JOHANNES KEPLER UNIVERSITÄT LINZ | JKU

TNF

Technisch-Naturwissenschaftliche Fakultät

Automatic Audio Defect Detection

BACHELORARBEIT

(PROJEKTPRAKTIKUM)

zur Erlangung des akademischen Grades

Bachelor of Science

im Bachelorstudium

INFORMATIK

Eingereicht von:
Rudolf Mühlbauer, 0655329

Angefertigt am:
Department of Computational Perception

Betreuung:
Univ.-Prof. Dr. Gerhard Widmer
Dr. Tim Pohle, Dipl.-Ing. Klaus Seyerlehner

Linz, Juni 2010

Kurzfassung

Diese Arbeit präsentiert Methoden zur automatischen Erkennung häufiger Defekte in Audio-Dateien. Nach einer Diskussion über den aktuellen Forschungsstand und verfügbare Softwareprodukte wird eine Übersicht ausgewählter Defekte, deren Quellen und Algorithmen zur Erkennung präsentiert. Es werden die Defekte “gap”, “jump”, “noise burst” und “low samplerate” beschrieben. Ein Evaluierungverfahren zur Bestimmung der Performance der Algorithmen wird erklärt sowie die Ergebnisse verschiedener Experimente präsentiert. Abschließend werden die Ergebnisse dieser Arbeit diskutiert und weitere Forschungsschwerpunkte vorgeschlagen.

Abstract

This work presents methods to automatically detect common quality problems in digital audio files. After a discussion about the state of the art in automatic audio defect detection and available software products, an overview over selected defects is provided, possible sources thereof, and algorithms to detect them are presented. The covered defects are “gaps”, “jumps”, “noise burst”, and “low samplerate”. An evaluation procedure to assess the performance of the algorithms is explained and the results of several experiments are discussed. The work concludes with an analysis of the results and suggestions for further research opportunities.

Contents

1 Introduction

The transition from traditional, analog audio storage media (such as gramophone records or audio tapes) to modern, digitally stored audio files introduces errors. The storage of analog media causes degradation of the information contained and introduces defects, as a Vinyl record would degrade over time [16] even without being played. On analog media, the playing also might greatly degrade the information as observable on cassette tapes [10].

When reading Digital Audio Compact Disk media for hard disk storage, mechanical problems on the CD's surface might introduce read errors if the defects break error correction as well as the classical problems we very well know from the portable CD players where jumps and gaps are introduced by vibrations on the player.

Digitally stored or transferred files are prone to bit level errors. While with raw PCM encoded audio data this would only result in incorrect samples, formats with error detection (`MP3` frame check sums for example, comp. [14] and [11]) would result in invalid frames.

There already exist tools to check `MP3` collections for problems, but they concentrate on `MP3` format errors such as missing tag information, broken file headers, or Variable Bit Rate (VBR) problems. Consult table 1.1 on the following page for a selection of available software solutions.

Also many noise removal and restoration applications are available, mainly based on noise gating, multiband noise gating or notch filtering. For studio applications these often require the user to train the algorithms with specific noise

Application	Reference
MP3 Diag	[22]
MP3Test	[23]
MP3val	[24]

Table 1.1: Selected `MP3` checking tools

samples. Various commercial audio applications provide click and pop detection and restoration geared towards digitization and post-production of vinyl as well as live recordings. Table 1.2 lists some available applications serving this purpose.

Application	Reference
Audacity	[19]
GoldWave	[20]
Izotope RX	[21]
Pristine Sounds	[26]
Ressurect	[27]
Soundsoap	[28]
Wavearts MR Noise	[29]
Waves Audio Restoration Plugins	[30]

Table 1.2: Selected noise removal applications

However, to ensure a certain level of audio quality in a large collection of music, it is no longer sufficient to manually review single audio files. An automated procedure to check such collections for defects is needed.

In this thesis a set of methods is presented to identify four common types of audio degradations typically introduced by digitization and encoding: gaps, jumps, noise bursts, and low sample rate. The algorithms provide a decision whether a song contains defects of the respective type or not. Special attention has been paid to the performance of the algorithm to enable the scanning of large collections in a feasible amount of time. The algorithms presented in this work try to detect defects without prior knowledge about the musical structure of the audio files under test. Only low-level signal processing methods without the usual prerequisites such as training as in traditional machine learning approaches are used. The specific defect types and the detection thereof are described in Chapter 3. Chapter 4 gives an overview of the implementation. The general organization of the source code is outlined and the auxiliary algorithms (such as filesystem traversal) are explained. The implementation provides different outputs: it is possible to create waveform plots of apparent defects and export `WAVE` files of these defects for later inspection. The performance of the implemented algorithms is evaluated with three experiments and the results are discussed in Chapter 5. Chapter 6 summarizes the results and explains strengths and weaknesses of the algorithms and possible future improvements to them.

The goal was to create a *push-button* application to crawl large collections for common defects, something that did not exist to that point.

1.1 Notations used in this document

Different typefaces are used depending on the context. Table 1.3 shows these typefaces.

Example	Meaning
text	Normal text
mean	Matlab function
i	Matlab variable
$\vec{X}$	Vector
`threshold`	Configuration variable
`MP3`	File format

Table 1.3: Notations

2 Related Work

Godsill and Rayner [5] present a method for click removal in degraded gramophone recordings. In their work, an auto regressive model is assumed for the signal and the model parameters are estimated from the corrupted audio data and used in a prediction error filter to calculate a detection signal. This detection signal is then thresholded to identify possible defects. Clicks are modeled using a *transient noise model*. Furthermore they describe methods to repair erroneous audio by replacing the audio samples in question based on different methods, including auto regressive estimation and median filtering. A Markov chain Monte Carlo (MCMC) method for similar applications is described in [3] and [4].

Fitzgerald et al. [2] provide an overview of Bayesian, model-based and MCMC methods with applications in signal and image processing such as audio enhancement and image restoration.

Vaseghi [13] covers many aspects of noise detection / reduction and includes detailed information about impulsive and transient noise. Statistical and model-based approaches to noise estimation and reduction are presented, as well as applications such as the restoration of gramophone records.

Much work has been done in the field of *perceived audio quality* to assess a subjective audio quality. An overview is given in Herrero [8]. PEAQ [15] for example provides methods to calculate a *mean opinion score* (MOS) that rates the quality of audio on a scale from 1 to 5. To get objective measurements also neural networks have been used to emulate a human assessment (Mohamed [12]).

However no literature was found that describes low-level signal processing to identify defects such as gaps or noise bursts in a way this work does.

3 Defect types and detection methods

This chapter describes four common audio defects and the respective detection methods that have been developed and implemented within the defect detection framework. The covered defects are: gaps, jumps, noise burst, and low sample rate.

