Real-Time Pitch Tracking Algorithms in C to Test Model Extraction

Janina Reuter

Bachelor's Thesis
April 2019

Real-Time and Embedded Systems Group
Department of Computer Science
Kiel University

Advised by
Christoph Daniel Schulze

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

_____

Abstract

Pitch tracking of human speech is one dimension of hearing and provides important information for our everyday live. While this is no effort for humans, an algorithm can hardly imitate human perception. Numerous algorithms for pitch estimation were developed, first improving on accuracy and then on robustness against background noise. At this point there exist pitch trackers for multiple voices at a time estimating and assigning the pitch accurately.

This work implements several pitch tracking algorithms to set up a versatile real-time system. Additionally, an algorithm combining the estimations from different pitch trackers based on properties of the human speech to further improve their estimations is presented. A qualitative evaluation shows the applicability of each algorithm and performance tests show that they are suited for real-time applications.

Visualizations of programs provide an intuitive and deeper understanding as an important property of software development. To provide a visual representation even for non-visual languages, such as C, a suitable representation has to be found. Led by the example of pitch tracking algorithms, different representation types along with a new diagram type called *rainbow rails* are discussed. Listed elements focus on either a holistic view or partial representations, which constitute a pseudo-holistic view if combined with interactivity.

Acknowledgements

I would like to thank Prof. Dr. Reinhard von Hanxleden and the Sonoware team for the unique opportunity to work on these two new and interesting topics. Especially, I would like to thank Merikan Koyun for his constant support in every concern and Christoph Daniel Schulze for inspiring discussions and his writing advise, which improved my understanding for graphics and writing techniques so much.

I want to thank Jette for her aesthecial and intellectual support and Mrs. Robinson showing me that only sky is the limit. Moreover, I want to thank every member of the working group for their company and discussions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Pitch is the "auditory perception of tone" [Tal95], for example in human speech. Pitch tracking has a wide range of applications in different areas, such as speech communication, phonetics and linguistics, education, and medicine [Hes12].

Audio data processing, transmission, analysis and synthesis of speech as well as semantics and speaker recognition in speech communication rely on a determination of the pitch. Besides speech and speaker recognition also the recognition of emotions and affect uses pitch tacking [SBS+11].

Phonetics and linguistics use pitch tracking for the "investigation of prosodic and phonetic features" [Hes12]. Prosody is concerned with properties of syllables including rhythm, intonation, and tone. For some languages, such as Chinese, the semantics of pronounced phonemes differ with intonation. Therefore, pitch tracking in these cases is required for speech recognition. In other languages the accentuation of words influences the semantics of single words, which supports differentiating the meaning of homonyms, which are equally spelled words with different meanings, or even the meaning of a complete sentence by the use of sarcasm and irony.

A common difficulty for everyone when learning a foreign language are different pronunciations, language specific prosody, and new phonemes. The range and modulation of pitch is speaker dependant. Therefore, pitch tracking can be used for speaker recognition. In the field of education, learning how to pronounce words correctly in various contexts is a very difficult task for deaf people. Comparing the course of their pitch to a course with correct intonation can support their understanding of speech pronunciation.

When hearing impaired regain auditory perception by cochlear implants, hearing and pitch estimation sometimes needs to be (re-)learned, which is especially difficult for pre-lingually deafened children. One specific use case of pitch estimation is, therefore, Mobile Games with Auditory Training (MOGAT) [ZST+12] providing a playful approach to support their auditory training and intonation. It contains three musical games and is extendible by auditory therapists to design individual training setups. The pitch estimation is implemented with an algorithm called YIN, which also has been implemented for this work.

Considering a system for pitch tracking, just like any other implementation of sufficient size, understanding each line of code is a prerequisite to understanding the system completely. However, the number of lines of code easily exceeds a limit for easy understanding. In fact, the human brain is only capable of processing up to 7 ± 2 independent pieces of information at a time [Mil56], which is exceeded by (almost) every computer program. Exploiting these characteristics of the human brain by graphical structuring of information is just one motivation for a diagram extraction from code.

Understanding the existing code is necessary for refactoring and extending it. Documentation can make the process of understanding legacy code easier, though textual descriptions are often not sufficient. When a system is developed using diagrams for modelling, these also offer an opportunity for additional documentation. However, there is a vast amount of legacy code, which was not developed and modelled using a graphical representation. Programmers documenting their just finished module or programmers maintaining an old code base might both profit from an automatic diagram generation from a code base. Visual representation of the interactions on specific data can aid in debugging.

1.1 Related Work

This section discusses related work and is followed by a section about the problem statement. Using this order, the implemented algorithms get introduced in the context of related work.

There are numerous pitch trackers and a full coverage of approaches goes beyond the scope of this work. However, the Subsection 1.1.1 gives an overview for some of the more relevant approaches. Related work regarding code visualization is discussed in Subsection 1.1.2.

1.1.1 Pitch Tracking

As already mentioned, pitch trackers are ten a penny. There even exists a pitch tracker named Yet Another Algorithm for Pitch Tracking (YAAPT) [KZ02]. This subsection focuses on naming relevant pitch trackers with respect to a brief historical development of the research focus. Details on the functionality are held back since the required fundamentals are to be discussed in the next chapter.

The first popular method for pitch tracking was Cepstrum. The method was developed in 1963 [BHT63] and applied to pitch tracking in 1967 [Nol67]. Cepstrum already works well on an audio signal of pure speech, but real-world applications with this condition are rare. The definition of pitch relies on human perception. Most pitch trackers, including Cepstrum, actually measure the Fundamental Frequency (F0), a quality of periodic signals, which correlates well with the actually perceived pitch and is easier to compute than the perceived pitch.

Precision of pitch determination generally degrades as the intensity of background noises increases. Yet, efforts for noise robustness were mainly the focus of research in the pitch tracking domain until around the 2000s. One algorithm clearly focussing on maximum robustness is RAPT [Tal95]. RAPT and another algorithm called YIN were considered best performing in the 2000s [PWP+11b]. The already mentioned YAAPT is based on RAPT. Similar to this, there are many pitch tracking algorithms with rather small differences. Some modify an existing algorithm to suite the needs of a specific use case: YIN-Bird, for example, improves YIN for the pitch tracking of bird vocalizations. Another example is the Simplified Inverse Filter Tracking (SIFT) [Mar72] algorithm, which is based on Cepstrum and a foundation also used by RAPT.

The approaches to pitch tracking reach from algorithms evaluating specific functions, such as RAPT and YIN, to statistical models, such as for example described by Tokuda et al. [TMM+99]. TAPS [HL13] and PEFAC [GB14] both take advantage of certain characteristics of the human voice. Additionally, pitch estimation based on TAPS can be done with reconstructing the estimation from a given data set of sufficient size. Slaney and Lyon present a pitch estimator actually estimating the human perception instead of F0 [SL90], which is based on the Duplex theory of pitch perception from Licklider [Lic51].

More recently the research also investigated on estimating multiple pitches at a time [CJ09; CSM93; KT99; AJC15]. Additionally to the problem of estimating each pitch, multi-pitch tracking also requires tracing of which estimation belongs to which speaker and estimating the number of speakers.

1.1.2 Visualization

There are several implementations for extracting a diagram from a code base. This section lists and explains some of the models.

Visual Paradigm[1], which is maintained by Visual Paradigm International, is mainly used for modelling in the Unified Modelling Languages (UML) including code generation from UML models. The tool also includes a generation of UML diagrams from source code.

[1] https://www.visual-paradigm.com/

ExplorViz[2], which is developed by the Software Engineering research group at Kiel University, is a monitoring system providing a live communication trace visualization of software landscapes. Objects of the visualization are packages, classes, and functions without details on the functionality.

NDepend[3] is a Visual Studio extension for the static analysis of .NET managed code. It supports several code analysis features, such as dependency visualization, heat maps, and hot spots visualizing data points evaluated by various metrics, and trend graphs visualizing the temporal development of the code by again different adjustable metrics.

Flow[4] is an IntelliJ plugin developed by Yiquan Zhou and Yoann Buch. It contains two parallel graphical views of the code base. The *call graph* shows dependencies between packages, classes, and functions and the *flame chart* visualises the execution paths of the program at a function level.

Sourcetrail[5], published by the Coati Software KG, is an offline cross-platform source explorer. Call graphs and abstracted data flow are visualized in an interactive environment. Filtering the desired information from the code base by any symbols is supported by fuzzy code search.

Code maps[6], which is available at the Visual Studio Marketplace, is a visualization tool for dependencies displayed as a hierarchical map. The representation is generated once without an interactive updating function.

A detailed visualization developed for legacy C code was presented by Smyth, Lenga, and von Hanxleden [SLH16]. The model extraction is implemented in the framework Kiel Integrated Environment for Layout Eclipse RichClient (KIELER), which is a research project developed by the Real-Time and Embedded Systems Group at Kiel University, using the representation of Sequentially Constructive Charts (SCCharts) [HDM+14]. The information contained in the C code represents all details, while utilizing the hierarchy of SCCharts structuring by functions and control statements to maintain readability.

While most of the related work focuses on generating a representation of fixed predefined structure, this work discusses different approaches on how to visualize which information. Although, there are several tools for model extraction, these often represent the code on an abstracted level. Insight into the functionality of specific methods is not contained in the representation but is especially important for the use case of understanding someone else's code. To combine this issue with the scalability needed for the problem size in real-world applications, interactivity is discussed on different levels in this work.

1.2 Problem Statement

The goal of this thesis is to set up a versatile system for real-time pitch tracking. In this context strategies for the visualization of C code have to be developed.

To build a versatile pitch tracking system, algorithms with different approaches and reportedly good estimation quality have to be implemented. The general classification of pitch tracking algorithms distinguishes between time and frequency domain algorithms. RAPT [Tal95] and YIN [DK02] are considered especially robust and well working algorithms in the time domain [PWP+11b]. TAPS [HL13] and PEFAC [GB14] operate in the time domain and utilize different characteristics of human speech. The algorithms are expected to complement each other well due their versatile approaches and reportedly good estimation quality [HL13; GB11]. To utilize the advantages of implementing multiple pitch

[2]https://www.explorviz.net/
[3]https://www.ndepend.com/
[4]http://findtheflow.io/
[5]https://www.sourcetrail.com/
[6]https://docs.microsoft.com/en-us/visualstudio/modeling/map-dependencies-across-your-solutions?view=vs-2019

trackers, an algorithm, which combines multiple estimations, has to be implemented. The estimation quality and the performance in regard to the real-time suitability has to be evaluated.

For solving the problem of code visualization, general approaches have to be developed and discussed with respect to usage scenarios. The question of whether to represent all of the information or only parts depends on the intention behind the creation of the diagram. Therefore, holistic and partial representations both have to be further discussed. In larger diagrams readability can be restraint. Interactivity is an approach, which can support the understanding of details in a larger diagram. These details are important because all information contained in a diagram cannot be grasped at first view, but rather is obtained by setting up a mental map puzzling pieces towards an understanding of the whole system.

1.3 Outline

Part I describes and evaluates the pitch tracking algorithms RAPT, YIN, TAPS, and PEFAC together with a combining algorithm that is based on candidates estimated by other algorithms, such as those presented. To achieve a deeper understanding of the pitch tracking problem, Chapter 2 provides fundamentals of speech production, digital signal processing, and a method for estimation refinement. The algorithms are then presented grouped by the domain they operate in. Chapter 3 describes the time domain algorithms RAPT and YIN. Then Chapter 4 describes the frequency domain algorithms TAPS and PEFAC. The composed system, which adds a combining algorithm evaluating the candidates of other pitch trackers, is presented in Chapter 5. The estimation quality and performance of each algorithm and the combining algorithm are then evaluated in Chapter 6.

Part II focuses on the visualization of C code, applying approaches on pitch tracking algorithms presented in Part I. For comparability most variants of the visualization are presented using the code example of RAPT. Chapter 7 presents three diagram types as a basis for the visualization components. Then Chapter 8 and Chapter 9 divide the visualization task into holistic and partial representations, respectively.

Finally Chapter 10 concludes on both parts with a summary and an outlook towards future work.

Part I

Real-Time Pitch Tracking

Chapter 2

Fundamentals for Pitch Tracking

Before pitch can be estimated some basics that hold for most of the following algorithms are described in this chapter.

Pitch is defined as the "auditory percept of tone" [Tal95]. Most pitch trackers estimate the Fundamental Frequency (F0), a local property of the speech signal, instead of the exact pitch since they are similar most of the time. Despite this difference, the terms pitch and F0 will be used as synonyms. Section 2.1 will go into more detail on human speech production.

Pitch tracking algorithms process the speech wave as a digital signal. Therefore, some knowledge about digital signal processing is provided by Section 2.2. A part thereof describes the representation of a continuous signal—such as the speech wave—in a suitable form in Subsection 2.2.1. Obtaining the frequency spectrum, which is a prerequisite for frequency domain algorithms, is touched upon in Subsection 2.2.2. Subsection 2.2.3 defines some of the correlation functions that are commonly used in the time domain.

Pitch estimations are to some extend limited in precision due to digital signal processing constraints. Often parabolic interpolation can be used to refine the estimation. Section 2.3 gives a short theoretical introduction to polynomial interpolation from which a formula for estimation refinement using parabolic interpolation is derived.

2.1 Speech Production

In normal speech there are many different ways of producing a sound. The exact ways that occur in normal speech is vary from language to language. Although different sounds are produced in independent regions of the *vocal tract*, we are capable of precisely adjusting muscles in a way that up to ten to twelve [Lev99] different *phonemes* can be produced within a second during normal speech. In spite of this speed of pronunciation, a very high precision of approximately one occurring error in 900 words is achieved [GSB+81], be it a wrong phoneme or even an unintended word. This section intends to give a rough idea of how sounds in normal speech are produced.

Several models of speech production exist. Most theories are based on the following structure. Before speech is produced, the message to express has to be chosen. Its semantics are mapped on conceptually fitting words. The context of those words and their syntactical properties, such as grammatical order, decide the time of utterance. Phonological properties are retrieved to prepare prosody. Before a sound is produced, articulatory gestures are planned for each phoneme [Lev99].

The sound production is located in the vocal tract, which is delimited by the vocal cords and the mouth opening [RS+07]. The biological composition of the vocal tract is shown in Figure 2.1. Sounds are produced either by pulses generated from the opening by the vocal cords (*glottis*) and manipulated by the vocal tract or by constrictions in the vocal tract that created turbulent airflow when air is forced trough [RS+07]. For the first case the sound is called *voiced* and for the second case it is called *unvoiced*. Since F0 is a property of periodic signals, F0 estimation is only reasonable for speech induced by glottal pulses. A simple visualization of the two signal stages for voiced speech is shown in Figure 2.2. In

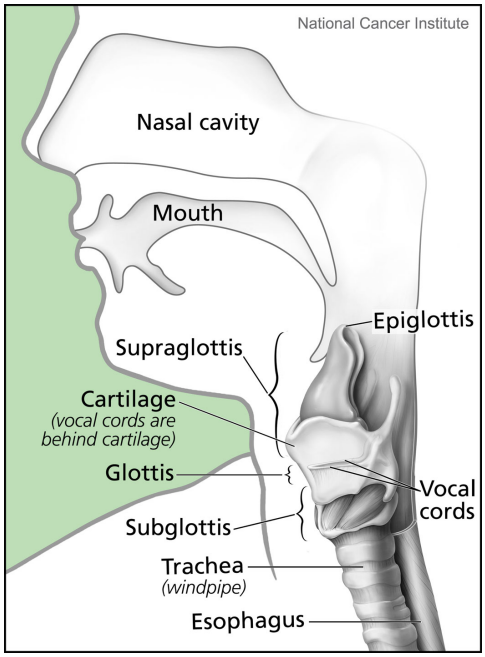

Figure 2.1. Larynx and nearby structures [Hoo03]

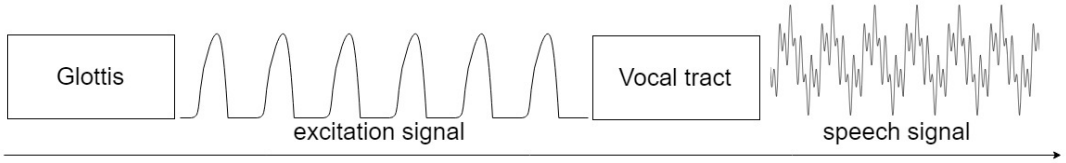

Figure 2.2. A simple model of speech production from the glottis to the final speech wave.

the strict sense, the voiced-unvoiced-domain is not as restricted to these two extremes as it may seem now, but is much rather a continuous spectrum. However, the simplification to the voiced or unvoiced categories offers considerable utility [Tal95]. An example is the decision of whether pitch estimation is reasonable in the first place. This is an considerably more difficult question for a continuous voiced spectrum.

The common theory of voice generation is called the *myoelastic/aerodynamic theory of phonation*. Voiced sounds are produced by the oscillation of the vocal cords. The theory states that the oscillation is a "forced oscillation of elastic material" [RS+07]. In contrast to this, the now disproven theory of Husson [Hus57] stated that each vibration cycle would be individually controlled by a neurophysiological process.

When muscles decrease the volume of the lungs, the pressure inside is lower then the pressure outside of the body. Pressure compensation generates an airflow from the lungs through the vocal tract and the mouth or nose outside of the body.

When the airflow passes the vocal tract, it is then manipulated to produce the versatile palette of different sounds. Some examples are given here. For sounds like /N/ or /M/, the mouth is blocked and the airflow is redirected through the nose creating nasal sounds [RS+07]. When the lips and the passage through the nose are closed, pressure builds up and is abruptly released creating a /P/ or

/B/ sound. Voiced sound production with different shapes of the opened mouth and angles of the jaw produce sounds such as /A/, /O/, /U/, and /I/. Fizzy sounds such as /S/, /SCH/, or /Z/ are produced by a constriction between the tongue and the palate. The differences in the sound are generated by minor adjustments of the tongue position and shape.

The glottal excitation has the most important role in the production of voiced sounds. Vibration of vocal cords are forced by the airflow through the glottis [Hes12]. Each vibration cycle is structured in two phases. In the *closed-glottis* phase no air can pass the vocal cords. Subglottal pressure builds up by pressure compensation from the lungs. This pressure builds up a force to the vocal cords to move apart. As the glottis opens the (aptly named) *open-glottis* phase begins. With this opened passage pressure can compensate upwards the vocal tract. The constriction by the glottis creates a particle velocity which leads to closing vocal cords according to the Bernoulli force [VZD57]. During the open-glottis phase passing airflow excites a pulse of sound pressure. This pulse is manipulated through the vocal tract to form an outgoing speech wave. The open-glottis and closed-glottis phases are rapidly repeated during voiced speech. The time between two subsequent pulses is called the Fundamental Period (T0) and the reciprocal of T0 defines the Fundamental Frequency (F0). For normal speech, F0 is mostly between 50 Hz and 500 Hz [Tal95].

An important property of voiced speech for pitch estimation is the occurrence of harmonics of the F0. The n-th harmonic of the F0 is at frequency n F0 Hz. Therefore, the first harmonic refers to the fundamental frequency, the second to F0 doubled and so on. Harmonics can be either produced by the collision of the vocal folds or by energy that is reflected from the vocal tract and fed back to the glottis, which then affects the glottal flow [Tit09]. Since higher harmonics are a product of lower harmonics, the amplitude of harmonics decreases with increasing frequency.

The *quality* of voiced sounds can vary. Although the voice quality range is continuous, some intervals share the same characteristics. *Breathy voice speech* is an alternation of *normal speech* with either less pressure from the lungs or a widening of the glottal opening [Tal95]. The glottis then closes more softly or does not close at all. This causes breathy voice speech to have a more aperiodic component and to be dominated by the first few harmonics [FL88]. In contrast during *pressed voice speech* the vocal cords are more forcefully pressed together. This extends the closed-glottis phase and the folds open more abruptly [Tal95]. Therefore, pressed voice speech tends to have lower amplitude for the first harmonics and a higher amplitude for higher harmonics. *Creaky voice speech*, which is the common voiced form for screaming, has a comparatively low airflow with respect to the subglottal pressure. Higher harmonics have, again, a higher amplitude comparing with the amplitude of F0. These differences in quality is one difficulty in pitch estimation. This especially holds for the aperiodic components of breathy voice speech. Also, extremely breathy voice speech can be quite similar to unvoiced speech in a continuous voiced state domain.

2.2 Digital Signal Processing

This section aims to establish the fundamentals of digital signal processing that are suited for pitch tracking. Section 2.2.1 explains how a continuous signal can be represented and gives an overview on terminology. For frequency domain algorithms it is necessary to obtain the *frequency spectrum*. This is covered in Section 2.2.2. Pitch trackers are often based on a function that measures the correlation of signals. Section 2.2.3 thus helpfully explains some of these functions that will be used in presented algorithms.

2.2.1 Signal Representation and Terminology

The speech wave is a continuous function of *sound pressure*. Processing a *continuous signal* in the digital domain is not possible due to the following technical limitations. Firstly there is no technology to record and forward such a continuous signal to a digital signal processing system. Second, even if we were able to hand a continuous signal over to the system, it would not be able to compute an estimation continuously. Every interval of time contains infinitely many points, but during a finite amount of time only a finite amount of values can be calculated. Since the system is required to terminate, it only has a finite amount of time to calculate the estimations and thus cannot calculate the estimation continuously.

The same problem of representing continuous behaviour appears with films. A film that seems to show continuously moving objects is indistinguishable from showing many pictures in very quick succession. This can also be done for speech signal processing. If new information is produced fast enough, the difference to the continuous signal will stay unnoticed. There is still a difference, but it does not have a serious impact on the outcome.

Extracting discrete data points from a continuous signal is called *sampling*. The *sample rate* is a quantity for how many *samples* there are for each time interval of a specific size, often one second. For a continuous signal $s(t)$ with $t \in \mathbb{R}$ and a sample rate f_s, the signal $s\left(n/f_s\right)$ with $n \in \mathbb{N}$ is discrete. Often the discrete signal $s\left(n/f_s\right)$ is written as a sequence s_n.

The rate in which new samples are acquired from a signal is often much higher than the rate in which a real time audio system wants to process samples. Therefore, multiple samples are grouped and processed together. The group size is then called the *frameshift*. For every estimation cycle of the algorithm then frameshift many input values are handed over to the algorithm.

Since this is often not enough information for estimating F0, algorithms keep track of past input. The number of samples with which an algorithm, then, computes the estimation is called the *analysis window length*.

The sample rate has a lower bound depending on the maximum frequency that the signal contains. It is given by the sampling theorem which was proposed by Nyquist in 1928 [Nyq28] and proven by Shannon in 1949 [Sha49].

Theorem 2.1 (Sampling Theorem [Sha49])**.** *If a function $f(t)$ contains no frequencies higher than f_{max} Hertz, it is completely determined by giving its ordinates at a series of points spaced $\frac{1}{2f_{max}}$ seconds apart.*

The other way around for a given sample rate f_s, functions containing a frequency of up to the *Nyquist frequency* $\frac{f_s}{2}$ can be completely determined [Nyq28]. Not following this can lead to *aliasing*. Figure 2.3 shows this effect. For a sampling rate of $f_s = 8$ Hz the samples for a sine with frequency $f_1 = 2$ Hz and a sine with frequency $f_2 = 10$ Hz are equal. This is because f_2 is greater than the Nyquist frequency for that sample rate $\frac{f_s}{2} = 4$ Hz. Therefore, the signal can not be determined completely.

As explained in Section 2.1, for pitch tracking requires determining frequencies up to 500 Hz. The sample rate should be at least $f_s = 1000$ Hz to estimate the pitch correctly. Since human voice also contains higher harmonical frequencies, the sample rate should be chosen higher.

2.2.2 Obtaining the Frequency Spectrum

For frequency domain algorithms it is a prerequisite that information about the frequencies a signal is composed of can be obtained from the speech signal. To obtain the frequencies and their amplitudes from a discrete time signal, the Discrete Fourier Transform (DFT) can be used [LG88].

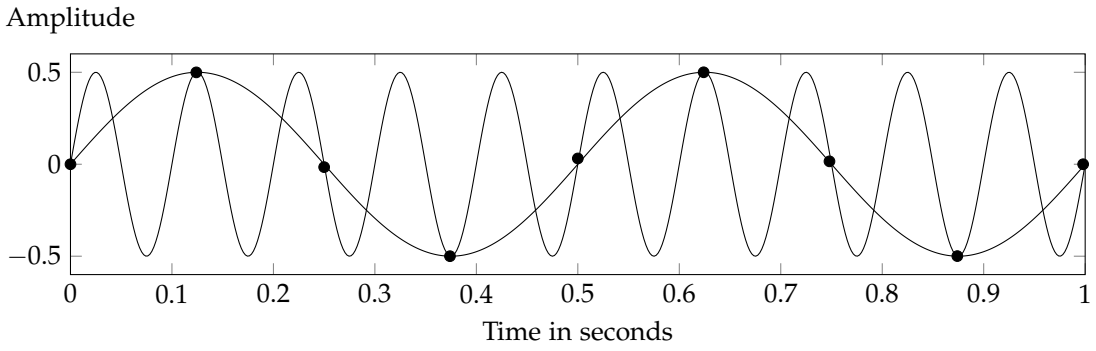

Figure 2.3. The alias effect. Different signals have the same value at data points with a too low sample rate.

For a sampled input speech signal $s_0, \ldots, s_{N-1}$ it is defined as [OS75]

$$X_m = \sum_{k=0}^{N-1} s_k e^{-\frac{2\pi i}{N} mk}.$$

There X_m represents the frequency $\frac{mf_s}{N}$ for the sample rate f_s and $m \in \{0, \ldots, N-1\}$. These frequencies and their amplitudes are, just like the input signal, discrete and are referred to as frequency bins.

In general a formula e^{ix}, with $x \in \mathbb{R}$, is a short term for expressing a circle. The term $e^{2\pi ix}$ then needs a change of x by exactly one to complete a circle. With $e^{2\pi ix\frac{m}{N}}$, then m circles are completed and within N samples, which defines the frequency of completed circles. The sum is a measurement of how well the frequency of the circles correlate with the input signal, thus this indicates whether the frequency occurs in the input signal.

The computational complexity of the DFT is in $O(N^2)$. Since $e^{-\frac{2\pi i}{N} mk}$ is periodic and symmetric, the computational cost can be reduced to $O(N \log(N))$ by the Fast Fourier Transform (FFT) using a divide and conquer strategy [OS75].

2.2.3 Correlation Functions

Many pitch estimation algorithms are based on a correlation function. In general, such a function somehow measures the similarity between signals. Often this is used with the speech signal and a delayed version of it. Some correlation functions are introduced in this subsection.

Before these functions can be defined, the delayed version needs to be formalized. The amount of delay is called *lag*. The value of a signal s at time k delayed by a lag τ is simply $s_{k+\tau}$. If a periodic signal s is delayed by its period τ_0, it has the same value $s_k = s_{k+\tau_0}$ again for every $k \in \mathbb{N}$ that the signal is defined for.

Figure 2.4 shows a signal that is delayed by different lags. A delay of 0.05 s is displaced by one fourth of the period compared to the non delayed signal. The signals do not completely correlate, but there are time intervals where the sign is equal or the values are equal at a specific point in time. The delay of 0.1 s is displaces the signal by half of the period. It does pretty much the opposite of the undelayed signal. It does not correlate. In this example, the lags of 0 s, 0.2 s, and 0.4 s correlate best. This is caused by the periodicity of the wave function. It has a fundamental frequency of F0= 5 Hz or equally a fundamental period of T0 = 0.2 s and, therefore, repeats five times per second. Often the repeated correlation for more than one period displacement causes octave errors in algorithms that

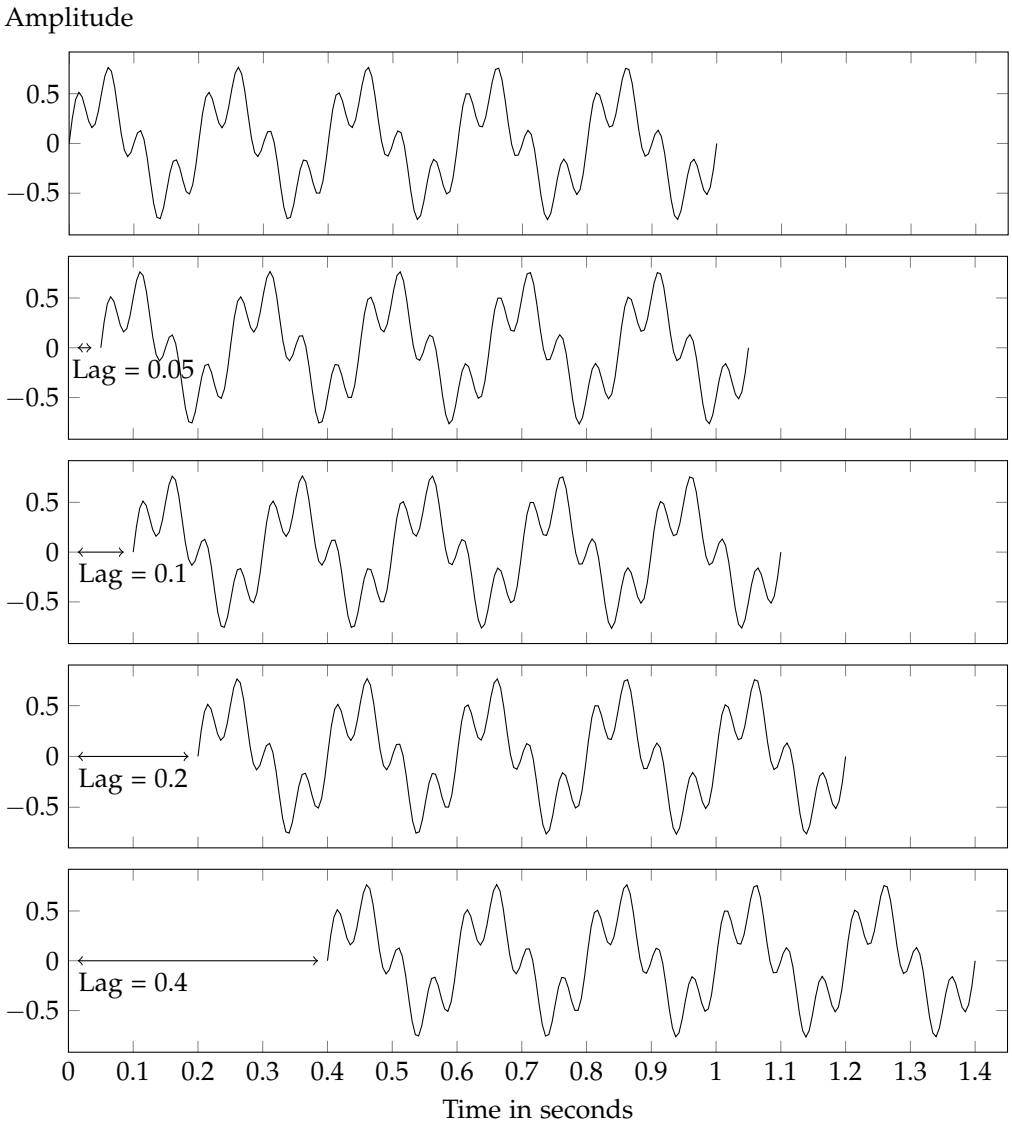

Figure 2.4. A signal with F0 = 5 Hz and one harmonic at frequency 15 Hz.

rely on a correlation function. Values of these are for some measurement optimal for lags $\tau = n\text{T0} = \frac{n}{\text{F0}}$, with $n \in \mathbb{N}$. Therefore, secondary candidates produced by an algorithm that uses such a function are often subharmonics. This will be important again in Section 5.2.

