Poznan University of Technology
Faculty of Computing Science
Chair of Control and Systems Engineering

Bachelor's thesis

APPLICATIONS FOR INSETTING AND RECOVERY OF HIDDEN INFORMATION AUDIO SIGNALS

Błażej Kotowski, 106603
Tomasz Pewiński, 106638

Supervisor
prof. dr hab. inż. Adam Dąbrowski

Poznań, 2015

Temat
pracy dyplomowej inżynierskiej
nr

Politechnika Poznańska
Wydział Informatyki
Katedra Sterowania i Inżynierii Systemów

Studia stacjonarne I stopnia
Kierunek: ~~**Automatyka i Robotyka**~~ Informatyka
Specjalność: -

Zobowiązuję/zobowiązujemy się samodzielnie wykonać pracę w zakresie wyspecyfikowanym niżej. Wszystkie elementy (m.in. rysunki, tabele, cytaty, programy komputerowe, urządzenia itp.), które zostaną wykorzystane w pracy, a nie będą mojego/naszego autorstwa będą w odpowiedni sposób zaznaczone i będzie podane źródło ich pochodzenia.

	Imię i nazwisko	Nr albumu	Data i podpis
Student:	**Błażej Kotowski**	106603	
Student:	**Tomasz Pewiński**	106638	

Tytuł pracy:	**Aplikacje do wstawiania i odczytu informacji ukrytej w sygnale dźwiękowym**
Wersja angielska tytułu:	*Applications for insetting and recovery of hidden information audio signals*
Dane wyjściowe:	**Dane literaturowe dotyczące standardów kodowania sygnałów audio, zjawisk maskowania i innych zjawisk psychoakustycznych oraz technik wstawiania sygnałów ukrytych (tzw. znaków wodnych)**
Zakres pracy:	**1. Opracowanie algorytmu wstawiania informacji ukrytej do sygnału dźwiękowego** **2. Budowa stanowiska laboratoryjnego do przeprowadzenia eksperymentów przy wykorzystaniu platformy mobilnej** **3. Opracowanie i testy metody odczytywania informacji ukrytej**
Miejsce prowadzenia prac:	**Katedra Sterowania i Inżynierii Systemów, Pracownia Układów Elektronicznych i Przetwarzania Sygnałów**
Termin oddania pracy:	**31 stycznia 2015 r.**
Promotor:	**Prof. dr hab. inż. Adam Dąbrowski**

Dyrektor Instytutu

PRODZIEKAN
Wydziału Informatyki
Politechniki Poznańskiej
prof. dr hab. inż. Zbyszko Królikowski

Dziekan

Poznań,
Miejscowość, data

Contents

Chapter 1

Introduction

1.1 Motivation

Majority of communications in modern age are carried through electromagnetic waves. However, this is not the only possible medium of communication. The ubiquity of cheap audio equipment, like loudspeakers and microphones, creates a possibility for data transmission via sound.

Unlike electromagnetic waves, a range of sound pressure waves can be perceived by humans. This creates an issue when one does not want listeners to be aware that data is transmitted. A way of solving this problem is insetting the data signal into a host sound in a specially crafted manner, exploiting the properties of human auditory system. The host sound should mask the data signal, making it imperceptible to humans, while still detectable by machines.

As sound production, transmission and recording is commonplace, such system would have many applications, some of which are discussed in Chapter 6.

1.2 Thesis Objective

This work aims at achieving the following goals:

- implementation of software system for insetting and recovery of data in host audio signal
- evaluation of implemented system with regard to criteria of transmission robustness and distortion imperceptibility.

The system should take existing audio file, like a piece of music, and data to be transmitted, and combine them to generate a sound to be played by loudspeakers. This sound should be as close to original audio as possible, i.e. the human listener should not be able to perceive the distortion introduced by hiding the payload data. On the receiving side, another program should be able to recover the payload from recorded sound.

1.3 Thesis Outline

- Chapter 2 gives a brief overview of the theoretical concepts required for the solution, focusing on digital signal processing concepts as well as communications and coding techniques
- Chapter 3 describes the proposed solution of the problem
- In Chapter 4, implementation details of the system are presented, including relevant excerpts of python source code

- Chapter 5 presents results of experimental evaluations, outlining system's performance in various conditions
- Finally, Chapter 6 sums up the work, including possible applications of the system, as well as identifying possible areas for improvement.

For purposes of this thesis, Błażej Kotowski is responsible for implementation and experimental evaluation. Tomasz Pewiński is responsible for designing a solution and implementation.

Chapter 2

Theory

This chapter presents definitions and concepts used in latter parts of the thesis.

2.1 Spectral analysis

Spectral analysis is concerned with decomposing signals into sums of simple trigonometric functions. In signal processing applications, it allows for easy detection of individual components of a waveform.

There exist many methods for spectral analysis. In this section, methods most often used in audio applications are presented.

2.1.1 Discrete Fourier Transformation

Discrete Fourier Transformation (DFT) is a method of transforming discrete, periodic, complex signals from time-domain representation into spectral representation [OS89].

An input x of N complex values is transformed into output X of N complex values according to formula 2.1. [Smi09]

$$X_k = \sum_{n=0}^{N-1} x_n \cdot \mathrm{e}^{-\jmath 2\pi kn/N}, k = 0, 1, 2, ..., N-1 \tag{2.1}$$

The phase and magnitude of each frequency component k are, respectively, absolute value and argument of complex number X_k:

$$\mathrm{mag}_k = |X_k| \tag{2.2}$$

$$\varphi_k = \arg(X_k) \tag{2.3}$$

Inverse operation to DFT is also possible, transforming spectrum X back to the signal x.

In practical applications, x consists of real numbers. In such case, X obeys a symmetry 2.4.

$$X_{N-k} = X_k^* \tag{2.4}$$

(where $*$ is complex conjugation)

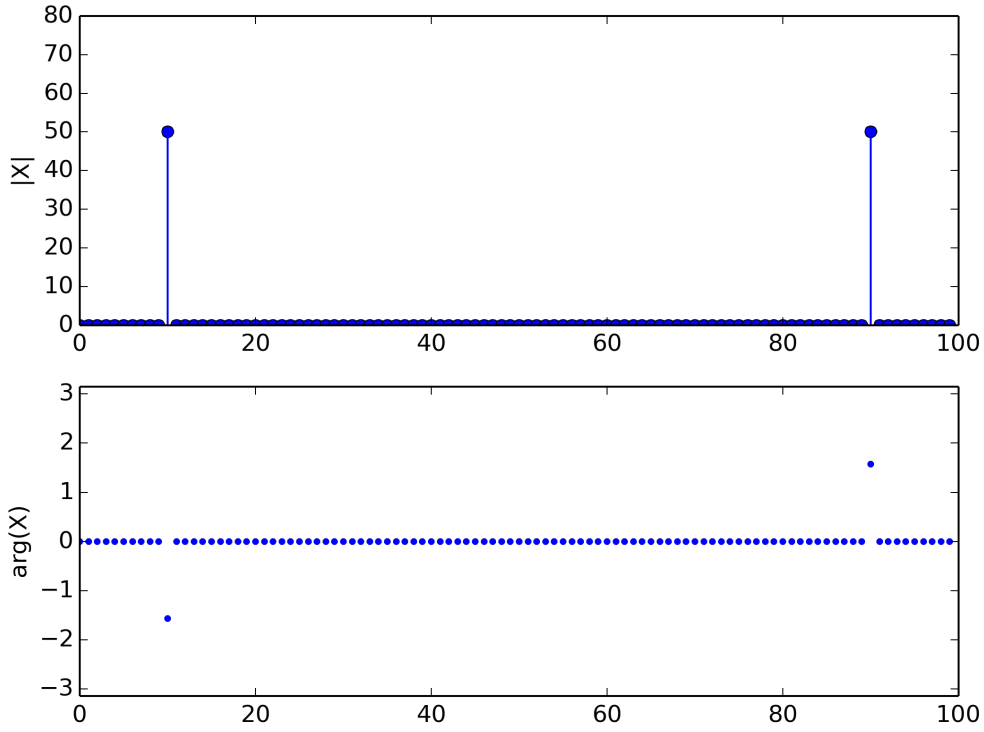


Figure 2.1: Magnitude and phase computed from DFT of 100 samples of 10 Hz sine signal, sampled at 100 samples/second.
Symmetry from (2.4) can be observed.

2.1.2 Fast Fourier Transformation

Fast Fourier Transformation (FFT) is an implementation of discrete Fourier transformation.

Computing DFT in naive way has $O(n^2)$ complexity, which is often too much for practical applications. FFT algorithms improve that significantly – the most popular one, Cooley-Turkey FFT, achieves $O(n \log n)$ [Smi09].

FFT is widely used in digital signal processing for its speed and ease of use.

2.1.3 Goertzel algorithm

Goertzel algorithm can be used to obtain magnitude and phase of single frequency component of a signal [SR12].

It is more computationally effective than FFT when only few components need to be extracted. Its main application is in Dual-Tone Multi-Frequency (DTMF) telephone tone dialing, where pairs of different frequency signals are transmitted to encode symbols and letters [Moc87].

Goertzel algorithm effectively computes a k-th coefficient of the DFT (see equation 2.1).

Both DFT and Goertzel algorithm can only compute exact values for frequency components being δf Hz apart (2.5).

$$\delta f = \frac{\mathrm{fs}}{N} \tag{2.5}$$

(where fs is sampling frequency and N is length of the signal)

```python
import math

def goertzel(x, k):
  """Computes spectral component at DFT bin k of signal x"""

  N = len(x) # length of signal
  A = 2*math.pi*k/N
  B = 2*math.cos(A)
  C = math.e ** (-1.0j*A) # e^(-iA)

  s0 = 0
  s1 = 0
  s2 = 0

  for i in range(0, N): # N ranges from 0 ... N - 1
    s0 = x[i] + B * s1 - s2
    s2 = s1
    s1 = s0

  s0 = B * s1 - s2
  return s0 - s1 * C
```

Figure 2.2: Goertzel algorithm implemented in python

In case of frequency components not corresponding exactly to integer multiples of δf, values in adjacent DFT bins are affected – this is called spectral leakage. 2.3

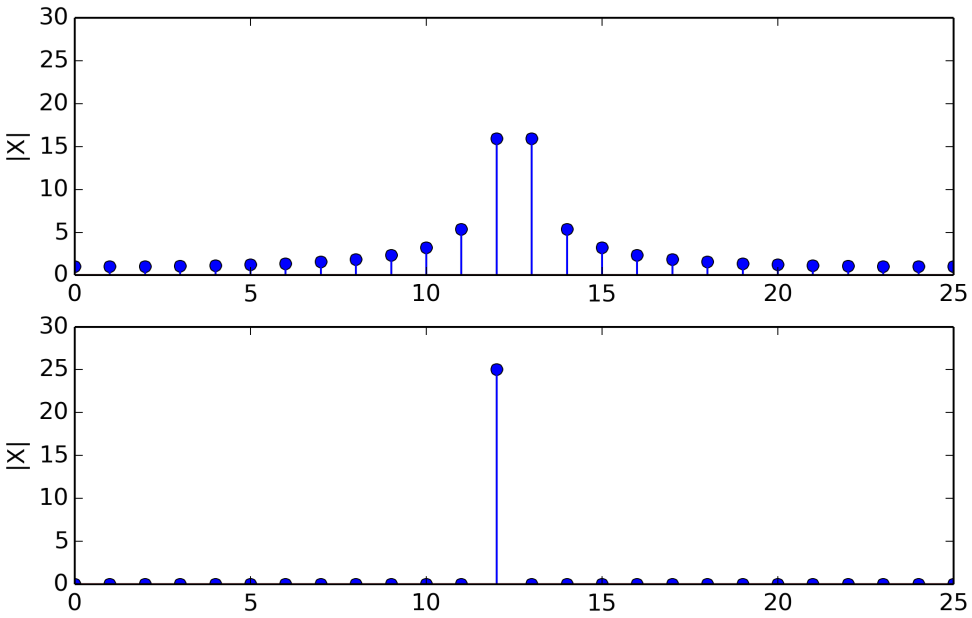


Figure 2.3: Spectral leakage observed in top spectrum for $f_1 = 25\,\text{Hz}$. Contrasted with bottom spectrum of $f_2 = 24\,\text{Hz}$ (fs $= 100\,\text{Hz}, N = 50, \delta f = 2\,\text{Hz}$).

