analyze window length, K represents the total number of frames, D is the maximum value for the computed lags d, and k is the frame index. The frame step size is given in samples by the constant value SZ. To decrease the periodic parts for non-voiced speech frames, the constant A_FACT is introduced, which represents the absolute signal level.

Speech Transitions

Between the frames, local results need to be related. The time-domain distortion is modeled using the root mean square (RMS) ratio $rr[k]$:

$$rr[k] = \frac{RMS[k,h]}{RMS[k-1,-h]} \tag{L.24}$$

$$RMS[h,k] = \sqrt{\frac{\sum_{j=0}^{J-1}(w[j]s[k\cdot SZ+j+h])^2}{J}} \tag{L.25}$$

A *Hanning window* $w[q]$ of length $J = 0.03F_s$ is applied, whereby F_s denotes the sampling frequency. h is used to adjust the window centers of two sequential windows which should be 20 ms apart.
The spectral domain distortion is analyzed by calculating the *spectral stationary function* $S[k]$, using the *Itakura Distortion* (Itakura [1975]):

$$S[k] = \frac{0.2}{intakura(k,k-1)} - 0.8 \tag{L.26}$$

$$itakura[k,k-1] = \frac{aVa}{\hat{a}V\hat{a}} \tag{L.27}$$

where V is the ACF matrix with signal windows from the root mean square (RMS) ratio calculation (see equation L.25), a and $\hat{a}$ are linear prediction coding (LPC) coefficients using the LPC order P. The order P can be calculated with:

$$P = 2 + round\left(\frac{F_s}{1000}\right) \tag{L.28}$$

Dynamic Programming (Post-Processing)

The RAPT algorithm uses *dynamic programming* (Bellman [1957], Bellman [2003]) within the *post-processor* step to estimate the optimum voiced state and the optimum F0. The cost function includes weights for voiced and unvoiced frames as well as for inter-frame transitions. The approach is related to global optimum decisions which have been re-implemented for real-time applications. Details of the post-processing step can be reviewed in Brookers [2011], Talkin [1995] and Luo et al. [2003].

L.3.6 YIN Algorithm

The F0 estimation algorithm proposed in de Cheveigné and Kawahara [2002] adds important modifications to the ACF which leads to a very robust algorithm: It calculates a modified version of the well-known distance function (DF) by usage of the ACF (see section L.3.6). Furthermore A feature called “cumulative mean normalized DF” (see section L.3.6) is extracted and prepared for robust F0 estimation. The main steps of the algorithm’s data stream are outlined in the pipeline diagram in figure L.11.

When interpreting the ACF as the Fourier transform of the power spectrum, it measures the regular spacing of the harmonics. Replacing the power spectrum by the log magnitude spectrum leads to the well-known cepstrum method which can be seen as a *spectral whitening method* (Noll

