Audio Steganography using Convex Demixing

Master thesis

Yann Schoenenberger
EPFL EE LTS2 – 2014
yann.schoenenberger@epfl.ch

Supervised by:

Pierre Vandergheynst – EPFL
Signal Processing Lab
http://lts2www.epfl.ch

Johan Paratte – EPFL
Signal Processing Lab
http://lts2www.epfl.ch

Abstract

In the first part, we first introduce steganography (in chapter 1) not in the usual context of information security, but as a method to piggyback data on top of some content. We then focus on audio steganography and propose a new steganographic scheme in chapter 2 as well as a model for the noisy, analog communication channel we are considering (in section 3.2). The method we use is based on signal mixing, the science of constructing vectors in a way that their sum can then be split back into the original components. This is presented in chapter 4. The data recovery, or demixing, relies on convex optimization (presented in chapter 5) and some further signal processing detailed in chapter 6.

In the second part we present our proof of concept implementation and show the results of simulation runs that have been made in order to study the properties of the overall system (chapter 7). We study how the different parameters of the system and the communication channel affect the performance of the steganographic scheme. Finally, we draw conclusions (chapter 8) about the findings and suggest (in chapter 9) what next steps could be taken in order to further study this steganographic scheme.

Contents

Acronyms

A/D Analogue to Digital

AWGN Additive White Gaussian Noise

D/A Digital to Analogue

DCT Discrete Cosine Transform

DFT Discrete Fourier Transform

FFT Fast Fourier Transform

HAS Human Auditory System

LTFAT Large Time-Frequency Analysis Toolbox

MDCT Modified Discrete Cosine Transform

RX Receiver

TX Transmitter

WMDCT Windowed Modified Discrete Cosine Transform

WGN White Gaussian Noise

Part I
Problem definition and proposed solution

1 Introduction

1.1 Steganography

This project is about using a new steganographic scheme with a somewhat unconventional aim. We first present what we want to do (in section 1.2 and section 1.4) and how we intend to do this (in chapter 2). Then, we test the ideas with simulations in chapter 7 and draw conclusions about the convex demixing scheme in chapter 8.

1.1.1 Definition

Steganography etymologically means "concealed writing". It is the science of hiding messages such that only entities who know that it is there (typically the sender and the intended recipients) are able to extract the message.

This broad definition covers techniques ranging from using invisible ink on a piece of paper to encoding data in the delays between the transmission of packets on a computer network.

Modern (digital) steganography typically consists of hiding the payload bits in an unrelated underlying data file known as the cover file.

An example would be for the sender to replace the least significant bits describing pixels in an image (the cover file) by some message. This would alter the original image slightly as noise would do and may not even be perceptible at all by a human. To recover the message the recipient would only have to keep the least significant bits and throw out the rest of the file.

We talk about audio steganography when the cover file is an audio file. In this work, we are going to deal with sound augmented with additional data added steganographically.

1.1.2 Security

As the definition suggests, the original application of steganography is in information security. Simply hiding a message was a common way of communicating secret messages.

In recent times however, security through obscurity[1], fell into disuse for state of the art secure communications because of the acceptance of Kerckhoffs's principle[2].

1.1.3 Augmenting data

Nowadays, there remain applications where steganography is still useful.

Digital watermarking, the process of steganographically adding identifying information to a piece of data (usually media files), is used to verify the integrity, authenticity or ownership of the cover file.

[1] Communicating privately through the use of a secret scheme (i.e. steganography)

[2] Kerckhoffs's principle states that the security of a communication system should not rely on the secrecy of its inner workings, but should rely on the secrecy of the cipher key.

Another example is the fact that modern printers add dotted patterns to every printed page. Those dots typically encode timestamps and the printer's serial number. These dots are routinely used by law enforcement agencies in investigations.

In this project, we are not concerned with the security applications of steganography but rather, we want to add metadata to audio recordings in order to make it available alongside the original audio (c.f. section 1.2).

1.2 Intended use case

Imagine having a smartphone application that could listen to the radio with you (thanks to the device's microphone) and display the lyrics of the currently playing song without needing access to the internet nor using complex speech-to-text software.

Alternatively, picture yourself in an elevator where you cannot get any WiFi signal, but there is still a way to get information about the weather or the latest headlines thanks to some data embedded in the cheesy elevator music that plays in the background.

This could be possible if there was a way to extract the relevant data from the audio you are listening to. One way to do this, through steganography, would be to have software able to demix the extra data from the cover audio. Of course that data would have to be mixed in before being played back on speakers.

1.3 System setup

In this project, we consider the system shown in figure 1.3.

There are three basic parts: the transmitter, the channel and the receiver. We make assumptions about the channel in section 3.2 and we design the TX/RX pair that can reliably transmit the message (hidden in the cover file[3]) over the channel.

The receiver simply undoes the steps taken by the transmitter to finally recover the original message. The message is hidden in the cover audio in the `data mixing`[4] step and the payload is extracted in the `data demixing`[5] step. The `payload encoding/decoding` as well as the `error correction encoding/decoding` are discussed in chapter 6

1.4 Goals of the project

Basically, this project is about implementing a proof of concept of a new scheme described in subsection 1.4.1 to study its feasibility and usefulness.

[3] Not represented in figure 1.3
[4] Discussed in chapter
[5] Discussed in chapter

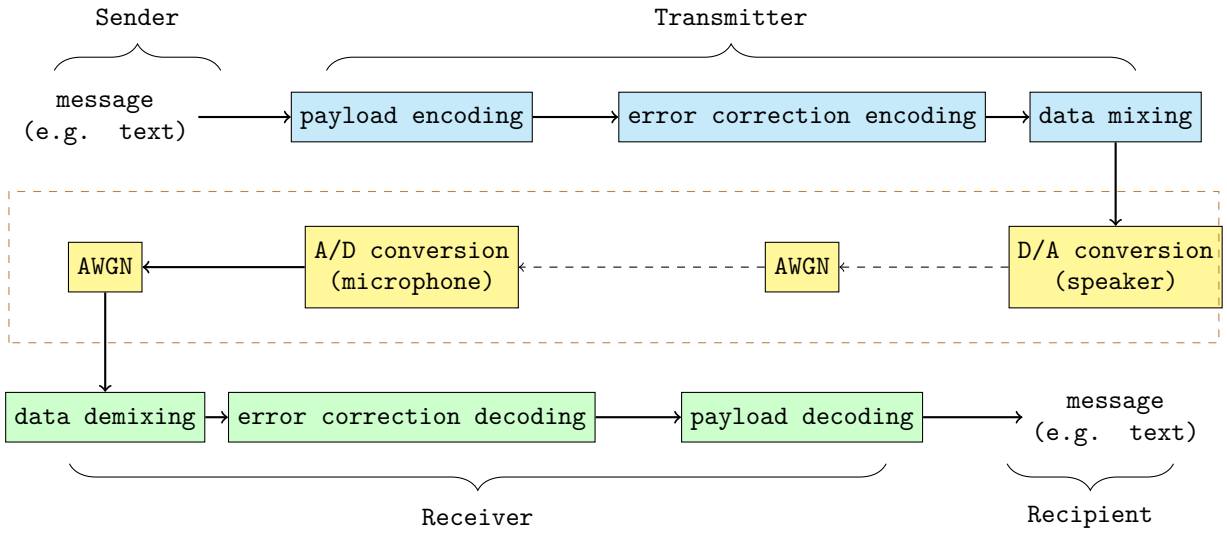

Figure 1: Complete chain of (sub-)systems the message goes through from the sender to the recipient. Dashed (resp. solid) arrows are links in an analog (resp. digital) medium. The part enclosed in a dashed rectangle is our working model of the channel between the TX and the RX.

