UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ-ROMÂNĂ

LUCRARE DE LICENŢĂ

Detecția și atenuarea distorsiunii în înregistrari audio de pe formate audio analogice mecanice

Conducător ştiinţific
Lect. Dr. Sterca Adrian

Absolvent
Drimba Alexandru

2018

BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

SPECIALIZATION COMPUTER SCIENCE - ROMANIAN

DIPLOMA THESIS

Distortion detection and attenuation on recordings from mechanical analog audio formats

Supervisor
Lect. Dr. Sterca Adrian

Author
Drimba Alexandru

2018

Contents

Chapter 1

Introduction. Problem statement and motivation

1.1. Introduction

We all listen to music. Some of us more, some – less, and some cannot live without it! Nowadays, it's very easy to listen to whatever you want, whenever you want, due to the technological advances made in the last decades. But this wasn't always the case. Until the late 1800's, there was no device that would allow sound playback, so music could be listened to only from live performances. With the invention of various sound record and playback devices, like the gramophone, the magnetic tape, and later – the digital formats, sound reproduction soon became accessible to everyone.

Even if many people nowadays consider the analog formats obsolete and rely solely on their electronic devices to listen to audio, those "considered to be extinct" formats are actually regaining their former popularity: new vinyl pressing plants are being opened due to the increasing demand[1] (even though turntable sales stay at a constant rate[2]), and cassette tape sales grew by 74% in 2016 (still, the amount of albums sold on cassettes is very small reported to the total number of albums sold) [3].

The melomaniacs are divided into three main groups: those who stick to the digital formats and don't want to hear about analog ones, those who consider that analog is by any means superior in quality to digital, and those who recognize the ups and downs of each of the formats. In reality, every storage medium has its advantages and disadvantages and favoring one of them is, in the end, a subjective choice.

Table 1.1 presents various qualities and defects of three main formats: digital formats (such as CDs, online streams, MP3s etc.), vinyl records and audio cassette tapes. One might say that the digital format is clearly the winner, but some still prefer the "touchable" formats for one or another of their unique features, being it the musical content inscribed on it, its historical value, the artwork and information on the cover or just for the feeling of owning a palpable collection.

Leaving aside preferences, it is a fact that there are recordings on non-digital formats that have not yet made their way off to a more easily to use media. Often, the analog media degrades over time, and in order to get the best out of a recording, restoration of some degree may be required. This is done by making use of **signal processing,** either using specialized electronic devices (analog signal processing), or by editing the sound on a computer (digital signal processing).

[1] Furnace Record Pressing Plant Is Ready To Roll: https://blog.discogs.com/en/furnace-record-pressing-plant/ (Retrieved 27.04.2018)
[2] Hipsters are buying vinyl records, but they aren't listening to them: https://qz.com/103785/hipsters-are-buying-vinyl-records-but-they-arent-listening-to-them/ (Retrieved 27.04.2018)
[3] U.S. Cassette Album Sales Increased by 74% in 2016: https://www.billboard.com/articles/columns/chart-beat/7662572/us-cassette-album-sales-increase-2016-guardians (Retrieved 27.04.2018)

	Format		
	Digital	Vinyl records	Cassette tapes
Advantages	Very easy to use and get access to	Artwork and packaging are much more intriguing than other formats'	Small and portable
	Great sound quality (when using appropriate encoding)	The engraved sound wave is visible; you can see what you are listening to	Cheaper than vinyl records
	Media itself does not degrade in time, only its storage medium does	Pretty good sound quality, relative to its predecessors	Easy to record, duplicate and edit
	Playback, recording and editing can be easily done	Can fit large amounts of information on the cover	Little wear over time
Disadvantages	Large file sizes for quality recordings	Prone to irreversible wear over time	Louder background noise than vinyl
	Audible compression artifacts (when using lossy encodings to reduce file size)	Needs specialized equipment for playback	Production faults, such as wow and flutter
	Easy to illegally share copyrighted content	Limited recording time	Limited recording time
		Large and heavy format; storage needs to be made carefully	Limited dynamic range and frequency response
		Costly to produce	

Table 1.1: Comparison between most common consumer-grade audio formats

1.2. A brief history of audio recording formats

History of sound recording begins with Édouard-Léon Scott de Martinville's phonoautograph (Fig. 1.1), an entirely mechanical device that was constructed as an analog to the human ear. It consisted of a funnel-like horn, with a flexible membrane covering the small end of the horn acting as a diaphragm. A lightweight stylus (or needle) was attached to the membrane and traced a line on the moving surface of a lampblack (carbon deposited by the flame of an oil lamp) coated paper. The sound waves in the air were captured and concentrated into the diaphragm by the horn, which would cause a movement in the diaphragm and, subsequently, in the stylus, causing a modulation in the traced line (Fig. 1.2) [1]. The phonoautograph recording is called a phonoautogram, but because of the recording medium's nature, it was impossible to play back. Way later, in 2008, playback was realized with computers, by optically playing high-quality scans of the recordings.

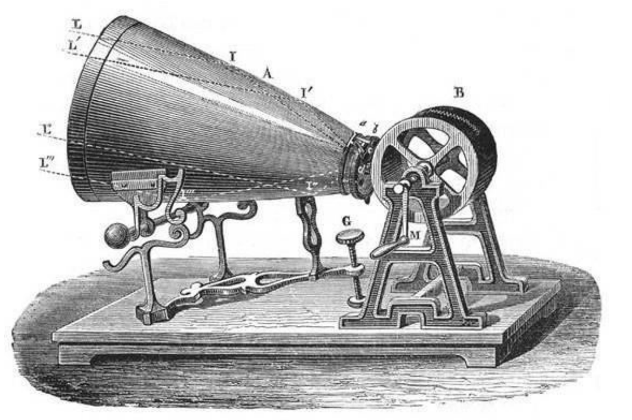


Fig. 1.1: The phonoautograph [4]

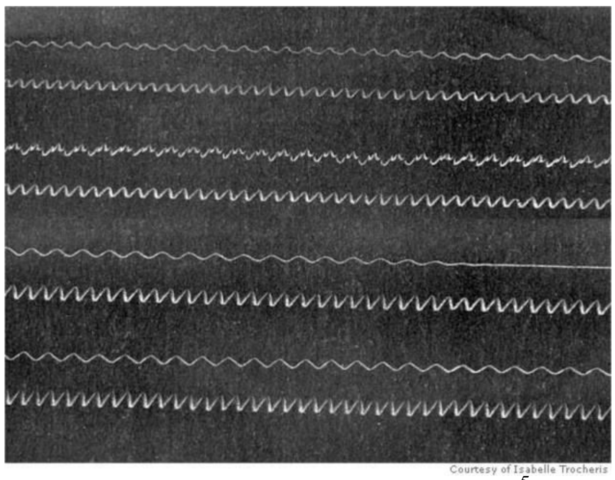


Fig. 1.2: Detail of a phonoautogram [5]

Using the same principle as de Martinville, Thomas Edison attached a stylus to a sound absorbing diaphragm and arranged such that the vibrations picked up by the diaphragm were translated into an up-and-down motion in the stylus. He also used a cylinder, but replaced the fumed paper with tinfoil. The stylus would rest on the cylinder, and as it rotated, a mechanism dragged the stylus sideways, creating a spiral. The stylus would press into the tinfoil and make a groove – a physical impression of the recorded sound. After running the machine while shouting "Mary had a little lamb" into the diaphragm, he ran it again, this time without speaking, and the machine spoke! The hill-and-dale groove in the cylinder moved the stylus, and thus the diaphragm, making it move in the same pattern as when recording.

Tinfoil was impractically fragile, as the recordings would wear out after a few plays. Changing the recording medium to wax and later, plastic materials, solved the durability problem, and it also sounded better. From 1888, Edison started producing phonographs (Fig 1.3) for home use. The actual records are called wax cylinders, and were on the market until 1929, when they died out in favor of other formats. [2]

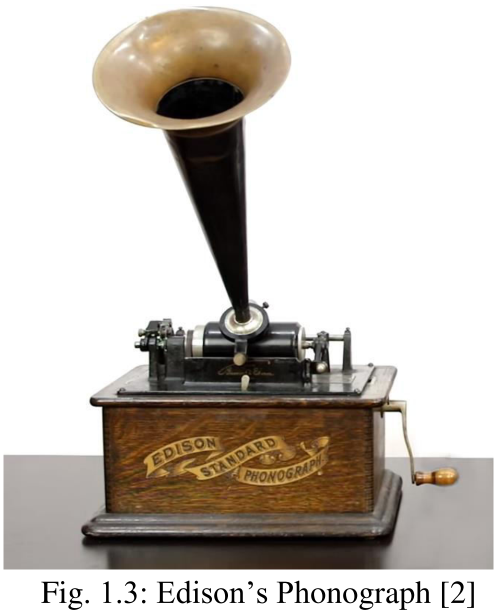


Fig. 1.3: Edison's Phonograph [2]

[4] Léon Scott de Martinville réalise le tout 1er enregistrement sonore: http://ginsteve.e-monsite.com/blog/un-dimanche-une-decouverte/9-avril-1860-leon-scott-de-martinville-realise-le-tout-1er-enregistrement-sonore.html (Retrieved 27.04.2018)

[5] Physicists convert first known sound recording: https://www.sfgate.com/news/article/Physicists-convert-first-known-sound-recording-3289341.php#photo-2437531 (Retrieved 27.04.2018)

Meanwhile, Emil Berliner was experimenting with another method of recording sound into a wax record. The first change he made was switching from the up-and-down motion of the stylus to a left-right wobble (Fig. 1.4). He realized that using a flat disc instead would make the engraving a lot easier, by recording into a softer material and using it as mold to create a hard stamper. Berliner's first discs were released around the same time as Edison's cylinders were hitting the market. [3]

With the advance of the electronic technology, electrical microphones, signal amplifiers and electromechanical devices were developed, and they heavily impacted the industry of audio recording. Starting from 1925, phonograph recording masters were cut with electrically powered cutting heads, which increased the quality of the recording by means of frequency response, dynamic range and sound clarity. Playback could also be made electrically, by picking up the stylus' movement as an electrical signal, amplifying and providing it to a loudspeaker. [4]

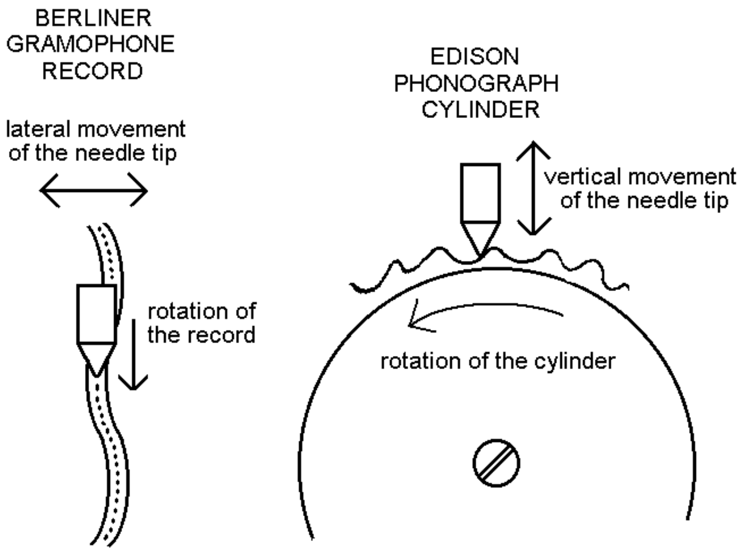

BERLINER GRAMOPHONE RECORD
lateral movement of the needle tip
rotation of the record
EDISON PHONOGRAPH CYLINDER
vertical movement of the needle tip
rotation of the cylinder

Fig. 1.4: Comparison between disc and cylinder, lateral and vertical-cut [6]

The gramophone disc was still present. It has increased in fidelity over years: the electric recording was introduced; then, in 1948, the vinyl record appeared. By using the more fine grained PVC instead of shellac, the grooves could be more compact and the speed went down, from 78 RPM to 33⅓RPM or 45RPM. Thus, the new format brought more durability, longer playing times and a great improvement in sound quality. In 1957, the first stereophonic disc was released, by combining vertical-cut with horizontal-cut to provide two moving axis for the reproducing stylus. The vinyl record (Fig. 1.5) stayed the most popular format until the rise of the digital formats, starting with the CD.

Storing the audio signal in mechanical or magnetic form introduces elements of confusion because of the storage medium itself. Analog recording devices have a limit in their capability of tracing a line between their input and output, mostly due to the inherent characteristics of the storage medium, like limited signal-to-noise ratio, frequency response, dynamic range and many others. The idea of recording something that is not a direct analog of the desired information, but a descriptive form that avoids the quality traps of ordinary recording is the idea of digital recording. [5]

[6] Vertical-cut (cylinder) vs lateral-cut (Berliner disc): http://www.soundfountain.com/amb/ttadjust.html, (Retrieved 10.06.2018)

In the basic system of digital recording, the electrical analog of the sound wave pressure variations being recorded is sampled at fixed intervals. This operation converts the smoothly varying waveform into a series of amplitude values. A number is then assigned to each of these values. This results in an approximation of the original waveform. The first commercial digital format was the CD (Compact Disc), which was designed to be the successor of the vinyl. It stored the audio signal optically, as a sequence of binary numbers, each representing the value of a sample's amplitude. It replaced the vinyl record and cassette tape by the early 2000's, just to be rendered obsolete with the advent of internet-based distribution of files.

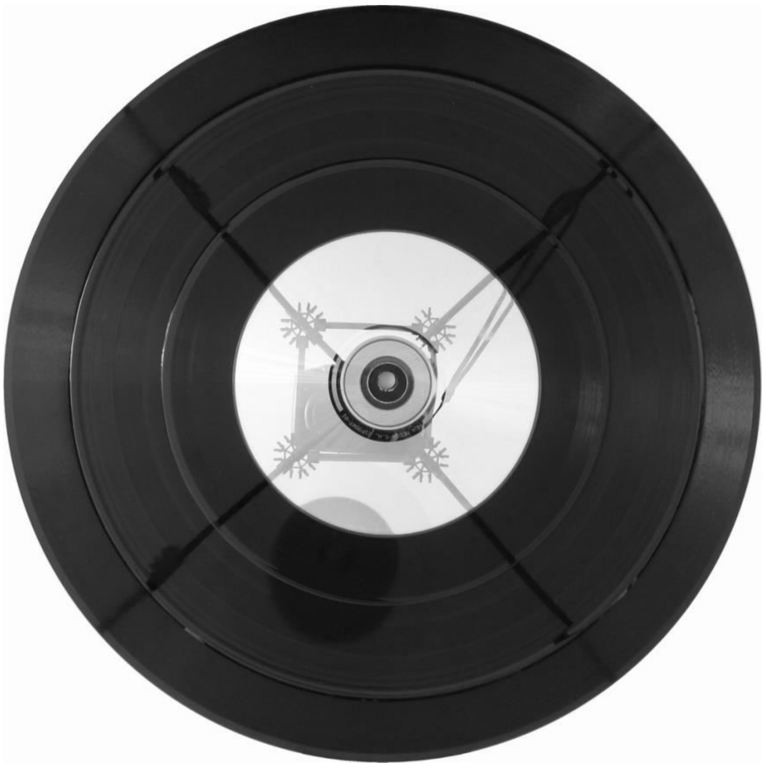


Fig. 1.5: Some of the "flavors" discs come into. From outside to inside: 12" shellac at 78RPM – max. 5 min/side; 10" vinyl at 33⅓ RPM, up to 20 min/side; 7" vinyl at 45 RPM, typically 4-5 min/side, but can go up to >9 min/side[7]; 5" CD, constant linear speed, max. 80 min

1.3. Mechanical analog storage. Recording and playback

The life of a record starts with sound being provided to a cutting head, which cuts into a lacquer. After the lacquer hardens, it is electroplated, resulting in a metallic negative stamper. The stamper is then used to press multiple copies of the original lacquer, but the pressed material being made of much durable plastic pellets.

Both cutting and playback styli are made of very hard materials, like sapphire and diamond, in order to limit the wear it receives over time. A worn stylus will, besides giving poor sound quality, also damage the groove it is playing. When recording or reproducing sound, styli move laterally (L-R) for monaural recordings, or a combination of vertical and lateral (Fig 1.9) – for stereo recordings (i.e. each channel is cut 45° from the vertical axis (Fig 1.12)).

The playback transducers (Fig 1.8), which transform the stylus' movement into electrical current, are called cartridges. Playback styli shapes are usually less complex than those of recording styli, the most common being spherical and elliptical, for they are much easier and cheaper to produce. More exotic shapes, like Hyper-Elliptical, Shibata, S.A.S and MicroLine exist (Fig 1.15). Their purpose is to track the groove as precisely as possible, resulting in increased sound quality, especially in highly modulated passages, all while wearing the grooves less than conventional styli.

[7] Deutsche Grammophon 30 053 EPL, 7" 45RPM vinyl – Wolfgang Amadeus Mozart - Eine Kleine Nachtmusik KV 525

Cartridges are characterized by lots of properties, which summed together show how the cartridge will act at playback time. These specifications include: frequency response, tracking force (the force the stylus puts on the groove), output current, impedance and capacitance, compliance and others.

Electric cartridges and cutting heads consist of one or more coils and one or more permanent magnets. When the magnetic flux through a coil's surface changes, Faraday's law of induction states that the conductor acquires an electromotive force (measured in volts), which is equal to the rate of change of the magnetic flux (Eq. 1.1) [6]. Similarly, when a current is passed through a coil, it generates a magnetic field and moves the coil relative to the permanent magnetic field surrounding it.

If Eq. 1.1 is interpreted for our case, the induced voltage in a coil is proportional to a coil's movement relative to the magnets. That means a cartridge's output voltage is given by the stylus' speed while tracking a groove. Analog, for a cutter head, the stylus' displacement (relative to its rest position) is given by the integral over the provided signal's voltage.

$$\mathcal{E} = -\frac{\partial \Phi_B}{\partial t} \qquad \text{(Eq. 1.1)}$$

$\mathcal{E} - the\ Electromotive\ force\ (V),$
$\Phi_B -\ the\ magnetic\ flux(Wb),$
$t - time(s).$

Because of this, when cut into the cutting lathe, low frequencies have much larger amplitudes than high frequencies at the same input signal amplitude. The equivalent frequency response of transducing an electric current into physical movement is shown in Fig 1.6. To counter this, various equalization curves were used (pre-emphasis before cutting, de-emphasis on playback). Besides reducing the groove's width (which would permit greater recording times), equalization curves also improve sound quality by amplifying the high frequencies to reduce surface noise and clicks. In 1954, the RIAA equalization became the industry standard for records. It consists of a pair of recording and playback equalization curves (Fig 1.7) which cancel each other out. The playback curve is applied in the phono preamp, a device that boosts the low voltage produced by the cartridge (<10mV) to greater levels (>500mV) so it can be used by an audio receiver such as an amplifier.