A description of these defects, possible sources, and an introduction to the detection algorithms is given in the following sections, while a detailed evaluation of the performance of the implemented algorithms is presented in Section 5 on page 21.

3.1 Gap detection

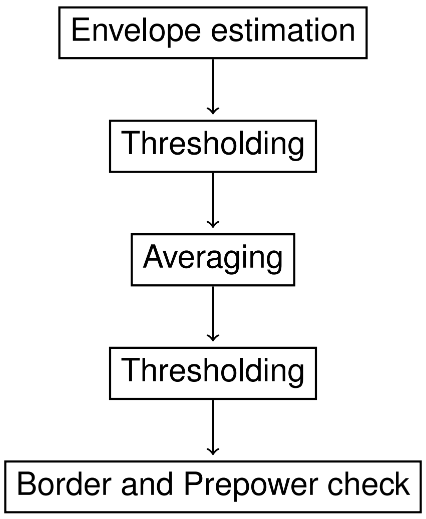

Figure 3.1: Gap Detection Flow

The gap detection algorithm finds silent parts in the audio, trying to distinguish between musically intentional pauses and real defects. Sources of such a defect can be for example:

- error reading an audio CD (optical problems)
- buffer underflow (bandwidth problems)

Figure 3.2 on the following page shows the waveform of a gap.

The proposed algorithm is depicted in Figure 3.1 and works as follows: the envelope of the signal is estimated by filtering the signal with an envelope follower similar to the one suggested by Herter [9]. The filtered audio data is thresholded to obtain a vector of logicals according to Equation 3.1 to find *silent* parts in the signal.

$$\vec{\mathrm{thresholded}}(i) = 1 \qquad \text{iff } \vec{\mathrm{filtered}}(i) \leq \texttt{threshold} \tag{3.1}$$

To exclude single outliers that might appear, the vector of logicals then is filtered with an averaging filter and again thresholded. The averaging is done by

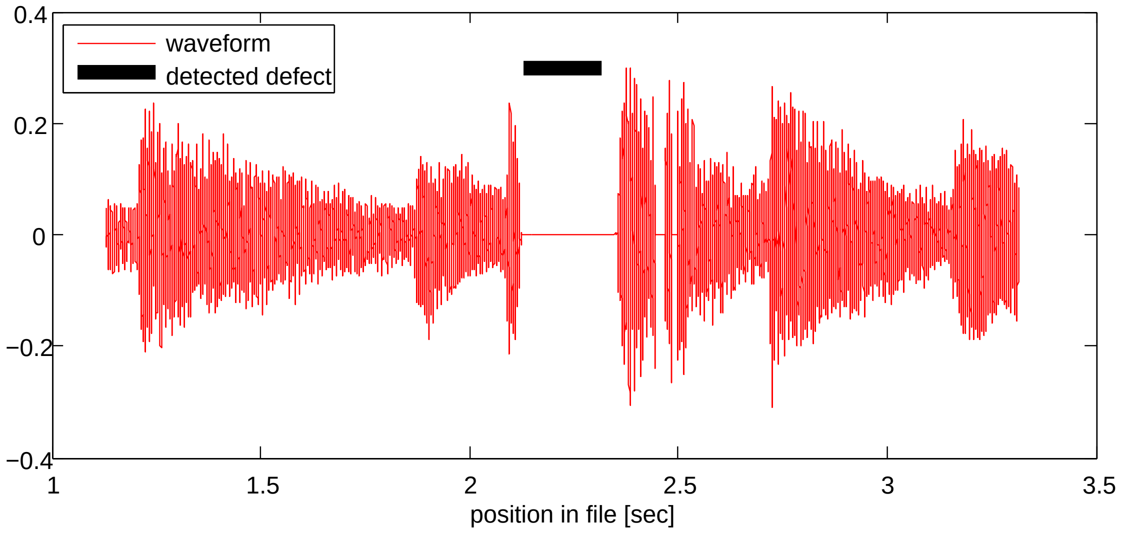

Figure 3.2: Example of a gap occurring in a defective file (non-synthetic)

filtering with a smoothing filter of size `len_tolerance` as described in Hamming (smoothing-by-5s, [7, p. 39]).

Gaps detected at the very beginning or the end (the so-called border) of the audio data are excluded. The remaining signal is scanned for regions below a certain threshold to identify potential gaps. For such a region the so-called *prepower* is calculated. The term prepower designates the cumulative power of `prepower_len` samples (the prepower frame) just before the beginning of the region (see Equation 3.2).

$$\text{prepower} = \text{mean}(\text{abs}(\overrightarrow{\text{prepowerframe}})) \tag{3.2}$$

If the prepower of the signal is less than `prepower_th`, it is considered to be a defect. Musically intentional pauses seem to tend to *fade out* before a pause, while errors stop *abruptly*, which is detected by this heuristic. Experiments showed this heuristic to be effective in most of the cases, however the algorithm creates many false positives when used on speech and electronic music (where samples of audio are put in sequence, possibly leaving gaps between those samples).

Gap detection accuracy would benefit from the distinction between *pauses* and *silence*. Figure 3.3 shows the difference: while in the left plots the source also stops (called *pause*), in the right one the source continues to progress, only the channel is muted (called *silence*). This distinction would add additional plausibility information and could be accomplished by using a beat-tracking algorithm. No further research was done in this direction but might prove beneficial to the performance of the algorithm.

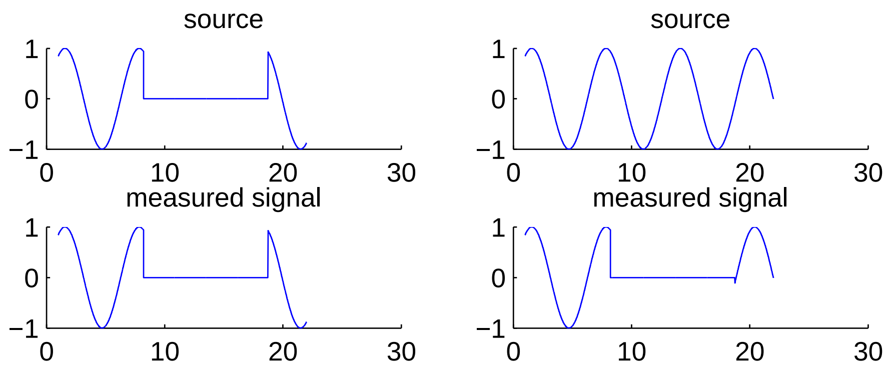

Figure 3.3: Gap Types: the left two plots show a *pause* situation: the signal starts where it stopped. The right plots show a *silence* situation: time progresses in the source.