Let $n \in \mathbb{N}$ be the length of the speech signal and for $k \in \{0, \ldots, n-1\}$ let s_k be the speech signal. Let $K < n$ be the summation size of the correlation function and τ the lag.

Average Magnitude Difference Function (AMDF) [Moo74]

The AMDF calculates the difference as a measure for aberration between the input signal and the input signal delayed by a lag.

The AMDF is defined as follows

$$AMDF : \{0, \ldots, n-K\} \to \mathbb{R}_{\geqslant 0},\ AMDF(\tau) = \sum_{k=0}^{K-1} |s_k - s_{k+\tau}|.$$

If the aberration is small then the signals correlate well. Therefore, minima of this function might refer to T0 and multiples of it. These minima are zero for periodic signals. Since the speech wave form is not perfectly periodic but rather pseudo periodic these minima are in reality not exactly zero since it is unlikely that subsequent periods match perfectly. Also local minima can be produced from the harmonics of F0 and the global minimum is probably also not T0 but a multiple of it.

Autocorrelation Function (ACF) [Rab77]

The ACF is the dot product of the signal and its delayed version. The summation size is smaller for bigger values of τ to favour T0 over multiples of it. For periodic functions the local maxima refer to T0 and multiples of it.

The ACF is defined as follows

$$ACF : \{0, \ldots, n-1\} \to \mathbb{R},\ ACF(\tau) = \sum_{k=0}^{K-\tau-1} s_k s_{k+\tau}.$$

Although the ACF performs well and is robust to noise, it needs a relatively big analysis window size to compute reasonable values even for longer lags. Therefore, fast changes of F0 might pass unnoticed by the ACF.

Cross Correlation Function (CCF) [Tal95]

The CCF only differs from the ACF in its summation size, which is independent from the lag. Similar to the AMDF it is possible that the global maximum does not refer to T0 but to a multiple of it instead.

The CCF is defined as follows

$$CCF : \{0, \ldots, n-K\} \to \mathbb{R},\ CCF(\tau) = \sum_{k=0}^{K-1} s_k s_{k+\tau}.$$

Since the summation size of the CCF is the same for every lag it does not have the disadvantages mentioned for the ACF. However, the ACF tends to decrease in value for a decreasing lag. Every multiple of T0 correlates well with the original signal. The decrease in value favours T0 from multiples of it. This is an advantage that the CCF does not have.

Normalized Cross Correlation Function (NCCF) [Ata68]

While the CCF has no theoretical bounds, the NCCF normalises the value by its energy. The advantage of $NCCF(\tau) \in [-1, 1]$ is that the value serves as a quantitative, signal-independent indicator of correlation.

The NCCF is defined as follows

$$NCCF : \{0, \ldots, n-K\} \to [-1, 1],\ NCCF(\tau) = \sum_{k=0}^{K-1} \frac{s_k s_{k+\tau}}{\sqrt{e_0 e_k}}$$

and the energy value e_i is defined as

$$e_i = \sum_{k=i}^{i+K-1} s_k^2.$$

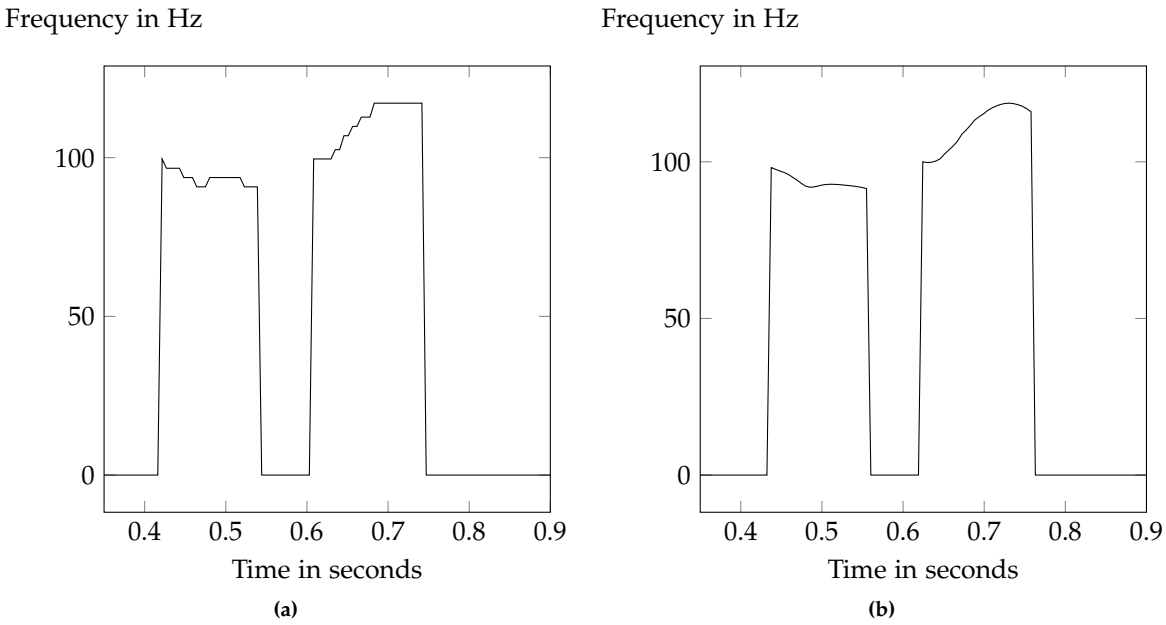

Figure 2.5. Estimations of PEFAC (a) without and (b) with using parabolic interpolation.

The normalization of the CCF helps to deal with several high values for every multiple of T0, which would otherwise make the decision of the estimation choice more difficult.

2.3 Estimation Refinement

The estimation produced by a pitch-tracking algorithm depends on the information it is based on. Often this is the index of the maximum or minimum of a function depending on the input. As stated above, these inputs are discrete. Therefore, the index of the turning point for an evaluating function that depends on discrete data points can only take discrete values, too. Subfigure 2.5a shows estimations made by PEFAC. F0 can take any value in the range between 50 Hz and 500 Hz and changes of the glottal period are either small or exactly one octave. Although the F0 of a human voice is, therefore, to some degree continuous, the estimations made by PEFAC only take discrete values. *Parabolic interpolation* can be used to refine the estimation. As the name suggests, a parabola approximates the behaviour of the function with continuous input in the vicinity of the turning point of the function with discrete input. The turning point of the parabola is then taken as the refined estimation. This refinement might take any value on the continuous frequency spectrum and is expected to be a better approximation of the F0. Subfigure 2.5b shows estimations of PEFAC combined with parabolic interpolation for the same input as for Subfigure 2.5a. The estimations with parabolic interpolation are continuous and state the course of the perceived pitch more precisely. The following will give a short theoretical introduction to *polynomial interpolation*, of which parabolic interpolation is a special case, before giving a formula for the refined estimation.

Let $\Pi_m := \text{span}\{x \to \sum_{k=0}^{m} a_k x^k : a_k \in \mathbb{R} \wedge k \in 0, \ldots, m\}$ for $m \in \mathbb{N}$ be the *vector space* of polynomial functions of degree less or equal m. Then Π_m has dimension $m + 1$, meaning every element of Π_m is uniquely defined by a basis and a set of $m + 1$ known values.

Theorem 2.2. *For mutually distinct data points $x_0, \dots, x_m \in \mathbb{R}$ and values $f_0, \dots, f_m \in \mathbb{R}$ there exists exactly one polynomial p with degree less or equal to m and $p(x_i) = f_i$ for every $i \in \{0, \dots, m\}$.*

This is proven by many introductory books on numerical mathematics (see for example Schwarz and Köckler [SK13]). The vector space of interest here is Π_2 for linear or parabolic functions. For identifying a parabola it is, therefore, sufficient to know the values of three mutually exclusive data points.

At this point we know that there exists exactly one parabola that meets three given data points with according values. Since parabolic functions have exactly one turning point, also the turning point for any parabolic function that meets the condition is uniquely defined as well. Otherwise it would not be clear which turning point estimates F0.

Since the purpose of parabolic interpolation is to obtain a more precise estimation, the quality of the interpolation is of interest.

Theorem 2.3. *For $a, b \in \mathbb{R}$ and $m \in \mathbb{N}$ let $f : [a, b] \to \mathbb{R}$ be a function of which the first m derivatives exist and are continuous. For $x_0, \dots, x_m \in [a, b]$ mutually distinct and $y_i := f(x_i)$ for every $i \in \{0, \dots, m\}$ let $p : [a, b] \to \mathbb{R}$ be the polynomial interpolation.*
Then for every $\tilde{x} \in [a, b]$ there exists $\xi \in (a, b)$ with

$$f(\tilde{x}) - p(\tilde{x}) = \frac{f^{(m+1)}(\xi)}{(m+1)!} \prod_{i=0}^{m} (\tilde{x} - x_i). \tag{2.1}$$

This again is fundamental fact of numerical mathematics and many proofs exist (again, see for example Schwarz and Köckler [SK13]).

Even though the functions for the discrete estimations are not differentiable since they are defined on *discrete values* instead of an *open set*, there exist reasonable extensions for those functions on continuous inputs and, since the function is fixed, so is the value of the derivative. From Equation (2.1) it follows that the error of interpolation also depends on the degree of the interpolation polynomial and, more importantly, of the distance between the data points and the evaluated point. The degree is fixed to two using parabolic interpolation, so this factor can not be improved to lower the error. However, the distance of the turning point to the data points can be minimized. To obtain the best possible approximation of the turning point, the index of the minimum or maximum of the discrete function and its two neighbours are chosen for the three data points. From the definition of the parabolic interpolation it is clear that there is no approximation error if the turning point is exactly one of the three data points. From the choice of data points it also follows that the approximation of the turning point is somewhere between the data points, which is especially reasonable since parabolic interpolation is only used as a refinement and the minimum or maximum of the discrete function should already be be in the vicinity of the F0.

For the calculation of the turning point, first the parabola for three data points and according values is derived from the *Lagrange polynomials* and then a formula for the turning point is derived from that parabola.

Definition 2.4 (Lagrange polynomials)**.** For mutually exclusive data points $x_0, \dots, x_m \in \mathbb{R}$ the Lagrange polynomial $l_i \in \Pi_m$ is defined as

$$l_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^{m} \frac{x - x_k}{x_i - x_k}$$

for $x \in \mathbb{R}$ and $i \in \{0, \dots, m\}$.

For $i \in \{0, \dots, m\}$ $l_i(x_i) = 1$, and for any data point x_j, $i \neq j$ and $j \in \{0, \dots, m\}$, it holds that $l_i(x_j) = 0$ yields. Let $f_0, \dots f_m$ be the values of the function at the given data points. Then it follows

that

$$p_m(x) = \sum_{i=0}^{m} f_i l_i(x) \tag{2.2}$$

fulfils the equations $p(x_i) = f_i$ for $i \in \{0, \dots, m\}$.

Theorem 2.5. *Let $x_0, x_1, x_2, f_0, f_1, f_2 \in \mathbb{R}$. Then for*

$$p(x) := \sum_{i=0}^{2} f_i \prod_{\substack{k=0 \\ k \neq i}}^{2} \frac{x - x_k}{x_i - x_k}$$

it is true that $p(x_0) = f_0$, $p(x_1) = f_1$, $p(x_2) = f_2$ and

$$x_p := \frac{f_0(x_1^2 - x_2^2) + f_1(x_2^2 - x_0^2) + f_2(x_0^2 - x_1^2)}{2(f_0(x_1 - x_2) + f_1(x_2 - x_0) + f_2(x_0 - x_1))}$$

is the turning point of p.

Proof. For $m = 2$, data points $x_0, x_1, x_2 \in \mathbb{R}$ and values $f_0, f_1, f_2 \in \mathbb{R}$ then p as defined in the theorem is equal to p_m in (2.2). The equalities are fulfilled with the same reasoning.

The turning point can be obtained by setting the derivative to zero. The derivative of the function is

$$p'(x) = f_0 \frac{2x - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)} + f_1 \frac{2x - x_2 - x_0}{(x_1 - x_2)(x_1 - x_0)} + f_2 \frac{2x - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)}. \tag{2.3}$$

The turning point of this functions is then obtained by

$$\begin{aligned}
0 &= p'(x) \\
&= f_0 \frac{2x - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)} + f_1 \frac{2x - x_2 - x_0}{(x_1 - x_2)(x_1 - x_0)} + f_2 \frac{2x - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)} \\
\Leftrightarrow 0 &= f_0(2x - x_1 - x_2)(x_1 - x - 2) + f_1(2x - x_2 - x_0)(x_2 - x_0) + f_2(2x - x_0 - x_1)(x_0 - x_1) \\
\Leftrightarrow x &= \frac{f_0(x_1^2 - x_2^2) + f_1(x_2^2 - x_0^2) + f_2(x_0^2 - x_1^2)}{2(f_0(x_1 - x_2) + f_1(x_2 - x_0) + f_2(x_0 - x_1))}.
\end{aligned} \tag{2.4}$$

□

For the use case of pitch tracking the data points for the parabolic interpolation will often differ exactly by one. Therefore, the equation for obtaining the turning point can be simplified to

$$x_p := \frac{4f_1x_1 - f_0(2x_1 + 1) - f_2(2x_1 - 1)}{2(2f_1 - f_0 - f_2)}.$$

Chapter 3

Time Domain Algorithms

As mentioned in 2.2 the inputs for a time domain pitch tracking algorithm are frameshift samples of the speech signal, which is not enough information to estimate a pitch. On the other hand, keeping track of the complete input speech is not a good idea either since the real-time system can run for an arbitrary amount of time and, since pitch is a time dependant property, most of the old information is not helpful. Therefore, a buffer of fixed size is needed to keep track of the newest input values. A *ring buffer* can manage the input with few instructions. New input is saved at the position behind the most recent input. Hereby the position behind the last buffer position is the first buffer position again, which motivates the buffer name. Therefore, one must keep track of the actual position for new input. However, during buffer management and buffer access from the algorithm the modulo operator, which has a relatively high computational cost, is needed quite often. To reduce overall computation time for buffer interactions the buffer is instead organized as follows. For every input cycle the values inside of the buffer are shifted to a frameshift higher position, thus dropping the oldest values. Then the new input can be written to the first frameshift positions of the buffer. This needs more operations for buffer management, but for buffer usage the i-th index now always refers to the i-th oldest input sample, which reduces the access time of input values for the algorithm.

This chapter describes two very common time domain pitch tracking algorithms, RAPT and YIN, in Sections 3.1 and 3.2 respectively. In both sections first the general algorithm is described followed by my modifications.

3.1 Robust Algorithm for Pitch Tracking (RAPT)

This section describes the RAPT algorithm as presented by Talkin [Tal95]. As the algorithm's name might suggest, RAPT is all about the robustness of the estimation. Although a low computational complexity is not the main focus, some optimizations that do not influence the estimation quality are included.

3.1.1 The Algorithm

In a nutshell, the algorithm is composed of the NCCF (as defined in Subsection 2.2.3) and a post processing stage. The NCCF is calculated in two stages for computational cost reduction. In the post processing stage, dynamic programming is used to select the best candidate.

As stated in Section 2.1 the F0 is between 50 Hz and 500 Hz. The pitch candidates are those lags for which the NCCF has a value near to one. Therefore, the NCCF has to be calculated for lags from $1/500$ s $= 2$ ms to $1/55$ s $= 20$ ms. The number of input samples during this interval determines the number of lags the NCCF is calculated for and is determined by the sample rate f_s. Each calculation of the NCCF has a summation size of K. The size K also depends on the sample rate f_s, since the summation size must include samples of at least one glottal period. Therefore, the computation time depends on f_s^2.

The sampling theorem states that for a periodic signal containing only frequencies up to a maximum frequency $f_{\max}$, a sampling rate of $2f_{\max}$ is sufficient. Since the voiced speech wave is quasi-periodic rather than exactly periodic and also contains higher harmonics, the estimation's precision increases with the sample rate.

A compromise between computational cost and precision of the estimation is achieved by the two-pass NCCF. First the NCCF is computed for a downsampled version of the signal. The lower sample rate is defined by Talkin as

$$f_{\text{ds}} := \frac{f_s}{\text{round}\left(\frac{f_s}{4f_{\max}}\right)} > \frac{1}{2f_{\max}}.$$

This sample rate is significantly lower than a sample rate for precise estimations had to be, but since it satisfies the sampling theorem for the purposes of pitch estimation, F0 will still be in the vicinity of one of the local maxima of this function.

To refine the estimation, the second step is to compute the NCCF for lags in the vicinity of those maxima. For a vicinity size of K_{vicinity} and a number of considered maxima K_{maxima}, the computational cost depends on $f_{\text{ds}}^2 + K_{\text{maxima}}K_{\text{vicinity}}f_s$. Since f_{ds} and $K_{\text{maxima}}K_{\text{vicinity}}$ are significantly smaller than f_s, the two-pass NCCF has a lower computational cost than the direct computation of the NCCF on the original sample rate f_s.

The lowest lag with an NCCF value greater than a threshold is then taken as the T0 estimation. One does not simply take the highest peak since for higher frequencies, also 2T0, 3T0, and even 4T0 can be in the range of calculated lags and these possibly have a higher NCCF value than the T0.

Figure 3.1 shows the NCCF functions for high sample rate f_s, the low sample rate f_{ds} and the high sample rate in the vicinity of maxima of the low sample rate NCCF. Maxima of the high sample rate version are also maxima in the low sample rate version. The highest maxima above an absolute threshold of the downsampled version are refined with vicinity NCCF computation. The resulting highest maxima is approximately at the same position as the highest maxima of the NCCF directly computed on the higher sample rate. Figure 3.2 shows the same information but for a female speaker. The possibility of a multiple of the T0 having a higher NCCF value than the T0 is represented by the second and fourth maxima. They correspond to T0 and 2T0, respectively, and the fourth local maximum has a slightly higher value than the second. For lags smaller than 2 ms the values of the NCCF are not calculated since they correspond to a frequency higher than the maximum frequency for voiced speech.

Figure 3.3 shows similar information for an unvoiced sound produced by a male speaker. The values of the NCCF are significantly lower than 1 for unvoiced speech due to missing periodicity. While in the voiced example in Figure 3.2 the global maximum has an NCCF value > 0.9, the global maximum in the unvoiced example shown in Figure 3.3 does not exceed 0.4. This can be used for voiced state estimation by using a threshold.

Based on the local NCCF calculation and contextual evidence, such as past estimations, dynamic programming selects the best T0 candidate to obtain a pitch estimation. Hereby, backtracking on up to the full set of known data is done. This goes beyond the acceptable delay scope for real-time pitch tracking and is therefore not further described here. The interested reader is referred to the original paper by Talkin [Tal95].

After the best lag is chosen by dynamic programming, the estimation is refined by parabolic interpolation. This is reasonable, since the selected lag most likely corresponds to a local maximum of the NCCF.

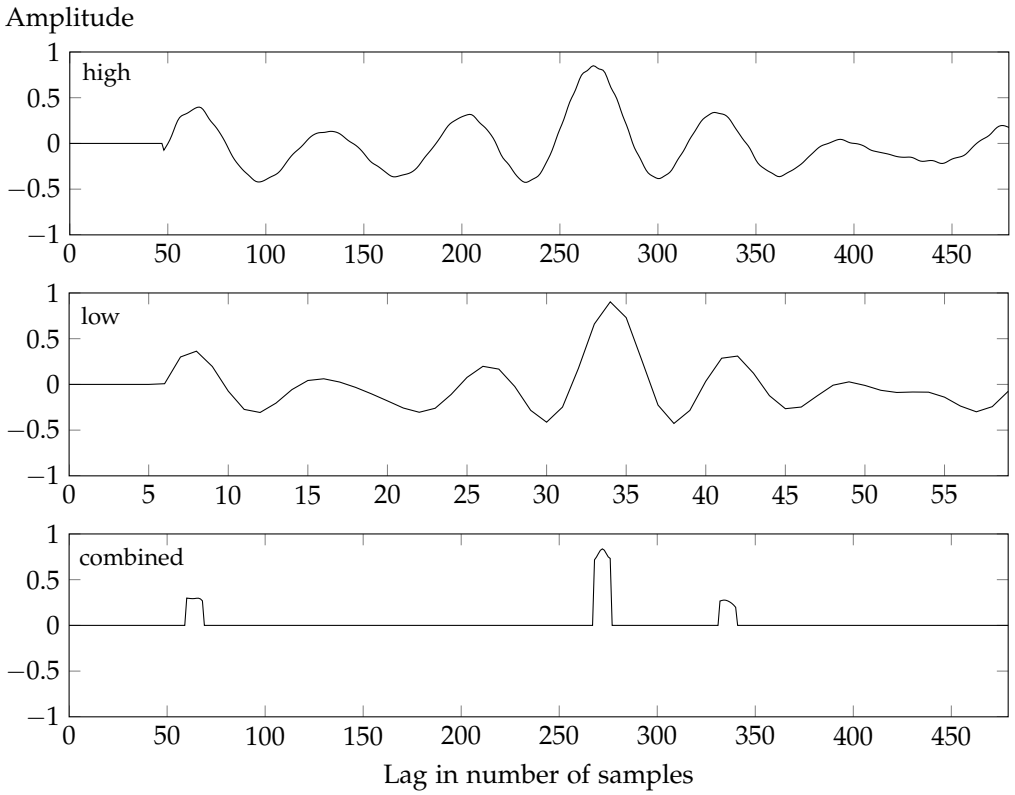

Figure 3.1. NCCF functions for the voiced part of *year* pronounced by a male speaker. From top to bottom the NCCF is calculated with the high sample rate, the low sample rate and the high sample rate in the vicinity of maxima of the low sample rate NCCF.

3.1.2 Modifications

Besides the fact that the post processing of RAPT is not implemented and, therefore, a more suitable name for the implementation would rather be *two-pass NCCF* than *RAPT*, the implementation only differs by the calculation of energy values and adds a threshold-based voiced decision.

The energy value for the NCCF could, if implemented naively, double the computational cost of the NCCF compared to the CCF due to the additional energy calculations. The solution explained by Talkin is to incrementally modify the last energy value to obtain the next value. This can be described as follows with the terminology from Subsection 2.2.3:

$$e_{i+1} = \sum_{k=i+1}^{i+K} s_k^2 = \left(\sum_{k=i}^{i+K-1} s_k^2 \right) + s_{i+K}^2 - s_i^2 = e_i + s_{i+K}^2 - s_i^2.$$

To avoid high accumulated errors due to the digital representation of numbers, the usage of double precision floating point numbers is necessary for the incremental calculation.

Most real-time audio systems work with single precision instead of double precision. This also applies to the framework this algorithm is implemented in. Since the accumulated error on single precision floating point numbers could influence the result, another approach is taken. Since the same energy values are needed multiple times, they are calculated once and saved in an energy buffer,

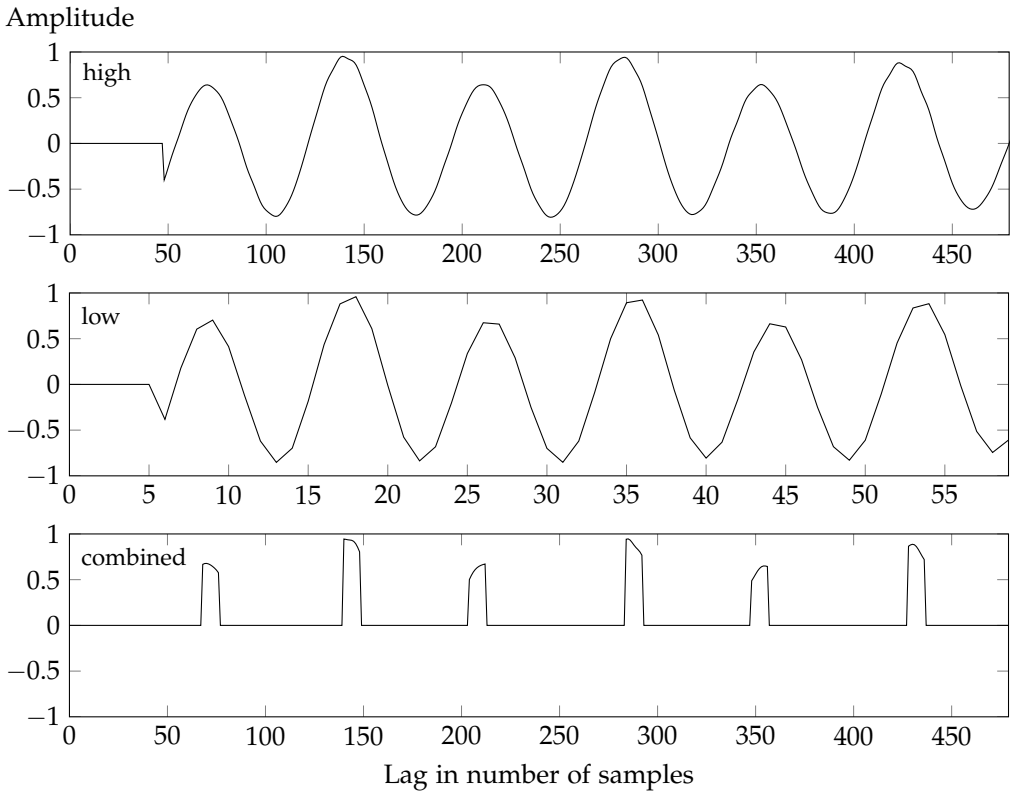

Figure 3.2. NCCF functions for the voiced part of *year* pronounced by a female speaker. From top to bottom the NCCF is calculated with the high sample rate, the low sample rate and the high sample rate in the vicinity of maxima of the low sample rate NCCF.

which is organized just like the input buffer as described at the beginning of this chapter.

The voiced decision in the algorithms is done during post processing. The implemented voiced decision is based on a threshold for the maximum NCCF value. If the correlation for a lag is high enough, then it is likely that it is voiced. Accordingly, if no lag has a good correlation then it is likely to be unvoiced.

3.2 YIN

This section describes the YIN algorithm as presented by Cheveigné and Kawahara [DK02]. The name YIN is derived from Yin and Yang from oriental philosophy. It is derived from the ACF and is related to the idea of the AMDF as mentioned in Subsection 2.2.3. Several independent steps then improve the results to obtain an even better estimation.

3.2.1 The Algorithm

The description by Cheveigné and Kawahara consists of six steps. The first step explains the ACF, which is replaced by the difference function described in step two. Therefore, only five of these steps actually contribute to the final estimation.

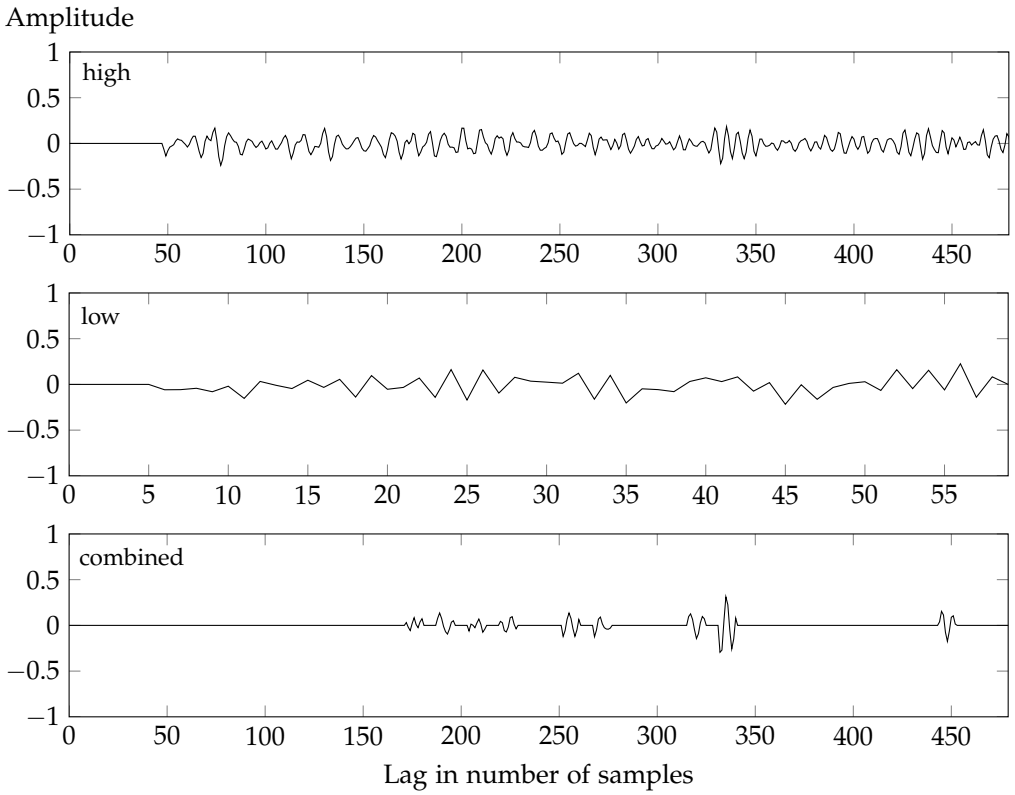

Figure 3.3. NCCF functions for the unvoiced sound /S/ of *greasy* pronounced by a male speaker. From top to bottom the NCCF is calculated with the high sample rate, the low sample rate and the high sample rate in the vicinity of maxima of the low sample rate NCCF.

The Squared Average Magnitude Difference Function (SAMDF)

For a periodic signal s_k with period t it holds that $s_k = s_{k+nt}$, which is equal to $s_k - s_{k+nt} = 0$, for every $k, n \in \mathbb{N}$. This is also true for the squared value $(s_k - s_{k+nt})^2 = 0$. For an analysis window size $W \in \mathbb{N}$ it yields

$$\sum_{k=0}^{W-1} (s_k - s_{k+nt})^2 = 0$$

for periodic signals. A voiced speech signal is at least quasi-periodic. Therefore, the Squared Average Magnitude Difference Function, defined by

$$\text{SAMDF} : \{0, \ldots, n-K-1\} \to \mathbb{R}_{\geqslant 0},\ \text{SAMDF}(\tau) = \sum_{k=0}^{K-1} (s_k - s_{k+\tau})^2$$

with the terminology from Subsection 2.2.3, takes on values close to zero for lags that are multiples of T0.