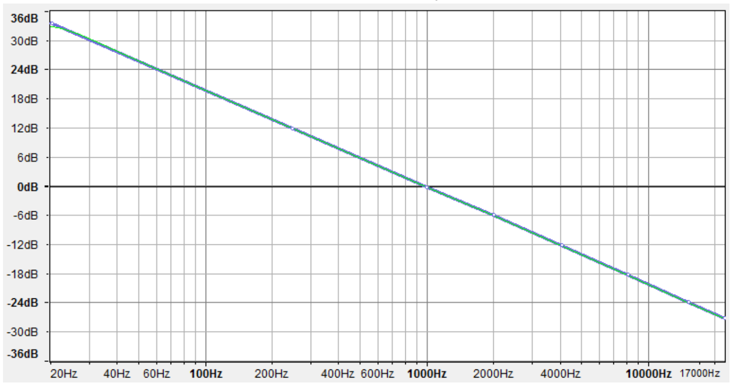


Fig. 1.6: Frequency response equivalent to transducing an electric current into physical movement

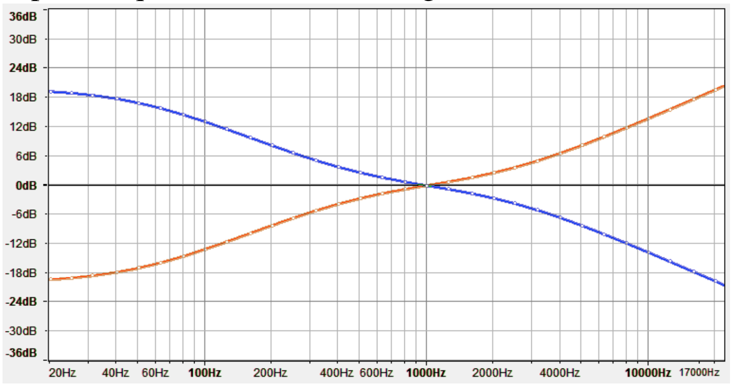


Fig. 1.7: RIAA playback curve (blue), RIAA recording curve (red)

1.4. Causes of distortion in mechanical analog formats

By using an unadjusted turntable or a poor quality tonearm or cartridge (worn stylus, big tracking force, low compliance), the record groove wears out (Fig 1.14). Irreversible. This is a big disadvantage of the discs. Also, having the recorded surface always exposed, it is prone to dust and grease accumulation, scratches (Fig 1.11) and other permanent damages. These damages will be generally called distortions, i.e. portions that significantly differ from the original state of the media.

Here are the most significant distortion types occurring on vinyl records:

- clicks and pops: When you hear “vinyl record”, you associate it with the sounds of clicks and pops. They are mostly caused by scratches that locally damage the grooves (Fig 1.11). Dust accumulation, electrostatic discharges or manufacturing defects are also causes for them. Clicks have a wide range of periodicities: from a few per record side, up to tens of them every second.
- white noise/hiss: This should not be confused with the inherent surface noise of the recording medium itself. Every play wears the groove a bit, even if the used stylus is in perfect condition. This is because all the tracking force (which varies from 1g up to over 10g for old cartridge types) is concentrated on two very small surfaces: the points the stylus is touching the groove. Big pressures cause large amounts of friction, which shears little by little the plastic material the record is made of. If a chipped or worn out stylus is used, the wear it causes to the groove is greatly increased, but still not as worse as the record being exposed to chemical solvents, fungi or high temperatures. White noise is usually heard as a continuous sound, lasting from a few hundreds of milliseconds up to whole minutes.
- mistracking: In highly modulated passages, when the groove violently wobbles on the two axes, the needle may not be able to accurately track the groove underneath it. This is caused by the large forces it is subjected to, and also the angles it attacks the groove walls with. The stylus will run into the groove wall with greater force than usual and cause deformations at the places of contact. The stylus may even go up the groove’s wall, losing the contact with the other wall, and start craving a new groove as it digs through the vinyl (Fig 1.10). Tracking ability of a cartridge depends on its stylus’ shape, material and wear, cantilever weight, compliance and tracking force. Also, when approaching the record’s center, the linear velocity decreases and the stylus’ radius in the contact points may become larger than the radius of the groove it touches (Fig 1.13). This is called the pinching effect. The stylus does not properly fit in the groove, causing groove deformation. Also a vertical movement occurs in the stylus, as it has nowhere else to go. When occurring, mistracking causes groove damage even in thousands of points in the time span of a second.

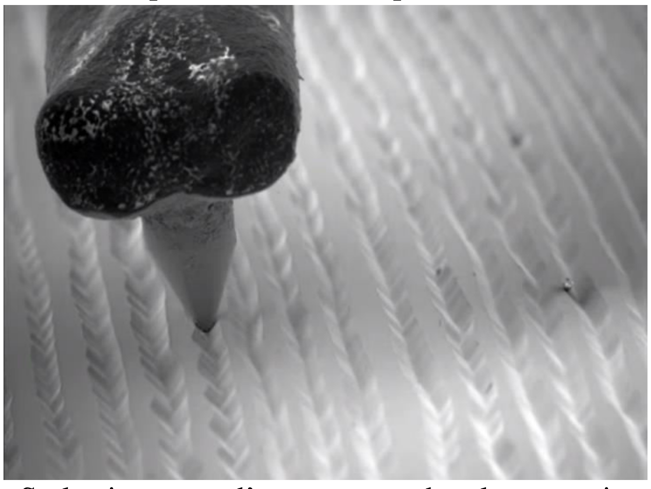


Fig. 1.8: Stylus in a record’s grooves under electron microscope [8]

[8] Electron microscope slow-motion video of vinyl LP: https://www.youtube.com/watch?v=GuCdsyCWmt8

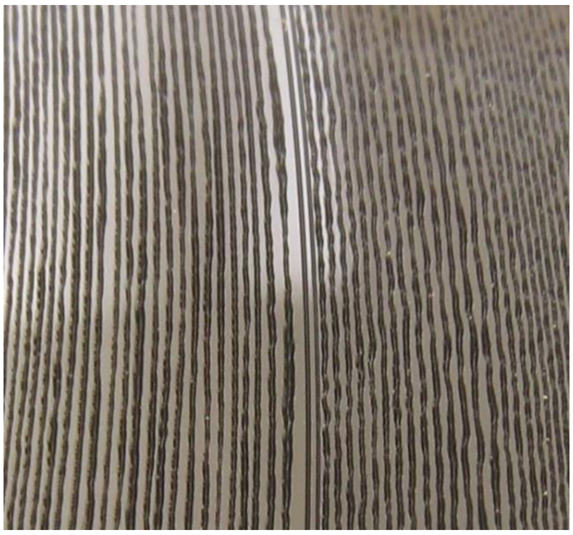


Fig. 1.9: A stereo groove moves both on the vertical and horizontal axis

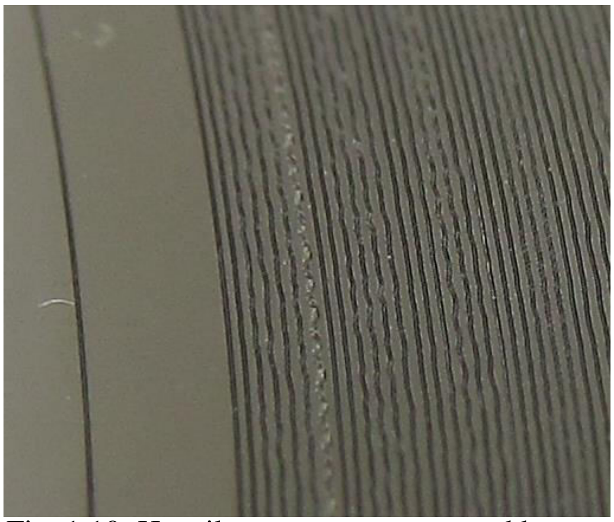


Fig. 1.10: Heavily worn groove, caused by poor playback equipment

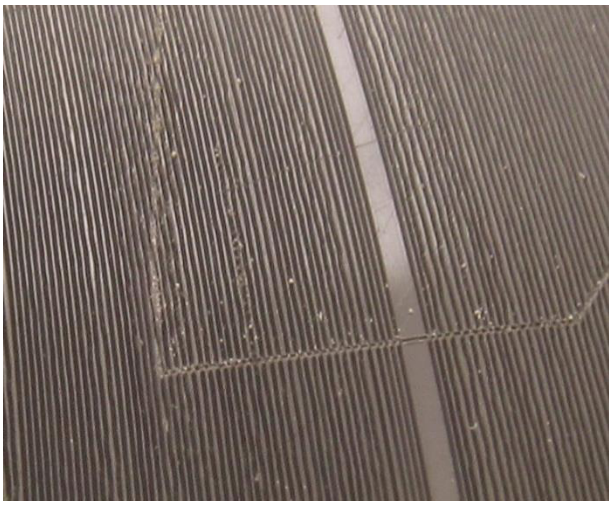


Fig. 1.11: Deep scratch causing the well-known pops

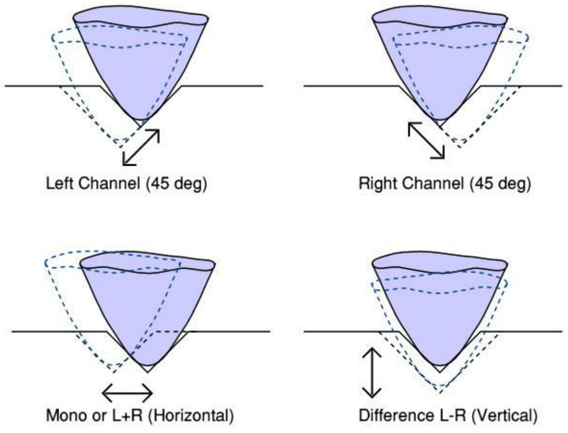


Fig. 1.12: Stylus' movements in a stereo recording [10]

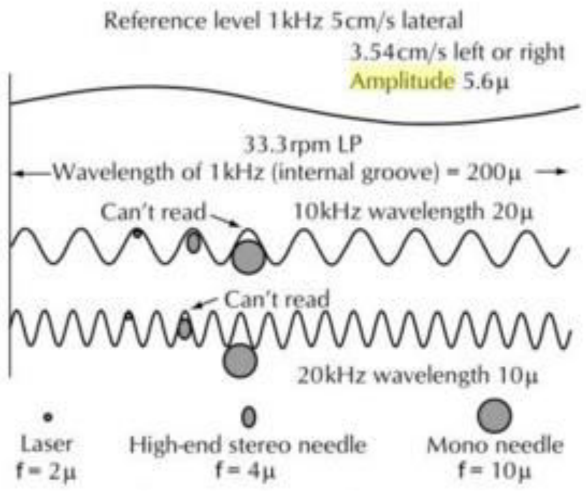


Fig. 1.13: The pinching effect [11]

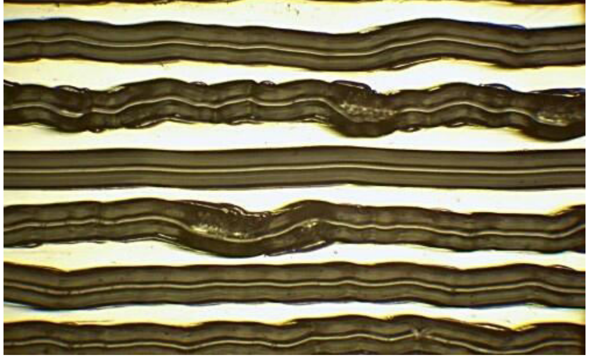


Fig. 1.14: Groove wear under microscope [12]

[9] Stereo groove modulation: https://www.psaudio.com/pauls-posts/the-great-mystery-of-vinyl/ (Retrieved 30.04.2018)
[10] Groove pinching: https://i.stack.imgur.com/wgerL.png (Retrieved 30.04.2018)
[11] Wear in the loud passages of a mono vinyl record: http://www.micrographia.com/projec/projapps/viny/viny0200.htm

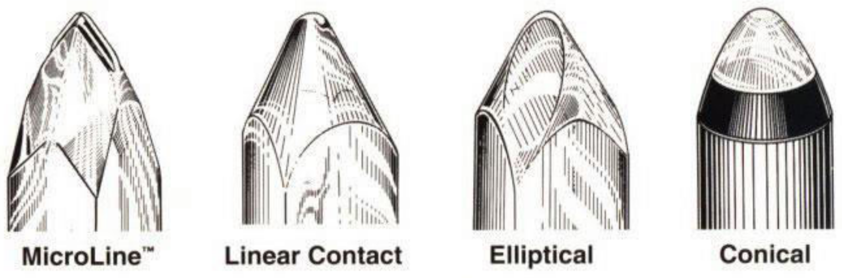


Fig. 1.15: Comparison of various styli shapes [12]

1.5 Purpose of this work

Excessive levels of distortion make the listening unpleasant. If the recording cannot be found anywhere else, either on another recording medium or on a better-shaped record, or it is simply too difficult to obtain another recording, one would like to recover the original sound or at least bring the current one to an acceptable condition.

The purpose of this work is to create a software application that will automate the process of correcting various distortions occurring at groove level. The targeted distortions will be clicks, pops and some types of mistracking. By using various digital signal processing methods, the goal is to obtain a sound that resembles as much as possible the original engraved sound.

[12] Advanced Stylus Shapes: Pics, discussion, patents: https://www.vinylengine.com/turntable_forum/viewtopic.php?f=19&t=22894 (Grabbed 01.05.2018)

Chapter 2

Related work (applications)

2.1. Audacity

Audacity is a free, easy-to-use, multi-track audio editor and recorder for Windows, Mac OS X, GNU/Linux and other operating systems [13]. It is open source software written in C and C++. Because of Audacity's free license, many people have contributed code, bug fixes, documentation and graphics [14]. Audacity provides lots of editing capabilities, from simple effects like amplifying, low- and high-pass filters, equalizers, fade in and fade out, to more complex effects such as speed or pitch change, noise reduction and dynamic compression.

One of the effects is called "Click Repair", and it seems to promise the user it will solve the problem of clicks appearing on records. The functionality has as inputs two parameters: "Threshold" and "Max Spike Width", which can be adjusted by the user. Its performance is not as impressing as other built-in effects, the results actually being rather poor and often causing more distortion than in the original recording.

In Fig 2.1, the results of the effect can be seen. With blue – the undamaged parts of the signal; with yellow – the damaged parts; with green – repairs made in damaged regions, with red – repairs made in undamaged regions. It can be seen that even the repairs made in damaged sections are inaccurate, and also there are lots of "repairs" in undamaged areas. The repairs use linear piecewise interpolation (it traces a straight line from one point to another), which is a very poor choice for audio signal processing purposes.

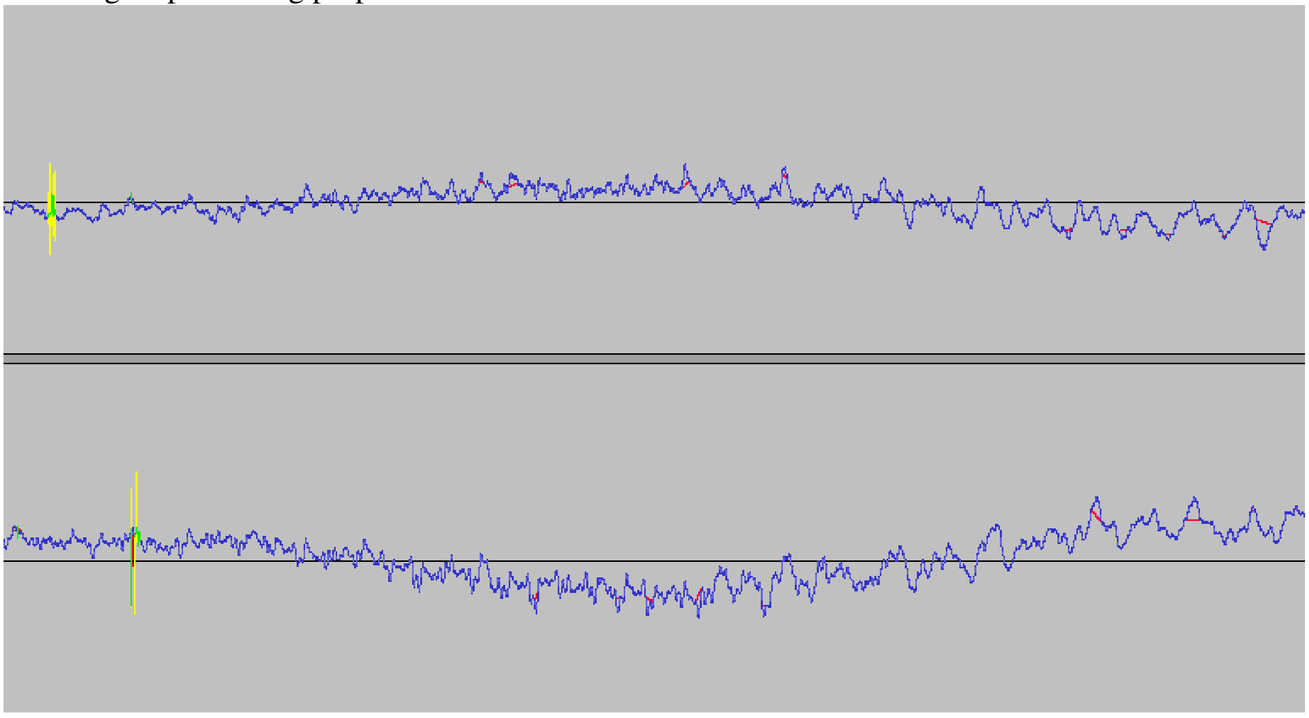


Fig 2.1: Click Repair effect in Audacity

[13] Audacity download page: https://sourceforge.net/projects/audacity/ (Retrieved 10.06.2018)
[14] FAQ:About Audacity: http://manual.audacityteam.org/man/faq_about_audacity.html (Retrieved 10.06.2018)

Audacity also has an effect called “Repair”. It can be used on regions up to 128 samples long and reconstructs the samples from a selected section based on the audio outside the selection. The region length limit is caused by the high computational complexity of the interpolation algorithm, which becomes less accurate and exponentially slower as the selection length increases. Its results are very satisfying and accurately reproduce a bad section based on the surrounding audio. It is understandable why this was not used in the “Click Repair” effect (high computational time), but it’s curious why the developers wouldn’t come up with some less complex but higher quality interpolation technique.