3.2 Jump detection

Figure 3.5 on the next page sketches what we will understand as jumps: a discontinuity in the signal progression. This might be a *skip-forward* triggered by read errors from a Digital Audio Compact Disk, as well as any other sudden change in playback position.

As the plot 3.5 on the following page suggests, we are looking for sudden changes in signal characteristics. The basic procedure to find such discontinuities is depicted in Figure 3.4 on the next page.

This algorithm works as follows: for the input signal $\vec{s}$ an autoregressive Model is used. In a sliding-window manner the model parameters of the signal are cal-

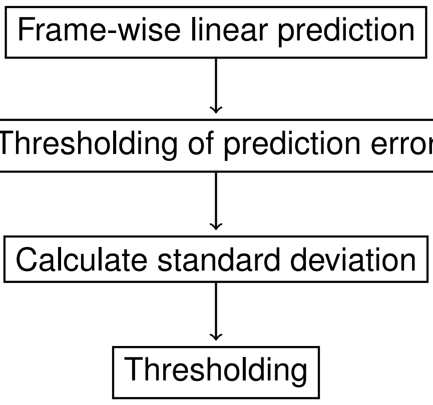
Frame-wise linear prediction
Thresholding of prediction error
Calculate standard deviation
Thresholding

Figure 3.4: Jump Detection Flow

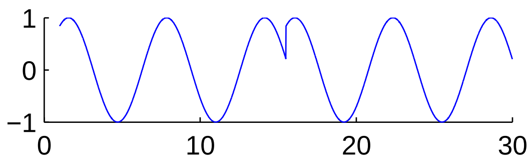

Figure 3.5: A discontinuity in the signal: sudden change of the signal

culated by solving the Yule-Walker equations (comp. [17] and [13, p. 209-238]). With this model and its parameters a prediction of the signal is estimated by filtering the signal with the prediction filter. This prediction signal $\vec{p}$ is compared to the original signal by building the sample-wise difference $\vec{d_n} = |\vec{s}_n - \vec{p}_n|$. The difference between the two, the residuals, is the prediction error. When the prediction error is high compared to a given threshold, a discontinuity is indicated. Additionally it is necessary to consider the signal's standard deviation at the point of discontinuity. This is a heuristic to distinguish between noisy parts in the music (for example the sound of a cymbal) and jump defects. This heuristic performs poor on some music genres, for example heavy rock music due to the noise parts created by distorted instruments.

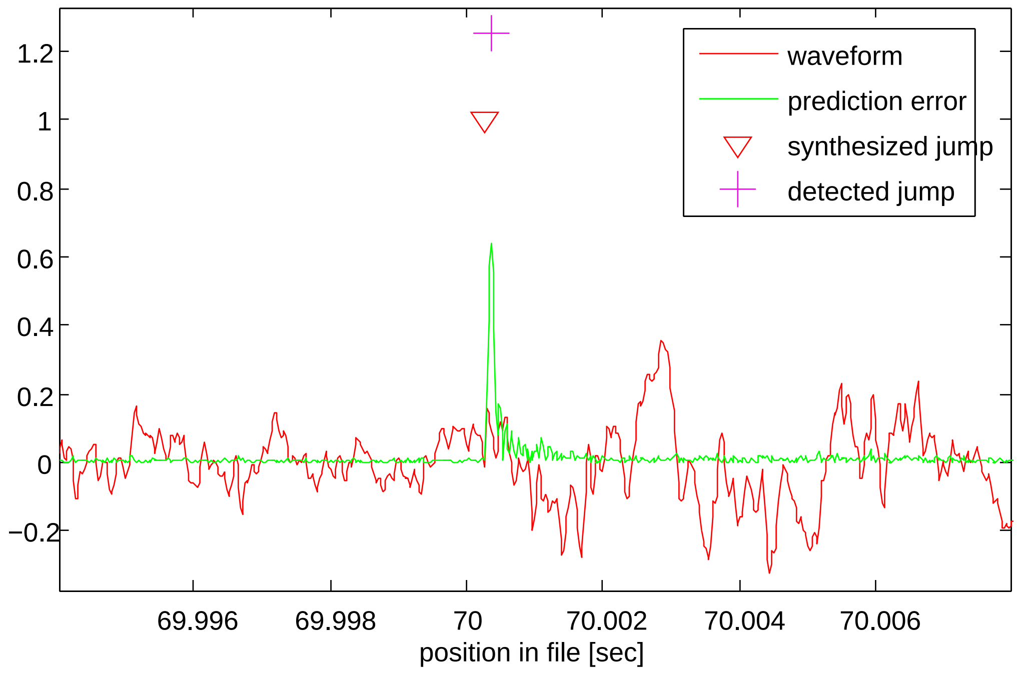

Figure 3.6: A successfully detected synthetic jump

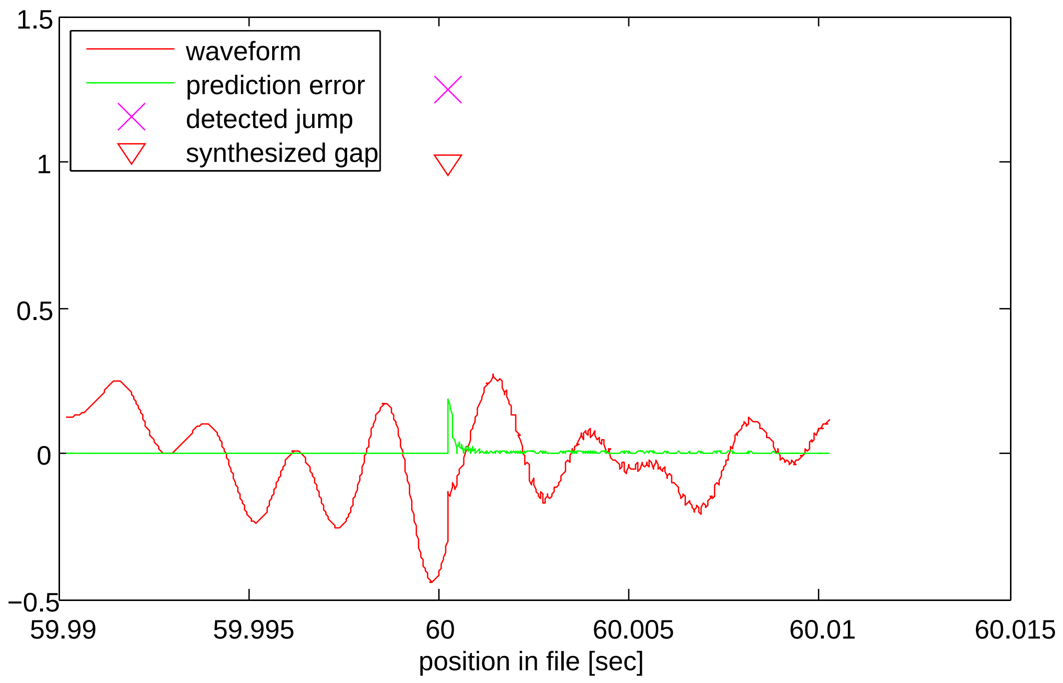

Figure 3.7: Another successfully detected synthetic jump

3.3 Noise detection

$$\begin{tikzcd}
\boxed{\text{Block-wise Fourier Transform}} \arrow[d] \\
\boxed{\text{Noise Estimation}}
\end{tikzcd}$$