1.4.1 Proof of concept of a new scheme

In this work, we propose and discuss a new method that allows one to steganographically hide data in a some signal in such a way that the signals can later be demixed using the solution to a convex optimisation problem. Professor Pierre Vandergheynst deserves all the credit for the intuition, idea and mathematical formulation of this steganographic scheme.
There is an extensive literature on various steganographic schemes. One could mention [7], [4], [1] or [6] to name a few, but our new demixing method seems to be new.

Our approach is different. We present the convex demixing scheme in chapter 2.

1.4.2 Check how the scheme handles D/A conversion

In the proposed application, we would like to be able to recover the hidden data while listening to the cover audio.[6] This implies that there must be a point where the file's bits are converted to analog sound waves and then back to digital bits (e.g. using a smartphone's microphone) for analysis and data extraction.

Thus, the cover file and the payload should be separable even after D/A and A/D conversion.

This is an important goal, because there is very little literature available on the subject. A rare example is [15] and even there the achieved bitrate of the steganographic method that survives D/A conversion is very low.

[6] Check section 1.2 to see why.

1.4.3 Study the different parameters of the setup

Obviously we want to be able to add as much data as possible to the cover audio. However, we do not want to degrade the audio too much. Ideally, the changes should not be audible.

There are several parameters that can be tweaked.

The ones we are going to test in our simulations are:

- Chunk size[7]
- Bitrate of the steganographically added audio
- The energy of the payload signal

We discuss the findings in chapter 7.
Moreover, we are going to test how resistant the scheme is to additive noise and resampling, both of which occur in a realistic setup.

[7]The captured audio will be split in smaller pieces and processed chunk by chunk.

2 Steganographic scheme

Now that we have a clear context and goals in mind, we need to look at how we want to piggyback our payload on the cover audio file.

Consider the communication system in figure 1.3. We are now concerned with the `data mixing` and `data demixing`

2.1 Concealing the payload

Consider our cover, an audio file: a vector $\mathbf{x}$[8] of size N. The entries of $\mathbf{x}$ are real-valued samples in the interval $[0, 1]$. We can express $\mathbf{x} = \Phi\mathbf{b}_x$[9] for some $\mathbf{b}_x$ where Φ is the matrix $N \times N$ corresponding to an orthogonal basis (e.g. DFT basis) and the entries of $\mathbf{b}_x$ are the coefficients in that basis (e.g. the Fourier coefficients).

One can consider, analogously the size N vector $\mathbf{y} = \Psi\mathbf{b_y}$[10] as the payload we want to mix in $\mathbf{x}$. Like Φ, Ψ is another orthogonal $N \times N$ base matrix.

We then simply transmit $\mathbf{s} = \mathbf{x} + \mathbf{y}$ instead of $\mathbf{x}$ over the channel. This is the data mixing step and $\mathbf{s}$ is called a mixture.

It can be shown that if Φ and Ψ[11] are optimally incoherent and thus maximally orthogonal, we can recover $\mathbf{b_x}$ and $\mathbf{b_y}$ (and thus separate $\mathbf{x}$ and $\mathbf{y}$) given $\mathbf{s}$.

2.2 Separating payload and cover audio

To recover $\mathbf{x}$ and $\mathbf{y}$, we can first compute $\mathbf{b_x}$ and $\mathbf{b_y}$. Given $\mathbf{s}$, we need to solve

$$\underset{\mathbf{b}_x, \mathbf{b}_y}{\operatorname{argmin}} ||\mathbf{s} - \Phi\mathbf{b}_x - \Psi\mathbf{b}_y||_2^2 + \mu_1||\mathbf{b}_x||_1 + \mu_2||\mathbf{b}_y||_1$$

where μ_1 and μ_2 are regularisation parameters. Solving this convex optimization problem is the demixing step. Such techniques are disscussed in [16], [8] and [12].

Note that what comes out of the communication channel is not exactly $\mathbf{s}$ but rather $A\mathbf{s}$[12] an altered version of $\mathbf{s}$ as is discussed in chapter 3.

[8] Note that the notation introduced in sections 2.1 and 2.2 will be used consistently throughout this report

[9] $\mathbf{b}_x$ is also a size N vector.

[10] $\mathbf{b}_y$ is also a size N vector.

[11] Note that Φ and Ψ are known parameters.

[12] A is a matrix that defines some kind of transformation on $\mathbf{s}$. In this way, it is possible to model a lot of things, such as additive noise, that can happen to $\mathbf{s}$.

3 Channel

We now take a look at the communication channel between the transmitter and the receiver in order to make a mathematical model that we can use in our simulations.

3.1 Intended real life usage

The scenarios discussed in section 1.2 describe basically the same situation.

A person listens to some music (the cover) but has the possibility to use software to extract additional data (that has been steganographically added) from that same audio. The music is broadcast by some sort of loudspeaker and the device running the software has to somehow capture that audio (e.g with a microphone) for analysis and payload extraction.

3.2 Model

The medium between the loudspeaker and the microphone (as well as the Human Auditory System (HAS), which we mention in at the end of this report in chapter 9) is analog and noisy. In this section, we will justify why we simulate the channel by simply resampling the signal and adding white gaussian noise.

3.2.1 D/A conversion

The signal is always digital in every system up to the loudspeaker where it is turned into an analog sound wave. On the other end, the microphone samples it and turns it back to a digital signal.

In other words, the loudspeaker performs a D/A conversion and the microphone performs an A/D conversion. Digital music is typically sampled at 44100 Hz and typical microphones sample sound at 8000 Hz (32000 Hz for high-end models).

Because of the properties of interpolation, the whole process of converting to an analog signal can be approximated for simulation purposes by upsampling the signal. The sampling rate can be made arbitrarily high and thus the analog signal approximation can be made as good as needed.

In that framework, the sampling of that analog signal done by the microphone is equivalent to downsampling the upsampled version of the signal.

The overall chain of events is therefore equivalent to resampling the original signal to the microphone's sampling rate which is typically lower than the sampling rate of the original audio and thus incurs some information loss.

This downsampling and then resampling to the original audio's sampling rate is all that is needed for an accurate model and thus what we are going to use in the simulation.

3.2.2 Noise

The modelling of the channel is not complete without a consideration of the noise.

Noise may degrade the signal in any system handling the analog signal (i.e. at both the D/A and A/D conversions), so we have to consider AWGN in a few places. Fortunately, thanks to properties of WGN, we know that we can consider all of the noise to be added in one place. Thus, in our simulation, we will simply add WGN after the resampling of the signal. The standard deviation of the noise is one of the parameters that affects the performance of the scheme and that can be set. We look at it in chapter 7.

4 Mixing

In this chapter, we assume that we have two vectors that we want to process in a way that allows us to add them and still be able to separate them when needed. Demixing is the process of getting back the two operands of the mixing process and is discussed in chapter 5. Note that all the variables in this chapter are defined as in section 2.1.

4.1 Problem definition

We assume we are given two vectors $\mathbf{b_x}$ and $\mathbf{b_y}$. We want to design two matrices Φ and Ψ such that

$$\mathbf{s} = \Phi\mathbf{b_x} + \Psi\mathbf{b_y}$$

can be demixed.[13]

In the context of our specific application and in order to have a simple consistent notation, we can assume that $\mathbf{x}$ is always the vector of samples of the cover audio and thus $\mathbf{b_x}$ is its coefficients in the domain of which Φ is a basis.

We can also assume that $\mathbf{b_y}$ is a vector initialised with all zeroes to which we add a few values to hold the payload. The specific way in which we do this is by turning the message into a bitstring and then replace every occurrence of 0 by -1. Thus, we end up with a vector which has elements in $\{-1, 1\}$ instead of $\{0, 1\}$. Let's call this modified bitstring $\mathbf{m}$. We then change the zero vector $\mathbf{b_y}$ to hold the elements of $\mathbf{m}$ in order. There are many ways to do this. A ans good simple way to do it is to place the elements of $\mathbf{m}$ at regular intervals to form $\mathbf{b_y}$. For example if the bitstring is 1001 and $\mathbf{b_y}$ needs to be of length 12 , then $\mathbf{m} = (1, -1, -1, 1)$ and $\mathbf{b_y} = (0, 0, 1, 0, 0, -1, 0, 0, -1, 0, 01)$.

