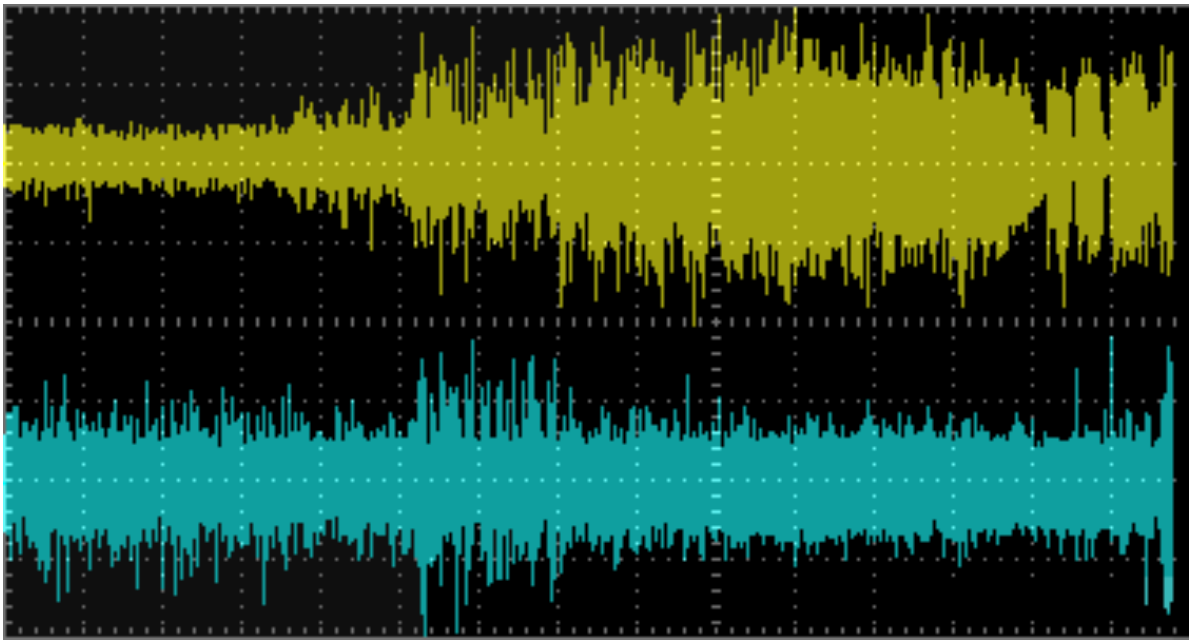

Dynamic Range Sound Compressor Unit

Can the "loudness war" be ended by giving the end-user control over the dynamic range compression?

Bachelor's thesis in Computer Engineering

Jesper Nilsson & Thorbjørn Bonvik

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY / UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Bachelor's thesis 2017:17-14

Dynamic Range Sound Compressor Unit

Can the "loudness war" be ended by giving the end-user control over the dynamic range compression?

Jesper Nilsson & Thorbjørn Bonvik

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Computer engineering undergraduate
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Dynamic Range Sound Compressor Unit
Can the "loudness war" be ended by giving the end-user
control over the dynamic range compression?
Jesper Nilsson
Thorbjørn Bonvik

Supervisor: Sven Knutsson, Department of Computer Science and Engineering
Supervisor: Alexander Hentschel, Tritech Gothenburg
Examiner: Peter Lundin, Department of Computer Science and Engineering

Batchelor's Thesis 2017:17-14
Department of Computer Science and Engineering
Division of H
Chalmers University of Technology / University of Gothenburg SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Waveforms of a song, as input and output of the dynamic range compressor prototype.

Typeset in LaTeX
Gothenburg, Sweden 2017

Dynamic Range Sound Compressor Unit
Can the "loudness war" be ended by giving the end-user
control over the dynamic range compression?

Jesper Nilsson & Thorbjørn Bonvik
Department of Computer Science and Engineering
Chalmers University of Technology

Sammanfattning

Musik produceras idag för att låta så högt som möjligt för att överrösta omgivningarna och låta mycket från små högtalare. Dock är inte denna effekt alltid önskad, och detta examensarbete undersöker möjligheten att motvärka denna tränd, som kallas för "the loudness war". Ideen är att ge slutanvänderen kontroll över mängden kompression i musiken. Detta vill ge producenterna möjlighet att fokusera på att göra sin musik så bra som möjligt for hi-fidelity system.

Rapporten vill ta för sig arbetet vid att skapa en prototyp, och diskutera hurvida den kan påverka "the loudness war".

Prototypen vill bli gjord på en liten linuxdator med fysiska in- och ut-portar genom ett externt ljudkort. Programmen JackD och ALSA används för ljudhanteringen, och koden är gjord i språket C.

Matematikkonceptet använt blir diskuterat, men själva koden berörs inte i denna texten. Den är istället open source och tillgänglig online.

Abstract

As music tends to get produced for low end speakers and to sound the loudest, this paper might suggest a solution to end the so called loudness war. If the destructive dynamic range compression can be applied at the end users level instead of in the sound file itself, the producers can master with high-fidelity in mind again, so this text will describe the development of a dynamic range compressor designed for this specific purpose and then break down if and/or how the results holds promise.

The prototype is based on a small Linux computer with physical input and output through an external soundcard. The utilities JackD 2 and ALSA is used for the sound handling, and the code is written in C.

Math concepts used to develop the code will be described in the Theory part, but the code itself will not be gone through in this paper. However, the code should be accessible online and it is open source.

Keywords: Music, Audio, The Loudness War, Dynamic range compression, linux audio

Acknowledgements

This thesis presents the project "Digital range signal compressor" for the degree Bachelor of Computer Engineering, at Chalmers Univiersity of Science. The work was carried out in Gothenburg at Tritech during spring of 2017.
We would like to thank our supervisor, Sven Knutsson, for the guidance and help during the project. Also we would like to thank Erik Agrell for a lot of good feedback. We would also like to thank our supervisor in Tritech, Alexander Hentschel, for the continued support and feedback.
Lastly a thanks to Tritech for lending us equipment and letting us use the office to work during the project time.

Jesper Nilsson, Gothenburg, June 2017
Thorbjørn Bonvik, Gothenburg, June 2017

Contents

Abbreviations

DRC, Dynamic Range Compressor
DTC, Dynamic Threshold Compressor
CUR, Compressor with Dynamic Ratio
DAC, Digital Audio Converter
DASC, Digital Audio Sound Compressor
DAFX, Digital Audio Effects
dBFS, Full Scale Decibel
PTP, Peak To Peak
RMS, Root Mean Square
DSP, Digital Sound Processing
RCA, A/V Cables
ALSA, Advanced Linux Sound Architecture
PDR, Possible Dynamic Range

Variables

T = Threshold (dB)
R = Ratio (X:1)
W = Knee Width (dB)
M = Make up Gain (dB)
TA = Attack Time (s)
TR = Release Time (s)
Fs = Sampling Frequency (44100 Hz)
X_{dB} = X in decibel range
g_c = Computed Gain
g_s = Smoothed Gain
α_A = Attack Time Coefficient
α_R = Release Time Coefficient

1

Introduction

1.1 Background

Music has always been everywhere humans go, but the digital revolution has made a big shift in how people listen to music and how the music itself sounds. Smartphones and computers gives the opportunity to listen to any song, anywhere. This is a great convenience, but a unfortunate phenomenon has come with the digital age of music.

In the 80's the CD was introduced, and it took over as the superior media over the old analogue vinyl. (Matthews 2014) On a vinyl one would print analogue signals into plastic, and then read it with a needle. On the digital CD however, it holds recorded sample values which is read by laser and the waveform can thereafter be recreated. This made way for a big shift in music production, as the digital media could recreate sounds physically impossible, or at least unpractical, to read from vinyl. Most important in this context is that the CD gives a greater *possible dynamic range* (Sreedhar 2007).

