Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science
Bachelor's Program in Computer Science

Bachelor's Thesis

Audio Denoising With Image Processing Methods

Jan Hendrik Dithmar

2008-10-1

Supervisor	Prof. Dr. Joachim Weickert
Advisor	Dr. Martin Welk
Reviewers	Prof. Dr. Joachim Weickert
	??

Affidavit

I hereby declare that this bachelor's thesis has been written only by the undersigned and without any assistance from third parties.
Furthermore I confirm that no sources have been used in the preparation of this thesis other than those indicated in the thesis itself.

Saarbrücken, 2008-10-1

Declaration of Consent

Herewith I agree that this thesis will be made available through the library of the Computer Science Department. This consent explicitly includes both the printed, as well as the electronic form attached to the last page of this thesis.
I confirm that the electronic and the printed version are of identical content.

Saarbrücken, 2008-10-1

Abstract

TODO: Abstract This thesis should give an overview whether image processing methods can be adapted to audio signals. To answer this question

In this thesis, methods are explored to assist the process of exchanging noisy parts by non-noisy ones where the inspiration comes from image processing.

Acknowledgements

I would like to express my sincere thanks to my advisor Dr. Martin Welk and my supervisor Prof. Dr. Joachim Weickert for their excellent mentoring and support during the development of this thesis. Their encouragement and the friendly atmosphere among colleagues in the Mathematical Image Analysis Group greatly assisted me in my work. I also like to thank ?? for consenting to review my thesis.

Special thanks also go to my family who enabled me to perform my studies as I did and who therefore also significantly contributed to the success I have enjoyed so far.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Post-editing and post-production of music becomes more and more important in music industry. Especially when recording a live performance it can happen that a certain part is degraded, for example caused by a cough. In a recording studio simply another take would be recorded which is not possible in a live performance.

Another possible scenario could be a noisy playback, say by a record player playing an old venyl. Here, the noise is due to the playing technique itself.

In general, noisy parts can be corrected. This is possible because music is highly redundant. So far, the process of correction was a manual process: people had to go through the whole piece of music and acoustically judge whether a certain part is similar to the noisy one. After having found a possible candidate it had to be copied at the location of the noisy data. Recently, this work is supported by spectrograms which represent the audio signal in a three-dimensional way, giving the time on the x-axis, the frequency on a y-axis and the intensity of the frequencies in a color representation where the blue color stands for a high intensity going over red and yellow to white which stands for a low intensity. An example is shown in figure 1.1.

At first glance, images and audio data seem to have nothing in common. But assisted by the Fourier Transform, it's possible to check whether similar patches can be found by the use of an appropriate similarity measurement. Furthermore the results can be judged by putting the patches categorized as similar at the position where a noisy patch is located (inpainting).

1.2 Goals

The goal of this thesis is to show whether it is possible to adapt methods like inpainting or NL-means used in image processing to audio signals. Since it is relatively new to image processing

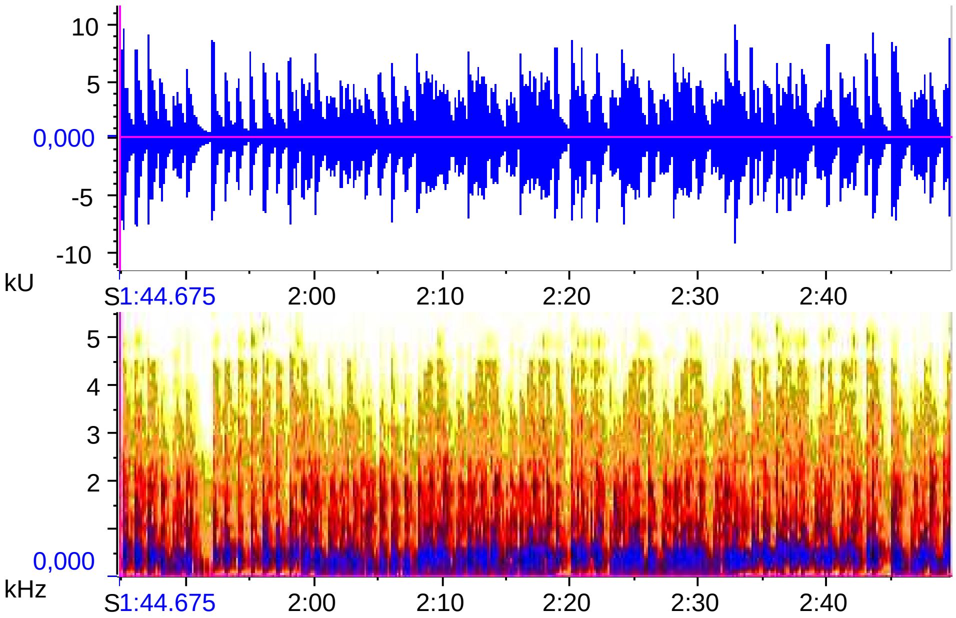

Figure 1.1: Spectrogram representing a part of *cyftlt-mono.wav*. **Top:** Waveform view. *x-axis:* Time. *y-axis:* Amplitude. **Bottom:** Spectrogram view. *x-axis:* Time. *y-axis:* Frequencies. *Colors:* Intensity. Blue is more intense, white is less intense. **Visualization:** using RavenLite 1.0 [2].

people to deal with audio data, a good foundation should be established and evaluated whether different approaches from image processing work for audio data and how well they perform on standard problems where the runtime is not in the main focus of this thesis.

For this purpose, a simple framework shall be established to find similar patches in a piece of music. Using this framework as a basis, different methods like simple inpainting, averaging the n best results on a sample basis or non-local means shall be evaluated.

This framework is realized by a C programme that is build from scratch except for the code used for the Fast Fourier Transform (FFT) which was provided.

First of all, the programme should get synthetically generated data as a test input where the result of the denoising can be easily verified. After having the knowledge that the programme runs correctly on these signal, real-world examples can be investigated.

1.3 Contents

The structure of this thesis is as follows. In Chapter 2 a short overview of the similarity measurement is given. The modification that is needed to use the Fourier transform on audio signals is mentioned (2.1) as well as how possible similarity measurements are computed

(2.2).

Chapter 3 discusses shortly non-local algorithms in image processing. This builds the basis for chapter 4, where the introduced methods are modified to fit the needs of audio processing.

Chapter 5 introduces different experiments. By first explaining the parameter selection (5.2) a few words are said about the memory requirements of the algorithm (5.2.1) and the runtime (5.2.2) under the different parameter settings. After that different methods to repair noisy data are investigated, starting in 5.3.

Chapter 6 concludes the thesis and gives ideas for possible future work.

The WAV file format which is the file format used in this thesis is described in the Appendix A where the file structure is given in A.1 and a short overview of what this data actually means is given in Appendix A.2.