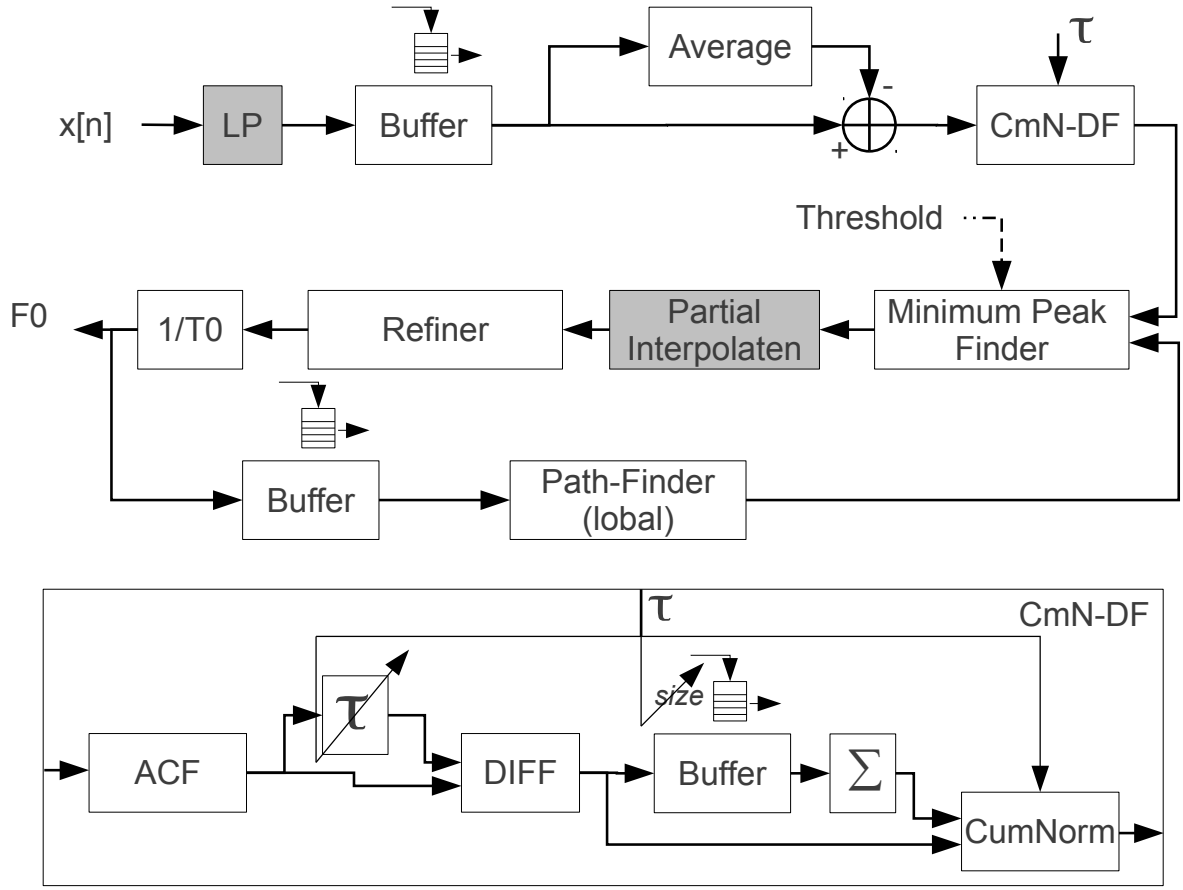


Figure L.11: The block diagram shows the main steps of the algorithm proposed in de Cheveigné and Kawahara [2002]: Low-pass filtering (LP), framing and local average subtraction (Buffer, Average, ⊕), the calculation of the "cumulative mean normalized DF" (CmN-DF, see section L.3.6) and the F0 estimation steps (Minimum-Peak-Finder, Interpolation,...). The main steps of the cumulative mean normalized DF are outlined in a separate block: The ACF if calculated from the windowed input signal, the difference function block (DIFF) simply implements equation L.30. The feature is extracted for increasing delays (τ) until a certain delay-threshold is exceeded (defined by minimum F0) or a suitable value for F0 is found (feature falls below a defined threshold).

[1967]). de Cheveigné and Kawahara enumerate further algorithms with similar effects. They evaluated the algorithm alterations regarding to the *gross (pitch) error rate*: The estimation is classified as wrong if the calculated value differs from a laryngeal-derived estimate by more than 20% (de Cheveigné and Kawahara [2002], Nakatani et al. [2008]). Furthermore gross errors are classified as *too low* or *too high*.

In de Cheveigné and Kawahara [2002], the algorithm is represented in six main steps. Each step adds significant improvements. These steps are summarized in the following part.

Step 1: Basic ACF Analysis

In de Cheveigné and Kawahara [2002] it is reported that applying ACF only (according to equation L.1) leads to a gross error rate of 10.0%. This evaluation served as reference for further improvements.

Step 2: DF

Adding the advantages of the DF to the algorithm, the gross error rate could be reduced to 1.95%:

$$x[q] - x[q+T] = 0 \quad \forall q \tag{L.29}$$

for exact periodic signal with period T. Expressing the equation in terms of ACF (according to L.1) leads to:

$$DF[d,q] = r[0,q] + r[0,q+d] - 2r[d,q+d] \tag{L.30}$$

where $r[d,q]$ is the short-term ACF for lag d, evaluated at sample offset q. T0 results are obtained from finding minima or maxima of $DF[d,q]$.