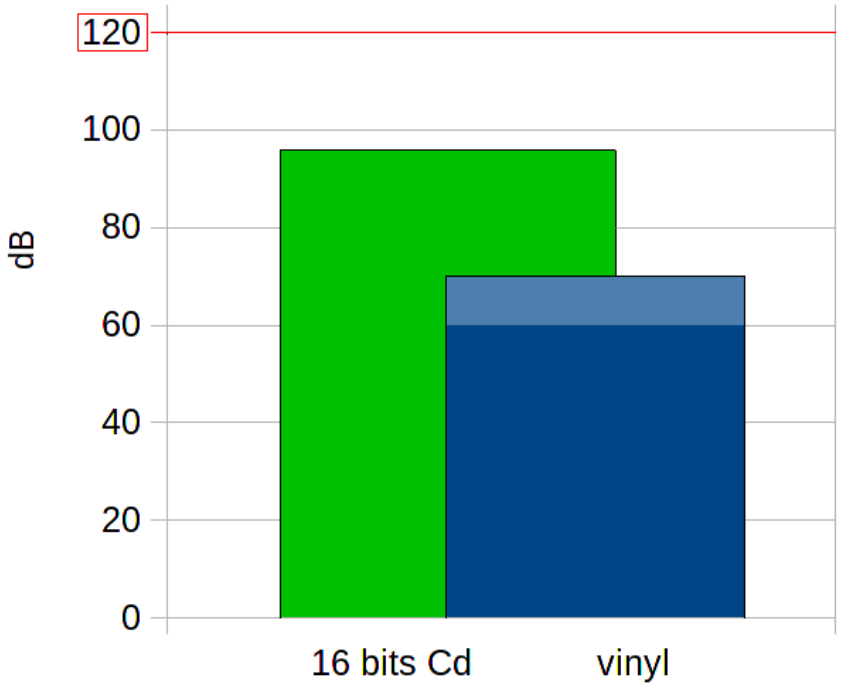

Figure 1.1: Possible dynamic range in vinyl versus in the first CD's. For comparison, 120 dB is the human threshold for pain.

Dynamic range is the difference between silence and the loudest. A LP-vinyl has about 60 to 70 dB PDR and a 16 bit CD has around 94 dB PDR. (hardly 2004) Having increased possible dynamic range gives opportunity to increase the jump between quiet and loud in the musical media. Unfortunately it has been widely misused, as producers pushes the limits to how *loud* their music sounds instead of using the space sparingly to produce dynamics. So the added 20 - 30 dB of headroom has mostly been used to increase average loudness, instead of the dynamic ranges.

1.2 Purpose

Expensive speaker systems is not something most people have. Actually most people seem to listen to music through small speakers like computer speakers, radios and headphones.Resnikoff (2015) Here, compression is useful due to the moderate players. On the radio, songs gets compressed to drown out the possible static noise and interfering signals. Currens (2011).

In a noisy environment like shopping malls and the outdoors, compression can be a great way to make the music sound consistent. And some more creative applications could be compressing so those with hearing impairments can listen to the quiet parts of a song without increasing the master volume. So the amount of compression one would like to add to a track varies from situation to situation, but there is no way to adjust the distorting compression added in the production. This is a negative feedback problem, where producers will keep compressing the music until a solution

is created. The solution would be redundant until the compression stopped. But if there was widespread, adjustable compressor at the end-user level, the culture could change for the music producers, to produce without the pressure for loudness.

1.3 Goal

The goal is to build a Dynamic Range Compressor prototype that can apply the desired amount of compression the music at end-user level. And then derive if the idea holds promise for adding compression where desired and if this could be a factor in removing compression added to the music sources.

This prototype would function as a intermediate stage between source and playback device, setting a threshold in decibel where the compression should start and the ratio of compression.

The ideal vision is a product with two knobs, one for volume and one for the amount of compression. The amount of compression will be decided by adjusting one impactful parameter in the code, and static pre-defined parameters for the less impactful. This will hopefully give an user the impression that they can control the maximum and minimum sounds volume independently.

1.4 Method

The prototype will be based on a Raspberry Pi using an Audio Injector'sStudios (n.d.) Digital-Audio-Converter (DAC) and JackdSolodovnichenko (Last edited: 25 Mar 2016), a ALSAKysela (1998) based program, to write C - code. The project will be done with an agile approach, with weekly sprints and meetings with supervisors. To be able to accurately compare the input and output signals, an oscilloscope will be used to get a visual wave representation and to read the values from the output. Equations will be based on the book *DAFX - Digital Audio Effects*Zölzer (2011) and MathworksMathworks (2017) equations on dynamic range compression. The code will be written based on Jackd's own library and type definitions, as well as regular C libraries. To present the results, recorded sounds will be compared between the compressed and uncompressed signal. They will then be presented with images generated on the oscilloscope.

1.5 Limitations

Perceived loudness is not the same thing as actual loudness, as the human ear is more sensitive for certain frequencies than others. This leads to that sounds with the same decibels of sound pressure might appear differently loud. There does exists weighted decibel scales for perceived loudness, but equalizing and frequencies will not be considered for this text.

Similarly, additionally to mastering with excessive compression, producers use different techniques called *noise shaping* to increase *percieved* loudness. This part of the loudness war will be left out of this project.
Also to limit the amount of work put into this thesis, custom hardware will not be built, instead the project will use Audio Injector soundcard for the Raspberry pi. The theory part will be compressed and not to much on focus will be put on, only learning how to transform digital sounds to expected values.

2

Theory

2.1 The Loudness War

The transition has been gradient, with subtle increase of volume always competing to sound a little bit louder than the last song the concumer heard. This is what has been called the "loudness war", and works as a subtle marketing trick and as a choice of style.

This included with the human ear often confusing loud music with good music,Blesser (2007) a lot of people prefer to have reduced dynamics to feel the rush of a hammering wall of sound to get pumped. And as the loudness war has been most in effect in pop music, maybe some people has been trained to confuse the sound of loud music as popular music.

Coalesced with the loudness war issue, so has the way and the locations music is played changed. Having the privacy and calm of a home speaker system or the setting of a live act gives room for much more dynamics as the music does not have to compete in the same way as with cheap earbuds on the move. So now music producers have a challenge to make the music sound the best in every possible environment, which is impossible.

The connoisseurs of high fidelity music feel they are disregarded. Arguably compressed music does sound better on low end speakers as the loudness will make the sound clearer, but the thing with distorting recordings with excessive compression is that it's irreversible and sounds terrible on premium speakers.

As the norm is for music to be produced for the mass marked of low end speakers, there is desperation over getting the highest dynamic range music files and albums. Online, there is plenty of hi-fi forums with threads that hunt for uncompressed bootlegs of albums with excessive mastering. There is even a website called "dynamic range database" with data for the dynamic range of albums so one can choose their music shopping based on sound quality.

One example of the loudness war leading to dissatisfaction by the hi-fi communities is the album Californication, by the Red Hot Chili Peppers. This album disappointed a lot of fans when it was released with almost no dynamic range in 1999. There are hunts for the least compressed versions of the album (apparently

it's the 2012 vinyl printing*Dynamic Range Database, Californication* (n.d.)), and there is even a illegal bootleg re-master that circulates online. Mikemoon (2012)

Another example of an album that the loudness war distorted is "Death Magnetic" by Metallica, here the soundtrack used in the video game "Guitar Hero" has been ripped from the video-game as the solution to the albums extremely poor dynamic range. Shepherd (2008)

It is a weird situation to build a high fidelity system, buy the music and it sounds flat, distorted and loud because the producers has mastered for low end speakers and shopping malls.

2.2 Discrete Sound

Modifications on digital signals differs from analog. Sound is as everything else in computers, a collection of ones and zeros. This can never be an continuous waveform, but this instead represent a discrete collection of values called *Samples*. To then turn it into waveform, samples are read in a quick succession, and can be modified with a lowpass filter to recreate the original analogue signal.

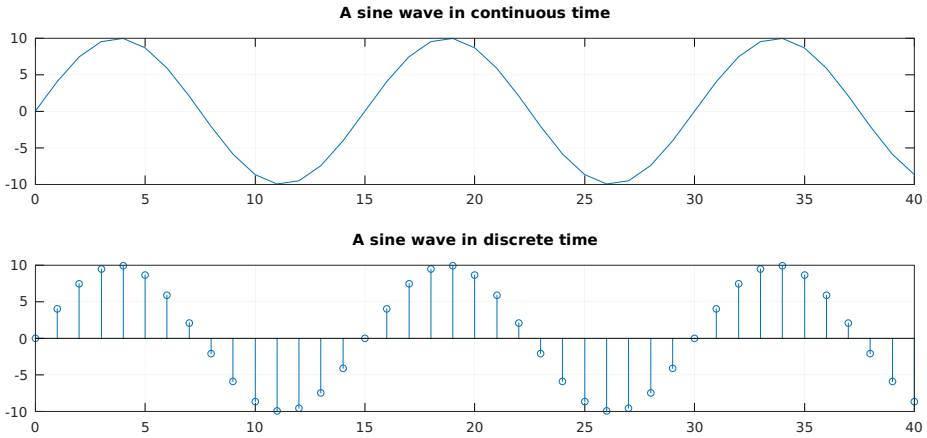

Figure 2.1: A continuous signal can be represented as sample values.

