Communication and Data Transmission via Sound Waves

Maturita Project

Author:	Teodor Machart
Class:	4.C
School year:	2021/2022
Subject:	Computer Science and IT
Supervisor:	Šimon Schierreich

Prague, 2022

GYMNASIUM JANA KEPLERA
Computer Science and IT

MATURITA PROJECT ASSIGNMENT

Student:	**Teodor Machart**
Class:	**4.C**
School year:	**2021/2022**
Validity:	**22. 10. 2021**
Supervisor:	**Šimon Schierreich**
Title:	**Communication and Data Transmission via Sound Waves**

Instructions:

I will endeavour to create a program which would facilitate communication between two devices using nothing but their loudspeakers and microphones to emit and receive the sound-carried data. I intend the paper to consist of two segments. The first segment being theoretical and covering instructions on how to encode data onto and subsequently decode data from a sound wave. The second segment being practical and implementing the theoretical findings. The practical segment shall also include measurements on how is the quality of transmission affected by different parameters at play and what impact do these parameters have on the detectability of the transmission process by an outsider.

References:

[1] MOSER, Stefan M. a Po-Ning CHEN. A student's guide to coding and information theory. New York: Cambridge University Press, 2012. ISBN 9781107601963.

[2] ROMAN, Steven. Coding and information theory. New York: Springer-Verlag, 1992. ISBN 978-0-387-97812-3.

[3] BUSHWICK, Sophie. Ultrasonic Attack Device Hacks Phones through Solid Objects. Scientific American [online]. 2020, 322(3). ISSN 0036-8733. Available at: https://www.scientificamerican.com/article/ultrasonic-attack-device-hacks-phones-through-solid-objects/

Repository URL:

https://github.com/MachTe/Maturitni_prace

_____   _____
student   *supervisor*

Prague October 22, 2021

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů. Nemám žádné námitky proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Praze dne March 20, 2022   Teodor Machart

Acknowledgement

Abstrakt

Tato práce pokládá na platformě nezávislé teoretické základy pro přenos dat za pomoci neslyšitelného ultrazvuku, následně dosažené výsledky demonstruje pomocí konkrétní implementace. Práci doplňují, takzvaně, „ready-to-use" Python soubory, které poslouží komukoli, kdo se snaží tento koncept pochopit a následně použít.

Klíčová slova

Abstract

This thesis lays out theoretical, platform-independent groundwork for inaudible data transfer via ultra-sound, it than continues by showcasing concrete implementation of such theoretical findings. The paper is complemented by ready-to-use Python files which ought to help anyone trying to get their head around the concept.

Keywords

Contents

1. Theoretical Segment

1.1   Introduction

Scientific American has recently published an article [1] which describes an elaborate attack on smartphones through ultra-sonic audio. Software utilising similar technology for legal purposes, like sending data, can be found on the internet, but sadly all of it is proprietary and undisclosed to the public, thus the lack of public information regarding this technology calls for an open-source solution, call, which shall be answered by this paper.

The paper aims to offer theoretical platform-independent solution to the problem of sending data via inaudible sound so that anyone can build upon it and implement it to their liking. Following chapters focus on a concrete implementation of the theoretical findings in order to prove the worth of the technology.

1.2   Frequency Impulse

For the purpose of this paper a single frequency is used to carry binary data, an impulse of this carrier frequency is seen as a binary 1 and a silence is seen as a binary 0, as can be seen on figure 1. More frequencies can be used to increase the data transmission speed, this matter is discussed in subsection 1.1.2.

figure 1. Binary data encoded in sound impulses

Unlike electricity or radio-signal, sound produced by audio-speakers cannot create square waves, the closest thing would be to maintain the amplitude of the transmitted frequency for a certain amount of time and then cut it off (figure 2 on the left). Such an abrupt change produces a crackling sound in the speaker, therefore smoothening of the edges is needed (figure 2 on the right). This result could be obtained for example through convolution by Gaussian function.
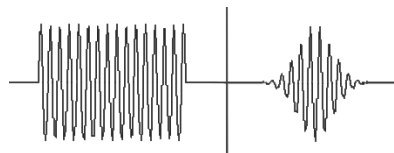

figure 2. Unconvoluted impulse and convoluted impulse

n-th sample of the sound impulse can than be obtain via this equation:

$$S(n) = \sin\left(2\pi\,\frac{f\,n}{f_s}\right)\cdot\delta^{\left|\frac{2n}{l}-1\right|^{\kappa}} \tag{1.1}$$

$$
\begin{aligned}
& l \in \mathbb{Z}^{+} && f_s \in \mathbb{R}^{+} \\
& f \in \mathbb{R}^{+} \;\; ; \;\; f < \frac{f_s}{2} && n \in \mathbb{Z} \;\; ; \;\; n \in \langle 0\,;\,l\rangle \\
& \delta \in \mathbb{R} \;\; ; \;\; \delta \in (0\,;\,1) && \kappa \in \mathbb{R}^{+}
\end{aligned}
\tag{1.2}
$$