Step 3: Cumulative Mean Normalized Difference Function

To overcome errors introduced by the non-periodic nature of speech signal, cumulative mean normalized DF is used. This leads to a gross error rate of 1.69%. The modified distance function $DF'[d,q]$ is defined as

$$DF'[d,q] = \begin{cases} 1 & d = 0 \\ DF[d,q] / \overline{DF}[d,q] & \text{otherwise} \end{cases} \tag{L.31}$$

$$\overline{DF}[d,q] = \frac{1}{d}\sum_{n=1}^{d} DF[n,q] \tag{L.32}$$

Step 4: Absolute Threshold

To decrease an influence of subharmonic errors (*octave errors*) an absolute threshold method is applied, which reduced the gross error rate to 0.78%. If present, the minimum lag d is chosen so that $DF'[d,q] < threshold$. If the criterion can't be fulfilled, the global minimum is taken instead.

Step 5: Parabolic Interpolation

Parabolic interpolation around each local minimum of $DF'[d,q]$ is used to lower errors which are introduced if T0 is no multiple of the sampling period. In terms of computational effort the interpolation and evaluation around minima of $DF'[d,q]$ is cheaper than up-sampling the signal itself. Here, only a small gross error rate reduction (from 0.78% to 0.77%) is observed.

Step 6: Best Local Estimate

This describes a technique which deals with fluctuations of the F0 estimations. By evaluation of the vicinity of $DF'[d,q]$ the gross error rate is reduced to 0.50%.

L.4 Evaluation

For evaluation of the PDAs the PTDB-TUG database (see section J.3) is used. The F0 reference signal is extracted from the laryngograph signal and is provided by the database.

Again, according to de Cheveigné and Kawahara [2002], the PDA results are automatically shifted in time to get the minimum error between the reference signal and the estimated signal.

Even if some algorithms are used "as-is" (black-boxes), an uniform pre-processing and post-processing step is intended to make the results more comparable. Therefore the basic extraction algorithms are embedded in an uniform framework, which is shown in figure L.12.

