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 Non-Local Algorithms from Image Processing

Jan Hendrik Dithmar

2008-10-1

Supervisor	Prof. Dr. Joachim Weickert
Advisor	Dr. Martin Welk
Reviewers	Prof. Dr. Joachim Weickert
	PD Dr. Meinard Müller

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 this thesis.
I confirm that the electronic and the printed version are of identical content.

Saarbrücken, 2008-10-1

Abstract

Noisy structures can appear in different variations in images. They might appear as noise that is present in the whole image, like Gaussian white noise, salt-and-pepper noise or speckle noise. Speckle noise, for example, is caused by the measurement of an ultrasound image. But noise can also appear locally in an image, e.g. as a black spot in the middle of a color image. These concepts are also present in audio signals. On the one hand side one can have Gaussian noise that degrades a whole signal or on the other hand side a structure that appears locally, for example a cough. Of course, noise in images and audio signals is different if one looks at them in detail, but the basic ideas of noise are present in both fields. This thesis investigates noise that appears locally as an isolated and intense event in the audio signal.

In the last few years, research interest was focussed on non-local algorithms in image processing by many groups. Since these algorithms worked well and were very easy to understand in most cases, the investigation of those is still going on. An algorithm that is very popular in that respect is the NL-means approach by Buades, Coll and Morel [8]. Furthermore, non-local algorithms in image processing are based on the same assumptions that could also be formulated for audio processing. An assumption is to find similar patches in an image or audio signal by doing a Fourier transform and analyze the spectrum afterwards by an appropriate similarity measurement. Having found similar patches, one can try to repair using a non-local approach.

This thesis gives an overview of the modifications in terms of the Fourier transform and the NL-means approach that are necessary to adapt these methods known from image processing to audio signals. Also, the usage of non-local inpainting – i.e. copying data already present in the audio signal – and an averaging on a sample basis are introduced. At the end, all the presented methods are evaluated by experiments.

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 in the Mathematical Image Analysis Group greatly assisted me in my work. I also like to thank PD Dr. Meinard Müller for consenting to review my thesis.

Special thanks also go to my family who enabled me to perform my studies and to pursue my musical career 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. When recording a live performance it can happen that a certain part is degraded that cannot be re-recorded again, for example caused by a cough.

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

Naturally, one is interested in correcting noisy parts of a recording. To achieve this, one tries to exploit the high redundancy in music. Typically, one would proceed manually as follows: people would have 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 (manually) 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 the y-axis and the intensity of the frequencies in a color representation. But also other representations exist. An example is shown in figure 1.1.

During this thesis, a preprint by A. Szlam [14] appeared which also studies the application of the NL-means algorithm as proposed by [8] for audio denoising. While Szlam investigates speech as audio data where the whole signal is degraded by Gaussian white noise, this thesis covers music data – to be more precise: piano music – where the noisy events appear only locally. Another difference is that the work by Szlam uses noise that is quite moderate while the noise investigated in this thesis is very intense. Nevertheless, the averaging approach which is the key of the NL-means algorithm is similar in both approaches, the one by Szlam and the one here.

Both, image processing and audio processing use the Fourier transform [7] to analyze the corresponding data which gives some evidence that it is also possible to adapt other image processing methods to do audio processing. The concept of noise is also present in both settings. Noise in an image can be something that degrades the image as a whole – Gaussian white noise or salt-and-pepper noise would be examples for this – or can appear locally. A local appearance of noise could be e.g. a black spot in the middle of someone's face in a color

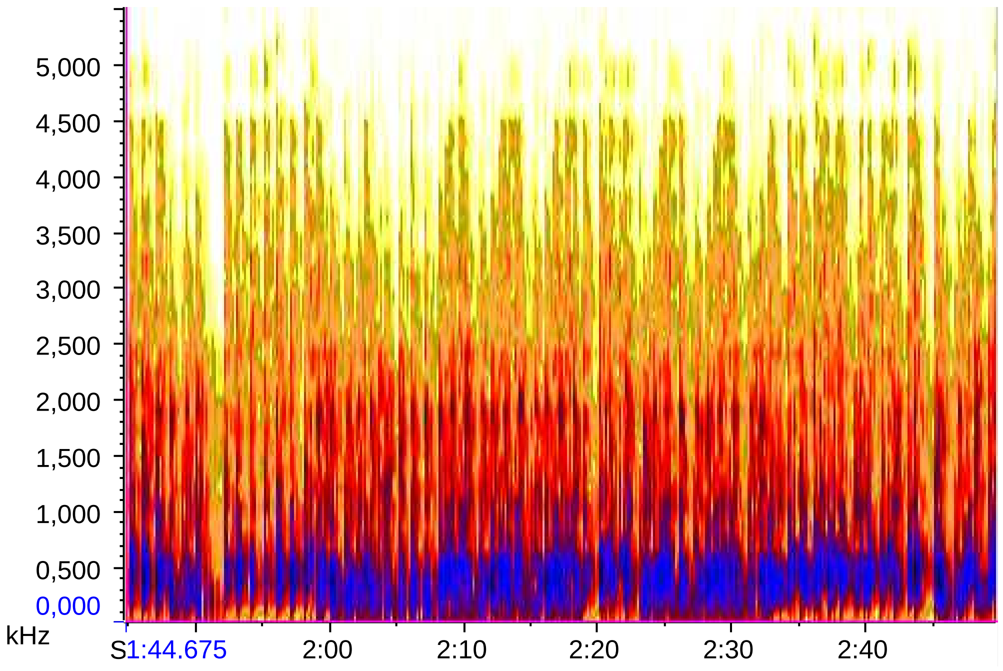

Figure 1.1: Spectrogram representing a part of cyftlt-mono.wav. *x-axis:* Time. *y-axis:* Frequencies. *Colors:* Intensity. The blue color stands for a high intensity going over red and yellow to white which stands for a low intensity. **Visualization:** using RavenLite 1.0 [4].

image. Methods to repair a defect like the non-local inpainting (see [10]) and the NL-means algorithm [8] are well understood in image processing.

In audio files, these types of noise can also appear. One might have to deal with Gaussian white noise as well as with degradations that appear locally like a cough. An in-depth introduction to audio processing is given by M. Müller in [12]. Degradations like Gaussian white noise are investigated by A. Szlam [14] and shortly touched here. In this thesis, the main focus lies on isolated noisy events.

1.2 Goals

Since it is relatively new to transfer ideas from image processing to 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.

The thesis deals with noise that can be represented as an isolated but intense event, e.g. a cough. Note that this is a restriction to a certain group of noise.

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

The software is realized by a C program that is built from scratch except for the code used for the fast Fourier transform (FFT) [13] which was provided.

As a first step in the analysis, the program 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 program runs correctly on these signals, 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 (section 2.1) as well as how possible similarity measurements are computed (section 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 investigates different experiments. By first explaining two parameter settings (section 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. Furthermore, different methods to repair noisy data are investigated, starting in section 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 Appendix A where the file structure is given in A.1 and a short overview of the meaning of this data is given in section A.2. Appendix B gives an overview of all the files used in this thesis including their denoised versions.

Chapter 2

Similarity Measurement

2.1 Prerequisites

Before the similarity measurement can be described, some prerequisites have to be introduced. These prerequisites are easy to understand since the ideas from image processing are only slightly modified. Here it becomes evident that some parallels exist between image processing and audio processing.

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 the latter 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.

A signal can be subdivided into different frames which usually overlap each other (see also section 2.1.3). The short-time Fourier transform takes one of these frames, multiplies it with a window function w and repeats this step with the next frame until the end of the signal. This means that a window of a certain width is moved over the whole signal where each window corresponds to one frame. The ”classical” Fourier transform is used to calculate the Fourier coefficients for each window.

Often used in music analysis, it can assist studio engineers by depicting the frequencies of a song with a spectrogram which has already been touched in section 1.1.

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.

A lot of possibilities exist for the window function. One could use a box function [2] which would result in a too hard windowing 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.

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 that it is smooth and that every sample in a certain time period should approximately make the same contribution. A Gaussian would fulfill this property where here the problem arises that it has to be cut at the window boundaries – due to its infinite extend – which would result in some kind of approximation being dependent on the point where the cutting is performed.

So one has to come up with another window which deals a bit more clever with that boundary problem. An example for such a window is the Hamming window (cf. figure 2.1) 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 $0 \leq n \leq N-1$.