To be able to recreate a specific sound, the sample rate has to be twice as fast as the frequency of the highest frequency according to the Nyquist Sampling Theorem. (Bengtsson & Karlström 2007, Chapter 5) Due to this the most commonly used sampling frequency is 44100 Hz, which is more than twice highest frequency humans can perceive of around 20 kHz.

2.2.1 Calculations

When a wave is sampled it is saved inside a buffer. This buffer can be transferred between medias, but it can also be modified. To do calculations on the sound, samples are both looked at as single values and combined into a sequence of discrete

values. These signals can then be modified by changing the sample values.(Bengtsson & Karlström 2007, Chapter 2)

2.3 Digital Sound Processing

2.3.1 Decibel

An important concept in this context is *Decibel*, a logarithmic ratio between two quantities. A reference point is often used as one of the values, to express the level compared to this reference. For example dBm is the power with a reference point at one milliwat Wolf (2011). When the two quantities have a large dynamic range, meaning they range from small to large values, the linear scales do not give a good visual representation of changes. It can then be converted to the logarithmic scales, called *bel* in honor of the scientist Alexander Graham BellEngineering & Wiki (Last edited: 13 January 2016).

The human hearing spectra is an example of a large dynamic range, from 20 Hz to 20kHz, being perceived logarithmic by us. It is therefor often represented in *Decibel*. Decibel means that the scale is based on a power of ten, making the scale size be the same between 0 and 10 as between 10^4 and 10^5, being one tenth of a *bel* (Molin 2013, Chapter 1.4).

A digital sound signal is measured in *dBFS* which is means *Full Scale Decibels*. This is a measurement where 0 dBFS is the highest intensity the system can handle before clipping the waveforms. The scale is then measured in the negative axis, meaning a -10 dBFS is 10 decibels below where you clip and distort the sound Mellor (2006). It ends around -90 dB as it is no longer perceivable by the human ear.
The equation used to convert a signal to its *dBFS* equality is:
$X_{dB} = 20 * log_{10}(X_{input})$.

2.3.2 Compression

The concept of *compression* is one of the more frequent *Digital Signal Processing* DSP effects. Compression takes away strength from signals that have an amplitude over a set threshold.
As mentioned before to calculate a waveform, more than one input is used. The buffer of samples that gets saved each clock-cycle can be used to check for either *Peak-to-peak* PTP values or *Root Mean Square* RMS values, where PTP searches for the peak values inside the wave and RMS checks for a squared mean value over the samples. This is then the value that becomes the input of the calculations. This value in *dBFS* is then compared to three dynamic values, *Threshold, Ratio* and *knee width*. Threshold is the decibel range where the compression starts, dynamically changeable. Ratio is the amount of reduction desired, scaled $Ratio : 1$. Knee Width is a value that compared to the threshold makes the compression around the desired

threshold decibel value with a quadric curve to smoothen the otherwise audible start of compression.

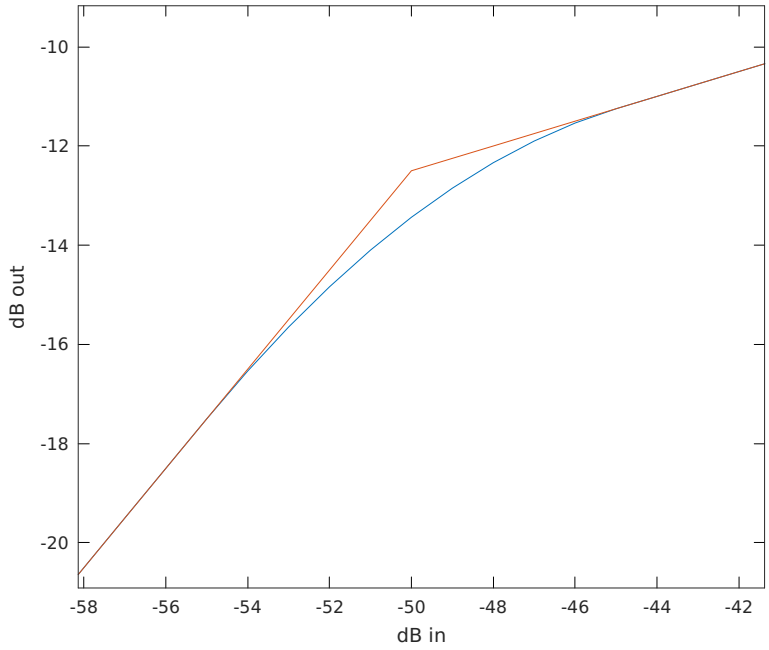

Figure 2.2: Compression with a soft knee coloured Blue and a hard knee coloured orange

Compressing a signal takes away volume from the dynamic range that triggers it. *Make-up gain* is then added to counteract the volume reduction, and this leads to the desired effect of making the quieter sounds, the range under the threshold, louder in the mix.

2.3.3 Expanding

Another common *DSP* is expansion, a complete opposite to compression. Instead of checking above the threshold and lowering those values, a expander instead checks the values below the threshold. Those values are increased to sound louder without changing the higher samples. The idea to implement both a compressor and an expander was so a system would compress and expand with the same threshold and thereby keep a linear equation, as in figure 2.3 expanding/compression.

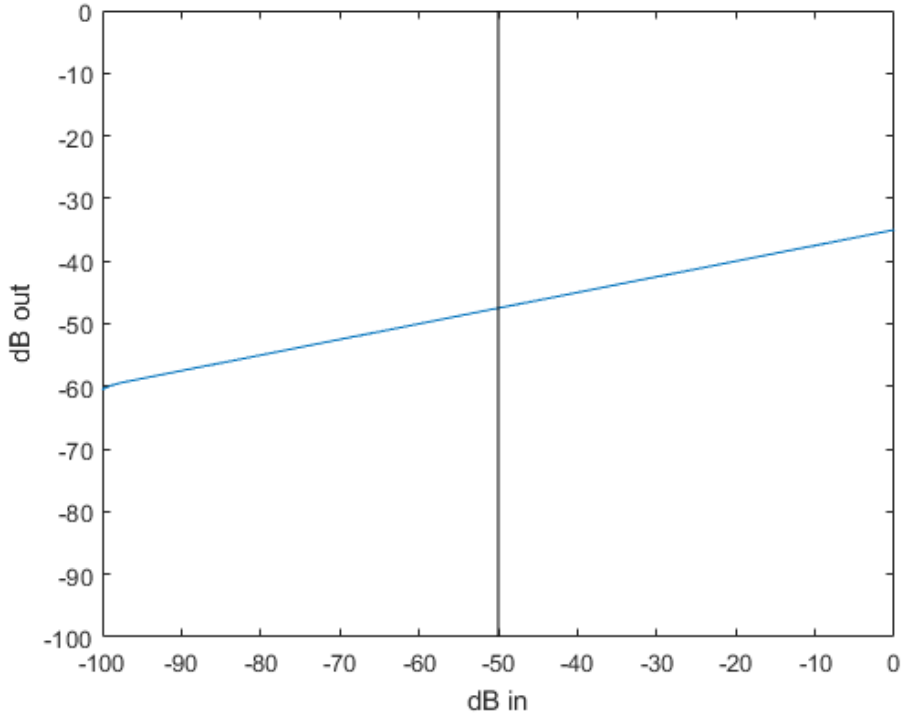

Figure 2.3: Compressor with a implemented Expander

This removes parts of the equations where the squared function around the knee would apply and instead make the entire system linear. The reason for a soft knee is to remove distortions between compressed and not, with the knee removed that would solve itself.

3
Methods

3.1 Preliminary investigations

There does exist some related technologies on the marked already, but there is none directly equivalent. Their ideas are similar, but they differ in the solution and implementations.

Noise cancelling headphones might be the most protruding example. Some of the same desired results of the product of this work is obtained through this technology, but with some major differences. With noise cancelling headphones you will still not be able to hear the most subtle sounds if your hearing is impaired at normal volume. Also, with compression of the sound levels, the risk of being startled or confused by surprising sounds will be reduced as the dynamics are reduced and one will have a possibility to hear both the world around one and the most subtle sounds by compressing hard and having low volume.

As the main platform for consuming music has gone from CD's to streaming and files, *replay volume normalization*, is a reoccurring phenomena. Through streaming services, the streaming provider can add their own "touch" to the digital audio. What Spotify does, the biggest streaming provider, is to adjust the volume for a track based on its dynamics.Shepherd (2015)

This is just the volume being adjusted track by track however, no compression. So firstly, the whole deal with making subtle sounds appear louder in the mix without increasing peak volume is missing, and the effect is not as freely adjusted to user preference as the compression concept. Also, if an album is produced with dynamics between tracks the end effect will *remove* dynamics and is therefore become somewhat counter productive.