For DTMF applications of Goertzel algorithm, a value of $N = 205$ is commonly used – that value minimizes deviations of signaling frequencies from integer multiples of δf, thus minimizing the effect of spectral leakage [Moc87].

However, Goertzel algorithm can also be generalized for non-integer multiples of δf [SR12].

2.2 Filtering

Filtering is a process of removing unwanted components from a signal. Filtering is possible in analog (continuous) as well as digital (discrete) domains [Smi97]. There are two types of digital filters: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR).

A FIR filter is defined by its impulse response b_i (list of coefficients of size N). It is applied to a signal x by convolving it with b:

$$y_n = \sum_{i=0}^{N} b_i \cdot x_{n-i} \tag{2.6}$$

FIR filters, as opposed to IIR filters, do not require feedback and are inherently stable. Their main disadvantage is high computational requirements.

2.2.1 Bandpass filter

A bandpass filter is a type of filter that passes frequencies within a certain range and attenuates frequencies outside that range [Smi07].

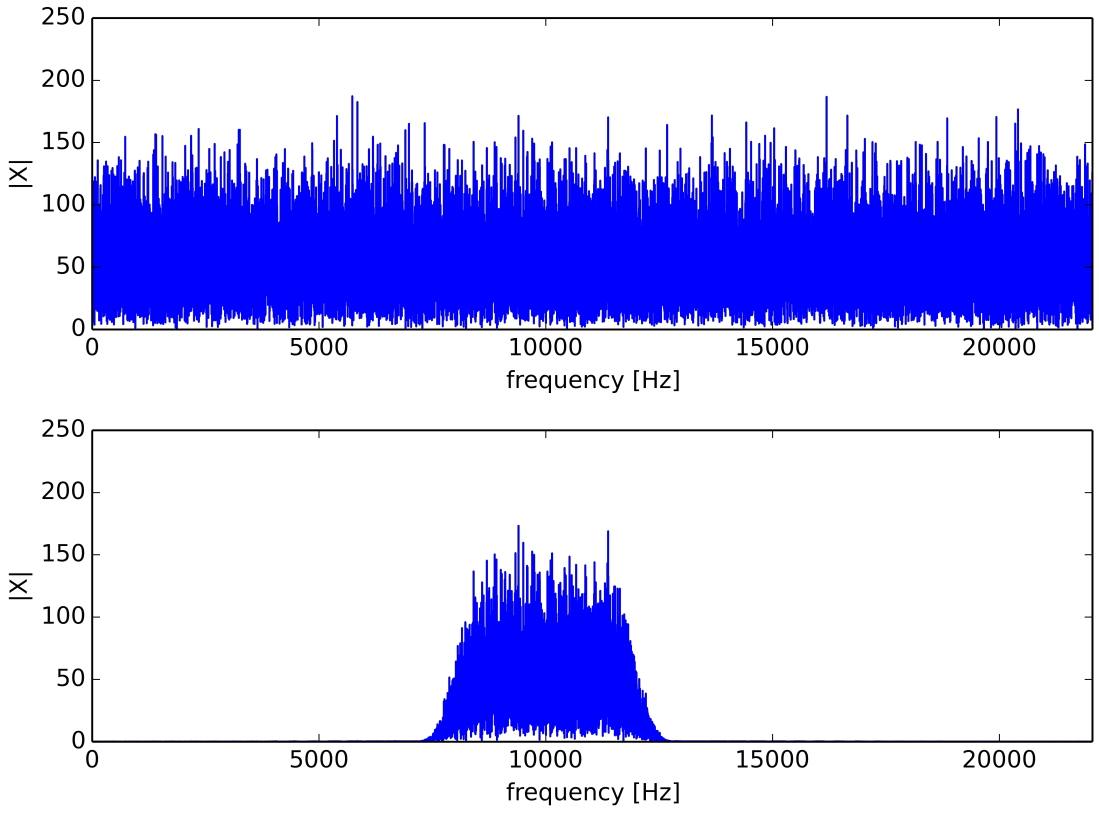


Figure 2.4: Spectrum of white noise and noise filtered with bandpass filter with range 8–12 kHz

Bandpass filters have many applications, for example in wireless receivers – where they are used to remove part of the signal outside transmission's frequency band.

2.2.2 Bandstop filter

Bandstop filter can be thought of as the inverse of bandpass filter – it attenuates frequencies within a certain range and passes frequencies outside that range [Smi07].

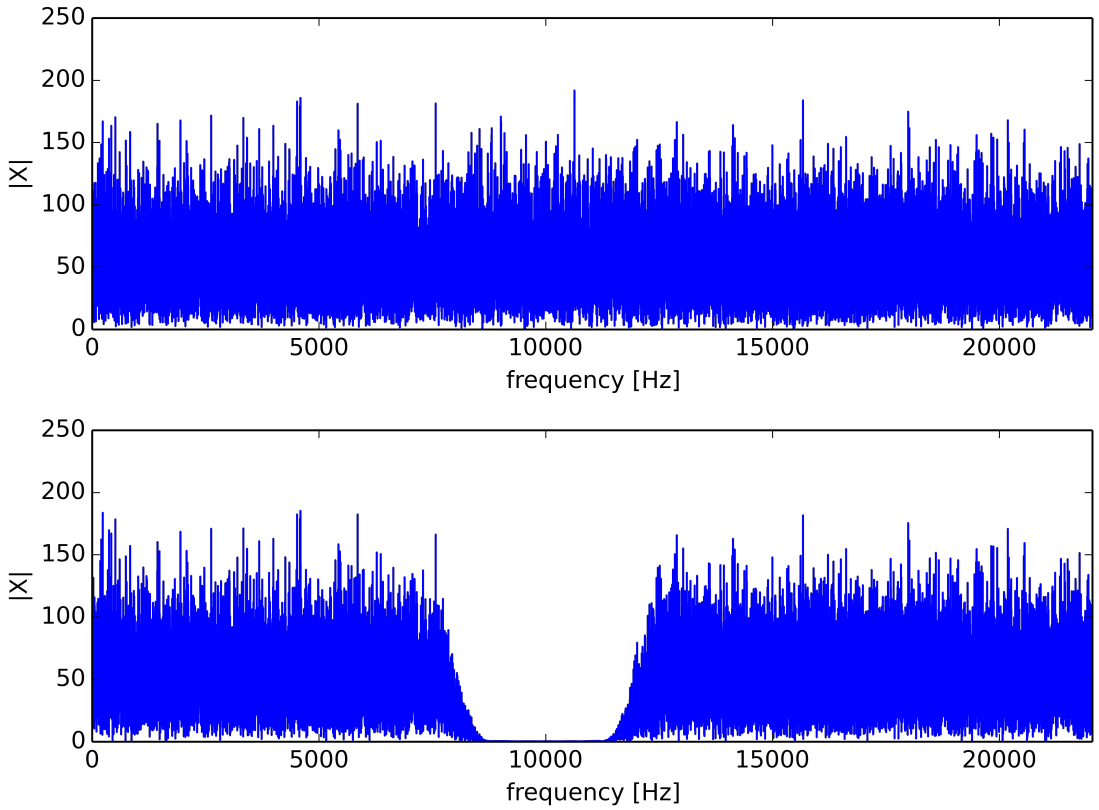


Figure 2.5: Spectrum of white noise and noise filtered with bandstop filter with range 8–12 kHz

One example of bandstop filter's application is removing the hum of 60 Hz alternating current power line in audio systems.

2.3 Modulation

Modulation means changing properties of one signal (carrier signal) with another signal (modulating signal).

Reverse process, called demodulation, can be applied to extract modulating signal from the modulated waveform. If modulating signal carries some kind of meaningful information, the modulation-demodulation process can be used for data transmission.

There are many types of modulation, both analog and digital. Examples of analog modulation include:

- Amplitude Modulation (AM) – modulating signal changes carrier's amplitude (Figure 2.6)
- Frequency Modulation (FM) – modulating signal changes instantaneous frequency of carrier

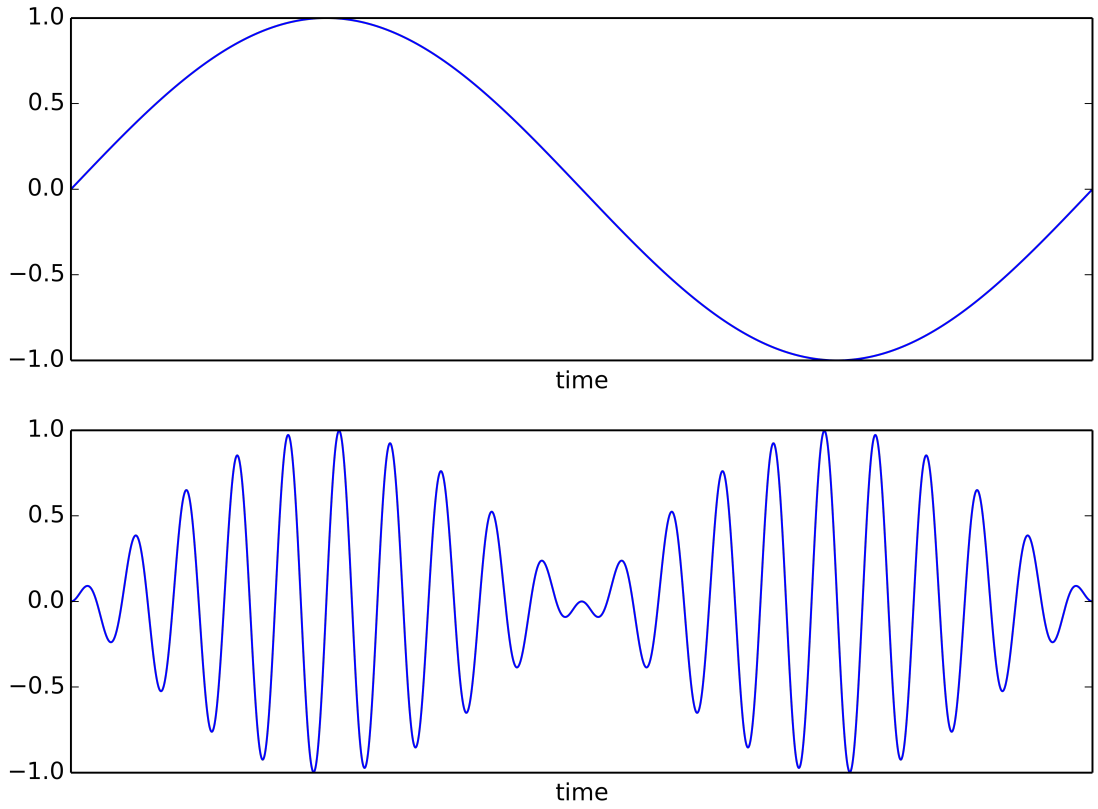


Figure 2.6: Amplitude modulation – 1 Hz sine (top) modulating the amplitude of 20 Hz sine (bottom)

2.3.1 Phase Shift Keying

Phase Shift Keying (PSK) is a type of digital modulation. In case of PSK, the modulating signal is a stream of bits, and modulation process varies the phase of carrier signal.