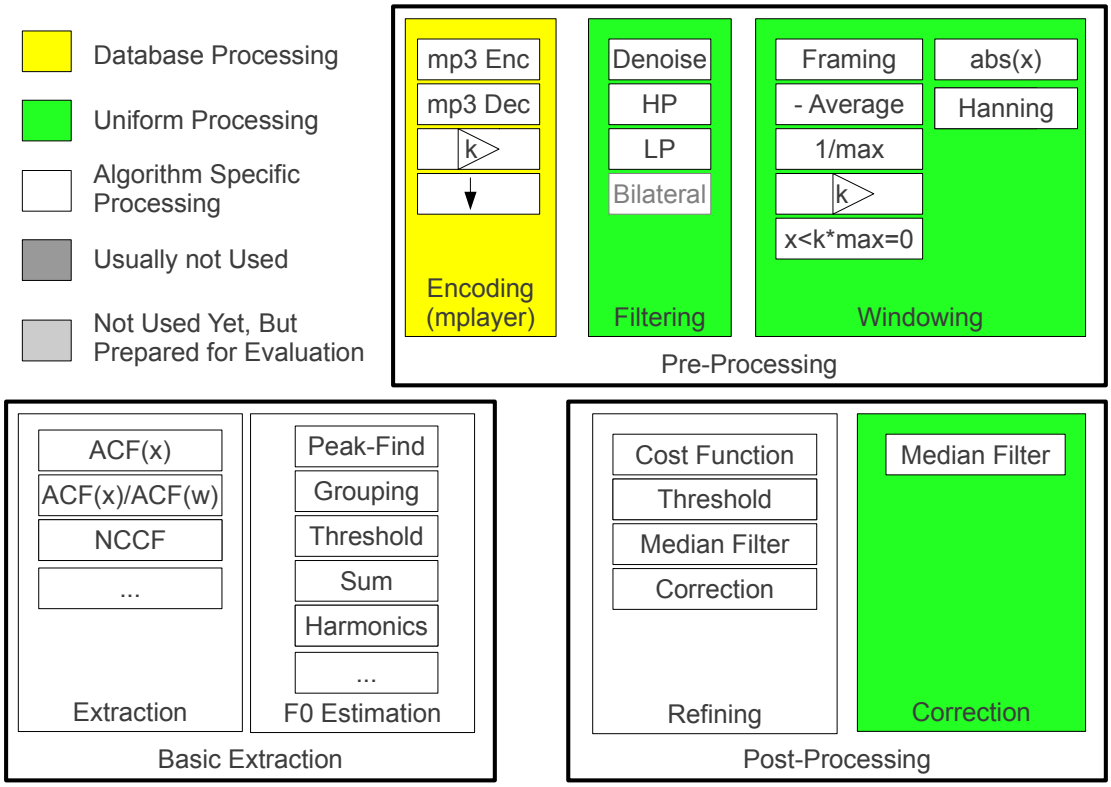


Figure L.12: Uniform PDA evaluation framework: Mp3-coding, mp3-decoding, amplifying and filtering/down-sampling is applied on the speech samples of the databases using Mplayer (MPlayer [2011]). The main filtering and windowing process is similar in all PDA algorithms. Special filter-operations like edge-preventing low-pass-filtering (e.g. bilateral filter), spectral whitening or window-related amplifying is implemented mainly for prospective purposes. The basic feature extraction step and parts of the post-processing step is formed by the individual algorithms. Finally median filtering refine the PDA feature results in an uniform post-processing-step.

Table L.1 includes the algorithms' name mapping between the specific names used in this report and the short labels used in the plots of the evaluational framework. Furthermore it is indicated if an algorithm was available in source-code or is re-implemented according to the related paper and the modifications stated in the corresponding section of this report.

Report Label	Short Label (Used for Plots)	Source-Code Freely Available
PRAAT related Algorithm, section L.3.2	HNR	yes, but re-implemented as Matlab Script
NCCF Algorithm, section L.3.4	NCCF	no, implemented as Matlab Script
RAPT Algorithm, section L.3.5	RAPT	yes, not re-implemented
ACF Algorithm, section L.3.1	XCORR1	no, implemented as Matlab Script
Modified ACF Algorithm, section L.3.3	XCORR2	no, implemented as Matlab Script
YIN Algorithm, section L.3.6	YIN	yes, not re-implemented

Table L.1: Algorithms' name mapping between the specific names used in this report and the short labels used in the plots of the evaluational framework. Furthermore it is indicated if an algorithm was available in source-code or is re-implemented according to the related paper and the modifications stated in the corresponding section of this report.

L.4.1 Evaluation Databases

From the PTDB-TUG database 2360 audio samples, spoken from men, and 1905 audio samples, spoken from women are used. The audio tracks have an average length of 7.3 seconds, enclosed in leading and trailing silence with an average length of about 0.1 seconds. Leading and trailing silence is assumed during the signal is lower than -60 dB related to the global maximum of the audio-file.

L.4.2 Evaluation Conditions

To test the algorithms in more realistic scenarios, the same conditions as proposed in section K.4.2 are used. Again, the evaluations are done with to different SNR definitions: Once using $SNR_{dB,A}$ and all available audio samples, and once using $SSNR_{dB,E}$ with a reduced audio set of 100 samples for each condition.

L.4.3 Results

A PDA estimation sample is defined as incorrect if the PDA estimation sample $est[n]$ differs from the reference signal sample $ref[n]$ more than 20%:

$$E[n] = \begin{cases} 1 & |est[n] - ref[n]| > 0.2 \cdot ref[n] \\ 0 & \text{otherwise} \end{cases} \tag{L.33}$$

where $E[n]$ is 1 if the PDA estimation result is incorrect, and 0 if the estimation result is within the allowed spread-range of 20%. As the window length and step sizes of the algorithms are different, the PDA results are interpolated so that they correspond with the reference signal. The reference signal itself provides a F0 estimation every 10 ms (which means the step size of the used PDA algorithm is 10 ms).