It might be obvious, but compression is not a new concept or invention. The idea is just to apply the concept in a new place, and making a compression algorithm fit for the application.

3.2 Equipment

This project will be done on a Raspberry Pi with an external sound-card. A lot of different hardware could be used, both for the sound-card and the processor, but

Rasperry Pi with a debian based stripped down operating system, was chosen due to its fast computation and well used platform.

The sound-card is an "Audio Injector" by Flatmax StudiosStudios (n.d.), which connects to the Raspberry Pi natively through its GPIO pins. It has some nice features and smart design, it leads the GPIO pins through and has knobs for controlling input gain and output volume, RCA in- and outputs, and an 3.5mm jack output.

Connected to the GPIO pins a basic DIP-switch for input values is connected. Ideally one might prefer using an analogue input like a knob or slider, as this will be much more user-friendly. But unfortunately the Raspberry Pi only has digital inputs, and converting an analogue signal or waiting for special parts to arrive would waste time as any variable input will do fine. An advantage with the dip switch is that 8 bits can be divided into 2 times 4 bits inputs.

The output is read from the RCA outputs by a oscilloscope, to get accurate reading and visual representations of the compression. A headset is used to at the same time listen and compare the audible differences and searching for distortions.

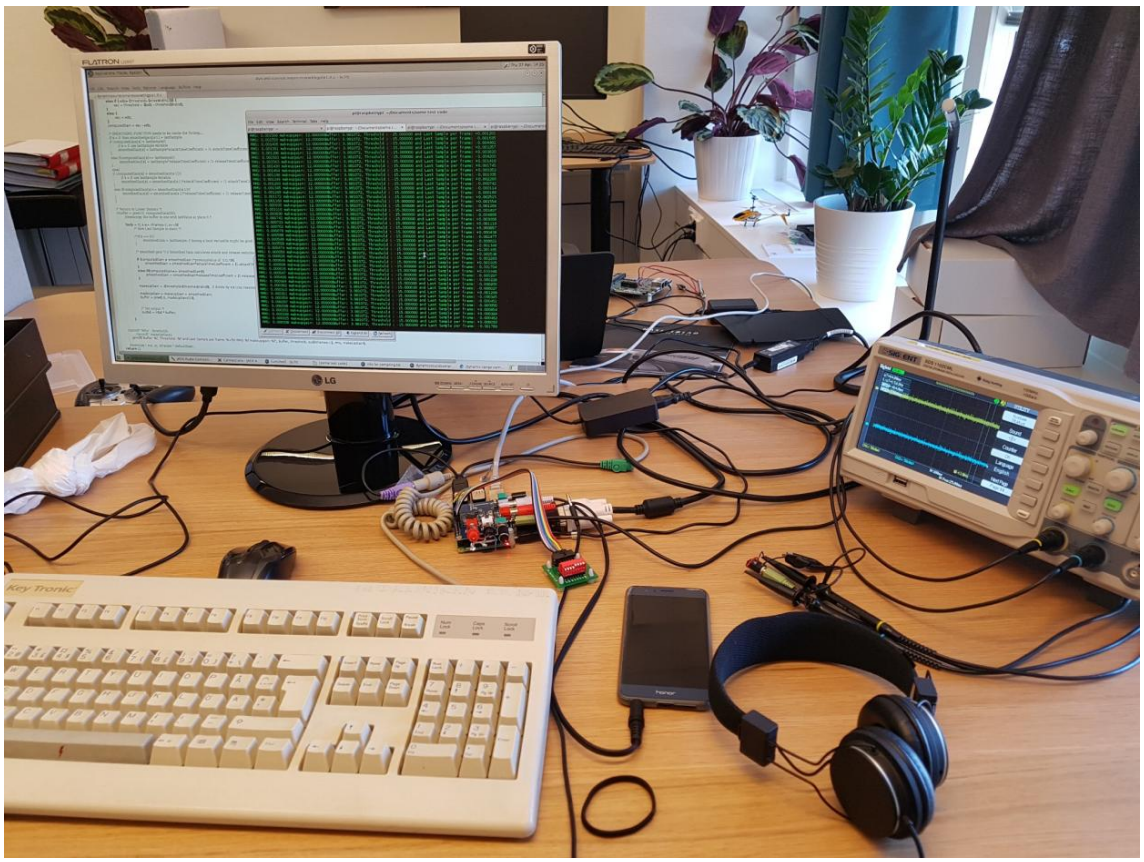

Figure 3.1: A picture of the setup, with the Raspberry Pi connected to a oscilloscope, a keyboard and a screen.

3.3 Sound in Linux

For sound computation, Linux is not that common. There is some support for the concept and it works quite well, but there is some shortcomings and complications with DSP in Linux.

First of all, since it is Linux there is no definitive way to do actual sound processing. This gets further complicated as the hardware is limited, without a capture port to use as input.

The basis for sound in Linux is ALSA, *Advanced Linux Sound Architecture*Kysela (1998). ALSA is a part of the Linux kernel and is the foundation for Linux sound processing, although bareboned and without a gui. To make the process of accessing and modifying the sound data easier, JackD Solodovnichenko (Last edited: 25 Mar 2016) is a overhead program that handles ALSA's I/O's. JackD lets one connect sound I/O between programs in Linux both graphically and in terminal, so using JackD gives the benefit of structure. It also comes with a lot of code-examples to try, which is a great way to get a feeling for how sound processing will be done.

3.4 Latency

A problem is the amount of time based delay this setup will lead to, both concerning hardware and software choices and the properties of later produced code. For this prototype any optimization for delay has been discarded as the prototype applications will not be in real time.

3.5 Matlab

It is difficult to analyze sound objectively, and therefore the math software MATLAB and GNU/Octave is used to visually show the calculations. The graphs can then be evaluated to check if the equations is working as intended. The syntax of MATLAB scripting is quite different from C, however the logic is similar and the visualization is a handy tool.

3.6 Approaches/Implemetation of Theory

To start implementing the theory in code, changing from stereo to mono makes it easier to notice changes in sound. This is due to being able to compare in real time the compressed and the original sound. Actual code is based on *MathWorks Dynamic Sound Compressor*Mathworks (2017), where values are compared to a set *Threshold* that decides where the compression should start in decibel. Combined with a *Ratio* and *Knee Width* the desired compression is then achieved. The equations:

3.6.1 Root Mean Square

$$RMS = \sqrt{\frac{1}{n} \cdot (x_1^2 + x_2^2 + \cdots + x_n^2)} \tag{3.1}$$

This caculation is used to get a mean value over a buffer of 512 frames. A frame is defined as one *float* per channel of sound. That can later be used to get a smoother compression.

3.6.2 Gain

$$X_{dB} = 20 \cdot log_{10}(RMS) \tag{3.2}$$

Using the RMS result, it can be converted to the logarithmic plane.

$$X_{sc} = \begin{cases} X_{dB} & \text{if } X_{dB} < \left(T - \frac{W}{2}\right) \\ X_{dB} + \frac{\left(\frac{1}{R} - 1\right)\left(X_{dB} - T + \frac{W}{2}\right)^2}{2W} & \text{if } \left(T - \frac{W}{2}\right) \leq X_{dB} \leq \left(T + \frac{W}{2}\right) \\ T + \frac{X_{dB} - T}{R} & \text{if } X_{dB} > \left(T + \frac{W}{2}\right) \end{cases} \tag{3.3}$$

Where T = Threshold, R = Ratio and W = KneeWidth. This is the main part of the equations, where the gain is computed according to the set values of T,R and W.

$$g_c = X_{sc} + X_{dB} \tag{3.4}$$

The computed gain and the original signal is added to each other.

3.6.3 Smoothing Gain

$$g_s = \begin{cases} \alpha_A \cdot g_s[n-1] + (1 - \alpha_A) \cdot g_c & \text{if } g_c[n] > g_s[n-1] \\ \alpha_R \cdot g_s[n-1] + (1 - \alpha_R) \cdot g_c & \text{if } g_c[n] \leq g_s[n-1] \end{cases} \tag{3.5}$$

$$\alpha_A = \exp\left(\frac{-log(9)}{Fs \cdot T_A}\right) \tag{3.6}$$

$$\alpha_R = \exp\left(\frac{-log(9)}{Fs \cdot T_R}\right) \tag{3.7}$$

Where TA = AttackTime Period, TR = ReleaseTime Period and Fs = Sample Frequency. The smoothing function is there to give a natural transition from no

compression to the calculated compression. Mathworks (2017)

3.6.4 Make-up gain

There is many ways to apply make-up gain. In this prototype the make-up gain is based on how much gain needs to be added for 0 dB in to give 0 dB out.