Chapter 2

Similarity Measurement

2.1 Prerequisites

Before the similarity measurement can be described, some prerequisites have to be made. These prerequisites are easy to understand for image processing people since the ideas from image processing are only modified in a slightly different manner. Here it becomes apparent that image processing and audio processing are not too much different.

2.1.1 Short-Time Fourier Transform

The short-time Fourier transform is a Fourier-related transform that deals with a problem of the Fourier transform, namely that it makes no statement about the spatial appearance – in the case of this thesis temporal appearance, respectively – of the frequencies. Since a temporal analysis is needed for a piece of music, the short-time Fourier transform is used to compensate this disadvantage of the Fourier transform.

The short-time Fourier transform only looks at a certain part of the signal where it is multiplied with a window function w. After that, the ”classical” Fourier transform is used to calculate the Fourier coefficients for each window.

One drawback of the short-time Fourier transform is that it either has a good frequency resolution or a good time resolution but not both. A wide window gives better frequency resolution while a narrow window gives good time resolution. Often used in music analysis, it can assist studio engineers by depicting the frequencies of a song with a spectrogram which is already touched in 1.1.

There exist a lot of possibilities for the window function. One can use for example a box function [1] which would result in a too hard windowing, especially at the boundaries, since the amplitude would change abruptly resulting in a broad spectrum in the frequency domain. Using a smooth window function avoids this problem. This can be also easily understood from another viewpoint: a box function would weight the most important point at the center as much as a point far away which would not make sense, because a point far away from a point to investigate cannot be of the same importance as the investigation point itself.

2.1.2 Hamming Window

Since a windowed Fourier transform is used for the interpretation of the data, a window function is needed that is suitable for the analysis meaning it should weight the central point of a frame the most and shouldn't weight the left and the right part of a frame too much. In other words, the function should have its maximum in the center of the frame where it decreases in the direction of the boundaries. A Gaussian would fulfill this property where here the problem arises that it has to be cut at a boundary – due to its infinite extend – which would result in some kind of approximation depending on the point where the cut is performed.

So one has to come up with another window which deals a bit more clever with the boundary problem. An example for such a window is the Hamming window [1] which can be seen as a modified cosine window and is defined as follows:

$$w(n) = 0.53836 - 0.46164 \cos\left(\frac{2\pi n}{N-1}\right)$$

where N is the width of a discrete-time window function in samples (typically a power of 2) and n an integer value with $0 \leq n \leq N-1$.

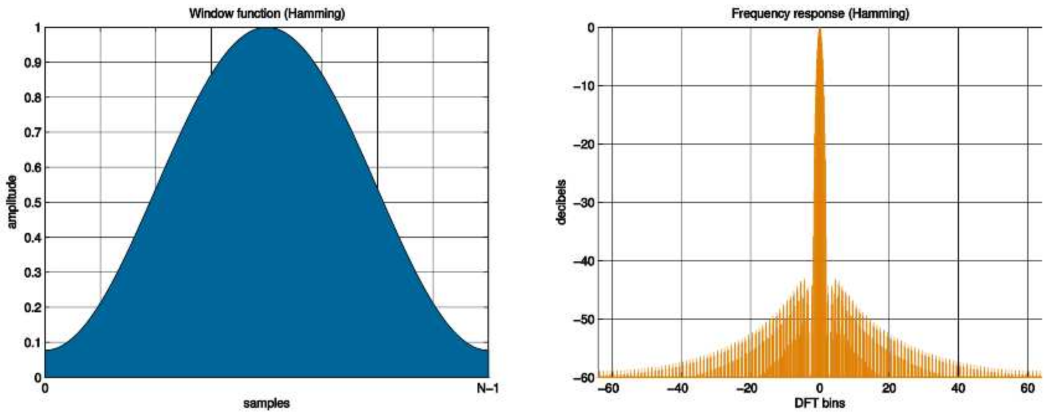

Figure 2.1: Hamming window. **Left:** Sample domain. **Right:** Frequency domain. **Source:** Wikipedia [1].

As one can easily see in the frequency domain, the central point is represented as a relatively narrow peak where already its neighborhood doesn't have a high weight anymore. Another possibility is the Hann window that also does not weight the boundaries too much but also doesn't deal that nicely with the central point one is interested in (the peak in the middle is a bit broader). This can be easily seen when comparing the different peaks in the frequency domain of the different windows as depicted in [1].

2.1.3 Step Size and Window Width

As already known, the short-time Fourier transform is the key to analyze pieces of music. Since it doesn't deal with the signal as a whole but processes only small pieces, a step size has to be specified which determines how big the steps are that are taken to go to the next input frame for the short-time Fourier transform.

But also another parameter is of importance: the window width. Since a Hamming window is applied to the data which is used for the Fourier transform, it needs to know how much data actually to consider. The window width should be a power of 2 since this is a requirement of the Fast Fourier Transform (FFT).

Using for example the values 256 for the step size and 1024 for the window width would actually mean that every 256 samples a data window of width 1024 is considered. Note that an overlap of 75% of the signal is present for this example. An overlap is in general a good idea since it means that a lot of different frames contribute to find frames that are similar to the reference frame. Considering that a box function weights every point in the window to analyze in the same way – as already mentioned in 2.1.1 – would result in having many data points that are not so important contributing significantly to the final result.

For a WAV file at 44100 Hz a step size of 256 would mean to go through the music in steps of 5.8 ms where a step size of 2048 would mean to process the data in steps of 46.4 ms. The latter one should be fine enough for most applications.

2.1.4 Frame – Time Conversion

The conversion from a frame number to the corresponding time and vice versa is also an important need for the whole analysis since the programme uses frames (cf. figure A.3) to access the locations in the audio data. The conversion from a frame number to the corresponding time can be calculated using the following formula

$$\text{time[s]} = \frac{\text{frame} \times \text{step size}}{\text{sample rate [Hz]}}$$

or vice versa via the formula

$$\text{frame} = \frac{\text{time [s]} \times \text{sample rate [Hz]}}{\text{step size}}.$$

Calculating a frame number with the given formula can result in a non-integer value. Since the programme only processes integer values for the frame numbers, one has to round.

Rounding should be done in the following way: if an area of frames has to be specified, always a bigger area should be covered to be on the safe side. This means, considering for example an area from frame 82.9 to frame 85.1, the area should be specified by the user with frames 82 until 86. If only a frame should be specified – say to specify the reference frame –, one can round in the standard mathematical sense, if needed.

2.2 Similarity Measurement

To find a frame that is similar to a given reference frame, there are several possibilities for a similarity measurement, e.g. the Euclidean distance of the power spectra of these two frames can be minimized or the scalar product of the power spectra of these two frames can be maximized. The latter is a good solution for the similarity measurement as proposed by [6].

Minimizing the Euclidean distance would mean to compute

$$\left\{ i \;\middle|\; \min_i \|\hat{f}_i - \hat{f}_{ref}\| \right\}$$

where $\hat{f}_{ref}$ is the power spectrum of the reference frame and $\hat{f}_i$ the power spectrum of the frame currently compared to the reference frame.