The demodulator must have access to reference signal to compare its phase value to incoming signal's. However, schemes without reference signal are also possible – in this case change of phase in modulated signal itself is used to convey information – this is called differential phase shift keying.

PSK is widely used in digital communications, for example in Wi-Fi and RFID.

2.3.2 Differential Binary Phase Shift Keying

DBPSK is a type of differential phase shift keying where phase shifts of 0 and π are used (hence "binary"). In most schemes, phase change of π encodes binary 1 and no phase change encodes binary 0 (see Figure 2.7). DBPSK does not require reference signal for demodulator.

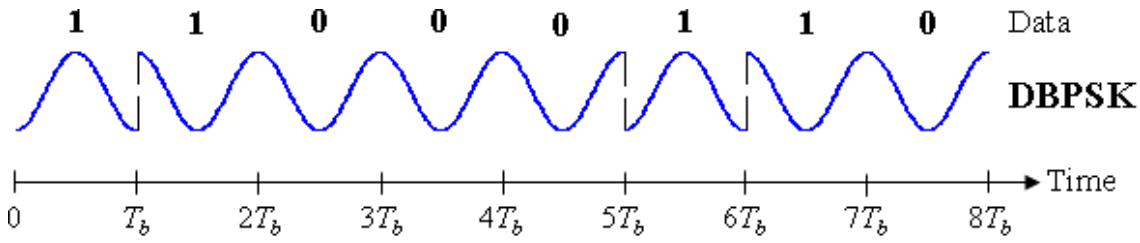


Figure 2.7: Differential binary phase shift keying. Source: `http://en.wikipedia.org`

2.4 Channel coding

Channel coding (also called forward error correction) is a method of detecting and / or correcting errors in transmission over noisy channel. It is accomplished by adding redundant data to original message, which allows the receiver to detect errors anywhere in the message.

Advantage of using forward error correction is that it allows correcting errors without need for re-transmission. However, this comes at a cost of decreased effective data bandwidth, since redundant data (error correcting code) must also be transmitted.

Two types of error correcting codes can be considered:

- block codes – requires message to be of fixed length
- convolutional codes – work on streams of data of any length

Forward error correction is widely used in mass storage applications for dealing with corrupted data, as well as in modems.

2.4.1 Reed-Solomon code

Reed-Solomon code is an example of block forward error correction code. This code of size t can correct up to $\lfloor t/2 \rfloor$ errors in received message [Cla02].

Reed-Solomon codes are used in mass storage, like CD and DVD standards, and QR codes [WB94].

2.5 Psychoacoustics

Psychoacoustics can be described as branch of science studying perception of sound. Its applications range from sound compression algorithms to music therapy.

2.5.1 Human auditory system

Most frequently quoted range of human hearing is 20 Hz – 20 kHz [Moo97]. However, the upper bound tends to decrease with age, and is 16 kHz for typical adult.

Human Auditory System (HAS) exhibits many properties studied by psychoacoustics. Among most prominent are:

1. Amplitude of the sound does not correspond linearly to perception of loudness for different frequencies. Equal loudness contour from figure 2.10 is obtained experimentally and shows increased sensitivity around 3–4 kHz (center frequency of human voice) [JN84][NL99].

2. Masking effects – an otherwise audible sound can be masked by another sound and become inaudible. Two aspects of masking effect are spectral masking and temporal masking [DM02].

   Spectral masking (also called simultaneous masking) can be observed when masker and maskee sounds are presented simultaneously. For example, presence of loud tone at 1 kHz can make quieter tone at 1.2 kHz inaudible.

   Temporal masking is observed when presence of masker makes sounds preceding the onset and following the offset inaudible. The power of temporal masking tends to decrease with time from onset/offset of masker.

3. Missing fundamental – humans tend to perceive pitch of f when hearing harmonic series $2f, 3f, 4f...$

2.5.2 Bark scale

Bark is a psychoacoustic scale named after Heinrich Barkhausen, who proposed first subjective measurement of loudness.

The scale divides human hearing range into 24 critical bands [Zwi61] (see Figure 2.8). It makes working with psychoacoustic models easier by approximating the curve of absolute threshold of hearing with constant (see Figure 2.9).

Equation 2.7 is used to convert frequency in Hertz to Bark [Zwi61].

$$\text{bark} = 13 \arctan(0.00076f) + 3.5 \arctan((\frac{f}{7500})^2) \tag{2.7}$$

2.5.3 Psychoacoustic models

Psychoacoustic models can be used to detect inaudible (masked) components of a sound. Their main application is data compression [BB97].

An example of psychoacoustic model introduced in [Vir] is presented. This model exploits spectral masking effects.

1. for a given masker sound, power spectrum is obtained using FFT [2.1.2]
2. tone or noise maskers are identified within masker sound
3. based on identified maskers, masking threshold is calculated for desired frequency.

A frequency component k of frequency f with power P_k is considered a tone if:

1. it is a local maximum, i.e. $P_{k-1} < P_k > P_{k+1}$
2. it is at least 7dB greater than frequency components in its neighborhood, where neighborhood is:
   - $P_{k-2}, \ldots, P_{k+2}$ if $f < 5.5\,\text{kHz}$
   - $P_{k-3}, \ldots, P_{k+3}$ if $5.5\,\text{kHz} \leq f < 11\,\text{kHz}$
   - $P_{k-6}, \ldots, P_{k+6}$ if $f \geq 11\,\text{kHz}$

To identify noise maskers, for each critical band (2.8), components which are not in neighborhood of a tone are added and placed at geometric mean location within the critical band.

Masking level of frequency component i on component j is defined by equation 2.8 for tones and 2.9 for noises.

$$\text{mask}(i,j) = P_j - 0.275z(j) + S(i,j) - 6.025\,[\text{dB}] \tag{2.8}$$

$$\text{mask}(i,j) = P_j - 0.175z(j) + S(i,j) - 2.025\,[\text{dB}] \tag{2.9}$$

where z is a function converting component frequency to bark scale (see equation 2.7) and S is spreading function defined in equation 2.10.

$$S(i,j) = \begin{cases} 17\delta z - 0.4P_j + 11 & \text{if } -3 \leq \delta z < -1 \\ (0.4P_j + 6)\delta z & \text{if } -1 \leq \delta z < 0 \\ -17\delta z & \text{if } 0 \leq \delta z < 1 \\ (0.15P_j - 17)\delta z - 0.15P_j & \text{if } 1 \leq \delta z < 8 \end{cases} \tag{2.10}$$

where $\delta z = z(i) - z(j)$.

Finally, masking threshold for component k can be calculated by adding absolute threshold of hearing $\mathrm{ath}(k)$ and masking levels of individual tone/noise maskers.

$$\mathrm{threshold}(k) = 10\log_{10}(10^{\mathrm{ath}(k)/10} + \sum_i 10^{\mathrm{mask}(i,k)/10})\,[\mathrm{dB}] \tag{2.11}$$

Number	Center frequency (Hz)	Cut-off frequency (Hz)	Bandwidth (Hz)
1	50	100	80
2	150	200	100
3	250	300	100
4	350	400	100
5	450	510	110
6	570	630	120
7	700	770	140
8	840	920	150
9	1000	1080	160
10	1170	1270	190
11	1370	1480	210
12	1600	1720	240
13	1850	2000	280
14	2150	2320	320
15	2500	2700	380
16	2900	3150	450
17	3400	3700	550
18	4000	4400	700
19	4800	5300	900
20	5800	6400	1100
21	7000	7700	1300
22	8500	9500	1800
23	10500	12000	2500
24	13500	15500	3500

Figure 2.8: Critical bands of Bark scale. Source: [Zwi61]