Back to the problem definition, the idea is to project $\mathbf{b_x}$ on one vector space and $\mathbf{b_y}$ on an orthogonal one. Then, the demixing step would be a matter of projecting $\mathbf{s}$ on those vector spaces.

Demixing can be done using convex optimization (see section 5) if $\mathbf{b_x}$ and $\mathbf{b_y}$ are sparse and the basis Φ and Ψ are mutually incoherent. We thus have to construct a system that will guarantee these hypothesis. The basis Φ is chosen so that the representation of the audio signal $\mathbf{x}$ in it is sparse (i.e. $\|\Phi^{-1}x\|_1$ is small[14]).

Afterwards, we select an orthogonal basis Ψ optimally incoherent to Φ. This way, thanks to the signals being sparse, as we will see in chapter 5, we are guaranteed that the demixing can be done.

We now explore two possibilities. First we look at Φ being the Discrete Fourier Transform (DFT) basis in section 4.2 and then after discussing some of the shortcomings of choosing this basis, we look at the basis for the Windowed Modified Discrete Cosine Transform (WMDCT) in section 4.3. Afterwards, in chapter 7 we will compare and contrast both approaches.

[13] In practice, we are given $\mathbf{x}$, we define Φ such that $\mathbf{x} = \Phi\mathbf{b}_x$ and then we have to design Ψ as a function of Φ.

[14] Actually, $\|\Phi^{-1}x\|_0$ should be small, but for our purpose the relaxed condition on $\|\Phi^{-1}x\|_1$ is good enough.

4.2 Discrete Fourier Transform

Consider the N-point DFT. We can define Φ to be the DFT matrix. This matrix defines an orthogonal basis and we can very simply derive a good Ψ. It can be shown that $\Psi = D\Phi$ with D a diagonal matrix with the elements on the diagonal being randomly selected in $\{-1, 1\}$, Ψ and Φ are optimally incoherent with a very high probability. A basis Ψ is optimally incoherent to a basis Φ if the coherence between Φ and Φ is minimal.

Moreover, if we consider the actual implementation of this scheme, this approach is computationally very efficient. Since Φ and Ψ are never used on their own, but only in a multiplication with a vector, we do not need to ever define or store them explicitly. We can replace every occurrence of $\Phi\mathbf{v}$ for any vector $\mathbf{v}$ by $\hat{\mathbf{v}}$ which is the DFT of $\mathbf{v}$ [15].

For Ψ, we only need to compute and store the values on the diagonal of D. This is simply a vector $\mathbf{d}$ (of size N) with the entries randomly selected in $\{-1, 1\}$. This way, every occurence of $\Phi\mathbf{v}$ for any vector $\mathbf{v}$ can be replaced by $\mathbf{d} \cdot \hat{\mathbf{v}}$

One problem of defining Φ this way is that, for sound in general, the Fourier representation is not really sparse. Indeed sound often covers a wide part of the spectrum. Defining Φ in this way may still be good enough, but to really ensure the sparsity of the representation of the audio, we will now take a look at another transform.

4.3 Windowed Modified Discrete Cosine Transform

We now consider the Windowed Modified Discrete Cosine Transform (WMDCT). The WMDCT is a lapped[16] version of the DCT[17].

The problem with using the DFT is that typical audio covers, over time, a wide variety of frequencies and thus the DFT may not really be a sparse representation of it.

To remediate to this situation, we can consider the Gabor transform. The Gabor transform analyses signals in time and frequency simultaneously by translating and modulating a window. This transform computes a mathematical score of a sound. Moreover, the Gabor transform can be sampled in time and in frequency. We do not compute the Gabor coefficient for every shift and modulation possible but only for M frequencies and a gap of a time units. The WMDCT is a particular case of the Discrete Gabor transform that is sampled using $a = M$ and a window with a sufficient overlap. If the window is tight, this leads to an orthogonal basis which we can use as our Φ.

The expression of c, the WMDCT of f, with window g and M bands ($m = 0, \ldots, M-1$ and $n = 0, \ldots, N-1$) turns out to be:

If $m + n$ is even:

$$c\,(m+1, n+1) = \sqrt{2} \sum_{l=0}^{L-1} f(l+1) \cos\left(\frac{\pi}{M}\left(m + \frac{1}{2}\right) l + \frac{\pi}{4}\right) g(l - nM + 1)$$

[15] The Fast Fourier Transform (FFT) is a well known and efficient algorithm to compute the DFT.

[16] A lapped transform is a block transform where the basis functions of the transform overlap to the adjacent blocks but still has the same number of coefficients as we would obtain from a non-overlapping block transform.

[17] An often used transform in digital signal processing and even watermarking.

If $m+n$ is odd:

$$c\left(m+1,n+1\right)=\sqrt{2}\sum_{l=0}^{L-1}f(l+1)\sin\left(\frac{\pi}{M}\left(m+\frac{1}{2}\right)l+\frac{\pi}{4}\right)g(l-nM+1)$$

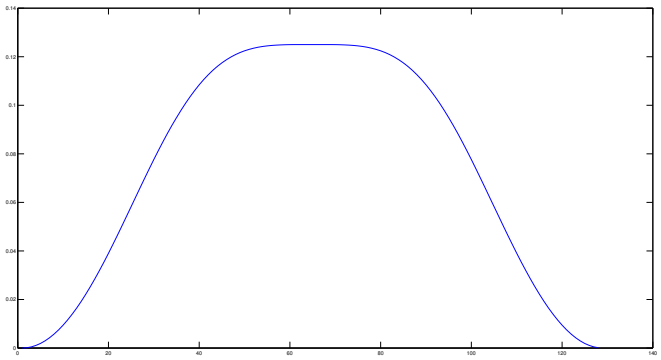

Figure 2: Example LTFAT's standard `itersine` window (of length 128) with $a=M=64$.

For our purpose, we can select $a=M=64$ and the itersine window[18] of size 128. Because itersine is tight, we get a basis.

Like in the case of the DFT, the WMDCT matrix gives us an orthogonal basis in which to express our vectors and experience shows that for audio signals the Gabor/WMDCT representation is usually more sparse than the DFT.

Similarly to what we did in the case of the DFT, in order to avoid computationally expensive multiplications by Φ we could use the actual WMDCT algorithm. However, the construction of Ψ proves to be more challenging. We can not use a nice trick as we did with the DFT, we actually have to compute Ψ as a function of Φ. To do this, we use the iterative algorithm described in [11]. This is done only once and can be done offline so although there is a significant computational cost[19], it can be made so that it does not hinder the performances of an actual realtime implementation.

However, Ψ still has to be stored and there might be insufficient memory available on the system performing those computations, typically embedded devices. This problem may arise in case one wants to process large chunks of audio.[20]

[18] Introduced in [14] and shown in figure 2

[19] Each iteration of the algorithm requires matrix multiplications and inversions.

[20] The size of Φ and Ψ grows as the square of the size of the signal chunk. This can result in significant memory requirements.

5 Demixing

In chapter 4, we have defined two vectors $\mathbf{x}$ and $\mathbf{y}$ that we added up to form $\mathbf{s}$. The way $\mathbf{x}$ and $\mathbf{y}$ were constructed (using Φ and Ψ) makes it possible to recover $\mathbf{x}$ and $\mathbf{y}$ from their sum $\mathbf{s}$. Note that all the variables in this chapter are defined as in section 2.1.

5.1 Problem definition

We are now dealing with the problem of undoing the mixing discussed in chapter 4. We assume we know Φ and Ψ and want to recover $\mathbf{x}$ and $\mathbf{y}$[21] given $\mathbf{s}$[22].

We are going to use convex optimisation to recover $\mathbf{b_x}$ and $\mathbf{b_y}$.