The power spectrum can also be understood as a one-dimensional vector which carries all the coefficients of the power spectrum. This makes it possible to maximize the scalar product, i.e. computing

$$\left\{ i \;\middle|\; \max_i \frac{\langle \hat{f}_i, \hat{f}_{ref} \rangle}{\|\hat{f}_i\| \cdot \|\hat{f}_{ref}\|} \right\}$$

where the scalar product is normalized by the standard vector norm of the two vectors $\hat{f}_i$ and $\hat{f}_{ref}$. This is used in the context of this thesis.

In both cases the result is the index i for which the result is minimal or maximal, respectively.

2.3 Summary

In this chapter, the basic ideas for the similarity measurement were presented. Starting with the short-time Fourier transform which adds a spatial/temporal component to the ”classical” Fourier transform by considering a window which is multiplied to a part of the signal where the Hamming window was introduced as a suitable windowing function. This window allows to treat the boundaries in a nice way by considering the central point as the most important one. Followed by a few lines about the step size and window width, the Hamming window and the short-time Fourier transform are deepened. Using something else than a hard windowing like a box function was pointed out as important since problems would arise with the boundaries and the weighting. The prerequisites were concluded with a simple formula to convert a frame number to the corresponding time and vice versa.

As a similarity measurement, two approaches were discussed, the minimization of the Euclidean distance or the maximization of the scalar product, where the latter one is used in this thesis.

Chapter 3

Non-Local Algorithms in Image Processing

3.1 Overview

This chapter gives a short overview over several methods used for the task of denoising in image processing. Starting with non-local inpainting, a method is introduced which (re-)uses patches of the image to repair noisy parts. This method is followed by NL-means, a currently hot topic in image processing. This is due to the fact that it is easy to understand and very powerful in denoising structures.

3.2 Non-Local Inpainting

Inpainting is an iterative method for repairing damaged pictures or removing unnecessary elements from pictures. The key idea is to copy patches already present in an image or blow up structures to repair missing, unnecessary or noisy data. To find possible candidates for the repairing step, a suitable similarity measurement is used, for example the Euclidean distance as introduced in 2.2. When dealing with non-local inpainting the search is even done on a whole neighborhood. After that, a candidate that is categorized as similar is copied to the noisy position including some neighborhood. The patch to be copied doesn't necessarily have to be the best match.

This idea was used for example in [7] where image texture is synthesized.

3.3 NL-Means

Non-local means (or short NL-means) – proposed by [5] – is as simple as powerful in image processing. The simple idea is to look for similar patches in an image, average them and repair the noisy data with the calculated averaging. This is done by first finding pixels with the same grey value as the one to be repaired and compare the geometrical configuration of their neighborhood.

For the weighting function, an exponential function is used which has the following structure (in the discrete case):

$$w(i,j) = \frac{1}{Z(i)} e^{-\frac{||v(\mathcal{N}_i)-v(\mathcal{N}_j)||^2_{2,a}}{h^2}}$$

where $Z(i)$ is the normalizing constant

$$Z(i) = \sum_j e^{-\frac{||v(\mathcal{N}_i)-v(\mathcal{N}_j)||^2_{2,a}}{h^2}}$$

and the parameter h acts as a degree of filtering.

This weighting function penalizes structures which have less similarity in the geometrical configuration of their neighborhood (cf. figure 3.1).

As a similarity measurement, NL-means uses the Euclidean distance which is minimized as introduced in 2.2.

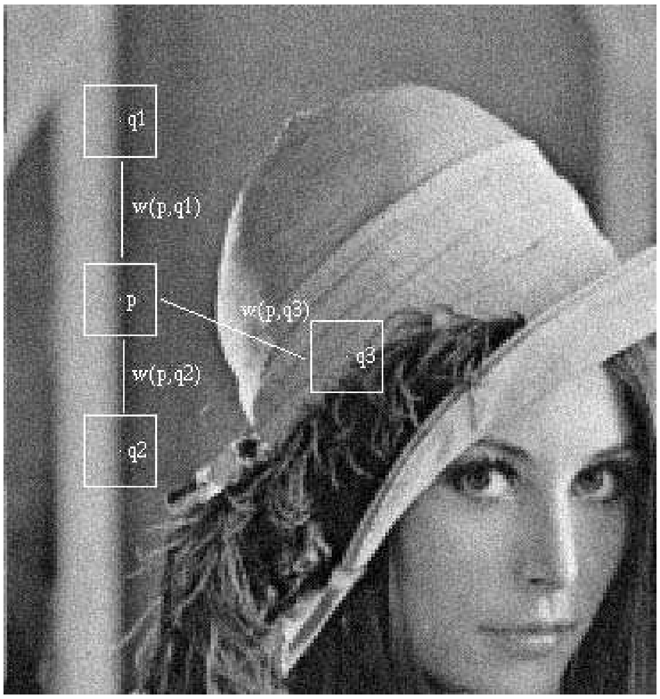

Figure 3.1: Scheme of NL-means strategy. Similar pixel neighborhoods give a large weight, $w(p, q_1)$ and $w(p, q_2)$, while much different neighborhoods give a small weight $w(p, q_3)$. **Authors:** [5].

Chapter 4

Non-Local Algorithms in Audio Processing

4.1 Overview

This chapter is dedicated to the modifications done in the context of this thesis to bring the ideas introduced in chapter 3 to audio signals. It is shortly discussed how stereo files are processed. Furthermore, the chapter shows how the similarity measurement is refined to match the needs to compare different frames followed by an algorithm for simple inpainting, the modification to realize a simple averaging and a NL-means like approach.

4.2 Stereo Files

For a stereo file both channels cannot be treated separately for obvious reasons. To be able to process stereo files, a simple assumption is made: internally the file is processed as a mono file. This is done by averaging the two channels in the sense of the arithmetic mean. Afterwards when writing back the file, the data is doubled to both channels so that a (pseudo) stereo file comes out again. No further work is done in this direction since stereo files are not the main focus of this thesis. If not explicitly stated, always mono files are processed.

4.3 Similarity Measurement

In chapter 2, possible similarity measurements were introduced and all prerequisites were made to formulate the similarity measurement used in this thesis. The only thing that has to be done is computing windowed Fourier transforms by using the Hamming window in the following way:

1. Take a part of the audio signal of length N.

2. Apply the Hamming window.

3. Use a standard Fast Fourier Transform (FFT) on this window to compute the coefficients.

4. Compute the power spectrum for each window

$$|\hat{f}|^2 = \mathrm{Re}^2(\hat{f}) + \mathrm{Im}^2(\hat{f})$$

   where $\mathrm{Re}^2(\hat{f})$ denotes the real part of the Fourier transform and $\mathrm{Im}^2(\hat{f})$ denotes the imaginary part of the Fourier transform, respectively.