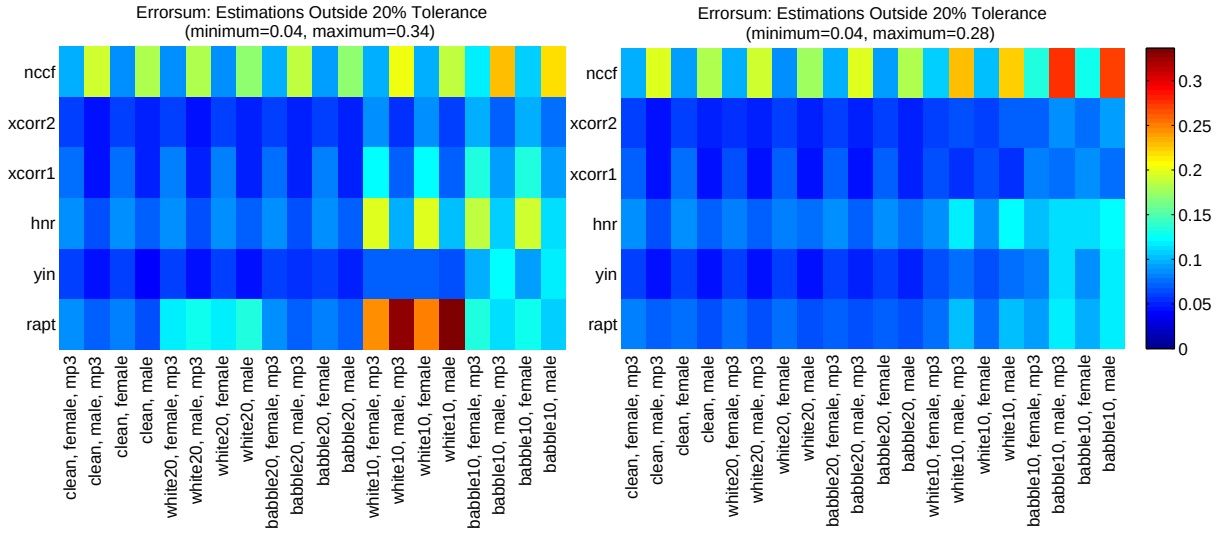
Statistical comparison of the total errors within the different conditions are shown in figure

L.13, where SNR_{dB} is used. The total error $E(alg_i)$ of a specific algorithm alg_i is calculated as follows:

$$E[alg_i] = \frac{\sum_{j=0}^{M-1} E_j[alg_i]}{M} \tag{L.34}$$

$$E_j[alg_i] = \frac{\sum_{n=0}^{N-1} E_j[n]}{N} \tag{L.35}$$

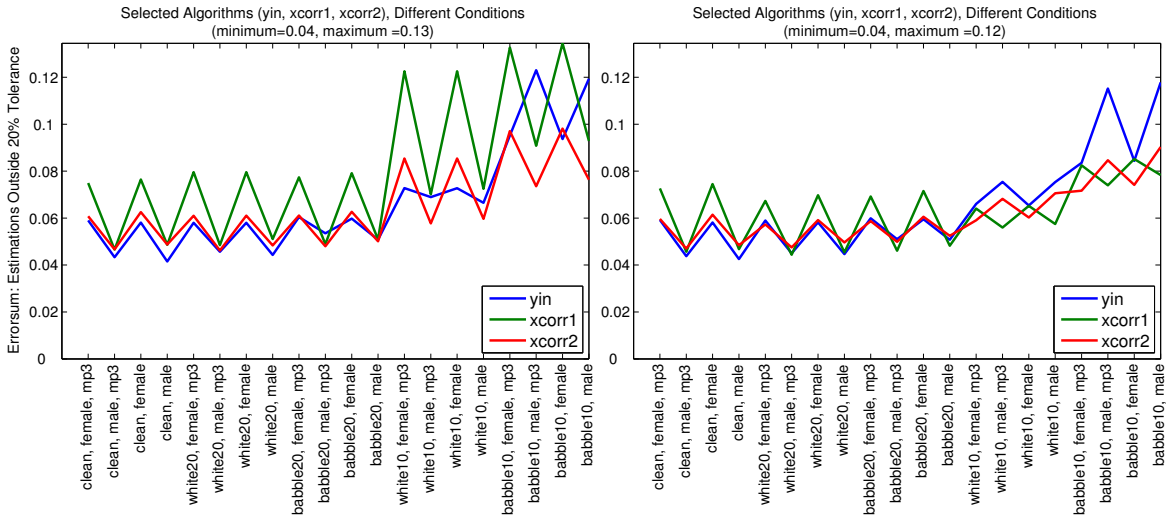
where M is the total count of test files, N is the number of F0 estimations within file j, $E_j[alg_i]$ is the estimation error of the specific algorithm alg_i for file j and $E_j[n]$ the estimation error count according to equation L.33.