5.2 Convex optimisation

We consider the following convex optimization problem:

$$\underset{\mathbf{b}_x,\mathbf{b}_y}{\operatorname{argmin}} ||\mathbf{s} - \Phi\mathbf{b}_x - \Psi\mathbf{b}_y||_2^2 + \mu_1||\mathbf{b}_x||_1 + \mu_2||\mathbf{b}_y||_1$$

The objective function is composed of three terms:

- $||\mathbf{s} - \Phi\mathbf{b}_x - \Psi\mathbf{b}_y||_2^2$ is the data fidelity term
- $||\mathbf{b}_x||_1$ is the sparsity condition on $\mathbf{b}_x$[23].
- $||\mathbf{b}_y||_1$ is the sparsity condition on $\mathbf{b}_y$[24].

Solving this will give us $\mathbf{b_x}$ and $\mathbf{b_y}$ within a very small margin of error. The acceptable margin of error can be set by tweaking μ_1 and μ_2 which are the regularisation parameters of the optimisation problem.

We can formulate this optimization problem in another way:

$$\underset{\mathbf{b}_x,\mathbf{b}_y}{\operatorname{argmin}} ||\mathbf{b}_x||_1 + \nu||\mathbf{b}_y||_1$$

for some weight parameter ν and with constraint:

$$||\mathbf{s} - (\Phi\mathbf{b}_x + \Psi\mathbf{b}_y)||_2^2 \leq \epsilon$$

for some small positive ϵ which is the energy of the noise[25]. This is needed because in the actual practical setup as shown in figure 1.3 and described in section 2.2 however, we are not getting exactly $\mathbf{s}$ as constructed in the mixing stage, but $\hat{\mathbf{s}} = \mathbf{x}+\mathbf{y}+noise$.

In order to solve the convex optimisation problem, we use the generalized forward backward method [9].

[21] Equivalently, we can recover $\mathbf{b_x}$ and $\mathbf{b_y}$ (and thus recompute $\mathbf{x} = \Psi\mathbf{b_x}$ and $\mathbf{y} = \Psi\mathbf{b_y}$) since we know Φ and Ψ.

[22] In practice, we are given a slightly altered version of $\mathbf{s}$ and we will only be able to get approximate values for $\mathbf{b_x}$ and $\mathbf{b_y}$ as we will see. Our model will take this into account.

[23] We picked a basis in which $\mathbf{b}_x$ is sparse.

[24] We constructed $\mathbf{b}_y$ to be sparse.

[25] No noise means $\epsilon = 0$ and we recover $\mathbf{b_x}$ and $\mathbf{b_y}$ exactly.

6 Data extraction

Now that we have seen how we can recover approximate versions of $\mathbf{x} = \Psi\mathbf{b}_x$ and $\mathbf{y} = \Psi\mathbf{b}_y$, we want to extract our hidden payload. Remember that $\mathbf{x}$ is the cover audio, so now we are only concerned with $\mathbf{b_y}$ which contains the steganographically hidden data. In figure 1.3, this data extraction step is still considered part of the demixing process[26].

6.1 Problem definition

At this stage, the vectors have been demixed, we are given $\mathbf{b_y}$ and we want to recover the bitstring that constitutes our original message. We are thus concerned with the `error correction decoding` and `paylod decoding` systems of the receiver shown in figure 1.3. We first need to extract the piggybacked bitstring from $\mathbf{b_y}$ and then decode it (in the source coding sense) to recover the message.

Given how $\mathbf{b_y}$ is defined. We have to figure out where the data bits are and what their value is. In other words, given a vector of samples, we want to map each sample to a value in $\{-1, 0, 1\}$. Then we strip all the 0s[27] and replace the -1s by 0. At this point, we are left with an actual bitstring.

This bitstring has now to be decoded, in the source coding sense and we are finally left with the original message.

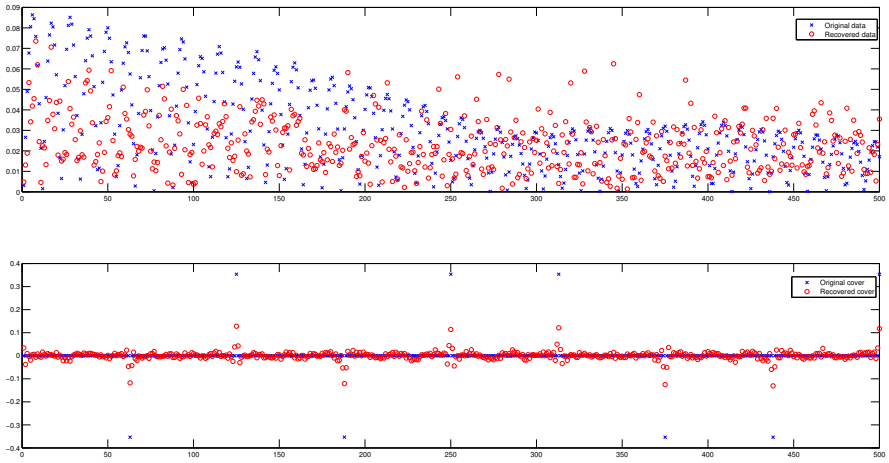

Figure 3: Typical example of absolute value of the cover audio samples (above) and payload vector $\mathbf{b_y}$ (below). Crosses (blue) are the original values and circles (red) are the recovered ones at demixing. The x-axis denotes the sample number and the y-axis denotes the sample value. Unless specifically noted, this arangement, notation and color scheme is consistent throughout this report.

Looking at the bottom graph in figure 3, what needs to happen is pretty straightforward.

[26] We point this out, because usually, strictly speaking, the demixing step is only about separating $\mathbf{x}$ and $\mathbf{y}$.

[27] At this point we are left with a shorter vector with entries in $\{-1, 1\}$

We need to recognize that the payload bits are in positions $64, 128, 192, 256, 320, 384, 448$ and 512 and that the original vector is $(-1, 1, -1, 1, 1, -1, -1, 1)$ which maps to the bitstring 01011001 which maps to some message[28].

6.2 Assumptions

Having extra knowledge about how the message is embedded in $\mathbf{y}$ can help dramatically in the design of a good extraction algorithm.

6.2.1 Known number of bits

If we know the number of bits that are embedded in $\mathbf{b_y}$, we can certainly devise an alogrithm that takes advantage of this information in order to better extract the payload than any algorithm that has no assumption about the number of bits embedded in $\mathbf{b_y}$. Typically, we can imagine an iterative algorithm that somehow find the most likely place to have a bit, removes it from $\mathbf{b_y}$ and reiterates until the appropriate number of bits has been processed.

The main shortcoming of this approach is that in the case of an actual deployment of the method, it would be very difficult to change the bitrate even if an improvement in another part of the system makes it possible to achieve a higher payload bitrate. Transmitting ahead of time the number of bits per chunk does not solve or change the problem, but only moves it in another part of the system. Thus, the update of the expected number of bits per chunk would have to be made offline which is inelegant to say the least.

That is why, in this project, we consider that we do not know the number of payload bits present in any given chunk.

6.2.2 Known position of the bits

Notice that, as discussed in chapter 4, we have arbitrarily decided to evenly space out the payload bits in $\mathbf{b_y}$. We could have done it differently. For instance, we could select randomly the indexes where we embed those bits (but still preserve the order of the bits) or any other way.

Of course, knowing the spacing between two consecutive payload bits gives us a lot of information about where to expect to find all the bits even if we manage to find only one on the first try.

Ideally, we would like to have the bits randomly scattered accross $\mathbf{b_y}$ and still be able to perfectly recover them[29]. However, for simplicity and visual clarity we still chose to spread them out evenly and maximally as it does not seem to be too bad of a constraint.

6.3 Naive approach: Thresholding

The first method that we considered to find where the bits are in $\mathbf{b_y}$ is very straightforward. The intuition comes from the histogram on figure 4. We clearly see that most samples are very close to 0, but there are two distinct side lobes. Those correspond to where there is a payload bit equal to 1 (resp. 0) and we can detect the thresholds very easily by starting

[28] In this case, it is simply the ASCII representation of the letter `Y`.