5. Go step size many steps forward and repeat.

Note, this would mean for the example given in 2.1.3 that every 256 samples a window of width $N = 1024$ is considered.

Furthermore, a few more refinements have been made to reach a higher accuracy of the proposal of similar frames where all of the following refinements are user inputs to give the user the highest flexibility possible in this context.

First of all, the noisy part of the data is excluded. This is obvious since the noise cannot be trusted. This is especially important if one has a larger degradation of the data since the accuracy is higher then. Of course, the user can also decide not to exclude anything.

Secondly not only the reference frame itself but a neighborhood is considered, so some kind of non-local approach is introduced as also done in 3.2. This is important since a certain tone or musical event can appear in a lot of different contexts, for example the tone A can be part of G – A – B as well as C – F – A. Not considering the context could result in a unnatural listening experience when replacing one ”same” tone by another since it could sound different due to its surrounding (cf. figure 4.1).

A third refinement is to require a minimal distance of the currently observed frame to the reference frame since it is expected to find the most likely matches around the reference frame itself, if not excluded. This is fixed to 10 which gives good results.

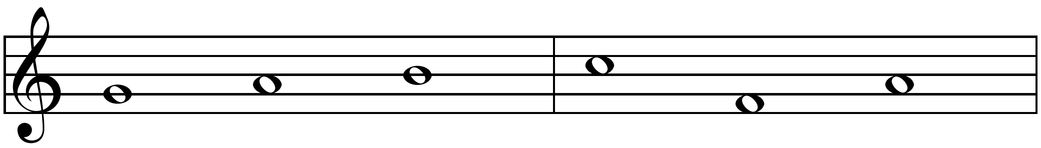

Figure 4.1: Example for two different occurrences of the tone A. **Left:** G – A – B (precisely: g’ – a’ – b’). **Right:** C – F – A (precisely: c” – f’ – a’).

4.4 Simple Inpainting

Simple inpainting for audio signals is inspired by the ideas of [7]. After having found the frames similar to the reference frame, the programme replaces the neighborhood – specified by an user input – around the reference frame including the reference frame itself by the same neighborhood around the frame to be similar, also including this one. There is no mirroring at the boundaries used which means that the inpainting doesn't consider the parts that are exceeding the boundaries. Assuming Neumann boundary conditions for an image processing problem is fine because the assumption only requires to mirror one pixel. In the case of audio signals, a small piece of music to mirror in contrast to the whole song would not suffice. Here one would have to mirror at least half of the width of a window. This would mean to mirror at least 2048 sample points in the case of a window width of 4096, because if the frame currently observed is centered at 0, then one would need 2048 to the left and 2048 to the right where the center could be regarded as part of the left or the right half. Another problem arises when mirroring at the boundaries: the music would be analyzed in a "reverse order" when passing the boundary. Introducing such a big error in contrast to not using data beyond a boundary is too inaccurate.

Let's imagine the user specified a neighborhood of 515 frames then in total 1031 frames are replaced, namely 515 frames left to the reference frame, the reference frame itself and 515 frames right to the reference frame. To do so, the start and end point in the wave data for the reference frame and the similar frame are calculated via the simple formulae

$$\begin{aligned}
start_ref &= (ref - n) \times STEPSIZE \\
end_ref &= (ref + n) \times STEPSIZE \\
start_frame &= (frame - n) \times STEPSIZE \\
end_frame &= (frame + n) \times STEPSIZE
\end{aligned}$$

where ref is the reference frame, $frame$ the similar frame and n the neighborhood. A simple example shows how this works: Imagine to have a step size $STEPSIZE = 256$. The reference frame is at 14298 and the neighborhood to include should be 515. Then the algorithm replaces the slots $start\_ref$ until end_ref in the original wave data:

$$\begin{aligned}
start_ref &= (14298 - 515) \times 256 \\
&= 3528448 \\
end_ref &= (14298 + 515) \times 256 \\
&= 3792128
\end{aligned}$$

This also holds for the following approaches in 4.5 and 4.6 since here only the data to be used is modified and not the method how to repair noisy part.

4.5 Averaging on a Sample Basis

With simple inpainting only a selected patch is filled in. In this approach, all the frames found as similar are averaged in the sense of the arithmetic mean where the result of this is copied at the position of the noisy signal, also possibly including a certain neighborhood. The number of frames classified as similar doesn't exceed the user input which means if the user chooses to find 10 similar frames then the number of frames proposed as similar is at most 10 but could be less.

This approach can be seen as a weighted averaging with a fixed weighting function

$$w = \frac{1}{frames}$$

where $frames$ is the number of frames classified as similar.

4.6 NL-Means

In principle, NL-means is a further development of averaging on a sample basis introduced in 4.5. Instead of using a fixed weighting function, an adaptive weighting with an exponential function is used here.

Since the original NL-means algorithm uses the minimization of the Euclidean distance and for this thesis the maximization of the scalar product is used, the weighting function has to be slightly changed: The programme gives a list of similar frames with their corresponding scalar products. These are used for the weighting function in the following way:

$$w(cur) = e^{-\lambda \cdot \frac{best-cur}{best}}$$

where cur is the scalar product as stated in 2.2 of the currently observed frame, $best$ the scalar product of the best match (also in the sense of 2.2) and λ a threshold parameter.

Chapter 5

Experiments

5.1 Overview

This chapter summarizes the experiments done during the development of this thesis. It delivers some insights in how the memory requirements evolved from the beginning until now, how the runtime behavior of the programme is and what methods are possible to repair defects in music.

Here is a list of files that are used where their corresponding degradation with an approximate time location in the audio file is stated in brackets. The list starts with self-produced files:

- *cyftlt-mono-short_sine.wav* (sine tone, time: 1:53.14 – 1:53.23),
- *cyftlt-mono-middle_sine1.wav* (sine tone, time: 1:52.99 – 1:54),
- *cyftlt-mono-middle_sine2.wav* (sine tone, time: 1:23.98 – 1:24.96),
- *cyftlt-mono-long_sine.wav* (sine tone, time: 1:20.01 – 1:25.04),
- *cyftlt-long_sine.wav* (sine tone, time: 1:20.01– 1:25.00),
- *piano1-mono-white.wav* (white noise, time: 0:02.01 – 0:03.02),
- *piano2-mono-sine.wav* (sine tone, time: 0:01.98 – 0:03.09),
- *piano3-mono-sine.wav* (sine tone, time: 0:03.75 – 0:04.70),
- *piano3-mono-short_white.wav* (white noise, time: 0:04.40 – 0:04.45)
- *t-style-mono-white_sine.wav* (white noise, time: 0:09.95 – 0:10.41, and sine tone, time: 0:59.04 – 0:59.63).

The files listed above were all synthetically created by using the MIDI sequencer software Propellerheads Reason 4[1] and edited afterwards with Audacity[2].