The SAMDF is connected to the ACF and CCF by $\text{SAMDF}(\tau) = \text{CCF}(0) + \text{CCF}(\tau) - 2\text{ACF}(\tau)$. Therefore, the results are to some extend similar to RAPT because RAPT is based on the NCCF, which is in turn based on the ACF.

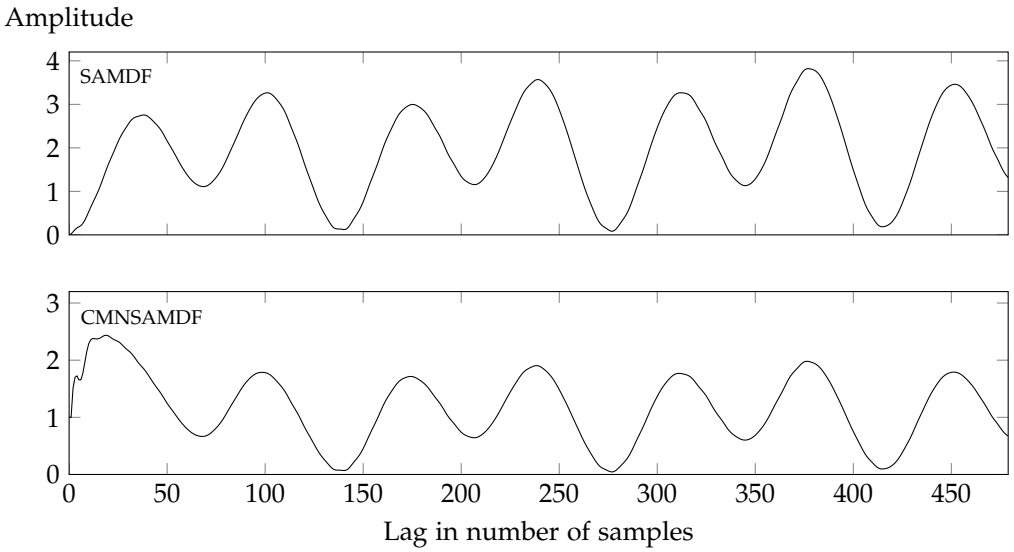

Figure 3.4. SAMDF and CMNSAMDF for the voiced part of *year* pronounced by a female speaker, shown in the upper and lower graphic, respectively.

The Cumulative Mean Normalized Squared Average Magnitude Difference Function (CMNSAMDF)

The SAMDF is close to zero for lags that are multiples of T0. Since the function is zero for a zero delay, taking the lowest valued lag is not a good choice since this might just be the zero lag or the first bin that is in T0 range.

The CMNSAMDF adds a normalization that is based on the mean of lower lag SAMDF values. It is defined as follows:

$$\text{CMNSAMDF} : \{0, \ldots, n-K-1\} \rightarrow \mathbb{R}_{\geqslant 0},\ \text{CMNSAMDF}(\tau) = \begin{cases} 1 & , \text{ if } \tau = 0 \\ \frac{\text{SAMDF}(\tau)}{\text{CMNF}(\tau)} & , \text{ otherwise} \end{cases}$$

with the Cumulative Mean Normalization Function (CMNF) defined as

$$\text{CMNF} : \{0, \ldots, n-K-1\} \rightarrow \mathbb{R}_{\geqslant 0},\ \text{CMNF}(\tau) = \frac{1}{\tau} \sum_{k=1}^{\tau} \text{SAMDF}(k).$$

Figure 3.4 shows the SAMDF and CMNSAMDF of a voiced speech segment. In this example the global minimum of the SAMDF is at zero lag and there are near-zero local minima at the lag bins 140, 277, and 417 corresponding to T0, 2T0, and 3T0 respectively. There are also local minima between those minima close to zero. These refer to harmonics of the F0. Also the minimum for 2T0 is smaller than the minimum for T0. The global minimum is zero at lag zero.

The CMNSAMDF shown in the bottom plot in Figure 3.4 has the global minimum for 2 T0, not at lag zero. Due to the normalization the values of the CMNSAMDF for lags greater than T0 are in $[0, 2]$. For lags before the first local maximum, the values are greater than or equal to one. Therefore, the problem of distinguishing between a minimum that is simply close to the zero lag and a lag that corresponds to a multiple of T0 is solved.

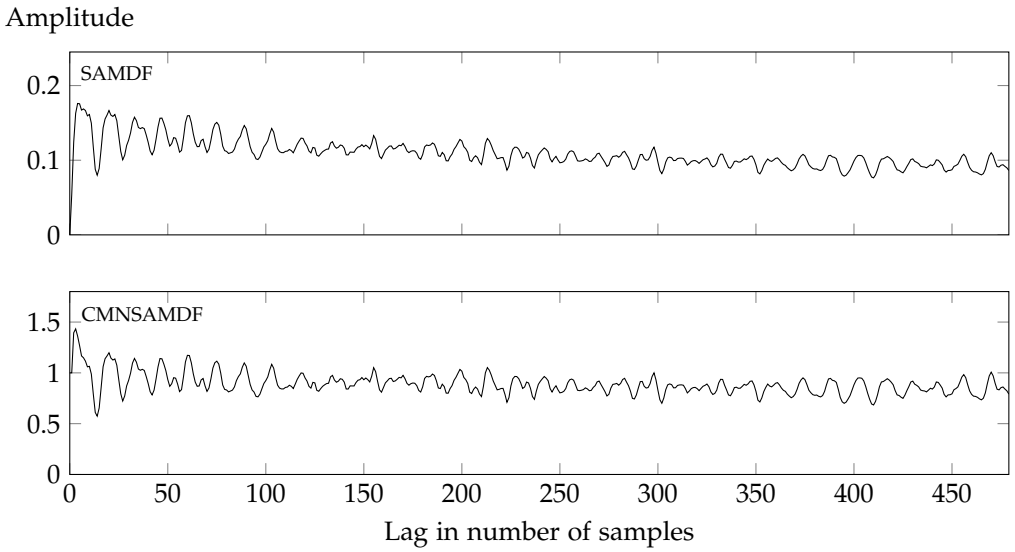

Figure 3.5. SAMDF and CMNSAMDF for the unvoiced /SH/ of *washwater* pronounced by a female speaker, shown in the upper and lower graphic, respectively.

Absolute Threshold

Usually the global minimum of the CMNSAMDF does not correspond to T0 as the example in Figure 3.4 shows. However, all of the lags close to zero correspond to multiples of T0. An absolute threshold is used to choose a first candidate. The first lag for which the CMNSAMDF is below that threshold is chosen as the candidate. This most likely is not a local minimum itself, but in the vicinity of the local minimum corresponding to T0.

For the example, any threshold less than or equal to 0.64 would be sufficient for estimating the lag corresponding to the right local minimum using the next step, since the lowest lag with a smaller value of the CMNSAMDF corresponds to the local minimum of T0. The authors suggest a threshold of 0.1, which is small enough that local minima referring to a harmonic does not reach this value, but large enough to still include the local minimum corresponding to T0. However, if there is no lag satisfying this threshold, then the global minimum is chosen to be the candidate.

Figure 3.5 shows the SAMDF and CMNSAMDF for an unvoiced speech segment pronounced by a female speaker. While the SAMDF values theoretically are not absolutely bounded, the CMNSAMDF values are all greater than 0.5 since the signal does not contain sufficient periodicity for the correlation function to result in a lower value. Hereby, a possible voicing state estimation can be derived.

Local Estimation Refinement

As mentioned before, the candidate obtained by the threshold does not necessarily have to be a local minimum for the CMNSAMDF. Imagine a threshold of 0.4 at Figure 3.4. Then the lag 129 would be chosen as the candidate. However, the local minimum is at lag 140. The refinement from the lowest lag fulfilling the threshold to the actual local minimum is done in this step.

To further improve the estimation quality, this steps takes time and lag neighbouring to the candidate, which was obtained by the threshold, into account. Time is considered by recalculating CMNSAMDF on an input buffer, which is slightly shifted in time to contain either some previous or

subsequent samples. The lowest CMNSAMDF value in the range of $\pm 20\%$ lag of the candidate and $\pm T_{\max}/2$, with $T_{\max} = 25$ ms, in time determines the refined candidate. For a sample rate $f_s = 48$ Hz, CMNSAMDF is recalculated on an input buffer shifted by up to $T_{\max} f_s = 1\,200$ samples to consider time in the local refinement.

Parabolic Interpolation

Since the lag always refers to a multiple of the sample rate f_s, also the estimated T0 is restricted to take a value n/f_s, or similar F0 to take a value f_s/n, with $n \in \mathbb{N}$.

Consider the local minima at lags 140 and 277 in Figure 3.4. While for lag 277 the minimum is significantly lower than its neighbours, for lag 140 and lag 141 the values are really close. In those cases it is likely that the actual T0 is between those discrete values given by the lag. Parabolic interpolation provides the opportunity to estimate T0 as any value in the interval $[50, 500]$ Hz, which is the range of human voice.

Therefore, parabolic interpolation is used to refine the estimation. Parabolic interpolation is applicable since the previously selected lag is a local minimum for the CMNSAMDF. Since lags are equally spaced, this is one case where the simplified formula from Subsection 2.3 can be used.

3.2.2 Modifications

With consideration towards the real-time implementation, the step of local estimation refinement does not minimize the CMNSAMDF values over a time interval but only searches the local minimum in the vicinity of the estimation candidate lag obtained from the threshold. The consideration of other time intervals has the following main disadvantage. First the computation of the CMNSAMDF in the vicinity of the candidate requires the computation of all SAMDF values for lags smaller than the candidate lag or in its vicinity. For *frames* in the past this is done, but the proposed evaluation evaluates this within an interval requiring a calculation of the CMNSAMDF for every sample. This leads to a computational complexity not realizable in real-time.

Similar to the voice of the local minimum, an estimation of the voicing state is derived from an absolute threshold as mentioned there. The lower the global minimum is, the better is the correlation. If the global minimum of the CMNSAMDF is smaller than the threshold, then the frame is considered voiced and in the other case it is considered unvoiced.

Chapter 4

Frequency Domain Algorithms

While for time domain algorithms the organisation of the input buffer was a prerequisite for pitch estimation, for frequency domain algorithms the conversion from the time signal into a frequency spectrum is necessary to perform the algorithm. As mentioned in Subsection 2.2.2 the frequency spectrum of a periodic time signal can be obtained by FFT. Just as the time domain algorithms need more samples than the input frame provides to compute an estimation, so does FFT. Therefore, a similar input buffer organises the time domain input and with each time frame the buffer is updated with the new input samples. On this buffer containing multiple frames of input data FFT is computed on. This procedure is called Short-Time Fourier Transform (STFT).

In this chapter two pitch tracking algorithms operating in the frequency domain are presented. They each take advantage of different characteristics of human pitch and are, therefore, expected to complement each other. The first algorithm described in Section 4.1 is based on Temporally Accumulated Peak Spectrum (TAPS), which takes advantage of the fact that the frequency spectrum of speech changes more slowly than the frequency spectrum of noise. The PEFAC algorithm described in Section 4.2 utilises the appearance of harmonics in the logarithmic frequency spectrum and uses the Long-Term Average Spectrum of Speech (LTASS).

4.1 Temporally Accumulated Peak Spectrum (TAPS)

This section describes the ACF-based TAPS algorithm as presented by Huang and Lee [HL13]. Two other algorithms based on TAPS are also described in the paper; they are based on sparse reconstruction requiring a data set to be trained on. The variant of TAPS described here utilises the ACF in the frequency domain instead of the time domain to obtain an estimation.

4.1.1 The Algorithm

As the name might already hint at, TAPS accumulates the peaks of the frequency spectra of consecutive input frames. The evaluation using ACF then produces relatively high values for F0 and its multiples due to the fixed distance between harmonics.

Unlike RAPT, YIN, and PEFAC, TAPS relies on inputs of subsequent points in time. To distinguish the frequency spectra of different points in time, input frames are indexed by l in this section.

The input for TAPS is a frequency spectrum $s^{(l)}$ with $n \in \mathbb{N}$ frequency bins at time index $l \in \mathbb{N}$. Based on this, the peak spectrum p is then calculated as follows:

$$p_i^{(l)} = \begin{cases} s_i^{(l)}, & \text{if } 0 < i < n-1,\ s_{i-1}^{(l)} < s_i^{(l)} \text{ and } s_{i+1}^{(l)} < s_i^{(l)} \\ 0, & \text{otherwise} \end{cases}.$$

So the peak spectrum takes the local maxima and sets any other value to zero. The first and the last value do not have two neighbours to compare with and are therefore also set to zero.

For an accumulation size $K \in \mathbb{N}$ the accumulated peak spectrum $y^{(l)}$ is then defined by

$$y_i^{(l)} = \sum_{k=0}^{K-1} p_i^{(l-\lfloor K/2 \rfloor + k)}$$

for every $i \in \{0, \ldots, n-1\}$. The peak spectrum reduces the information of the frequency spectrum to local maxima. Adding up consecutive peak spectra then emphasises bins containing local maxima in multiple of these peak spectra.

The frequency spectrum of voiced speech changes much more slowly than the noise spectrum since voiced speech is *quasi-stationary*. For subsequent peak spectra of voiced speech, therefore, is it expected that peaks corresponding to multiples of F0 in neighbouring time frames are also in neighbouring frequency bins or even the same frequency bin. When F0 causes local maxima in the same frequency bin in time-neighbouring peak spectra, then the accumulated peak spectrum has a peak of approximately doubled amplitude at this frequency bin. The frequency spectrum for most kinds of noise changes more rapidly. Therefore, the peaks in neighbouring points in time are expected not to correlate and, therefore, not to accumulate to peaks of high amplitude.

The accumulation size K determines the number of peak spectra to accumulate. If K is too small, the effect of accumulated harmonic peaks does not show. If, on the other hand, this size is too big, the F0 might change too quickly for the algorithm to react, or voiced intervals, which are too short, might pass unnoticed by the algorithm. Also the delay between voiced input and the pitch estimation increases with this constant K, but the real-time requirement wants to minimize the delay. Huang and Lee come to the conclusion that TAPS performs best for an accumulation size of $K = 4$, which could be reproduced on a small set of data for this implementation.

The ACF is calculated on the accumulated peak spectrum for a frequency of up to 1 000 Hz. This is expected to have high values for the F0 corresponding bin and multiples of it since their peaks, which are expected to have a high amplitude due to the accumulation, are separated by F0. The size of ACF is chosen such that it includes at least one harmonic for every possible F0 in the range of the human pitch.

Since the accumulated peaks for F0 and multiples of it might not hit exactly the same bin, the ACF of the peak spectrum might have multiple local maxima in close proximity that correspond to the same harmonic. However, it is easier to handle only one local maximum for each harmonic of F0. The smoothed ACF for a lag $j \in \{2, \ldots, n-3\}$ reduces the number of local maxima by averaging over five neighbouring values of the smoothed ACF and is defined by

$$SACF(j) = \frac{1}{5} \sum_{q=-2}^{2} ACF(j+q).$$

The algorithm as described by Huang and Lee takes the bin numbers of the highest $L \in \mathbb{N}$ maxima, $j_1, \ldots, j_L$, where $j_1 < \ldots < j_L$, of the smoothed ACF and calculates the estimation f_0' on the assumption that these are the first L harmonics using

$$f_0' = \frac{1}{L} \sum_{k=1}^{L} \frac{1}{k} j_k.$$

Since the n-th harmonic has a frequency of n times F0, the bin containing the n-th harmonic is divided by n in this formula to normalize to F0. These approximations of the estimated F0 are averaged to obtain a more precise estimation.

The the number of local maxima, which are considered here, is critical for estimation quality. If there are too few, then the F0 might happen to be the $L+1$ highest maximum. In this case the

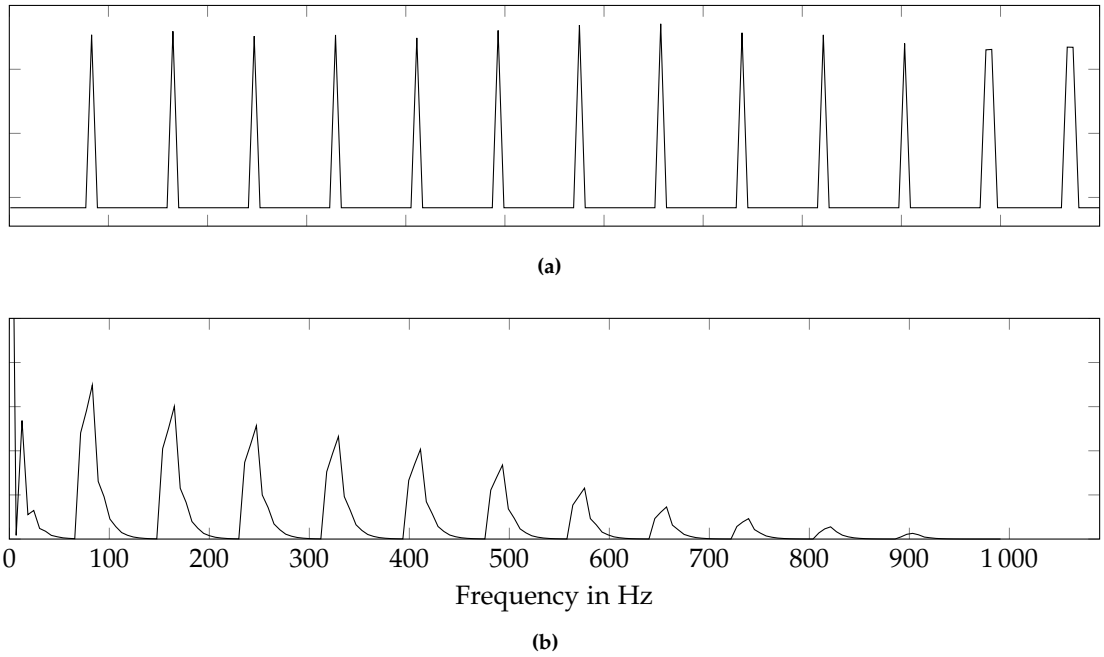

Figure 4.1. Calculation steps for TAPS. (a) the accumulated peak spectrum and (b) smoothed ACF thereof.

calculation fails, since the harmonics are divided by wrong values that do not normalize the harmonics to F0, but rather a multiple of it. Also without averaging, which is the case for $L = 1$, the precision of an estimation is limited to the bin resolution of the frequency spectrum. If otherwise too many maxima are considered, then there might not be enough harmonics present in the evaluation range of the ACF to produce fitting local maxima. In both cases the averaging is corrupted with one or more misleading values. Huang and Lee set $L = 2$ based on empirical data. Regarding the number of harmonics contained in the ACF range, this seems to be a reasonable choice. For every F0 in the range of human pitch there are at least F0 and the first harmonic in the range of considered frequencies in ACF, and for very high values of F0 such as for example 450 Hz, any higher harmonic is not considered by ACF, since $1\,350 > 1\,000$ Hz (the biggest frequency considered in the smoothed ACF).

Figure 4.1 shows the accumulated peak spectrum and the smoothed ACF thereof for a voiced sound with F0 = 83 Hz pronounced by a male speaker. In Subfigure 4.1a not the highest peak but rather the first corresponds to F0. Each peak is equally spaced by F0 for harmonics up to a frequency of 900 Hz, the peak of these harmonics hit the same frequency bins in every peak spectrum. Therefore, the intensity of the accumulated peaks is higher for these. Also there is no peak in this accumulated peak spectrum not corresponding to a multiple of F0. The local maxima of the smoothed ACF in Subfigure 4.1b are monotonically decreasing since the peaks in the accumulated peak spectrum have similar values and the summation size of the smoothed ACF decreases with an increasing lag. Therefore, the highest $L \in \{1, \ldots, 11\}$ local maxima also correspond to the first L harmonics.

4.1.2 Different Evaluation Strategies

Figure 4.1 shows the optimal case for TAPS—every harmonic has corresponding peaks in the accumulated peak spectrum and there are no other peaks that could degrade the results. However, the optimal case is not the usual case. An example for this is shown in Figure 4.2. While the F0 = 194 Hz

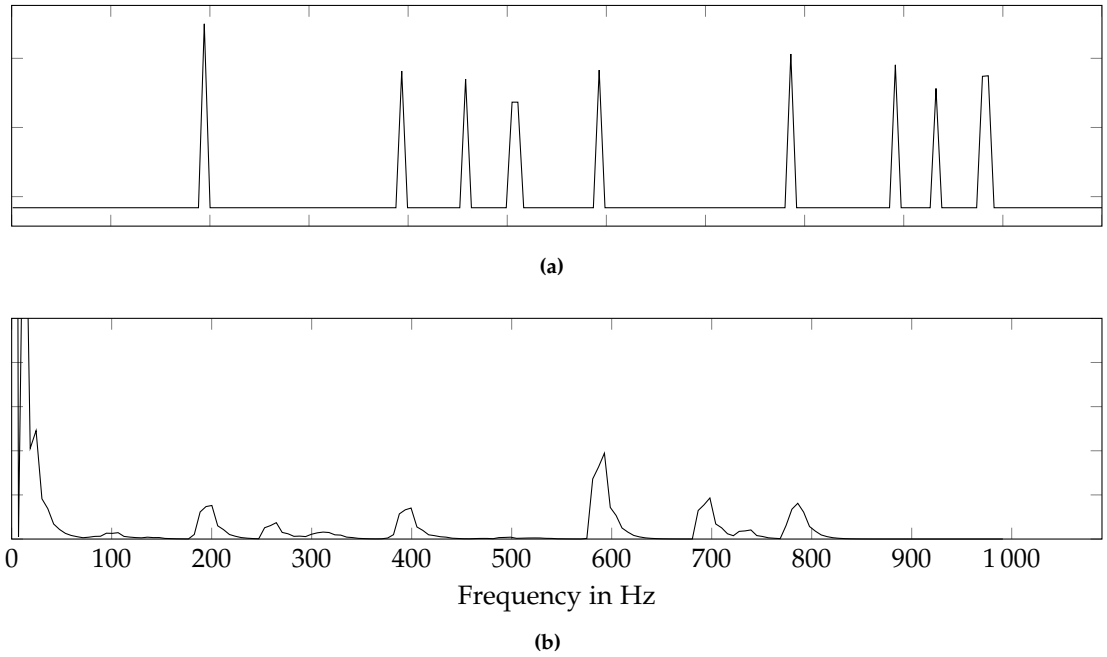

Figure 4.2. Calculation steps for TAPS. (a) the accumulated peak spectrum and (b) smoothed ACF thereof.

in Subfigure 4.2a has the highest peak in the accumulated peak spectrum, the corresponding local maximum in the smoothed ACF of 4.2b is only the fourth highest. The first and second highest peak are at 592 Hz and 698 Hz respectively. The estimation of TAPS is $\frac{1}{2}(592/1 + 698/2) = 470.5$ Hz. Although the estimation is in the range of human pitch, the first considered maximum, which should correspond to F0, is not. In general there are multiple scenarios for which the assumption of the highest local maxima corresponding to the first harmonics does not hold. This section presents some evaluation approaches, which I developed based on observed error types, to improve on these weaknesses.

Curtail of Considered Local Maxima to Possible Harmonics

One source of erroneous pitch estimation by TAPS are not monotonically decreasing or not harmonics-related local maxima, which makes it more difficult for TAPS to match local maxima and harmonics. For $L = 2$, as proposed by the authors, a third or fourth harmonic causing a higher local maximum than the second harmonic is sufficient to mislead the algorithm. Also, a local maximum not corresponding to a harmonic might have a high value. A consideration of such a frequency bin again leads to wrong estimations.

Again the bin numbers of the highest $L \in \mathbb{N}$ maxima, $j_1, \ldots, j_L$, where $j_1 < \ldots < j_L$, of the smoothed ACF are required. Additionally, each local maximum needs to pass a threshold, which is relative to the highest value of the smoothed ACF. Using this, a higher value of L can be chosen while still considering only reasonable local maxima. Then j_k, with $k \in \{1, \ldots, L\}$, is interpreted as i_k-th harmonic, where $i_k = \text{round}(j_k / j_1)$. Hereby every local maximum is assigned the harmonic number that seems most reasonable due to the relation to the first local maximum. The pitch estimation f_0' is then calculated by

$$f_0' = \frac{1}{L} \sum_{l=1}^{L} \frac{1}{i_k} j_l.$$

Although this does not rely on every j_k to be the k-th harmonic, it heavily relies on j_1 corresponding to F0 since otherwise the harmonic number calculation is wrong.

Using this formula, local maxima that do not correspond to any harmonic might still influence the result. To cope with this, considered local maxima can be restricted to those in the vicinity of a multiple of j_1.

Robust F0 Bin Estimation

The approach above fails if j_1 refers to anything but F0. This approach tries to estimate the F0 bin more robustly. The F0 bin estimation is bounded in precision to the frequency bin resolution. To obtain a higher resolution this approach can be combined with the previous one.

Every harmonic of the F0 has a corresponding local maximum in the smoothed ACF. Although these might alone not be the highest, the sum will most likely stand out. For every frequency bin f the summed up harmonic value (SHV) with summation size $K \in \mathbb{N}$ is calculated as follows

$$SHV(f) = \sum_{k=1}^{K} SACF(kf).$$

The frequency bin, which scores highest, is then estimated as the bin corresponding to F0.

Averaging Peak Strategy

A very simple approach takes a step back and estimates the pitch on the accumulated peak spectrum rather than the smoothed autocorrelation. This was motivated by observing many peaks, which nothing to do with the pitch, at high frequencies producing local maxima in the smoothed ACF.

Before the peak spectrum is calculated, in this approach frequency smoothing is applied on the frequency spectrum to reduce the number of peaks not corresponding to F0 or a multiple thereof. Then the lowest indexed peak above a relative threshold is interpreted as corresponding to F0. Averaging with peak in the vicinity of multiples the F0 bin divided by that multiple then refines the estimation.

4.2 Pitch Estimation Filter with Amplitude Compression (PEFAC)

This section describes the PEFAC algorithm presented by Gonzalez and Brookes [GB14]. A second description of probably an earlier version of PEFAC is given by Gonzales and Brookes [GB11]. PEFAC benefits from characteristics of the harmonics in the logarithmic frequency spectrum representation and utilises the LTASS to obtain a higher robustness. The (supposedly) earlier version of the algorithm consists of the following four computational steps [GB11]:

1. Transform the time domain signal to the frequency domain as described in Subsection 2.2.2.

2. Transform the frequency spectrum to a logarithmic scale.

3. Compute the LTASS normalized spectrum.

4. Apply the analysis filter for harmonic peaks to the normalized spectrum and select the highest peak in the range of F0 as the estimation.

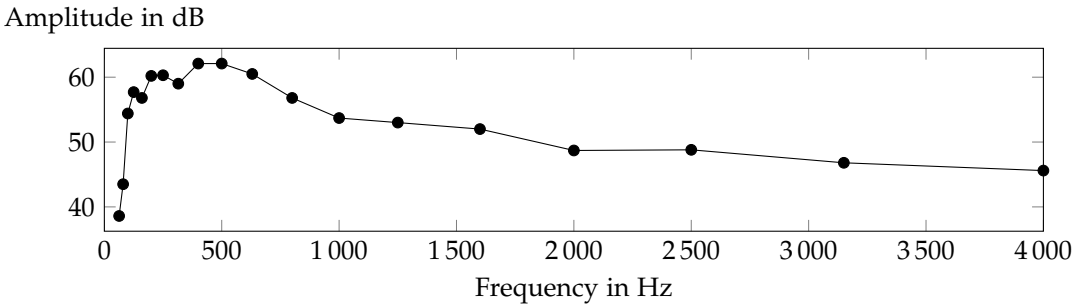

Figure 4.3. LTASS as determined by Byrne et al. [BDT+94].

4.2.1 Long-Term Average Spectrum of Speech (LTASS)

The LTASS measures the average amplitude of frequencies contained in human speech. This includes voiced *and* unvoiced speech. Byrne et al. determined LTASS for 12 languages including German and several dialects of English [BDT+94]. Figure 4.3 visualises LTASS over all examined languages taken from Table 2 of their work for frequencies up to 4 000 Hz. Although F0 is not larger than 500 Hz, the original LTASS contains frequencies up to 16 000 Hz because of the higher harmonics of F0.

When the input frequency spectrum is weighted by LTASS, then frequencies are weighted by their occurrence in normal speech. Further evaluations will then favour frequencies produced by human speech over noise.

Since LTASS is only given for a few points on the frequency spectrum and not for the whole frequency axis, piecewise linear interpolation is used to approximate its values on the continuous frequency spectrum. Note that this is already visualized by the solid line in Figure 4.3. For $n \in \mathbb{N}$ monotonically increasing data points $x_0, \ldots, x_{n-1}$ with corresponding values $LTASS_0, \ldots, LTASS_{n-1}$ the value for a frequency $f \in [x_i, x_{i+1}]$, with $i \in \{0, \ldots, n-2\}$, can be approximated by

$$LTASS(f) = LTASS_i + \frac{f - x_i}{x_{i+1} - x_i} \left(LTASS_{i+1} - LTASS_i\right).$$

This can be derived from the Lagrange polynomials, which were defined in 2.3. However, verifying this formula can be done without further knowledge. The function is obviously linear and for $f = x_i$, with $i \in \{0, \ldots, n-1\}$, it holds that $LTASS(f) = LTASS_i$.

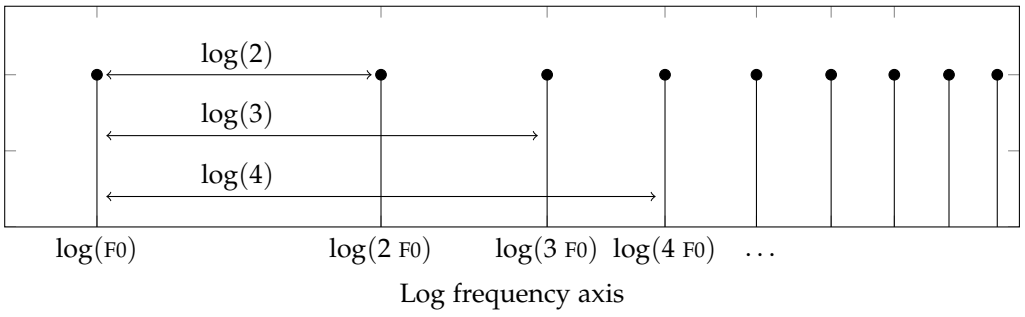

Figure 4.4. Equidistant points, such as the harmonics, on a logarithmic axis.