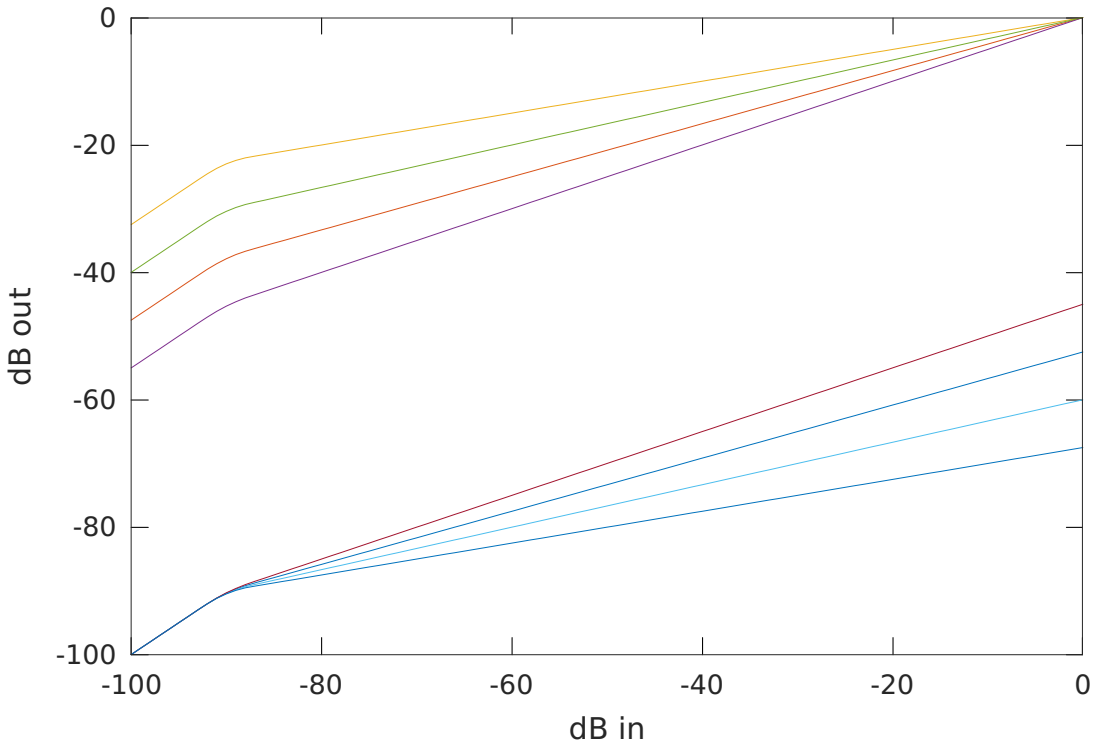

Figure 3.2: Compression with ratios between 1:2 and 1:4 with a threshold of -90, with and without make-up gain.

Calculating and adding make-up gain equal to the inverse of X(0 dB) will make sure that 0 dB in gives 0 dB out with a set of parameters.

$$M = -X_{sc}(X_{dB} = 0) \tag{3.8}$$

$$g_m[n] = g_s[n] + \mathrm{M} \tag{3.9}$$

$$g_{\mathrm{lin}}[n] = 10^{\left(\frac{g_m[n]}{20}\right)} \tag{3.10}$$

When the make up gain is added to the smoothed gain, it is returned to the linear

domain to match the input/output domain.

$$\text{out}[n] = \text{in}[n] * g_{lin}[n] \tag{3.11}$$

The final step is to multiply the input sample with the gain calculated from the values set. Mathworks (2017)

3.7 Offsets

When the make-up gain is calculated, it is done with regards to only threshold and ratio, resulting in make-up gain where there is no compression. This was expected, but a unforeseen problem was detected in the hardware setup with it. The inputs were not normalized, as taking in a peak volume of 0 dB did not register as even close to 0 dB in the PCM.

This was a problem only present when playing from a different device, all sounds from within the Raspberry Pi gave expected values. Since the main goal for the prototype was to be able to dynamically compress input data without saving it, a offset was implemented. By feeding the computer with a generated square wave, and all volume knobs turned to max, the highest possible registration was still below 1, around -8 dB. The offset was nonlinear and based on what sounded the best, as well as checking the highest input for some songs.

The normalization issue could be problematic for the calculation of the make-up gain based on the variable threshold, as the threshold would need be lower in the raspberry than relative in the input signal. To trigger at -5 dB relative to the inputs peak of 0 dB would mean the threshold would need to be at -5 dB + (- 8) dB and the calculations would be skewed.

3.8 Variable threshold v.s. variable ratio

Make-up gain for the variable threshold would however not make to much sense if the ratio was static, as the following illustrations show. The make-up gain would lead to the curve being static.

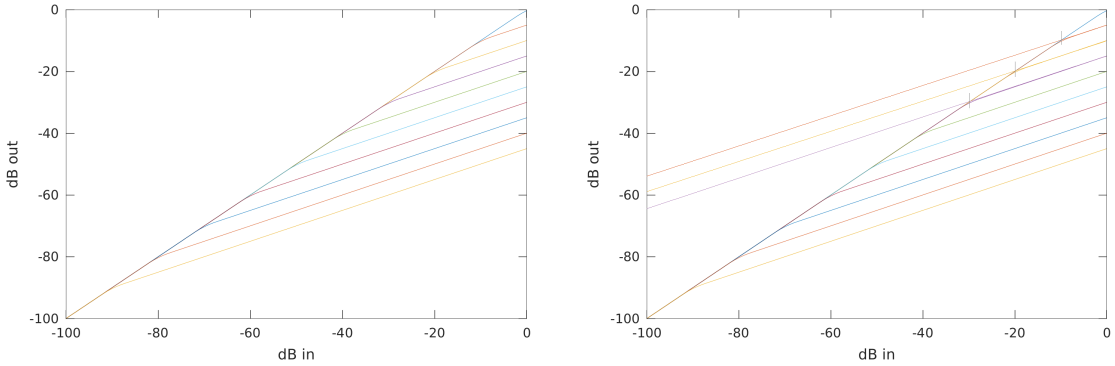

Figure 3.3: How changing the threshold will change the output, the right shows what would happen with adding an expander.

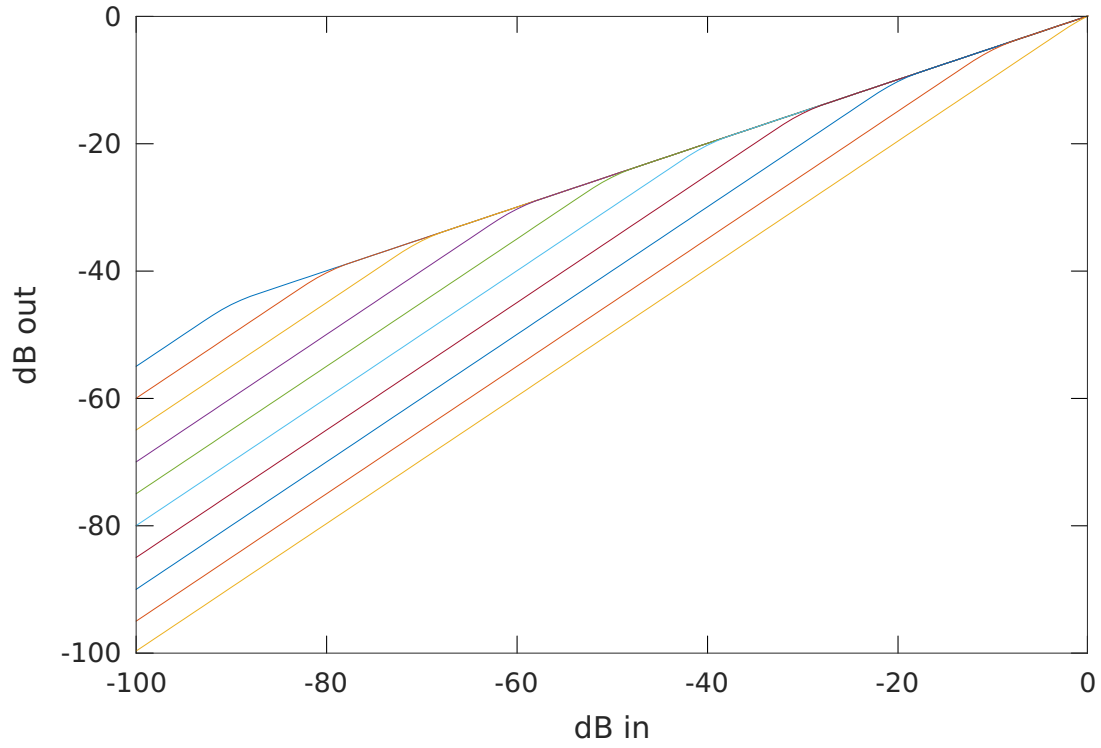

Figure 3.4: How changing the threshold will change the output with automatic make-up gain.