[29] This is not really required, nor important. It is just a more general case.

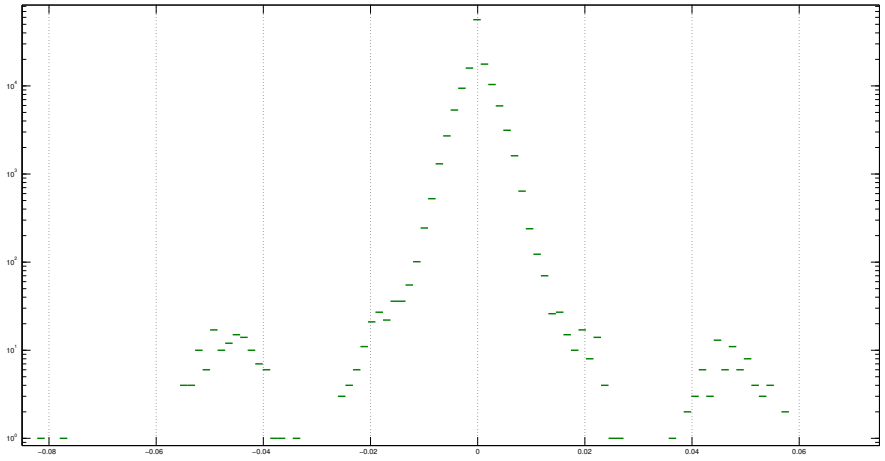

Figure 4: Histogram of the sample values of $\mathbf{b_y}$. The thresholds are at about 0.03 and -0.03

at 0 on the x-axis of the histogram and moving right (resp. left). Indeed, looking at every successive value of the histogram, when we hit 0 samples and then detect an increase, we can assert that we are at the threshold (at the point right before the increase). This allows us to split the samples into three categories, those mapped to -1, 0 and 1 and we can proceed with the extraction as discussed in section 6.1.

In some cases, like the one shown on figure 5, this method works very well.

6.4 Robust approach: Peak detection

In many cases however, the thresholding method proves too naive and does not work at all (as shown in figure 6).
It is often very easy for a human to tell where the bits are, just by looking at outliers in an otherwise fairly smooth signal. In this particular instance, our inutition tells us that an algorithm should be able to do the same.

The first approach we consiedered was to subtract some moving average from every sample and then run the thresholding algorithm on this smoothed signal. None of the tests were conclusive.

The second approach was to try an compute for every sample how high (resp. low[30]) is a local maximum (resp. minimum) with respect to the adjacent valleys (resp. peaks).

This was done[31] by summing the difference between two adjacent samples all the way from the two neighbouring local minima to the peak. The intuition was that peaks where payload bits are encoded would get a very high score however this did not give satisfactory results either. Normalising with respect to the number of samples it took to reach a peak

[30] In case of peaks at -1

[31] For simplicity, we only consider peaks. For valleys, the case is symmetrical.

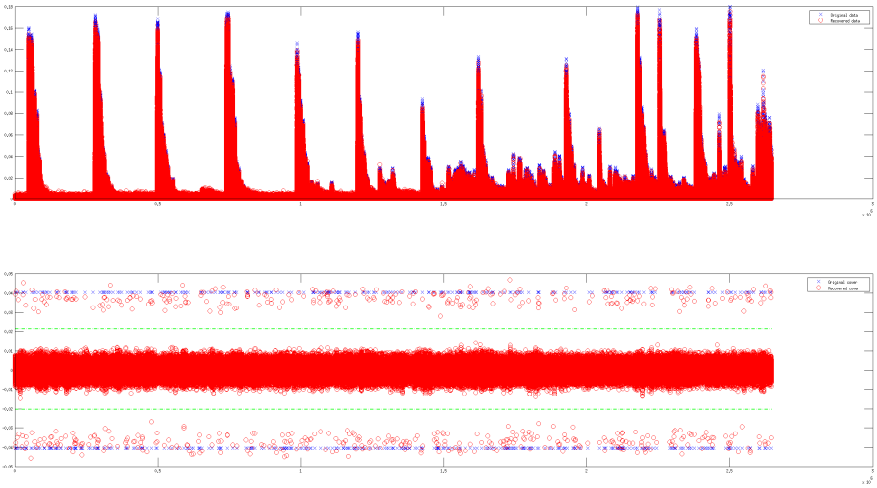

Figure 5: Example were the payload is successfully recovered using thresholding with the DFT mixing method. The dashed (green) lines denote the computed threshold values. In this particular case, the recovered bitstring exactly matches the original message without false positives nor false negatives.

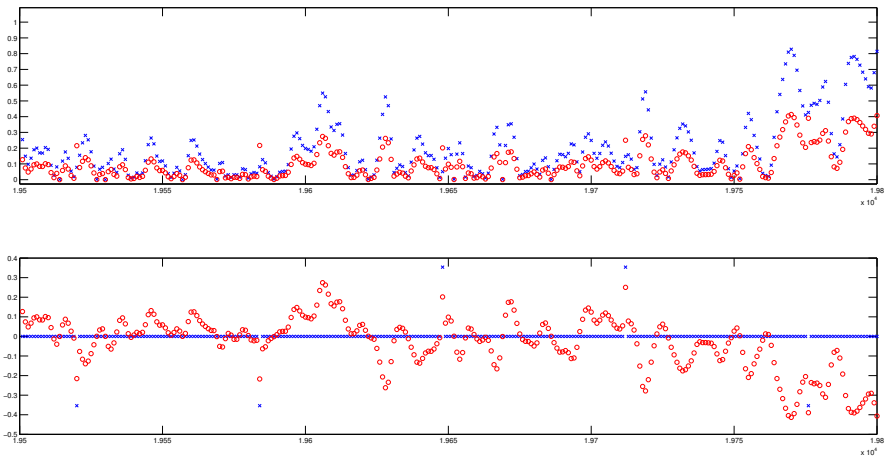

Figure 6: Example were the payload is mixed in with the DFT mixing method. Our thresholding method unfortunately does not work. There is generally too much deviation from 0 for the samples that do not hold a payload bit. However, it is quite easy to see that the signal exhibits a kind of smoothness except where payload bits are encoded. At those places we can see a peak.

from a valley (i.e. how steep of a climb it was up to the peak) seemed marginally better but did not yield any promising results.

Unfortunately, none of these ideas proved very effective. Integrating an effective peak detection mechanism may prove to be the single most effective step one can take in order to achieve a truly usable convex demixing steganographic scheme as described in this report. We suggest doing so in section .

6.5 Insertion-Deletion channel

If data extraction works perfectly everytime, the $\mathbf{b_y}$ we recover exactly matches the $\mathbf{b_y}$ that is constructed in the mixing stage, we do not need any error correction codes. However, as we have seen, errors can occur.

There are several types of errors that can occur when extracting the bitstring from $\mathbf{b_y}$. For every sample:

- With some probability, the data extraction algorithm thinks there is a bit where actually the sample is just a noisy 0 (false positive).
- With some probability, a payload bit is missed (false negative) because the algorithm computed a 0 when actually a payload bit (1 or -1) was present.
- With some probability, a bit flip occurs. The data extraction algorithm computes a 1 where actually a -1 is present or vice versa.

In this setup, a false positive is called an insertion, a false negative is a deletion and a bit flip is a deletion followed by an insertion (or vice-versa, equivalently). We are thus dealing with an insertion-deletion channel.

This is a fairly unknown and challenging communication channel. There is some literature such as [5] on the theory, but the exact capacity is still unknown although there are known bounds [3]. Reliable communication over such a channel can be achieved through the use of marker codes as discussed in [2] and [10].

Part II
Tests and analysis

7 Experimental data

We now have a theoretical understanding of the whole system. The next step is to empirically look at how the setup behaves under different conditions.