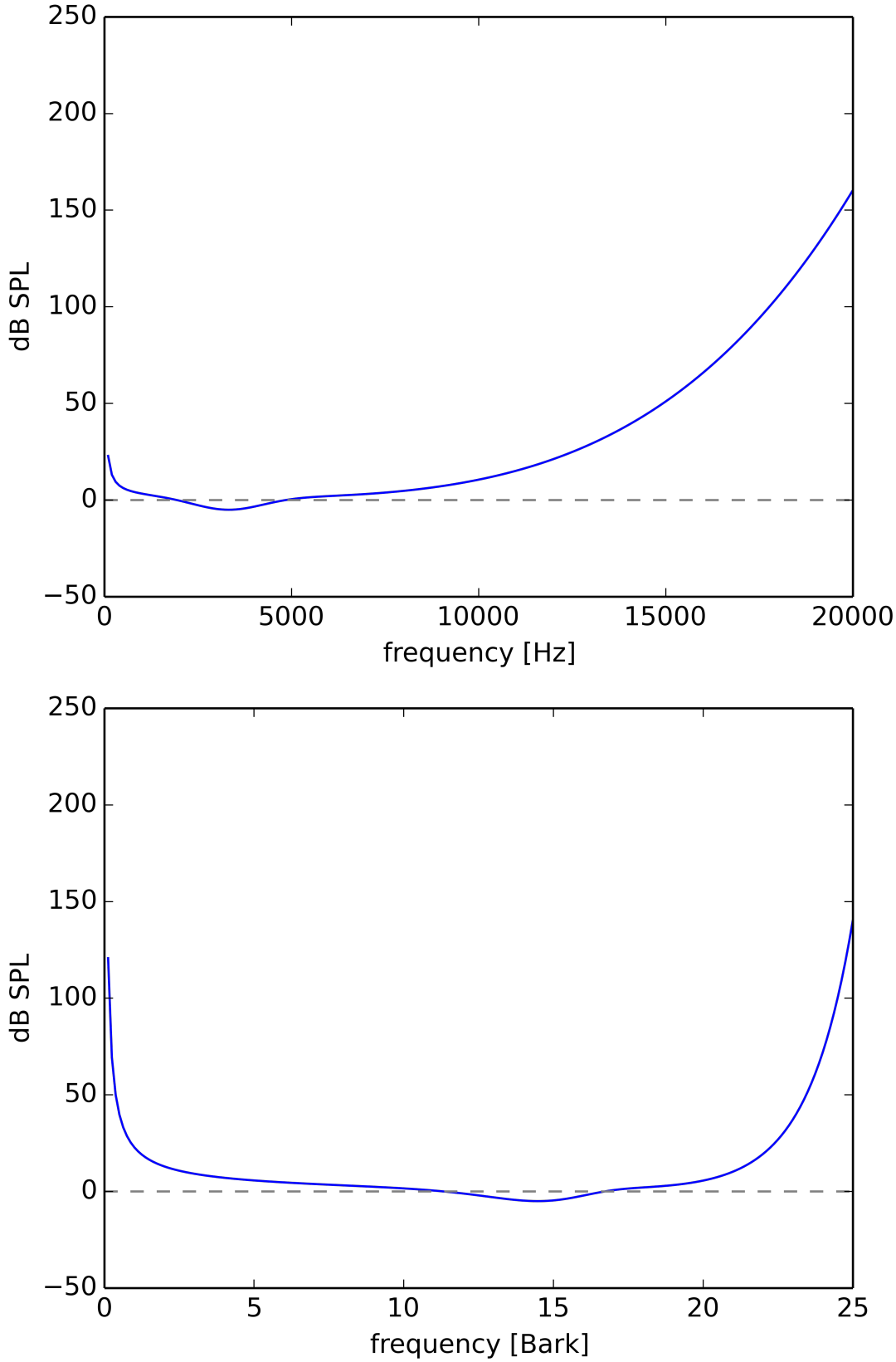


Figure 2.9: Absolute Threshold of Hearing on hertz scale and Bark scale

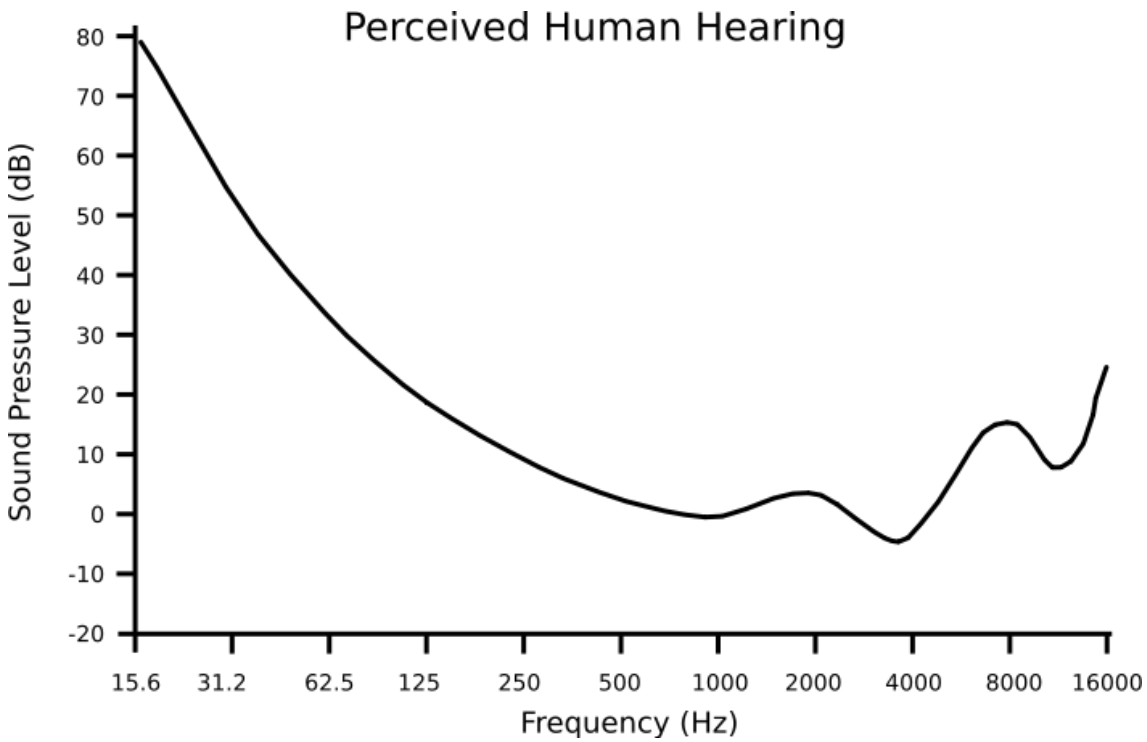


Figure 2.10: Equal loudness contour
Source: `http://en.wikipedia.org`

Chapter 3

Solution

3.1 Overview

The proposed solution is to transmit payload data using nine sinusoidal signals hidden in host signal (eight carrier signals and one synchronization signal). Two algorithms are required: one for sound synthesis and one for recording analysis.

The synthesis algorithm:

1. splits payload data into frames
2. uses frame bytes to modulate carrier signals, utilizing DBPSK [2.3.2]
3. adjusts amplitude of data signal using a psychoacoustic model
4. filters host signal, removing frequencies around data signal
5. adds data signal to filtered host signal

The analysis algorithm:

1. demodulates the recorded signal into a stream of bytes
2. uses a frame detection algorithm to extract frame data
3. for each frame, applies error correction and outputs the result bytes

3.2 Synthesis algorithm

The signal synthesis algorithm has some configurable parameters:

- d – duration of signal which encodes single byte (seconds)
- fs – sampling rate
- sync – synchronization signal frequency (Hertz)
- carriers – array of carrier signals frequencies (in Hertz)

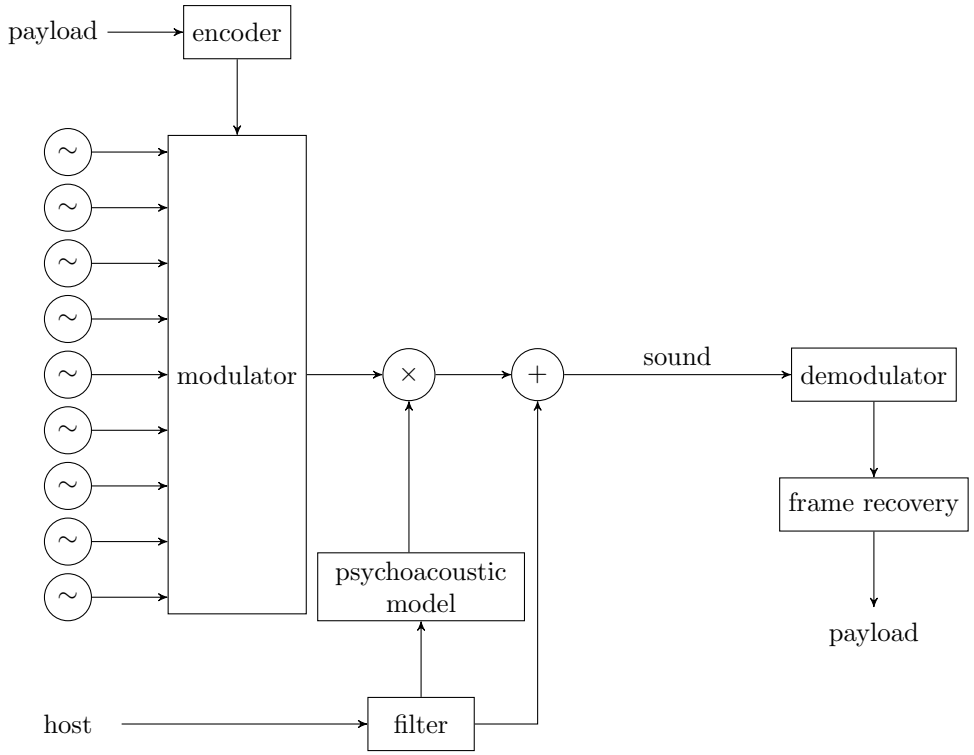


Figure 3.1: Block diagram of proposed solution

3.2.1 Data frame

To ensure robust payload decoding in spite of transmission errors, data is encoded in frames consisting of 24 bytes:

- preamble – 2 bytes
- payload – 16 bytes
- error correction data – 6 bytes

A preamble is used in frame recovery algorithm [3.3.2] to detect frame beginning. It consists of 2 ASCII synchronous idle characters (`0x16`). `0x` notation denotes a byte in hexadecimal format.

Error correction data is calculated from payload using Reed-Solomon [2.4.1] code.

3.2.2 Carrier modulation

Single carrier signal is a sine wave, modulated by a stream of bits using differential binary phase shift keying [2.3.2].

DBPSK has the advantages over other modulation methods for audio domain: [App13]

- more robust and resilient to noise than amplitude modulation
- uses less bandwidth than frequency modulation, meaning narrower band will be removed from host signal
- does not require reference signal in demodulator

However, sudden phase transitions required by DBPSK cause audible parasitic effects ("clicking"). To mitigate that, the parts of the signal at transition time are multiplied by cosine window.

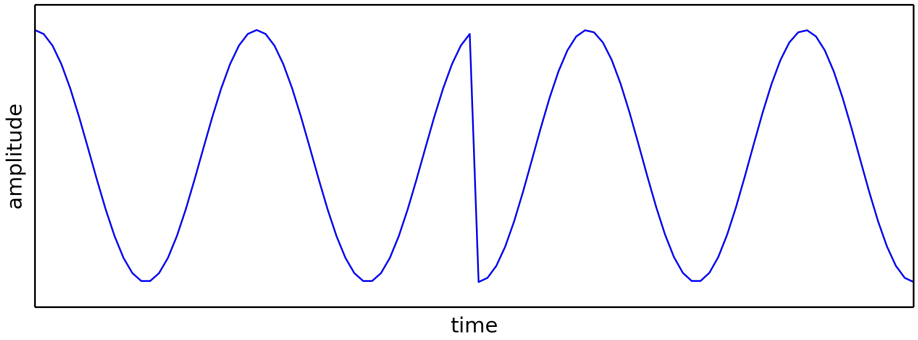


Figure 3.2: Sudden phase transition of π

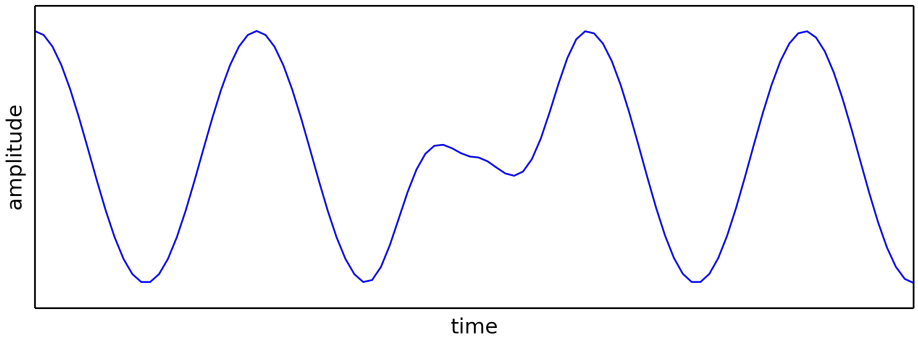


Figure 3.3: Smooth phase transition of π

3.2.3 Byte transmission

Total of nine sine signals are transmitted simultaneously. Eight of them each encode a single bit of a byte. The remaining signal is a synchronization signal, modulated by stream of binary ones, is used for detecting byte start / end.