So an idea came up when it was time to implement the expander. Applying the automaticly calculated make up gain would lead to the curve characteristics only to change to the ratio, which was intended to be static. But if the sound gets compressed and expanded around a threshold to become a linear function, setting a low threshold and adjusting the ratio would lead to the same linear characteristics above the threshold and the make-up gain could be calculated with the input provided to the user.

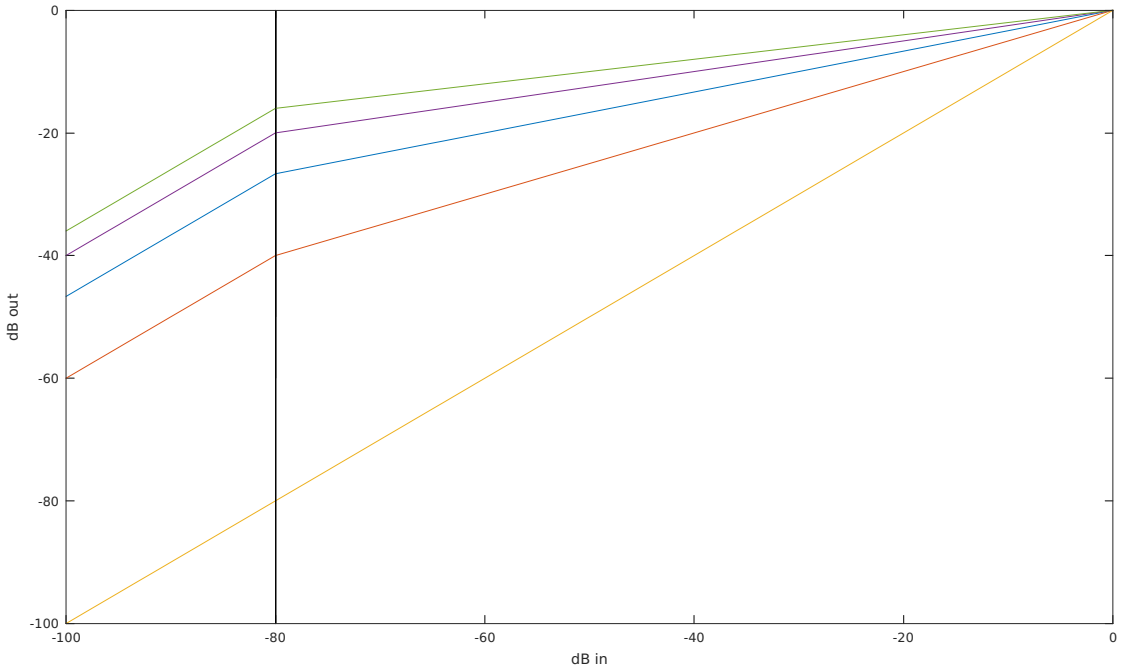

Figure 3.5: The relationship between in and out with an variable ratio and make-up gain.

3.9 GPIO Pins/DipSwitch

The dipswitch was connected directly on to the GPIO pins. This worked alright, but some adjustments to handle the inputs needed to be implemented. The problem was the read value was not 0 or 1, but instead 0 or the value of the GPIO pin squared. This meant that the GPIO pin with number 25, out of Raspberry Pi's 40 pins, was either 0 or 25^2.

The first implementation was to read the values of the dipswitch and save them inside a array. When the values were found wrong the solution was instead to check if the value was 0 or not. If not, then a one was saved instead of the actual value. This was to be done continually trough the execution, to achieve the dynamic control. The method of saving the values was a simple continuous left shift.

The function could then be called inside the main process and the values can then be changed to appropriate scales. If the value is the threshold, then nothing had to be changed. But for dynamic ratio, a change from one to two is large. This value was instead divided down so each step on the dipswitch was smaller than one.

4

Results

The results are based on the second implementation of the Dynamic Range Compressor Unit, where the Ratio became the dynamic value. The reason is discussed further in the next chapter. The input is Spotify songs and they were played through the Audio Injector shield. The output from Audio Injector is directly sent to a Oscilloscope where the waveform is printed. One channel on the oscilloscope is displaying the original song in mono and the other channel displays the compressed result. A pair of headphones allows the user to simultaneously listen to the original and the compressed song in the left and right earpiece.

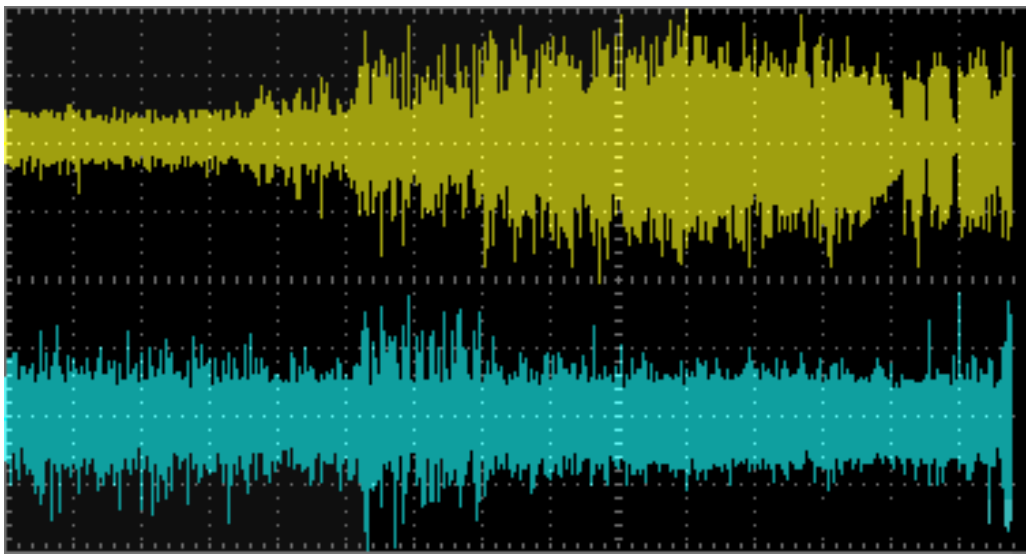

Figure 4.1: Baba O Riley, The Who, yellow is the original input, blue is compressed

Offset	Threshold	Ratio	Knee-width
16 dB	-80	4.75	2.4 dB

This is the first 2 minutes of the The Who song Baba o'riley. As can be seen in the picture, the compressed signals contains large spikes in the middle. This is due to attack time and release time, further optimiziation here would be needed. The reason this phenomena exists is explaned in section 4.1.

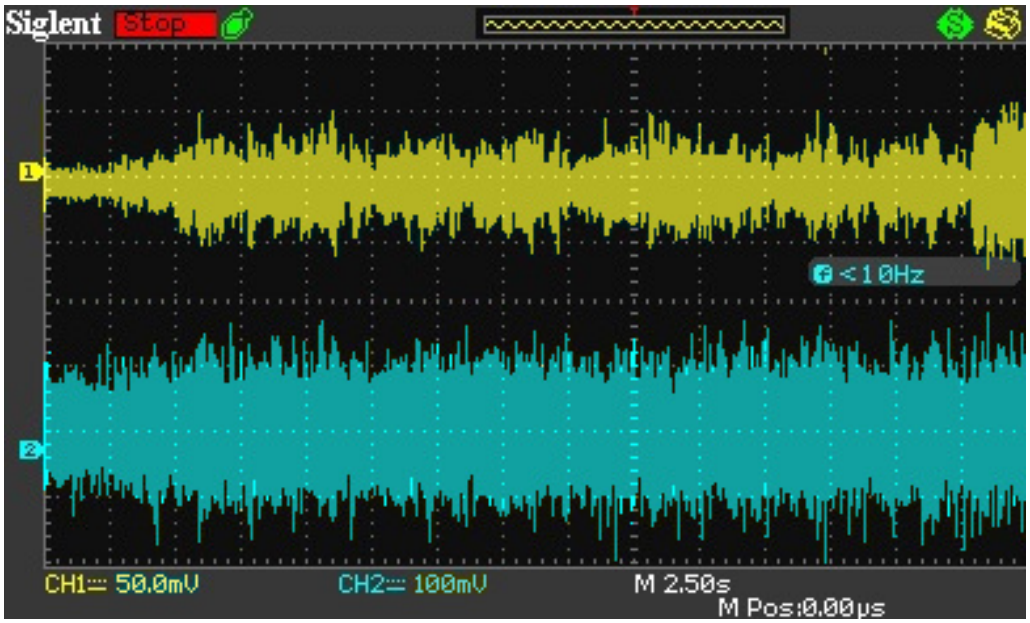

Figure 4.2: Tjajkovskij, Waltz of the flowers, yellow is the original input, blue is compressed

Offset	Threshold	Ratio	Knee-width
20 dB	-80 dB	4.75	2.4 dB

It is clear that the songs are overall extremely compressed, proving the possibility to compress songs dynamically on the end-user side.