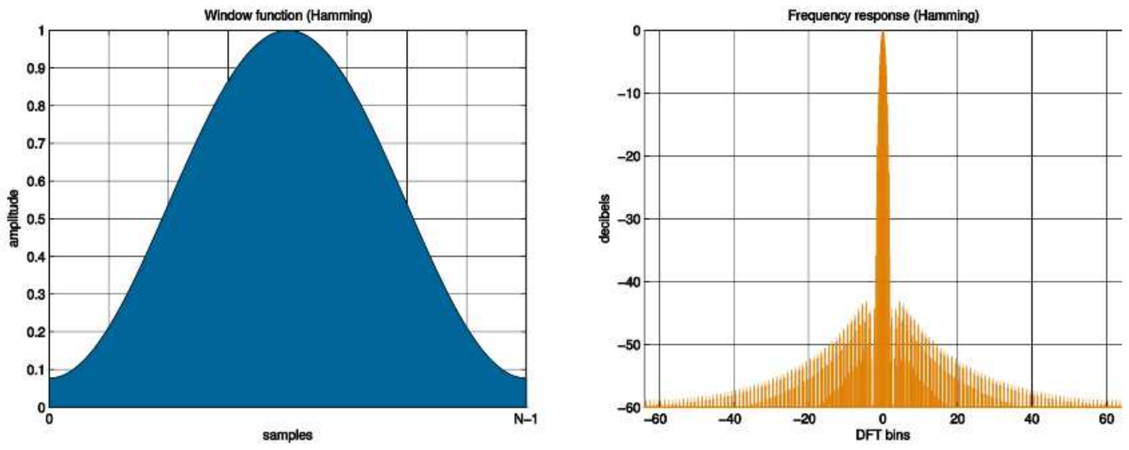

Figure 2.1: Hamming window. **Left:** Window function in the sample domain. **Right:** Frequency response. **Source:** Wikipedia [2].

The frequency response depicts how a window deals with a sharp frequency. A narrow peak in the middle and nothing else is optimal since it would mean that only the concrete frequency one is interested in and nothing else is considered. In the case of the Hamming window the actual frequency is represented quite well. It only smears to the neighbor frequencies in a quite moderate way.

Another possibility is the Hann window (cf. figure 2.2). As one can see in the frequency response, it doesn't deal that nicely with the frequency one is interested in: the peak in the middle is a bit broader. So it seems that the Hamming window is a good choice, because it has a relatively narrow peak in the middle while not smearing the neighbor frequencies too much, meaning the neighbor frequencies do not contribute that much to the sharp one like it happens with the Hann window.

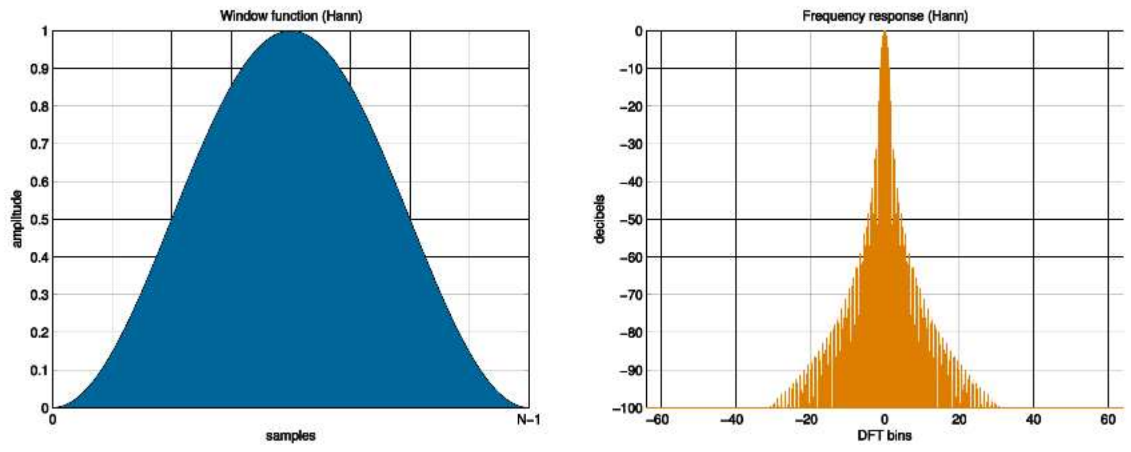

Figure 2.2: Hann window. **Left:** Window function in the sample domain. **Right:** Frequency response. **Source:** Wikipedia [2].

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).

Imagine to use the values 256 for the step size and 1024 for the window width. This would 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 results that are similar to the reference frame. This becomes evident when requiring that every acoustic event is sufficiently represented in at least one window to guarantee that the event will appear in its frequency decomposition in the short-time Fourier transform.

For a WAV file at 44100 Hz, a step size of 256 would mean to go through the recording 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 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 program 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 program 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 frames 82.9 until 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.

2.2 Similarity Measurement

To find a frame that is similar to a given reference frame, several possibilities for a similarity measurement exist, 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 [9]. The power spectrum is explained in a bit more detailed fashion in section 4.2.

Minimizing the Euclidean distance would mean to compute

$$\left\{ i \;\middle|\; \min_i \left\| |\hat{f}_i|^2 - |\hat{f}_{ref}|^2 \right\| \right\}$$

where $|\hat{f}_{ref}|^2$ denotes the power spectrum of the reference frame and $|\hat{f}_i|^2$ 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

$$\left\{ i \;\middle|\; \max_i \frac{\left\langle |\hat{f}_i|^2, |\hat{f}_{ref}|^2 \right\rangle}{\left\| |\hat{f}_i|^2 \right\| \cdot \left\| |\hat{f}_{ref}|^2 \right\|} \right\}$$

where the scalar product is normalized by the standard vector norm of the two vectors $|\hat{f}_i|^2$ and $|\hat{f}_{ref}|^2$. 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. The short-time Fourier transform was introduced which adds a spatial/temporal component to the ”classical” Fourier transform. A window which is multiplied to a part of the signal – where the Hamming window was introduced as a suitable windowing function – is moved over the entire signal while performing the Fourier transform for each window to calculate its Fourier coefficients. The concepts of step size and window width were introduced and the Hamming window and the short-time Fourier transform were deepened in this context. Using something else than a hard windowing like a box function was pointed out as important since problems would arise at the boundaries due to an abrupt change of the amplitude. The prerequisites were concluded with a 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. 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 a method for repairing damaged pictures or removing unnecessary elements from pictures. The key idea of non-local inpainting 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. This could be the Euclidean distance as introduced in section 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 a specified neighborhood. The patch to be copied doesn't necessarily have to be the best match.

The idea of non-local inpainting was used for example in [10] where image texture is synthesized.

3.3 NL-Means

Non-local means (or short NL-means) – proposed by [8] – is as simple as powerful in image processing. The 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 minimization of the Euclidean distance.

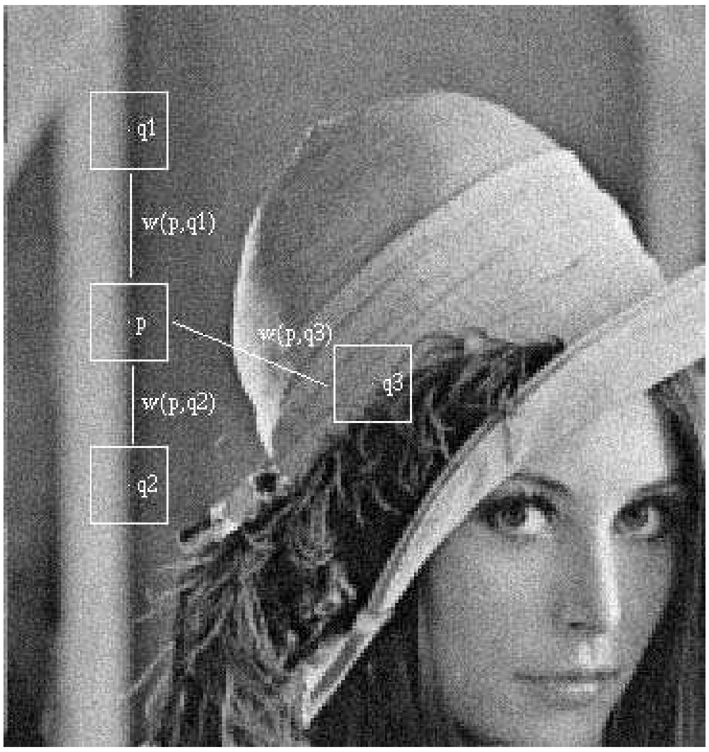

Figure 3.1: Scheme of NL-means strategy. Similar pixel neighborhoods give a large weight, $w(p,q_1)$ and $w(p,q_2)$, while much different neighborhoods give a small weight, $w(p,q_3)$. **Authors:** A. Buades, B. Coll and J.-M. Morel [8].

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 shows how the similarity measurement is refined to match the needs to compare different frames followed by an algorithm for non-local inpainting, the modification to realize an averaging on a sample basis and a NL-means like approach. In section 4.7, it is shortly discussed how stereo files are processed. Nevertheless, this thesis concentrates on mono files.