Also, two audio files from the CD "Elton John: A Piano Tribute" by Boko Suzuki are used where they are also edited afterwards with Audacity. These are:

- *candle_in_the_wind-mono-silence.wav* (complete silence, time: 0:46.50 – 0:47.00)
- *your_song-mono-white.wav* (white noise, time: 1:31.80 – 1:32.60)

The type of degradation doesn't matter if the noisy part is left out which is possible as already mentioned in 2.2.

5.2 Parameter Selection for Step Size and Window Width

As already mentioned in 2.1.3, the wave file is processed by going through the data in certain steps while looking at a window of a certain width for each step. Two different configurations have been explored. In the first configuration, the step size was set to 256 and the window width to 1024, resulting in an overlap of 75%. In the second configuration, the step size was set to 2048 and the window width to 4096, resulting in an overlap of 50%.

Unfortunately not all the files could be reconstructed with the first parameter setting during the development of the thesis. The file *t-style-mono-white_sine.wav* has white noise in the first half and a degradation by a sine tone in the second half. With the first configuration only the white noise could be removed.

Using the second configuration, the white noise on the one hand side and the sine tone on the other hand side in *t-style-mono-white_sine.wav* could be removed but not both.

5.2.1 Memory Requirements

The first configuration with a step size of 256 and a window width of 1024 works pretty well at least in finding the best similar frame. Unfortunately it needs for example for a 21 MB wave file in mono[3] – 4:04 minutes of piano music – about 300 MB of main memory which is pretty huge in comparison to the actual file size.

The second configuration with a step size of 2048 and a window width of 4096 works as good as the first one, but the memory requirements drop for the same file as above from about 300 MB to about 165 MB.

At the beginning of the thesis, the step size was set to 50 and the window width to 1024 resulting in a memory requirement of more than 3 GB for the same 21 MB file as above.

[1] http://www.propellerheads.se/
[2] http://audacity.sourceforge.net/
[3] *cyftlt-mono-long_sine.wav* was used in this example.

These relatively small values in comparison to the 3 GB are possible because two rather simple tricks are used. The first trick is not to store the whole window in a small array for the real part, a small array for the imaginary part and a small array for the spectrum for every step size but to store all the data in one large array for the real part, a large array for the imaginary part and a large array for the spectrum and to use only pointers to state which values belong to which window. To save memory is expected in this case since it is always ”rounded” to next power of 2 when allocating memory. For example, allocating three times 513 slots of floats results in allocating in three times 1024 slots of floats. Allocating one big array would result in 2048 slots of floats instead of 3072 slots. This becomes even more important if this allocation step is done more than 1,000 or 10,000 times.

A second trick is based on the fact that the Fourier Transform is symmetric. The symmetry makes it possible to throw away half of the data and recompute it later when needed.

The resulting memory requirements are actually expected. To verify this, an exemplary calculation is made for the case that the step size is 256 and the window width is 1024 where *cyftlt-mono-long_sine.wav* is used as input file. The actual data size of this file is 21,520,800 bytes and it has 16 bits per sample.

$$\begin{aligned}
\text{datasize} &= \begin{cases} 21520800 & \text{, bits per sample = 8} \\ 21520800/2 & \text{, bits per sample = 16} \\ 21520800/3 & \text{, bits per sample = 24} \end{cases} \\
\text{arraysize} &= \text{datasize}/\text{step size} \\
&= 10760400/256 \\
&= 42032.8125 \approx 42033 \\
\text{width} &= 1024 \\
\text{ft_size} &= (\text{width}/2) + 1 = 513
\end{aligned}$$

Now it is possible to calculate the whole memory requirement for this file which is given in table 5.1. Furthermore, the operating system needs some extra space for the memory management which increases the calculated requirement (287.82 MB) to a actual memory requirement of 288.66 MB. To conclude this example there has to be said that a few helper arrays are created for computational purposes and destroyed immediately after the computation. These are in the range of about 80 to 100 MB[4] which increases the whole main memory requirement to a worst case maximum of 400 MB during programme run.

5.2.2 Runtime

The runtime of the programme clearly depends on several factors. One important factor is the dependency on the problem itself, i.e. the size of the file being processed, the size of the

[4]actual data size $\times$ sizeof(float) = 21520800 bytes $\times$ 4 = 86083200 bytes $\approx$ 82 MB

Description	Size		[bytes]
WAV data array	datasize × sizeof(float)		43041600
real part of FT data	arraysize × ft_size × sizeof(float)	+	86251716
imaginary part of FT data	arraysize × ft_size × sizeof(float)	+	86251716
spectra of FT data	arraysize × ft_size × sizeof(float)	+	86251716
array with similar frames	how_many_similar_frames × sizeof(int)	+	40
array with distances	how_many_similar_frames × sizeof(float)	+	40
		=	301796828
		≈	287.82 MB

Table 5.1: Example calculation for the memory requirement of *cyftlt-mono-long_sine.wav*.

neighborhood, the extend of the noise and so on. The measurements here should give a flavor of how fast the programme approximately is. All files are processed on an Apple MacBook Pro with a 2.2 GHz Intel Core 2 Duo and 2 GB of RAM.

Table 5.2 corresponds to the wave file already mentioned above. Please note that the approximate location in the piece of music is stated in brackets in the format (mm:ss).

step size / window width	256 / 1024	2048 / 4096
Reference frame	14298 (1:23)	1787 (1:23)
Minimal distance	10	2
Search neighborhood	515 (3 sec.)	65 (3 sec.)
Frames to exclude	13781 – 14643 (1:20 – 1:25)	1723 – 1830 (1:20 – 1:25)
Replacement neighborhood	515	65
Runtime	≈ 28 seconds	≈ 7 seconds

Table 5.2: Processing of *cyftlt-mono-long_sine.wav*, duration: 4:04 minutes (size 21 MB).

Table 5.3 shows the same scenario for a 700 KB wave file.

Furthermore the programme should give the 10 best results and the best result was chosen for the replacement. Of course, only the computation time is measured.

One point is of importance here, namely that the coarser scale is about four times faster while giving the same best match.

5.3 Simple Inpainting

Figures 5.1 and 5.2 show exemplarily how a noisy signal looks like and how the simple inpainting works to repair the signal.

step size / window width	256 / 1024	2048 / 4096
Reference frame	861 (0:05)	107 (0:05)
Minimal distance	10	2
Search neighborhood	525 (3 sec.)	65 (3 sec.)
Frames to exclude	646 – 810 (0:03.75 – 0:04.7)	81 – 101 (0:03.76 – 0:04.7)
Replacement neighborhood	175	22
Runtime	≈ 2 seconds	≈ 0.5 seconds

Table 5.3: Processing of *piano3-mono-sine.wav*, duration: 0:08 minutes (size 700 KB).

Figure 5.2 also clearly shows the limitation of this simple inpainting method. *piano3-mono.wav* contains actually two chords (A minor and E minor) where the first and the third chord are A minor and the other two E minor. Furthermore it is noisy in such a way that the degradation starts shortly before the third chord. The degradation is actually a sine tone similar to the tone E. The programme finds a similar frame which is at the beginning of this short sequence, but cannot fill in information at the beginning of the noise, because there is obviously no data before the first chord which could be plugged in due to the fact that no mirroring is used.