4.2.2 Advantages of Using the Log-Frequency Axis

While the occurrence of harmonics can degrade the estimation accuracy of many pitch tracking algorithms like RAPT or YIN, PEFAC takes advantage of their relation to F0.

The frequency of the n-th harmonic of F0 is nF0 as stated in Section 2.1. Due to logarithm laws it holds that for every $n \in \mathbb{N}$

$$\log(n\text{F0}) = \log(\text{F0}) + \log(n).$$

Therefore, the distance between F0 and its multiples does not depend on the actual value of F0. Figure 4.4 shows this property of the logarithmic axis. An algorithm operating on the logarithmic frequency domain can, therefore, search for this pattern of absolute peaks with fixed predefined distances in the range of F0. The frequency, which fits this pattern best, then becomes the estimation for F0.

4.2.3 Obtaining the Log-Frequency Axis

The input for PEFAC is a frequency spectrum containing discrete frequencies and their amplitudes. The following roughly describes one possibility for obtaining the logarithmic frequency spectrum from the given (linear) frequency spectrum using a filterbank. Overlapping triangular filter are applied on grouped frequency bins with a logarithmically increasing group size [Nie13]. The amplitude of each triangular is chosen such that the area is the same for every triangle. Since the group size increases with frequency, the height of the triangular filters decreases. An example for these overlapping filters of adjusted amplitude is shown in Figure 4.5. One degree of freedom is the choice of the logarithmic function determining the groups size of (linear) frequencies merged to one frequency on logarithmic scale.

For speech processing with regard to speech recognition the *mel-scale* is widely used. It was developed to describe the human perception of pitch, which happens to be kind of logarithmic, on a linear scale [Nie13]. Therefore, the mel-scale could be a reasonable choice for the logarithmic frequency axis. A conversion from a frequency f_{Hz} in Hertz to a frequency f_{mel} in mels is given by [Fan68]

$$f_{\text{mel}} = \frac{1000}{\log(2)} \log\left(1 + \frac{f_{\text{Hz}}}{1\,000}\right).$$

However, adding 1 inside of the logarithm is suboptimal, since then logarithm laws as needed in the previous subsection do not hold.

An alternative is the *cent-scale* that is suited for comparing the size of intervals [Loy11]. For a reference frequency f_{ref} the frequency in cent f_{cent} for a frequency in Hertz f_{Hz} can be calculated by [Loy11]

$$f_{\text{cent}} = \frac{1\,200}{\log_{10}(2)} \log_{10}\left(\frac{f_{\text{Hz}}}{f_{\text{ref}}}\right).$$

4.2.4 The Algorithm

First the input frequency spectrum s with $n \in \mathbb{N}$ frequency bins is transformed to any logarithmic frequency spectrum Y containing $\tilde{n} \in \mathbb{N}$ frequency bins.

This is then smoothed in time and frequency obtaining the smoothed spectrum $\bar{Y}$. Then it is normalized by the LTASS using

$$Y'(q) = Y(q) \frac{LTASS(q)}{\bar{Y}(q)} \tag{4.1}$$

The resulting spectrum Y' is expected to improve the estimation. Short noise utterances are reduced in amplitude due to the time smoothing and noises outside of the the frequency range of normal speech

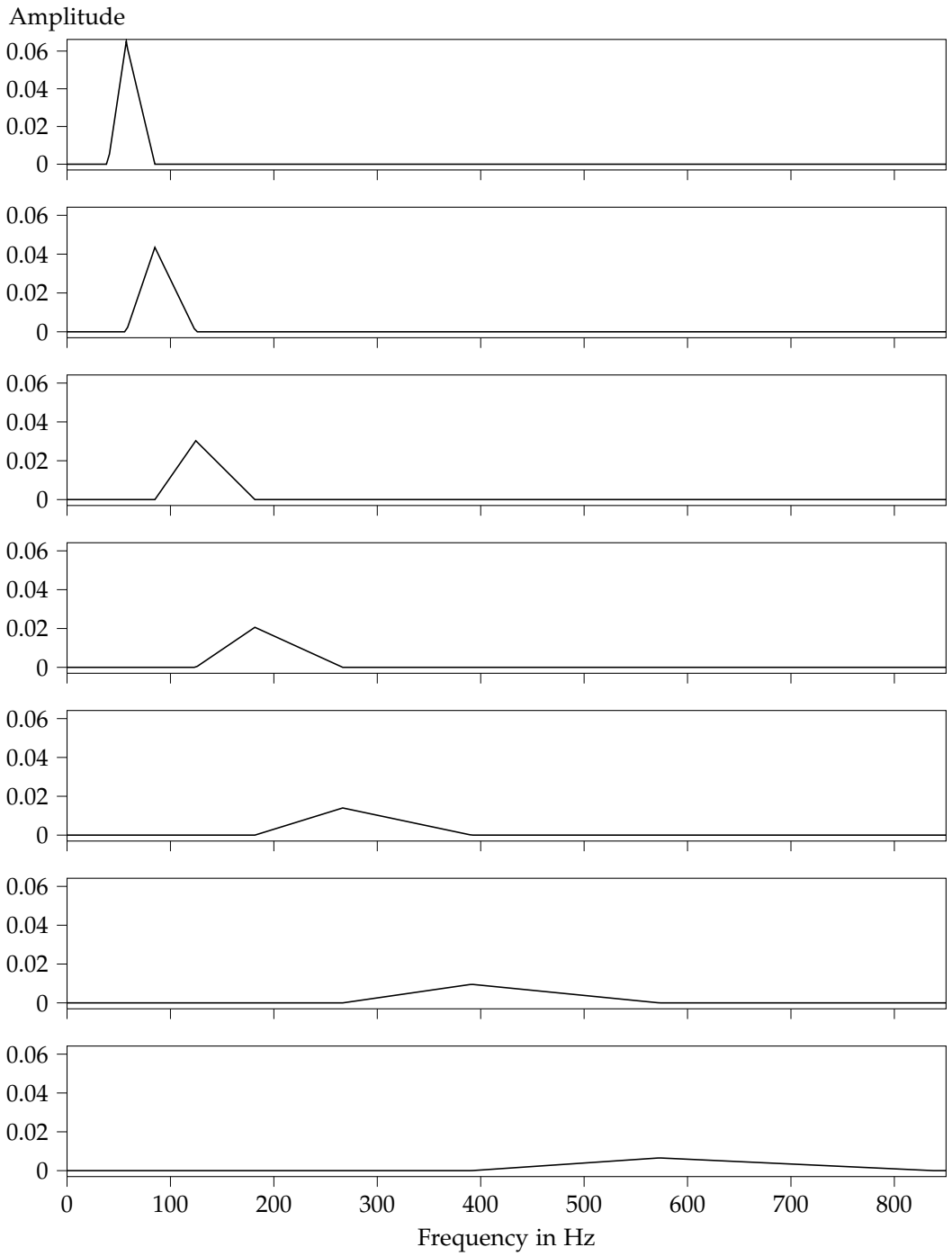

Figure 4.5. Overlapping triangular filters with adjusted amplitudes.

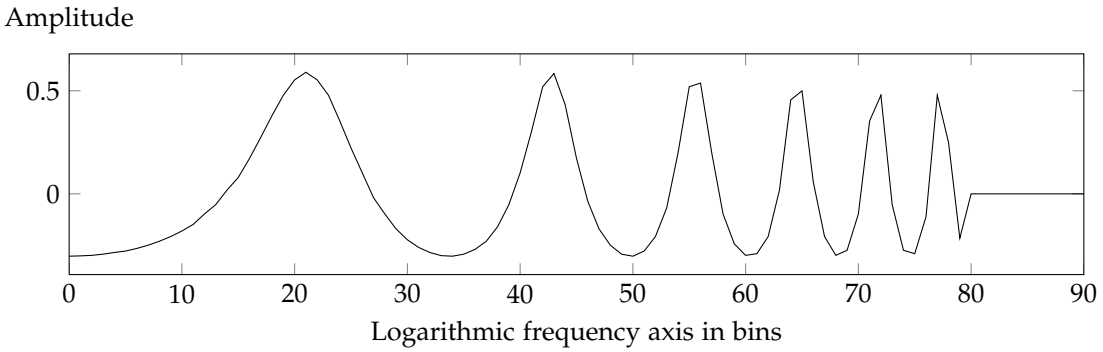

Figure 4.6. Filter design according to Equation 4.2 with $\gamma = 1.8$ and 6 peaks for harmonic maxima on 256 bins on the logarithmic axis. Values for bins greater than 90 equal zero and are, therefore, omitted.

are degraded in amplitude by the LTASS normalization.

In theory for a periodic signal the pattern visualized in Figure 4.4 would be sufficient for determining F0. Instead a pattern with widened peaks is used since the signal of voiced speech is quasi periodic and arithmetical errors can occur determining a logarithmic frequency bin *exactly*. Therefore, the filter h used by Gonzales and Brookes is defined by

$$h(q) = \frac{1}{\gamma - \cos(2\pi e^q)} - \beta \tag{4.2}$$

with $\gamma > 1$ controlling the peak width and $\beta \in \mathbb{R}_{\geqslant 0}$ chosen such that $\int h(q)dq = 0$. For the discrete case this means that the sum over the discrete values is zero, which is fulfilled by choosing β as the mean. A possible filter design according to Equation 4.2 is shown in Figure 4.6.

The filter is then convolved with the smoothed logarithmic frequency spectrum by

$$Y'(q) * h(q) = \sum_{k=0}^{M-1} Y'(q-k)h(k).$$

The location of the global maximum of this function is then chosen as the estimated pitch according to the first version of the algorithm. The (supposedly) new version of the algorithm [GB14] then

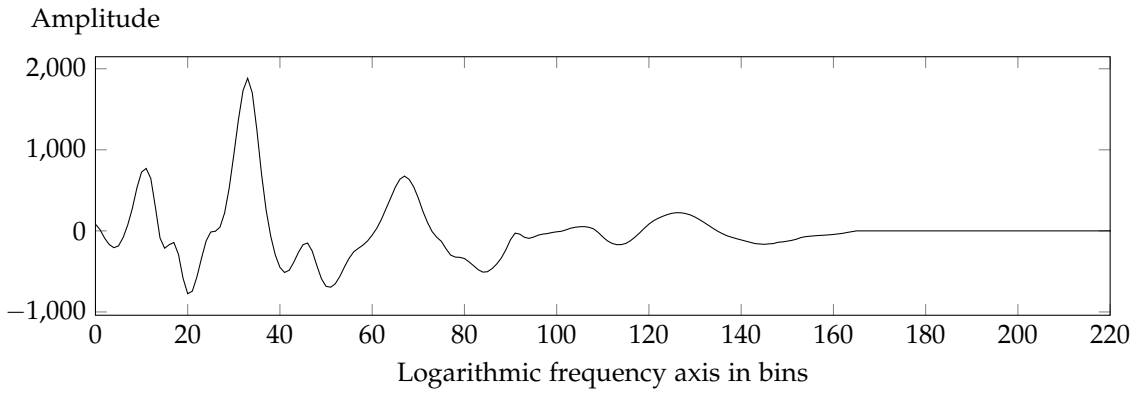

Figure 4.7. Example values of the convolution.

adds a post processing step that chooses the estimation out of the three highest peaks using dynamic programming. Hereby, the relative amplitude of the peaks, the rate of change from the last estimations, and the deviation from the median of expected pitches are considered in the minimal cost function.

4.2.5 Implementation

The implementation desists from the dynamical programming in the post processing described in the (supposedly) new version for the algorithm to be applicable in real-time.

Figure 4.7 shows the convolution for a voiced speech signal. The maximum, which determines the estimated pitch, is at bin 33. Since bin 32 has a higher value than bin 34, the actual pitch is probably between the frequencies that the 32nd and 33rd bin refer to.

The accuracy of the estimation is limited to the bin precision. In the logarithmic domain the distance between two bins increases with increasing frequency. Since again a maximum of a function determines the estimation, parabolic interpolation as described in Subsection 2.3 was added in this implementation.

Chapter 5

Proposed System

The previous Chapters 3 and 4 describe four different algorithms for pitch tracking in the time and frequency domains. In this chapter the results of these algorithms are used as an input for a combining estimation. Section 5.1 describes the connection of submodules as needed for further handling. This includes the data flow from the input signal to each of the algorithms and the output of the combining pitch estimation algorithm, which is then described in Section 5.2.

5.1 Composition of the Pitch Tracking Module

This section first describes the framework of the implementation and the structure of the composite pitch tracking module. Then details regarding the output of every algorithm implementation for the candidate evaluation are described.

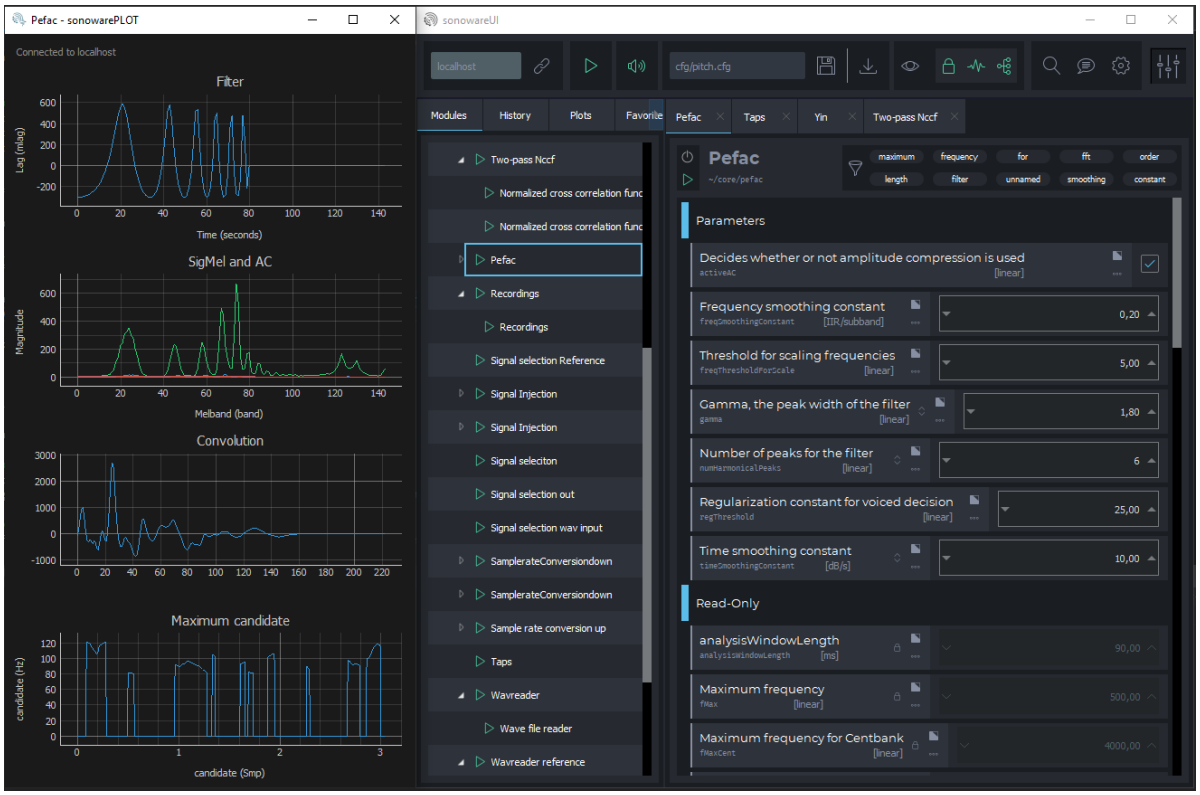

Figure 5.1. User interface.

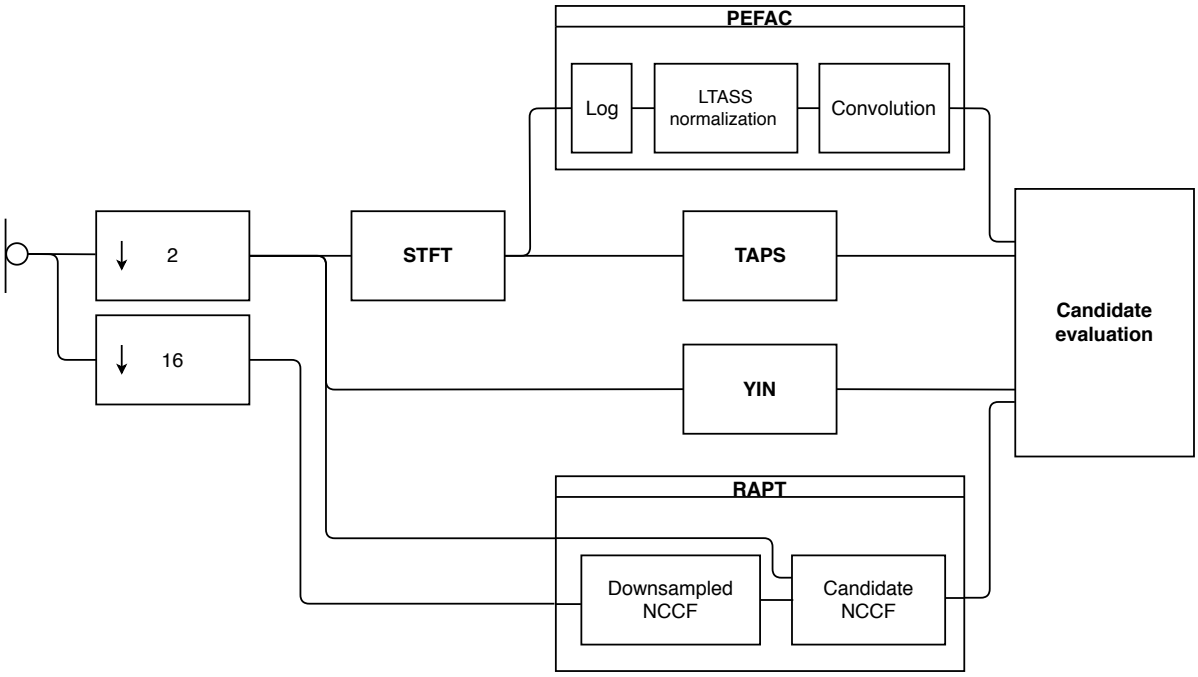

Figure 5.2. Block diagram of the composed pitch tracking module.

The implementation is in C and YAML with unit tests in C++. For each algorithm a processing function for one frame is implemented. Modules for sample rate conversion and the STFT already existed in the framework and could, therefore, be used. A filterbank using the mel-scale was slightly modified to use the cent-scale (see Subsection 4.2.3) to fit the needs of PEFAC. A graphical environment for plugging modules together combined with automatically generated code including memory management facilitated the connection of the submodules. For visual verification of (partial) results, real-time plotting of variables used by an algorithm is also supported by the framework. The previously presented algorithms as well as the CE module described in the next section are configurable by many parameters such as the voicing thresholds. For a complete listing of all parameters see Appendix A. With the possibility of adjusting these parameters in real-time during the execution it was easier to find a well-working set of parameter values. Figure 5.1 shows the user interface to configure parameters and a real-time plot.

Figure 5.2 shows a block diagram of the composed module for pitch tracking. The input, which can be a microphone signal or an audio file, was chosen to have a sample rate of 48 kHz, which is common for audio applications. The signal is downsampled by a rate of two, resulting in a signal with a sample rate of 24 kHz, to lower the computation cost. This does not degrade the estimation quality, since F0 is lower than 12 kHz and every algorithm has a refining component, such as parabolic interpolation. Additionally, RAPT needs another version of the signal at an even lower sample rate, since it calculates the NCCF on two sample rates. This is provided by downsampling the input signal by a rate of 16 to a sample rate of 3 kHz. RAPT and YIN, since they operate in the time domain, then compute their estimations on these downsampled time signals. For TAPS and PEFAC the input is a frequency spectrum. This is provided by an STFT based on the 24 kHz signal. The outputs of the four algorithms are then used by CE to obtain a combined estimation and voiced decision. The algorithms have different delays from the input to the according estimation, which was considered using delay

buffers of the size of the individual delay for each of the algorithms.

Every algorithm is implemented such that a main pitch candidate, a voiced decision, and a set of secondary candidates are calculated to provide more information for CE.

For every algorithm a maximum number of secondary candidates restricts the choice of secondary candidates to the most reasonable ones and also prevents one algorithm from falsifying the estimation by producing too many secondary candidates. The following describes how each algorithm obtains its secondary candidates and what kind of information they might provide that is useful for the candidate evaluation.

For RAPT and YIN the smallest lag satisfying a threshold was chosen to define the main candidate. Therefore, it seems reasonable that frequencies corresponding to lags satisfying the threshold are chosen as secondary candidates. Since there is a maximum number of secondary candidates, the smallest lags satisfying this condition are chosen. A different approach could take the lags with the largest (for RAPT) and smallest (for YIN) value. This choice has a rather small impact, since the number of lags satisfying the threshold does not often exceed the maximum number of secondary candidates (see Figure 3.2 with a maximum number of secondary candidates of five for an example). The smallest lag meeting the threshold often corresponds to F0 and due to the periodicity of the used correlation functions, the secondary candidates obtained by RAPT and YIN most likely refer to lags that are multiples of T0. Therefore, these secondary candidates are interpreted as subharmonics of F0.

The implementation of TAPS consists of different strategies for which the choice of secondary candidates differ. For the averaging peak approach, the first peaks satisfying a threshold are taken as secondary candidates. For the other strategies, which involve the smoothed ACF, the highest local maxima of the smoothed ACF are taken as secondary candidates. While in both cases the secondary candidates are again obtained by multiples of the original candidate, in the case of TAPS they are multiples in the frequency domain instead of the time domain. Therefore, they are interpreted as harmonics.

The main candidate of PEFAC is the frequency corresponding to the bin with the highest convolution value on the logarithmic axis. Therefore, the frequencies corresponding to the highest convolution values seem to be a reasonable choice for the secondary candidates. Due to the matched filter convolution on the logarithmic frequency axis, the classification of these secondary candidates as harmonics or subharmonics is not as easy. Comparing a set of thereby generated candidates to F0 indicates that these often refer to subharmonics. This might be caused by the second or third filter peak scoring well on F0.

5.2 Candidates Evaluation (CE)

The algorithms RAPT, YIN, TAPS, and PEFAC are chosen based on their relevance and their different approaches using specific characteristics of the pitch. RAPT and YIN represent the time domain algorithms both considered to be the "best performing" pitch tracking algorithms [PWP+11b]. Both algorithms are based on a correlation function and, therefore, both follow the idea of measuring the similarity of the signal with and without delay, different approaches optimize the estimation quality. TAPS utilises the temporal aspects differing for human pitch and noise. Last, but definitely not least, PEFAC combines the LTASS and a matched filter on the logarithmic frequency axis to obtain an estimation, which is robust to noise. If an algorithm fails to estimate the pitch correctly, then, due to the different approaches, it is expected that other algorithms might produce a correct estimate. It might also be contained in the secondary candidates or at least information about the correct estimation is contained in the form of harmonics or subharmonics, which can be used to then obtain the correct estimation.

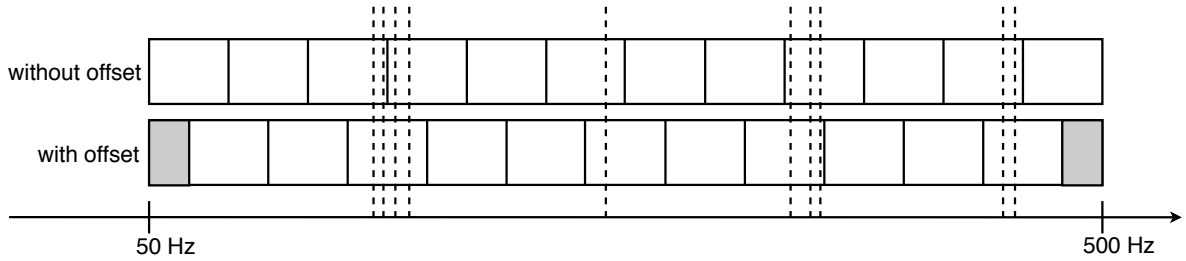

Figure 5.3. Distribution of frequency bins with $K = 37.5$: top without and bottom with offset. The dashed lines indicate a possible candidate input. For the bin distribution without an offset a maximum of three dashed lines are contained in one bin. However, if the bins are arranged with an offset, the third bin contains four candidates.

There are numerous possibilities regarding what to base the combined estimation on. Somehow a measurement of continuity and the question of whether a candidate refers to a (sub-)harmonic or not needs to to be answered for some set of values this is tested for. One possibility is to bind these to input candidates. Thereby, one could evaluate which of the input candidates most likely refers to F0 and this candidate is then chosen to be the final estimation. Using this approach it is necessary to identify candidates of different algorithms that refer to a similar estimation since this is an indicator for the relevance of that candidate. However, this is not an easy task since relative or absolute thresholds for this might fail for a set of nearby candidates. Also the choice of estimation is limited to the input candidates. If, for example, 200 Hz, 300 Hz, 400 Hz, 500 Hz, and 600 Hz are input candidates and it is known that the candidates are likely to be harmonics of F0, then 200 Hz would be the most reasonable choice from the input candidates. However, the appearance of 300 Hz and 500 Hz rather indicate that a better estimation would be 100 Hz.

To separate the concerns of given candidates and the estimation made by the combining algorithm, instead the following approach is taken. The range of F0 comprising 50 Hz to 500 Hz is partitioned into relatively large frequency bins of equal size $K \in \mathbb{R}_+$. Two versions of these frequency bins are used, one starting directly at 50 Hz and one starting with an offset of half the bin size at $50 + K/2$ Hz. The second version is intended to handle border cases where F0 is near to $50 + nK$ Hz for $n \in \mathbb{N}$. For the bin distribution without an offset, the candidates might then favour both neighbouring bins equally, resulting in a bin unrelated to F0 containing most candidates, as shown in Figure 5.3. In the following, "frequencies" and "frequency bins" with and without offset are used as synonyms. The conversion is determined by the bin size.

The estimation is then based on a reward system for these frequency bins, which gain or lose points based on different information retrieved from the input candidates and the past estimation. The final score for each bin is the sum of points obtained by the scoring steps considering continuity, secondary candidates, and the main candidate. The mean of candidates in the highest scoring bin then serves as the final estimation.

F0 can change with every glottal period. The difference of F0 for two neighbouring glottal periods is either small or exactly one octave [Tal95]. Since most estimation errors are octave errors and octave jumps rarely occur, only the first case of F0 being to some extend continuous is considered by the algorithm. The continuity of the estimation is supported by adding a constant to bins in the vicinity of the estimation from the last frame. The actual meaning of vicinity depends of the choice of frequency bin size. For a bin size of 10 Hz a vicinity of two bins performed well with regard to fast F0 changes and exclusion of octave errors.

While for TAPS the secondary candidates contain harmonics of F0, for RAPT, YIN, and PEFAC these

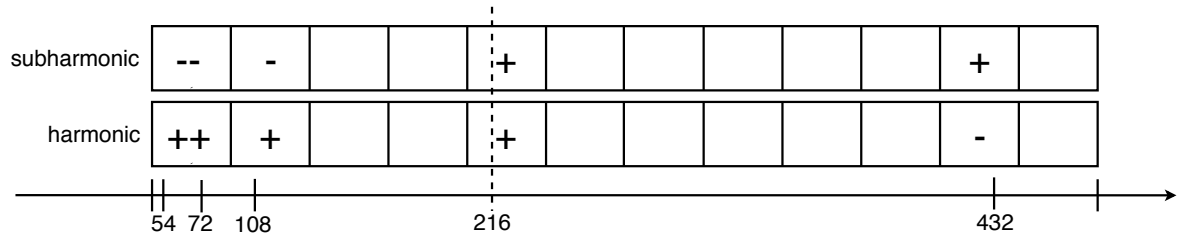

Figure 5.4. Secondary candidates scoring example for a candidate at 216 Hz interpreted as a subharmonic in the upper bins and interpreted as a harmonic in the bottom bins. Each plus indicates a bin gaining points and each minus indicates a bin losing points.

often contain subharmonics of F0. Although it is not known which (sub-)harmonic the secondary candidate refers to, this can still be utilised as a rewarding factor. For a subharmonic f_{sub} there exists $n \in \mathbb{N}$ with $nf_{\text{sub}} = \text{F0}$. The other way around, for a harmonic f_{har} there exists $n \in \mathbb{N}$ with $f_{\text{har}} = n\text{F0}$. Bins fulfilling the first equation for a secondary candidate, which is interpreted as a subharmonic, gain points in the scoring system. Similarly this is done for bins fulfilling the second equation for a secondary candidate, which is interpreted as a harmonic. This direction utilises secondary candidates as indicators for bins corresponding to F0. In the other direction, secondary candidates can also indicate which bins are less likely to correspond to F0. If an algorithm is expected to produce subharmonics, then bins corresponding to subharmonics of these candidates might also have to some extent good values since they are also subharmonics of F0. Since the candidate is already a subharmonic of F0, its bins containing the subharmonic of this subharmonic lose points, since they are not containing F0 but rather a subharmonic of it. The same holds for harmonics of secondary candidates expected to refer to harmonics. An example for the scoring by one candidate for the bins without offset is shown in 5.4.

The final piece of the rewarding system are the main candidates. Harmonics and subharmonics of these are not considered since the main candidate often has an non-refined equivalent in the secondary candidates. Each input main candidate adds a constant reward value to the bins it is contained in. If algorithms differ in reliability, this can be considered in the choice of the constant.

For one speaker the deviation from the mean F0 is often small. Therefore, mean-based rewarding of bins would be an option. Since pitch-tracking is not restricted to one speaker, this implementation desists from taking the mean value into account. One example of the mean being harmful could be a conversation between two people, one with a mean F0 of 70 Hz and one with 210 Hz. The pitch of the second person might then be interpreted as a harmonic of the first person and, therefore, the mean might falsify the estimation.

Besides voiced decisions of the individual algorithms, the combined voiced decision also takes the old voiced decision and the estimation continuity into account. From this information a voicing score is determined. If a threshold is passed, then the frame is considered voiced and otherwise it is considered unvoiced. Taking the previous voiced decision into account reduces fluctuations of the voiced state. For this implementation each input voiced estimation and the previous estimation being voiced increase the voicing score by one. For unvoiced speech or silence, the estimations of neighboured frames are unrelated and thus the continuity is an indicator for the voiced state. Therefore, if the estimation does not behave continuously despite of the continuity reward, the voicing score is reduced. For this a reduction by one or even a reset to zero, which leads to an unvoiced decision, are reasonable. For this implementation a threshold of 3 voiced decisions decides on the combined voiced decision. Also TAPS's voice decision was excluded from the combined voiced decision because the accumulation of subsequent frames lowered the reaction time for the voiced decision, which falsified the results.