3.2.4 Insetting in host signal

Insetting carrier signal in host signal consists of 4 steps:

1. host signal is filtered with bandstop filter, removing frequencies around carrier signal
2. for each window, masking threshold for carrier signal is calculated using psychoacoustic model described in [2.5.3]
3. amplitude of carrier signal is multiplied by value under masking threshold
4. carrier signal is added to filtered host signal

3.3 Analysis algorithm

Analysis algorithm shares configuration parameters with synthesis algorithm, and can further be parameterized with:

- ws – window size for analysis (samples)

Each ws consecutive recorded samples are analyzed by analysis algorithm.

3.3.1 Carrier demodulation

A phase reading is obtained for each of the carrier frequencies and synchronization frequency.

For each frequency:

1. recorded frame is filtered with a bandpass filter centered at that frequency and multiplied by Hanning window
2. phase shift relative to beginning of frame is calculated using Goertzel algorithm [2.1.3]
3. absolute phase shift ($|\varphi|$) is obtained by calculating phase offset ($\delta\varphi$) of current frame and subtracting it from relative phase (φ).

$$\delta\varphi = 2\pi(\frac{\text{framestart} \cdot \text{frequency}}{\text{fs}} \pmod{1.0}) \tag{3.1}$$

Where framestart is the index of the first sample of a frame in a stream of all recorded samples, and fs is sampling rate.

$$|\varphi| = \varphi - \delta\varphi \tag{3.2}$$

When the synchronization signal has stable phase for the predefined number of readings (rpb – readings per byte), it signals a byte transmission. Phase is considered stable when standard deviation of rpb previous readings is below $\pi/5$.

$$\text{rpb} = \lfloor\frac{\text{fs} \cdot \text{d}}{\text{ws}}\rfloor \tag{3.3}$$

When byte transmission occurs, previous rpb bit carrier phase readings are averaged and saved. Then the averages are compared to previous averages and if difference of π is detected, a binary 1 is decoded. Otherwise, binary 0 is decoded.

Eight bits decoded from carrier signals are then combined to form a single byte, which gets forwarded to frame recovery algorithm.

3.3.2 Frame recovery

A simple frame recovery algorithm scans the byte stream for preamble bytes. However, since transmission error might have occurred and preamble might be scrambled, the algorithm has to evaluate each consecutive pair of bytes as a candidate for being a preamble.

It assigns a score for each position in bit stream, calculated as follows:

$$\text{score} = 1 - \frac{\text{pop}(\text{bit0 xor } \texttt{0x16}) + \text{pop}(\text{bit1 xor } \texttt{0x16})}{16} \tag{3.4}$$

(where pop is a function returning number of set bits in a byte).

Score of 1 means exact preamble is detected. However, if no preamble position is found, the algorithm waits for next 24 bytes to calculate the score of a position a frame size bytes ahead. If that score is large enough, previous position is selected as frame beginning.

Once beginning of frame is identified and all frame bytes are received, the algorithm extracts payload data, tries to apply Reed-Solomon error correction to correct any transmission errors, and returns decoded part of payload data.

Chapter 4

Implementation

4.1 Programming environment

The solution is implemented in python programming language. Python was chosen because of authors' previous experience with it, ease of programming and prototyping, and availability of digital signal processing and utility libraries.

Following python libraries are used:

- numpy – math and array manipulation library
- scipy – digital signal processing algorithms
- matplotlib – graphing
- reedsolo – implementation of Reed-Solomon code
- pyaudio – playing and recording sound

Main result of work is a python library, with few dependencies, which is portable and can be used in many applications (some are discussed in the Summary chapter)

4.2 Outline of implementation

The library consists of core emitter and receiver modules, as well as several other additional modules.

Emitter and receiver are meant to be used separately each for its own application, and are orthogonal to sound playing/recording concerns. This makes them portable across platforms that support python programming language. Only configuration parameters (synchronization and carrier frequency values) must be shared between receiver and emitter.

4.3 Emitter

Emitter class implements synthesis algorithm described in chapter 3. With emitter object, an application can obtain samples of host sound with payload data hidden in it.

Public methods:

- `Emitter(host, sample_rate, sync_frequency, carrier_frequencies, payload)`

  Constructor

  Arguments:

  - `host` – sampled host signal (`numpy.float32` array)
  - `sample_rate` – sampling rate of host signal (`float`)
  - `sync_frequency` – synchronization frequency in Hertz (`float`)
  - `carrier_frequencies` – list of carrier frequencies in Hertz (list of eight `float`)
  - `payload` – bytes to be transmitted (list of `int`)

  Return value: `emitter`

  - `emitter` – instance of Emitter

- `outstream(buf)`

  When this method is called, emitter will start putting chunks of synthesized samples on the `buf` queue. The caller can then forward these samples to hardware loudspeakers.

  Putting `False` boolean on the queue signals end of synthesized signal.

  Arguments:

  - `buf` – FIFO queue for putting chunks of synthesized sound on (python `Queue`)

  Return value: `buf`

  - `buf` – the same as first argument.

See Figure 4.1 for example usage of emitter in an application.

```
from emitter import Emitter
from Queue import Queue

queue = Queue()

# SYNTHESIS thread
fs = 44100 # sampling rate
host_sound = get_host_sound() # get host sound
payload = get_payload() # get payload data
sync_frequency = 8000.0 # setup frequencies
carrier_frequencies = [ 7000 + 100*i for i in range(0, 8) ]

em = Emitter(host_sound, fs, sync_frequency, carrier_frequencies, payload)
em.outstream(queue)

# PLAYING thread
while True:
  samples = queue.get(True) # blocking call to pop an item from queue
  if samples == False: # end of synthesized signal
    break
  else:
    play_samples(samples) # play sound with hardware
```

Figure 4.1: Example usage of emitter in an application

4.4 Receiver

Receiver class implements analysis algorithm described in chapter 3. With receiver object, an application can decode payload data from recorded sound samples.

Public methods:

- `Receiver(instream, sample_rate, sync_frequency, carrier_frequencies)`

  Constructor

  Arguments:

  - `instream` – queue for incoming recorded samples (python `Queue`)
  - `sample_rate` – sampling rate (`float`)
  - `sync_frequency` – synchronization frequency in Hertz (`float`)
  - `carrier_frequencies` – list of carrier frequencies in Hertz (list of eight `float`)

  Return value: `receiver`

  - `receiver` – instance of Receiver

- `payload_stream(outstream)`

  After calling this method, decoded bytes of data will be put on the queue.

  Putting `False` boolean on the queue means end of decoded data.

  Arguments:

  - `outstream` – FIFO queue for putting decoded bytes on (python `Queue`)

  Return value: `outstream`

  - `outstream` – the same as first argument.

See Figure 4.2 for example usage of receiver in an application.

```
from receiver import Receiver
from Queue import Queue

recorded_queue = Queue()
payload_queue = Queue()

# ANALYSIS thread
fs = 44100
sync_frequency = 8000.0 # setup frequencies
carrier_frequencies = [ 7000 + 100*i for i in range(0, 8) ]

rc = Receiver(recorded_queue, fs, sync_frequency, carrier_frequencies)
rc.payload_stream(payload_queue)

# RECORDING thread
while True:
  samples = device.record()
  recorded_queue.put(samples)

# OUTPUT thread
data = []
while True:
  byte = payload_queue.get(True) # blocking call to pop a byte from the queue
  if byte == False: # end of data
    print "decoded data: {0}".format(payload)
    break
  else:
    data.append(byte)
```

Figure 4.2: Example usage of receiver in an application

4.5 Other modules

4.5.1 Encoder

This sub-module implements *encoding* block from Figure 3.1.

Encoder is responsible for splitting the payload data into frames (adding preamble, calculating Reed-Solomon error correction) and outputting stream of bits ready for transmission. In case payload data cannot be split evenly into frames, it is zero-padded to required length.

Public methods:

- `Encoder(payload)`

  Constructor

  Arguments:

  - `payload` – data to be encoded (list of `int`)

  Return value: `encoder`

  - `encoder` – instance of Encoder

- `encode()`

  Encodes payload into stream of bytes ready for transmission.

Arguments: none

Return value: `out`

- `out` – encoded bytes (list of `int`)

See Figure 4.3 for example usage in code.

4.5.2 Decoder

This sub-module implements *frame recovery* block from Figure 3.1.

Decoder is a stateful object responsible for scanning incoming stream of data, extracting correct frames (see 3.3.2), applying error correction code and outputting decoded payload data.

Public methods:

- `Decoder()`

  Constructor

  Arguments: none

  Return value: `decoder`

  - `decoder` – instance of Decoder

- `add(byte)`

  Registers incoming byte in decoder

  Arguments:

  - `byte` – incoming byte (`int`)

  Return value: none

- `can_decode()`

  Arguments: none

  Return value: `can`

  - `can` – `True` if decoder has enough bytes to decode a frame. Otherwise `False`

- `decode()`

  Decodes payload data from incoming bytes

  Arguments: none

  Return value: `(bytes, frame_count)`

  - `bytes` – list of decoded payload bytes (list of `int`)
  - `frame_count` – how many frames were decoded (`int`)

See Figure 4.4 for example usage in code.

4.5.3 Goertzel

This sub-module implements Goertzel algorithm generalized to non-integer multiple of fundamental frequency, as described in [SR12].

Public methods:

- `goertzel(x, k)`

  Calculates spectral component `k` of signal `x`.

  Frequency of component can be calculated as `k * fs / N` where `fs` is sampling rate, and `N` is length of `x`.

  Arguments:

  - `x` – signal (list of `float`/`complex` numbers)
  - `k` – spectral component index (`float`)

  Return value: `y`

  - `y` – spectral component (`complex` number)

4.5.4 Psychoacoustic analyzer

This sub-module implements *psychoacoustic model* described in 2.5.3.

Public methods:

- `masking_threshold(x, f, fs)`

  For given chunk of sound and frequency, this method calculates masking threshold under which components with given frequency are inaudible.