A problem also encountered with simple inpainting is that after replacing the noisy data with good one the music from this point on is shifted a little bit to the left – in other words: a little part is cut out at the insertion point – in some configurations. This is actually something that is not expected.

Both, *cyftlt-mono-long_sine.wav* and *piano3-mono-sine.wav* are somehow extreme examples. In the case of *cyftlt-mono-long_sine.wav* about 5 seconds of completely missing data should be reconstructed which is a pretty tough task. From this viewpoint, simple inpainting works great with the right parameter setting.

5.4 Averaging on a Sample Basis

When performing averaging on a sample basis with the same weight for every sample point the number of the data to be averaged plays an important role. Experiments have shown that for a small number of similar frames to be averaged the result is relatively good in reconstructing something close to the original while on the other hand providing an unnatural listening experience. For the test file *cyftlt-mono-long_sine.wav* already the task of finding 4 or more similar frames and average them results in something that can be called chaotic when expecting something close to the original: different pieces of music overlap that do not actually fit together. This can be compared to having say five musicians playing all different parts of a piece of music. The problem here is that the other frames have a distance to the reference

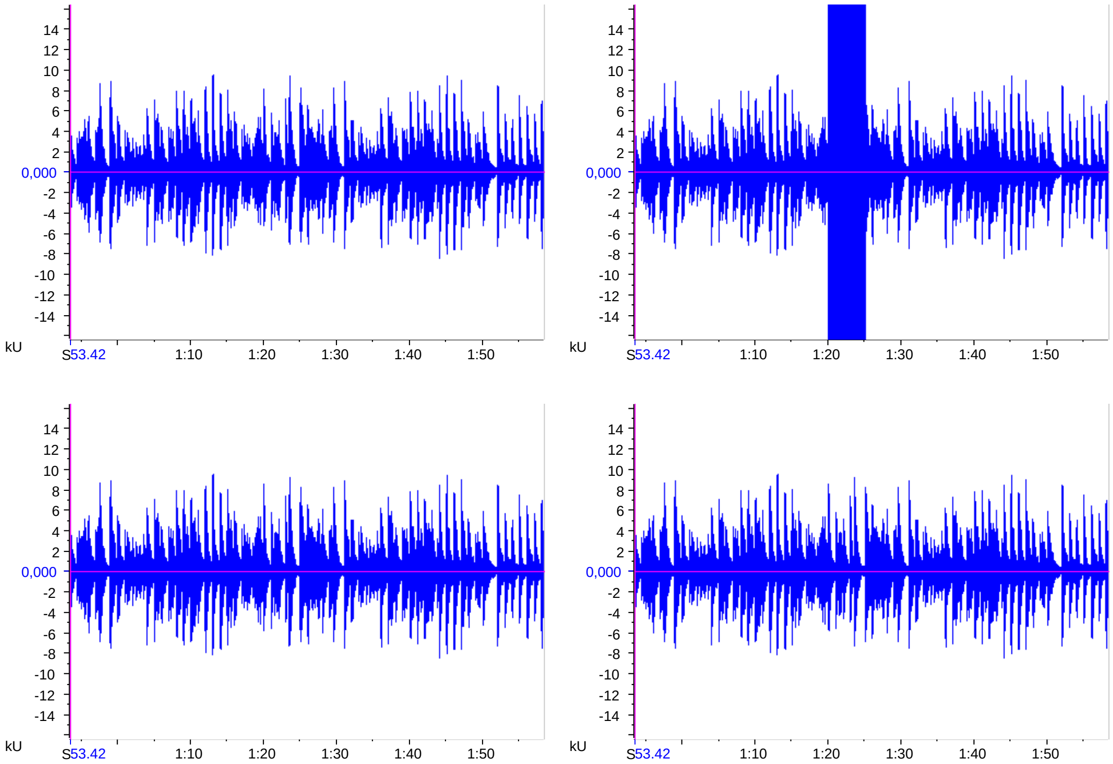

Figure 5.1: Correction with simple inpainting for *cyftlt-mono.wav* (4:04 minutes). **Top left:** Original file. **Top right:** Original degraded with a sine tone of 440 Hz (*cyftlt-mono-long_sine.wav*). **Bottom left:** Corrected version with a step size of 256 and a window width of 1024. **Bottom right:** Corrected version with a step size of 2048 and a window width of 4096.

frame which is large enough to stand at the fourth or higher position in the similarity list, but these frames are far away so that they cannot be possible candidates for the noisy frame.

Nevertheless, the result is somehow better than expected from the viewpoint that the algorithm finds pieces that fit harmonically. Not knowing the song might result in acceptance of the result. This is due to the fact that the programme doesn't have any previous knowledge of musical structures.

For the case that 3 or less similar frames are considered – in other words the first three slots in the similarity list – the result is reconstructing the noisy part, but is louder and the frames are not matched exactly: they are a little bit shifted. This is a problem already mentioned in 5.3.