The general structure of this strategy of CE has many degrees of freedom. Firstly, the impact of different algorithms and rewarding systems can be regulated by the constants. Then new reward strategies can be added or strategies not helpful in a context can be removed. The frequency bin size influences the accepted deviation from estimations still considered to refer to the same value. Also algorithms can be added or removed to suite the contextual needs best. For example TAPS has the highest delay, so in a scenario where reaction time is critical, one would probably not include TAPS. In contexts, where TAPS performs best in the sense of estimation quality, the system benefits from including TAPS in the combination algorithm. Adding more well working approaches as inputs would probably increase the stability even more. Also CE works with only one input algorithm. Whenever an algorithm is added constants for its weight need to be added and whenever changing the number of input algorithms the voicing threshold has to be adjusted. The latter could be avoided by using a threshold relative to the number of algorithms used. However, this implementation sticks to the absolute threshold, since the nature of different voiced state estimations, such as TAPS reacting too slowly, has to be taken into account, too. An automatic computation of parameters for an arbitrary combination of algorithms could be added. Again due to algorithms performing individually different and since the the algorithmic setup has only to be done once, this algorithm sticks to a manual configuration. Since the different approaches are computed independently, the computation could be performed concurrently to reduce computation time.

Chapter 6

Evaluation

This chapter evaluates the implemented algorithms with regards to the estimation quality and execution time per frame. In Section 6.1 a speech database with reference pitch and voiced decision is used to evaluate the estimation quality. The average execution time for each algorithm and the combined estimation is evaluated in Section 6.2.

6.1 Estimation Quality

This section aims to evaluate the estimation quality of the different algorithms based on a database of *clean* speech and *noise corrupted* speech. For noise corrupted speech the Signal to Noise Ratio (SNR) is the ratio between the power of the speech signal and the power of the unwanted background noise and often this quantity is denoted in decibels. The SNR in dB of a signal $s_0, \ldots, s_{N-1}$ and noise $n_0, \ldots, n_{N-1}$ is defined as [PTG+14]

$$SNR = 10 \log_{10} \frac{\sqrt{\frac{1}{N} \sum_{k=0}^{N-1} s_k^2}}{\sqrt{\frac{1}{N} \sum_{k=0}^{N-1} n_k^2}}.$$

Besides SNR also the noise type influences the estimation quality for a noisy speech signal. For example *white noise* has equal intensities for all frequencies. The name refers to white light, due to the similarity that white light has equal quantities for all colours [Man03].

This evaluation aims to test the robustness of the algorithms against noise. Although there are methods to reduce the noise in speech signals, these are not applied here to really evaluate the algorithms estimation quality. However, when estimating pitch in a noisy environment, applying modules for noise reduction is a reasonable preprocessing step.

Subsection 6.1.1 first describes the experimental method. Then 6.1.2 presents the results, which are further discussed in 6.1.3.

6.1.1 Experimental Method

A common reference for the evaluation of speech related work is the Texas Instruments/Massachusetts Institute of Technology (TIMIT) [LKS89] database. It was designed for acoustic phonetic studies and contains 2342 different sentences from three categories. The first category contains two sentences, which were designed to differ in pronunciation by dialects. The *phonetically compact sentences* set was designed to, as far as possible, contain each pair of phonemes. This set consists of 450 sentences. The third set, which contains 1890 sentences, were randomly collected for contextual coverage.

For the evaluation of pitch estimation quality the Pitch Tracking Database from Graz University of Technology (PTDB-TUG) [PWP+11b; PWP+11a] is used. The database consists of 4720 recordings from 10 female and 10 male native English speakers pronouncing the 2342 sentences from the TIMIT corpus. Two of these sentences are pronounced by each speaker and the other 2340 each by one female and one male speaker, hence leading to the total number of 4720 recordings. The speech signal,

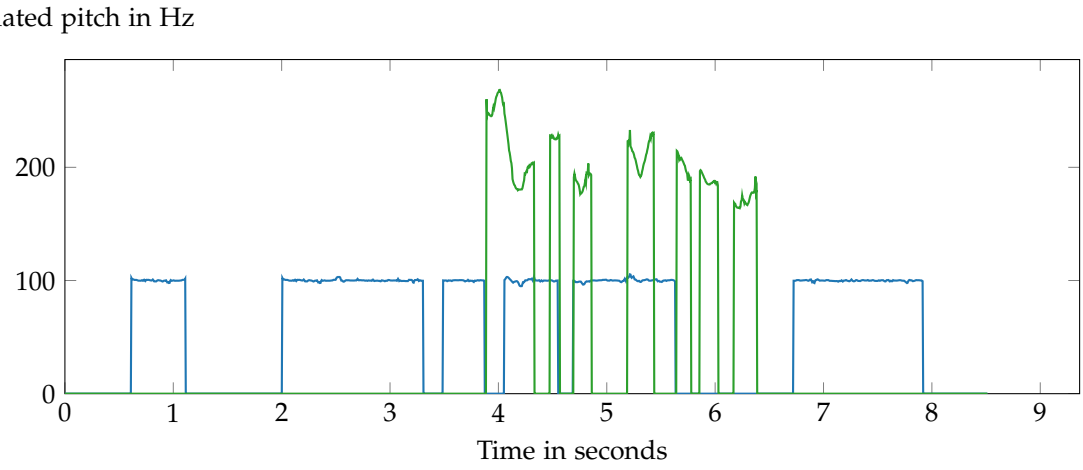

Figure 6.1. Broken reference. The reference pitch is shown in blue and the estimated pitch is shown in green.

the laryngograph signal, which measures the contact of vibrating vocal folds, and a reference pitch determined by RAPT on the laryngograph signal are provided for each utterance. The calculation on the laryngograph is expected to be more precise than for clean speech since the glottal excitation is not manipulated by the vocal tract.

For the speech signals of sample rate 48 kHz the estimations by each of the four implemented algorithms and the combining algorithm were recorded and evaluated. Additionally, estimations of the four different approaches on TAPS strategies were recorded and evaluated.

Since the estimation rate of the reference differs from the estimations made by the algorithms, the reference pitch was interpolated linearly and then resampled to the estimation rate of the implementations. Since unvoiced frames are denoted with an estimation of zero by the reference, interpolation calculates unreasonable estimations for frames between an unvoiced and a voiced reference. To reduce the hereby caused pitch estimation error, the reference for such frames is set to zero. An estimation was considered correct if it fell within 5% of the reference pitch, which is similar to the error definition by PEFAC [GB11].

To evaluate the robustness against noise, white noise and car noise have been added, respectively, for each SNR in $\{-20\,\text{dB}, -15\,\text{dB}, -10\,\text{dB}, -5\,\text{dB}, 0\,\text{dB}, 5\,\text{dB}, 10\,\text{dB}, 15\,\text{dB}, 20\,\text{dB}\}$. For the energy calculation of the speech signal only active speech parts are considered as suggested by the ETSI technical specification [ETS17]. Therefore, each file was partitioned in frames of 30 ms length. For each frame the energy value was calculated and frames with less then 10% energy of the maximum energy frame are considered silent and were omitted for the speech energy calculation. Thereby, the noise' and the speech signal's size for the energy calculation differ. To compensate this, the noise energy was then scaled, such that both energy values refer to the same signal length. From these energy values a factor for the desired SNR is applied to the noise signal. White noise was modelled by randomly generated numbers and car noise was represented by in-car recordings of a Smart on a country road without the occurrence of irregular noises, such as produced by the windscreen wiper and the indicator.

During the first evaluation phase single files with an error of 100% between the reference pitch and the estimated pitch stood out. The examination of these files yielded that some of the laryngographs were incorrect, which led to false reference pitches. One example for this is shown in Figure 6.1. The voiced decision and the pitch estimation are displayed together by setting the estimation to zero for unvoiced frames. The reference pitch shown in blue stays at approximately 100 Hz for every voiced

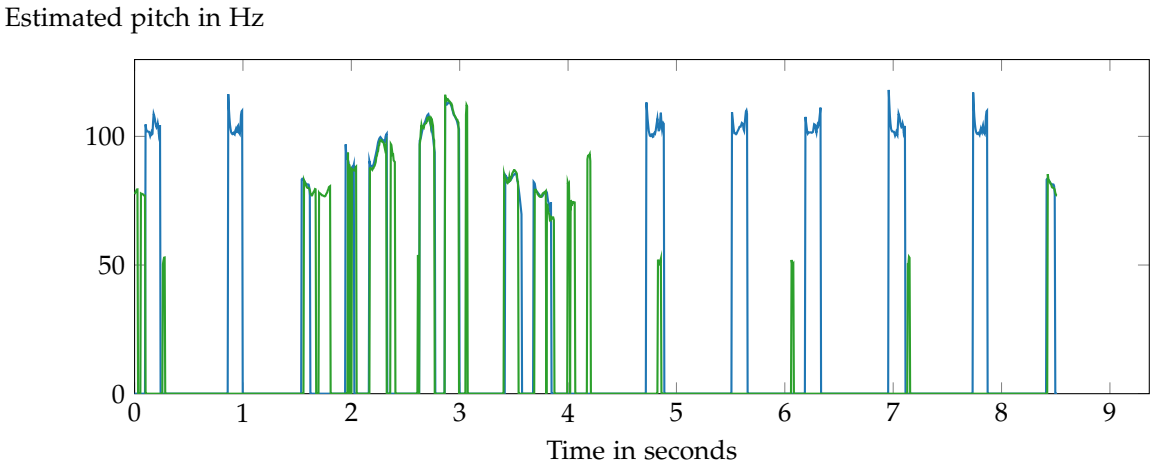

Figure 6.2. Half broken reference. The reference pitch is shown in blue and the estimated pitch is shown in green.

estimated frame in the file. However, the perceived pitch from the speech signal conforms to the estimation made by this implementation, shown in green. Since the corrupted reference pitches would falsify the evaluation result, such files were excluded from the evaluation.

However, this is not an exhaustive solution, since reference pitches, which are not completely, but in most instances, incorrect, still influence the results. A single file error of above 50% is still conspicuous compared to an overall error of below 10%. For Figure 6.2 the estimation by the implementation, denoted in green, again matches the auditory perception. During the voiced part the estimation and the reference pitch coincide. However, there are equally spaced reference estimations again at approximately 100 Hz occurring at silent intervals. To avoid the need of manual reference verification for each file, random files with an error rate above a threshold were inspected to define a reasonable error rate threshold for which files with an error above that threshold are likely to contain corrupted references and files with a lower error rate are likely not to contain corrupted references. Files with an error above 15% regarding the whole file or an error rate above 30% regarding voiced frames only were excluded from further evaluation. This affects 290 of the 4720 files.

Determined quantities for each algorithm are the error combining the voiced decision and pitch estimation, the error only regarding pitch estimation, and the voiced estimation error. The combined error is defined by the number of frames with an estimation not within 5% of the reference, with unvoiced frames set to zero in the estimation and reference, divided by the total number of frames. The pitch error only considers the frames labelled voiced by the reference. Additionally, for the pitch estimation error the rates of too high and too low estimations were calculated and for the voiced estimation error there is a distinction of errors occurring because the estimation is voiced, but the reference is unvoiced, and because the estimation is unvoiced, but the reference is voiced.

6.1.2 Data Analysis

This section presents the evaluation results. The different TAPS strategies are referred to as TAPS-r for the strategy as described by Huang and Lee [HL13], TAPS-h for the harmonics based approach, TAPS-f for the F0 bin determination, and TAPS-p for the averaging peak strategy. This chapter presents a fraction of the evaluation results. A table with complete coverage is given in Appendix B.

For clean speech the percentage of incorrectly estimated pitch for frames considered voiced by the

reference is shown in Figure 6.3. With 7.96% PEFAC has the lowest error rate. While TAPS-p improves the error of TAPS-r by approximately 5% from 17.09% to 12.26%, TAPS-h and TAPS-f estimate incorrectly for more than a third of the voiced frames. Due to these results, further evaluations of TAPS will use the averaged peak strategy. YIN estimates incorrectly in 15.20% of the voiced frames and RAPT in 12.56%. For CE, PEFAC, and TAPS-p most pitch estimation errors estimate the pitch higher than the reference pitch. For RAPT and YIN there are about equal too high and too low estimation errors.

Subfigure 6.4a shows the voiced decision error. CE has the lowest error rate of 5.58%. RAPT and YIN have an voiced error rate of 6.00% and 6.42%, respectively. With an error rate of 7.38% PEFAC has a slightly higher error rate. The highest error rate of 10.9% belongs to TAPS. Note that the different strategies of TAPS do not influence the voiced state decision and, therefore, there is no further distinction

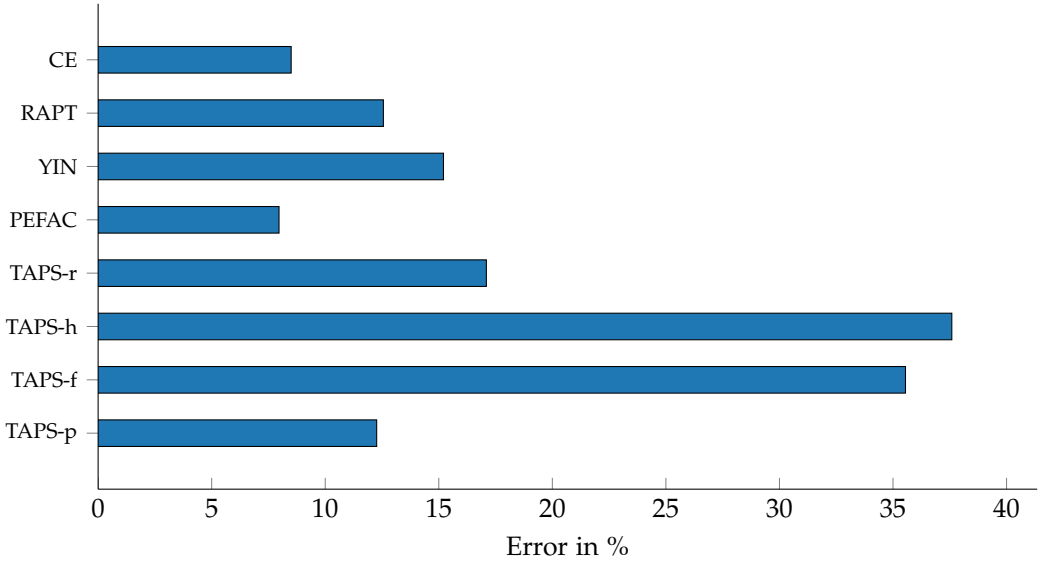

Figure 6.3. Pitch estimation error for the different algorithms.

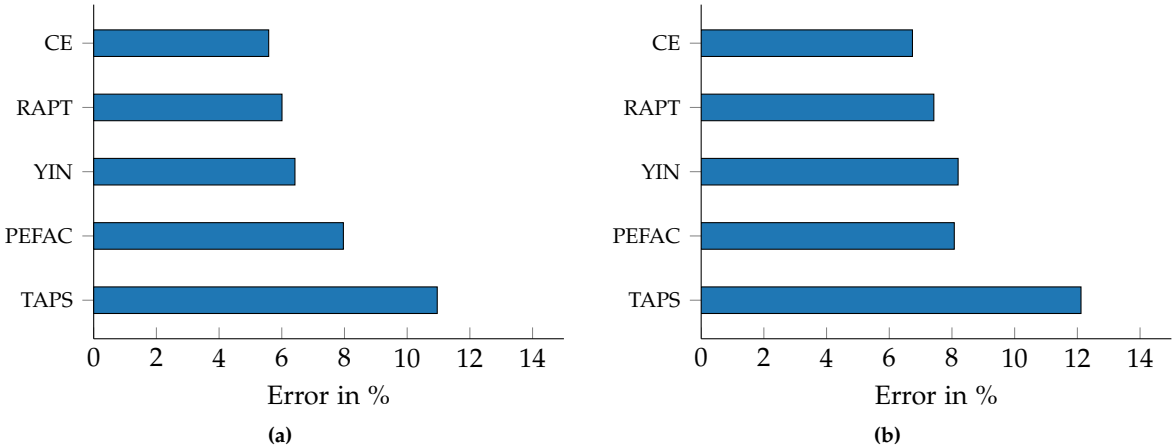

Figure 6.4. Error rates for a) the voiced decision and b) combining the voiced decision and pitch estimation.

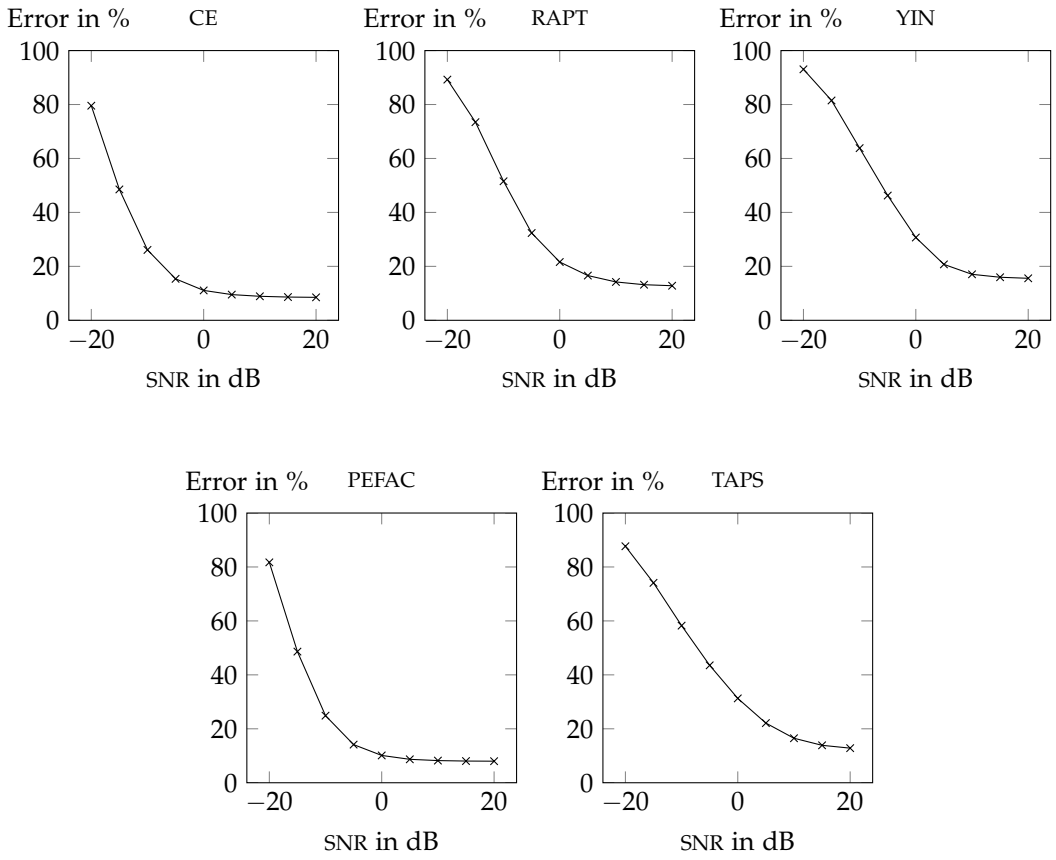

Figure 6.5. Error rates for pitch estimation corrupted by white noise.

needed for TAPS in this error category.

Subfigure 6.4b shows the error for combining pitch estimation and voiced decision. CE performs best with 6.74% incorrectly estimated frames. The error of RAPT is slightly higher with 7.42%. YIN and PEFAC have similar results with 8.19% and 8.07% error, respectively. Clearly the most incorrect estimations are produced by TAPS with an error rate of 12.11%.

Figure 6.5 shows the pitch estimation results for noise corrupted speech. For every algorithm the error rate decreases for an increasing SNR. For SNRs of at least 10 dB every algorithm estimates less than 20% of the voiced frames incorrectly. PEFAC performs best with less then 15% incorrect estimated voiced frames for an SNR of at least -5 dB. For this SNR one third of the pitch estimations of RAPT, YIN, and TAPS are incorrect. The error rate of CE is similar to the error rate of PEFAC, but slightly higher for lower SNRs.

Figure 6.6 shows the pitch estimation error rates for car noise corrupted speech. Best performing for lower SNRs ($\leqslant$ 10 dB) is TAPS. For -10 dB SNR it has an 20% lower error rate than the other algorithms. Without considerint CE (CE benefits from TAPS' performance), the error rate is even 25% lower. The most errors by CE, RAPT, YIN, and PEFAC are caused by estimating the pitch lower than the reference pitch.

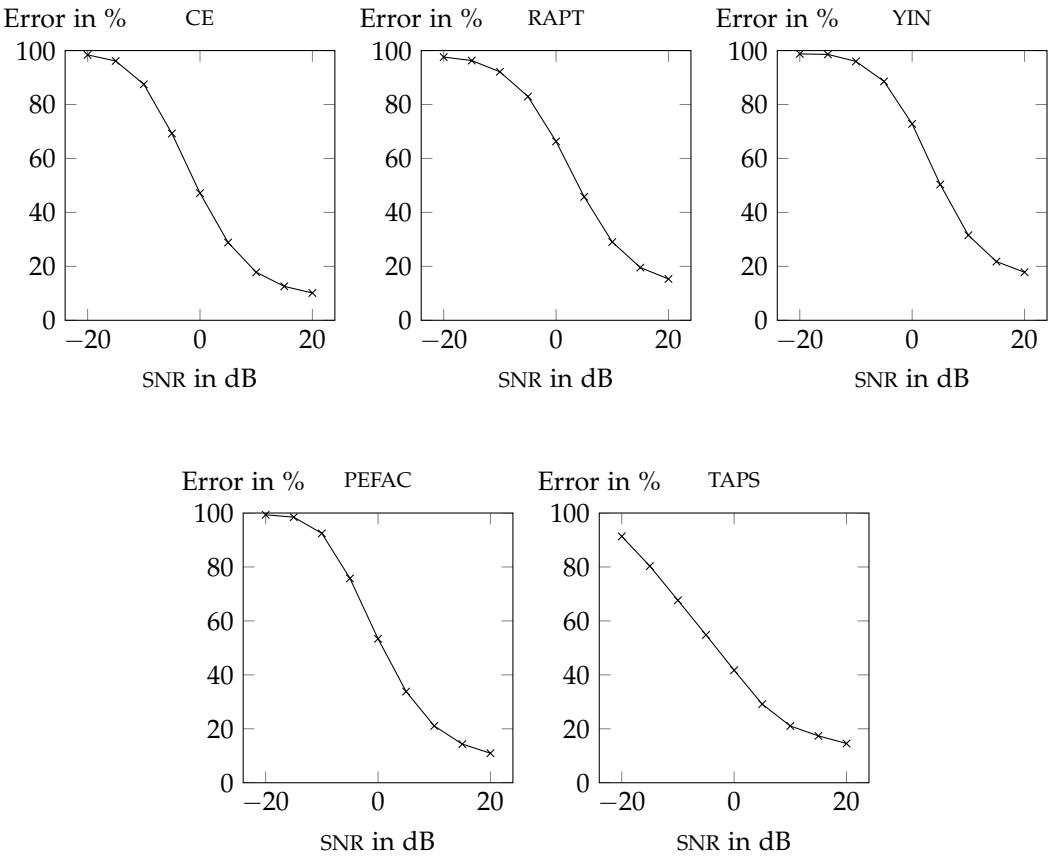

Figure 6.6. Error rates for pitch estimation corrupted by car noise.

The voiced state estimation error for a white noise corrupted speech signal is shown in Figure 6.7. Again the error rate decreases for an increasing SNR. The limiting factor for every algorithm at very low SNRs is that every frame is estimated unvoiced. Since the database contains about 22.47% voiced speech, towards −20 dB every algorithms voiced decision error rate approaches this value. An exception to this is RAPT. Although every voiced frame is estimated unvoiced, there are a few unvoiced frames estimated voiced by RAPT, which enlarges the error by 0.13% at −20 dB SNR. For 0 dB approximately every second voiced frame, which is considered voiced by the algorithm for clean speech, is then estimated unvoiced.

Figure 6.8 finally shows the results for the voiced decision for car noise corrupted speech. Already at 20 dB the error rate is higher than 12% for every algorithm. For YIN and PEFAC it is about 15% higher than for clean speech. For CE, YIN, and PEFAC the error rate increases comparing −20 dB to −15 dB SNR.

6.1.3 Discussion

For clean speech and white noise the pitch estimation by CE does not improve the estimation by PEFAC, which might be caused by the lower performance of TAPS, YIN, and RAPT. Although the pitch estimation

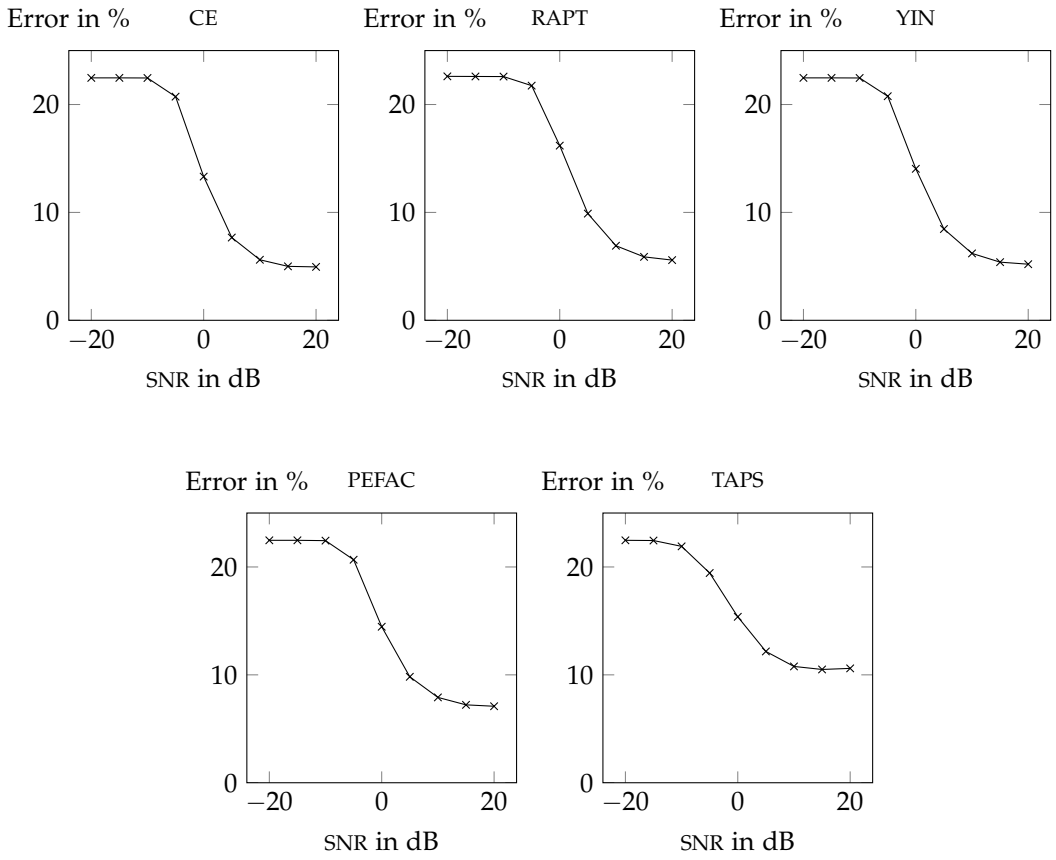

Figure 6.7. Error rates for voiced state estimation corrupted by white noise.

by TAPS is more precise then for RAPT and YIN, it has the highest error rate in the overall estimation due to the higher error rate for the voiced decision. The consideration of multiple frequency spectra might have a negative impact on the precision of onset and offset of voiced segments, which could be a reason for the higher error rate for the voice decision.

In general car noise corrupted speech leads to higher error rates than white noise corrupted speech. This might be caused by the periodicity of car noise. Although the period often changes in adjacent frames, it is reasonable that correlation functions have good values for the period of the car. White noise has no period and, therefore, has a smaller impact on the set of local turning points of correlation functions, which define the estimation made by YIN and RAPT. Car noise also contains harmonics, which influences PEFAC to estimate the frequency of the car noise than F0 of human pitch for lower SNRs.

Since most of the voiced decisions measure the periodicity of the signal, it seems reasonable that algorithms are more likely to consider the signal as unvoiced for lower SNRs of white noise, which has no period due to the equal distribution of intensities over the entire frequency axis. There are also voiced state estimators, which are less related to the periodicity of the signal. In such an use case, it is, therefore, recommendable to configure CE to rely on input by those algorithms instead.

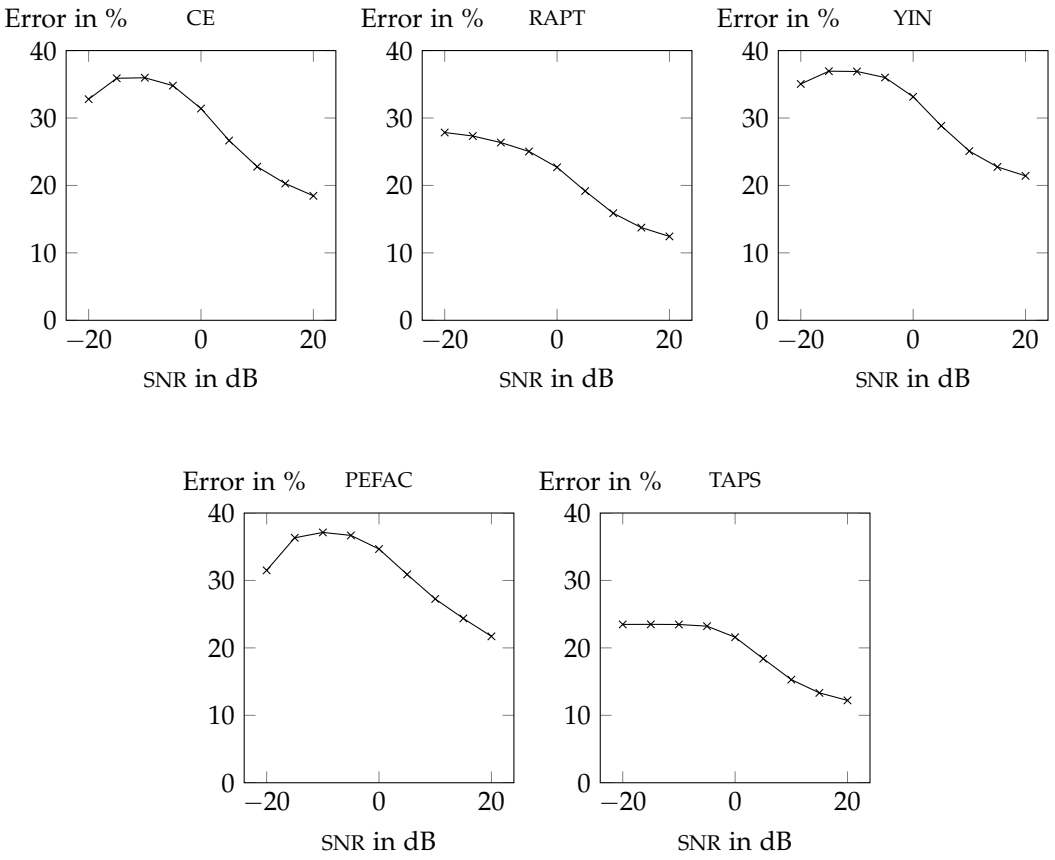

Figure 6.8. Error rates for voiced state corrupted by car noise.