7.1 Simulation run and observations

Here is the detail of a complete run of the proof of concept software. In this instance, Φ is the DFT matrix.

First of all, it must be noted that the audio is processed iteratively. The whole audio cover is cut into fixed-length vectors (called chunks). Then every chunk is processed independently and the results are concatenated. This allows for independent treatment of short pieces of the audio. Thus, in a real setup, one would not have to wait a long time to capture the audio before being able to recover the data. It is also more efficient computationally. In this instance we have 4 chunks of 500 samples each and a 32 bit message[32].

The first step is to compute $\mathbf{s}$ (the payload mixed in with the cover audio) which is shown in figure 7. Note that when Ψ is optimally incoherent with the DFT matrix, we essentially have a spread spectrum embedding.

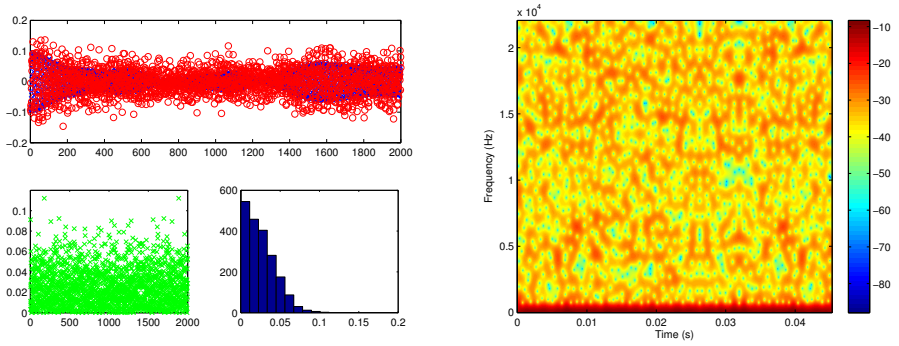

Figure 7: In the upper left corner we have the cover audio samples $\mathbf{x}$ (blue crosses) and $\mathbf{s}$ the altered version of it once the payload has been added (red circles). In the lower left, we have a graph of $|\mathbf{s} - \mathbf{x}|$ (green crosses) as well as a histogram of those values (lower middle). The graph on the right is the spectrogram of the embedded payload.

In this example, for the sake of clarity of the graphs, we assumed the use of a good microphone that samples sound at 32000 Hz and very little noise. We have AWGN with standard deviation $\sigma = 0.02$.

[32] At 44100 Hz, this corresponds to more than 2.8 kilobits per second.

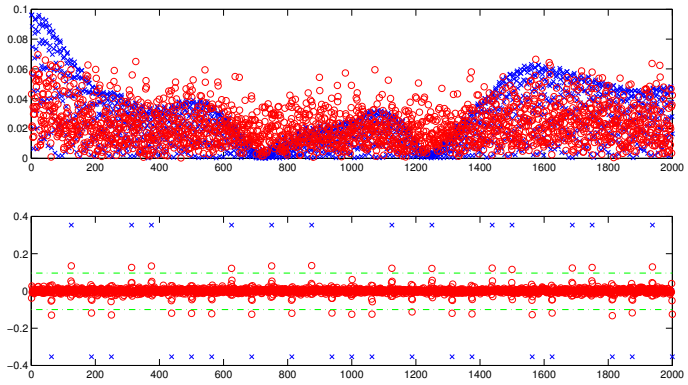

Figure 8: The graph interpretation is the same as usual with both vectors before and after demixing. In this case we see that the thresholding (green dashed lines) manages to recover all the data perfectly.

We can run this setup with different parameters as many times as needed in order to collect data about how changing the individual parameters affects the whole steganographic scheme.

7.2 Experimental setup

In our simulation setup, we have defined a baseline configuration and we always alter the parameters just one at a time.
We test each configuration with both the DFT and WMDCT mixing methods discussed in chapter 4 on three audio signals, one with very calm music, one with average music and one with very energetic and loud music. The spectrograms are shown in figure 9.

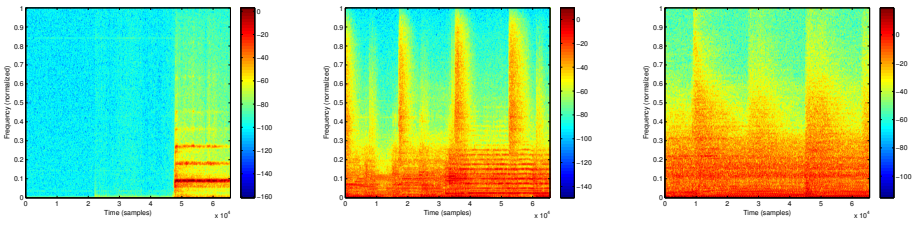

Figure 9: Spectrograms of our cover files (all sampled at 44100 Hz). On the left, the very calm beginning of Pink Floyd's Echoes. In the middle, Pink Floyd's Echoes 10 seconds into the song. On the right, Nirvana's very energetic and loud Anorexorcist 60 seconds after the beginning.

The baseline configuration for all parameters is as follows.

- We assume an ideal channel with no noise and no resampling of the audio.
- We run the procedure on 128 chunks of 512 samples each for a total of 65536 audio samples.

- The payload bitrate is 1 payload bit for every 64 audio samples of the cover (i.e. 8 bits or 1 byte per chunk for a total of 1024 bits or 128 bytes).

- The energy of the payload signal is not scaled.

We always alter one of these values, the rest are kept as described here.

7.3 Changing channel parameters

We are now discussing the simulation data gathered over hundreds of runs in as many different setups with every time only one variable being altered. In this section, we look at channel parameters, those that model the communication channel. In the next section we will look at system parameters that can be used to fine tune the performance of the whole system.

Notes on how to interpret the graphs in this section:

Since all configurations have been tested on three different cover songs, every figure comes with six graphs: three pairs. The leftmost pair of graphs depicts the data gathered on the calm audio. The pair in the middle depicts the data gathered on the medium audio. Finally, the rightmost pair depicts the data gathered on the loud audio.

Now, for each of these pairs, the graph on the right hand side corresponds to a setup where Φ is the DFT matrix. The left hand side graph corresponds to a setup where Φ is the WMDCT matrix. Finally, in each graph, the (blue) crosses are the normalised mean absolute value of the samples during the given run. The (red) stars are the normalised maximal absolute value of the samples during the given run. The coordinate on the x axis is the actual value being tested for that data point. These average sample absolute value and maximal sample absolute value measurements are very relevant to us since the average sample absolute value tells us something about the payload vector $\mathbf{b_y}$ where its entries should be 0 and the maximal sample absolute value measurements tells us something about how prominent the peaks are.

If the difference between the average sample absolute value and the maximal sample absolute value gets smaller it gets harder to detect peaks and thus recover the payload bits, making it much more likely to have insertions and deletions (in the insertion-deletion channel sense).

7.3.1 Altering noise levels

We are interested in looking at the effect of changing the value of the standard deviation of the AWGN.

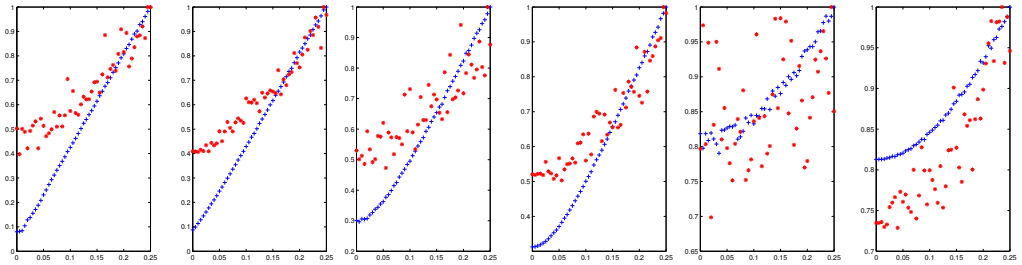

Figure 10: Measurements for different values of σ, the standard deviation of the WGN.