  Arguments:

  - `x` – signal to calculate masking threshold for (list of `float`)
  - `f` – frequency to calculate masking threshold for (`float`)
  - `fs` – sampling rate (`float`)

  Return value: `threshold`

  - `threshold` – masking threshold for frequency `f` in sound `x` (`float`)

4.5.5 Graphing utilities

The system includes graphing sub-module that can optionally plug into receiver module and plot phase readings, bit synchronization, and frame synchronization moments (see Figure [4.5]). This module has capability to plot values after all payload data was received as well as in real time.

```python
from encoding import Encoder

payload = [104, 101, 108, 108, 111] # "hello" in ASCII, 5 bytes

# encoded single frame
# [22, 22, 104, 101, 108, 108, 111, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 210, 201, 10, 89, 213, 255
stream = Encoder(payload).encode()
```

Figure 4.3: Usage of encoder

```python
from encoding import Decoder

stream1 = [22, 22, 104, 101, 108, 108, 111, 0, 0, 0, 0, 0]
stream2 = [0, 0, 0, 0, 0, 0, 210, 201, 10, 89, 213, 255]

decoder = Decoder()

for byte in stream1:
  decoder.add(byte)

decoder.can_decode() # False

for byte in stream2:
  decoder.add(byte)

decoder.can_decode() # True

# decoded single frame
# payload == [104, 101, 108, 108, 111, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
# frame_count == 1
payload, frame_count = decoder.decode()
```

Figure 4.4: Usage of decoder

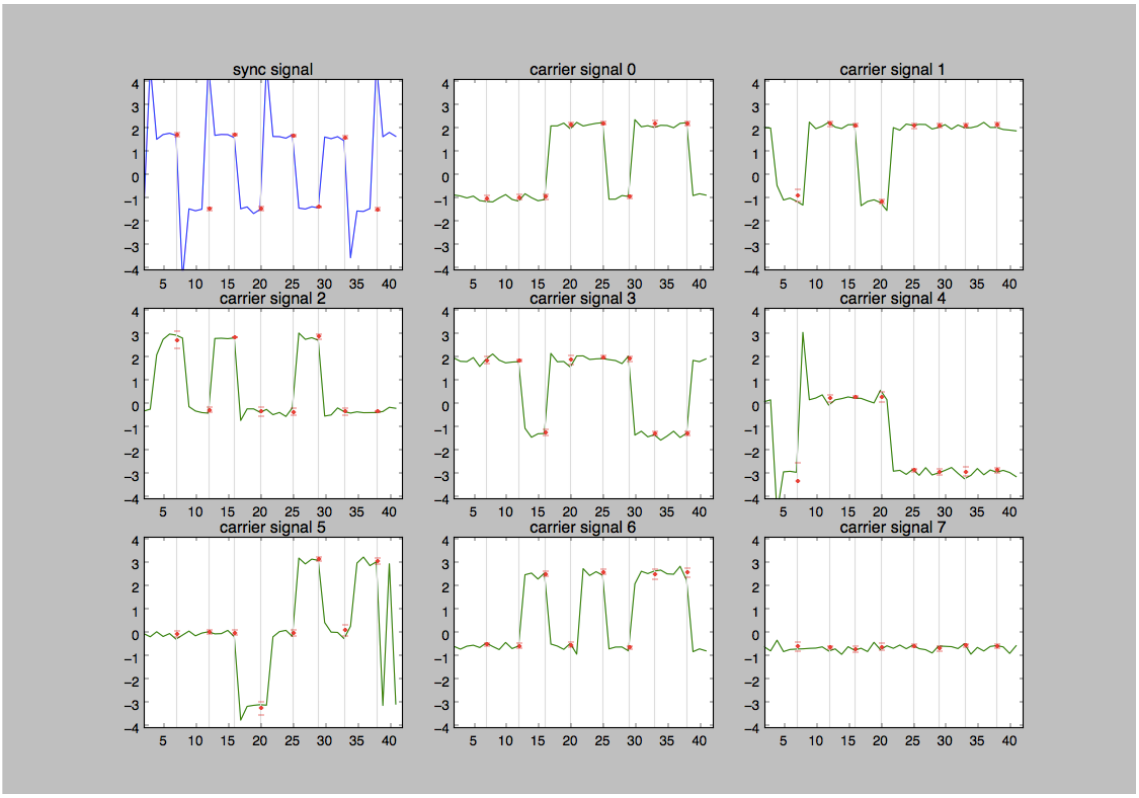


Figure 4.5: An example screen of graph utility

Chapter 5

Tests

5.1 Tests outline

Two kinds of tests examining different sides of solution are performed:

- performance tests – laboratory tests examining designed solution's performance in various conditions
- psychoacoustic test – experiment examining audibility of signal hidden by presented solution

In every type of test, synthesis / analysis algorithms are parameterized with following values:

- byte duration = 200 ms
- carrier signal band = 8000–8800 Hz

With this setup, the solution achieves effective data bandwidth of 26.66 bps (bits per second).

Unfortunately tests are not performed on mobile platform because of longer time required for implementation. All performance tests are conducted in acoustic laboratory using stationary devices.

5.2 Performance tests

5.2.1 Overview

The purpose of performance tests is to examine the possibilities of implemented solution. Information about algorithm effectiveness is expected as an procedure output. Performance tests are conducted in a laboratory environment to enable environment manipulation. To get most informative output data and effective testing environment, following features are required:

- value-neutrality – any kind of unwanted environmental noise is eliminated
- versatility – various environment conditions are simulated
- repeatability – tests are easy to conduct and repeat many times in the same form.

To fulfill the requirements, anechoic chamber and professional recording / playback facilities are employed. See Figure 5.1 for block diagram of test environment setup.

5.2.2 Environment setup

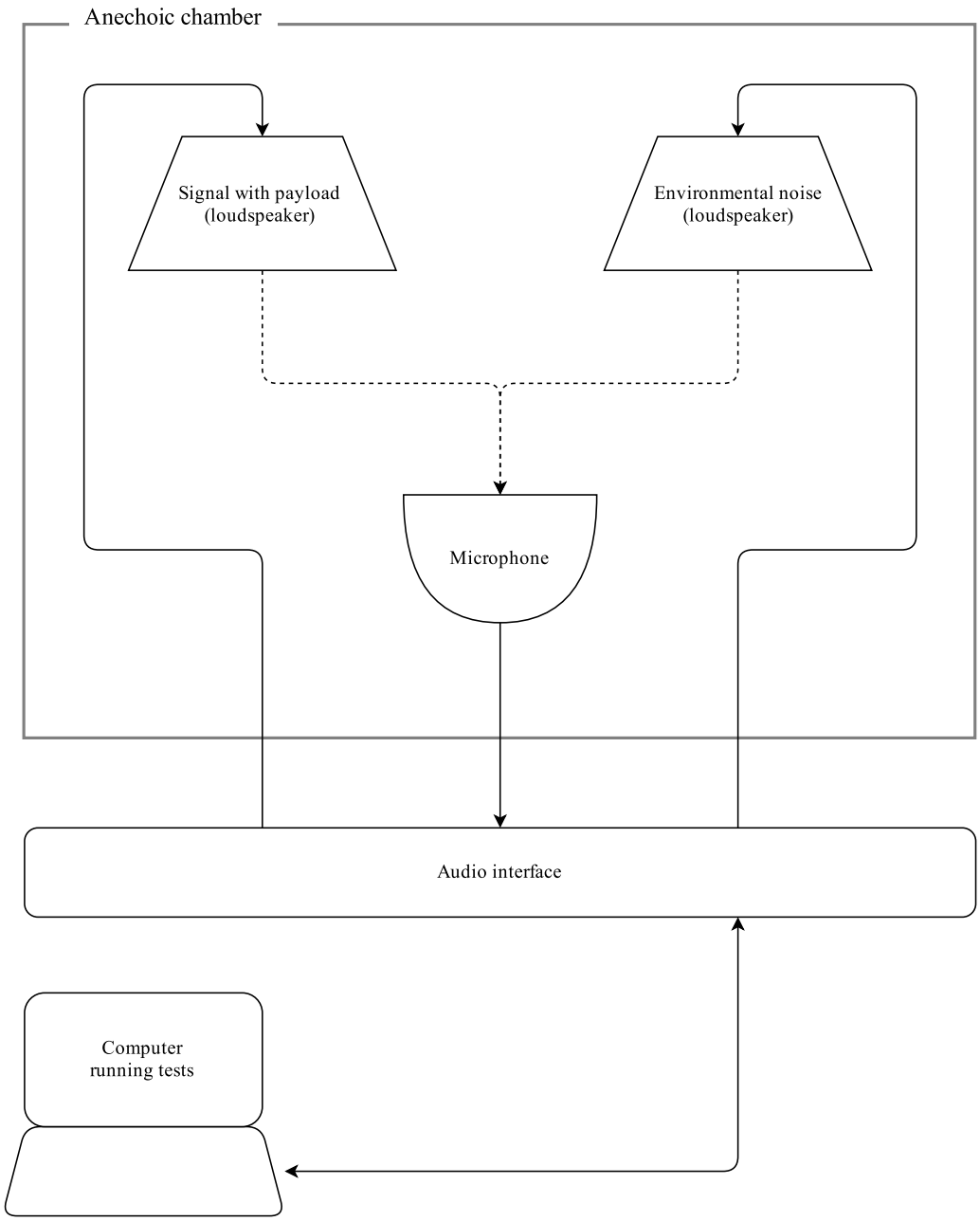


Figure 5.1: Test environment diagram

Two loudspeakers are directed to microphone in monophonic mode. Microphone output is connected to audio interface input. Loudspeakers are connected to separate interface outputs. Computer plays encoded signal on one speaker and environmental noise on the other one. At the same time, audio signal is recorded using the microphone. Recorded samples are then analyzed using developed solution. The decoded payload is compared to original and Bit Error Rate (BER) is computed.

Following equipment is used to realize described solution:

- 2 × Tannoy 800A Active monitor loudspeakers
- Shure VP88 MS-stereo-condenser microphone
- Focusrite Scarlett 18i20 USB Audio interface
- MacBook Air with OS X Yosemite (10.9)

5.2.3 Implementation

Test software is implemented in python programming language, like the algorithm library. The language was chosen to stay consistent across entire solution. It is also easy to utilize encoding / decoding library written in the same language. The `Experiment` class is responsible for conducting performance tests.

Public methods:

- `Experiment(host, noise, host_db, noise_db, payload)` Constructor

  Arguments:

  - `host` host signal file path (`string`)
  - `noise` noise signal file path (`string`)
  - `host_db` amplitude of host signal in dB (`float`)
  - `noise_db` amplitude of noise signal in dB (`float`)
  - `payload` bytes to be transmitted (list of `int`)

- `run()`

  When this method is called, experiment procedure starts. After conducted experiment, all parameters with decoded payload are appended in python format to `experiment_output.txt` file.

  Return value: `decoded_payload`

  - `decoded_payload` – payload decoded from recorded sample (array of `int`)

- `BER(correct, computed)`

  Static class method. It reckons bit error rate for decoded payload data.

  Arguments:

  - `correct` – original payload (array of `int`)
  - `computed` – decoded payload (array of `int`)

  Return value: `bit_error_rate`

  - `bit_error_rate` – reckoned bit error rate

See Figure 5.2 for example usage of `Experiment` class in an application.

```
from experiment import Experiment

host_path = 'host.wav'
noise_path = 'noise.wav'
payload_string = "Lorem ipsum dolor sit amet consectetur adipisicing elit"
payload = [ ord(l) for l in payload_string ]

experiment = Experiment(host_path, noise_path, 0.0, -10.0, payload)
decoded_payload = experiment.run()

ber = Experiment.ber(payload, decoded_payload)
```

Figure 5.2: Example usage of experiment class in an application

5.2.4 Sound samples

Two types of sound samples are needed to conduct performance tests:

- host samples – three sampled pieces of music to host encoded payload
- noise samples – four various environmental noises

All audio samples used in tests are 24-bit monophonic WAV files. All spectrum charts have red-marked region in frequency range used by presented solution. In case of noise signals – they should disturb decoding process as much as there is power in this range.