CE does improve the voiced estimation in most scenarios of noise type and SNR. Pitch estimation on clean speech is more precisely estimated by PEFAC and for lower SNRs for white noise PEFAC and for car noise TAPS perform at least slightly better than CE based on all four algorithms. This might be caused by the other algorithms performing worse than PEFAC, which degrades the quality of input estimations giving CE a hard time. Therefore, the configuration of base algorithms producing input for CE has to be designed to suit the application. The flexibility of CE allows such an arbitrary composition of input algorithms even restricting the input to be reduced to a single algorithm, if needed.

Results presented here are not set in stone due to corrupted files and the source of the reference values. On the one hand, the exclusion of corrupted files might not be complete and false references due to recording errors of the laryngograph might still falsify parts of the results. On the other hand, also non corrupted files for which the algorithms fail to estimate correctly might have been ignored. This might have misleadingly improved the results. The second threat to validity is the source of the reference pitch, which is obtained by RAPT. In contrast to the present implementation of RAPT the reference pitch is obtained by RAPT with post processing, which is calculated on the laryngograph instead of the actual speech signal. These differences also explain the estimation differences between the estimations of those two versions of RAPT. Since the reference is also obtained by an algorithm, it

Figure 6.9. Average execution time per algorithm.

is for sure not completely correct. For car noise the evaluation is based on the same segment of 15 s length for every file. This was done for comparability, but limits the generalizability of the evaluation.

6.2 Performance

Regarding the real-time requirement the runtime of the algorithms is of interest. This section presents an experiment evaluating the performance of each algorithm. It is structured similar to the previous section.

6.2.1 Experimental Method

Each algorithm is executed 10^5 times on a signal frame containing zeros. Since the implementation does the complete computation irrespective of the input, the average execution time for such a frame does not differ from the execution time for frames filled with actual speech data. For each call of the process method the execution time (per frame) is measured. Since the first frames might have a higher execution time due to set up operations, the average instead of the maximum execution time per frame is measured. The algorithms are executed on a BeagleBoard-X15 on a single core.

To determine the execution time per frame for each algorithm separately, the system is configured for each algorithm to execute only required modules. Therefore, the minimum module configuration for which the algorithm works correctly, is executed to represent the algorithm.

The input is sampled by a sampling rate of 48 kHz, as already mentioned in Chapter 5, with a frameshift of 256 samples. Therefore, the calculation time an algorithm must not exceed for real-time applicability is

$$\frac{\text{frameshift}}{\text{sample rate}} = \frac{256}{48\text{kHz}} = 5333.\bar{3}\mu\text{s per frame.}$$

6.2.2 Data Analysis

The results are shown in Section 6.2. CE has the largest average execution time with 1574 μs per frame. YIN and PEFAC have similar performance, since both need about 650 μs per frame. TAPS executing in 223 μs per frame on average needs only one third of the time needed by YIN and PEFAC. Fastest is RAPT with an average execution time of 85 μs per frame.

6.2.3 Discussion

Since CE depends on the estimations of the other four algorithms as inputs to work correctly, the whole system is executed. Therefore, the execution time of CE adds up the execution times of the other algorithms with its actual execution time.

RAPT performs best due to the two-pass calculation of NCCF. Without two-pass the execution time would probably be similar to that of YIN, since the computational complexity of underlying correlation functions are similar.

The higher execution time of PEFAC compared to TAPS might be caused by the very high FFT order needed by PEFAC for obtaining a well resolved logarithmic frequency axis.

Every algorithm, even CE, finishes on average within less then a third of the theoretical maximum time per frame. Therefore, they are all suitable for real-time applications performed on a BeagleBoard-X15. The average execution time indicates that the real-time applicability can be extended to various computers.

The experiment only considers the execution time on one embedded board, which is a threat to validity. Some relations of the results might not be universally valid for an arbitrary system.

Part II

Visualization

Chapter 7

Diagram Types

Every visualization must follow a clear syntax and semantics to provide understandability. This chapter briefly describes three different diagram types to build up fundamentals for a discussion of the prospects of code visualization by the following chapters. Two of the types are well known and widely used, namely data flow diagrams described in Section 7.1 and control flow diagrams described in Section 7.2. The third diagram type is called rainbow rails and is developed in this thesis in Section 7.3.

7.1 Data flow

Data flow diagrams represent a "sequencing of a parallel computation by a finite directed graph" and were introduced by Karp and Miller [KM66]. Edges are first-in first-out data queues and each operation is represented by nodes, which can be formalized by a single-valued function [KM66]. They are used for the description and analysis of parallel computations, especially in the digital signal processing domain [LM87].

Figure 7.1 shows an example of a data flow diagram modelling the following rules of *ABRO*, which is the *Hello world!* of synchronous languages. Whenever the reset input *R* is present, the system restarts. If the inputs *A* and *B* both were present at least once since the last (re-)start then the output *O* is true, otherwise it is false. This representation uses Java syntax for the operations, with input and output data queues adjoined at the left and right side, respectively. The *pre* operation obtains the previous value of the input data.

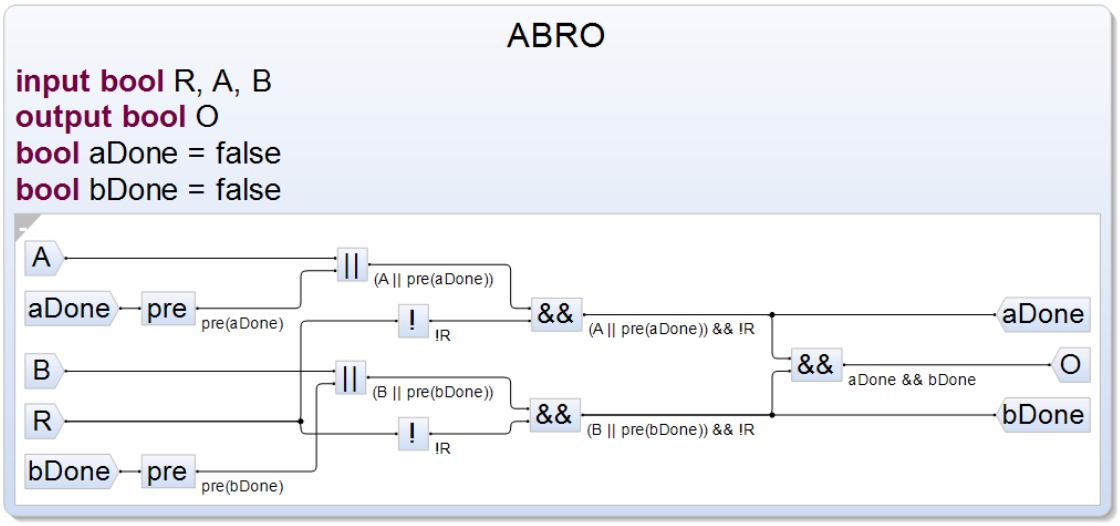

Figure 7.1. Example dataflow diagram representing *ABRO* modelled using KIELER

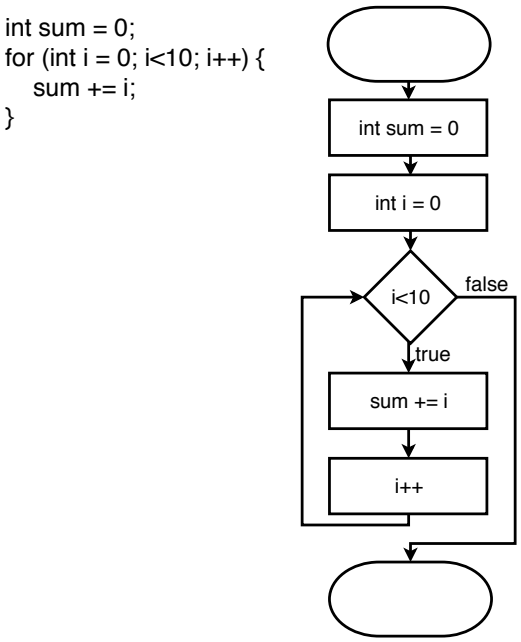

Figure 7.2. Example control flow graph of a for loop.

7.2 Control flow

Control flow graphs were established by Allen [All70], though earlier considered by Prosser [Pro59]. The "flow relationships" [All70] of a program are represented by a directed graph which denotes possible paths to take in the program. In contrast to data flow diagrams, control flow diagrams focus on the program structure and less on the interaction with data. An example of a control flow graph is shown in Figure 7.2.

7.3 Rainbow Rails

Rainbow rails is a new diagram type introduced in this thesis. It is derived from visual representations of the voter migration based on Sankey diagrams [HWE+18] and displays control and data flow elements each in one dimension: control flow is listed vertically and data horizontally. Although several aspects deviate from traditional Sankey diagrams, the idea of single lanes for each unit of data constitutes rainbow rails. The main difference to Sankey diagrams is that not the starting and endpoint are carriers of information, but rather their interaction with blocks representing code indicate the semantics. The lanes are directed vertically and a colour scheme (which motivates the diagram name) serves as a distinction aid in larger diagrams.

The basic blocks for rainbow rails diagrams are shown in Figure 7.3. To visualize operations, semantically associated lines of code are grouped to blocks. The default abstraction level groups blocks of code such as shown in Figure 7.3a into three categories:

Method calls are represented by their name.

Control flow statements are represented by their defining line of code.

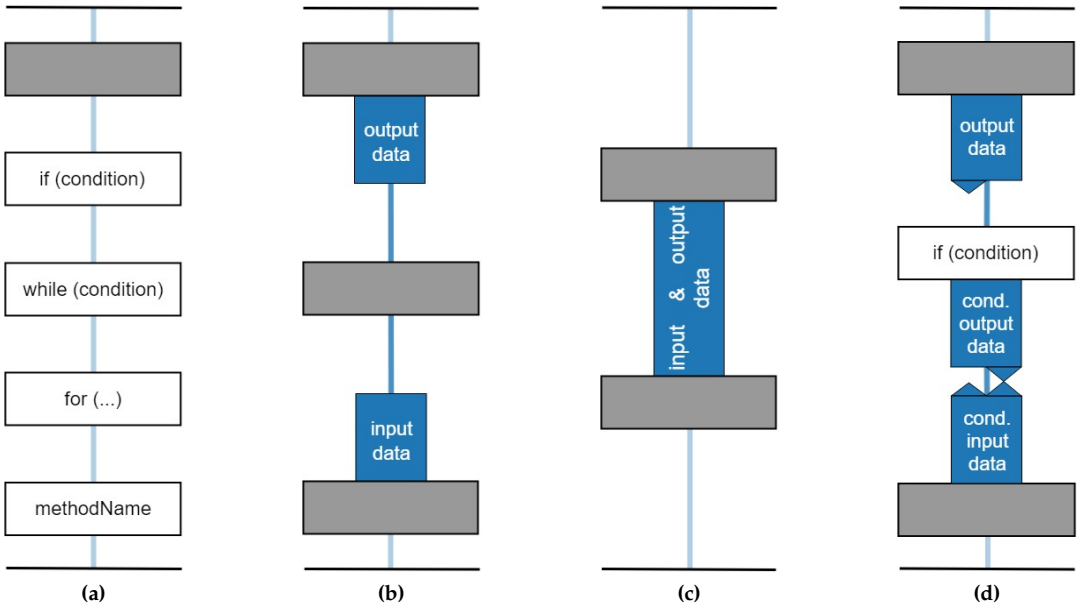

Figure 7.3. Building blocks of a rainbow rails diagram. (a) Computational block types, (b) input and output interaction with a computational block, (c) output interaction directly followed by input interaction, and (d) conditional input/output interaction.

Code abstraction visualizes code not intended to be further specified, such as single instructions.

On an interactive level these blocks can be expanded to visualize more details or collapsed to obtain a higher abstraction level or a more compact representation. The order of blocks from top to bottom is the sequential order of the blocks of code.

A data lane is visualized as a blue vertical line. The corresponding data name can be visualized in each interaction rectangle since the name is most relevant there. Alternatively, it can be displayed on-the-fly next to the lane, which ensures that even when displaying only parts of the diagram each data lane's label is still visible in the viewport. The colour saturation is low to indicate the data interaction state, which is distinguished into active and inactive. An active lane, visualized by a higher colour intensity, indicates that data has been used or changed before and will be used or changed below. An inactive lane, visualized by a lower colour saturation, indicates that the data lane has not yet interacted with a code block or does not interact with a code further below. This can be observed in Subfigure 7.3b. Above the first computational block the data lane is inactive since there was no block interaction yet. Similar the lane is inactive below the third block. Between the first and the third block the lane is active due to the *input interaction* and the *output interaction*. A lane coloured rectangle adjoined to a computational block indicates an interaction of the computational block with the unit of data. If the interaction rectangle adjoins to the upper border of the computational block, then it is used as input, i.e. the value of the data lane is read and influences the calculation of the computational block. If, on the other hand, the interaction rectangle adjoins to the bottom border of the computational block, then the computational block changes the value of the data lane. The interaction rectangle for input and output interactions of adjacent computational blocks are combined to one interaction rectangle as shown in Subfigure 7.3c.

In the usual case, the actual value of a data lane is given by the previous output interaction

rectangle. However, it can happen that it is not directly known whether a block changes a data value as it is the case for assignments in conditional blocks. As visualized in Subfigure 7.3d, the following input rectangle is equipped with input possibilities, visualized as triangles. Each input possibility refers to an output produced by a previous output rectangle denoted by an outgoing triangle at the same horizontal position adjoined to the output rectangle.

Chapter 8

Holistic Views

Starting on the problem of code visualization, this chapter copes with visualizing the code of an entity holistically. One question which directly comes up is what kind of code entity is reasonable to visualize. While for the documentation of one module it is not necessary to represent the whole framework that the module is implemented in, a complete framework representation could be useful for new employee to get started. An entity could also be just one method for example to present an algorithmic idea to colleagues.

Given a code example from the implementation of the first part, this chapter discusses possible approaches to holistic views. While discussing the applicability for different code base scenarios, visual examples stick to representing one specific example code base.

Parts of the implementation of RAPT are visualized in this section to illustrate some of the concepts of code visualization. An excerpt of the main code parts is shown in Figure 8.1. The function computes the first-pass NCCF with the candidates for the second-pass NCCF as output. The module can be used for pitch estimation only using NCCF. In order to restrict the example regarding the code base and the visualizations to a displayable size, functions starting with `sonoNccf`, which are part of the module, are omitted in the code representation.

As already mentioned, representing the whole code always depends on what is considered the whole code. In particular, when using a commonly referenced function such as `printf`, it is very likely that the implementation of `printf` is not reasonable to represent, since it is (most likely) not the main part of the module. The same holds for common functions of a specific framework. The implementations of `SONO_TIC` and `SONO_TOC`, which are used for profiling, have nothing to do with the functionality of the two-pass NCCF, but are used in the implementation. Additionally, functions are often not only used once. Displaying the same information multiple times produces a higher work load than necessary in the process of understanding. On a higher level, RAPT is only one module of the pitch tracking system presented in Part I. Representing the entire system is another possibility of a holistic view. This chapter sticks to representing on a module level since the modules of the system operate independently.

The main distinction between different representations in this work is the extend of interactivity of the diagram. In terms of scalability, readability, and level of abstraction, interactivity has to be considered.

Static representations as described in Section 8.1 require to be unambiguous and complete in the sense of what the purpose of the diagram is. Although it is not likely to find one representation to fit every use case, this section argues about elements of the static representations which might support understanding.

Section 8.2 analyses the aspect of interactivity. Subsection 8.2.1 describes possibilities to interact with the diagram to show desired information or structure the diagram to fit individual needs. In the usual case the amount of information displayed, exceeds the capacity of working memory of the human brain, which is generally approximated as 7 ± 2 [Mil56]. Therefore, interactivity can support condensing information contained at different locations of the diagram. Some possibilities for this are listed in Subsection 8.2.2.

```
1  void __sonoNccfProcess(struct SonoNccf *self, const tfloat *in, tint *cands)
2  {
3    if(!self->active) {
4      sonoMatSetIntToZero(cands, self->numCands);
5      return;
6    }
7
8    SONO_TIC(self->resman);
9
10   // Needed for time smoothing
11   tfloat oldFrequencyEstimation = self->candidate;
12
13   // Setup of input and energy buffers
14   sonoNccfPrepare(self, in);
15
16   // Actual energy value
17   self->energy = self->energyBuffer[0];
18
19   self->maxVal = 0;
20   // NCCF calcuations for lags kMin to kMax corresponding to maximum and minimum frequency.
21   for(tint k = self->kMin; k < self->kMax; k++) {
22     sonoNccfCalcNccf(self, k);
23     self->maxVal = self->nccfVals[k] > self->maxVal ? self->nccfVals[k] : self->maxVal;
24   }
25
26   // Setup of candidates buffer.
27   tint candBufferPos = 0;
28   for(tint k = self->kMin + 1; k < self->kMax - 1; k++) {
29     sonoNccfCheckCand(self, k, &candBufferPos);
30   }
31   for(tint i = candBufferPos; i < self->numCands; i++) {
32     self->cands[i] = 0;
33   }
34
35   // Maximum candidate exraction and candidate output set.
36   self->maxCand = self->kMax - 1;
37   for(tint i = 0; i < candBufferPos; i++) {
38     self->maxCand = self->nccfVals[self->cands[i]] > self->maxVal*self->maxValAcceptanceRange
39         && self->cands[i] < self->maxCand
40         ? self->cands[i] : self->maxCand;
41
42   cands[i] = self->cands[i];
43   }
44
45   // Voiced state decision.
46   self->isVoiced = self->nccfVals[self->maxCand] > self->thresholdVoiced;
```

Figure 8.1. Main code base.

```
47    // Estimation refinement using parabolic interpolation.
48    sonoNccfParabolicInterpolation(self);
49
50    // Candidate conversion from lag to frequency domain.
51    if(self->candidate > 0) {
52      self->candidate = self->sampleRate / self->candidate;
53    }
54
55    // Combining voiced state and estimation.
56    if(!self->isVoiced) {
57      self->candidate = 0.0;
58    }
59
60    // Time smoothing
61    tbool isValidSmoothing = self->candidate >= self->fMin && self->candidate <= self->fMax
62      && oldFrequencyEstimation >= self->fMin
63      && oldFrequencyEstimation <= self->fMax;
64    if(isValidSmoothing) {
65      self->candidate = self->candidate * (1 - self->smoothingConstant) +
66      oldFrequencyEstimation * self->smoothingConstant;
67    }
68
69    SONO_TOC(self->resman);
70  }
```

Figure 8.1. Main code base (continued).

8.1 Static Representations

For the graphical representation to be usable it needs to be understandable and the interpretation needs to be equivalent to the code representation. The usefulness of a graphical representation scales with an easy the information retrieval is easier and the amount of retrievable information. Therefore, the goal is to create a view to make information easier accessible.

To represent every information needed to understand the functionality of the code, a certain amount of text in the diagram inevitable. Consider line 19 of the example code. The number zero has to be somehow contained in the diagram to represent the meaning. Additionally, names of variables might be reasonable to display since they should hint at their purpose and help trace their usage. Therefore, a reasonable question is what kind of text supports understanding and which graphical representations are interpreted equally.

Although a holistic static representation can not influence the amount of information shown at a time, the term *units of information* is worth considering. In a sentence not every letter represents a full unit of information. More likely the meaning of words or even the entire sentence has a work load of one unit of information. What is considered an independent piece of information in this case depends on the individual, but assuming the person has some experience in programming this previous knowledge can be utilized. Consider the control flow representation of a for loop in Figure 7.2. The functionality of a for loop is visualized in detail, although this should be clear to every programmer. The representation consists of six additional arrows, although the one word `for` in a

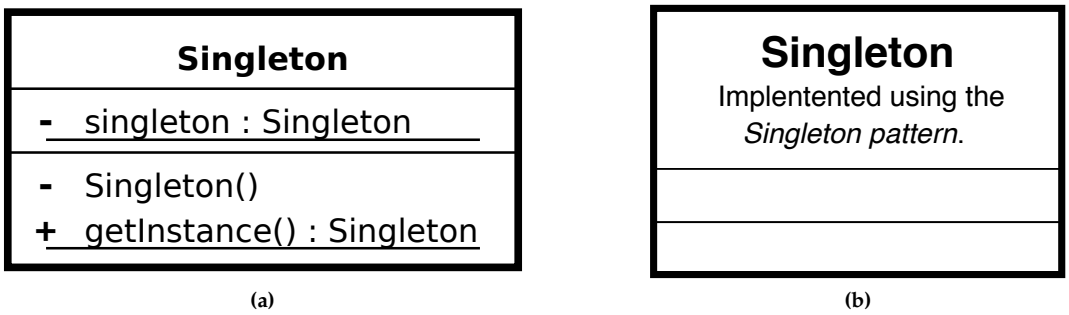

Figure 8.2. Visualization examples of a Singleton pattern. (a) the singleton pattern in UML [Tra06] and (b) a representation abstracting from details of the singleton pattern.

suitable connection with A, B, C, and D would be sufficient for understanding. An adequate choice for the unit of information, which determines the right level of abstraction in a holistic sense, can lead to a diagram which is easier to understand.

In the previous example of the usage of unit of information, the displayed amount of information is reduced. However, not only the reduction of information might help in the process of understanding but also how the information is arranged. On the one hand the diagram should respect common graph aesthetics such as edge crossings or edge lengths. On the other hand, grouping elements can reduce the number of units of information needed for understanding. Multiple iterations in the process of understanding are necessary since the complete interpretation will likely exceed the capacity of the working memory of the brain. Grouping information can make is easier to choose a suitable subset of information to process at a time.

If, for example, a specific programming pattern is used in a represented system, then a box around the modules concerned with the pattern saying "This is pattern X for a functionality called Y." might be easier to understand than processing the meaning of each module separately, especially if the used pattern is known. An example for the singleton pattern is shown in Figure 8.2.

A concrete view for a holistic representation of the example code is shown in Figure 8.3, with a readable extract shown in Figure 8.4. The representation is a combination of aspects from control and data flow diagrams. The name of the represented method and its parameters are shown in the top row. Although the fields of the centric struct *self* are not further specified in the code snippet, it might be useful information and is therefore partly shown in the diagram. The list of fields is not complete since there are many of them and a complete listing is not the main purpose of the example diagram. In an interactive scenario this information toggles between hidden and shown whenever the user clicks the +/- symbol. This symbol is greyed out for the other parameters since these are pointers to floats and have no named fields to show, although in a different approach, information such as the size of the allocated memory could be shown.

Blocks in the centric column refer to the control flow from top to bottom, while the lateral blocks of *self*, *cands*, and *in* refer to data. Arrows between control flow blocks and data blocks indicate the data flow. Changes of the self struct are indicated by a new self block with new information by incoming arrows with the changed components annotated. Unchanged components are taken from the previous self block denoted by a plain arrow. The method might return if the content of the first if-statement is executed, which is denoted by a double rectangle.

The structure of the control flow equals the code structure. Therefore, conditionals and loops add a new hierarchy level containing their code in an additional block. Following the previously

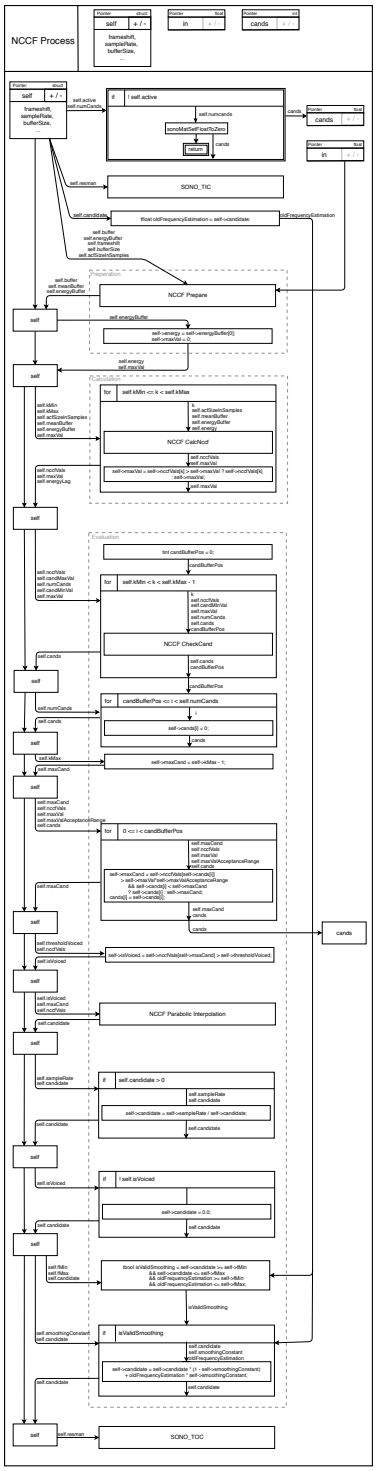

Figure 8.3. A holistic representation for the example code base.

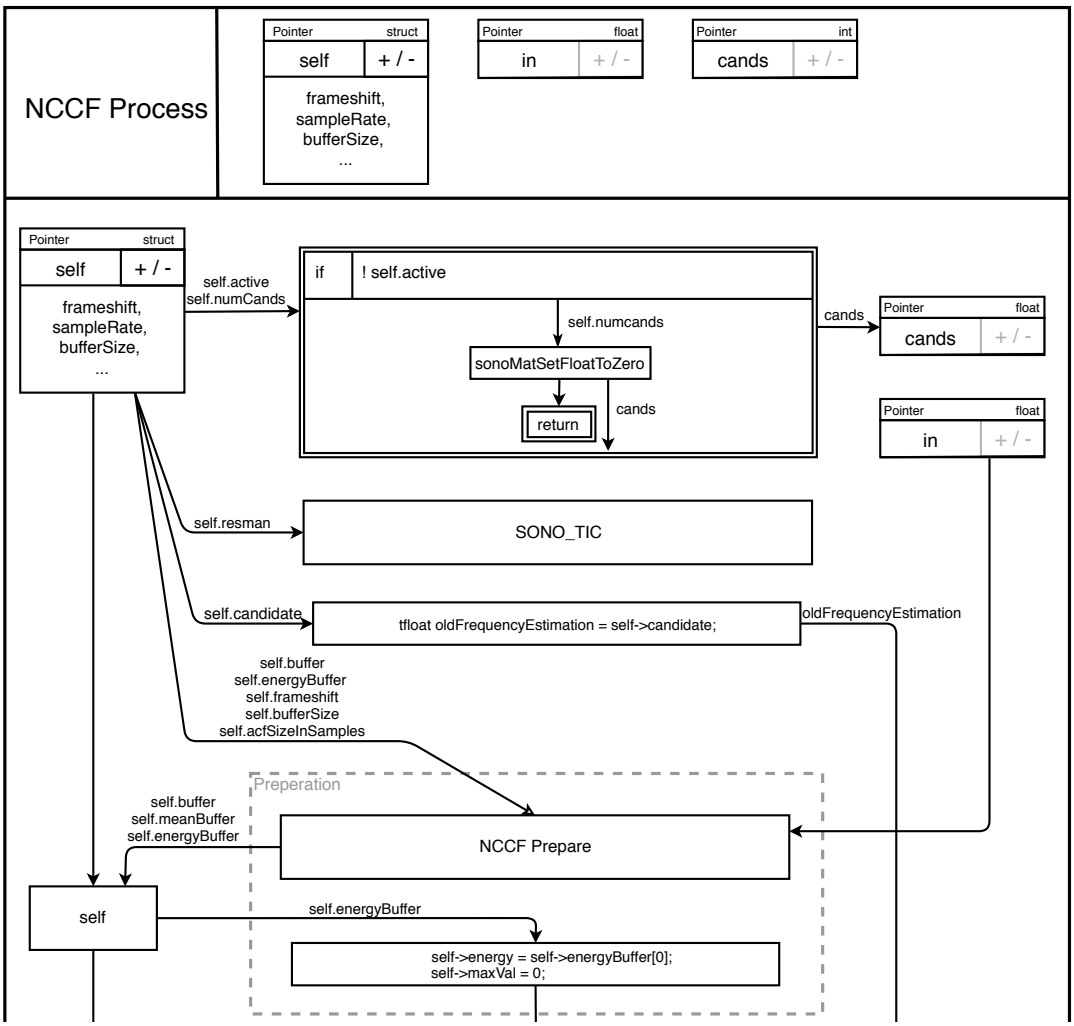

Figure 8.4. Top part of holistic representation.

described units of information for loops, the functionality of loops and conditionals are not visualized. Additionally, method calls are represented in a separate block to visualize their data flow separately. The representation does not differentiate methods defined in the same module from others, although the previous knowledge of imported methods often exceeds the previous knowledge for methods defined in the same module, which could be used for a concept for units of information. Consecutive statements of a form not listed above are grouped and represented in a collecting block.

The size of the diagram exceeds the size of the code if scaled to the same font size. This is not very surprising since not only every piece of information of the textual form is represented, but also boxes and arrows are added for an easier traceability of connected information. Although a retrieval of used variables at a time seems easy due to the input and output lists annotating arrows, retracing the usages of a specific variable requires searching in the column the variable is used.

Components of the self struct are represented with a “self.” prefix. Since most concerned arrows connect a “self” labelled box with a control flow block this reference is already clear and this additional notion not mandatory. Still name clashes can occur on the hierarchy level inside of conditionals and

Figure 8.5. A holistic representation using rainbow rails.

loops. Therefore, this representation sticks to the prefix to provide unique naming in any case.

The grey blocks with dashed lines could be extracted from additional code annotations structuring the code in three phases: preparation, calculation, and evaluation. These additional hierarchical level can support the structuring of the diagram, such that one level in the optimal case does not contain more than 7 units of information.