(a) No Denoising (b) Denoised Input

Figure L.13: Comparison of the PDAs, applied on different conditions (no noise, white noise, babble noise) and different genders (male, female); The algorithms are assigned to the ordinates, the conditions are shown at the abscissas.

In figure L.14 all algorithms with a total estimation error $E(alg_i)$ lower than 15% in any condition (using non-denoised sound samples of plot L.13) are shown (plot a).

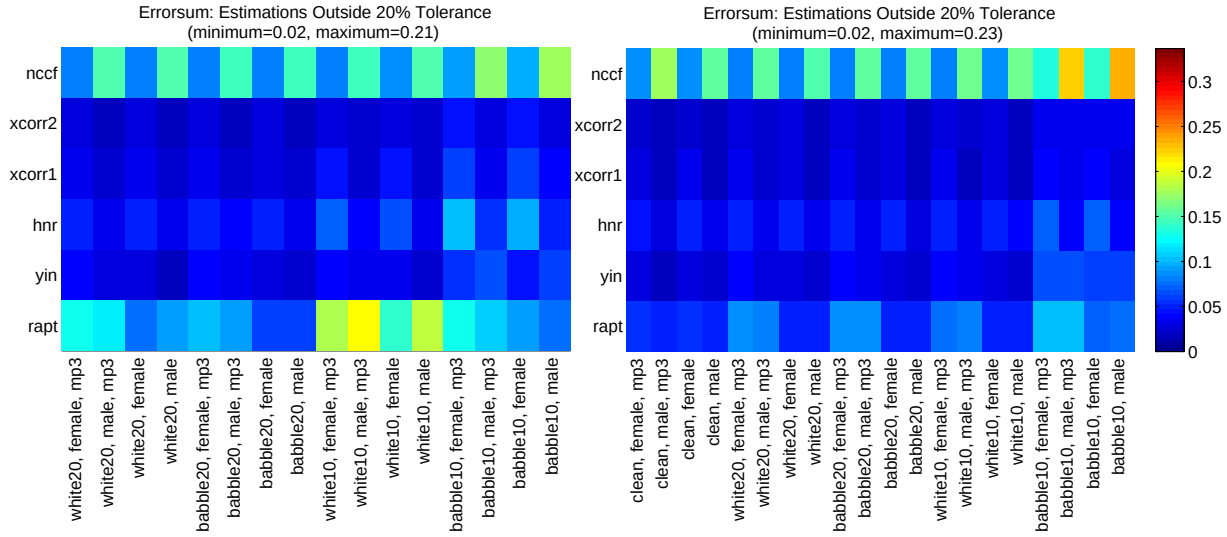


(a) No Denoising (b) Denoised Input

Figure L.14: Comparison of selected PDAs, applied on different conditions (no noise, white noise, babble noise) and different genders (male, female); The total error rates are assigned to the ordinates, the conditions are shown at the abscissas.