2.2. Nero WaveEditor

Nero 7 Ultra Edition is a software suite that, besides the well known CD and DVD burning facilities, also includes tools for image, audio and video processing. One of those tools is “Nero WaveEditor”, a program for editing and recording audio files. Like Audacity, it provides standard audio processing effects such as equalization, dynamic compression, fading and noise reduction.

The discussed effect will be “Declicker”, which also promises to get rid of pops and crackles. This one does its job, and it does it very good! As it can be seen in Fig 2.2, it accurately repairs clicks in not-that-bad records. Yellow represents the damaged sections, green – the reconstructed data, blue – untouched samples. The data reconstruction algorithm is way better than Audacity’s linear interpolation, and the detection of damage is more precise.

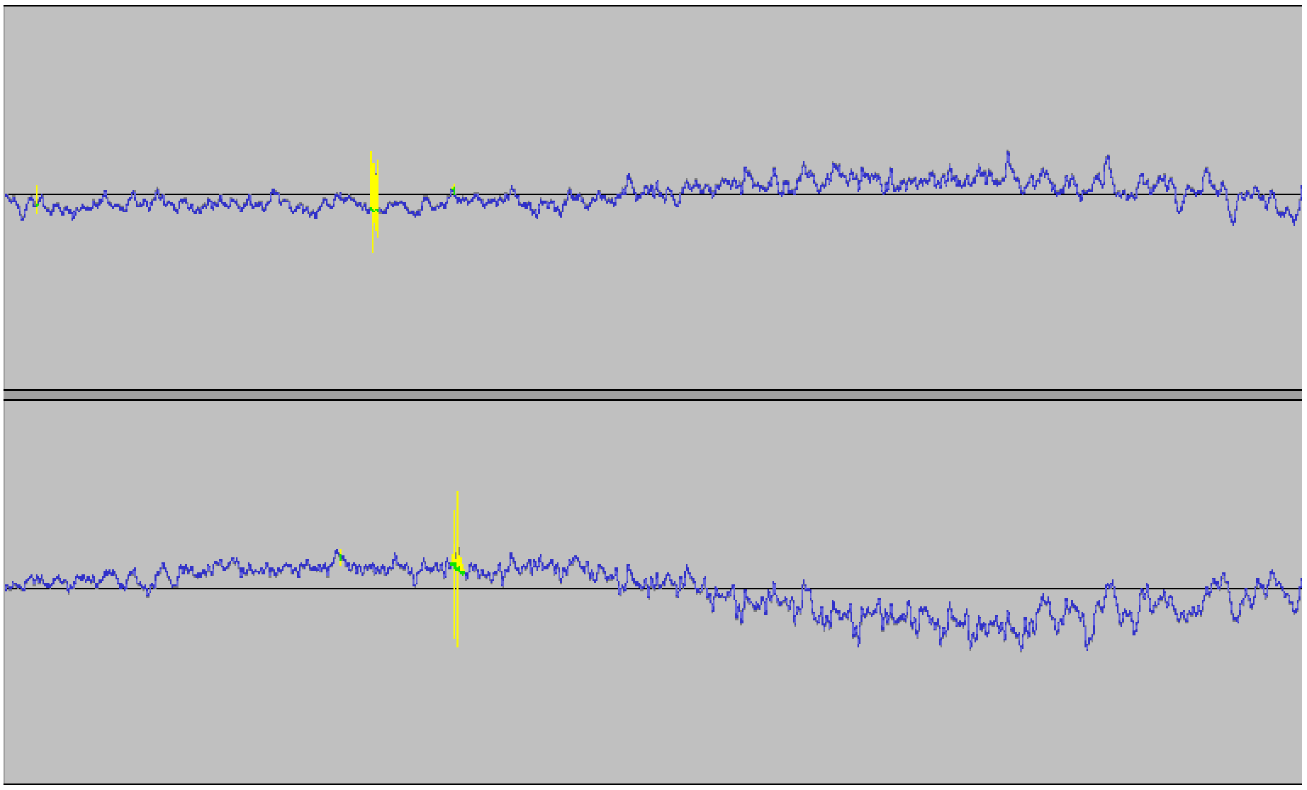


Fig. 2.2: Declicker effect in Nero WaveEditor, applied on the same recording as in Fig. 2.1

Its capability does not stop here. It can also handle very clicky recordings, like the one in Fig. 2.3. At this level of sound degradation, it is impossible to exactly reconstruct the recording; some noise will still be there, in a form or another. It is unrealistic to expect the processed recording to come out as clear as a freshly pressed record was, but the great reduction of distortion does make a huge difference.

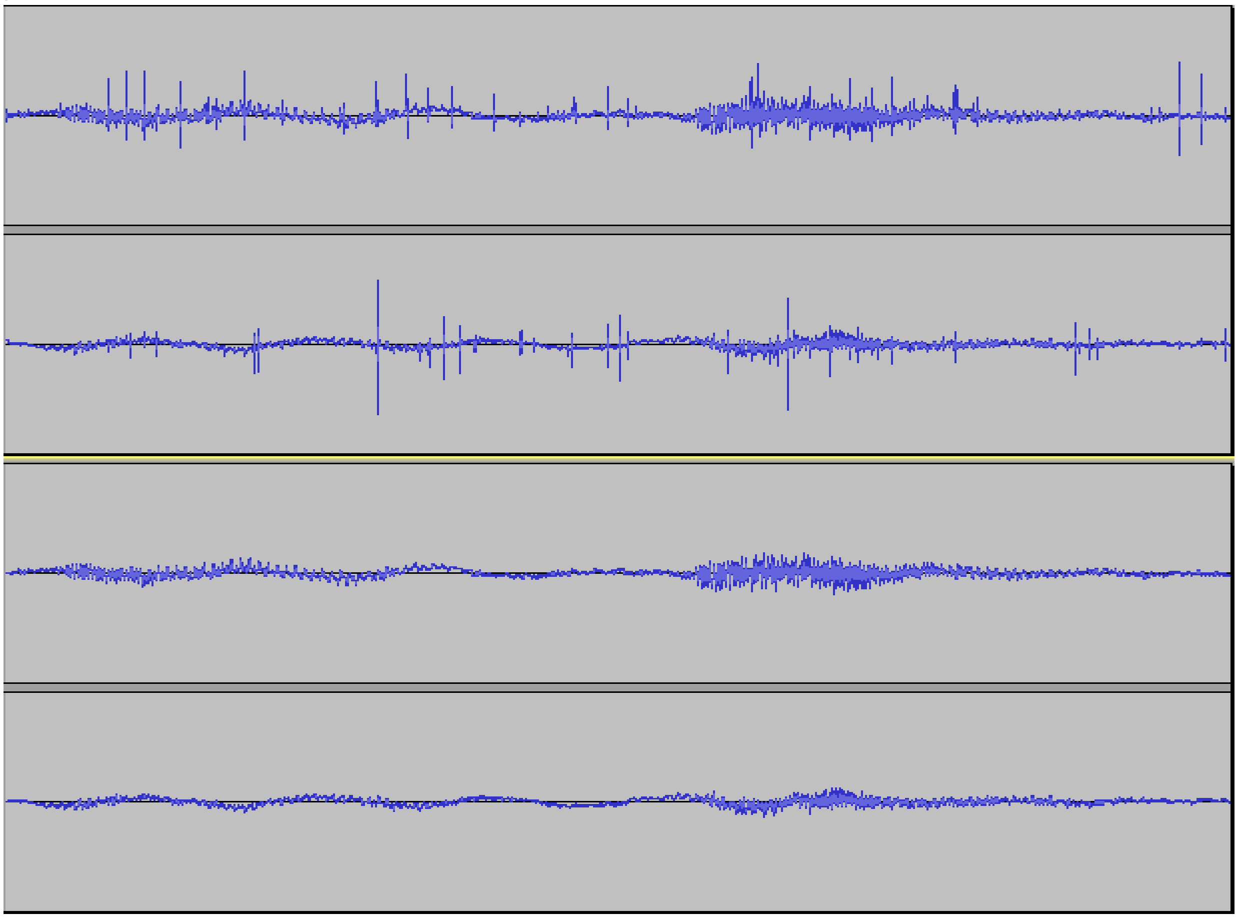


Fig. 2.3: Nero WaveEditor's Declicker applied on a really scratched record.
Top two tracks: before; bottom two tracks: after.

This effect also handles some forms of mistracking pretty good. With high enough sampling rate, it corrects very narrow spaced clicks with outstanding results. This is very similar to the example presented in Fig. 2.3, but with the clicks less intense and way more often. Fig. 2.4 shows a side by side comparison between three versions of the same recording. The top spectrogram is plotted based on the recording of a vinyl record heavily affected by groove wear. The middle spectrogram is obtained from the previous recording after passing it twice through the Declicker. The bottom spectrogram shows the same music as on the vinyl, but this time taken from an audio cassette. Declicker does a nice job here too, but the results it can achieve are limited by the very bad condition of the recording.

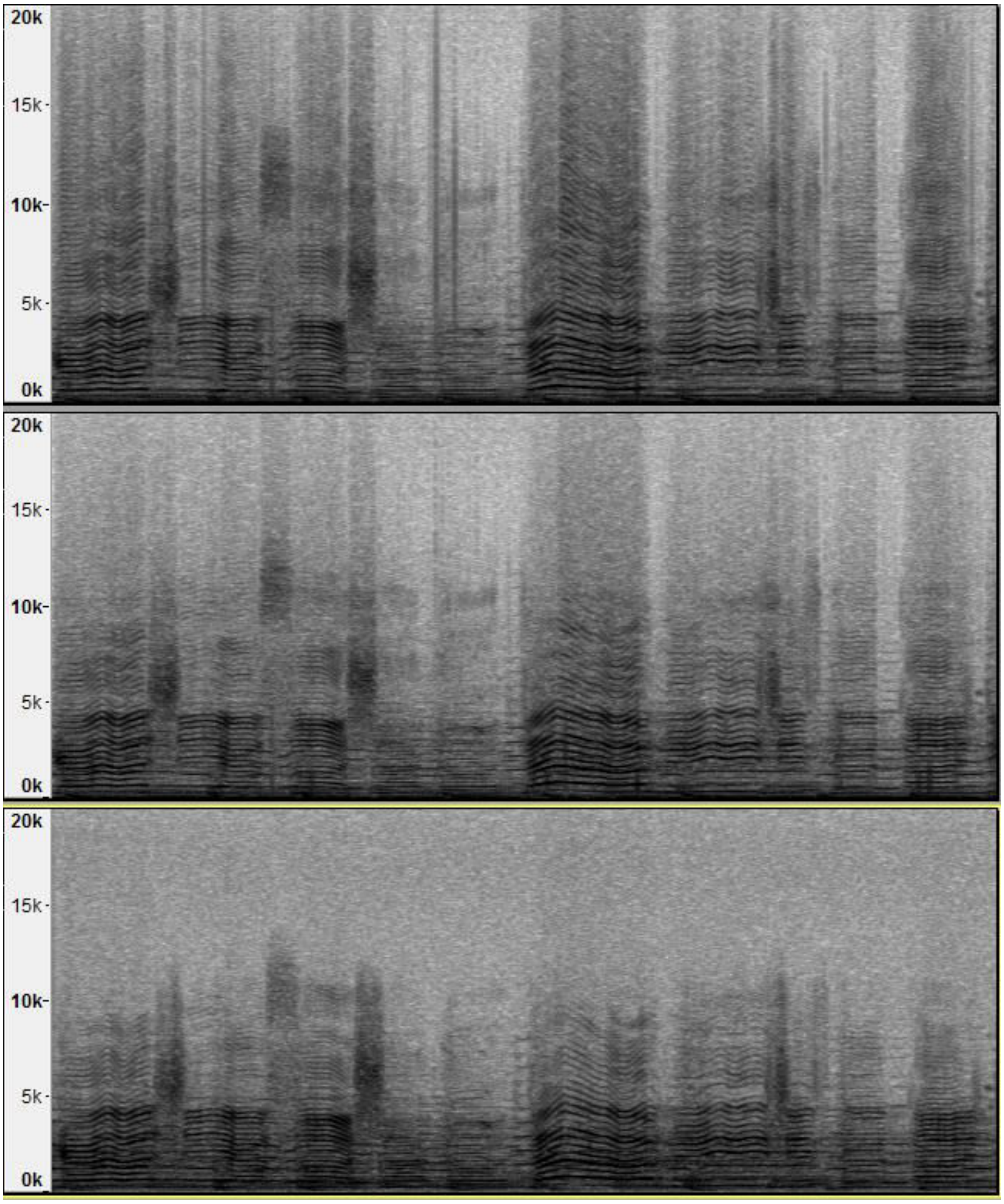


Fig. 2.4: Spectral comparison of the same audio: From a worn vinyl, after "declicker" was applied on the worn recording, and from an audio cassette.

Chapter 3

Basic digital signal processing

3.1. Digital audio signal representation

Sound is defined as an oscillation in pressure, stress, particle displacement, particle velocity, etc., propagated in a medium with internal forces (e.g., elastic or viscous), or the superposition of such propagated oscillation[15]. This oscillation can be represented as a continuous function that describes the variation in time of the medium's pressure, allowing us to "see" sounds. (Fig. 3.1)

Sound is transmitted through gases and liquids as longitudinal waves, and through solids both as longitudinal and transverse waves. A transmitting medium is required, so sound cannot travel through vacuum. In a longitudinal wave, the direction of displacement is the same as the direction of propagation, while in a transverse wave, the direction of displacement is perpendicular to the direction of propagation[16]. To better understand the difference between the longitudinal and transversal waves, Fig. 3.1 depicts sound waves in air (longitudinal waves), while Fig. 3.2 presents transverse waves travelling through a metal wire. As seen in Fig. 3.1., a pure tone sound wave travels through air as a sinusoidal pressure variation in the air. The air motion which accompanies the passage of the sound wave will be back and forth in the direction of the propagation of the sound, a characteristic of longitudinal waves[17]. In Fig. 3.2., the bow's movement drags the chord, its displacements being transmitted as transverse waves moving along the chord with the chord's speed of sound.

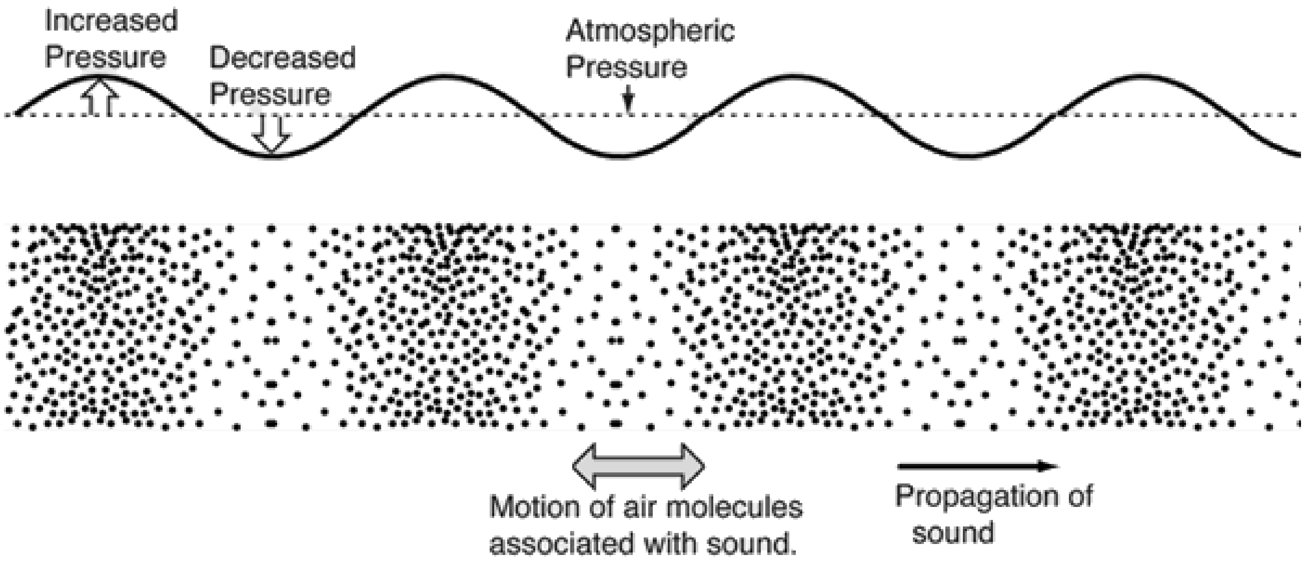


Fig. 3.1: Sound Waves in Air [18]

[15] American National Standard on Acoustical Terminology, ANSI/ASA S1.1-2013 – definition of sound
[16] Definitions of longitudinal and transverse waves: http://www.dictionary.com (Retrieved 10.04.2018)
[17] Sound waves in air: http://hyperphysics.phy-astr.gsu.edu/hbase/Sound/tralon.html (Retrieved 10.04.2018)