4.2 Similarity Measurement

In chapter 2, 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 (N: window width).
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

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

   where $\mathrm{Re}(\hat{f})$ denotes the real part and $\mathrm{Im}(\hat{f})$ the imaginary part of the Fourier transformed signal f.

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. The number of frames classified as similar is at most 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.

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 provide the highest flexibility possible in this context.

First of all, the noisy part of the data is excluded which is obvious since noise cannot be trusted. This is especially important if one has a larger degradation of the data because the accuracy is higher then. Of course, the user can also decide to not exclude anything, but note that in the case of this thesis, noise is an isolated but intense event. The algorithm doesn't perform any automatic noise detection, the noisy locations are known in advance.

Secondly not only the reference frame itself but a neighborhood is considered, so a 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 an unnatural listening experience when replacing one "same" tone by another since it could sound different due to its surrounding (cf. figure 4.1). In addition to that, one has to make sure that the algorithm doesn't find similar frames that are very close to the reference frame when performing the search on a neighborhood. To overcome this problem, a minimal distance for the search is introduced. Let's denote this minimal distance with d and the reference frame with ref. Then only the frames that do not lie in the interval $[ref - d, ref + d]$ are considered for the similarity search.

A third refinement is to require a certain distance between the results in the similarity list since it is expected that frames cluster around similar matches. This distance is currently fixed to 10 which gives good results.

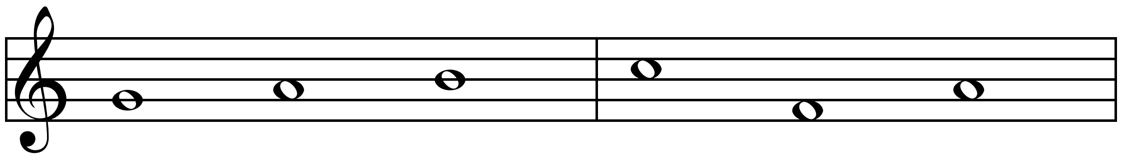

Figure 4.1: Example for two different occurrences of the tone A. **Left:** G – A – B (precisely: g' – a' – b'). **Right:** C – F – A (precisely: c" – f' – a').

4.3 Non-Local Inpainting

Non-local inpainting for audio signals is inspired by the ideas of [10]. After having found frames that are similar to the reference frame, the program replaces the neighborhood – specified by an user input – around the reference frame including the reference frame itself by the same neighborhood around the selected similar frame, also including this one. No mirroring at the boundaries is used. Assuming mirroring boundary conditions[1] for an image processing problem is fine, but in the case of audio signals, this is not a good choice. Instead, zero-padding is used here, because when mirroring at the boundaries, the music would be analyzed in a ”reverse order” when passing the boundary.

Suppose, the user specified a neighborhood of k frames. Then in total $2k+1$ frames are replaced: k frames left to the reference frame, the reference frame itself and k frames right to the reference frame. To do so, the start and end point of the reference frame and the similar frame in the WAV data are calculated via the formulae

$$\begin{aligned} start\_ref &= (ref - n) \times step\ size \\ end\_ref &= (ref + n) \times step\ size \\ start\_frame &= (frame - n) \times step\ size \\ end\_frame &= (frame + n) \times step\ size \end{aligned}$$

where ref is the reference frame, $frame$ the similar frame and n the neighborhood. An example shows how this works. Imagine to have a $step\ size = 256$, 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 WAV 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 4.4, 4.5 and 4.6 since here only the data is modified and not the method how to repair noisy parts.

4.4 Averaging on a Sample Basis

With non-local inpainting as explained in 4.3, only a selected frame is filled in. Now, not only a selected frame is used, but 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 including a certain neighborhood.

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.

[1]for PDE-based methods these are Neumann boundary conditions

4.5 NL-Means

In principle, NL-means is a further development of averaging on a sample basis introduced in section 4.4. 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 where the maximization of the scalar product is used for this thesis, the weighting function has to be slightly changed. The program 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.

The threshold parameter λ controls how fast the exponential function decreases. This is an important parameter since it determines in which way the similar frames are considered for the NL-means approach. Choosing a small value for λ means to consider more frames, choosing it large means to consider less frames (since the exponential function decreases faster for large λ). Another aspect that should be considered is the choice of the parameters step size and window width (cf. 2.1.3). Depending on them it could happen that λ has to be slightly changed.

4.6 NL-Means on Continuous Noisy Signals

At the end of the development of this thesis, a preprint by Arthur Szlam appeared [14] which is also concerned with using the NL-means algorithm on audio data. In his work, he considers the whole signal to contain moderate Gaussian white noise where this thesis deals with isolated noisy events. Nevertheless, it is possible to adapt to this setting since both approaches are similar.

The only modification is to perform the similarity search for each frame using a small neighborhood without excluding frames and iteratively go over the whole signal frame by frame. For each similarity list, the NL-means approach developed during this thesis is applied for the denoising step where a special parameter setting is employed to do so. This parameter setting is changed in two parameters: first of all, the algorithm should not consider any neighborhood for the repairing, meaning to exchange only the currently observed frame, and secondly not excluding any frames when doing the NL-means averaging. The latter is due to the fact that excluding (noisy) frames only makes sense if the noise is an isolated event in the audio file which is not the case here.

In all other parameters the algorithm behaves as explained in section 4.5.

4.7 Stereo Files

In a stereo file, both channels cannot be treated separately for obvious reasons. To simplify this problem, the file is processed internally 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.

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 and the runtime behavior evolve under two different configurations and shows methods that are candidates to repair defects in music.

Here is a list of files that are used for the experiments where their corresponding degradation with an approximate time location in the audio file is stated in brackets.

- cyftlt-mono-short_sine.wav (sine tone, time: 1:53.14 – 1:53.23),
- cyftlt-mono-long_sine.wav (sine tone, time: 1:20.01 – 1:25.00),
- cyftlt-long_sine.wav (sine tone, time: 1:20.01– 1:25.00),
  (corresponding stereo file to cyftlt-mono-long_sine.wav)
- piano1-mono-white.wav (white noise, time: 0:02.01 – 0:03.02),
- piano3-mono-sine.wav (sine tone, time: 0:03.75 – 0:04.70),
- piano3-mono-white.wav (white noise, time: 0:04.40 – 0:04.45)

The files listed above were all synthetically created by using the MIDI sequencer software Propellerheads Reason 4 [3] and edited afterwards with Audacity [1] to insert the noise.

Also, two audio files from the CD ”Elton John: A Piano Tribute” by Boko Suzuki are used as real-world examples 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 4.2. In current experiments, only piano music was used.

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%. It is referred to this configuration by using `configuration1`. In the second configuration, the step size was set to 2048 and the window width to 4096, resulting in an overlap of 50%. It is referred to this configuration by using `configuration2`.

5.2.1 Memory Requirements

The first configuration with a step size of 256 and a window width of 1024 works quite well. Unfortunately it needs for a 21 MB wave file in mono[1] – 4:04 minutes of piano music – about 300 MB of main memory which is pretty huge in comparison to the effective 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.

With a straightforward implementation without doing any optimization, instead of the aforementioned 300 MB even about 530 MB would be needed, but two tricks are used to save memory. The first trick is not to use a small array for the real part, a small array for the imaginary part and a small array for the spectrum of the Fourier transform for each window but to store all the data in a large array for all the real parts, a large array for all the imaginary parts and a large array for all the spectra. Using pointers helps to state which values belong to which window. This means that for each window three pointers are used in total: one for the real part, one for the imaginary part and one for the spectrum. All the data is arranged in order which means that all the values between two pointers – including the first one of the two – belong to the same window. It is expected that memory is saved in this way since the compiler always ”rounds” to the next power of 2 when allocating memory. Allocating three arrays containing 513 slots of floats each results in allocating three times 1024 slots of floats. Allocating one big array would result in 2048 slots of floats[2] instead of 3072 slots. This becomes even more important if this allocation step is done more than 1,000 or 10,000 times.

The 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. For this thesis, the recomputation will not be necessary.

The resulting memory requirements are in fact expected. To verify this, an exemplary calculation is made for `configuration1` where cyftlt-mono-long_sine.wav is used as input

[1] cyftlt-mono-long_sine.wav was used in this example.

[2] $3 \times 513 = 1539 \rightsquigarrow 2048$

file. The effective data size of this file is 21,520,800 bytes and it has 16 bits per sample.