A second representation in Figure 8.5, with a readable abstract shown in Figure 8.6, uses the structure of rainbow rails with a similar heading row as before. There are two differences to the description in Section 7.3: loop, conditional, and single instruction blocks additionally contain detailed information about the actual code since the diagram is required to preserve a holistic view.

On a horizontal level, the actual data influencing or influenced by a control flow block are listed by adjoining the block. This leads to a similar readability of the interaction between data and control flow as in the first representation. Additionally the development of one data line can be trailed in a vertical view.

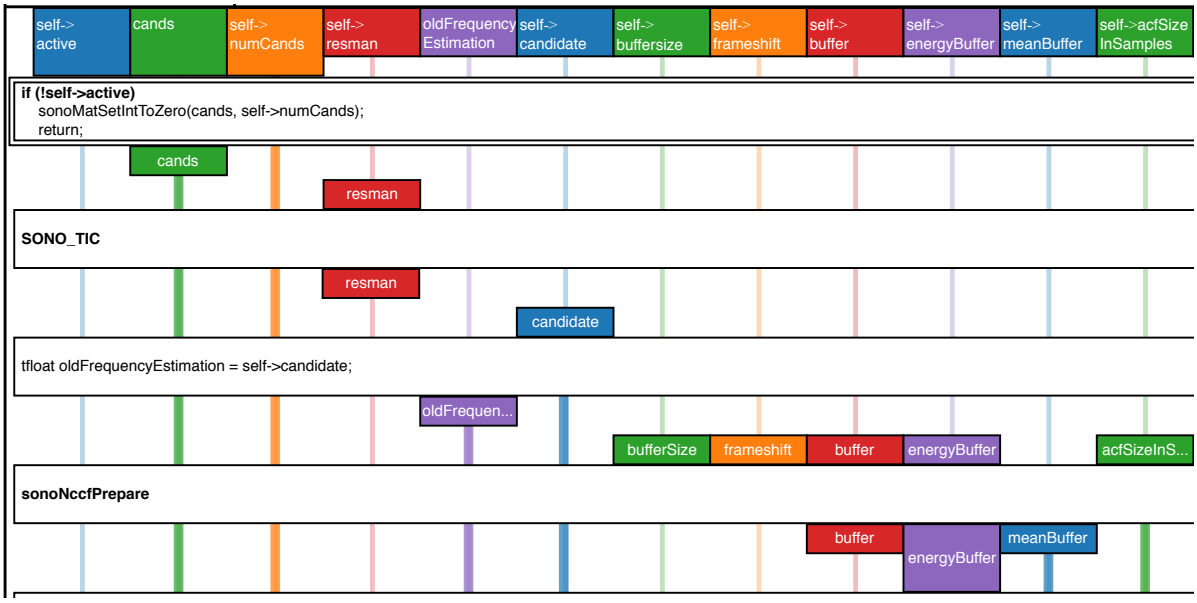

Figure 8.6. Top left part of the holistic representation using rainbow rails. Note how thick rails indicate that a variable is going to be used again in the future, even if the exact point of use is not currently visible.

8.2 Interactive Representations

Although the amount of shown information is to some degree predefined for holistic views, still there are possibilities for the interactivity. Subsection 8.2.1 discusses possibilities of the level of interactivity, ranging from highlighting to modifying. Representations of sufficient size can be supported by navigation interactions, which are discussed in the second subsection. For each interaction a scenario of usability is given as a motivation.

On the one hand, some possibilities of interactivity might not be reasonable for every diagram type. On the other hand, the applicability of listed interactions is not restricted to one specific diagram type. Therefore, a concrete implementation first needs to choose a diagram layout before reasonable methods of interactivity can be chosen.

8.2.1 Levels of Interactivity

The diagram could be used for debugging. If, for example, the input buffer contains values differing from expectations, the user might want restrict the area of search to the code influencing the buffer. In a different scenario, the user might want to look up every usage of the buffer to understand what it is used for. In both cases highlighting every occurrence of the buffer in the diagram can support the efforts of the user.

To understand the role of a method for the module it might be of interest where the method is used. Therefore, highlighting of every occurrence of the method in the diagram can support the users efforts.

Due to the highlighting of the code, the user in the debugging scenario might have at this point located the bug in the diagram. The user might then search for the according code in the text or click on the diagram element to be redirected to the according code equivalent. An interactive possibility to

redirect in both directions between code and diagram can preserve the user from tiresome searching in a multitude of scenarios.

Regarding an increasing size of the diagram, the diagram will not always fit the screen or paper in a readable manner. Therefore, zooming and focussing on parts of the diagram is necessary for larger diagrams. To aid the overview of the diagram when focussing on a specific element, a possibility is to enlarge elements in context to the specific element.

While the represented information in terms of code must not be changed in a holistic view, there might be additional information in the form of comments to display on demand. These might explain bounds of a *for-loop* or be the Javadoc for a method. In the framework for the implementations of Part I an explanation for each component of the self struct as it is used in the example code might be given in the corresponding YAML. Instead of searching or inferring such information the user could just hover over an element and get such additional information, if present.

Similar to displaying comments in the diagram, also adding additional comments through the diagram might support the user's efforts in understanding or documenting. This could find a practical usage for poorly documented legacy code.

Additional structuring of the code can also improve the understanding of a diagram. In Figure 8.3 dashed boxes indicate three phases: the method mainly consists of a preparation, a calculation and an evaluation phase. Every pitch tracker of Part I can be partitioned in these three phases for the estimation. A documentation of these could benefit from such a classification for every pitch tracker since new information is easier to understand if it is presented in a known form or structure.

At a very high level of interactivity the user could modify the code through the diagram for example by adding blocks of loops or conditionals. Although manual editing of larger diagrams tends to be a tedious task, a sophisticated system might be a helpful tool.

8.2.2 Navigating Through the Diagram

In larger diagrams one might easily lose track of element placement. A functionality for searching elements by name, type, or connection to another element can be useful. Additionally, overviews listing whatever seems to be interesting—such as all buffers with their YAML description—can help the user by finding the desired information. A searched element could then get a focus in the diagram by highlighting or enlarging to support the navigation.

The connections to other modules used by or using the module can sometimes require to display their functionality for a complete understanding. Displaying those information per se could lead to much superfluous information, which would complicate the understanding and information retrieval.

A focus and zoom depending on the element of current interest can support navigation. For example when clicking on a block the block could be centred and the zoom level adapted for readability of the contained text or when the element of interest is an arrow for data flow then again centring and zooming to show connected blocks seems reasonable.

In the case of live code editing, the diagram could automatically zoom or focus on the concerned elements to serve as a cross reference for whether the changes do what they are expected to.

Chapter 9

Focussing on Details

In contrast to the previous chapter, which discussed holistic representations from which the exact code base could be reproduced, this chapter sets the focus on a subset of the code to represent. This can reduce the units of information needed to understand since there exist situations where the understanding does not require all information. Thus a reduction of available information prevents from processing unrelated information. The importance of reducing the amount of information needed for understanding can again be explained by the limited capacity fo the human brain.

The freedom of displaying only a subset of the available information leads to the new question about which subset of information is reasonable to represent. As usual, there likely is not one answer optimal for every situation. However, Section 9.1 lists some reasonable sets of information for some scenarios. Similar to the previous chapter, Section 9.2 and Section 9.3 discuss static and interactive representations, respectively.

9.1 Relevant Information

Depending on the intention the subset of code considered relevant differs. There does not exist one type of subset optimal for every use case and also there exist approximately as many approaches as there are different scenarios for diagram usage. This section lists some partial views on the code base given in Chapter 8 from different perspectives.

The first approach shown in Figure 9.1 is close to a holistic view but abstracts instructions which are no method calls, loops, and conditionals. Additionally, code inside of loops or conditionals is not shown. This preserves a basic overview of the represented code and interactive representations could add a possibility to show the omitted information if the user wants to see them. The diagram visualises which data is read or written in which part of the control flow and when it was previously used or which parts of the control flow will use the written data. Data read by a control flow block connects to the left border and data written connects to the right side of the block. If data was not modified before in the shown part, then the connection comes from the left border of the global rectangle for the whole diagram, which is compatible with multiple hierarchies. If written data is not further used in the displayed diagram, then the connection goes to the right border of the global rectangle. For the example code this is the case if the written data is necessary for the next iteration of the process method when the next estimation has to be done.

A second approach shown in Figure 9.2 uses a user defined abstraction to provide a clear first impression or a very abstracted view on the method. Therefore, control flow and data usage are restricted to the three phases of preparation, calculation, and evaluation, which could be added manually by the developer for documentation or it could be generated automatically from annotations in the code base. In an interactive scenario this could be the starting point of exploration with a less overwhelming size than the previous approaches, especially if this is transferred to larger code bases. Additionally, used constants are listed as one input for which an interactive representation should provide a possibility to show each constant as a separate input.

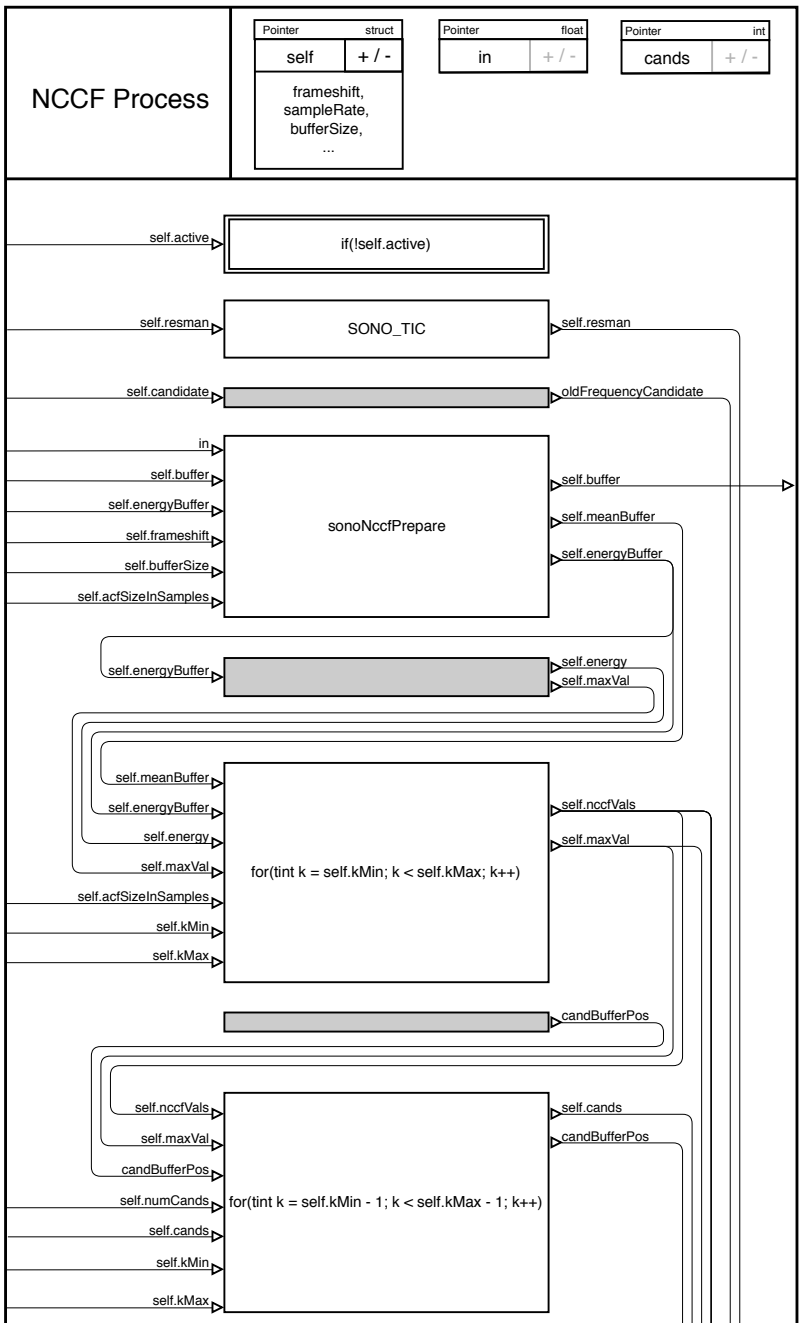

Figure 9.1. A holistic view for the example code.

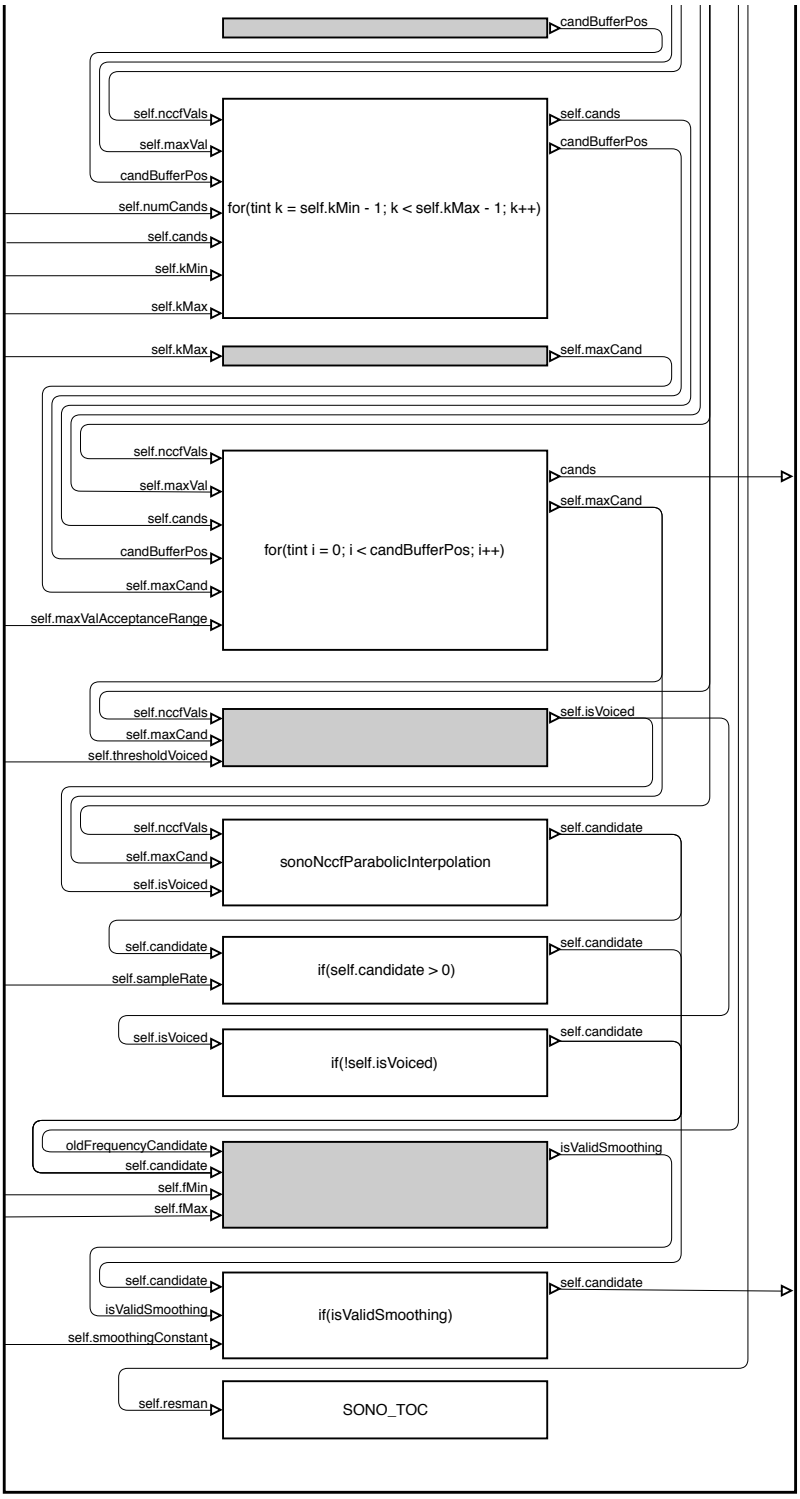

Figure 9.1. A holistic view for the example code (continued).

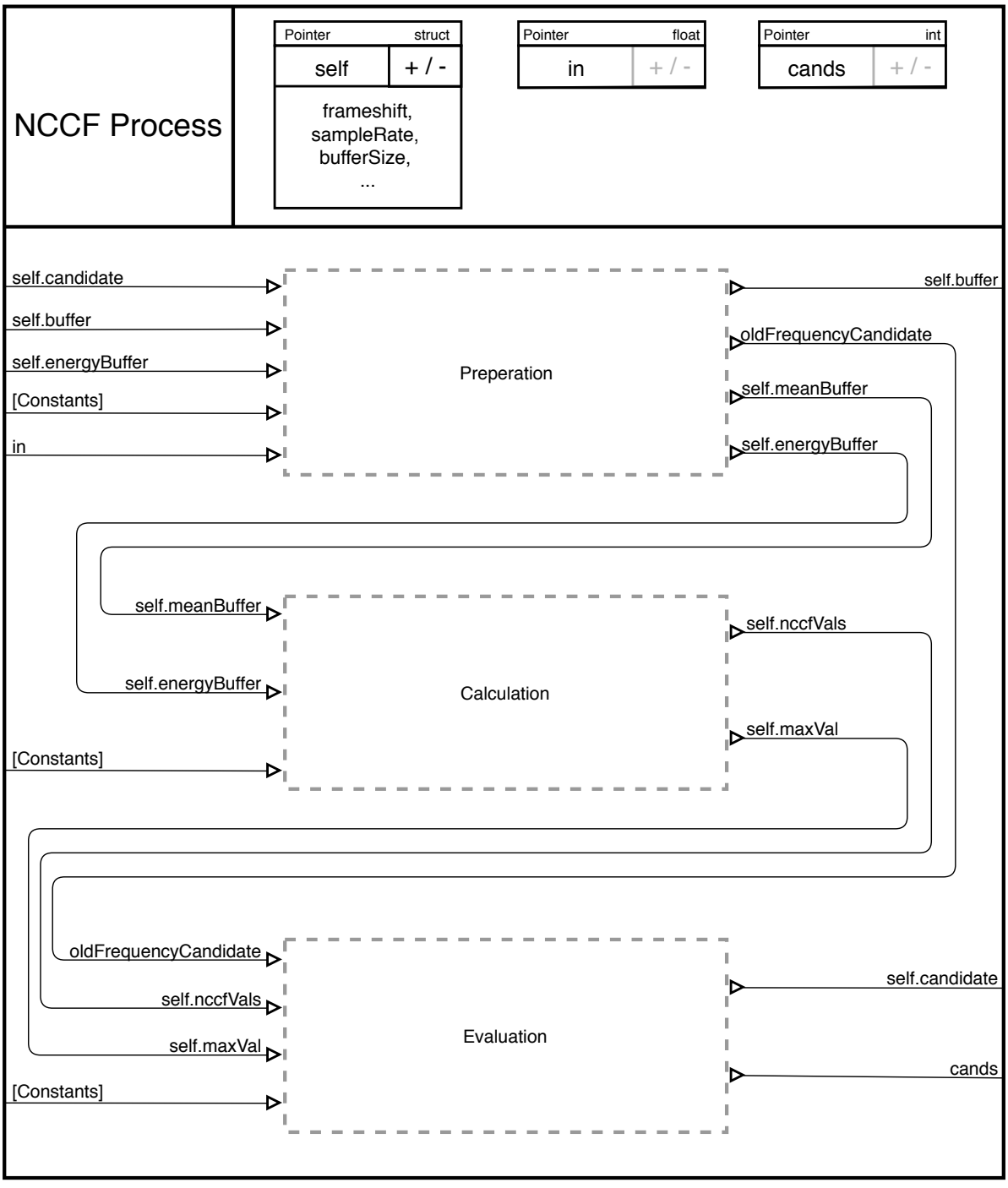

Figure 9.2. Simplified diagram.

A third possibility of an information subset is shown in Figure 9.3. In principle, the diagram is similar to Figure 8.5 (the holistic rainbow rails view), but is restricted to three of the data lanes. The control flow blocks list the complete control flow, although in this case it seems reasonable to abstract from control flow not in contact with any of the displayed data lanes. This could be done through the usage of the grey code abstraction blocks described in Section 7.3.

9.2 Static Representations

A static representation of a subset of information comes in handy for the documentation of a specific view on a code segment. Whenever someone wants to document the usage of a method, a new algorithmic idea, or the usage of a specific variable throughout the program, such a specific view can be supportive.

However, this actually is not a new scenario of representing code. In each of the previously listed use cases one could define a fitting subset of the code to represent a holistic view for. Therefore, finding a static representation for a subset of code is not different from finding a holistic view for the subset, then regarded as the whole code base. Any use case in this category is solvable in the two steps: find a fitting subset of code and represent it holistically.

9.3 Interactive Representations

Readability and understandability of a diagram are enhanced if exactly the information, which the user is searching for, is shown. Interactivity is a handy tool to achieve such a view since the user probably knows best which information is useful and which would only irritate or deviate.

The methods of interactivity in this chapter are nothing but a more general case of the interactions from the previous chapter. Therefore, all of the methods described there could also be used in a more detailed view. Additionally to the possibilities of user interaction already described, this section lists some interactions based on changing the amount of represented code.

9.3.1 Levels of Interactivity

Interactivity in the context of partial representations offers an opportunity of a pseudo-holistic view without overwhelming the user with displaying every information directly at the start. The information is theoretically fully accessible by interaction with the diagram but the first impression is restricted to give an overview. Figure 9.2 already showed an abstracted overview of the example code. Holistic views scale with the size of the reference code. In Chapter 8, I briefly argued why a holistic representation of the complete pitch tracking system does not seem to be reasonable. However, this is different for interactive partial representations. From this point of view, Figure 5.2 could already be a view on the complete system with the opportunity to go into details for a module by clicking on the corresponding rectangle.

Hierarchy is an important tool for structuring, which can be used in different ways for clear representations. As described in Section 8.2.1 manually added structuring blocks can improve readability and as it is done in Figure 9.2 these blocks can be used for hierarchical structuring of the diagram. Another example for this are method calls, which were already abstracted to their name. In an interactive mode adding and removing details, such as an equivalent diagram for the method definition inside of the method call block, can be useful.

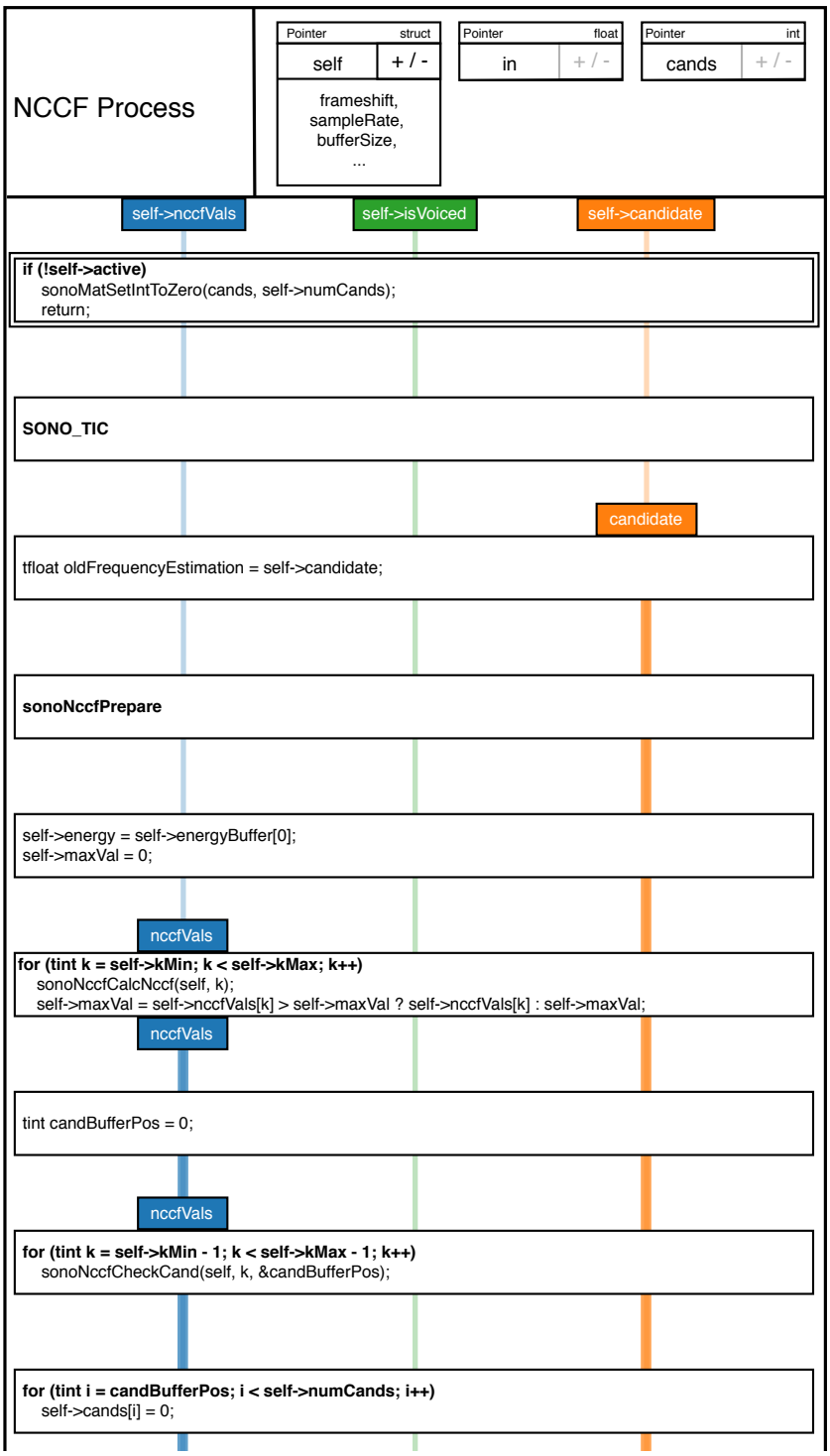

Figure 9.3. A rainbow rails representation for a subset of data lanes.

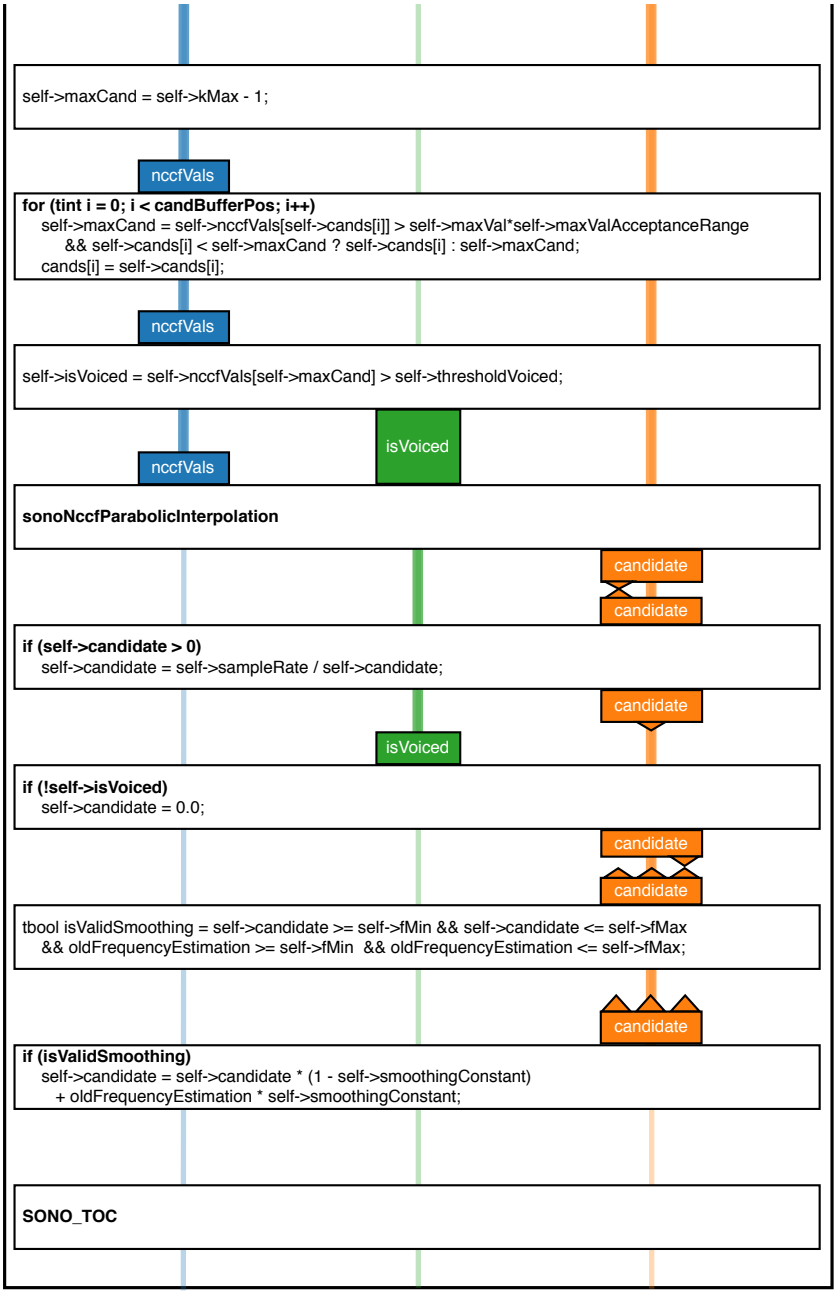

Figure 9.3. A rainbow rails representation for a subset of data lanes (continued).

Not only the control flow can be hierarchically grouped to produce a clear first representation but also the data. Though this seems more reasonable if more than one struct or object is used, components of a hierarchical structure in the program can be grouped in the diagram and shown separately if intended by the user.

Figure 9.3 showed the rainbow rails representations for only three of the data lanes. As already mentioned in the corresponding section, it seems reasonable to abstract from the control flow not influenced by or influencing the data lanes since they are not the main matter of interest. This is similar to unmentioned data in Figure 9.2, which might be used inside of the preparation, calculation, or evaluation but are not shown since they do not interact on the level of the three phases.

9.3.2 Navigating Through the Diagram

In the previous chapter about holistic views the equally named subsection is motivated by the holistic diagrams tending to be large and the user might lose track in such a detailed diagram. The problem of losing track in a diagram too large for a readable representation on the screen is reduced by hierarchical structuring. The possibility of hiding unrelated information reduces the diagram size and therefore the problem value. Still the user might get lost in the levels of hierarchy or if there are just too many blocks opened showing their details. Therefore, a possibility to reset the diagram to the default representation can support navigation.

We already discussed methods to compare the actual code to the diagram representation. In the case of only partially represented code, the comparing code base could show the not represented code in grey to make the comparison easier. This could help the user by the correlation between the visual and textual representation, since in case further editing of the code is intended, the user has to navigate through the code just as well as understanding the procedures.