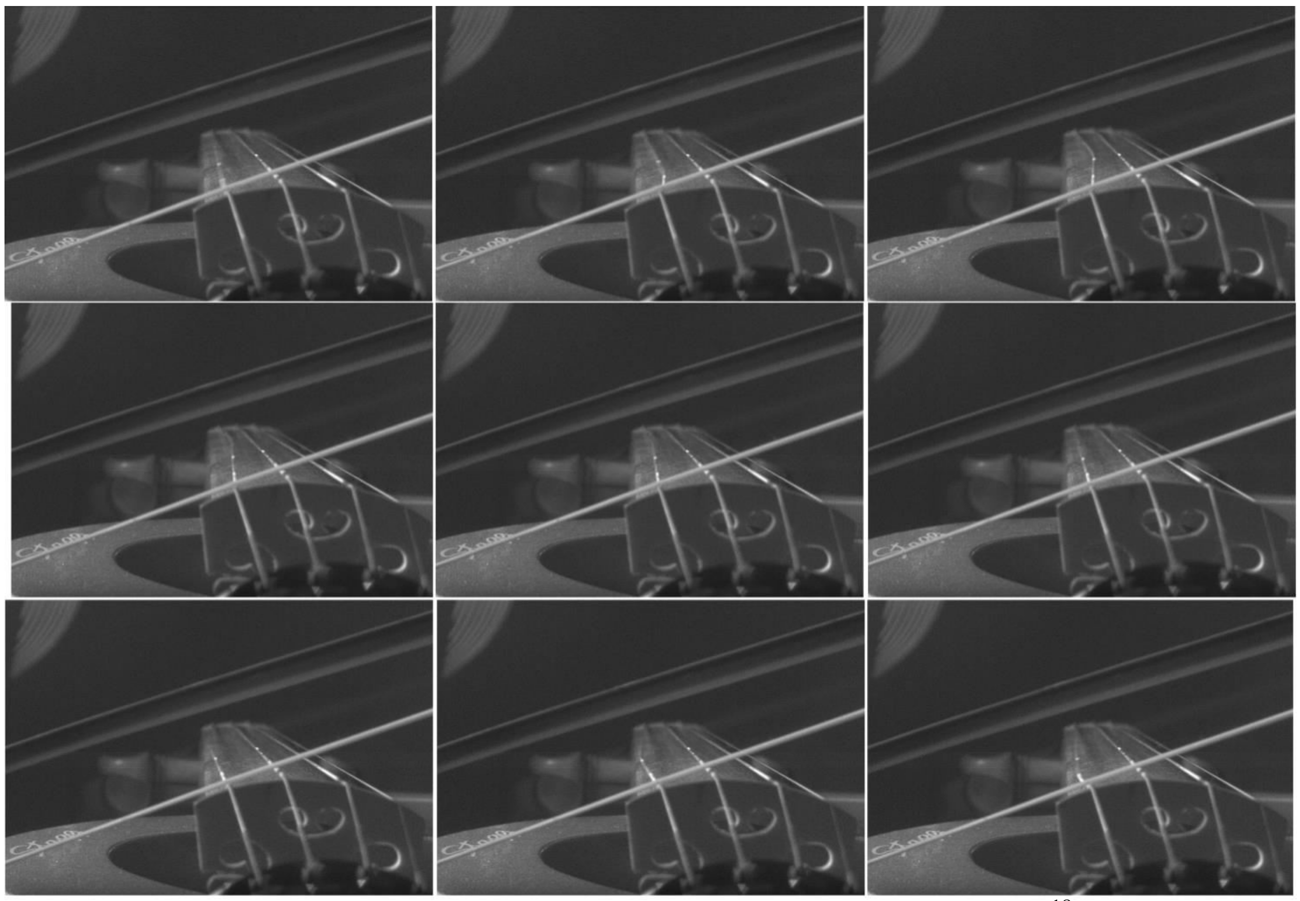


Fig. 3.2: Transverse waves seen in a bowed violin chord [18]

Audio signals are representation of sound, typically as an electrical voltage (analog) or as discrete numerical values (digital). Conversion from the analog continuous-time signal to the digital discrete-time signal (also called sampling) is made usually with ADCs (Analog to Digital Converters), and with DACs (Digital to Analog Converters) from digital to analog. A **sample** is a signal's value at a point in time**.** When converting from analog to digital, some of the information is lost because of factors like:

a. discretization – the resulted signal is no longer continuous (precisely defined in every point in time), but discrete: the intensity of the analog signal is recorded at fixed time points. The number of equidistant time points in a second is called **sample rate** (or sampling frequency)**.** The highest frequency that can be carried by the signal, called the **Nyquist frequency**, is given by the formula 3.1.;

$$F_{Nyquist} = \frac{F_{sampling}}{2} \tag{3.1}$$

b. storage as finite numbers – as opposed to analog values, the precision of the digital values is finite, so only some of the significant digits can be stored.

Figure 3.3 and Table 3.1 show the process of conversion from an analog signal to a digital one. In Fig. 3.3, a sine wave is sampled 20 times for each cycle. Each sample is then stored as a 8-bit signed integer (values in [-128, 127]). Table 3.1 shows the discretization errors raised at the conversion. The digital samples are stored as 8-bit signed integers, with values in [-128, 127], rescaled here to [-1,1) to show the error. Analog samples are in the range [-1,1], where -1 is the smallest possible signal value, and 1 is the maximum. The errors are pretty large for the chosen sample encoding. By choosing an appropriate sample rate and sample encoding though, the lost information (error) can be small enough to be considered negligible.

[18] Bowed violin string in slow motion: https://www.youtube.com/watch?v=6JeyiM0YNo4 (Retrieved 11.04.2018)

After getting the sample values from the original analog signal, the samples are then stored as digital numbers in audio files. These sample values can be stored either as uncompressed files, like the WAV and AU formats, or as compressed files (to decrease file size). Compressed file formats can be lossy (the decompressed data is an approximation of the original), such as MP3, or lossless (compression preserves the exact original values), such as FLAC. Audio files typically contain information about the sampling rate, number of channels and sample encoding (float/integers, signed/unsigned, bit-depth, companding and others).

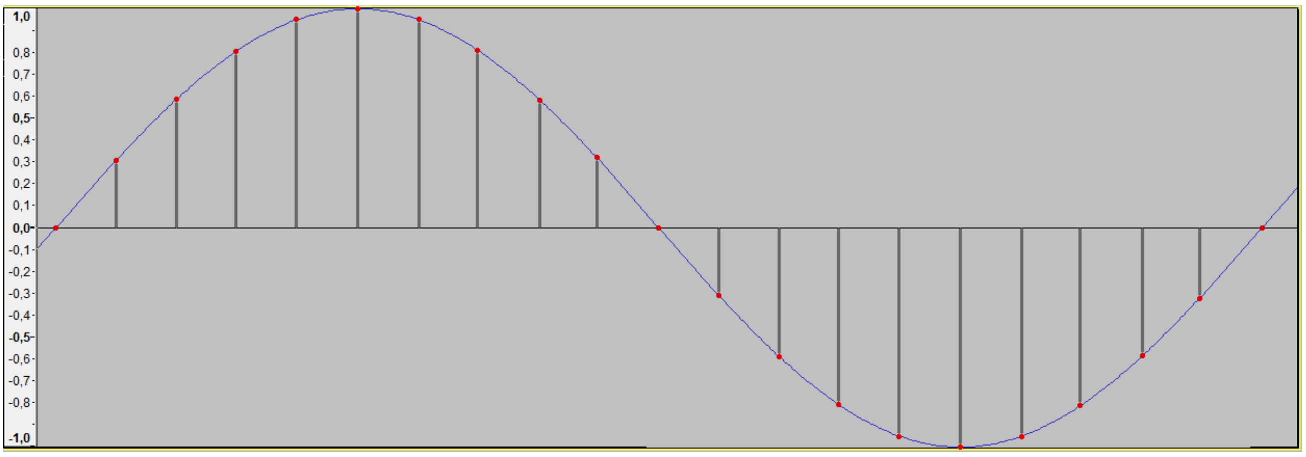


Fig. 3.3: Conversion from continuous-time to discrete-time.
With blue – the original signal, with red – the sampled values.

Sample no.	0	1	2	3	4	5	6	7	8	9	10
Actual value	0,00000	0,30902	0,58779	0,80902	0,95106	1,00000	0,95106	0,80902	0,58779	0,30902	0,00000
Sampled value	0,00000	0,30469	0,58594	0,80469	0,94531	0,99219	0,94531	0,80469	0,58594	0,30469	0,00000
Error	0,00%	0,43%	0,19%	0,43%	0,57%	0,78%	0,57%	0,43%	0,19%	0,43%	0,00%

Sample no.	11	12	13	14	15	16	17	18	19	20
Actual value	-0,30902	-0,58779	-0,80902	-0,95106	-1,00000	-0,95106	-0,80902	-0,58779	-0,30902	0,00000
Sampled value	-0,31250	-0,59375	-0,81250	-0,95313	-1,00000	-0,95313	-0,81250	-0,59375	-0,31250	0,00000
Error	0,35%	0,60%	0,35%	0,21%	0,00%	0,21%	0,35%	0,60%	0,35%	0,00%

Table 3.1. Errors at conversion from analog to digital samples.

3.2. Filters: Finite Impulse Response and Infinite Impulse Response

Signal filtering is one of the main applications of signal processing [[7], p.185]. Filters are used for many purposes, but, in audio signal processing, they are mostly used to achieve a desired frequency response, the most basic being low-pass (which only let low frequencies pass), high-pass (same for high frequencies), band-pass and band-stop filters. Low- and high-pass filters are widely used in speaker cabinets to separate the input signal to each of the drivers, depending on drivers' frequency response. An example for use of band-pass filters is in radio communication, to isolate the required frequency band (representing a radio channel) from the others.

Depending on the considered audio signal, there are two types of filters: analog (electronic) and digital. Electronic filters can range from simple circuits, made just from simple passive components: resistors, inductors and capacitors, to complex ones, including along the usual passive components, active components: transistors, amplifiers, operational amplifiers and others.

The digital signal processing systems use samples of input signals, which constitute series of numbers. The result may be also series of numbers, to be used as output signals [[7], p. 239]. Each sample of the output signal is computed as a weighted sum of the previous few input and output samples. This weighted sum is also known as a convolution. There are two primary types of digital filters: FIR (Finite Impulse Response) and IIR (Infinite Impulse Response). The impulse response of a system is its output when presented with a brief input signal, called an impulse. Applying a filter to a signal will alter each frequency's magnitude and phase according to the filter's impulse response. What's left is to design the filter knowing the desired response, which will be detailed in the next section.

When using a FIR filter, each output sample is computed from the previous input samples and a fixed set of coefficients, the weights. The weights are associated to input samples based on each sample's delay (how far it is from the most recent input sample). Input samples, ordered by delay, are stored in a so-called delay line. The resulted sample is calculated by a set of multiply-accumulate operations: each is input sample is multiplied with its weight, and summed to the output sample:

$$\begin{aligned} y[n] &= b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N], \\ &= \sum_{i=0}^{N} b_i \cdot x[n-i] \end{aligned} \tag{3.1}$$

where:

• $x[n]$ is the input signal,

• $y[n]$ is the output signal,

• N is the filter order and the number of taps. A "tap" is simply a coefficient-delay pair [8]; an N th-order filter needs N previous input samples and has $(N+1)$ terms on the right-hand side,

• b is the set of the coefficients; b_i is the weight associated to the $(n-i)^{\text{th}}$ input sample.

For an FIR filter, the filter coefficients are, by definition, the impulse response of the filter [8]. The impulse response is finite (when the input samples become 0, the output will eventually become 0) because there is no feedback in the FIR. A lack of feedback guarantees that the impulse response will be finite [9].

The other class of digital filters is IIR filters which, unlike FIR filters, use feedback, i.e. samples already computed by the filter are used in the next iterations. Each output sample is computed from the previous input sample (feedforward), but also from the previous output samples (feedback). Like FIR filters, each sample has an associated weight, based on the sample's delay, so there are needed two sets of coefficients: feedforward and feedback. The formula is as following:

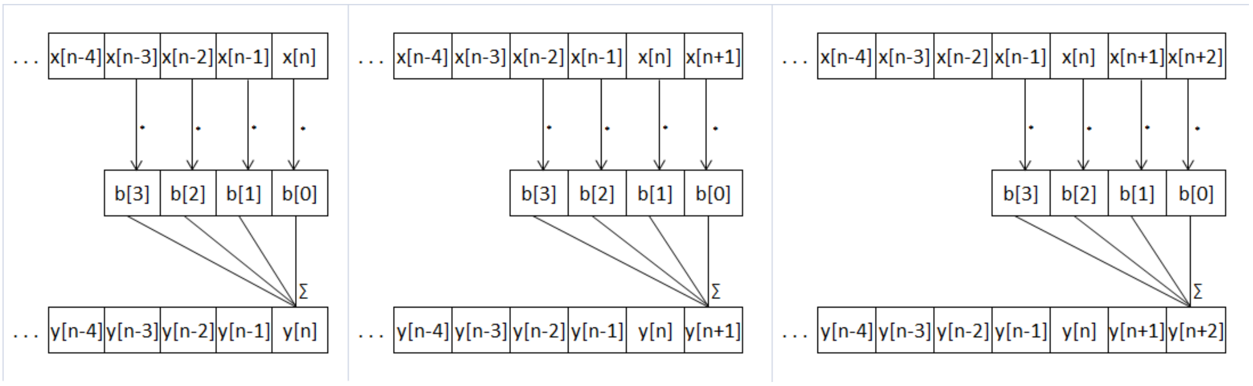
$$\begin{aligned} y[n] &= \frac{1}{a_0}\begin{pmatrix} b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N] \\ -a_1 y[n-1] - a_2 y[n-2] - \cdots - a_P y[n-P] \end{pmatrix}, \\ &= \frac{1}{a_0}\left(\sum_{i=0}^{N} b_i \cdot x[n-i] - \sum_{i=1}^{P} a_i \cdot y[n-i] \right) \end{aligned} \tag{3.2}$$

In many digital signal processing applications, FIR filters are preferred over their IIR counterparts. The main advantages of the FIR filter designs over their IIR equivalents are the following [10]:

1. FIR filters with linear phase response (all frequencies are delayed by the same constant time amount) can easily be designed;
2. They are simple to implement (two nested loops, one for iterating the input samples, one for the coefficients);
3. FIR filters are always stable, i.e. given a bounded input signal, the output signal will also be bounded. If designed wrong, IIR filters can diverge.
4. They have desirable numeric properties. The use of finite-precision arithmetic in IIR filters can cause significant problems due to the use of feedback, but FIR filters can usually be implemented using fewer bits, and the designer does not have to worry about floating-point arithmetic errors [9];
5. Excellent design methods are available for various kinds of FIR filters with arbitrary specification [10].

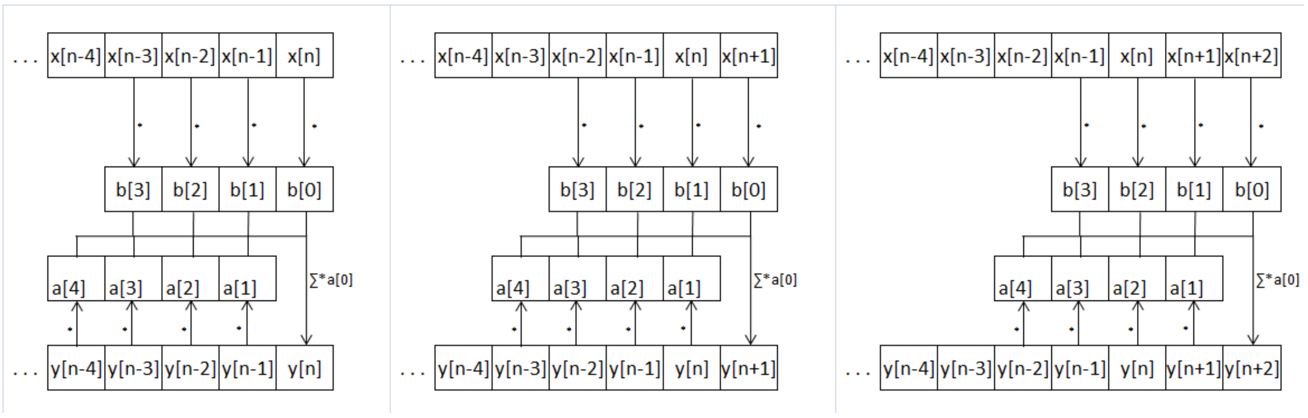
The main disadvantage of FIR filters is that they may require much more computational effort and memory than a comparable IIR counterpart.

Figures 3.4 and 3.5 give an example as how an FIR filter and IIR filter work along the input data:



. . . x[n-4] x[n-3] x[n-2] x[n-1] x[n]
b[3] b[2] b[1] b[0]
Σ
. . . y[n-4] y[n-3] y[n-2] y[n-1] y[n]
. . . x[n-4] x[n-3] x[n-2] x[n-1] x[n] x[n+1]
b[3] b[2] b[1] b[0]
Σ
. . . y[n-4] y[n-3] y[n-2] y[n-1] y[n] y[n+1]
. . . x[n-4] x[n-3] x[n-2] x[n-1] x[n] x[n+1] x[n+2]
b[3] b[2] b[1] b[0]
Σ
. . . y[n-4] y[n-3] y[n-2] y[n-1] y[n] y[n+1] y[n+2]

Fig. 3.4: Application of a digital FIR filter



. . . x[n-4] x[n-3] x[n-2] x[n-1] x[n]
b[3] b[2] b[1] b[0]
a[4] a[3] a[2] a[1]
Σ*a[0]
. . . y[n-4] y[n-3] y[n-2] y[n-1] y[n]
. . . x[n-4] x[n-3] x[n-2] x[n-1] x[n] x[n+1]
b[3] b[2] b[1] b[0]
a[4] a[3] a[2] a[1]
Σ*a[0]
. . . y[n-4] y[n-3] y[n-2] y[n-1] y[n] y[n+1]
. . . x[n-4] x[n-3] x[n-2] x[n-1] x[n] x[n+1] x[n+2]
b[3] b[2] b[1] b[0]
a[4] a[3] a[2] a[1]
Σ*a[0]
. . . y[n-4] y[n-3] y[n-2] y[n-1] y[n] y[n+1] y[n+2]

Fig. 3.5: Application of a digital IIR filter

3.3. Frequency domain and Fourier transforms

As it was previously described, an audio signal is represented as a variation of intensity over time (being it air pressure or electric voltage). The human auditory system picks up the eardrum's vibrations and transduces them into nerve impulses, which are then perceived in the brain as "sound". However, the brain does not interpret the sound by its pressure wave, but rather by its frequencies' amplitudes, phase and pitch. For example, an audible sine wave will be perceived not as the periodic function the sine function looks like, but as a pure tone, with a certain pitch and loudness. The function that gives the variation of wave's intensity is called **time-domain.** The function that gives the component frequencies is called **frequency-domain.**

In 1822, Fourier in his work on heat flow made a remarkable assertion that every function f(x) with period 2π can be represented by a trigonometric infinite series [[11], p. 1] of the form given by formulas 3.3, 3.4 and 3.5.

$$f(x) = \frac{1}{2}\,a_0 + \sum_{n=1}^{\infty}(a_n \cos nx + b_n \sin nx), \qquad (3.3)\ [[11], p.1]$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x) \cos nx \, dx, \qquad (3.4)\ [[11], p. 6]$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x) \sin nx \, dx. \qquad (3.5)\ [[11], p. 6]$$

An infinite series of this form is called a Fourier series [[11], p.1]. a_n and b_n are called the Fourier coefficients. Using the Fourier series, one can transform from time-domain to frequency domain and vice-versa. Fig. 3.6 shows the same sound, plotted in time-domain and in frequency-domain. In the time-domain plot, x-axis is time, y-axis is signal's amplitude. In the frequency-domain plot (spectrogram), x-axis is time (the signal was partitioned into chunks and the Fourier transform was run on each of them), y-axis is frequency, and z-axis (color intensity) is amplitude. A note played on flute has the fundamental considerably louder than the harmonics, so the sound wave generated looks close to a sine wave. The spectrogram clearly shows the harmonics.