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

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 manage-

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

Description	Size		[bytes]
WAV data array	$datasize \times$ sizeof(float)		43041600
real part of FT data	$arraysize \times ft\_size \times$ sizeof(float)	+	86251716
imaginary part of FT data	$arraysize \times ft\_size \times$ sizeof(float)	+	86251716
spectra of FT data	$arraysize \times ft\_size \times$ sizeof(float)	+	86251716
array with similar frames	$how\_many\_similar\_frames \times$ sizeof(int)	+	40
array with distances	$how\_many\_similar\_frames \times$ sizeof(float)	+	40
		=	301796828
		≈	287.82 MB

ment which increases the calculated requirement (287.82 MB) to a real memory requirement of 288.66 MB. To conclude this example it has to be mentioned that a few helper arrays are created for computational purposes and destroyed immediately after the computation. These are in the range of about 80 MB[3] which increases the whole main memory requirement to a worst case maximum of 400 MB (again including space used by the operating system for the memory management) during program run for this file.

5.2.2 Runtime

The runtime of the program clearly depends on several factors. The most important one is the dependency on the problem itself, i.e. the size of the file being processed, the size of the neighborhood, the extent of the noise, whether a certain part is excluded and so on. The measurements here should give a flavor of how fast the program approximately is. All files

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

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

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

step size / window width	256 / 1024	2048 / 4096
Reference frame	14298 (1:23)	1787 (1:23)
Minimal search 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 5.3: Processing of piano3-mono-sine.wav, duration: 0:08 minutes (size 700 KB).

step size / window width	256 / 1024	2048 / 4096
Reference frame	861 (0:05)	107 (0:05)
Minimal search 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

Furthermore the program should give the 10 best results and the best result was chosen for the replacement using non-local inpainting. Of course, only the computation time is measured.

Note, that the coarser scale is about four times faster while giving the same listening experience in the result.

5.3 Non-Local Inpainting

If the similarity measurement provides a good result, non-local inpainting performs well from a musical point of view. In some cases, non-local inpainting can even reconstruct the noisy part such that one cannot tell a big difference to the original, non-noisy sound file except for a small click at the cutting points that could be avoided by using a cross-fade which can be understood as a weighting function. A cross-fade is the application of two functions to a signal where one function linearly goes from 1 to 0 (fade-out) and the other linearly from 0 to 1 (fade-in). Having this, a smooth transition between two signal parts is created whose result makes the discontinuity at a cutting point in most cases imperceptible.

Figures 5.1 and 5.4 show exemplarily how a noisy signal looks like and how it might look after performing non-local inpainting.

As already explained in section 4.3, a similar patch is searched and inserted using a specified neighborhood that is also copied.

In the experiments it turned out that the inpainting approach has a disadvantage in some settings which is depicted for piano3-mono.wav in figure 5.4: it either cannot completely repair the defect or starts repairing at the beginning of the noise, resulting in a shift of the inserted data to the left. Figure 5.2 shows a schematic drawing of how the non-local inpainting approach works if a part of the noise is untouched. As one can see, some information cannot be filled in due to the fact that the patch used for the inpainting step overlaps to a region where no data is present. Figure 5.3 shows how the non-local inpainting works if the repairing starts at the beginning of the noise. The type of repairing depends on the selection of the reference frame and the neighborhood used for the replacement.

Considering cyftlt-mono-long_sine.wav, no problems in repairing the defect appeared. Here it is important to choose a neighborhood which is on the one hand large enough to find a frame that is similar and on the other hand large enough to repair the whole defect. Note, it must always be satisfied that the replacement neighborhood is not greater than the search neighborhood. The parameters that worked best in the experiments are stated in table 5.2. These parameters also work for the file cyftlt-long_sine.wav which is in stereo.

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 five seconds of data is completely destroyed and should be reconstructed which is a pretty tough task. In the case of piano3-mono-sine.wav the difficulty clearly lies in the fact that the best matching patch for the defect is exactly at the beginning of the file. From this viewpoint, non-local inpainting works great with the right parameter setting. Unfortunately, finding the right parameter setting requires a lot of ”tweaking the knobs” for some files at the moment since the program currently asks a lot of parameters that give various possibilities to experiment. This has also been encountered with the file piano1-mono-white.wav.

In the following, the performance of the algorithm on more realistic problems is investigated. These experiments stand in contrast to the introduction of distortions of artificial character, like long defects. piano3-mono-white.wav is an example where only a short distortion by white noise of size 0.05 seconds is present. To repair this file is not difficult, but in `configuration1` some clicking noise of low intensity is introduced at the location where the inpainting is performed. In `configuration2`, one can also hear clicking noise, but it is very short in its extent and even lower in its intensity in comparison to `configuration1`.

When having short noisy parts it is also worth to investigate the situation where no frames are excluded. In most cases, the results are comparable in quality to the ones created with an exclusion of the noisy part. Among others, not excluding anything was tested with the two files candle_in_the_wind-mono-silence.wav and your_song-mono-white.wav. For the file candle_in_the_wind-mono-silence.wav, the results are better when excluding the noisy frames where for the test file your_song-mono-white.wav it is interestingly the other way round. In the latter case, the version where the noise is not excluded sounds more ”natural”. This

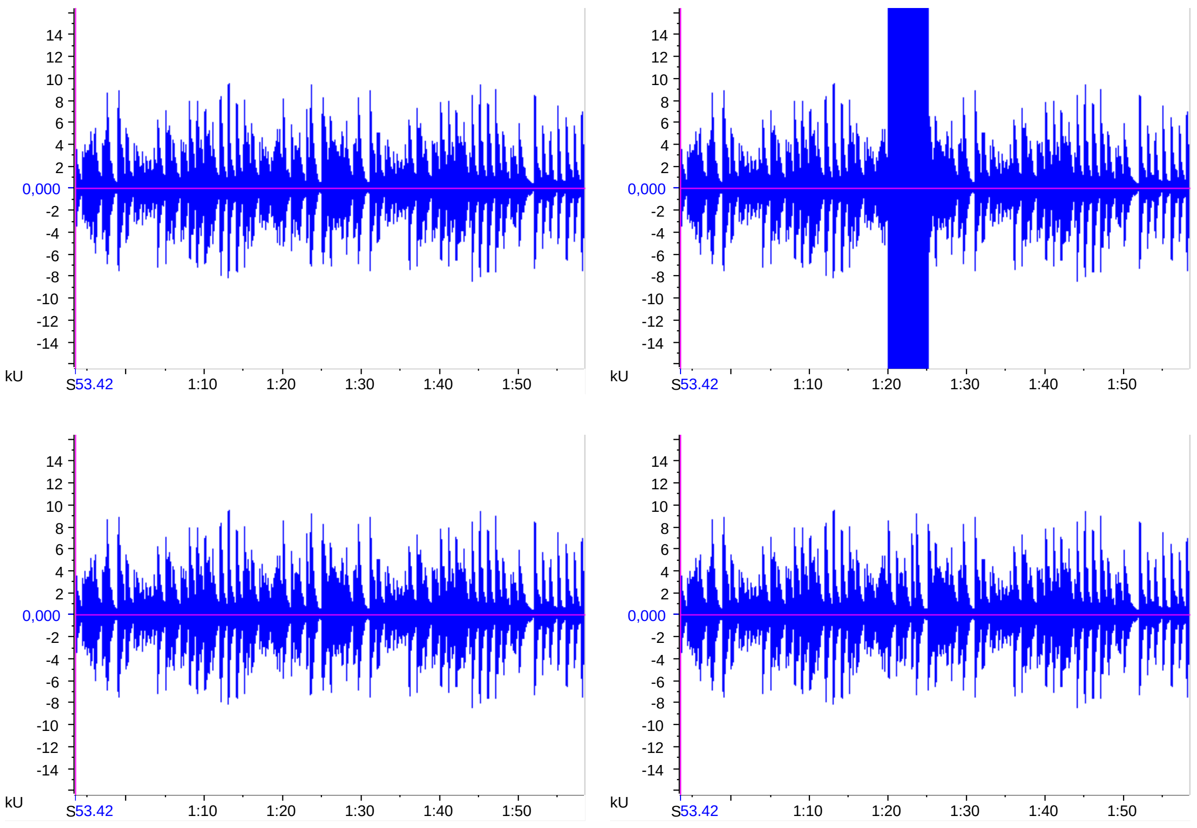

Figure 5.1: Correction with non-local inpainting for cyftlt-mono.wav (4:04 minutes). **Top left:** Original file. **Top right:** Original degraded with a sine tone of 440 Hz (cyftlt-mono-long_sine.wav). **Bottom left:** Corrected version with a step size of 256 and a window width of 1024. **Bottom right:** Corrected version with a step size of 2048 and a window width of 4096. **Visualization:** using RavenLite 1.0 [4].

is due to the fact that the noise is in the middle of a part that only contains a piano chord which decreases in volume. The algorithm finds – if no frames are excluded – similar frames located around the noisy part, so low structures are reproduced. Nevertheless, this repairing is accompanied by low clicking noise.