Figure 3.8: Noise Flow

Some audio files tested showed random noise bursts. The reason for those defects might be as diverse as `MP3` frame errors [14] or other transmission / coding errors. However, observed defects showed similar characteristics: high energy and almost random distribution of sample values across the full bandwidth. Figure 3.9 shows such a defect. The plot shows a *power plot* at the top, a *spectrogram* in the middle and the waveform at the bottom. The shown defect is clearly audible as loud noise.

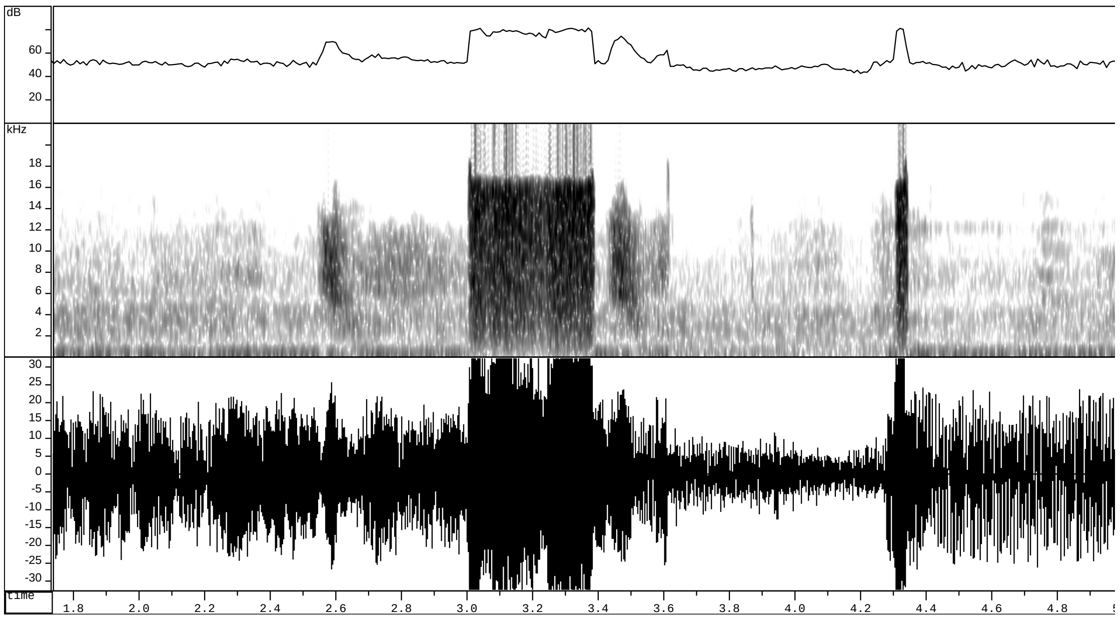

Figure 3.9: Plot of noise burst: power plot, spectrogram and waveform

The general approach to detect this type of error is depicted in Figure 3.8. In a frame-wise manner the audio data is scanned for regions with high energy and almost equally spread spectrum, which are considered to be noise. To detect the spread spectrum the mean value of the spectrum for a frame $\vec{X}$ of N audio samples is calculated according to Equation 3.3.

$$\text{mean}(\vec{X}) = \frac{\sum_{i=1}^{N} i\vec{X}_i}{\sum_{i=1}^{N} \vec{X}_i} \tag{3.3}$$

3.4 Low sample rate detection

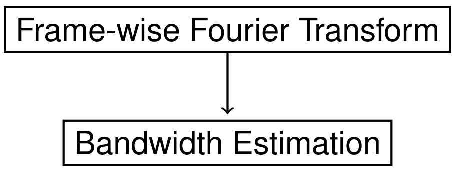

Figure 3.10: Low Sample Rate Flow

Even if the sample rate of the audio file would offer it, the signal in that file might not utilize the full bandwidth. This could happen when vintage recordings are digitized or the file format uses insufficient bandwidth. A common source for this defect is an encoding in the `MP3` format with low sample rate. To detect such shortcomings the bandwidth usage of the file is analyzed and compared to a configurable, normalized bandwidth.

To reduce the computational effort, only some frames are taken from the audio data and analyzed. These frames consist of `windowsize` samples and are taken every `probe_dist` samples in a sliding window manner.

Each frame f is Fourier transformed and the magnitude spectrum is calculated:

$$\vec{s} = \text{abs}(\text{fft}(\vec{f})) \tag{3.4}$$

The cumulative spectral power $\vec{C}(i)$ is the function:

$$\vec{C}(i) = \sum_{j=1}^{i} \vec{s}(j) \tag{3.5}$$

The overall bandwidth is estimated to be the frequency where the cumulative spectral power reaches 80% of the overall spectral power:

$$\vec{C}(i_{80\%}) \geq 0.8 \sum_{j=1}^{\texttt{windowsize}/2} \vec{s}(j) \tag{3.6}$$

The bandwidth usage of the whole audio file is then estimated by taking the overall maximum of all probed frames. This value is compared to the minimum bandwidth usage `th` and the reference sample rate of 22 050 samples per second in the as follows:

$$\frac{i_{80\%} \cdot \texttt{fs}}{22050 \cdot \texttt{windowsize}} < \texttt{th} \tag{3.7}$$

Experiments have shown that a value of 0.6 is reasonable for `th`. It properly detects audio data that was sampled with 22 050 Samples per second, even if it is contained in a file with 44 100 Samples per second. It also identifies vintage recordings as defective, which might not be what the user wants. However there is no simple heuristic to distinguish these cases.

4 Implementation

The detection framework is implemented as a set of Matlab functions and scripts. The implementation of the test- and evaluation framework is briefly described in the following sections. Figure 4.1 shows the basic interaction between the detection functions.