We see in figure 10, that the effect of adding more and more noise has a very predictable impact on the average sample absolute value (the blue crosses), which is itself some measure of the noise in the payload vector $\mathbf{b_y}$. The change is clearly quadratic in σ for the louder pieces of music and resembles a straight line for the quietest piece.

The maximal sample absolute value (the red stars), although it is harder to see, seems to follow the same quadratic growth pattern but at a slower pace. This means that the peaks get harder and harder to detect the noisier it gets, which makes perfect sense.

Generally speaking, the DFT measurements seem themselves noisier than the WMDCT measurements and this is indeed the case. This trend is consistent across all experiments. We discuss this apparent difference between the DFT and WMDCT methods in section 8.1.

7.3.2 Altering the resampling factor

We now look at the resampling factor. This is the parameter that basically models how much the microphone degrades the signal just prior to demixing.

If our audio is originally sampled at 44100 Hz and the microphone samples at 8000 Hz, then our resampling factor for that particular data point is $8000/44100 \approx 0.1814$ (which is a unitless ratio).

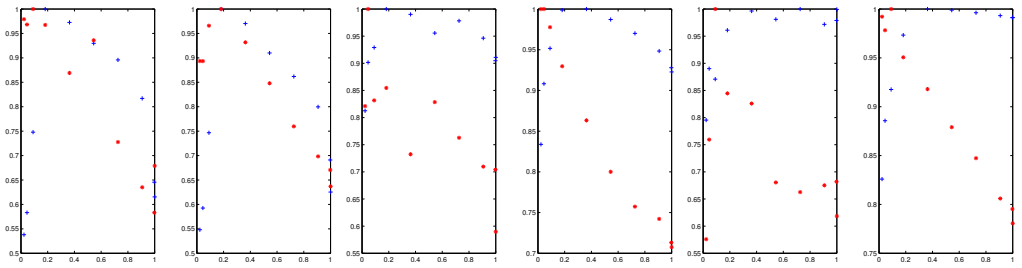

Figure 11: Measurements for different sampling rates of the microphone. The cover audio is always originally music sampled at 44100 Hz.

Interestingly, the mean absolute value of the samples first grows with with the resampling factor, which makes sense since a very low resampling factor basically just flattens the signal. After a point though, faster resampling seems to hurt the performance. This is the case because subsampling effectively removes some of the high frequency noise.

The presence of those high frequency components might also explain why the maximal absolute value of the samples decreases as the resampling factor grows.

From this simulation data and assuming the cover audio is sampled at 44100 Hz, the sweet spot seems to be to resample at about 11000 Hz.

7.4 Changing system parameters

Now we look a three more parameters. These are parameters that we have control over and that we can fine tune to get the most performance[33] out of our implementation.

7.4.1 Altering the chunk size

We do not process an arbitrarily long piece of audio. Imagine having to wait a long time to capture some audio and then wait some more to process it. As discussed in the first part of this report, we do the processing chunk by chunk using a sufficiently big chunk size so that we can actually piggyback some data in each chunk[34]. On the other hand the chunk size should be small enough so that one can do the processing as close to real time as possible. We do not want to have to wait a long time to gather many samples before being able to extract some data.

Note that here we simply want to see the impact of changing the chunk size and nothing else. We adapt the length of the payload message in each run such that the payload bitrate (i.e. the number of payload bits per audio sample) is kept constant. In this simulation, the payload bitrate is kept at 1/64.

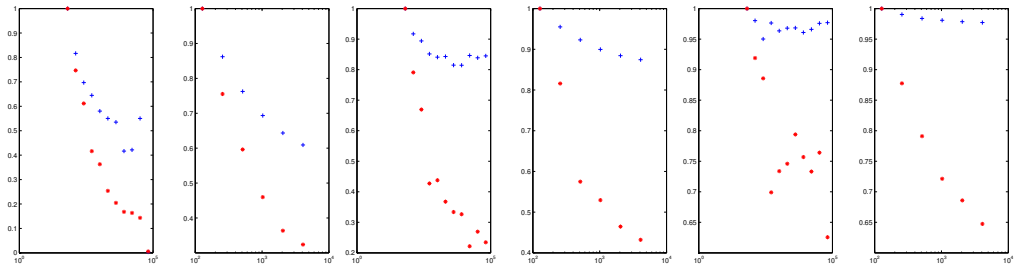

Figure 12: Measurements for different chunk sizes (notice the log scale) with constant payload bitrate.

We see that both the maximal absolute value and mean absolute value of the samples decrease as the chunk size grows. This is an interesting observation and we conjecture that it has something to do with the dimensionality of the vector spaces we are working with that becomes too large. The high dimensionality of the vector space implies that vectors are usually very close to the basis vectors and this results in lower values. Read [13], specifically section 1.3.5 for more on this.

Note that in louder music and thus noisier data, the average value of the samples decreases more slowly. This is due to the fact that the different chunk sizes are a much less important factor than the noise in the signal that comes from the louder audio.

[33] Highest possible payload throughput while staying (mostly) inaudible

[34] Piggybacking even 1 bit in a very small chunk, necessitates to really significantly alter the signal. This in turn implies and thus a very high throughput, but also a very degraded cover signal.

7.4.2 Altering the throughput

This simulation is very straight forward. We simply try to increase the number of payload bits per sample of cover audio.

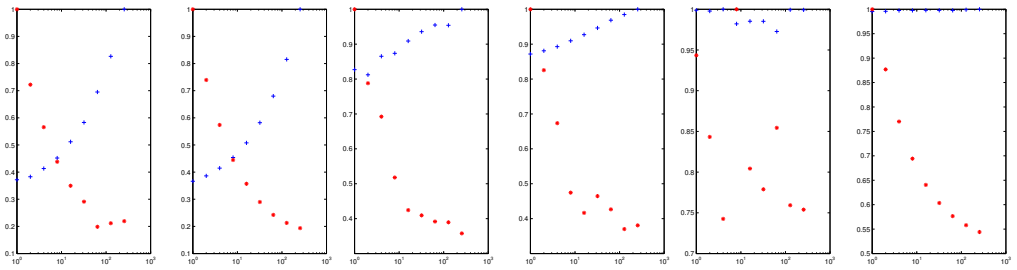

Figure 13: Measurements for different payload bitrates. The sample rate of the cover audio is always 44100 samples per second.

In this case, the maximal absolute value of the samples behaves in a way very similar to what we observed when we were altering the chunk size.

On the other hand, as the throughput is increased, the changes made to the cover audio become more substantial and thus and thus the noise increases resulting in higher mean absolute value of the samples.

Again, we see a big difference between calmer and louder music. This is again due to the fact that when demixing, we get a noisier signal from the louder music.

7.4.3 Altering the energy of the payload

We finally, take a look at the energy of the signal. Recall that at the mixing step of the steganographic scheme, we compute $\mathbf{s} = \mathbf{x} + \mathbf{y}$, however, we can instead compute $\mathbf{s} = \mathbf{x} + c\mathbf{y}$. This new parameter c, makes it possible to choose the relative energy between the signals $\mathbf{x}$ and $\mathbf{y}$.

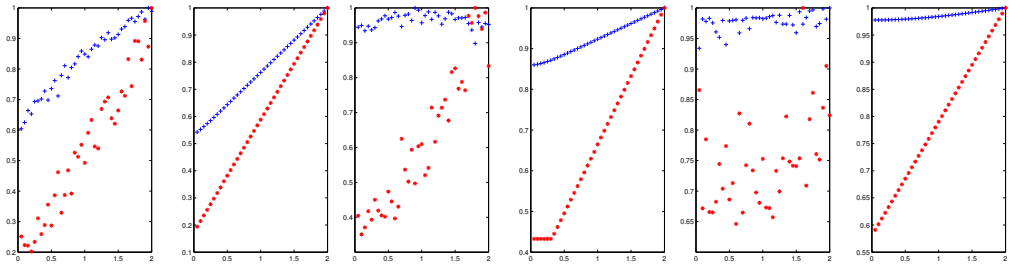

Figure 14: Data set for different values of c. The range is from absolutely no payload signal (i.e. $\mathbf{s} = \mathbf{x}$), to having twice as much energy in the payload signal than in the cover audio signal.