5.4 Averaging on a Sample Basis

Since only different data is used for the repairing step in this approach, it also suffers from the shifting problem that was exemplarily described for piano3-mono-sine.wav in section 5.3.

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 the algorithm is relatively good in reconstructing something harmonically close to the original while on the other hand providing an unnatural listening experience. For the test file

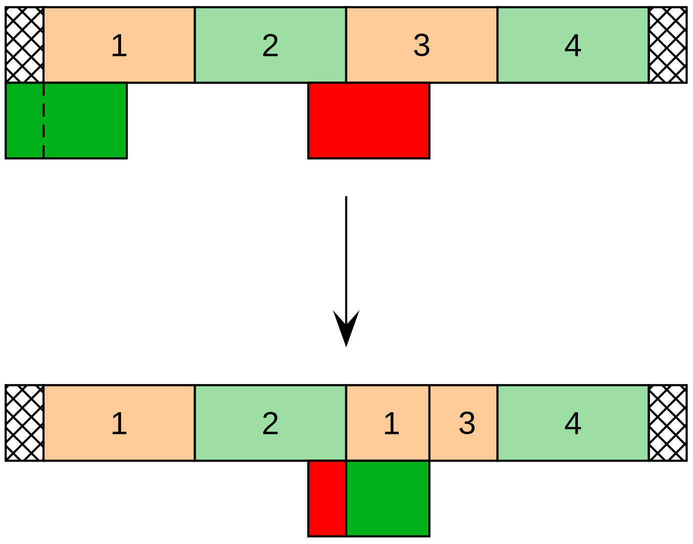

Figure 5.2: Schematic of how the non-local inpainting approach works in the case of piano3-mono-sine.wav. The crossed areas denote the beginning and the end of the music file. The chords are denoted by numbers, where same colors indicate same chords. The red box denotes the noise where the green one denotes the part of music that could be used for the inpainting step. **Top:** Noisy music file with marked candidate for inpainting. One can see that some information cannot be filled in due to the fact that the green box overlaps to a region where no data is present. **Bottom:** Same file after non-local inpainting when not repairing the whole defect.

cyftlt-mono-long_sine.wav and the corresponding stereo version cyftlt-long_sine.wav the task of averaging similar frames results in something that can be called chaotic when expecting something musical: different pieces of music overlap that do not really fit together. The chaotic behavior can sometimes be compared to having a few musicians playing different parts of a piece of music at the same time. For the case that three or less similar frames are considered, the result is repairing the noisy part from a musical viewpoint in a better way whereas the problem exists that some frames are a little bit shifted. The listening experience can be compared to a chorus effect[4] in the mono case for `configuration1` and to the impression that a few melodies run in parallel for `configuration2` (also in mono) as well as for `configuration1` in the stereo case. In `configuration2`, the repairing of the stereo file gives a quite ”normal” playback.

In `configuration1`, the files candle_in_the_wind-mono-silence.wav and your_song-mono-white.wav are restored harmonically pretty well while still giving the ”chaotic behavior”

[4]chorus: the perception of similar sounds from multiple sources as a single, richer sound.

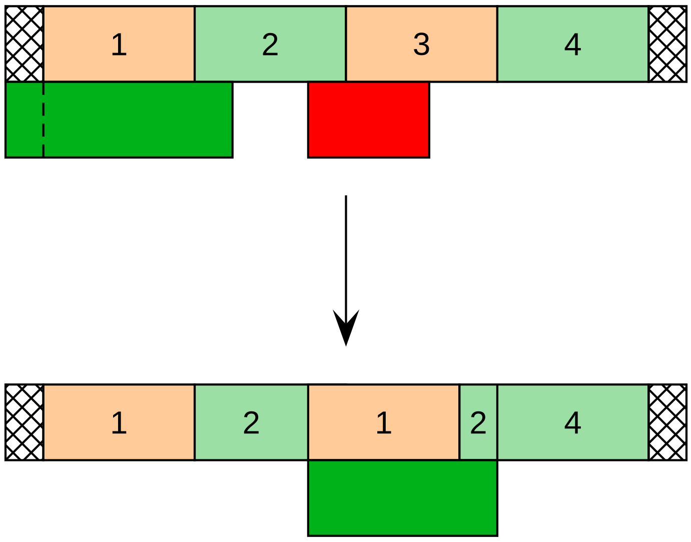

Figure 5.3: Schematic of how the non-local inpainting approach works in the case of piano3-mono-sine.wav. The crossed areas denote the beginning and the end of the music file. The chords are denoted by numbers, where same colors indicate same chords. The red box denotes the noise where the green one denotes the part of music that could be used for the inpainting step. **Top:** Noisy music file with marked candidate for inpainting. One can see that some information cannot be filled in due to the fact that the green box overlaps to a region where no data is present. **Bottom:** Same file after non-local inpainting when starting to repair the defect at the beginning of the noise.

as mentioned above. Interestingly, it doesn't make a big difference whether the noisy frames are excluded or not: one cannot judge which type of repairing is the better one. Similar results were achieved when using cyftlt-mono-short_sine.wav or piano3-mono-white.wav as test files.

In general, *`configuration2`* gives the same listening experience as already described for *`configuration1`*, except for piano3-mono-sine.wav where *`configuration2`* gives better results than *`configuration1`*.

From a musical point of view, the listening experience is not like one would expect a song to be. One problem is that the rhythmic structures of the different frames don't match, another one that parts with different melodies are mixed. Since the program doesn't have any previous knowledge of musical structures, it cannot fit the different rhythmic and melodic structures. From a computer science point of view the algorithm works better than expected since it finds structures that fit in terms of their tuning (harmonically). Not knowing the song might result in acceptance of the repairing, except for the fact that one might again perceive a small click at the point where the new data is connected to the old data which can be avoided as explained