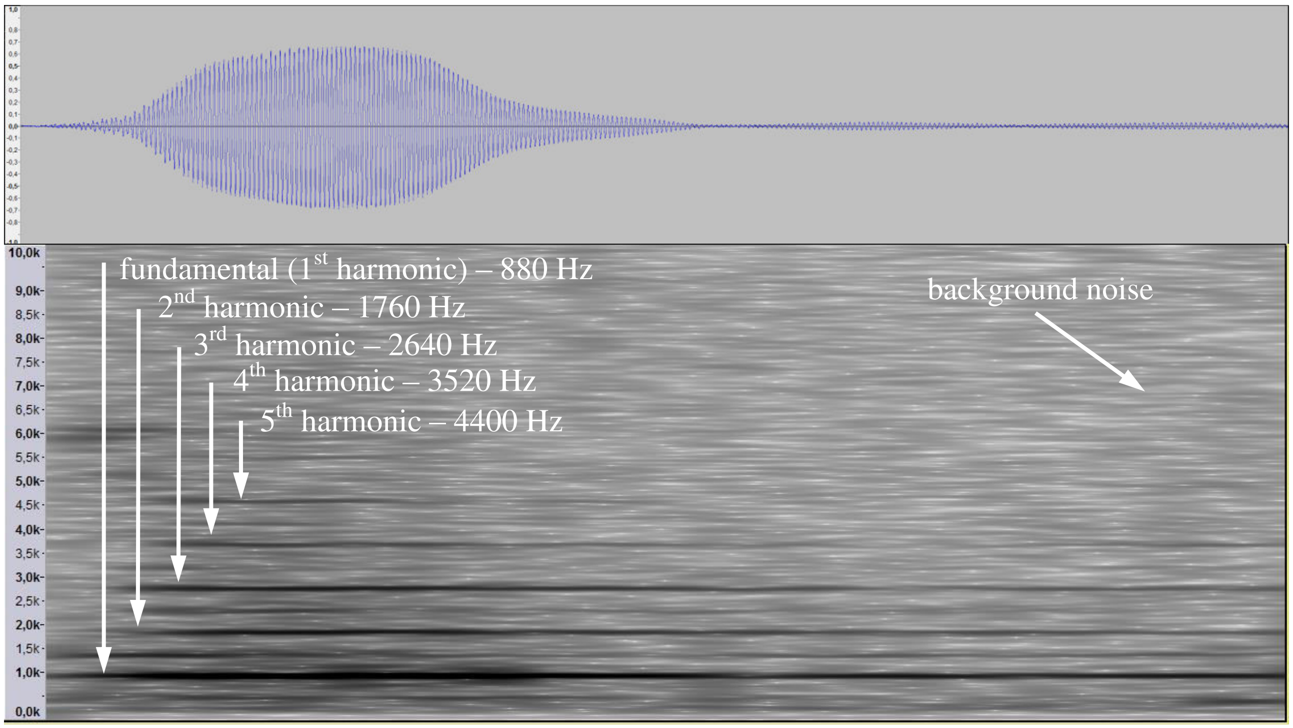


Fig. 3.6.: Note A_5 (880Hz) played on a flute. Top – plot of the time-domain function. Bottom – plot of the frequency-domain function.

As this work uses digital signal processing, it will not be working with continuous functions, but rather with discrete ones. A DFT (Discrete Fourier Transform) takes a discrete number of samples of a time wave and converts them into finite number of frequency components. The spectrum of the signal from Fig. 3.6 can be seen in Fig. 3.7. This time, the whole shown signal was input into the Fourier transform. The opposite of DFT is the IDFT (Inverse Discrete Fourier Transform), which transforms the frequency spectrum to a discrete number of samples. The equation for the Discrete Fourier Transform is:

$$F(n) = \sum_{k=0}^{N-1} x(k) e^{-i2k\pi\frac{n}{N}} \quad \text{for n = 0 ... N − 1,} \tag{3.6}$$

where F(n) is the amplitude of the frequency n, and N is the number of discrete samples taken. Note that:

$$e^{ix} = \cos x + i \sin x \quad (\text{Euler's identity}), \tag{3.7}$$

i.e. each frequency has a cosine and a sine component, each with its own amplitude.

The IDFT formula looks similar to the DFT:

$$f(n) = \frac{1}{N} \sum_{k=0}^{N-1} F(k) e^{+i2k\pi\frac{n}{N}}. \tag{3.8}$$

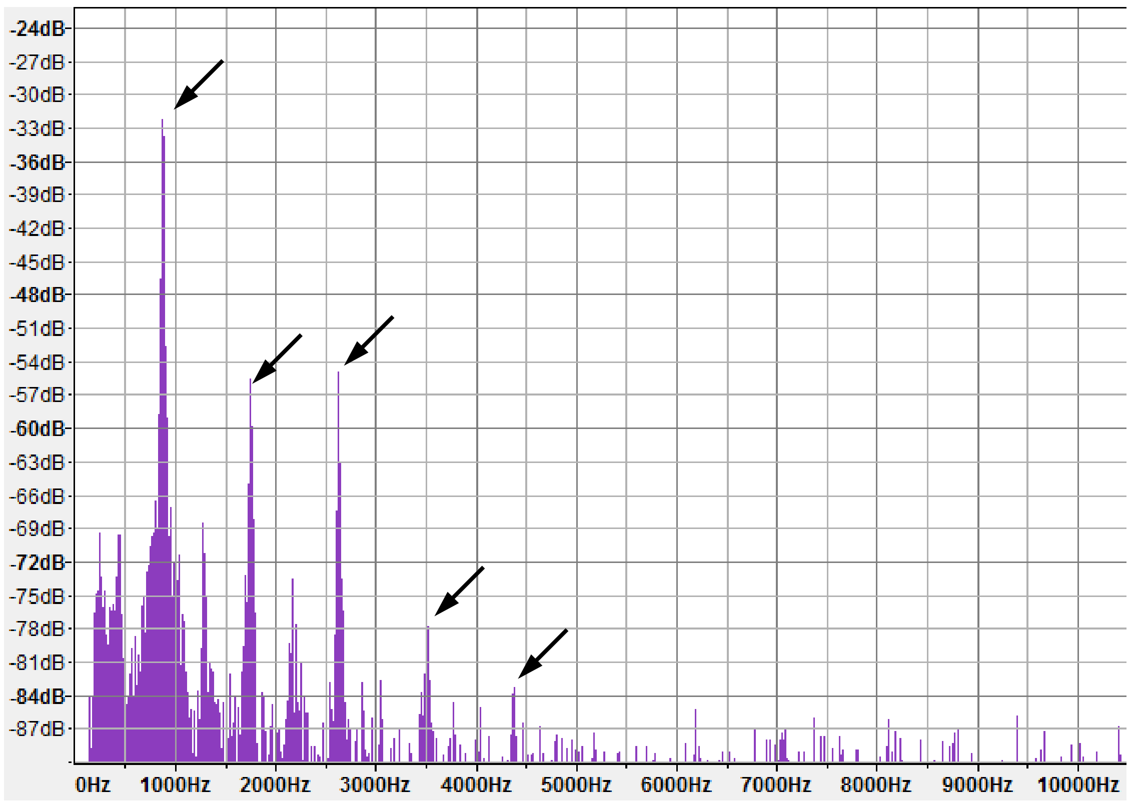


Fig. 3.7: Spectral plot of the same signal from Fig. 3.6. Arrows point to sound's harmonics, also visible in the spectrogram in Fig. 3.6.

3.4. Filter design and frequency equalization

In this work filters will be used, among others, to change the output signal's frequency response, i.e. to apply an equalization curve. When willing to create a FIR filter from a given equalization curve, the problem stands in calculating its coefficients based on the desired frequency and phase response.

For the design of FIR filters, there have been developed several methods:

1. Direct Calculation: In the case of some types of filters, such as high-pass and low-pass filters, their coefficients can be directly calculated from formulas. For example, the ideal low-pass filter follows the $sinc(x) = \frac{\sin(x)}{x}$ function, as shown in Fig. 3.8.
2. Parks-McClellan: The Parks-McClellan algorithm is an iterative algorithm used to design efficient and optimal FIR filters. It has become a standard FIR design method as it directly accepts filter specifications in terms of pass-band and stop-band frequencies, pass-band ripple, and stop-band attenuation [13].
3. Windowing: Using the property that, the DFT of the impulse response gives the filter's frequency response, the coefficients can be calculated by applying the IDFT on the wanted response. After that, the impulse response can be refined by applying a windowing function.

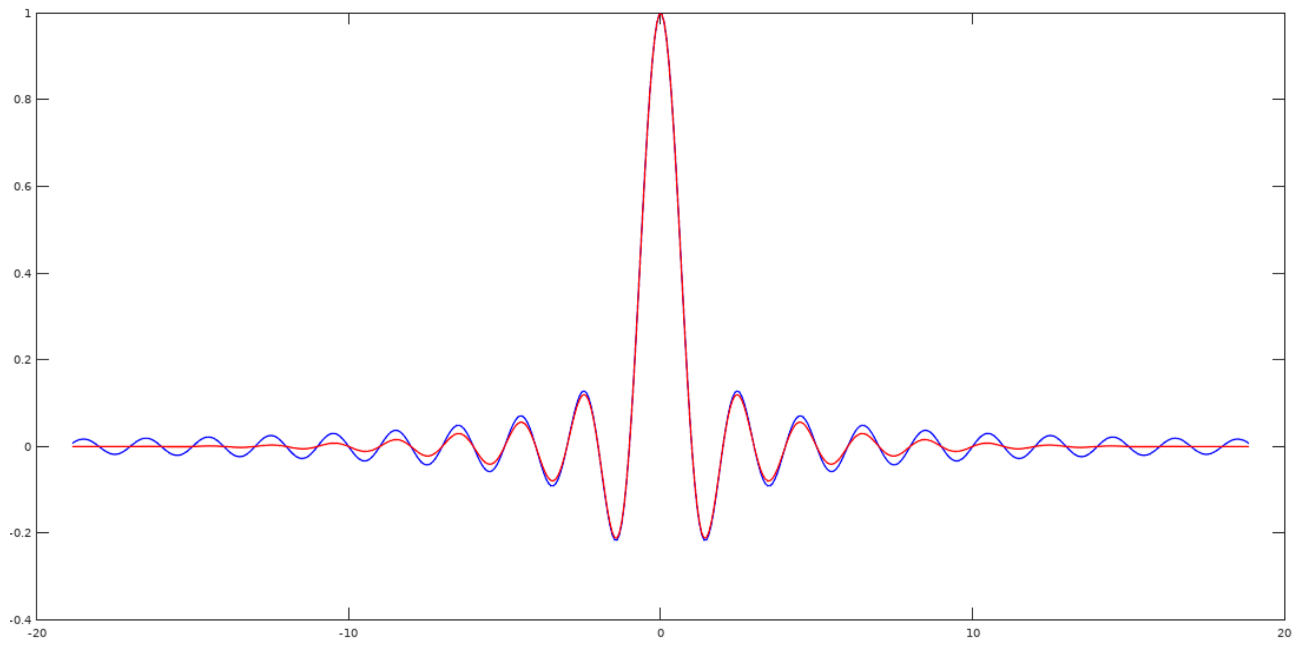


Fig. 3.8.: The ideal low-pass filter, the sinc function. With blue, the FIR coefficients following the sinc function. With red, the coefficients after applying a Blackman window

Chapter 4

Detecting and correcting distortion

4.1. Automated marking of distorted samples using neural networks

As stated previously, the purpose of this work is to create a software application that will automate the process of correcting various distortions in recordings on the analog format of vinyl records. The targeted distortions will be clicks, pops and some types of mistracking. The chosen approach in solving this problem consists of two steps: identifying the damaged portions and reconstructing them.

Identifying the damaged portions means, in this case, creating a set of non-overlapping intervals for a given audio sequence. Each of these intervals (further called "marking") represents a portion of the signal which is considered to be damaged and needs correction. Finding them is not a deterministic job, as there's no way to precisely differentiate between what is signal and what is noise. At the end, this whole marking process can be viewed as classifying each sample of the digital audio signal as either "marked" (damaged) or "unmarked" (not damaged).

A possible approach for classifying data is using artificial intelligence. The intelligent agent, i.e. the system that will solve this classification problem, has as input a sequence of samples of odd length, and as output – one of the two possible labels: "marked" or "not marked". In the input data, the sample in the middle is the one being classified, while the rest of them are used to help decide the output.

Before an intelligent agent can be used, it needs to be trained. A training set, a set of pre-labeled inputs, is used for both the training and validation. Validation is made using the trained system to label already classified inputs in order to calculate the system's accuracy.

Before a system can be trained, big enough training sets are needed. As the chosen number of inputs is quite big – 129(each central sample has 64 samples to the left and 64 to the right), a good training set must have at least a few tens of thousands of examples. Thus, generating the training sets by manually marking samples is a no-go. A method that proved to be very efficient and accurate was to use monaural records. By using a stereo cartridge to pick up the sound from a mono groove, one would expect the left and right channels to be identical. This is not the case when distortion occurs, as it is almost sure it will affect the channels differently. Using the assumption that every difference big enough between the channels is distortion (as it can be seen in Fig. 4.1), large and accurate datasets can be quickly produced. In fact, all the trainings done in the following examples use solely training sets generated this way. Table 4.1 lists the training sets used in this work: the record, its condition, dataset size, type of music. All sets use a 96 000 samples / second sampling rate.

All machine learning implementations used in this work are those provided by the scikit-learn API, a machine learning tool written in Python[19].

[19] **scikit-learn -** *Machine Learning in Python*: http://scikit-learn.org/stable/index.html

Dataset name	Record label/series	Record condition			Audio length	Music type
		Scratches	Mistracking	Groove wear		
Shostakovich	Мелодия НД-02243 track A2	moderate density, small intensities	no	moderate	05:18	Strings, Percussion, Brass, Winds
Chopin	Международная Книга 33Д-0008830(а)	high density, small intensities	moderate	no	03:35	Piano
Beethoven	The Classics Record Library MAQ 3333 track B4	moderate density, medium intensities	no	moderate	04:30	Strings
Andries	Electrecord EDE 03765 track B2	moderate density, small intensities	light	light	02:05	Guitar, vocals
Salesbury	Ryemuse RP 7016 track A1	moderate density, medium intensities	light	moderate	02:35	Organ
Dvorak	Musical Masterpiece Society MMS-121 track B2 excerpt	moderate density, small intensities	no	moderate	02:48	Strings, Percussion, Brass, Winds
Enescu	Electrecord ECD 23 track A	small density, small intensities	heavy	no	11:34	Strings, Percussion, Winds

Table 4.1: The datasets used in this work for intelligent agent training

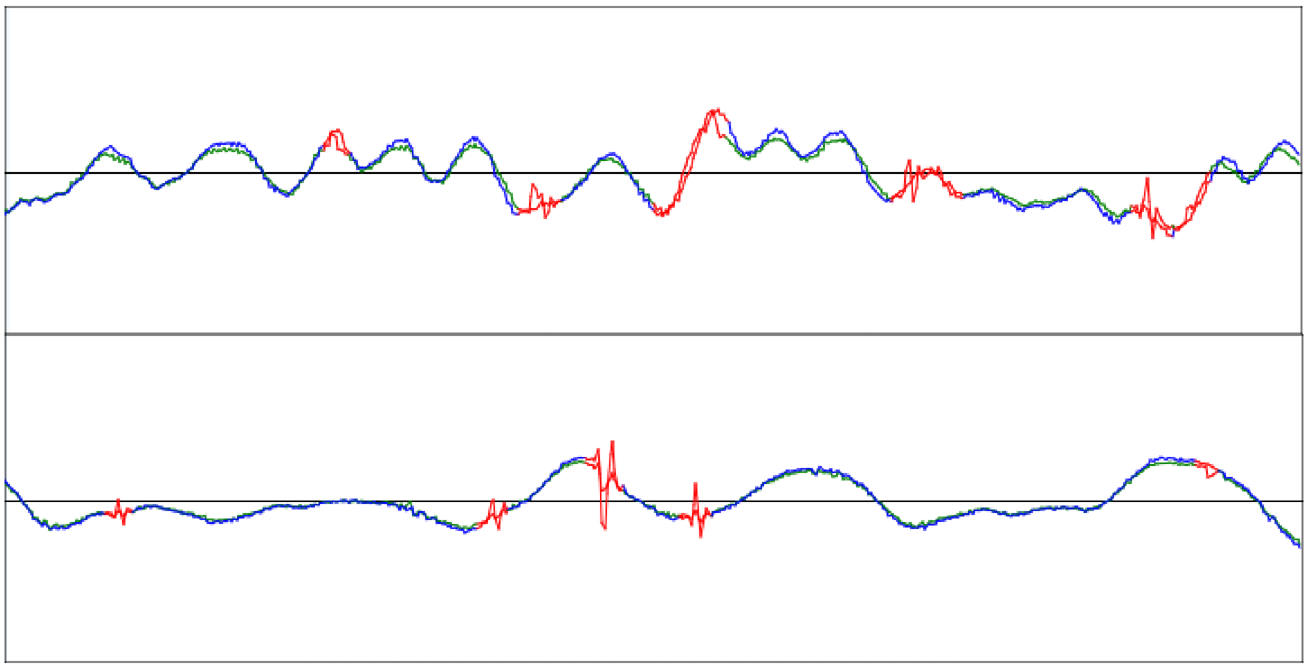


Fig. 4.1: Automated marking on mono recordings. With green – left channel, blue – right channel, red – marked damage. Datasets: "Beethoven" (top), "Chopin" (bottom)

There are many intelligent classification methods. Table 4.2 shows a comparison between various classifiers and their precision. The training dataset used for this example is a subset of length 20 000 of the "Shostakovich" dataset. It can be seen that KNN and CART were the most accurate, but they were discarded as viable options due to the very high training times for the relatively small training set.