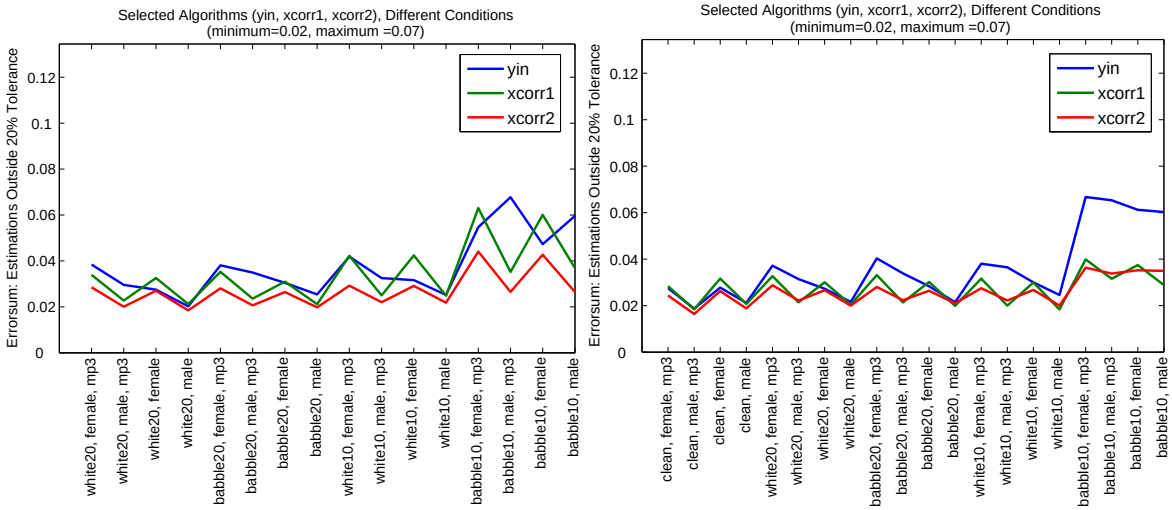
SSNR

Again, the same evaluations are repeated, using $SSNR_{dB,E}$, but only 100 audio tracks for a single condition. In figure L.15, the results of the evaluation are shown. The best three algorithms are extracted and shown in figure L.16.



(a) No Denoising (b) Denoised Input

Figure L.15: Comparison of the PDAs, applied on different conditions (no noise, white noise, babble noise) and different genders (male, female); The algorithms are assigned to the ordinates, the conditions are shown at the abscissas. Here, SSNR is used to calculate the noise levels, but only 100 audio files are analyzed.



(a) No Denoising (b) Denoised Input

Figure L.16: Comparison of selected PDAs, applied on different conditions (no noise, white noise, babble noise) and different genders (male, female); The total error rates are are assigned to the ordinates, the conditions are shown at the abscissas. Here, SSNR is used to calculate the noise levels, but only 100 audio files are analyzed.

M Discussion and Conclusion

In this chapter the results of the project are summarized and a prospect on future work is given. Regarding to the project goals a subset of "best matching algorithms" are chosen to provide a good basis for follow-up projects.

M.1 Evaluational Framework

For the project, a Matlab framework is designed, which provides different methods for F0 estimations and VAD, speech database management, statistical evaluation of speech features and interfaces to common speech processing tools like PRAAT. The design is based on multiprocessor simulation concepts and provide a semi-automatic task management. All algorithms use uniform preprocessing and post-processing steps, which can be enabled or disabled according to the evaluational requirements.

The algorithm implementations are not optimized for runtime, but provide a step-by-step computation according to the related papers. This is useful for understanding what is going on, but requires a re-implementation if runtime or even real-time is a criterion.

If source code of algorithms were available, those are used. Information about if an algorithm was implemented or available source code was used is provided in the tables L.1 and K.1.

M.2 Preferred Algorithms

The essential features for all algorithms are *reliability in noisy environments*, *real-time capability (with low latency)*, and *low computational complexity* to decrease the total power consumption when using a mobile device. Another determining factor is if implementations of the algorithms are freely available and evaluated, whereby the preferred programming languages are *ANSI-C* and *C++*.

According to this requirements, the preferred algorithms are discussed and suggestions for the best-matching candidates are provided.