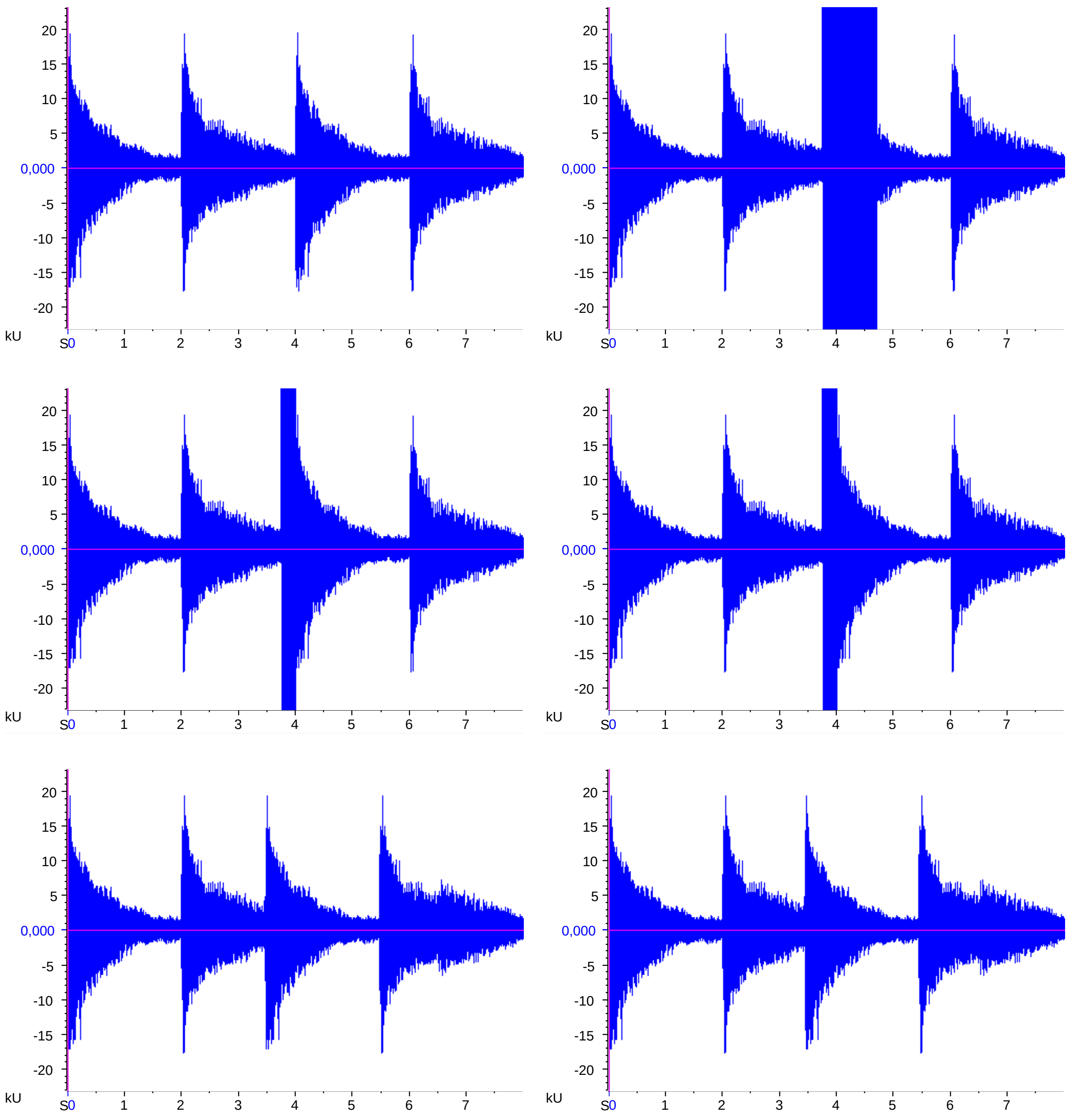

Figure 5.4: Correction with non-local inpainting for piano3-mono.wav (0:08 minutes). **Top left:** Original file. **Top right:** Original degraded with a sine tone of 330 Hz (piano3-mono-sine.wav). **Middle left:** Corrected version with a step size of 256, a window width of 1024 and a replacement neighborhood of 175. **Middle right:** Corrected version with a step size of 2048, a window width of 4096 and a replacement neighborhood of 22. **Bottom left:** Corrected version with a step size of 256, a window width of 1024 and a replacement neighborhood of 263. **Bottom right:** Corrected version with a step size of 2048, a window width of 4096 and a replacement neighborhood of 33.

in section 5.3.

Also, it was encountered that the averaged patch turns more and more quiet the more similar frames are specified to consider. Averaging the five best results is louder than averaging the 50 best results if all other parameters stay unaltered. This happens due to the cancellation of waveforms with different phases.

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 section 5.4. Also, the shifting problem investigated in section 5.3 still exists. This is again due to the fact that just the data used for the repairing step is different.

Experiments have shown that the threshold parameter λ significantly contributes to the final result by also partially overcoming the problems like the ”chaotic behavior” as mentioned in section 5.4. Choosing a higher threshold results in a better – for an image one would say ”sharper” – denoising. Of course, choosing λ too large almost results in the non-local inpainting approach presented in section 4.3 since the exponential function weights the best match always with 1[5] while all other matches receive weights close to 0 (because of the large λ).

The volume problem as presented at the end of section 5.4 doesn't appear with this approach. This is due to the fact that the exponential function can be scaled using λ and this strongly influences the weighting of the different samples. Choosing a small decrease[6] of the function – i.e. a very tolerant threshold parameter –, the first samples of the list get a high weight, resulting in an almost unweighted addition of all the suggested frames. If a lot of frames with distances close to each other cluster in combination with a small decrease of the exponential function, this can result in over- and undershoots, appearing as oversteer in the sound file.

Choosing a higher decrease of the exponential function (λ should be larger) clearly performs better than the averaging on a sample basis not only in terms of the tuning of the different frames but also in terms of their ”fitting”. This is obvious since choosing a more radical weighting function almost doesn't consider frames that are not on the first positions in the similarity list, i.e. frames that do not have a high similarity.

Having a sound file that contains a lot of loud structures, λ should be chosen higher, because adding high values (loud structures) and weighting them with an exponential function using a small λ results in over- and undershoots quite fast. The opposite is also true: having a sound file that has not much loud structures, λ should be chosen smaller. Furthermore it always has to be considered which configuration of the step size and window width is used. Depending on that, differences in choosing λ can appear, too. This point is also discussed in the following experiments.

[5] $\exp(-\lambda \cdot \frac{best-cur}{best}) = 1$ for $best = cur$

[6] This exponential function is decreasing because of the negative sign in the exponent of the exponential function (cf. section 4.5).

First, let's investigate `configuration1`. The files cyftlt-mono-short_sine.wav, piano1-mono-white.wav and piano3-mono-white.wav are denoised quite well. Nevertheless, one can hear an abrupt ”discontinuity” in the melody. Besides, it doesn't make a big difference whether one excludes the noisy part or not since one cannot judge which result is the better one – they are both convincing.

For candle_in_the_wind-mono-silence.wav and your_song-mono-white.wav one achieves also good denoising results. With these two files it is important to choose a large λ since both files are very loud and dense in comparison to other files tested during the development of this thesis. For candle_in_the_wind-mono-silence.wav, $\lambda \geq 80$ is a good choice and for your_song-mono-white.wav, $\lambda \geq 100$ is a good point to start with. Also, for these two files it is possible to denoise without excluding noisy frames. Denoising your_song-mono-white.wav gives again better results when not excluding the noisy frames (cf. results in section 5.3).

cyftlt-mono-long_sine.wav can be denoised with similar parameters as stated in table 5.2 and $\lambda \geq 30$ very well. For the stereo file cyftlt-long_sine.wav one can use the same parameters but has to choose $\lambda \geq 60$ to achieve the same result as for the mono case. This might be due to the basic averaging approach done for stereo files as introduced in section 4.7.

When denoising piano3-mono-sine.wav with the same parameters as given in table 5.3 and $\lambda \geq 30$, one can hardly tell a difference between the NL-means approach and the non-local inpainting.

Let's look at `configuration2`. Interestingly, the files piano1-mono-white.wav, piano3-mono-white.wav and cyftlt-mono-short_sine.wav give better denoising results in comparison to `configuration1` when using a smaller λ. Choosing λ around 10 already suffices to get a result that better fits the volume of the patch inserted in comparison to the whole piece of music. When expecting the repaired files to sound convenient, one has to choose $\lambda \geq 40$ for cyftlt-mono-short_sine.wav.

With the two files candle_in_the_wind-mono-silence.wav and your_song-mono-white.wav, one can achieve a better denoising for a small λ, too. candle_in_the_wind-mono-silence.wav can already be denoised acceptably using $\lambda = 10$ where a λ around 25 is needed to repair the defect in your_song-mono-white.wav quite well. Nevertheless, one has the feeling that something is used for the denoising that really doesn't fit here in terms of the melodic structure. Better results can be achieved when choosing $\lambda \geq 25$ for candle_in_the_wind-mono-silence.wav and $\lambda \geq 80$ for your_song-mono-white.wav.

As expected, it is also possible with this approach to not exclude any noisy frame in case of the test files with a short degradation. Again, it cannot be judged which denoising variant is the better one.

For piano3-mono-sine.wav, the parameters in table 5.3 also work here where λ can be chosen relatively small: something around 10 is a good point to start with.

The file cyftlt-mono-long_sine.wav can be denoised by the same parameters as stated in table 5.2. For $\lambda = 20$, one can already get a good result and for $\lambda \geq 30$ the result for the NL-means approach is that good that it almost cannot be decided whether the non-local inpainting or the NL-means approach is used in the denoising step. Surprisingly, $\lambda = 30$ is also adequate for the stereo case (cyftlt-long_sine.wav) in this configuration.

5.6 NL-Means on Continuous Noisy Signals

The preprint by Szlam deals with files that are completely degraded by moderate Gaussian noise. After running the algorithm on this kind of data, one can perceive that the Gaussian noise is reduced, but also a lot of clicking noise is introduced. This might be due to the fact that the algorithm is not optimized for such problems. Interestingly, the clicking noise is not always reduced when iterating a few times. It can even be the case that the more iterations are done, the more clicking noise is introduced. This can be explained by the following situation. The algorithm runs over the whole signal for the first time. After that, a few appearances of clicking noise can be noticed in the result. If the algorithm is iterated, it can happen that such a clicking noise is in the (small) search neighborhood of the currently observed frame, meaning that a frame is searched that contains noise. As a consequence, this noise contributes to the averaging used to repair the current frame, probably resulting in more noise, depending on how the noisy frames are weighted.

5.7 Summary

This chapter investigated the memory requirements and the runtime of the developed algorithm under two different parameter settings. The first parameter setting introduced an 75% overlap of the data where the other one a 50% overlap of the data. The memory requirements were exactly expected in this way which was shown in a simple calculation. Concerning the runtime, it turned out that the parameter setting with the 50% overlap was four times faster than the other one by giving approximately the same results.

Furthermore, three methods to repair noisy data were investigated in detail: non-local inpainting, averaging on a sample basis and NL-means. It turned out that all the methods strongly depend on the proposals made by the similarity measurement. Non-local inpainting works well, if the proposal was good. Sometimes it even can reconstruct the noisy part in such a way that the repaired signal comes very close to the original (non-noisy) file.

With the approach to average on a sample basis one can get harmonically good results. The problem from a musical point of view is that the approach is pretty sensitive as far as the number of similar frames to search for is concerned.