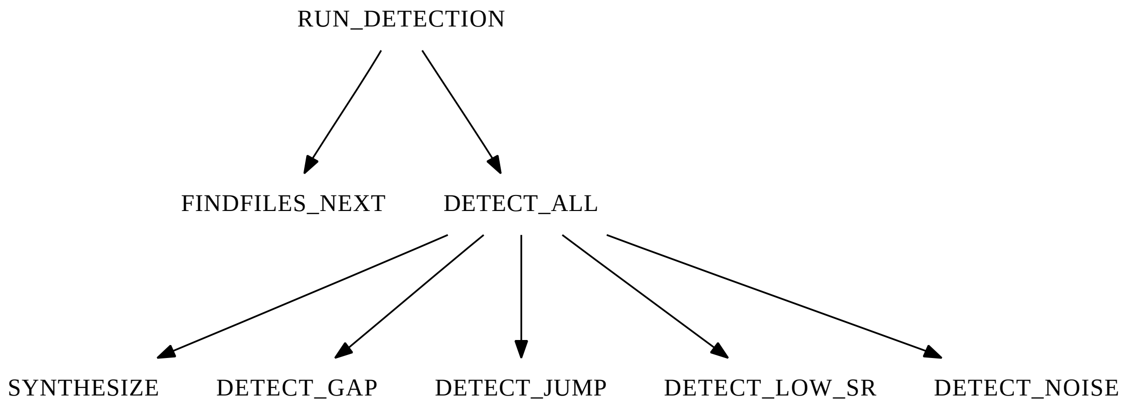

Figure 4.1: Interaction between functions

4.1 The detection framework

The basic interface to the detection framework is the function `RUN_DETECTION`. This function takes the path to a folder that contains the files under test and the configuration of the algorithms and provides the means to scan the audio files for defects. All necessary parameters are set up, the statistics functions as well as the file traversal functions are initialized. The parameters include the configuration for every detection algorithm, selection of the detection algorithms to apply, synthesis settings (see Section 5.1 on page 21), and output options.

The file traversal functions (see Section 4.3) provide the means to search the given path for audio files. For every file found the detection algorithms are invoked. Additionally to the detection the statistics are updated and log messages are produced.

For every defect detected, depending on the configuration, the program can export a plot showing the defect to `PDF` and `FIG` format. Also, a segment of the signal around a detected defect can be exported to a `WAVE` file for later inspection.

4.2 Statistics

The program keeps statistics such as timing information, defect classification, synthesis-, and file information to evaluate the performance of the detection algorithms.

All statistics are kept in a global structure. This allows simple performance assessment and is tightly integrated in the test environment. Calculated fields in this global structure allow easy interpretation as well as high-level analysis of the gathered data.

For every type of defect separate timing information is stored. This includes the detection time and as a calculated field the speedup (See also 5.3 on page 24 for more details).

4.3 File traversal

The File Traversal module consists of two functions for traversing a folder structure and is invoked by the detection framework. This is implemented as a stateful generator, saving memory by generating the file list *in-place* while detecting rather than creating a file list first and processing this list later. The current state of the generator is kept in a global structure.

File access is implemented through a generic interface providing an easy way to access different audio file types. At this time, read functionality is only available for `WAVE` and `MP3` (through the use of the external program `mpg123`, [25]) files.

5 Evaluation

No ground truth was available for evaluation. To assess the performance of the detection algorithms a simple solution was implemented: defects are synthesized and added to the audio data. This way a ground truth is available and the quality of the assessment depends only on the quality of the synthesis and the files under test. This chapter describes the synthesis of defects, an explanation of the evaluation procedure, the evaluation metrics, and three experiments. The results of the experiments are presented and discussed.

5.1 Synthesis of defects

If configured accordingly, synthetic defects are added to the audio data of the file under test. Defects can be synthesized for the defect types gap, jump, and noise burst. Of course a synthetic defect is only an approximation of defects observed in real audio data. The synthesis algorithm emulates some characteristics of real audio to a limited extent.

To create more realistic defects some synthesis parameters are randomized. So for example, the length and the position of the defects are randomized. The user supplies the μ and σ and the parameters are taken from a $N(\mu, \sigma^2)$ - normal distribution. The random number generator is initialized with a constant seed value at the beginning of every experiment to ensure reproducible results. Furthermore defects are only created at positions where the audio signal has a minimum volume to prevent creation of undetectable defects.

The use of synthetic defects is problematic: it tempts to optimize the detection parameters to find synthetic defects. Depending on the quality of the synthesis, this might be counterproductive for the detection of real, observable defects.

5.1.1 Synthetic pauses

For the distinction of *pause* and *silence* refer to Section 3.1 on page 10.

Pauses are created by inserting samples at the position of the defect. Sample values inserted are drawn from a $N(0, 1)$ - normal distribution with an adjustable gain to emulate noise. This gain could be set to zero to create absolute silence. As a result the length of the whole audio signal is increased.

5.1.2 Synthetic silences

When creating silence a consecutive sequence of samples is *overwritten* by samples drawn from a $N(0, 1)$ - random variable adjusted by a variable gain. However, in contrast to the synthesis of pauses the length of the whole audio is unchanged. Although a distinction between *pauses* and *silences* in the synthesis of defects is made, no such distinction is done in the detection of those defects. Therefore all generated *silences* will be detected as *pauses*. Still this distinction is made to support a future implementation of a *beat-aware* gap detection, as discussed in Chapter 6 on page 30.

5.1.3 Synthetic jumps

Synthetic jumps are created by deleting a consecutive sequence of audio samples. The samples right of the block (i.e. *in the future*) are moved to the left, overwriting some parts of the signal. This creates a defect very similar to a real observable jump. The whole signal is shortened by the number of samples removed.

5.1.4 Synthetic noise

Noise synthesis is based on silence synthesis. The only difference is that the noise signal has a higher power. A block of noise is generated by inserting samples drawn from a uniform distribution in the range -0.9 and 0.9. The noise inserted is not additive: the original samples are replaced. The waveform depicted in Figure 5.1 shows a synthetic noise burst in the left part and a natural defect in the right part.