4.1 Attack and release / shortcomings / discoveries

One important discovery in the prototype testing, is that attack and release time causes loud spikes at moderate to heavy compression. The make up gain boosts the loudness and the attack time gives sudden loud noises a couple of milliseconds to hit really loud volume. So when the compressor works as intended most of the time, a sudden solo guitar pluck will sound really loud compared to the rest of the mix.

The same issue becomes even more apparent in speech. Having a single voice talking gives loudness spikes at hard pronunciations like "p" and "k".

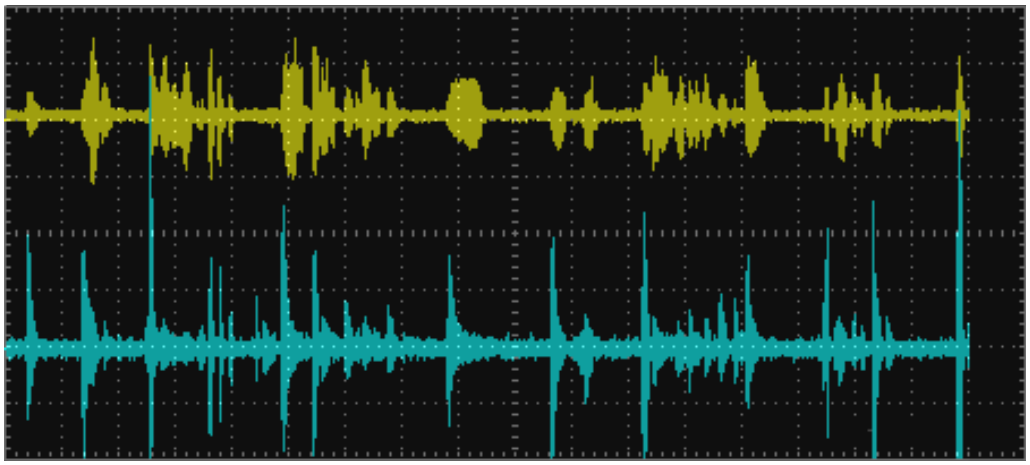

Figure 4.3: Comparison between input and compressed output, showing the attack spike problem.

5

Discussion

Overall the prototype seems to prove that the concept is possible, with the only major shortcoming being the issue with the attack time leading to loudness spikes already at moderate compression and the make up gain. However, only sudden loud sounds lead to this problem. Having a guitar pluck aggressively does not lead to the same phenomenon if there is a backing band that triggers the compressor and attack time already, in the implementation with dynamic ratio. Having a dynamic threshold will lead to attack loudness spikes at anything that suddenly crosses the threshold.

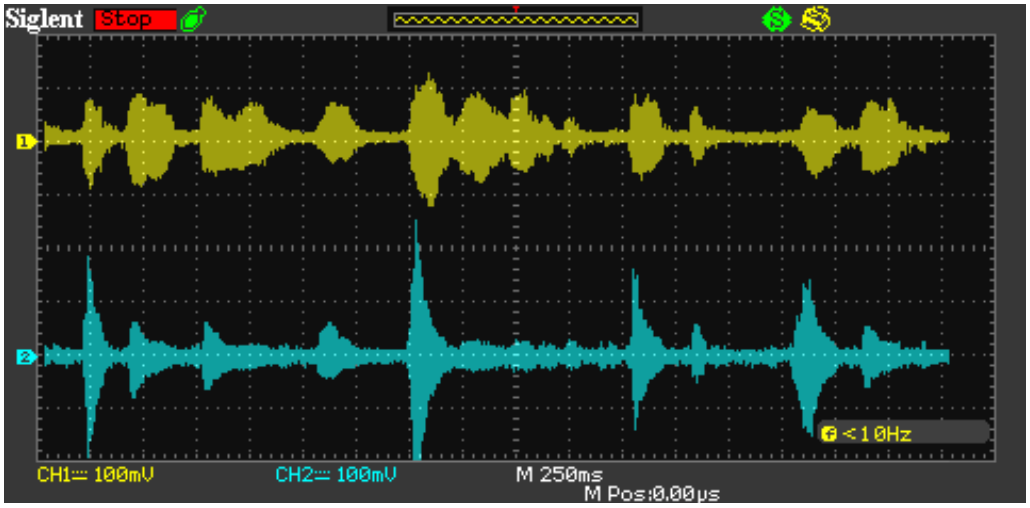

Figure 5.1: A couple of spoken words with heavy compression, demonstrating the excessive gain added before the compressor kicks in. Yellow is the original, blue the compressed

5.1 Controls

To control the amount of compression at any given time, dynamic input is needed. This can be achieved in many ways, like a digital signal inside a mixer application or analogue knobs.

The vision to use the two knobs on the Audio Injector, that control the in and out volume, to control the compression and volume was not possible due to audio injector's structure. Instead a DIP-switch was used to gain the dynamic control. This was a poor solution, attaching the DIP-switch to the Raspberry Pi and using

the inputs had to be done through the GPIO pins directly. Without protection the switch fit loosely and caused the program to sometimes read it 255 or max, independent of what was really put in. The result was not only a loud noise, but also a problem in reading the offset when running the program with the switch. A preferred way to solve this would be to use a protected cable, that can connect over a collected number of pins or even another input altogether, connecting a more user friendly controller.

However this was sufficient to test the prototype. The limited adjustments possible by only tweaking one parameter worked relatively good. The attack- and release time would be good to adjust to a specific song however, as a countermeasure to the problems discussed earlier. Maybe these parameters could be calculated dynamically in some more complex programming.

5.2 The dynamic Ratio

After rigorously testing the implemented dynamic compressor where the threshold was the value connected to the dipswitch, the decision to try a different approach was made. This was due to make-up gain not working as wanted it to and after some time spent testing not improving. The original idea behind make-up gain would work if the input was normalized, but after testing, it was concluded that the Raspberry Pi had some offset. It was nonlinear and a correct algorithm was not found. The next implementation, akin to implementing a expander and combine it with the compressor, was to set the threshold to a extremely low value of around -80 dB and have the ratio decide the compression. The result was then a better all around compressor, independent of the strength of the input.

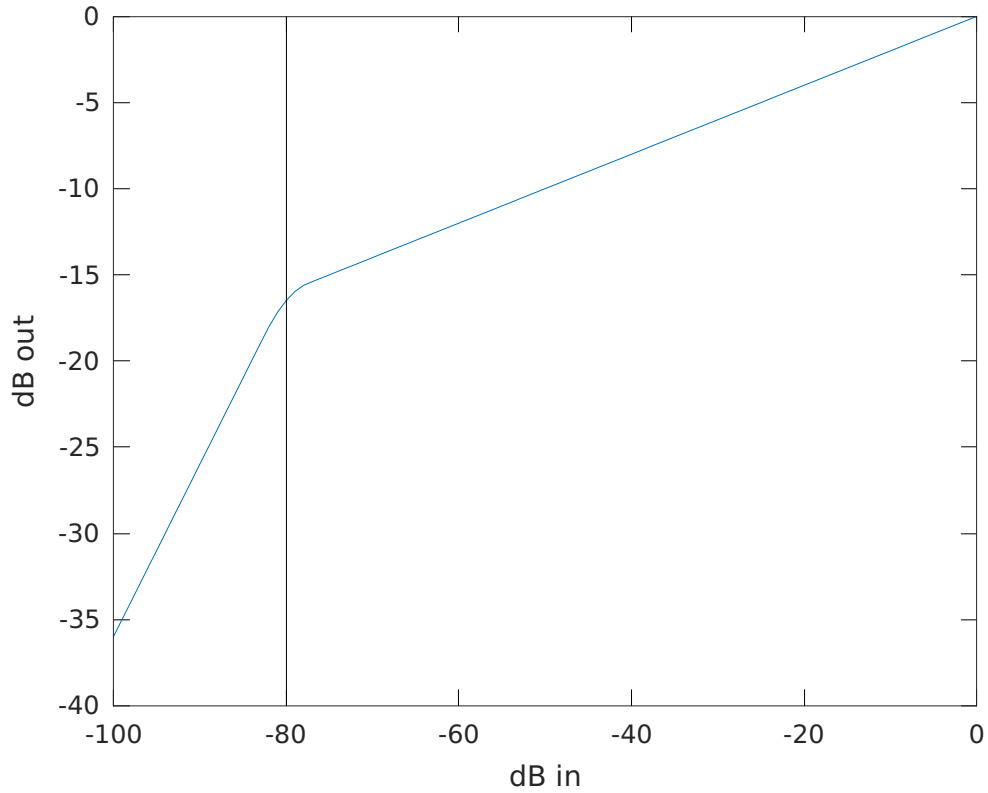

Figure 5.2: A picture of the dynamic ratio concept with hard compression of 5:1 and a soft knee.