Classifier	Accuracy	Training time (s)
Logistic Regression (LR)	0.579	8.62
Linear Discriminant Analysis (LDA)	0.572	2.91
K-Neighbors Classifier (KNN)	0.921	90.25
Decision Tree Classifier (CART)	0.838	45.28
Gaussian Naive Bayes (GNB)	0.711	0.70
C-Support Vector Classification (SVC)	?	> 300, timeout

Table 4.2: Performance of various classification methods

Much better performances were obtained by using Artificial Neural Networks. Table 4.3 lists ANN training results on various combinations of datasets, signal source and pre-processing, input sizes and ANN parameters. The possible signal sources are the following:

- direct pick-up – the current produced by the cartridge, passed through the RIAA stage
- pre RIAA – the current produced by the cartridge, without the RIAA equalization (simulated by applying the inverse RIAA equalization over the direct pick-up signal)
- groove modulation – the physical movement of the stylus as it follows the groove (simulated by discretely integrating over the pre RIAA signal, as to reverse the effects of Equation 1.1)

As for signal pre-processing, a high-pass filter may or not be applied. Its purpose is the removal of the lower frequencies, which are often not affected by distortion and thus might be redundant for the intelligent system.

Dataset(s)	Set size	Signal source	Preprocessing	Scaling	Layers	Changes from the default NN configuration [20]		Precision		
								average	class 0 (unmarked)	class 1 (marked)
Chopin + Shostakovich	74000	direct pick-up	4500Hz HighPass	no	(129, 64, 16, 1)	-	-	86%	-	-
					(129, 64, 32, 1)	-	-	86%	-	-
						logistic activation function	-	79%	-	-
						tanh activation function	-	84%	-	-
						identity activation function	-	59%	-	-
				yes	(129, 64, 32, 1)	-	-	86%	-	-
						tolerance = 1e-5	-	86%	-	-
							training set shuffled	86%	-	-
					(129, 64, 1)	-	training set shuffled	-	-	88%
						tolerance = 1e-5		-	79%	89%
Shostakovich	53000	direct pick-up	-	yes	(129, 64, 32, 1)	tolerance = 1e-5	training set shuffled	-	85%	85%
					(129, 65, 1)			-	85%	85%
					(257, 128, 64, 1)			-	88%	86%
					(65, 32, 1)			-	82%	86%
improved the datasets after this point										
Shostakovich	59116	direct pick-up	-	yes	(129, 64, 32, 1)	tolerance = 1e-5	training set shuffled	-	95%	93%
			4500Hz HighPass					-	85%	86%
		pre RIAA	-					-	95%	92%
			4500Hz HighPass					-	83%	84%
		groove mod.	-					-	93%	91%
			4500Hz HighPass					-	90%	87%
Chopin	93388	direct pick-up	-	yes	(129, 64, 32, 1)	tolerance = 1e-5	training set shuffled	-	82%	88%
			4500Hz HighPass					-	90%	89%
		pre RIAA	-					-	91%	92%
			4500Hz HighPass					-	90%	90%
		groove mod.	-					-	65%	68%
			4500Hz HighPass					-	90%	90%

Table 4.3: Results of different NN configurations

[20] The default NN configuration is detailed here: http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Based on the results presented in Table 4.3, it has been decided that a good combination of NN configuration and input data would be the following: pre RIAA signal source, no high-pass, (129, 64, 32, 1) layer configuration, standard scaling, shuffling on training, tolerance = 0.00001.

As the NN was designed to classify only one sample at a time (the one in the middle of the inputs), the marking of damaged regions on an audio signal will be done as following: for each sample of the audio signal, a set of samples is chosen such that its length matches the NN's number of inputs and the sample to be marked is in the middle. The outputs of the intelligent agent are then stored in a suitable data structure.

4.2. Extrapolation and linear prediction

Having marked the bad sections, the remaining step is to reconstruct the samples in those regions. The idea is simple: the damaged values inside a marked region must be reconstructed by using the good values to the left and right sides of the marking.

One of the tried methods was using Lagrange interpolation through the good samples and then replacing the damaged values with the interpolated ones. The results were simply disastrous with the Lagrange polynomial greatly diverging, as it can be seen in Figs 4.2 and 4.3.

Another method for extrapolating values is the "auto-regressive moving average", also known as Linear Predicting Coding. In this model, each sample is assumed to be a linear combination of previous samples[21]. This is basically the same as in a IIR filter where only feedback is present (representing the weights of the linear combination mentioned above), as the feedforward set of coefficients is { 0 } (as the current sample value is ignored). Those feedback coefficients still have to be computed to be as accurate as possible, so the prediction error is as small as possible. One of the algorithms that deal with this is "Burg's method", an algorithm which will be discussed soon.

The results of predicting samples with Burg's method can be seen in Fig. 4.4. Samples were extrapolated both from left and right, from the undamaged samples. After applying a smooth transition between the two extrapolated signals, the new signal (Fig 4.5) looks way better than the original (Fig 4.2).

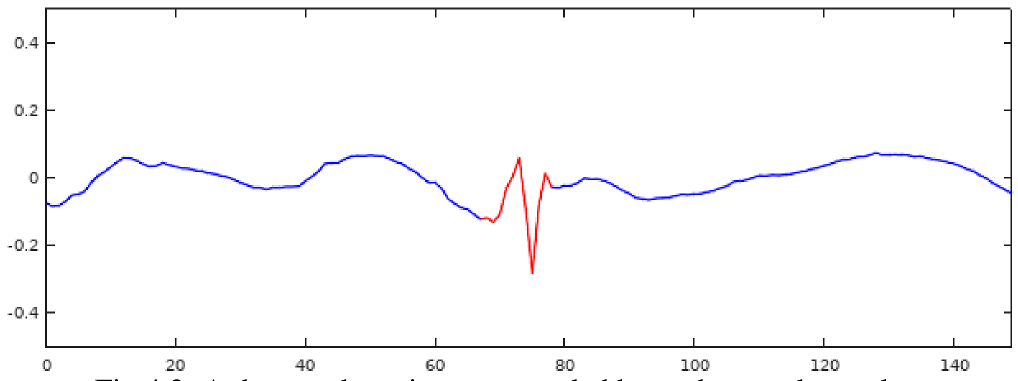


Fig 4.2: A damaged portion, surrounded by undamaged samples

[21] Signal Processing Stack Exchange: How do I extrapolate a 1D signal? – https://dsp.stackexchange.com/questions/101/how-do-i-extrapolate-a-1d-signal (Retrieved 10.11.2017)

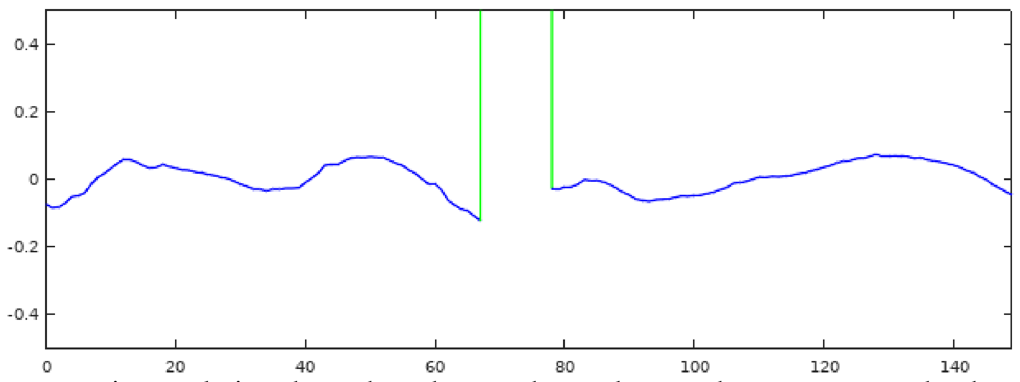


Fig 4.3: Lagrange interpolation through undamaged samples, used to reconstruct the damaged part

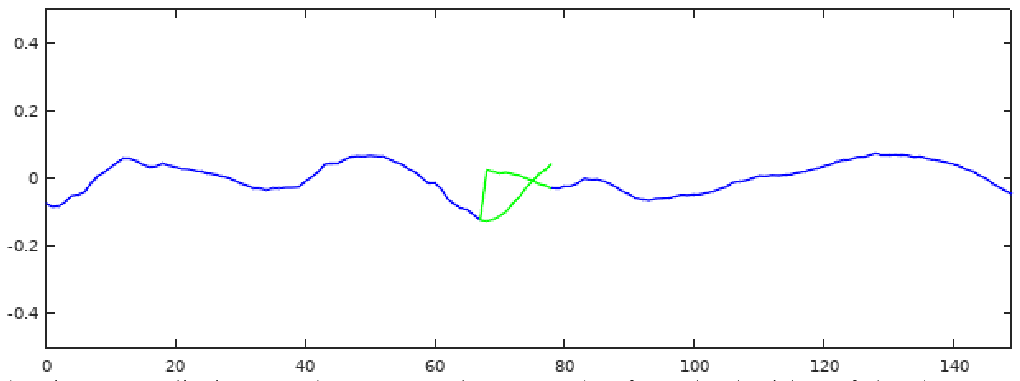


Fig. 4.4: Linear prediction used to extrapolate samples from both sides of the damaged section

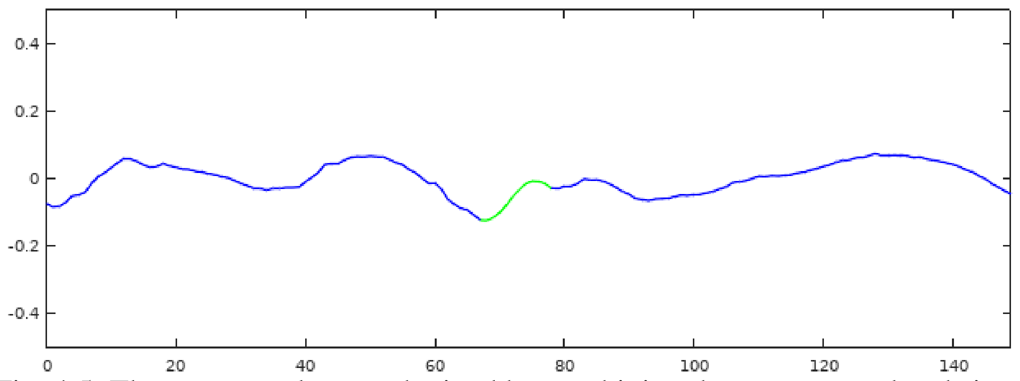


Fig. 4.5: The new samples are obtained by combining the two extrapolated signals

4.3. Burg's method for calculating LP coefficients

Given a discrete set of N + 1 original values $(x_n)_{n\in[0,N]}$, k coefficients $(a_n)_{n\in[1,k]}$ are used to approximate the original values by $y_n = -\sum_{i=1}^{k} a_i x_{n-i}$ for what is called the forward linear prediction, and by $z_n = -\sum_{i=1}^{k} a_i x_{n+i}$ for what is called the backward linear prediction. Note that y_n is only defined for $n \in [k, N]$ and z_n is only defined for $n \in [0, N-k]$. [14]

Burg's method of computing (a_n) is based on the Levinson-Durbin recursion, in which the coefficients are stored in a vector $A_k = [1 \quad a_1 \quad a_2 \quad \cdots \quad a_k]^T$ and an inverted order vector $V_k = [0 \quad a_k \quad \cdots \quad a_2 \quad a_1 \quad 1]^T$. The recursion formula is given in Eq. 4.1.

$$A_{k+1} = A_k + \mu V_k \qquad \text{(Eq. 4.1)}$$

Burg changed the way μ is computed, as to minimize to sum of $F_k + B_k$, with F_k and B_k defined in Eq. 4.2 and Eq. 4.3. [14]

$$F_k = \sum_{n=k}^{N} (x_n - y_n)^2 \qquad \text{(Eq. 4.2)}$$

$$B_k = \sum_{n=0}^{N-k} (x_n - z_n)^2 \qquad \text{(Eq. 4.3)}$$

The formula for computing μ is given in Eq. 4.4. For those who would like to know more details about how this equality was deduced, or just want to go deeper in Burg's algorithm in general, please refer to [14].

$$\mu = \frac{-2\sum_{n=0}^{N-k-1} f_k(n+k+1) b_k(n)}{\sum_{n=k+1}^{N} f_k(n)^2 + \sum_{n=0}^{N-k-1} b_k(n)^2} \qquad \text{(Eq. 4.4)}$$

where $a_0 = 1$ and $f_k(n)$ and $b_k(n)$ are defined in Eq. 4.5 and 4.6.

$$f_k(n) = \sum_{i=0}^{k} a_i x_{n-i} \qquad \text{(Eq. 4.5)}$$

$$b_k(n) = \sum_{i=0}^{k} a_i x_{n+i} \qquad \text{(Eq. 4.6)}$$

The version of the algorithm used in this work consists of the following steps [14]:

1. Choose m, the number of wanted coefficients
2. Initialize $A_0 = [\,1\,]$
3. Using Eq. 4.4 and Eq. 4.5, initialize all $f_0(n) = b_0(n) = x_n$
4. For k from 0 to m – 1
   a. Calculate μ using Eq. 4.6
   b. Update (a_n) using $a'_n = a_n + \mu a_{k+1-n}$ and defining $a_0 = 1$
   c. Update $\left(f_{k+1}(n)\right)_{n \in [k+1,N]}$ using $f_{k+1}(n) = f_k(n) + \mu b_k(n-k-1)$
   d. Update $\left(b_{k+1}(n)\right)_{n \in [0,N-k-1]}$ using $b_{k+1} = b_k + \mu f_k(n+k+1)$

4.4. Repairing the distorted sample intervals

Knowing how to compute the linear prediction coefficients, all that's left is how to use it in the context of our problem: reconstructing audio data. The main idea was already presented:

1. Select two sets of $\alpha \cdot N$ samples, one to the left (noted *LS*) and one to the right (noted *RS)* of the marked region. N is the length of the marking and α is a parameter of the repair function;
2. Compute the linear prediction coefficients for both sets of samples;
3. Forward predict N samples from the LS and backward predict N samples from the RS;
4. Overlap the two predicted sets using a pair of suitable windowing functions, for example the ones in Fig. 4.6.
5. Replace the samples in the marked interval with the new set of samples.

Experimentally, the best sounding results were obtained with relatively high values for α, like 8 and 16. These values are considered high because of the extra computational effort it takes to compute the coefficients relative to $\alpha=1$. Also, repair quality was significantly improved when only the high frequencies were reconstructed (as it can be seen in Fig. 4.7). In other words, the signal was split in two: a high-pass (with the cutoff frequency between 2-6 kHz) and a residue (the high-pass signal subtracted from the original signal). Because the distortion mostly occurs in the upper part of the spectrum, there is no need to repair the low frequencies. Also, because the high-pass signal is less complex (as it is made up of fewer frequency components), the prediction algorithm gives more satisfactory results. After the repairing is done in the high-pass, it is recombined with the residue signal to construct the final result.

Following this result, the process was generalized to allow signal decomposition and repair for more than one cutoff frequency. Having $n \geq 0$ cutoff frequencies, the signal is decomposed into $n+1$ frequency bands by consecutively applying carefully designed band-pass filters. For $n=0$, the decomposed signal is the same as the input signal. The lowest band is called the residue signal, as it is calculated by subtracting the sum of all the other bands from the original signal. The repair process repairs each band individually and then merges them to give the final result.

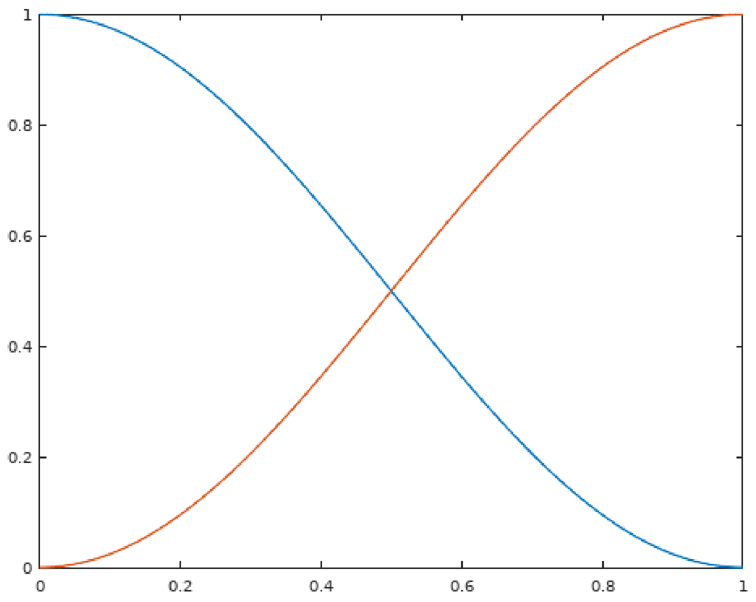


Fig. 4.6: Example of windowing functions for combining forward and backward predictions. Blue – for the forward predicted samples, Red – for the backward predicted samples

Though multi-band repair is generally better than all-pass repair, this is not always the case. In some clicks and crackles, which appear as a sudden spike in the signal, the band decompositions will result in a ripple to the left and right of the damaged area. This ripple will cause the repair to not properly attenuate the signal spike. For these cases, all-pass repair (i.e. repair on the original signal, without band decomposing) works better, as it can be seen in Fig. 4.8. In order to identify which repair method is the best for a given damaged portion, we must tell if the damage appears as a big sudden spike or not. To do this, the maximum signal amplitude is computed for the marked signal, a fragment to its left and one to its right. If the marked signal's amplitude is significantly greater than the other two, all-pass repair is used in the favor of multi-band repair.