M.2.1 VAD Algorithms

Selection According to "Reliability in Noisy Environments"

As shown in figure K.16, the algorithm based on LRT ("SOHN", section K.3.5), the algorithm based on pattern matching, using common speech signal parameters ("ECSE", section K.3.4) and the new designed method using signal variance and bilateral preprocessing ("BILAT", section K.3.7) outperformed all other evaluated VAD algorithms.

As denoising is usually provided in serious speech processing devices, only conditions with denoised signals participated in the final algorithm selection.

Adjusting the parameters of freely-available algorithms (SOHN, ECSE) would decrease the total error rates (see figure K.17), and similar results are expected from all three algorithms. That may be the reason why in Sohn et al. [1999] it is denoted that the SOHN algorithm outperforms the ECSE algorithm.

Selection According to "Availability of Implementation"

Most parts of the ECSE algorithm are available in Matlab and C++ source code, the SOHN algorithm is only available in Matlab source code.
As the BILAT algorithm is developed in this paper, only a Matlab source code implementation exists yet.

Selection According to "Complexity"

ECSE calculates an instantaneous pattern for each frame, which exists of 4 features. Additionally short-term and long-term energy is used. The hangover scheme itself doesn't introduce much complexity.
SOHN is based on statistical estimations, using sub-band features, and a simple noise estimation algorithm. The hangover scheme is based on a hidden markov model (HMM).
BILAT is designed to outsource the main computational complexity to the preprocessing step (bilateral filtering, denoising). Tests have shown that the core feature extraction also work well **without bilateral filtering** within "harmless conditions" (if only additive, white noise is present, and if the $SNR_{dB,A}$ value is at least 20 dB).

From the results discussed above, the BILAT algorithm seems to be a very good choice. In a more advanced implementation, bilateral filtering may be switched on or off according to the background conditions, which would save power and lower the average of the algorithm's latency. If a graphics processing unit (GPU) is available on the hardware device, the filter can be implemented very efficient with low latency.

M.2.2 F0 Algorithms

Selection According to "Reliability in Noisy Environments"

It is shown in figure L.13 that the algorithms based on ACF ("XCORR1", see section L.3.1, "XCORR2", see section L.3.3, and "YIN", L.3.6) outperformed all other algorithms. When denoising is used, the results of all three algorithms were very similar. Even if it denoted in de Cheveigné and Kawahara [2002] that the YIN algorithm should be significantly better and more robust than simple ACF algorithms, this wasn't the case in this evaluation.

Selection According to "Availability of Implementation"

The algorithms XCORR1 and XCORR2 are implemented in Matlab for the current work. The YIN algorithm is available as source code, which is basically Matlab code, whereby parts of the implementation are ANSI-C.

Selection According to "Real-Time Capability"

For the evaluation process, all global estimations are disabled. As all three algorithms are based on ACF, the estimations have similar runtime. Real-time would be easy to archive for all three algorithms, even if the YIN algorithm adds many modifications to the basic ACF.

From the results discussed above, non of the PDA algorithms really beats the other ones. As the XCORR1 method is very simple, it has the lowest computational complexity. The XCORR2 is a very similar method, but worked slightly better in very noisy environments. As in de Cheveigné and Kawahara [2002] the YIN algorithm is evaluated against standard correlation algorithms like XCORR1 and XCORR2, and many improvements are added and evaluated, it might would be the best choice.

M.3 Further Work

Speech analysis is a very important research area for all speech processing tasks. While different F0 and VAD algorithms are addressed in this work, many estimation concepts are not discussed here. One possible further direction is the extension of the framework with other state-of-the-art algorithms. If this goal is addressed, Matlab's object oriented programming capabilities should be taken into account. For database management, SQL would be a good choice.

To allow easy inspection, the evaluated algorithms aren't implemented efficiently. If runtime or even real-time is a criterion, re-implementation would be required.

Smartphones are a new type of mobile computing platforms, including many different capabilities for real-time speech analysis. Evaluating the proposed algorithms on such devices might be a further goal.

Beside special audio processing hardware, GPUs of conventional computers can be used for speech processing. Outsourcing computationally intensive parts of algorithms to GPUs is an increasing research field. As such GPUs also available on smartphones, this method may solve many real-time issues.