Noise samples

All environmental noise is recorded especially for the purpose of the project. From many recorded sounds, only three probes were selected. The selection is based on spectral analysis. Only sound samples containing much power in carrier signal band are used. Following four pieces were selected:

- `ambulance.wav` – ambulance emergency signal (See: Figure 5.3)
- `bottle-scratch.wav` – sound of scraping glass bottles (See: Figure 5.4)
- `crowd.wav` – speaking crowd (See: Figure 5.5)
- `keys.wav` – clanking keys (See: Figure 5.6)

To record these sounds following equipment has been used:

- Tascam DR-40 Linear PCM hand-recorder
- Shure VP88 MS-stereo-condenser microphone with foam windscreen

Noise samples spectral charts:

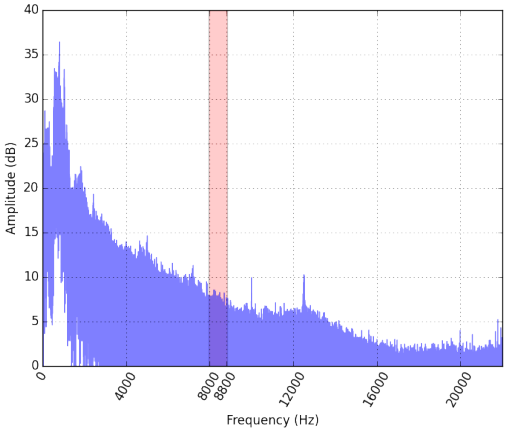


Figure 5.3: ambulance.wav spectrum

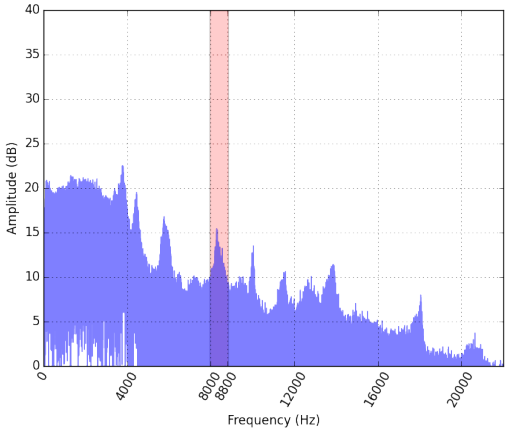


Figure 5.4: bottle-scratch.wav spectrum

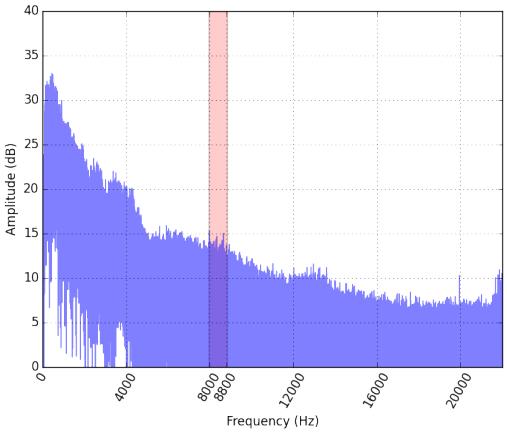


Figure 5.5: crowd.wav spectrum

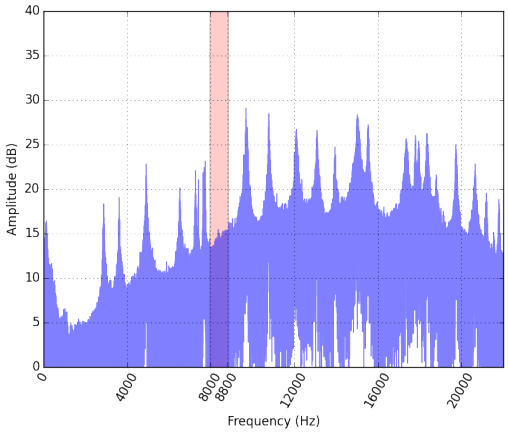


Figure 5.6: keys.wav spectrum

Host samples

An important requirement is to make the carrier signal inaudible. In the experiments, only musical pieces are used as a host signals, as music contains various partials which can be used for masking. Musical genres diversity is guaranteed. Three host signals are selected for testing purposes:

- `bach.wav` – (classical) fragment of "Gloria in excelisis deo" composed by J.S. Bach (See: Figure 5.7)
- `rockvocals.wav` – (rock) fragment of "Can't stop" by Red Hot Chilli Peppers(See: Figure 5.8)
- `skalpel.wav` – (hip-hop) fragment of "Adventures in space" by Skalpel (See: Figure 5.9)

Host samples spectral charts:

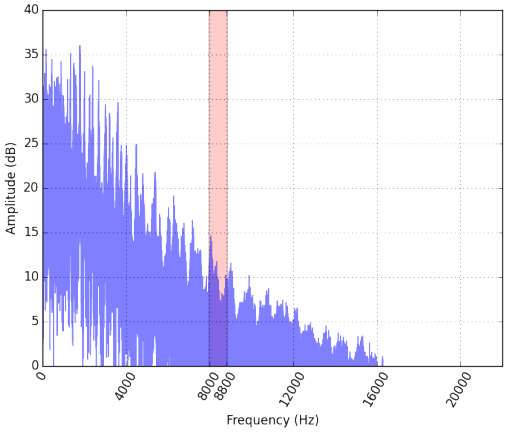


Figure 5.7: bach.wav spectrum

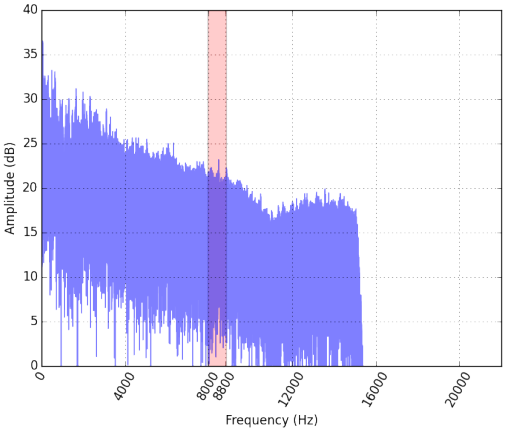


Figure 5.8: rockvocals.wav spectrum

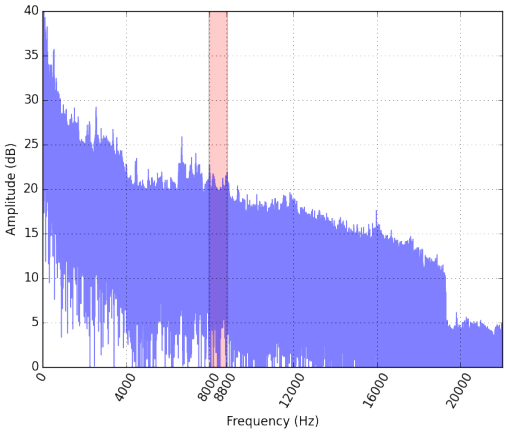
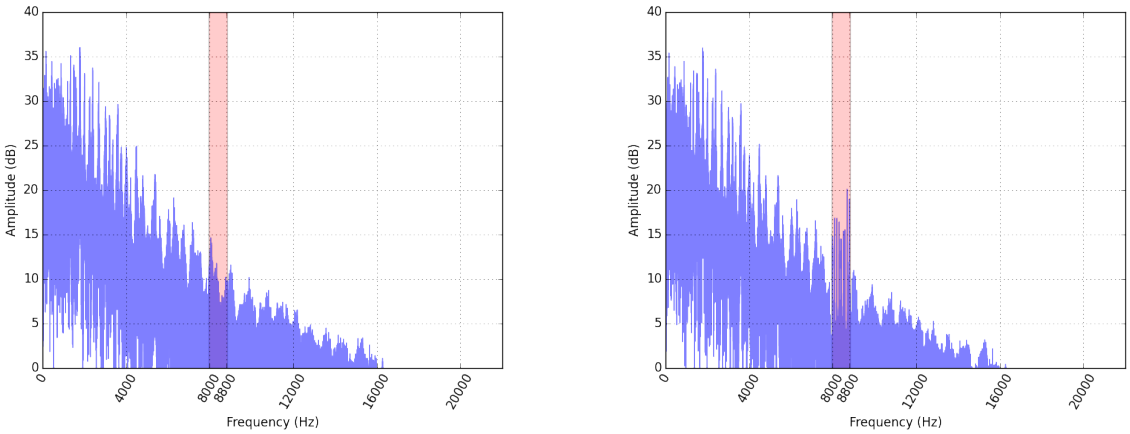


Figure 5.9: skalpel.wav spectrum

Sound sample with hidden information

After insetting encoded payload information into the host signal, spectrum changes significantly. The amount of influence that can be seen on spectrum depends on psychoacoustic model adjustment of amplitude, which in turn depends on number and power of maskers around chosen frequency band.



(a) Original signal

(b) Signal with an inset information

Figure 5.10: Difference between signal spectrum

5.2.5 Test results

Tests are performed according to following rules:

1. for each host signal take every noise signal
2. for each `(host, noise)` pair take every amplitude from `[-30,-20,-15,-10,-5,0]` dB
3. perform test for each `(host, noise, amplitude)` tuple

All tests are performed three times and average from their BER is reckoned. In most cases BER is equal to 0, however there are two particular cases worth explaining.

Frame detection failure

This situation causes very high BER value. It happens when the system is unable to recover frame beginning and it waits till next frame. In effect it omits single bit of data. It causes a shift of entire decoded data by 1 byte to the left. It is dangerous situation, because decoded data can be completely useless even if this type of failure does not seem to be a big mistake.

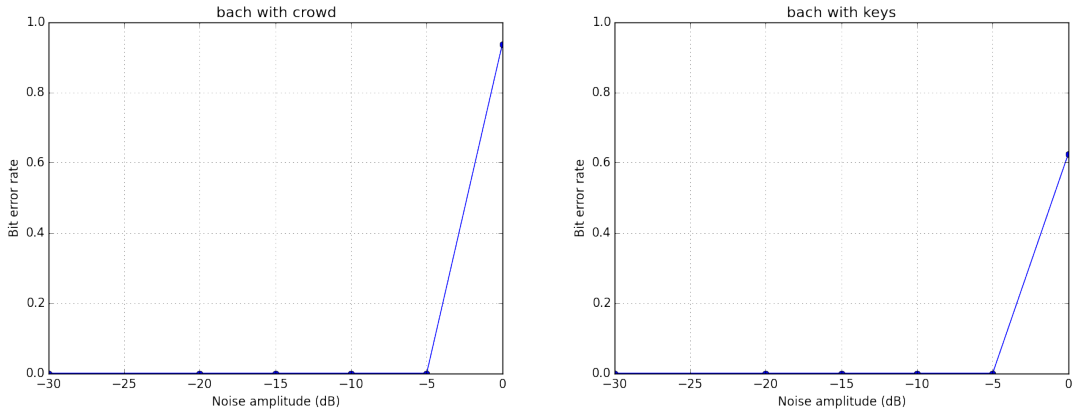


Figure 5.11: Frame detection failure

Single bit mis-decoding

It is the case when the system decodes single bit improperly. It raises the BER index, but not as much as in the frame detection failure case. In this situation the recovered payload is more likely to be usable, because it is not shifted.