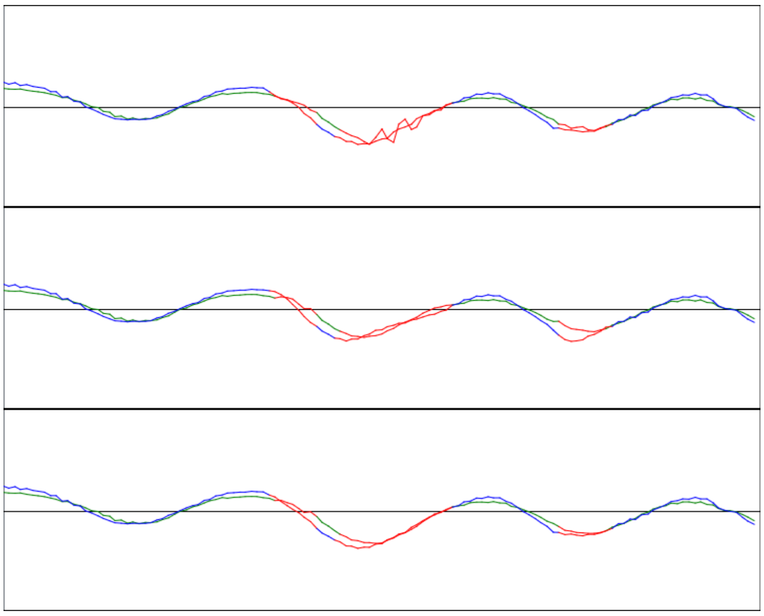


Fig. 4.7: Normal repair vs high-pass repair, with α=16. Top – the original stereo signal, middle – signal after normal repair, bottom – signal after applying the repairing process only on frequencies over 5kHz.

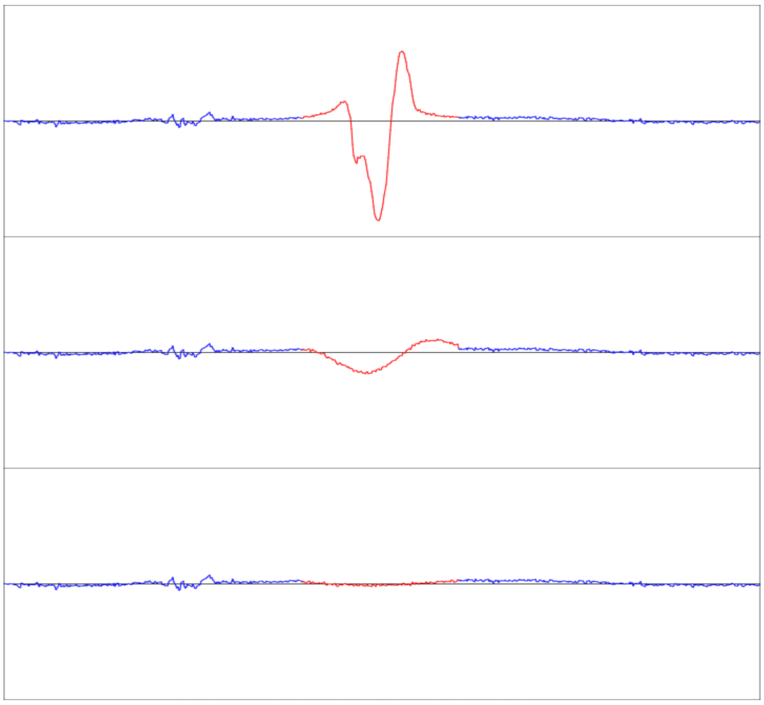


Fig. 4.8. A signal spike (top) and repairs made on it: 2000Hz cutoff, repair both high-pass and residue (middle), all-pass repair (down)

Chapter 5

The application

5.1. Requirements and specification

The application must provide the following features:

1. It must be able to read audio data from audio files. The required format types are WAV and AU files, which are both uncompressed formats. Complete format support is not required, but support for Linear Pulse Code Modulation encoding and arbitrary sample rates, channel numbers, byte depths and file lengths is mandatory;
2. It must use in-memory caching for the audio data, in order to reduce the number of read and write operations to the disk;
3. It has to be able to execute basic signal processing operations: applying arbitrary FIR and IIR filters on arbitrary segments of the audio signal;
4. The application must have version control on an audio signal processing project: starting from the original file, each modification shall create another project version. The original file will never be altered; all alteration must be stored in auxiliary files. By using the versions, undo and redo operations must be supported. Each version will depend on the previous ones, so writing operations done in the auxiliary files shall not affect those older versions;
5. It must give the possibility to export the current version of the project as a whole, continuous audio file, in one of the formats mentioned at point 1;
6. It has to be capable of applying an equalizer with a requested frequency response.
7. It must have means of repairing portions of signal, based on the adjacent values, by using a combination of forward and backward linear prediction;
8. The program should be able to store and manage “markings”, i.e. set of intervals that mark portions of the signal that is considered to be damaged and needs reconstruction, as well as to generate the set of “markings” for a given interval of audio signal;
9. It should offer the possibility of using the repair feature on all the “markings” inside a certain interval;
10. The application shall provide an easy to use and intuitive GUI.

Specifications:

1. Audio data: for storage (in memory), it will be stored into “Audio Samples Windows”: matrixes of floating-point values; each channel has a row assigned, each row contains the audio samples. Metadata such as the interval the window maps to the project, the number of channels, sample rate must be included;
2. Audio files: for the WAV and AU formats, first, metadata is be read from the header and store in memory. While reading the header, the program must check for any inconsistencies. After that, each read/write operation will randomly access the file, read/write the necessary bytes, convert them to/from floating-point values (samples) and output/input an “Audio Samples Window”;
3. Cache: The purpose of it is to minimize the accesses to the disk. Caching policy will be **Least recently used (LRU)**, and it will use pagination to even more reduce the number of disk accesses;
4. The coefficients for the linear prediction will be calculated using Burg’s method;
5. FIR filters will be designed from a given frequency response using the “windowing” method. Thus, the application must also be able to compute FFT and IFFT;
6. The program must implement and make use of an efficient data structure for a set of ordered non-overlapping intervals;
7. The application must have a well defined architecture that respects architectural design patterns. It must be easily extensible (adding of new modules), follow coding standards and have a well-developed exception treatment system.

5.2 Project architecture and design. Packages and classes

The application was written in Java, and consists of several packages and modules, each being made up of one or more java source code files. The project presents a stratified architectural style (Fig. 5.1) that goes from the GUI level down to low-level memory and file management modules.

Below is the list of packages in the application, as well as a short description of what functionalities each package has to offer:

- “Utils” package: contains the data types of “Interval”, “Complex” (as in complex number), “Pair”, the data structure of “Ordered Non-Overlapping Interval Set”, as well as helper various helper functions.
- “GUI” package: contains the code related to the graphic user interface windows and functionalities.
- “ProjectManager” package: contains a class filled with various project properties and parameterization data, and also a class that manages everything about the audio project, from applying effects and loading or exporting files to thread safety mechanisms.
- “MarkerFile” package: contains a class that deals with reading and writing sets of markings from files, as well as adding or deleting markings from the set after it has been read.
- “Exceptions” package: contains the custom defined exception type that’s widely used in the project.
- “AudioDataSource” package: contains all the means to access and work with audio data. The package exposes the interface “IAudioDataSource”, which all ADS derive from. There’s also the interface “IFileAudioDataSource”, which extends the “IAudioDataSource” by adding a method for getting the file’s path. Classes that implement the base interface are: “CachedAudioDataSource”, “AudioDataSourceVersion”, “SingleBlockADS”, “InMemoryADS“, WAVFileAudioSource” and “AUFileAudioSource”. The package also contains a “Cached_ADS_Manager”, which makes sure a file ADS does not have more than one cache on top of it, and the “AudioSamplesWindow” class.

- “SignalProcessing” package: this only gathers similar packages in a single place:
    - “Windowing” package, which offers methods of providing window functions and applying them on arrays
    - “LinearPrediction” package, which has capabilities of applying forward and backward linear prediction, given the coefficients. It also offers an implementation of Burg’s method for computing the LP coefficients.
    - “FunctionApproximation” package. This package offers interpolation and extrapolation methods based on either linear interpolation or Fourier interpolation.
    - “FourierTransforms” package offers methods for computing the Fast Fourier Transform and the Inverse Fast Fourier Transform.
    - “Filters” package hold together implementations for FIR and IIR filters. The class responsible for the FIR filter also offers methods of creating a FIR off a given frequency response, as well as creating frequency response curves based on {cutoff frequency, high-band gain/octave, low-band gain/octave} tuples.
    - “Effects” package. Here are classes that act as a bridge between signal processing effects listed above and the Audio Data Sources. Each class has an associated processing, and it is being applied gradually, on chunks, as to not store all the processed data in memory. The effects this package provides are: “FIR” and “IIR” filters, “Equalizer” (which is an offset kind of FIR), “Repair_One” (which restores an interval of samples using linear prediction) and “Multi_Band_Repair_Marked” (which applies the “Repair_One” effects on a set of intervals, and also offers the refinements presented in Chapter 4.4).

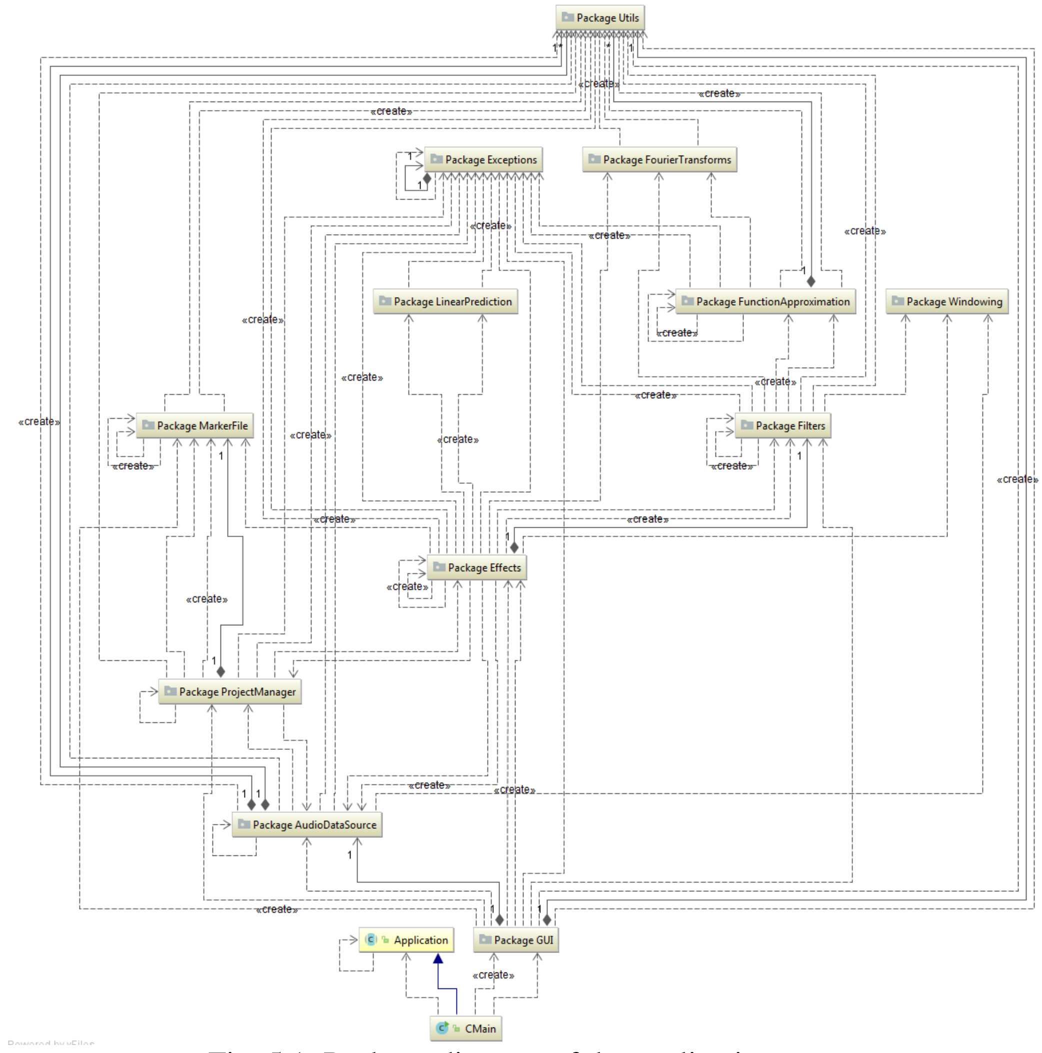


Fig. 5.1: Package diagram of the application

5.3 Audio Data Sources

5.3.1 File Storage. WAV and AU formats

An Audio Data Source (ADS) is an abstract place where audio data can be read from or written to. It also provides means of getting metadata like sample-rate, bit-depth, number of channels and others. An ADS can be stored in memory, in a file or in multiple files. A small continuous fragment of audio, called an Audio Samples Window (ASW), holds a number of samples, divided into channels. Each ASW is aware of the number of channels and samples it stores, as well as where it is located in the project.

ASWs are the basic form of transmitting audio data inside the application. ADSs return ASWs for "get" operations and require ASWs for "put" operations. The low-level signal processing algorithms use number arrays as input and output, but their high-level counterparts (the ones in the "Effects" package), which also take care of things like fragmenting the task into smaller pieces, mainly use ASWs.

The application will work starting from an already existent audio file, and after being done editing, the user would want to save the edited sound to a file on a persistent media. Thus, the application must be capable of reading and writing audio files.

Even if some file formats compress the audio data to take up less space, the processing of audio signal requires that reads and writes are made as efficient as possible, so only uncompressed formats are used in the editing process. The file types supported by this application are the WAV format and the AU format. Both are uncompressed, LPCM formats, and make use of very simple file structures. The only notable difference between the two formats is that AU has a simpler and non-redundant header.

The process of using a file as ADS is quite simple. First, read the header and keep the metadata in memory. Reading samples is done by reading bytes from a certain position in file, converting them to numerical values (samples), and returning them as arrays. Writing does the exact opposite: reads samples from arrays, converts them to bytes which are then written to the file in the right position. If after writing, the total number of samples is bigger than before, an update of the header may be necessary.

5.3.2 Caching

As it is well known in the industry, read/write operations on disk are significantly slower than in memory. To reduce the number of hard disk access operations, a method of caching audio data in memory is required. As well as allowing faster access to some portions of the project's audio signal, a cache helps to reduce the number of consecutive small-length reads/writes in neighbor sections of a file by condensing them into a single, larger-length operation.

In this application, the caching system is divided in two parts. One of them is a container that stores multiple ASWs, as well as a history of their usage (the cache policy is to remove the least recently used ASW). It has a limit on how much data it can store, and when trying to add a new cache entry when there is not enough free space, the operation will fail and require external intervention to release some of the cached data. The other component of the caching system is the "CachedAudioDataSource", which uses the cache container to implement the ADS interface. The CachedADS has an underlying data source for permanent storage when flushing the cache. By using fixed-size cache pages, it minimizes the number of read/write operations to the underlying file by redirecting write and read operations to the cache, the file being accessed only when flushing a cache page or when samples that are not cached are requested.

5.3.3 Version control on an audio project

While editing an audio project, multiple modifications can be made one after another, and the user will want to have the possibility of undo and redo actions. Each modification creates a new "version" of the audio project. Audio data can get big enough to become impractical to be kept all in memory. Raw audio data, at CD quality (16 bits, 2 channels, 44100 samples/second) takes up over 600 MB for an hour of recording, so having a copy of the whole project each time a change is made could take up a lot of HDD space.

To avoid unnecessarily using disk space, a scheme for linking each version of the project to its parent was thought. It uses auxiliary files and project-to-file mappings in order to only store the differences between a version and its parent. This is as well an Audio Data Source, and is called "Versioned Audio Data Source". Each step of the project's editing process is an "ADS Version". The algorithm that manages a Versioned ADS goes as following:

- Initialization of a Versioned ADS can be either from scratch (0 samples stored) or from an already existing audio file.
- All the versions present are kept in an ordered list; each version has a parent, excepting the first version.
- There is a counter that points to the current version used by the project. Moving the cursor up and down the version hierarchy basically implements the undo-redo system.
- Read/Writes to the Versioned ADS are redirected to the version currently being pointed to. Then, for "get" operations, the version uses its project-to-file mapping to read the required data from files, and for "put" operations – it alters the mapping accordingly.
- A version can have up to one child. When creating a version as a child of another version, all of its siblings are deleted recursively. The new child then copies its parent's project-to-file mapping.

The mentioned auxiliary files, which store all the audio data modified since the first version, and project-to-file mappings are used at "ADS Version" level. There's also a component that manages the life-cycle of the auxiliary files. It keeps, for each temporary file, a list of ADS Versions that refer that file. The functionalities it provides are creating a new temporary file and mapping a portion of the project's audio data to a portion of a file. When a file is no longer referred by any ADS Version, it can be safely deleted (and it will automatically be).

5.4. Applying effects on data sources

The application of an effect takes an ADS as the source for audio samples and an ADS as the destination for the processed audio samples, but it is sometimes permitted for both to be the same instance. When signal processing over an ADS, the amount of data that has to be processed in one go can get as large as hundreds or thousands of MB, so bringing it all into memory for processing is a silly thing to do. It must be split into smaller chunks, which will be processed consecutively, one at a time. But sometimes this task is not as easy as it seems because saving a processed chunk to disk can affect the processing of the next chunk to have the effect applied on.

5.4.1. FIR and IIR Filters. Equalizers

The application of FIR and IIR filters over an array of samples was previously explained in Chapter 3.2. To overcome the problem of splitting the task in multiple chunks, the FIR implementation applies the filter from right to left (backwards in time). This guarantees that processed samples will not affect processing the remaining data, as the FIR filter only uses previous signal values.

The IIR filter though only works from left to right, because of its feedback. As a result, some of the already processed samples, needed by the feedback, must be kept when switching from a chunk to the next one.

The equalizer effect is basically an odd-length FIR filter with the coefficients centered on the current sample (Fig. 5.2). Equalizers change the frequency response of the signal without inducing the delay normal FIR filters do. Because samples from both left and right are needed, a circular buffer is used to temporary store the processed samples until they won't get in the way of applying the filter anymore. FIR coefficients for equalization can be generated from a given frequency response by using the inverse Fourier transform and a windowing function, as described in Ch. 3.4.