Both of the implementations worked and they both have a make up gain that is not optimized. The difference is instead the way the unnormalized input is handled. The Compressor with dynamic Ratio, *CUR*, could handle any input and apply a compression, where the dynamic threshold compressor, *DTC*, only worked on the values below the threshold. Comparisons between the DTC and the CUR lead to preferring the DRC for most cases since the sound itself becomes better.

5.3 Future developments

This project was thought of with the vision to change the never ending compression which terrorize media these days. To reach as many people as possible, it will be open source and usable to anyone. With this, it could potentially reach producers as well.

The future for this project is branched and can be used in many different ways. If the DASC (*Digital Audio Sound Compressor*)stays on a Raspberry Pi, it could function as a media centre, being a computer. This would only require a very simple OS that allows media applications and a small screen to handle the different inputs.

Something that would affect the loudness war would be to implement this concept into streaming and playback applications. It could be as simple as a option for

"reduce dynamics / increase loudness" to make it simple for the users, and possibly with additional options for the different parameters. This could become an built in mixer feature in phones and portable media players. This would remove all the costs of hardware and make the product available to an even larger audience. However then the entire project would have to be rewritten and the code restructured to fit the current standards of phone application coding. Having a entirely software based project would also make a thing like security a much larger question.

Changing from a Raspberry Pi to another microprocessor would bring some new implementations as well. The Raspberry Pi as a hardware base is wasteful if only a compressor is implemented, a microprocessor that executes in real time with some controllers and a DAC would be enough. By compressing the hardware to only fit I/O compression, the size could be significantly smaller. The processor could then be changed to a small stick that only reads the input and with two controls sets the amount of compression. This would fit all media players that transfer digital signals. Another implementation would be to integrate the processor into a headphone or a stereo. This would then work both as the player and the compressor, removing the intermediate step between phone and player. As mentioned before a mixer application could be implemented, removing the need for hardware control, controlling the player with the phone.

5.4 Offset optimizing

The reason why max volume in does not register as max volume in the pi might be related to 24-bits sound vs 16-bits sound. Unfortunately there was limited time to research this problem, but an idea is that the PC's and cell phones used as sound sources have 16 bits output while the soundcard is ready to recieve 24 bits.
There might also be some configuration in either of the Linux sound components, ALSA, JackD, etc. Also every sound differs in peak volume as they are produced in different manners and with different intentions.

6
Conclusion

To conclude this project, a successful prototype was built and programmed to behave as expected. The compressor can with the set values and the dynamic input from the Dipswitch compress any incoming audio and set it to the output. The question if this could stop the loudness war can not be answered, but the prototype demonstrates that compression can be applied at the end-user level. The next steps from here is however uncertain, but any idea that is open source can start spreading.

This prototype has a lot of potential and with more work could be turned to a proper product. The goal set was to bring the compression to the consumer side. This was due to music tastes being a diverse and divided question, both differences in genre and taste are deciding factors. That added to hearing impairments and noisy surroundings, there are both pros and cons for compression.

The Theory part is where this project have the most work left, the equations are not optimized and depends on normalization, and the attack and release time are set values in the code. This project was not supposed to dwell to deep into the equations, but continued work would improve the entirety of the project significantly.

When handling sounds in Linux, some small issues were present in the beginning. This was due to the small framework and Linux being a rather unused platform for sound computation. The Raspberry Pi is a hardware that do not have inputs and outputs except for the 3.5mm headphone jack. That means that additional hardware is needed on top of the Raspberry. If a different hardware would have been chosen, perhaps some thing could have been solved without additional hardware, like I/O's and dynamic control. The cost could have possibly gone down some as well.

The results were better than first expected, the only apparent problems was attack and release time, and make-up gain. Make-up gain proved to be a problem that came from the hardware and was therefor worked around with a dynamic ratio instead. This was concluded to be the best way around the problem, even though it did not completely solve it.

Music is a big part of our life, we are susceptible to it wherever we go. It is time to start changing back to the way music was before the loudness war, before the digital age.

Bibliography

Bengtsson, L. & Karlström, B. (2007), *Transformer*, Studentlitteratur.

Blesser, B. (2007), 'The seductive (yet destructive) appeal of loud music', [ONLINE]. [Last Visited 2017-06-19].
URL: *http://www.blesser.net/index.html*

Currens, D. (2011), [ONLINE]. [Last Visited 2017-06-19].
URL: *https://www.quora.com/How-does-the-sound-quality-of-a-same-song-broadcasted-on-FM-Radio-compare-to-MP3-encoded-at-320kbps*

Dynamic Range Database, Californication (n.d.).
URL: *http://dr.loudness-war.info/album/list?artist=Red+Hot+CHili+Peppersalbum=Californica*

Engineering & Wiki, T. H. (Last edited: 13 January 2016), 'Alexander graham bell', [ONLINE]. [Last Visited 2017-06-19].
URL: *http://ethw.org/Alexander$_G$raham$_B$ell*

hardly (2004), 'Cd vs vinyl', [ONLINE]. [Last Visited 2017-06-19].
URL: *https://everything2.com/user/hardly/writeups/CD+vs.+vinyl*

Kysela, J. (1998), 'Advanced linux sound architecture', [ONLINE]. [Last Visited 2017-06-19].
URL: *http://www.alsa-project.org/main/index.php/Main$_P$age*

Mathworks (2017), 'compressor system object', [ONLINE]. [Last Visited 2017-06-19].
URL: *https://se.mathworks.com/help/audio/ref/compressor-class.html*

Matthews, D. (2014), 'Vox', [ONLINE]. [Last Visited 2017-06-19].
URL: *https://www.vox.com/2014/4/19/5626058/vinyls-great-but-its-not-better-than-cds*

Mellor, D. (2006), 'Audio masterclass', [ONLINE]. [Last Visited 2017-06-19].
URL: *http://www.audiomasterclass.com/what-is-the-difference-between-0-db-and-0-dbfs*

Mikemoon (2012), 'Steve hoffman music forum', [ONLINE]. [Last Visited 2017-06-19].
URL: *http://forums.stevehoffman.tv/threads/californication-reissue-cut-by-bernie-grundman.291679/*

Molin, B. (2013), *Analog elektronik*, Studentlitteratur.

Resnikoff, P. (2015), 'Digital music news', [ONLINE]. [Last Visited 2017-06-19].
URL: *http://www.digitalmusicnews.com/2015/12/21/55-of-americans-prefer-their-music-through-computer-speakers/*

Shepherd, I. (2008), 'Mastering media blog', [ONLINE]. [Last Visited 2017-06-19].
URL: *http://mastering-media.blogspot.se/2008/09/metallica-death-magnetic-sounds-better.html*

Shepherd, I. (2015), 'Production advice', [ONLINE]. [Last Visited 2017-06-19].
URL: *http://productionadvice.co.uk/spotify-same-volume-setting/*

Solodovnichenko, D. (Last edited: 25 Mar 2016), 'Jackd audio', [ONLINE]. [Last Visited 2017-06-19].
URL: *https://github.com/jackaudio/jackaudio.github.com/wiki*

Sreedhar, S. (2007), 'The future of music, part one: Tearing down the wall of noise', [ONLINE]. [Last Visited 2017-06-19].
URL: *http://spectrum.ieee.org/computing/software/the-future-of-music*

Studios, F. (n.d.), 'Audio injector', [ONLINE]. [Last Visited 2017-06-19].
URL: *http://www.audioinjector.net/rpi-hat*

Wolf, J. (2011), 'Unsw, school of physics sydney, australia', [ONLINE]. [Last Visited 2017-06-19].
URL: *http://www.animations.physics.unsw.edu.au/jw/dB.htm*

Zölzer, U. (2011), *DAFX: Digital Audio Effects, Second Edition*, John Wiley Sons.