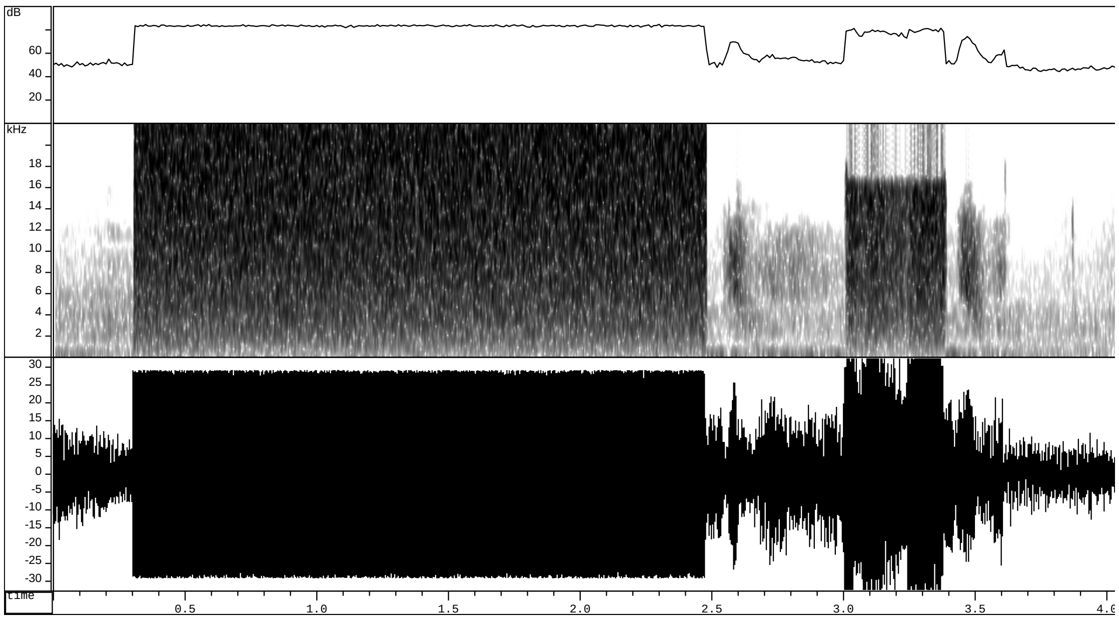

Figure 5.1: Plot of synthetic noise: power plot, spectrogram and waveform

5.2 Evaluation requirements

The purpose of the evaluation is clear: it must be possible to get metrics of the performance of the algorithms to be able to:

- asses the quality of the detection
- perform optimization of parameters

- find genre-dependent settings
- optimize computational efficiency without affecting quality

5.3 Evaluation specification

Following metrics were chosen for evaluation:

Speedup S The ratio of audio playing time and defect detection time.

$$S = \frac{\text{playing time}}{\text{detection time}}$$

as defined in [18].

This metric provides information about the efficiency of the algorithm. It is defined as $(\mathbb{R}_+ \times \mathbb{R}_+) \mapsto \mathbb{R}_+$.

Recall R is used as a measure of *completeness* of the detection:

$$R = \frac{|\text{synthetic} \cap \text{detected}|}{|\text{synthetic}|} = \frac{|\text{true positive}|}{|\text{true positive}| + |\text{false negative}|} \tag{5.1}$$

Recall is an effectivity metric and defined as $(\mathbb{N} \times \mathbb{N}) \mapsto [0, 1]$. A result of 1 indicates that all defects were detected

Precision P describes the *exactness*:

$$P = \frac{|\text{synthetic} \cap \text{detected}|}{|\text{detected}|} = \frac{|\text{true positive}|}{|\text{true positive}| + |\text{false positive}|} \tag{5.2}$$

Precision is effectivity metric as well and defined as $(\mathbb{N} \times \mathbb{N}) \mapsto [0, 1]$. A result of 1 indicates that no non-defects were classified as defect

F-Score F combines Recall and Precision:

$$F = 2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall + Precision}} \tag{5.3}$$

F-Score is defined as $(\mathbb{N} \times \mathbb{N}) \mapsto [0, 1]$. A Result of 1 only occurs when both Recall and Precision are 1.

5.4 Evaluation plan

The evaluation procedure can be easily executed for a directory containing audio files. The general algorithm is depicted in Figure 5.2.

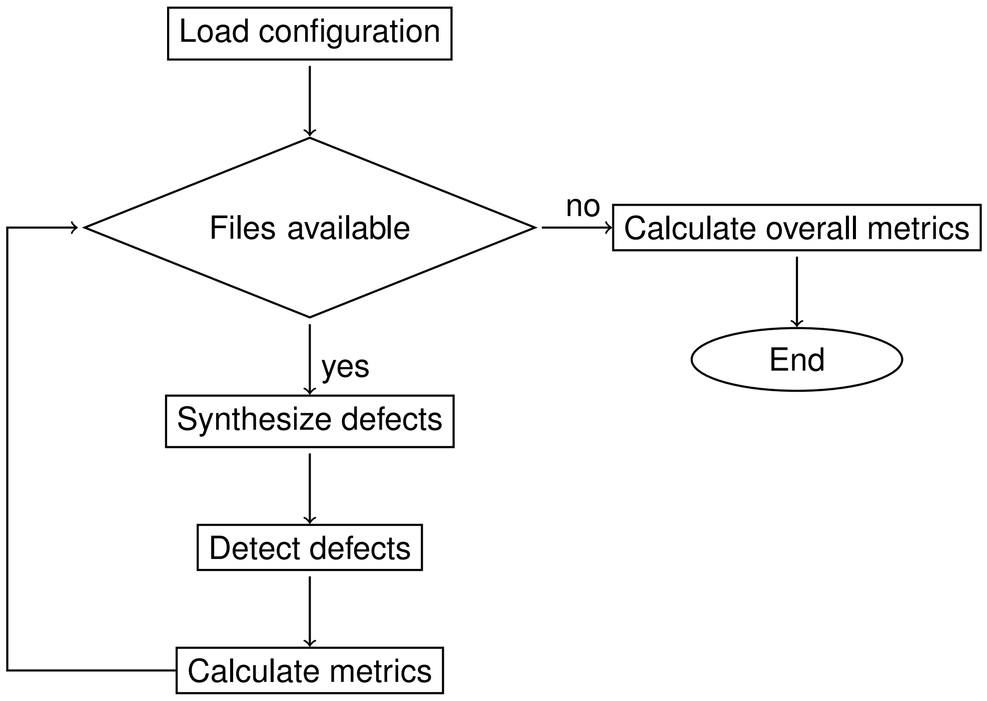

Figure 5.2: Evaluation Algorithm

Three different experiments have been conducted for evaluation purposes. The first experiment (“good_bad”) uses a small number of files with known defective files (the “bad” set) and files without defects (the “good” set). The second experiment (“collection”) was made with a rather large collection of music files with defect synthesis enabled. The third experiment is a parameter optimization for the jump detection algorithm, aiming at simultaneous optimization of several parameters. The results are documented in the following sections.

5.5 Experiment 1: classification

The set “good_bad” contains 19 files, 13 thereof considered defective. A whole file is either considered “good” or “bad”.

For these files all detection algorithms were enabled and defect synthesis was disabled. The evaluation only considers files marked as defective and therefore does not consider individual defects found in the file. The classification in “good” and “bad” is considered as the ground truth.