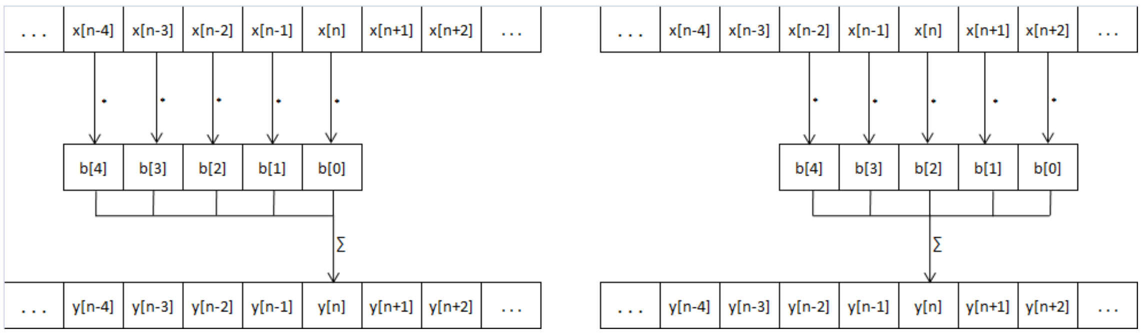


Fig. 5.2: Applying an FIR filter (left) vs an Equalizer (right)

5.4.2. Repair using linear prediction

There's not much to be said about this effect. The used algorithms have already been presented in Chapters 4.3 and 4.4, and are used exactly as they are. The only notable aspect is that the ratio between the repaired region's length and the side lengths (number of LP coefficients) can be adjusted, as a mean to choose the prediction's accuracy.

5.4.3. Finding the distorted regions in an audio recording. Repairing

Automatically creating the list of markings for an audio file is done based on the method described in Chapter 4.1. The application makes use of the Python scikit-learn implementation of a multi-layer neural network by using a python script to create the markings. The script is run as a parent process of the Java application, and communication between them is made through pipes by overwriting the script's stdin and stdout.

The script loads the NN from a file and then, in a loop, waits for input as an integer *n* and an array of *n* samples, and then outputs an integer *n-128* and an array of *n-128* probabilities (in [0,1]) for the samples to be damaged. The difference in the number of outputs and inputs is caused by the fact that first and last 64 samples are not being categorized and only serve as non-central input for the NN.

After receiving all the probabilities (example of generated probabilities in Fig. 5.3), the main application runs them thorough a threshold to give each sample a final label: marked (as damaged) or not. The set of markings is then used for the repairing process, in which the samples of each marking (a continuous interval of marked samples) is replaced with samples computed using linear prediction. As detailed in Chapter 4.4, better results were obtained when reconstruction occurred only in a frequency band of the signal. But for some types of damage, direct reconstruction, without signal decomposition, gave better results. The repair method implemented gives the freedom of choosing the cutoff frequencies (if any), whether to repair the residue signal as well and whether to detect and differently treat spikes.

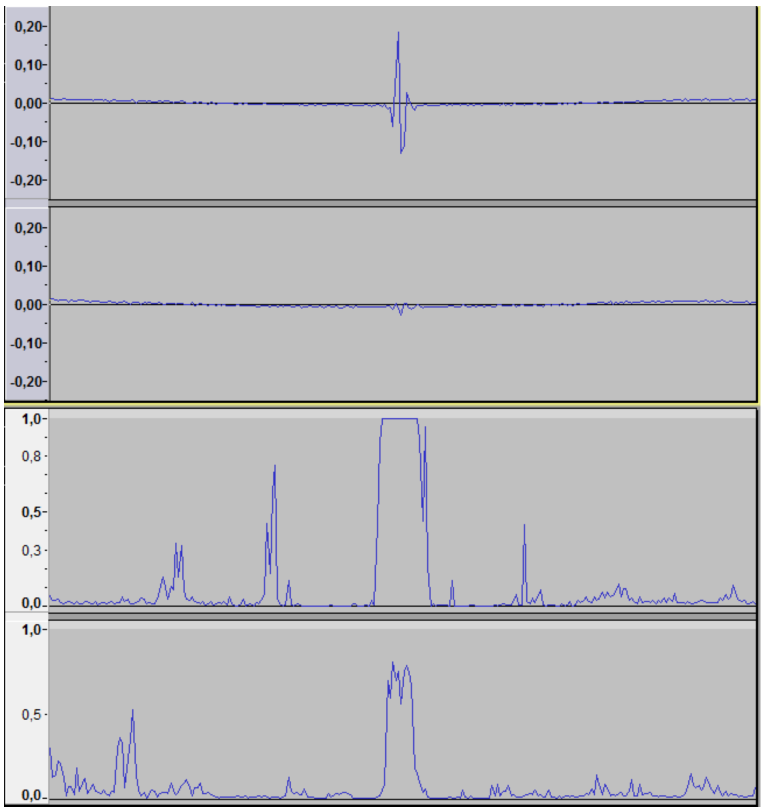


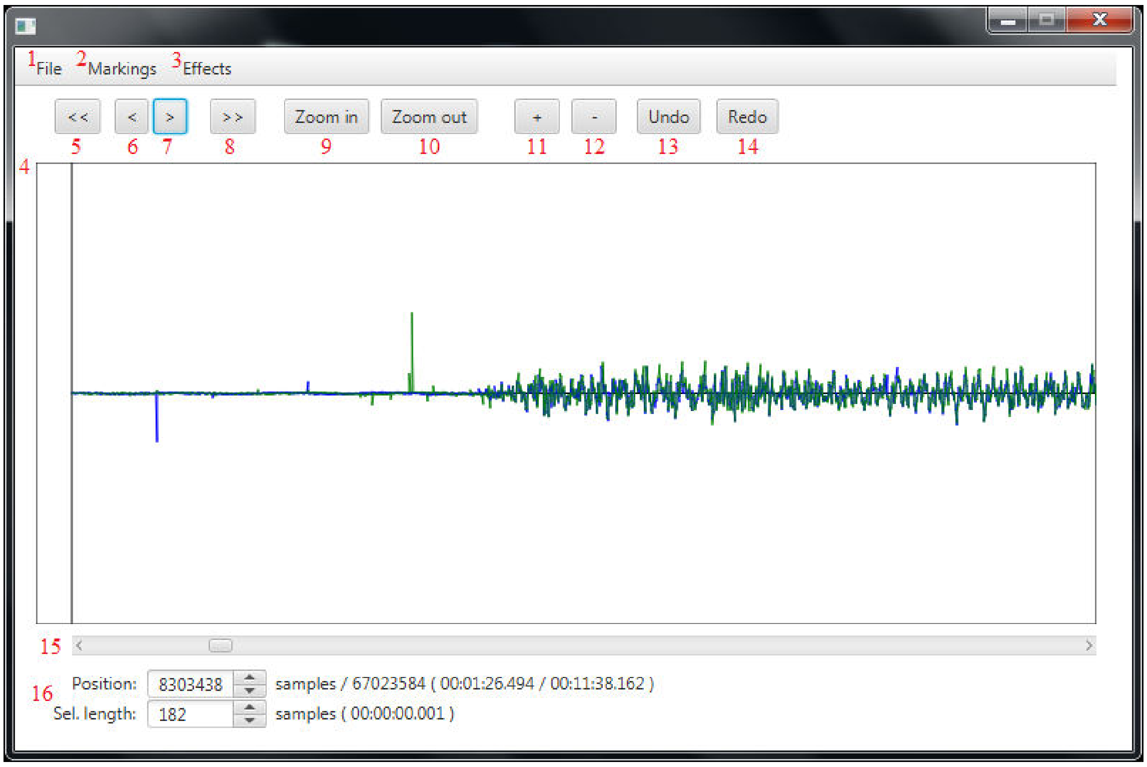
Fig. 5.3: Example of probabilities (bottom two) outputted by the NN for a signal (top two, L-R channels)

5.5. How to use the application

The application is written in Java, so an up-to-date installation of the Java Runtime Environment is required. Also, as the neural network implementation is written in python using the scikit-learn tool, python 2.7 with scikit-learn and all its dependencies must be installed on the system and be visible from the environment path.

After starting the program, the main window, as in Fig. 5.4, shows up. All the functionalities can be accessed from this main window. As it can be seen in Fig. 5.4, there are 16 controls accessible from the main window:

1. The “File” menu, which contains the audio file load and save operations.
2. The “Markings” menu, which contains the marking-related functionalities.
3. The “Effects” menu, which stores various signal processing facilities.
4. The display window, where the audio signal can be seen and selected or slid to left and right.
5. A button which moves the displayed audio interval to the left by a window’s width.
6. A button which moves the displayed audio interval to the left by a pixel.
7. A button which moves the displayed audio interval to the right by a pixel.
8. A button which moves the displayed audio interval to the right by a window’s width.
9. A button which reduces the number of displayed samples by half.
10. A button which increases the number of displayed samples by two.
11. A button which vertically zooms-in the displayed signal.
12. A button which vertically zooms-out the displayed signal.
13. The “Undo” functionality.
14. The “Redo” functionality.
15. A time slider that sets and shows the current displayed interval’s position in the project.
16. A place where the current selection’s position and length is shown in samples and seconds.



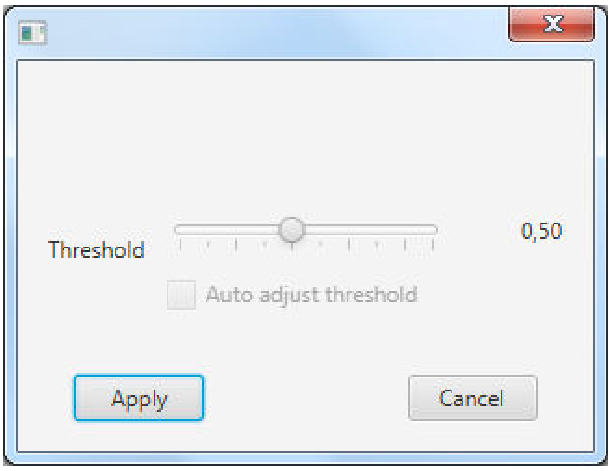
Position: 8303438 samples / 67023584 (00:01:26.494 / 00:11:38.162)
Sel. length: 182 samples (00:00:00.001)

Fig. 5.4: The application's main window

The functionalities in the "File" menu are: "Open file", which opens an audio file and discards the current project, "Export project" and "Export selection:, which save the whole data in the project and, respectively, the selected interval to an audio file on disk, and "Close", which closes the application.

In the "Markings" menu, all the necessary operations related to markings are present: markings can be added from file or exported to file and the set of markings can be cleared. Manual marking is also possible by first selecting some samples out of the displayed ones and then choosing either the "Mark selection" or "Unmark selection" item. There's also the automated marking option under the name "Generate markings", that is the damage detection part of this work. Clicking this opens a window (Fig. 5.5) where you can select the detection threshold. Marked sample intervals appear in the display window as red lines instead of the normal green or blue.

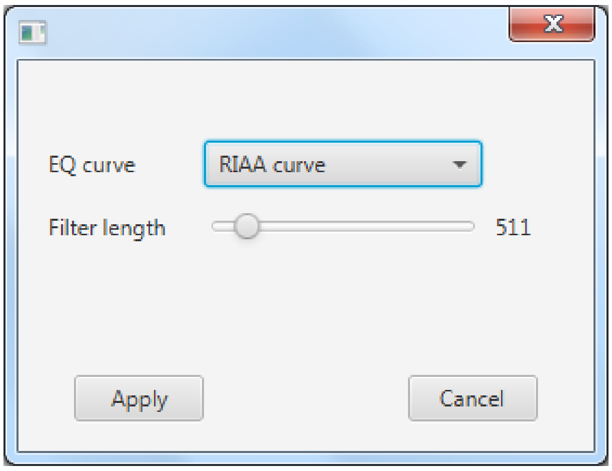
The "Effects" menu provides some basic signal processing features such as the "Amplify" effect, but also "Discrete derivation" and "Discrete integration", which can be used to transform the signal to and from the "groove modulation" form described in Ch. 4.1. The "Equalizer effect" (Fig 5.6) creates and applies a filter based on the requested frequency response. Last but very important is the "Repair selected markings", which does the damage repair part of the work. The windows that pops up (Fig 5.7) gives the user the opportunity to choose how the signal will be repaired: which frequencies will be used as cutoff for the multi-band repair, whether to repair the residue signal as well and whether to check for separate treatment of signal spikes or not. Also, the ratio between the number of LP coefficients and the marking's length can be selected under the label "LP coefficients".


Threshold

0,50

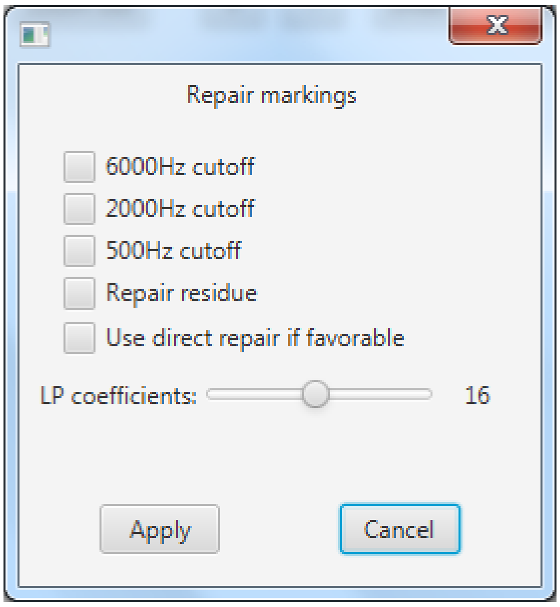
Auto adjust threshold

Fig. 5.5: The automated marking window


EQ curve: RIAA curve

Filter length: 511

Fig. 5.6: The “Equalizer” window


Repair markings

- 6000Hz cutoff
- 2000Hz cutoff
- 500Hz cutoff
- Repair residue
- Use direct repair if favorable

LP coefficients: 16

Fig. 5.7: The “Repair” window

Chapter 6

Conclusions and Future Work

This thesis approached the use of digital signal processing on audio recordings to improve their quality or change some of their characteristics. The goal was to implement a software product that would reduce the level of distortions occurring in recordings stored on vinyl records, distortions such as groove wear and surface scratches. In order to do that, we started with understanding how sound is stored on a record, how it's played back, and how distortion occurs. Then, in the second part, we tested a couple of other products which offer the functionality of attenuating the amount of damage in the recording: Audacity and Nero WaveEditor. The third part serves as an introduction to signal processing: time and frequency domains were defined and we learned how to transform a signal between those two domains. FIR and IIR filters were introduced as basic notions in signal processing theory and it was shown how they can be used to alter a signal's frequency response. Chapter 4 approached the distortion detection and removal problem. For detection, neural networks will be used, as for correction – linear prediction seems to be an adequate solution. The final part presents the practical part of the thesis: a Java application which, along the distortion removal functionality, also serves as a rudimentary audio editing program which can be easily extended in the future. The system's architecture was presented and also details about how each module or component works were given. The chapter dedicated to the application concludes with a small how-to-use guide.

A lot of personal contribution was involved in the design and implementation of this work. It began by analyzing how different types of distortion sound, occur and look, by both listening to worn records and inspecting their signal's waveform. After researching this subject on the internet, but also from information located on record inner sleeves, it became clearer what the causes of groove wear are. Then, the idea of finding and repairing damage came up, and we started researching ways of reconstructing data. For damage detection, the initial proposed solution was to use an intelligent agent, and after some performance tests we settled with a neural network. For the repair – linear prediction was the chosen solution, and we further improved the repair quality by splitting the signal in two or more frequency bands when repairing, as it gave much better results. A great part in this whole work constituted the semi-automatically markings done on mono recordings. More than 25 minutes of such markings have been created, and they served as relevant data for testing the repair process, as well as training input for the neural network. Also, it was found that playing back a vinyl at a slower speed than normal (15 RPM instead of 33⅓ RPM) gave much clearer sound where distortion took place (although a high-end AT440MLb cartridge was used). In order to play the records at a constant slower speed, a precise mechanical speed adaptor was designed, created and then installed on the used turntable.

For the future, we'd like to improve this work even more, in terms of quality, efficiency and functionalities. For example, the applying of filters can be made faster by using parallelism. There's also a type of groove distortion for which there seems to be a much better method of reconstruction than the detect-and-repair presented in this paper.

Bibliography

[1]. Wikipedia article on “Phonoautograph”, https://en.wikipedia.org/wiki/Phonautograph (Retrieved 27.04.2018)

[2]. Edison's Impression: Laying Sound into a Groove, https://www.youtube.com/watch?v=0vbyoZDQaIY (Retrieved 27.04.2018)

[3]. Emile Berliner's Fix: Flatten the Cylinder to a Disc https://www.youtube.com/watch?v=w_g4cAXkz80 (Retrieved 27.04.2018)

[4]. Electrical Recording, http://ethw.org/Electrical_Recording (Retrieved 10.06.2018)

[5]. John Pfeiffer: Quality by the numbers, Liner notes of RCA Red Seal Digital ARC1-3459 (Stravinsky – The Firebird Suite (1919), Symphony in three movements)

[6]. Electromagnetic Induction, https://nationalmaglab.org/education/magnet-academy/watch-play/interactive/electromagnetic-induction (Retrieved 30.04.2018)

[7]. Jose Maria Giron-Sierra: Digital Signal Processing with Matlab Examples, Volume I, Springer, ISBN 978-981-10-2534-1

[8]. Kathirvelu M.: Certain Investigation On Optimized Area And Power Delay Product In Digital Circuit Applications. Chapter 5: FIR Filter Design, http://shodhganga.inflibnet.ac.in/bitstream/10603/24055/10/10_chapter%205.pdf (Retrieved 10.04.2018)

[9]. FIR Filter Basics: http://dspguru.com/dsp/faqs/fir/basics/ (Retrieved 10.04.2018)

[10]. Tapio Saramäki: Finite Impulse Response Filter Design, https://www.cs.tut.fi/~ts/Mitra_Kaiser.pdf (Retrieved 11.04.2018)

[11]. Tang, Kwong-Tin: Mathematical Methods for Engineers and Scientists 3. Fourier Analysis, Partial Differential Equations and Variational Methods, Springer, 2017

[12]. Discrete and Fast Fourier Transforms: http://www.alwayslearn.com/DFT%20and%20FFT%20Tutorial/DFTandFFT_BasicIdea.html (Retrieved 11.04.2018)

[13]. FIR Filter Design: https://dspguru.com/dsp/faqs/fir/design/ (Retrieved 11.04.2018)

[14]. Cedrick Collomb: A tutorial on Burg's method, algorithm and recursion, 2009: http://www.emptyloop.com/technotes/A%20tutorial%20on%20Burg's%20method,%20algorithm%20and%20recursion.pdf