The results are straightforward but very telling. The more energy we put into the payload signal, the more noise we get and thus the mean absolute value of the samples gets higher. Thus, as most of the energy still goes towards the actual payload data, the maximal value of the samples grows much faster than the mean absolute value of the samples as a

function of the energy of the payload signal. Thus, the peaks become more and more prominent and thus easier to detect, which is exactly what is expected.

8 Conclusion

We have achieved the main goals of this project. We have discussed a steganographic scheme that can survive D/A conversion and made a prototype implementation. The different parameters have then been individually tested in extensive simulations and the results have been interpreted in chapter 7 where we detail the main findings of this project.

The steganographic scheme works as expected and looks promising for practical use.

Without a robust peak detection algorithm, it is difficult to precisely estimate the achievable throughput. However, it seems likely that we can achieve better throughput than the 64 bits per 30 second interval mentioned in [15].

We have also looked at two different transforms, the DFT and the WMDCT, that allow us to perform the mixing step of our steganographic scheme.

8.1 Comparison of DFT and WMDCT

We have gathered all the elements needed to compare the two approaches.

In all the graphs of section 7.4 and section 7.3, the data coming from runs using the DFT is noisier than the data coming from runs using the WMDCT. This is almost certainly due to the fact that when using the DFT matrix, Φ and Ψ are not maximally orthogonal. We have statistical guarantees, but on an individual instantiation we may not end up with two maximally incoherent matrices[35]. Thus there is a kind of leakage from one vector space to the other when doing projections. As we have seen in chapter 7, because of the spread spectrum quality of the projections, the leakage is effectively additional noise.

Thus, as suspected in chapter 4, using the WMDCT instead of the DFT, leads to better results in that respect. Also, since the computational cost of generating Φ and Ψ is not really an issue since it can be done offline beforehand, using the WMDCT seems to be the better choice in our proposed steganographic scheme.

[35] Note that it is even possible that the random diagonal of D is all 1 and thus D is the identity matrix and we end up with $\Psi = \Phi$. Fortunately, the probability that this happens is vanishingly small for any reasonable size of Φ.

9 Future work

Several topics related to the concepts discussed in this report are outside the scope of this project. There is a fair amount of further work that could. Here are a few examples of things that could make the ideas discussed in this report usable in practice.

9.1 Good algorithms for peak detection

In order to have a better working prototype and especially as a tool for better understanding of how changing different parameters affects the throughput of the payload, it is important to develop and implement a good peak detection algorithm.

One of the main challenges, as pointed out in section 6.5 is the fact that the Insertion-Deletion channel is a relatively misunderstood and challenging one.

9.2 Source coding

Insertion-deletion channels are, as we have pointed out, very challenging. It would be interesting to study how channel parameters[36] affect the properties of the channel.

Moreover, it would be worthwhile experimenting with various codes for this particular channel and see how payload throughput is altered.

9.3 Human auditory system modelling

In the case of audio steganography, modifying the cover file to piggyback data alters the sound that is output by the speakers. There is a trade-off between the amount of data we want to hide steganographically (which adds noise to the cover audio) and the fact that we do not want to distort the sound so much that it negatively affects the listener's experience. Ideally, the changes in the audio should be inaudible by a human.

In order to study this trade-off more formally, one could use computer models of the human auditory system and use simulations in order to determine precisely how and how much data can be piggybacked on the audio without degrading the sound too much[37].

[36] Noise and resampling.

[37] Note that the acceptable amount of distortion depends on the exact application. HAS modelling is useful in any case.

Technical remarks

The proof of concept implementation of the steganographic scheme discussed in this report has been implemented in MATLAB[38]. It uses The Large Time-Frequency Analysis Toolbox (LTFAT)[39] for the signal processing aspects of the code such as performing DFTs and WMDCTs or plotting spectrograms. It also uses UNLocBoX[40] for solving the convex optmisation problem when demixing signals. Both of these dependencies are Open Source.

The code is available upon request and so is the complete data set generated during the simulation runs.

[38] `http://www.mathworks.com/products/matlab/`
[39] `http://ltfat.sourceforge.net/`
[40] `http://unlocbox.sourceforge.net/`

Acknowledgements

Working on this exciting subject, which is in a field quite different from what I am used to has been interesting and challenging. The work can be extended quite a bit and may even lead to interesting applications.

I would like to thank Prof. Pierre Vandergheynst for his great supervision. His insights have always been very valuable and I very much enjoyed working in his lab.

I would also like to thank Johan Paratte and Nathanaël Perraudin for their input, for their help and for being great friends.

Finally, I would also like to acknowledge Johan Paratte for providing the LaTeXtemplate for this report.

References

[1] Cvejic, N., and Seppanen, T. Increasing the capacity of lsb-based audio steganography. In *Multimedia Signal Processing, 2002 IEEE Workshop on* (2002), pp. 336–338.

[2] Davey, M., and MacKay, D. J. C. Reliable communication over channels with insertions, deletions, and substitutions. *Information Theory, IEEE Transactions on 47*, 2 (2001), 687–698.

[3] Fertonani, D., Duman, T., and Erden, M. Bounds on the capacity of channels with insertions, deletions and substitutions. *Communications, IEEE Transactions on 59*, 1 (2011), 2–6.

[4] Gopalan, K. Audio steganography using bit modification. In *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on* (2003), vol. 1, pp. I–629–32 vol.1.

[5] Kavcic, A., and Motwani, R. Insertion/deletion channels: reduced-state lower bounds on channel capacities. In *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on* (2004), pp. 229–.

[6] Kirovski, D., and Malvar, H. Spread-spectrum watermarking of audio signals. *Signal Processing, IEEE Transactions on 51*, 4 (2003), 1020–1033.

[7] Matsuoka, H. Spread spectrum audio steganography using sub-band phase shifting. In *Intelligent Information Hiding and Multimedia Signal Processing, 2006. IIH-MSP '06. International Conference on* (2006), pp. 3–6.

[8] McCoy, M. B., and Tropp, J. A. The achievable performance of convex demixing. *arXiv preprint arXiv:1309.7478* (2013).

[9] Raguet, H., Fadili, J., and Peyré, G. Generalized forward-backward splitting. *arXiv preprint arXiv:1108.4404* (2011).

[10] Ratzer, E. Marker codes for channels with insertions and deletions. *Annales Des Télécommunications 60*, 1-2 (2005), 29–44.

[11] Schnass, K., and Vandergheynst, P. Dictionary preconditioning for greedy algorithms. *Signal Processing, IEEE Transactions on 56*, 5 (2008), 1994–2002.

[12] Tropp, J. Just relax: convex programming methods for identifying sparse signals in noise. *Information Theory, IEEE Transactions on 52*, 3 (2006), 1030–1051.

[13] Wang, J. *Geometric structure of high-dimensional data and dimensionality reduction.* Springer, 2012.

[14] Wesfreid, E., and Wickerhauser, M. Adapted local trigonometric transforms and speech processing. *Signal Processing, IEEE Transactions on 41*, 12 (1993), 3596–3600.

[15] Westfeld, A., Wurzer, J., Fabian, C., and Piller, E. Pit stop for an audio steganography algorithm. In *Communications and Multimedia Security*, B. Decker, J. Dittmann, C. Kraetzer, and C. Vielhauer, Eds., vol. 8099 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 123–134.

[16] Zhang, L., and Wang, J. Adaptive information hiding based on local sparsity. In *Information Management, Innovation Management and Industrial Engineering (ICIII), 2013 6th International Conference on* (2013), vol. 2, IEEE, pp. 273–2777.