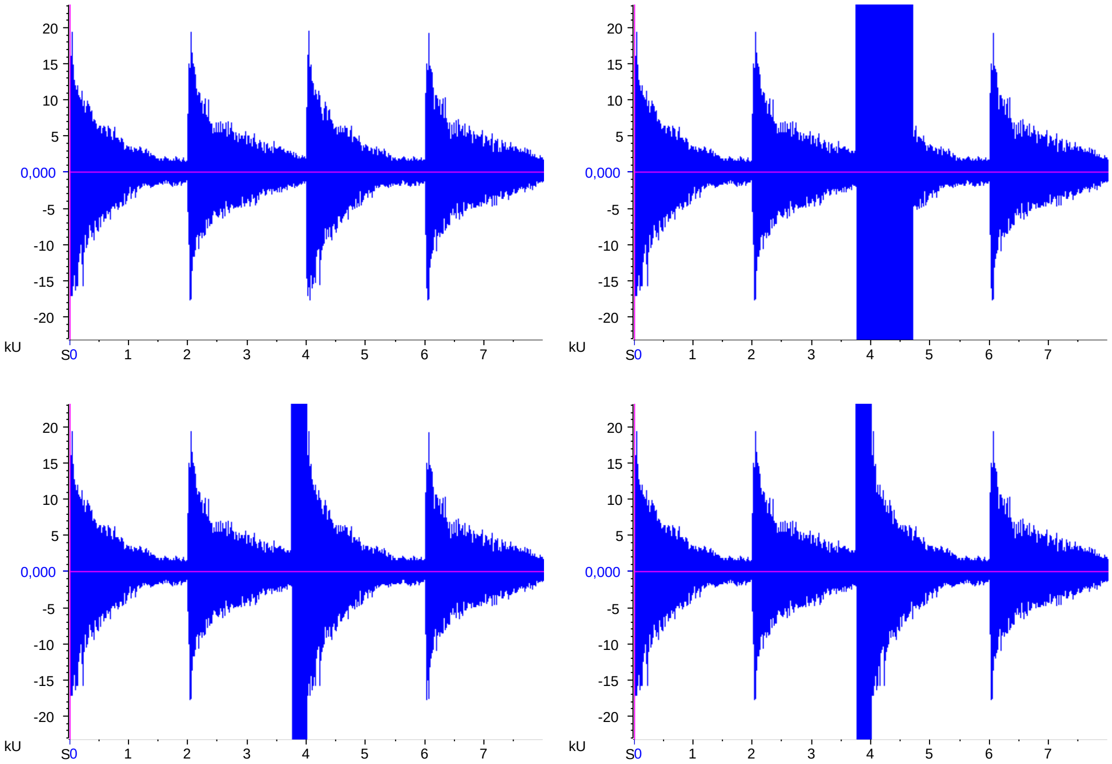

Figure 5.2: Correction with simple inpainting for *piano3-mono.wav* (0:08 minutes). **Top left:** Original file. **Top right:** Original degraded with a sine tone of 330 Hz (*piano3-mono-sine.wav*). **Bottom left:** Corrected version with a step size of 256 and a window width of 1024. **Bottom right:** Corrected version with a step size of 2048 and a window width of 4096.

5.5 NL-Means

Since the NL-means approach is a further development of the averaging on a sample basis method it basically suffers from the same problems described in 5.4.

Experiments have shown that the threshold parameter λ significantly contributes to the final result. Choosing the threshold higher and higher results in a better – for an image one would say sharper – denoising. Of course, choosing λ too large results in the inpainting approach presented in 4.4 since the exponential function weights the best match always with 1 and all the other matches with 0.

5.6 Results

Experiments have shown so far that averaging on a sample basis is no good idea. The error introduced by this method is too high in comparison to the gain which one gets, if any.

One important problem is the sensitivity as far as the choice of parameters is concerned. Taking too much frames from the similarity list into account may result in an overlapping of different fragments of music. simple inpainting obviously performs better in this case, at least if one of the first entries in the similarity list is chosen. This is of course not surprising since simple inpainting actually inserts only a frame matching the noisy one.

To conclude the methods used so far one can say that both methods have their limitations. Averaging on a sample basis seems to be a complete failure where simple inpainting is as good as the frame to be inserted.

To this end, the results for the simple inpainting approach and the approach to average on a sample basis both perform as expected.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

As presented in this thesis, it is possible to analyze music coming from an image processing background. It was presented how to change tools like the Fourier transform to fit the needs of music analysis. Also, it was shown how to look at music, namely by windowing it and going forward in time by having different windows overlap each other to get more than one contribution to the currently observed part while computing the power spectrum for each window.

By having the simple ingredients like the short-time Fourier transform and the Hamming window as a windowing function, one could proceed as it was done in this work: interpret the power spectrum of each window as a one-dimensional vector which can be used for the similarity measurement. To find out what frames are similar, the scalar product of two vectors – the power spectrum of the reference frame and the power spectrum of the currently observed frame – comes into play where the result is normalized using the vector norm of each vector.

Experiments showed different methods how to replace a noisy part with a non-noisy one.

6.2 Future Work

So far, only the maximization of the scalar product is used as a similarity measurement. Maybe there exist other similarity measurements that can be used to find similar frames.

Also, repairing multiple degradations at the same time – i.e. in one run – could be a possibility to make the denoising process more comfortable.

Another possible work for the future is to make the software more user-friendly. Maybe there is also a way to reduce the parameter. A possibility could be to establish a dependency between the neighborhood which is considered for the search and the neighborhood which is actually replaced then.

Appendix A

Wave Files

A.1 Wave File Format

The wave file format [4] was developed by Microsoft and IBM and is nowadays the standard audio file format for storing an uncompressed audio file on a computer. It is based on the RIFF bitstream format [3] which allows to store the data in different chunks. The default bitstream encoding is the Microsoft Pulse Code Modulation (PCM) format. Unfortunately, the format of a wave file is not uniquely defined because it can have a lot of custom chunks. This thesis only considers the following format where all other types of chunks are not processed:

In figure A.1 a possible arrangement for the data of a WAV file is depicted.

A.2 Meaning of the Data

A WAV file is a discrete version of an audio signal and describes the development of an oscillation in time. The quality of a wave file is determined by the sample rate. An example for a common visualization of a WAV file is given in figure A.2.

Assuming to have a stereo file – i.e. having two channels –, the data of a WAV file is arranged as depicted in figure A.3. Here can be seen that the first sample point of all channels is combined to build the first sample.

In the audio sector often the sample points in figure A.3 are called samples and the samples in figure A.3 are called frames. Since one needs a natural expression that builds on top of these two concepts, the renaming is performed in the presented way.

WAV files are stored in general in little endian where the least significant bit stands at the first position. If the length cannot be divided by 8 a zero-padding is applied. Figure A.4 shows this nicely.

Position [bytes]	Field Name	Field Size [bytes]	Description / Contents
0	Chunk ID	4	”RIFF”
4	File Size	4	Entire File Size - 8
8	File Format	4	”WAVE”
12	SubChunk1 ID	4	”fmt ” (header signature)
16	SubChunk1 Size	4	16 for PCM files.
20	Audio Format	2	1 for uncompressed audio.
22	Number of Channels	2	1 = mono, 2 = stereo, ...
24	Sample Rate	4	44100 Hz for CD quality
28	Byte Rate	4	Sample Rate $\times$ Block Alignment
32	Block Alignment	2	Channels $\times$ Bits Per Sample $\div$ 8
34	Bits Per Sample	2	8, 16 or 24
36	SubChunk2 ID	4	”data” (header signature)
40	Data Size	4	Number of bytes following the header.
DATA			

Table A.1: The WAV file format. **Adapted from:** Wikipedia [4].