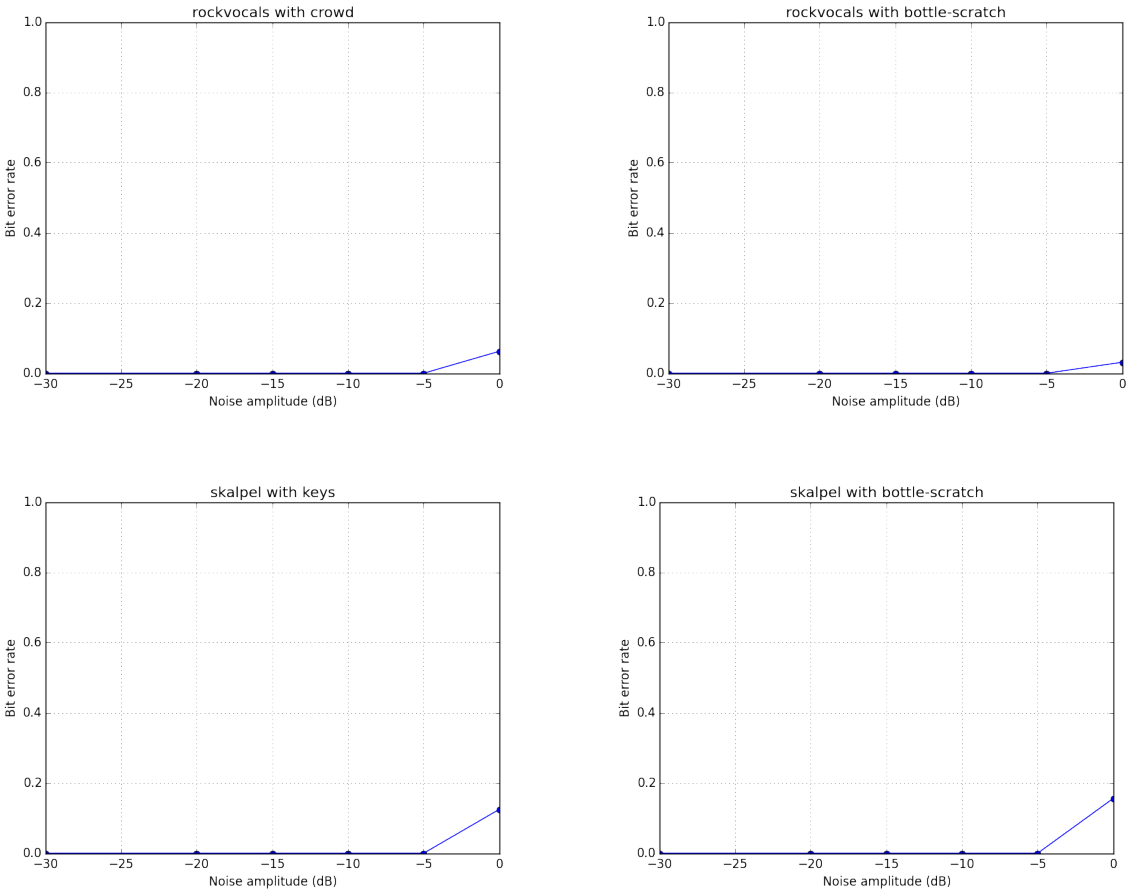
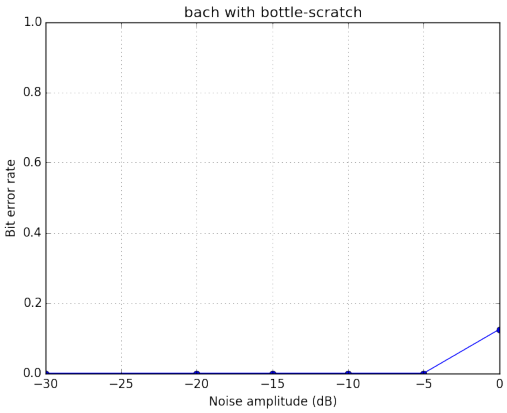


Figure 5.12: Single bit mis-decoding



5.3 Psychoacoustic test

5.3.1 Overview

A purpose of this type of test is to examine how human auditory system (See: 2.5.1) reacts to data hidden in host audio signal. The goal of presented solution is to make hidden carrier signal inaudible for humans. The experiment is conducted on a sample of 20 people.

5.3.2 Experiment process

1. two versions of every examined sound sample are prepared. The first one with original sound and the second one with hidden carrier signal
2. a test form is provided to each participant. The form contains three rows (sample names) and two columns (sample versions)
3. for each examined sound sample pair a coin toss is performed. Toss result determines sound sample playback order
4. participant listens to two sound samples
5. participant puts a mark in column corresponding to the version they think is deformed by the hidden carrier
6. filled forms are compared to the reference form containing only sound samples correctly identified as containing payload

5.3.3 Test results

Obtained results indicate significant problems with differentiating original sound from sound with hidden data. It means signal is well-hidden and inaudible (in most cases) for humans.

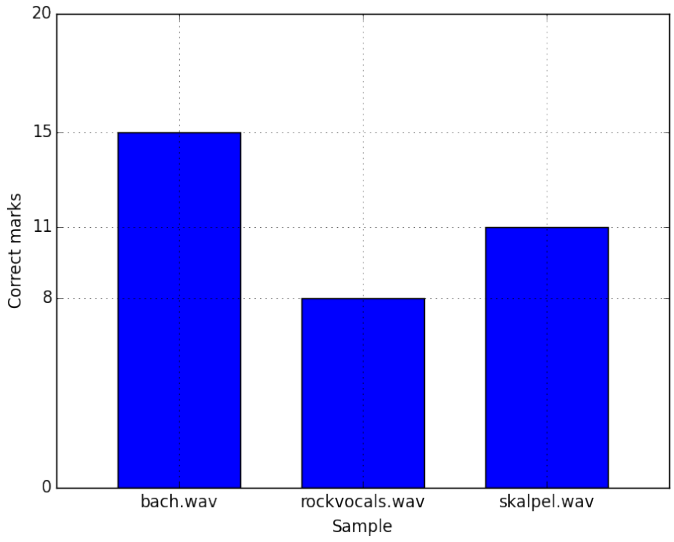


Figure 5.13: Results of psychoacoustic test

As it can be seen on above chart, the most accurate identifications belong to `bach.wav` sound sample – less opportunity for masking causes easier recognition of signal deformation. See Figure 5.7 for `bach.wav` spectrum.

5.4 Conclusions

The performed tests prove the requirements are met:

- hidden signal is inaudible for humans
- encoded payload is recoverable from the recorded sound sample.

However, designed solution is not perfect. Tests created a possibility to notice major issues like frame detection failure cases (see: 5.2.5). Although it can be a significant trammel in using developed library, solution still handles really difficult environment conditions. All failure recovery tries are related to noise source set to maximum tested amplitude. In all cases when noise signal amplitude is lower than encrypted host amplitude, receiver decodes data correctly. The system needs only little enhancement to become profoundly usable python library for hiding data in audio signals.

Chapter 6

Summary

The thesis presents a solution for insetting and recovery of data in sound in imperceptible way, as well as implementation of the solution.

The solution is shown to allow a robust transmission in presence of interference (in laboratory conditions) and to be imperceptible to human listeners (See 5.3).

6.1 Possible applications

The python library developed as a part of this thesis can be utilized on many platforms that support python programming language.

Example applications might include:

1. In marketing – embedding data in advertisements.

   Data like product details or promotional coupons could be hidden in advertisement music. Consumers would be able to extract and use this data using smartphone application.

2. In broadcasting – embedding meta-data in music.

   Meta-data like artist name, song title or even song lyrics could be hidden in music broadcasted on live events. Listeners would be able to access it using mobile application.

3. In digital rights management – a simple watermark.

   Watermark data could be hidden in music or film score to detect copyright infringement. The disadvantage of presented solution is ease of watermark removal.

6.2 Possible improvements

- Increase data bandwidth

  Transmission bandwidth achieved in the implemented solution might limit the number of applications. Bandwidth could be increased by decreasing single byte transmission duration and using phase recovery method less susceptible to noise.

- Improve frame recovery algorithm

  Implemented frame recovery algorithm is shown to fail to correctly detect a frame in some conditions (see 5.2.5). This could be improved by using longer preamble or another method for synchronizing frames.

- Evaluation in real-life conditions

  The tests in Chapter 5 are performed in laboratory conditions. Additional tests should be performed to evaluate the system's performance in presence of natural phenomena (like echoes) and using commodity hardware.

- Advanced psychoacoustic model

  Another psychoacoustic model might be implemented, exploiting additional properties of human auditory system, like temporal masking (2).

- Implementation of mobile application

  A showcase application for easy broadcasting and receiving of data, running on any of major mobile operating systems.

6.3 Acknowledgments

Authors would like to thank prof. dr. hab. inż Adam Dąbrowski and dr. Szymon Drgas for support in designing a solution, and dr. inż. Andrzej Meyer for help with laboratory and audio equipment.

List of Figures

Bibliography

[App13] Applidium. Audio modem: data over sound. [on-line] `http://applidium.com/en/news/data_transfer_through_sound`, 2013.

[BB97] K. Brandenburg and M. Bosi. Overview of mpeg audio: current and future standards for low-bit-rate audio coding. *J. Audio Eng. Soc.*, 4, 1997.

[Cla02] C. K. P. Clarke. Reed-solomon error correction. *BBC Research & Development*, 2002.

[DM02] A. Dąbrowski and T. Marciniak. *The Computer Engineering Handbook*, chapter Audio Signal Processing. CRC Press LCC, 2002.

[JN84] N. S. Jayant and P. Noll. *Digital coding of waveforms: principles and applications to speech and video.* Prentice-Hall, Englewood Cliffs, NJ, 1984.

[Moc87] P. Mock. Add dtmf generation and decoding to dsp-up designs. *Digital Signal Processing Applications with the TMS320 Family, vol. 1.*, 1987.

[Moo97] B. C. J. Moore. *An introduction to the psychology of hearing.* Academic Press, London, 1997.

[NL99] P. Noll and T. Liebchen. Digital audio: from lossless to transparent coding. In A. Dabrowski, editor, *Proc. IEEE Signal Processing Workshop.* IEEE Poland Section, Chapter Circuits and Systems, Poznan, Poland, 1999.

[OS89] A. Oppenheim and R. Schafer. *Discrete-time signal processing.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[Smi97] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing.* California Technical Pub, 1997.

[Smi07] J. O. Smith. Introduction to digital filters, with audio applications. [on-line] `https://ccrma.stanford.edu/~jos/filters`, 2007.

[Smi09] J. O. Smith. Mathematics of the discrete fourier transform, with audio applications. [on-line] `http://www.dsprelated.com/dspbooks/mdft`, 2009.

[SR12] P. Sysel and P. Rajmic. Goertzel algorithm generalized to non-integer multiples of fundamental frequency. *EURASIP Journal on Advances in Signal Processing*, 2012:56, 2012.

[Vir] A. et al. Virani. W. a. v. s. compression. [on-line] `http://www.aamusings.com/project-documentation/wavs/index.html`.

[WB94] S. B. Wicker and V. K. Bhargava. *Reed-Solomon codes and their applications.* 1994.

[Zwi61] E. Zwicker. Subdivision of the audible frequency range into critical bands. *The Journal of Acoustical Society of America*, 1961.

Poznan University of Technology
Faculty of Computing Science
Chair of Control and Systems Engineering

Typeset using LaTeX in Computer Modern.

BibTeX:

```
@mastersthesis{ key,
    author = "Błażej Kotowski \and Tomasz Pewiński",
    title = "{Applications for insetting and recovery of hidden information audio signals}",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2015",
}
```