The NL-means approach works good as long as the threshold parameter λ is chosen in such a way that the involved exponential function decreases fast enough.

Also, an approach similar to the one by Szlam [14] is investigated in the experiments. Here one can perceive sudden appearances of clicking noise. A more detailed investigation of this phenomenon could not be carried out within the time frame of this thesis.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

As presented in this thesis, image processing ideas can be used fruitfully to analyze music. It was presented how the tools like the Fourier transform have to be changed to fit the needs of audio analysis. By windowing and going forward in time, having different windows overlap each other (sliding window) to get more than one contribution to the currently observed part, one can subdivide the whole signal into separate windows and compute the power spectrum for each window. Of course, computing the power spectrum implies to perform a Fourier transform using the data of the corresponding window as input.

Having the basic ingredients like the short-time Fourier transform as the modification of the ”classical” Fourier transform and the Hamming window as a windowing function, one can interpret the power spectrum of each window as a one-dimensional vector which can be used for the similarity measurement. In a first step, the scalar product of two vectors – the power spectrum of the reference frame and the power spectrum of the currently observed frame – was normalized using the vector norm of both vectors. To find out which frames are similar, the distances between the scalar product of the reference frame and the currently observed frame were computed where two frames are similar if their distance is maximal.

This work showed that it is possible to use methods known from image processing and modify them to do audio processing. It turned out that non-local inpainting works well if the proposed similar frame is good, averaging on a sample basis repairs the defect at least in a harmonic sense and that the NL-means approach is controlled by a threshold parameter λ which significantly contributes to the result: if λ is chosen well the results are good.

6.2 Future Work

So far, only the maximization of the scalar product is used as a similarity measurement. This approach is sensitive to loud structures, in other words it prefers them. If a melody appears at a certain location very muted and at another location pretty loud, the latter one would be preferred. Here, some efforts can be made to make the similarity measurement more robust against the volume of the different patches. Maybe also other similarity measurements exist that can be used to find similar frames. The use of a similarity matrix as proposed by [11] and used for example in [9] might be an intuitive possibility.

Also, repairing multiple degradations at the same time – i.e. in one run – could be a possibility to make the denoising process more comfortable. A basic approach would be to use an iteration during the program run. This might not be the most efficient way, but would work.

The short-time Fourier transform has some drawbacks, e.g. that the frequency resolution is not sufficient to separate musical notes of low frequency. A possible remedy is to use several short-time Fourier transforms that work at different sampling rates with different window sizes to increase the frequency resolution for lower notes, see e.g. [12].

It would also be possible to do an alternative approach by not using the short-time Fourier transform, but to use a wavelet-based approach: it gives a good time resolution for high-frequency events and a good frequency resolution for low-frequency events.

Another possible work for the future is to make the software more user-friendly. Maybe the number of parameters can be reduced. This might be possible by establishing a dependency between the neighborhood which is considered for the search and the neighborhood which is effectively replaced then.

Arthur Szlam proposes in [14] to run the NL-means algorithm on speech where Gaussian noise is used for the degradation of the whole signal. Since first attempts to include the idea by [14] didn't work well so far, it could be worth to invest more effort into changing the algorithms to perform better on such a setting. This was not possible within the time frame of this thesis.

Appendix A

Wave Files

A.1 Wave File Format

The wave file format [6] 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 [5] 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:

Table A.1: The WAV file format. **Adapted from:** Wikipedia [6].

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	Num 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	Num of 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			

In figure A.1, an example for a possible data arrangement of a WAV file is depicted.

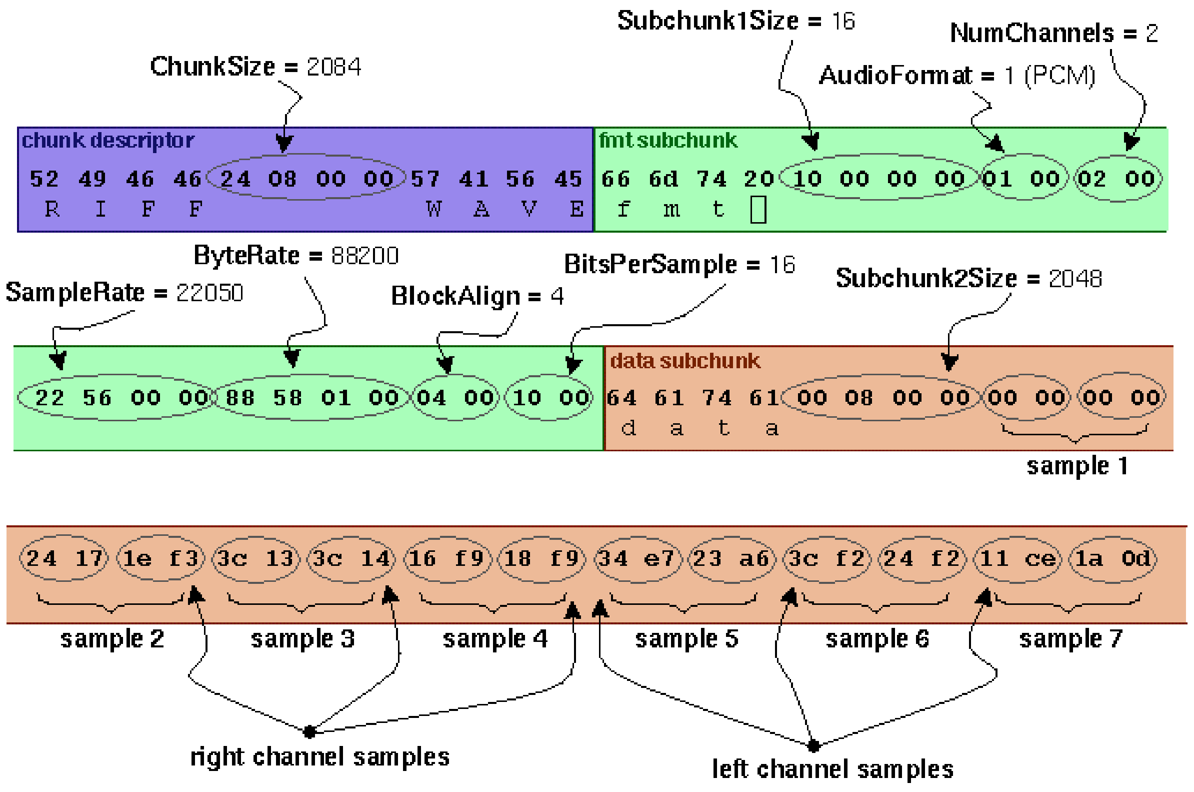

Figure A.1: First 72 bytes of an example WAV file. **Source:** `http://ccrma.stanford.edu/courses/422/projects/WaveFormat/`.

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 two channel samples are combined to build a sample and N samples are combined to build one frame.

In the audio sector, the ”channel samples” in figure A.3 are often 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.

For a more detailed description of the waveform representation, see section 2.1.2 in [12].