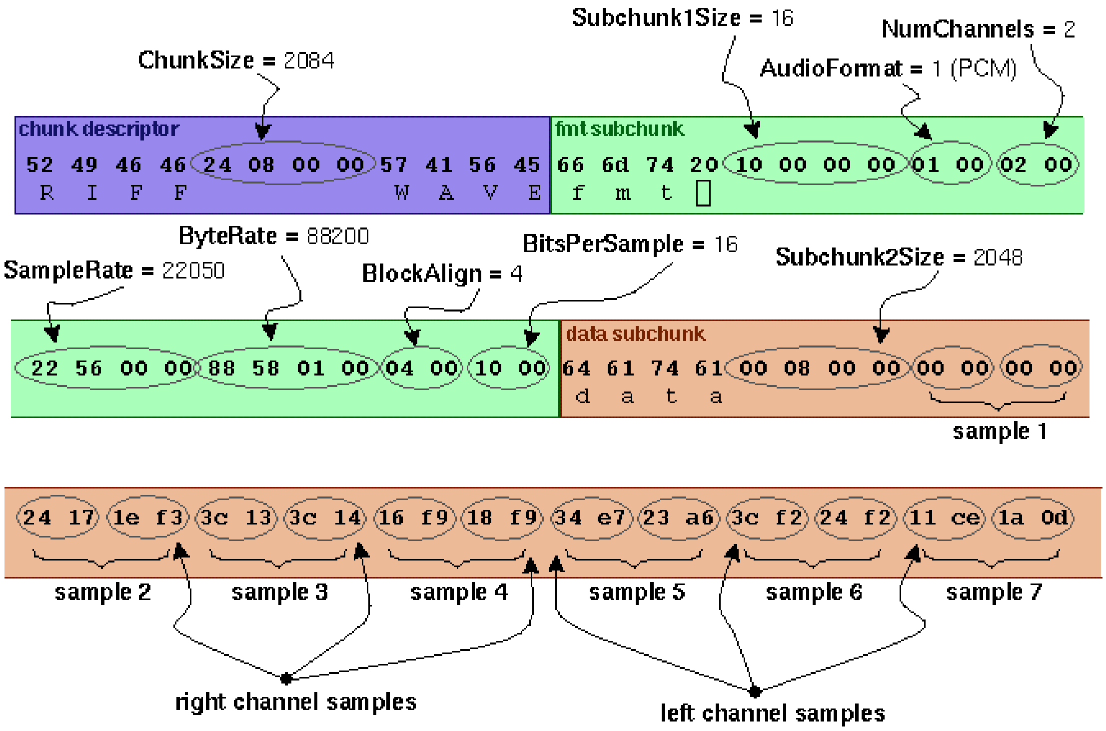

Figure A.1: First 72 bytes of an example WAV file. **Source:** `http://ccrma.stanford.edu/courses/422/projects/WaveFormat/`.

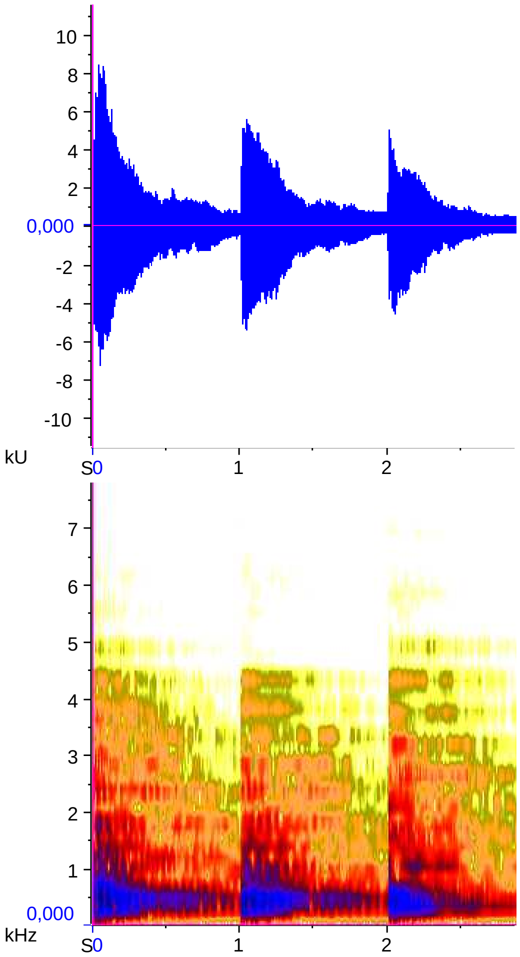

Figure A.2: Example for a visualization of a WAV file. **Left:** First three chords of *cyftlt-mono.wav*. **Right:** Zoom into the first chord of the same file. **Top:** Waveform representation. The x-axis depicts the time in seconds where the y-axis shows the amplitude. **Bottom:** Frequency representation. The x-axis depicts the time in seconds where the y-axis shows the frequency distribution. Colors give the intensity of a certain frequency.

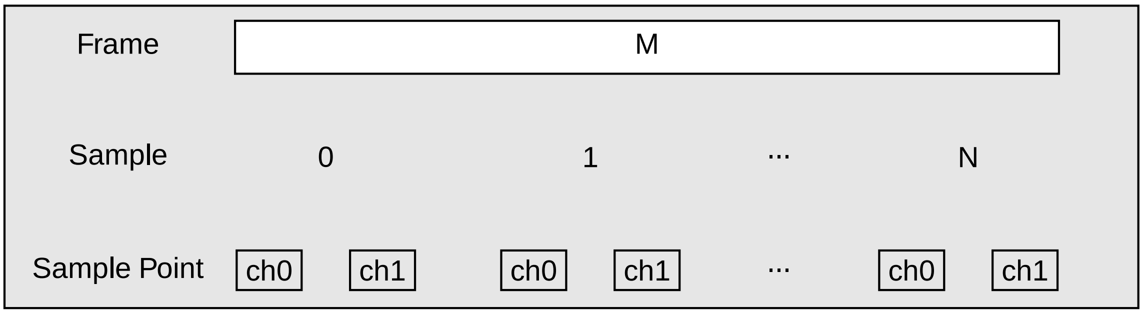

Figure A.3: Difference between frames, samples and sample points for a stereo WAV file. In case of this thesis, M starts with 1 and N is the window width. **Adapted from:** Wikipedia [4].

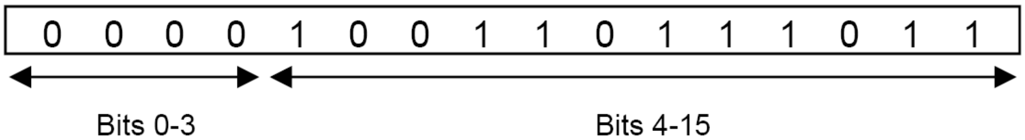

Figure A.4: Bitordering for an example 12-bit sample. The 12 bits are padded by zeros to get 16 bits, which can be divided by 8. **Source:** Wikipedia [4].

Bibliography

[1] Different window functions. `http://en.wikipedia.org/wiki/Window_function`. Retrieved: 2008-08-16.

[2] RavenLite version 1.0. `http://www.birds.cornell.edu/brp/raven/RavenOverview.html`. Downloaded: 2008-08-08.

[3] RIFF format. `http://en.wikipedia.org/wiki/Resource_Interchange_File_Format`. Retrieved: 2008-08-16.

[4] Wave file format. `http://en.wikipedia.org/wiki/Wav`, `http://de.wikipedia.org/wiki/Wav`. Retrieved: 2008-08-16.

[5] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Volume 2:60 – 65, June 2005.

[6] Matthew Cooper and Jonathan Foote. Automatic music summarization via similarity analysis. *2002 International Symposium on Music Information Retrieval*, October 2002.

[7] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. *IEEE International Conference on Computer Vision*, September 1999.