Table 5.1 on the next page shows the defect classification for every file in the “good_bad” set while Table 5.2 on the following page shows efficiency information of the detection algorithms.

Counting the positive and negative outcomes of Table 5.1 on the next page on a per-file basis, following numbers can be counted:

$$|\text{true positive}| = 11 \tag{5.4}$$

$$|\text{false negative}| = 2 \tag{5.5}$$

$$|\text{false positive}| = 1 \tag{5.6}$$

This results in values for Precision, Recall and F-Score:

$$P = 0.92 \tag{5.7}$$

$$R = 0.84 \tag{5.8}$$

$$F = 0.88 \tag{5.9}$$

Set	File name	detected defects
good	`Good_FIR.mp3`	-
good	`Good_myf.mp3`	-
good	`Good_TRY.mp3`	-
good	`Good_Zhou_1.mp3`	-
good	`Good_Zhou_2_.mp3`	jump
bad	`1.mp3`	jump, pause
bad	`2.mp3`	jump
bad	`3.mp3`	-
bad	`604504000000005209.mp3`	low_sr, pause
bad	`604526000000000255_h.mp3`	-
bad	`604526000000000255.mp3`	low_sr, pause, jump
bad	`6.mp3`	low_sr, jump
bad	`7.mp3`	low_sr
bad	`Bad_FIR.mp3`	jump
bad	`Bad_myf.mp3`	noiseburst
bad	`Bad_TRY_bad.mp3`	noiseburst, jump
bad	`Bad_zhou_1.mp3`	low_sr
bad	`Bad_zhou_2.mp3`	low_sr, jump

Table 5.1: Classification results for experiment 1

	Jump	Pause	Noise	Low_SR	Total
Total playing time [sec]					1423
Detection time [sec]	731	25	49	6	812
Speedup					1.75

Table 5.2: Efficiency for experiment 1

5.6 Experiment 2: synthetic defect detection

The set “collection” is a huge set of music files (1715 files in total) of diverse genres and quality. Some of these files contain defects, introduced by bad encoding, ripping of defective Digital Audio Compact Disks, or file errors while most of the files are in good shape.

In this experiment defect synthesis was enabled, where not only synthetic errors are detected but also the real errors of the audio files are counted. Therefore the calculation of Recall and Precision suffer on correctness but the major part of the detected defects should originate in synthetic defects. With synthesis enabled the exact position of the (synthetic) defects are known. That information are used as ground truth to calculate recall and precision.

For low sample rate detection no effectivity metrics are available since no defect synthesis is available for this type. Recall and Precision for the detection of jumps is very poor, this can be attributed to both the implementation and the parametrization of the detection algorithm. The actual results are shown in Table 5.3. The experiment used audio data of a total playing length of approximately 7 045 minutes and finished in 2 206 minutes, yielding a total speedup of 3.19.

	Jump	Pause	Noise	Low_SR
Detection time [sec]	1.08×10^5	8.02×10^3	1.48×10^4	2.02×10^3
Speedup	3.93	52.71	28.66	208.64
Recall	0.446	0.998	0.999	-
Precision	0.359	0.496	0.998	-
F-Score	0.398	0.663	0.999	-

Table 5.3: Results for experiment 2

5.7 Parameter optimization

Exemplary for the detection of jumps a parameter study was conducted. A set of files was scanned repeatedly for defects with different detection parameters. For each of these experiments the F-Score (See also equation 5.3 on page 24) was calculated to assess the quality of the parametrization.

As input the five “good” files out of the “good_bad” set were used. For these files synthetic jumps were generated.

The parameters `threshold` and `sigma` were swept through a reasonable range. Figure 5.3 shows the results of this optimization: the F-Score for every parametrization is inscribed in the plot and show a clear maximum around `threshold` $= 45$ and `sigma` $= 20$.

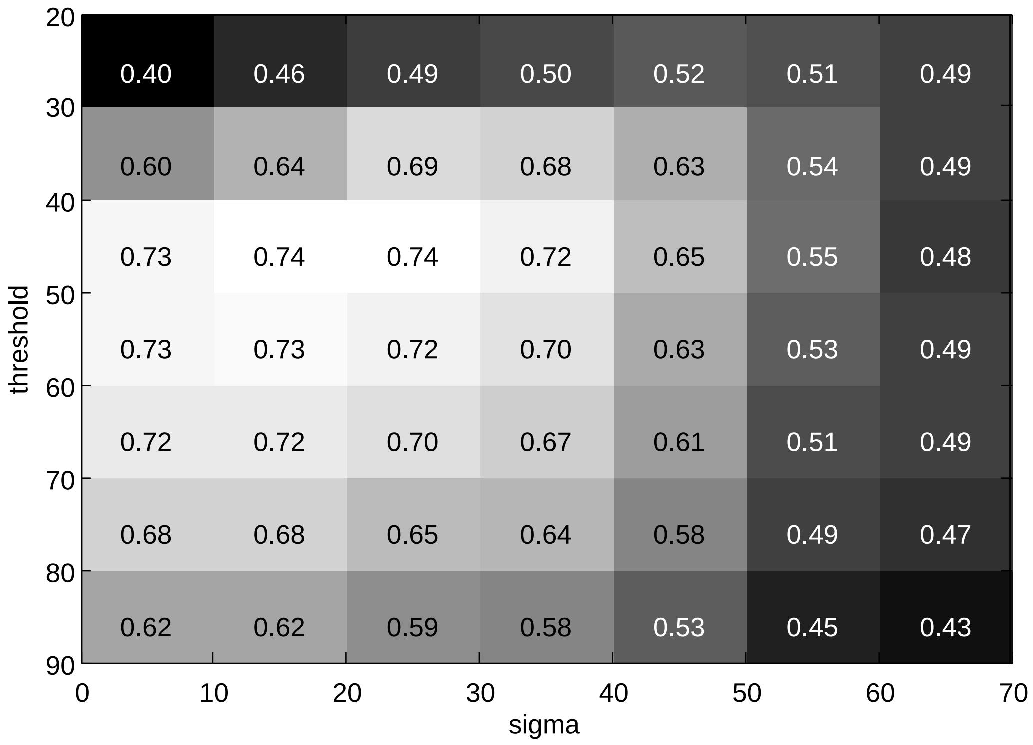

Figure 5.3: Results of parameter optimization

6 Conclusions and further work

The overall performance of the detection algorithms is reasonable and delivers clear indications of bad quality when applied to large music sets. Due to the low recall rates many false positives are created, which makes it necessary to manually review the files reported to be defective.

Generally the implemented algorithms give good hints which files in a collection might be defective but are results are not good enough for fully automatic classification or to justify automatic deletion of apparently defective files.