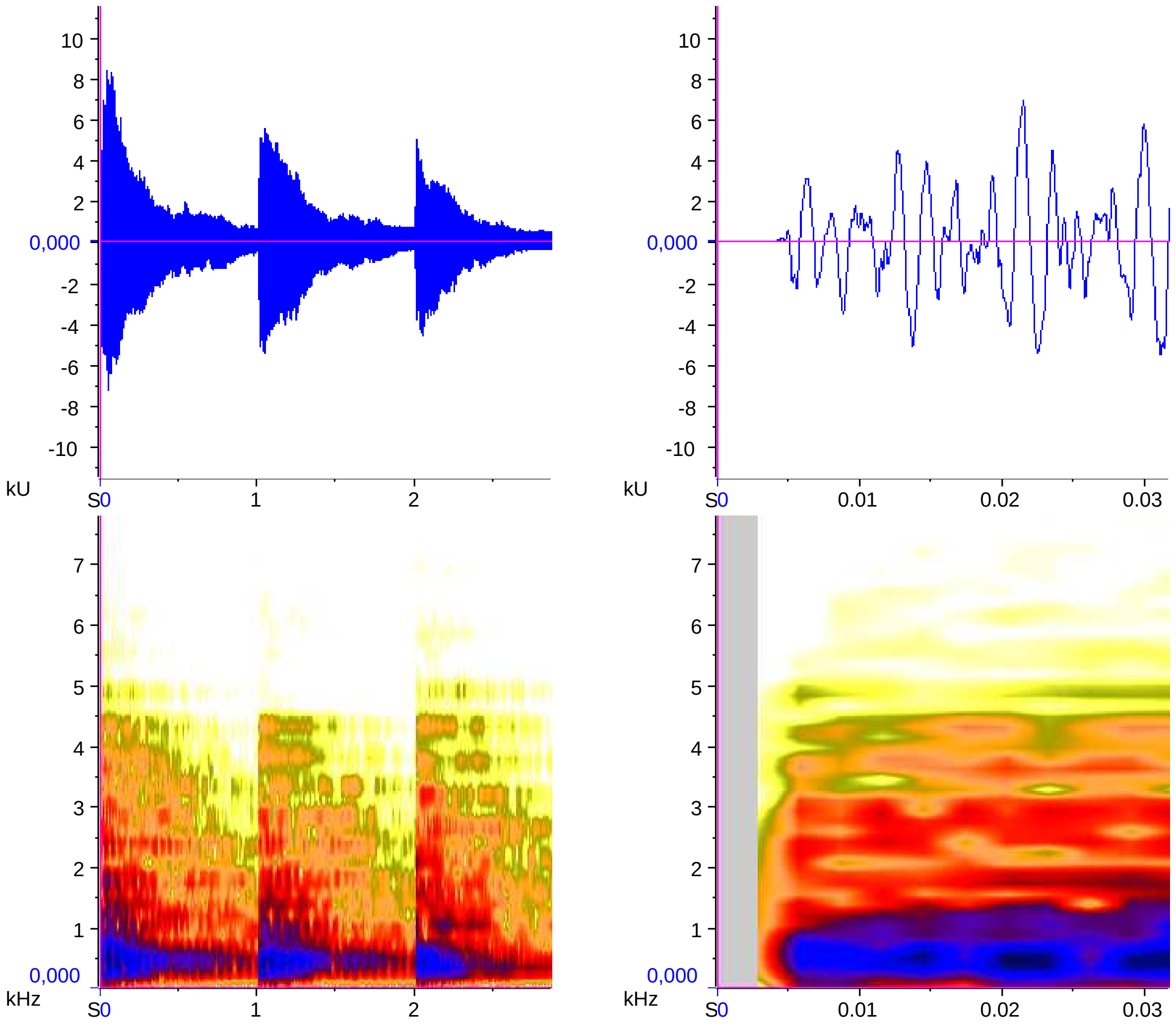

Figure A.2: Example for a visualization of a WAV file with a sampling rate of 44100 Hz. **Left:** First three chords of cyftlt-mono.wav. **Right:** Zoom into the first chord of the same file. **Top:** Waveform representation. The x-axis depicts the time in seconds where the y-axis shows the amplitude. **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. The blue color stands for a high intensity going over red and yellow to white which stands for a low intensity.

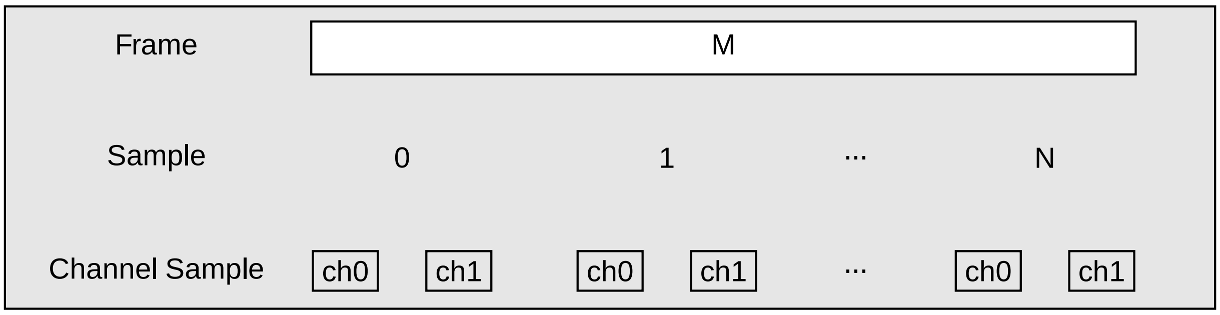

Figure A.3: Difference between *frames*, *samples* and *channel samples* for a stereo WAV file. In case of this thesis, M starts with 1 and N is the window width. **Adapted from:** Wikipedia [6].

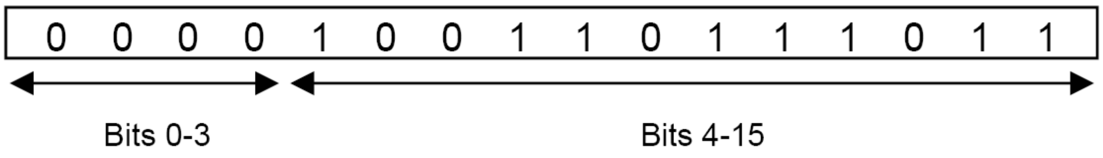

Figure A.4: Example for the bit alignment of a 12-bit sample. The 12 bits are padded by zeros to get 16 bits (which can be divided by 8). **Source:** Wikipedia [6].

Appendix B

DVD Contents

The attached DVD contains all the files used for this thesis including their denoised versions. These are located in the folder `experiments` which contains the following subfolders:

```
experiments
            ↪ audio_sources
            ↪ avg_samples
                              ↪ 2048-4096
                              ↪ 256-1024
            ↪ nl_inpainting
                              ↪ 2048-4096
                              ↪ 256-1024
            ↪ nl_means
                              ↪ 2048-4096
                              ↪ 256-1024
            ↪ nl_means-szlam
                              ↪ 2048-4096
                              ↪ 256-1024
```

The folder `audio_sources` contains all the input files used in the experiments.
In the subfolders of `nl_means-szlam`, the following files can be found (except for the clean version):

clean version	**noisy version**	**denoised version**	**iteration**
piano1-mono.wav	piano1-mono-noisy.wav	piano1-mono-noisy-corr-i1.wav	1
		piano1-mono-noisy-corr-i2.wav	2
		piano1-mono-noisy-corr-i3.wav	3
piano3-mono.wav	piano3-mono-noisy.wav	piano3-mono-noisy-corr-i1.wav	1
		piano3-mono-noisy-corr-i2.wav	2
		piano3-mono-noisy-corr-i3.wav	3

Each subfolder of `avg_samples`, `nl_inpainting` and `nl_means` contains the following denoised versions:

noisy version	**denoised version**
cyftlt-mono-short_sine.wav	cyftlt-mono-short_sine-corr.wav
cyftlt-mono-short_sine.wav	cyftlt-mono-short_sine-corr-1.wav
cyftlt-mono-long_sine.wav	cyftlt-mono-long_sine-corr.wav
cyftlt-long_sine.wav	cyftlt-long_sine-corr.wav
piano1-mono-white.wav	piano1-mono-white-corr.wav
piano3-mono-sine.wav	piano3-mono-sine-corr-version1.wav
piano3-mono-sine.wav	piano3-mono-sine-corr-version2.wav
piano3-mono-white.wav	piano3-mono-white-corr.wav
piano3-mono-white.wav	piano3-mono-white-corr-1.wav
candle_in_the_wind-mono-silence.wav	candle_in_the_wind-mono-silence-corr.wav
candle_in_the_wind-mono-silence.wav	candle_in_the_wind-mono-silence-corr-1.wav
your_song-mono-white.wav	your_song-mono-white-corr.wav
your_song-mono-white.wav	your_song-mono-white-corr-1.wav

Files in this table that carry a ”-1” at the end of their name are generated without excluding noisy frames where all others are created with the exclusion of the noisy part.

The parameter settings can be found in the files that have a ”.txt” attached to the name of the denoised version, for example candle_in_the_wind-mono-silence-corr.wav.txt.

Bibliography

[1] Audacity. `http://audacity.sourceforge.net/`.

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

[3] Propellerheads Reason 4.0. `http://www.propellerheads.se/`.

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

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

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

[7] Ronald N. Bracewell. *The Fourier Transform and its Applications*. McGraw-Hill, New York, 3rd edition, June 1999.

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

[9] Matthew Cooper and Jonathan Foote. Automatic music summarization via similarity analysis. In *Proc. 2002 International Symposium on Music Information Retrieval*, pages 81 – 85, Paris, October 2002.

[10] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *Proc. IEEE International Conference on Computer Vision*, pages 1033–1038. IEEE Computer Society, September 1999.

[11] Jonathan Foote. Visualizing music and audio using self-similarity. In *Proc. ACM Multimedia 99*, pages 77 – 80, Orlando, November 1999.

[12] Meinard Müller. *Information Retrieval for Music and Motion*. Springer-Verlag, Berlin, 1st edition, September 2007.

[13] Daniel N. Rockmore. The FFT: An algorithm the whole family can use. *Computing in Science & Engineering*, 2(1):60–64, 2000.

[14] Arthur Szlam. *Non-Local Means for Audio Denoising*. CAM preprint 08-56. University of California, Los Angeles, August 2008.