Chapter 10

Conclusion

As a conclusion to this thesis, this chapter first summarises the results in Section 10.1. Finally, Section 10.2 discusses possible future work to enhance and evaluate the presented solution.

10.1 Summary

This work has the two emphases pitch tracking and code visualization. For the discussion of pitch tracking first foundations of speech production, digital processing and estimation refinement are provided.

The pitch tracking algorithms RAPT, YIN, TAPS, and PEFAC were described and implemented for this work, chosen from a large pool of existing pitch trackers. An algorithm to combine the estimations from different pitch trackers based on the continuity of the estimation, main and secondary candidates, and on a scoring system also considering the harmonics and subharmonics of the estimation was described for the four implemented algorithms.

Using the Pitch Tracking Database from Graz University of Technology the pitch tracking algorithms were evaluated regarding accuracy, both with and without background noise. Overall PEFAC performed best in the task of pitch estimation, although the combining algorithm could improve the results when combing pitch estimation with a voicing state determination.

Rainbow rails are a new diagram type introduced for the purpose of code visualization. Together with data flow and control flow graphs, representations using rainbow rails were discussed. While holistic views have the advantage of delivering every piece of information at once, in a greater scale they tend to overwhelm the user. In a more focussed and restricted approach grouped and abstracted elements create a clear overview and with the use of interactive elements all information remain available.

10.2 Future Work

Although the implemented algorithms were already chosen by best performance and most diversity in the approaches, there exists such a great amount of pitch tracking approaches not yet used in the pitch tracking system. Adding further approaches—such as statistical models and neural networks—could further enhance the pitch estimation quality. Since pitch as the perception of tone is not a calculable quantity, also an approach estimating the actual pitch instead of F0 could be implemented in order to further improve the estimations.

LTASS normalization serves in the PEFAC algorithm for the reduction of noise or equally the enhancement of pitch related frequencies. There seems to be no obvious reason for not applying this method to an arbitrary frequency domain pitch tracking algorithm. Performing LTASS normalization before the peak extraction would be an easy extension of the TAPS algorithm.

Another obvious combination of two algorithms works the other way around. The usage of a temporally accumulated spectrum could also be used for PEFAC, though the advantages from the accumulation are greater for peaks than for the complete spectra. The use of the accumulated peak spectrum then transferred to the logarithmic frequency axis and filtered could be prone to small changes in the peak choice and could therefore be less robust. In that case frequency smoothing on the accumulated peak spectrum could improve stability of the filter. A possible order of executed methods could be: convert to the logarithmic spectrum, then apply LTASS normalization on each spectrum, select the peaks and accumulate the spectrum, apply frequency smoothing, apply the filter, and finally choose the global maximum of the filtering result.

When it comes to code visualization this work only lists possible forms, but they have yet to be implemented. Also a study to compare different approaches is important to evaluate them. In an opinion poll with participants both with academical and economical background (to gain more general opinions) the elements of visualization could be rated regarding their usefulness. With an implementation also the qualitative advantages of the usefulness could be tested by giving the participants the task to extract a specific piece of information from a diagram or adding a new functionality to the program and measuring the time needed. Additionally, the participants could be asked for one integral element of code visualization they think is indispensable.

Appendix A

System Parameters

A full list of configurable parameters is given here. These are grouped by the algorithm they belong to. For each parameter the name in the implementation, a short description, and the value used for the evaluation are listed.

There are some parameters common to all modules. The `enabled` parameter determines whether or not to fully initialize the module, while the `active` parameter determines whether to execute the full process method or a fallback behaviour. Since every module is executing in the normal case, these are `true` per default. `fMin` and `fMax` determine the estimation range of each algorithm.

Table A.1. System parameters.

Name	Description	Value
`YIN`		
`allowedSpikes`	Number of spikes in the passed `localEstimationRange` estimations that are allowed to still consider the previous estimations as continuous.	`2`
`continuityRange`	Used for time smoothing, this defines a range for which the previous estimations are considered continuous.	`0.3`
`continuitySpike`	Used for time smoothing, this defines a relative threshold for the new estimation to be considered continuous.	`0.4`
`isOldEstimationActive`	Boolean whether to apply the *Local Estimation Refinement* step of the algorithm.	`true`
`localEstimationRange`	Number of old estimations considered in the decision whether to apply the forced continuity.	`6`
`numCands`	Size of candidate buffer and therefore an upper limit on the maximum number of secondary candidates.	`5`
`smoothingConstant`	Time smoothing constant. This weights the impact of the previous estimation and its value is between 0.0 and 1.0.	`0.0`
`threshold`	Threshold for the *Absolute Threshold* step of YIN	`0.3`
`thresholdVoiced`	Threshold for the voiced decision. The frame is estimated voiced if, and only if, the global minimum is lower than this threshold.	`0.4`

Table A.1. System parameters (continued).

Name	Description	Value
TAPS		
candidateSelection	TAPS' evaluation strategy to be used.	peak
freqSmoothingConstant	Amount of frequency smoothing applied in dB per kHz.	280
harmonicsSearchRange	Range for peak detection of harmonics for estimation refinement in the peaks based approach.	3
numAveragedHarmonics	Number of harmonics considered in the original approach.	2
numCands	Size of candidate buffer, and therefore an upper limit on the maximum number of secondary candidates.	5
numHarmonics	Number of considered harmonics for the F0 score in the F0 bin based approach.	10
numHarmonicsSearched	Number of considered harmonics in the estimation refinement in the peaks based approach.	4
numMax	Number of highest maxima of the ACF used for further calculations.	10
peakThreshold	A relative (to the maximum) threshold a peak must pass in the peak based approach.	0.0001
thresholdCands	Threshold for secondary candidates in all strategies except for the peak based approach.	0.1
thresholdHarmonics	Relative (to the maximum) threshold that a harmonics has to pass to be taken into account in the harmonics based approach.	0.2
voicedThreshold	Relative (to the mean) threshold that the highest maximum has to exceed for a frame to be estimated as voiced.	8.5
RAPT		
candMinVal	Minimum NCCF value of the candidates of the first-pass NCCF.	0.3
maxValAcceptanceRange	Threshold for choosing the main candidate. The lowest lag with a local NCCF maximum and an NCCF value greater than this threshold times the global maximum is chosen.	0.9
numCands	Size of candidate buffer and therefore an upper limit on the maximum number of secondary candidates.	10

Table A.1. System parameters (continued).

Name	Description	Value
numTriesPerCand	Vicinity size in lags per candidate in the estimation refinement in the second-pass of NCCF.	9
smoothingConstant	Time smoothing constant. Between 0.0 and 1.0 weights the consideration of the previous estimation.	0.0
thresholdVoiced	Threshold that the global NCCF maximum needs to exceed for a frame to be estimated as voiced.	0.7
PEFAC		
activeAC	Whether or not to compute the amplitude compression.	true
analysisWindowLength	Analysis window length size of FFT in ms.	90
fftOrderSec	FFT size in seconds.	0.36
fMaxCent	Maximum frequency of the logarithmic axis computed by the centbank in Hz.	4000
freqSmoothingConstant	Amount of frequency smoothing of the spectrum to obtain the smoothed spectrum.	0.2
gamma	Defines the peak width for the filter.	1.8
numCands	Size of candidate buffer and therefore an upper limit on the maximum number of secondary candidates.	5
numHarmonicalPeaks	Number of peaks referring to harmonics in the filter.	6
regThreshold	Relative (to the mean) threshold for the voiced decision.	25
timeSmoothingConstant	Amount of time smoothing of the spectrum to obtain the smoothed spectrum.	10
CE		
binWidth	The bin size named K in the according section.	10
continuityAcceptanceRange	Vicinity size of the previous estimation for bins considered continuous to the previous estimation in Hz.	20
continuityWeight	Weight for continuous bins.	20
nccfCandsWeight	Weight of secondary candidates for RAPT.	1.5
nccfEstimationCalculationWeight	Consideration weight of the RAPT's candidate for the weighted mean based estimation calculation.	1
nccfMainCandWeight	Weight of the main candidate for RAPT.	5

Table A.1. System parameters (continued).

Name	Description	Value
`numConsideredHarmonics`	Number of harmonics of which the secondary candidates influence the bin value.	`4`
`pefacCandsWeight`	Weight of secondary candidates for PEFAC.	`1`
`pefacEstimationCalculationWeight`	Consideration weight of the PEFAC's candidate for the weighted mean based estimation calculation.	`1.5`
`pefacMainCandWeight`	Weight of the main candidate for PEFAC.	`8`
`tapsCandsWeight`	Weight of secondary candidates for TAPS. Remember that this is negative since TAPS' secondary candidates refer to harmonics and not similar the other algorithms secondary candidates to subharmonics.	-1
`tapsEstimationCalculationWeight`	Consideration weight of the TAPS's candidate for the weighted mean based estimation calculation.	`0.5`
`tapsMainCandWeight`	Weight of the main candidate for TAPS.	`5`
`voicedCountNeeded`	Number of voice counts needed for a decision to be voiced.	`1`
`yinCandsWeight`	Weight of secondary candidates for YIN.	`1.5`
`yinEstimationCalculationWeight`	Consideration weight of the YIN's candidate for the weighted mean based estimation calculation.	`1.2`
`yinMainCandWeight`	Weight of the main candidate for YIN.	`5`

Appendix B

Results of the Estimation Quality Evaluation

		Pitch Estimation			Voiced Decision		
	Combined	Total	Too high	Too low	Total	Voiced	Unvoiced
CE	6.74	8.46	6.29	2.17	5.58	3.66	1.91
RAPT	7.43	12.56	6.20	6.35	6.00	3.52	2.47
YIN	8.19	15.20	7.37	7.83	6.42	4.10	2.32
PEFAC	8.07	7.96	5.45	2.51	7.38	2.08	5.30
TAPS-r	12.78	17.09	–	–	10.95	4.99	4.95
TAPS-h	14.23	37.59	–	–	10.95	4.99	4.95
TAPS-f	16.51	35.55	–	–	10.95	4.99	4.95
TAPS-p	12.11	12.26	10.27	1.99	10.95	4.99	4.95

Table B.1. Error for clean speech

		Pitch Estimation			Voiced Decision		
SNR in dB	Combined	Total	Too high	Too low	Total	Voiced	Unvoiced
White Noise CE							
−20	22.47	79.56	23.41	56.15	22.47	0.00	22.47
−15	22.47	48.52	16.62	31.90	22.47	0.00	22.47
−10	22.46	26.09	11.48	14.61	22.46	0.00	22.46
−5	20.76	15.37	8.79	6.57	20.73	0.04	20.69
0	13.53	11.08	7.55	3.52	13.32	0.29	13.02
5	8.19	9.53	7.05	2.47	7.67	0.89	6.78
10	6.40	8.90	6.72	2.18	5.61	1.58	4.03
15	5.97	8.62	6.51	2.10	5.00	2.19	2.80
20	6.02	8.51	6.38	2.12	4.95	2.67	2.27

Table B.2. Error for noise corrupted speech.

		Pitch Estimation			Voiced Decision		
SNR in dB	Combined	Total	Too high	Too low	Total	Voiced	Unvoiced
White Noise RAPT							
−20	22.61	89.24	34.40	54.84	22.61	0.13	22.47
−15	22.60	73.46	27.63	45.83	22.60	0.13	22.47
−10	22.59	51.55	18.91	32.64	22.59	0.12	22.47
−5	21.77	32.36	12.93	19.43	21.75	0.11	21.64
0	16.38	21.61	10.15	11.45	16.19	0.25	15.93
5	10.37	16.59	8.62	7.96	9.89	0.72	9.16
10	7.68	14.22	7.62	6.60	6.91	1.37	5.53
15	6.89	13.20	7.04	6.16	5.88	2.00	3.87
20	6.77	12.82	6.72	6.09	5.58	2.50	3.08
White Noise YIN							
−20	22.47	93.05	24.04	69.01	22.47	0.00	22.47
−15	22.47	81.50	19.79	61.71	22.47	0.00	22.47
−10	22.46	63.83	13.92	49.90	22.46	0.00	22.46
−5	21.03	46.25	9.85	36.40	20.77	0.04	20.73
0	14.88	30.69	8.20	22.49	14.04	0.31	13.72
5	9.50	20.73	7.94	12.79	8.46	0.89	7.56
10	7.44	17.05	7.88	9.17	6.20	1.51	4.68
15	6.88	15.96	7.91	8.05	5.39	2.04	3.35
20	6.83	15.57	7.86	7.71	5.20	2.44	2.75
White Noise TAPS							
−20	22.47	87.68	8.47	79.21	22.47	0.00	22.47
−15	22.45	74.11	7.07	67.03	22.45	0.00	22.45
−10	21.92	58.25	6.59	51.65	21.91	0.01	21.90
−5	19.51	43.53	7.05	36.48	19.44	0.07	19.26
0	15.66	31.28	8.07	23.20	15.38	0.74	14.63
5	12.79	22.12	9.06	13.06	12.17	1.76	10.40
10	11.67	16.50	9.62	6.88	10.78	2.88	7.90
15	11.53	13.88	9.89	3.98	10.50	3.77	6.72
20	11.71	12.83	10.04	2.79	10.60	6.24	4.36

Table B.2. Error for noise corrupted speech.

		Pitch Estimation			Voiced Decision		
SNR in dB	Combined	Total	Too high	Too low	Total	Voiced	Unvoiced
White Noise PEFAC							
−20	22.47	81.70	15.12	66.58	22.47	0.00	22.47
−15	22.47	48.62	10.90	37.71	22.47	0.00	22.47
−10	22.44	24.87	8.01	16.85	22.44	0.00	22.44
−5	20.69	14.14	6.54	7.59	20.67	0.03	20.63
0	14.60	10.12	6.07	4.05	14.46	0.22	14.24
5	10.15	8.68	5.82	2.86	9.82	0.60	9.22
10	8.43	8.21	5.68	2.53	7.91	1.03	6.88
15	7.48	8.04	5.60	2.43	7.22	1.40	5.81
20	7.76	7.99	5.57	2.41	7.09	1.69	5.39
Car Noise CE							
−20	36.35	98.40	3.65	94.74	32.79	13.88	18.90
−15	40.42	96.14	2.24	93.87	35.89	18.00	17.88
−10	39.87	87.49	3.42	84.06	35.97	17.90	18.06
−5	37.62	69.27	5.27	63.99	34.80	17.50	17.29
0	33.16	47.15	6.48	40.67	31.40	16.98	14.41
5	27.78	28.83	6.32	22.50	26.66	16.55	10.11
10	23.69	17.81	5.81	12.00	22.78	16.86	6.49
15	21.22	12.58	5.70	6.87	20.28	16.00	4.27
20	19.47	10.13	5.73	4.40	18.46	15.36	3.09
Car Noise RAPT							
−20	30.00	97.59	14.02	83.57	27.85	19.58	7.60
−15	31.49	96.30	14.14	82.16	27.33	18.08	9.25
−10	31.36	92.13	22.78	69.34	26.36	16.78	9.57
−5	30.12	82.95	29.55	53.40	25.04	15.42	9.61
0	27.09	66.31	27.17	39.14	22.69	13.22	9.47
5	22.43	45.78	19.10	26.67	19.17	9.91	9.25
10	18.02	29.02	12.15	16.86	15.88	6.77	9.11
15	15.25	19.55	8.50	11.04	13.75	4.67	9.08
20	13.76	15.33	6.93	8.40	12.44	3.52	8.92

Table B.2. Error for noise corrupted speech.

		Pitch Estimation			Voiced Decision		
SNR in dB	Combined	Total	Too high	Too low	Total	Voiced	Unvoiced
Car Noise YIN							
−20	39.66	98.77	3.15	95.61	35.05	17.84	17.20
−15	42.63	98.58	1.09	97.49	36.94	16.75	20.19
−10	42.42	95.99	1.93	94.04	36.90	16.77	20.12
−5	41.32	88.56	3.71	84.85	36.00	16.18	19.81
0	38.15	72.81	4.87	67.94	33.15	13.80	19.34
5	32.64	50.34	5.30	45.03	28.84	9.96	18.87
10	27.53	31.54	5.66	25.88	25.09	6.57	18.52
15	24.54	21.77	6.02	15.75	22.74	4.43	18.31
20	23.10	17.82	6.41	11.40	21.42	3.29	18.12
Car Noise TAPS							
−20	23.62	91.35	53.37	37.97	23.48	22.32	1.16
−15	23.72	80.32	63.63	16.30	23.49	22.22	1.26
−10	23.69	67.63	56.22	11.40	23.47	22.22	1.24
−5	23.34	54.81	47.13	7.68	23.22	21.98	1.24
0	21.67	41.71	35.95	5.76	21.58	20.24	1.33
5	18.59	29.14	24.31	4.82	18.40	16.62	1.77
10	15.68	21.04	15.16	5.88	15.29	12.68	2.61
15	13.95	17.38	10.56	6.81	13.33	9.75	3.58
20	13.04	14.56	9.17	5.38	12.21	7.81	4.40
Car Noise PEFAC							
−20	32.65	99.34	1.59	97.74	31.50	21.32	10.18
−15	37.88	98.47	0.49	97.97	36.33	20.92	15.41
−10	37.95	92.46	0.55	91.90	37.12	21.60	15.51
−5	37.12	75.77	1.11	74.65	36.68	21.58	15.10
0	34.95	53.37	2.40	50.97	34.66	19.77	14.89
5	31.17	33.81	3.82	29.99	30.89	16.10	14.79
10	27.60	21.07	4.85	16.21	27.26	12.54	14.72
15	24.80	14.32	5.41	8.90	24.37	9.88	14.48
20	22.23	10.95	5.63	5.32	21.71	8.05	13.65

Table B.2. Error for noise corrupted speech.

Bibliography

[AJC15] Stefan I. Adalbjörnsson, Andreas Jakobsson, and Mads G. Christensen. "Multi-pitch estimation exploiting block sparsity". In: *Signal Processing* 109 (2015), pp. 236–247.

[All70] Frances E. Allen. "Control flow analysis". In: *Proceedings of a Symposium on Compiler Optimization*. Urbana-Champaign, Illinois: ACM, 1970, pp. 1–19.

[Ata68] Bishnu Saroop Atal. "Automatic speaker recognition based on pitch contours". PhD thesis. Brooklyn, NY, USA, 1968.

[BDT+94] Denis Byrne, Harvey Dillon, Khanh Tran, Stig Arlinger, Keith Wilbraham, Robyn Cox, Bjorn Hagerman, Raymond Hetu, Joseph Kei, C Lui, et al. "An international comparison of long-term average speech spectra". In: *The journal of the acoustical society of America* 96.4 (1994), pp. 2108–2120.

[BHT63] B. P. Bogert, M. J. R. Healy, and J. W. Tukey. "The quefrency analysis of time series for echoes : cepstrum, pseudo-autocovariance, cross-cepstrum and saphe cracking". In: *Proc. Symposium on Time Series Analysis, 1963* (1963). URL: https://ci.nii.ac.jp/naid/10022304707/en/.

[CJ09] Mads Græsbøll Christensen and Andreas Jakobsson. "Multi-pitch estimation". In: *Synthesis Lectures on Speech & Audio Processing* 5.1 (2009), pp. 1–160.

[CSM93] Dan Chazan, Yoram Stettiner, and David Malah. "Optimal multi-pitch estimation using the EM algorithm for co-channel speech separation". In: *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 2. IEEE. 1993, pp. 728–731.

[DK02] Alain De Cheveighné and Hideki Kawahara. "YIN, a fundamental frequency estimator for speech and music". In: *The Journal of the Acoustical Society of America* 111.4 (2002), pp. 1917–1930.

[ETS17] TS ETSI. "102 250-2 v2. 2.1 speech and multimedia transmission quality (STQ)". In: *QoS aspects for popular services in mobile networks; Part 2: Definition of Quality of Service parameters and their computation* (2017).

[Fan68] Gunnar Fant. "Analysis and synthesis of speech processes". In: *Manual of phonetics* 2 (1968), pp. 173–277.

[FL88] Gunnar Fant and Q. Lin. "Frequency domain interpretation and derivation of glottal flow parameters". In: *STL-QPSR* 29.2-3 (1988), pp. 1–21.

[GB11] S. Gonzalez and M. Brookes. "A pitch estimation filter robust to high levels of noise (PEFAC)". In: *2011 19th European Signal Processing Conference*. Aug. 2011, pp. 451–455.

[GB14] S. Gonzalez and M. Brookes. "PEFAC - a pitch estimation algorithm robust to high levels of noise". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22.2 (Feb. 2014), pp. 518–530.

[GSB+81] Alan Garnham, Richard C Shillcock, Gordon DA Brown, Andrew ID Mill, and Anne Cutler. "Slips of the tongue in the london-lund corpus of spontaneous conversation". In: *Linguistics* 19.7-8 (1981), pp. 805–818.

[HDM+14] Reinhard von Hanxleden, Björn Duderstadt, Christian Motika, Steven Smyth, Michael Mendler, Joaquin Aguado, Stephen Mercer, and Owen O'Brien. "SCCharts: sequentially constructive statecharts for safety-critical applications". In: *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '14. Edinburgh, United Kingdom: ACM, 2014, pp. 372–383. ISBN: 978-1-4503-2784-8. DOI: 10.1145/2594291.2594310. URL: http://doi.acm.org/10.1145/2594291.2594310.

[Hes12] Wolfgang Hess. *Pitch determination of speech signals: algorithms and devices*. Vol. 3. Springer Science & Business Media, 2012.

[HL13] F. Huang and T. Lee. "Pitch estimation in noisy speech using accumulated peak spectrum and sparse estimation technique". In: *IEEE Transactions on Audio, Speech, and Language Processing* 21.1 (Jan. 2013), pp. 99–109. ISSN: 1558-7916. DOI: 10.1109/TASL.2012.2215589.

[Hoo03] Alan Hoofring. *Larynx and nearby structures*. File: nci-vol-4357-150.jpg. 2003. URL: https://visualsonline.cancer.gov/retrieve.cfm?imageid=4357&dpi=150&fileformat=jpg.

[Hus57] Par Raoul Husson. "Physiologie de la phonation". In: *Practica Oto-Rhino-Laryngologica* 50.2 (1957), pp. 123–139. DOI: 10.5631/jibirin.50.123.

[HWE+18] FH-Prof Mag DI Peter Hofer, Conny Walchshofer, FH-Prof Mag Dr Christoph Eisl, FH-Prof Mag Dr Albert Mayr, and Lisa Perkhofer. "Sankey, sunburst & co-interaktive big data visual-isierungen im usability test". In: *CARF Luzern 2018* (2018), p. 92.

[KM66] Richard M Karp and Rayamond E Miller. "Properties of a model for parallel computations: determinacy, termination, queueing". In: *SIAM Journal on Applied Mathematics* 14.6 (1966), pp. 1390–1411.

[KT99] Matti Karjalainen and Tero Tolonen. "Multi-pitch and periodicity analysis model for sound separation and auditory scene analysis". In: *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*. Vol. 2. IEEE. 1999, pp. 929–932.

[KZ02] Kavita Kasi and Stephen A Zahorian. "Yet another algorithm for pitch tracking". In: *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 1. IEEE. 2002, pp. I–361.

[Lev99] Willem J.M. Levelt. "Models of word production". In: *Trends in cognitive sciences* 3.6 (1999), pp. 223–232.

[LG88] L. Lee and A. A. Girgis. "Application of DFT and FFT algorithms to spectral analysis of power system load variation". In: *[1988] Proceedings. The Twentieth Southeastern Symposium on System Theory*. Mar. 1988, pp. 26–29. DOI: 10.1109/SSST.1988.17009.

[Lic51] Joseph Carl Robnett Licklider. "A duplex theory of pitch perception". In: *The Journal of the Acoustical Society of America* 23.1 (1951), pp. 147–147.

[LKS89] Lori F. Lamel, Robert H. Kassel, and Stephanie Seneff. "Speech database development: design and analysis of the acoustic-phonetic corpus". In: *Speech Input/Output Assessment and Speech Databases*. 1989.

[LM87] Edward A. Lee and David G. Messerschmitt. "Synchronous data flow". In: *Proceedings of the IEEE* 75.9 (1987), pp. 1235–1245.

[Loy11] Gareth Loy. *Musimathics: the mathematical foundations of music*. Vol. 1. MIT press, 2011.

[Man03] Ron Mancini. *Op amps for everyone: design reference*. Newnes, 2003.

[Mar72] John Markel. "The SIFT algorithm for fundamental frequency estimation". In: *IEEE Transactions on Audio and Electroacoustics* 20.5 (1972), pp. 367–377.

[Mil56] George A. Miller. "The magical number seven, plus or minus two: some limits on our capacity for processing information." In: *Psychological review* 63.2 (1956), p. 81.

[Moo74] J. Moorer. "The optimum comb method of pitch period analysis of continuous digitized speech". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 22.5 (Oct. 1974), pp. 330–338. ISSN: 0096-3518. DOI: 10.1109/TASSP.1974.1162596.

[Nie13] Heinrich Niemann. *Klassifikation von Mustern*. springer-Verlag, 2013.

[Nol67] A. Michael Noll. "Cepstrum pitch determination". In: *The journal of the acoustical society of America* 41.2 (1967), pp. 293–309.

[Nyq28] H. Nyquist. "Certain topics in telegraph transmission theory". In: *Transactions of the American Institute of Electrical Engineers* 47.2 (Apr. 1928), pp. 617–644. ISSN: 0096-3860. DOI: 10.1109/T-AIEE.1928.5055024.

[OS75] Alan V. Oppenheim and Ronald W Schafer. *Digital signal processing*. Prentice-Hall, 1975.

[Pro59] Reese T. Prosser. "Applications of boolean matrices to the analysis of flow diagrams". In: *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*. ACM. 1959, pp. 133–138.

[PTG+14] Pavlos Papadopoulos, Andreas Tsiartas, James Gibson, and Shrikanth Narayanan. "A supervised signal-to-noise ratio estimation of speech signals". In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, pp. 8237–8241.

[PWP+11a] Gregor Pirker, M. Wohlmayr, S. Petrik, and F. Pernkopf. "Database for multi-pitch tracking". In: *Graz University of Technology, Signal Processing and Speech Communication Laboratory, Tech. Rep. www. spsc. tugraz. at/tools* (2011).

[PWP+11b] Gregor Pirker, Michael Wohlmayr, Stefan Petrik, and Franz Pernkopf. "A pitch tracking corpus with evaluation on multipitch tracking scenario". In: *Twelfth Annual Conference of the International Speech Communication Association*. 2011.

[Rab77] L. Rabiner. "On the use of autocorrelation analysis for pitch detection". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 25.1 (Feb. 1977), pp. 24–33. ISSN: 0096-3518. DOI: 10.1109/TASSP.1977.1162905.

[RS+07] Lawrence R. Rabiner, Ronald W. Schafer, et al. "Introduction to digital speech processing". In: *Foundations and Trends® in Signal Processing* 1.1–2 (2007), pp. 1–194.

[SBS+11] Björn Schuller, Anton Batliner, Stefan Steidl, and Dino Seppi. "Recognising realistic emotions and affect in speech: state of the art and lessons learnt from the first challenge". In: *Speech Communication* 53.9-10 (2011), pp. 1062–1087.

[Sha49] Claude E. Shannon. "Communication in the presence of noise". In: *IEEE Proceedings*. Vol. 37. 1949, pp. 10–21.

[SK13] Hans-Rudolf Schwarz and Norbert Köckler. *Numerische Mathematik*. Springer-Verlag, 2013.

[SL90] Malcolm Slaney and Richard F. Lyon. "A perceptual pitch detector". In: *International Conference on Acoustics, Speech, and Signal Processing*. IEEE. 1990, pp. 357–360.

[SLH16] Steven Smyth, Stephan Lenga, and Reinhard von Hanxleden. “Model extraction for legacy c programs with SCCharts”. In: *Proceedings of the 7th International Sympsium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA ’16), Doctoral Symposium*. Vol. 74. Electronic Communications of the EASST. Corfu, Greece, Oct. 2016.

[Tal95] David Talkin. “A robust algorithm for pitch tracking (RAPT)”. In: *Speech coding and synthesis* (1995), pp. 495–518.

[Tit09] Ingo R. Titze. “How are harmonics produced at the voice source?” In: *Journal of Singing* 65.5 (2009), p. 575.

[TMM+99] Keiichi Tokuda, Takashi Masuko, Noboru Miyazaki, and Takao Kobayashi. “Hidden markov models based on multi-space probability distribution for pitch pattern modeling”. In: *icassp*. Vol. 1. 1999, pp. 229–232.

[Tra06] Trashtoy. *UML class diagram for singleton software design pattern.* 2006. URL: https://de.wikipedia.org/wiki/Datei:Singleton_UML_class_diagram.svg.

[VZD57] J.W. Van den Berg, J.T. Zantema, and P. Doornenbal. “On the air resistance and the bernoulli effect of the human larynx”. In: *The Journal of the Acoustical Society of America* 29.5 (1957), pp. 626–631.

[ZST+12] Yinsheng Zhou, Khe Chai Sim, Patsy Tan, and Ye Wang. “MOGAT: mobile games with auditory training for children with cochlear implants”. In: *Proceedings of the 20th ACM international conference on Multimedia*. ACM. 2012, pp. 429–438.

List of Abbreviations

KIELER Kiel Integrated Environment for Layout Eclipse RichClient

UML Unified Modelling Languages

SCCharts Sequentially Constructive Charts

ACF Autocorrelation Function

F0 Fundamental Frequency

T0 Fundamental Period

NCCF Normalized Cross Correlation Function

CCF Cross Correlation Function

AMDF Average Magnitude Difference Function

SAMDF Squared Average Magnitude Difference Function

CMNSAMDF Cumulative Mean Normalized Squared Average Magnitude Difference Function

CMNF Cumulative Mean Normalization Function

PEFAC Pitch Estimation Filter with Amplitude Compression

RAPT Robust Algorithm for Pitch Tracking

TAPS Temporally Accumulated Peak Spectrum

TAPS-r TAPS reference strategy

TAPS-h TAPS harmonics strategy

TAPS-f TAPS F0 bin strategy

TAPS-p TAPS averaged peak strategy

YIN Yin and Yang

YAAPT Yet Another Algorithm for Pitch Tracking

SIFT Simplified Inverse Filter Tracking

CE Candidates Evaluation

LTASS Long-Term Average Spectrum of Speech

DFT Discrete Fourier Transform

FFT Fast Fourier Transform

STFT Short-Time Fourier Transform

SHV summed up harmonic value

PTDB-TUG Pitch Tracking Database from Graz University of Technology

SNR Signal to Noise Ratio

TIMIT Texas Instruments/Massachusetts Institute of Technology

ETSI European Telecommunications Standards Institute

MOGAT Mobile Games with Auditory Training