6.1 Gap detection

Generally the gap detection performs well in both run-time and effectivity. Further heuristics would be needed to increase recall rate on speech and electronic music. On speech data the distinction between erroneous gaps and intended musical pauses fails, the implemented heuristics would either need more careful parametrization or additional plausibility checking. Electronic music fabricated by using prerecorded audio segments tends to introduce gaps between these segments unlike the signals produced by real instruments.

The distinction between pauses and silences as discussed in Section 3.1 on page 10 could be accomplished by implementing a beat-tracking algorithm as described by Eck [1] or Hainsworth [6, p. 121 ff] and could improve the effectivity of the detection.

6.2 Jump detection

Jump detection needs improvement in both efficiency and effectivity: the computational effort should be reduced to gain larger speed-ups. To accomplish that several speed-limiting factors have to be considered. First, the sliding-window-scheme must be over-thought and optimum values for frame sizes and frame offsets chosen. The detection has poor recall rates when used on electronic or heavy rock music. Further heuristics should be implemented to support plausibility checking to exclude false-positive defects due to genre-inherent sound properties.

6.3 Noise burst detection

Noise bursts are detected quite well with feasible computational efforts. The recall rates are high when applied “noisy” music such as heavy rock. The distortion of the instruments, as well as the loud drums are interpreted as noise bursts. It is either possible to improve the parametrization or include further heuristics to exclude such false positives. The precision of the algorithms is reasonable.

6.4 Low sample rate detection

Low Sample rate detection is implemented quite efficient and delivers good precision and recall rates and well distinguishes between high-fidelity and low quality recordings. One unresolved problem is the classification of vintage recordings. Since the distinction between low-quality and vintage bandwidth usage is purely subjective it is hard to check for plausibility. Machine Learning approaches could resolve this issue, but are contradictory to the initial assumptions for the implementation.

List of Figures

7 Bibliography

[1] Douglas Eck. Beat tracking using an autocorrelation phase matrix. In *In Proceedings of the 2007 International Eck Research Statement 9*, 2007.

[2] W. J. Fitzgerald, S. J. Godsill, A. C. Kokaram, and J. A. Stark. Bayesian Methods in Signal and Image Processing. In *In Bayesian Statistics 6*, pages 239–254. Oxford University Press, 1999.

[3] Simon Godsill and Peter Rayner. Robust Treatment of Impulsive Noise in Speech and Audio Signals, 1995.

[4] Simon Godsill and Peter Rayner. Statistical Reconstruction And Analysis Of Autoregressive Signals In Impulsive Noise, 1998.

[5] Simon Godsill, Peter Rayner, and Olivier Cappé. Digital Audio Restoration. In *Applications of Digital Signal Processing to Audio and Acoustics*, pages 133–193. Kluwer Academic Publishers, 1997.

[6] Stephen Webley Hainsworth, Stephen W. Hainsworth, and Stephen W. Hainsworth. Techniques for the Automated Analysis of Musical Audio. Technical report, University of Cambridge, 2003.

[7] R. W. Hamming. *Digital Filters*. Dover, third edition, 1989.

[8] C. Herrero. Subjective and objective assessment of sound quality: solutions and applications. In *CIARM conference*, 2005.

[9] E. Herter and W. Loercher. *Nachrichtentechnik*. Hanser, 8. edition, 2000.

[10] Richard L. Hess. Tape Degradation Factors and Predicting Tape Life. In *AES Convention:121 (October 2006)*. Audio Engineering Society, 2006. `http://www.aes.org/e-lib/browse.cfm?elib=13804`.

[11] Davis Pan. A Tutorial on MPEG/Audio Compression. *IEEE MultiMedia*, 2:60–74, 1995.

[12] Hossam Afifi Samir Mohamed, Francisco Cervantes-Pérez. Audio Quality Assessment in Packet Networks, 2001.

[13] S.V. Vaseghi. *Advanced digital signal processing and noise reduction*. Wiley, third edition edition, 2006.

[14] MP3 Format. `http://www.mpgedit.org/mpgedit/mpeg_format/MP3Format.html`. As seen on 2010/03/12.

[15] PEAQ - The ITU Standard for Objective Measurement of Perceived Audio Quality. `http://www.aes.org/e-lib/browse.cfm?elib=7019`. As seen on 2010/03/12.

[16] Biological Agents of Vinyl Degradation. `http://micrographia.com/projec/projapps/viny/viny0200.htm`. As seen on 2010/03/20.

[17] Linear prediction. `http://en.wikipedia.org/wiki/Linear_prediction`. As seen on 2010/03/12.

[18] Speedup. `http://en.wikipedia.org/wiki/Speedup`. As seen on 2010/03/12.

[19] Audacious: Click Removal. `http://wiki.audacityteam.org/index.php?title=Click_Removal`. As seen on 2010/03/12.

[20] GoldWave. `http://www.goldwave.com/release.php`.

[21] Izotope RX. `http://www.izotope.com/products/audio/rx/`. As seen on 2010/04/01.

[22] Diagnose Your Mp3 Collection With MP3 Diag. `http://www.ghacks.net/2009/08/20/diagnose-your-mp3-collection-with-mp3-diag/`. As seen on 2010/03/12.

[23] MP3Test. `http://www.maf-soft.de/mp3test/`. As seen on 2010/03/12.

[24] MP3val is a small, high-speed tool for MPEG audio files validation and (optionally) fixing problems. `http://mp3val.sourceforge.net/docs/manual.html`. As seen on 2010/03/12.

[25] mpg123 - Fast console MPEG Audio Player and decoder library. http://www.mpg123.de. As seen on 2010/05/10.

[26] Pristine Sounds 2000. http://www.sonicspot.com/pristinesounds/pristinesounds.html. As seen on 2010/03/12.

[27] Resurrect your old recordings. http://wwwmaths.anu.edu.au/~briand/sound/. As seen on 2010/03/12.

[28] SoundSoap. http://xserve1.bias-inc.com:16080/products/soundsoap2/. As seen on 2010/03/12.

[29] Wavearts MR Noise. http://wavearts.com/products/plugins/mr-noise/. As seen on 2010/04/01.

[30] Waves Audio Restoration Plugins. http://www.waves.com/content.aspx?id=91#Restoration. As seen on 2010/04/01.

RUDOLF MÜHLBAUER

Staatsangehörigkeit: Österreich

Geburtsdatum: 29. März 1983

Geburtsort: Burgkirchen

AUSBILDUNG

- 1989 - 1993 Volksschule Neukirchen a.d.E
- 1993 - 1997 Hauptschule Neukirchen a.d.E
- 1997 - 2002 HTL Braunau, Nachrichtentechnik
- Studium der **Informatik** seit September 2006