Where l is the length of the impulse in samples; f_s is the sample rate of the the desired sound, industry standard is 44100 Hz; f is the frequency of the impulse; δ and κ are parameters of the Gaussian function. κ sets how gradual the transition from an impulse to a silence is, whilst, δ is the y value of the Gaussian function at the end of the frequency impulse. Figure 3 illustrates the influence of these two parameters on the shape of the frequency impulse.
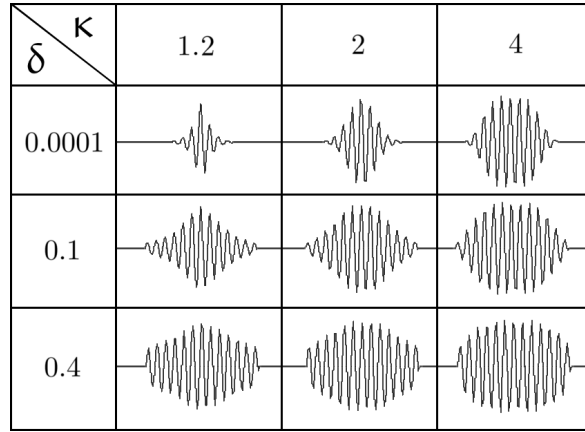

figure 3. Influence of κ and δ on shape of the impulse

Values of κ and δ are determined experimentally in the implementational segment based on the quality of transmission they offer.

1.2.1   Multiple Frequencies

The idea of using frequency impulses to transfer data can be expanded upon by using multiple frequencies. Such advantage can be harnessed in different ways with different degrees of implementational difficulty and crucially with different degrees of success.

First approach is just an extension of the binary one which is used throughout this thesis. Each impulse is made of just one of the n frequencies or silence, and hence carries $\log_2(n+1)$ bits of data.

In the second approach each impulse consists of a sum of any combination of the n frequencies, and hence carries $\log_2(2^n) = n$ bits of data. This approach easily outperforms the first one, nevertheless, it has many drawbacks to it, chief among them being the fact that the sum of two inaudible frequencies tends to create crackling sound when run through low-quality speakers.

1.3   Transmission Protocol

As has been hinted in the introduction, the transmission is meant to be one-way, meaning, one of the devices acts as a transmitter and one as a receiver. Therefore, communication protocol is reduced to a simple sequence of chunks of data that are to be transmitted as depicted in table 1.

Chunk Order	Name	Size
1 st	Series of ones	5 Bytes
2 nd	Initial sequence : 11010001	1 Byte
3 rd	Error-correction symbols for the 2 nd chunk	2 Bytes
4 th	Length of the 6 th chunk in Bytes = k	2 Bytes
5 th	Error-correction symbols for the 4 th chunk	2 Bytes
6 th	Transmitted data and its error-correction symbols.	k Bytes

table 1. Transmission Protocol

The first chunk constitutes 40 ones, they serve to alert the receiver that data is being transmitted. On this chunk the receiver measures the average amplitude of the carrier frequency and locates the peaks of the individual frequency impulses, so that the area where the frequency is examined overlays precisely the impulses from that moment on. The first impulse in the first chunk is not guaranteed to trigger the receiver, therefore, the size of this chunk has to be slightly larger than what is needed for the aforementioned procedures.

The second and third chunks allow the receiver to precisely pinpoint the start of the transmitted message. the fourth chunk cannot be used for this purpose as it is not constant and changes based on the length of the message. The sequence 11010001 has been chosen randomly.

The fourth and fifth chunks inform the receiver about the length of the data with its error-correction symbols. The program than terminates when it has received the required number of Bytes. Two Bytes allow for maximal length of 65535 Bytes which is satisfactory for the purposes of this paper, nevertheless, the fourth chunk can be expanded to more Bytes based on individual needs.

The sixth chunk contains the data itself and its error-correction symbols. Any binary data can be send, ranging from simple text to images and videos, although text is by far the easiest to send, as it is naturally chopped up into Bytes.

1.4   Error-correction

Due to ambient noise, some bits are bound to flip during the transmission process, to account for this, some kind of safety mechanism has to be introduced. In this case simple parity check or some variation of Hamming code will not do, as the data is expected to get corrupted in "bursts", meaning, flipped bits will not be spread equally across the data, but rather will be concentrated to a few heavily

damaged spots. On these grounds an error-correction code called Reed-Solomon Code had to be chosen for this project as it is highly effective against such "bursts".

Reed-Solomon Code takes in a sequence of symbols and returns the same sequence of symbols plus a sequence of error-correction symbols. (m-bit symbol is any sequence of ones and zeros of length m) If any of the symbols gets corrupted during the transmission, it can be reconstructed, regardless of how many bits have flipped within that symbol. The number of symbols that are protected against flipping depends on the number of error-correction symbols.

According to [3], Reed-Solomon Code (n, k) on m-bit symbols exist if this condition is satisfied:

$$1 < k < n < 2^m + 2 \tag{1.3}$$

$$n, k, m \in \mathbb{N} \qquad\qquad m \geqslant 2 \tag{1.4}$$

Where k is the number of symbols our data is made of; n is the number of error-correction symbols $+ k$; m is the length of a single symbol.

For the purposes of this paper m has been chosen to be 8 as it coincides with the number of bits needed to describe a single letter in IBM852 encoding (encoding used in the Implementational Segment). In the transmission protocol an R-S $(3, 1)$ is used to protect the second chunk, an R-S $(4, 2)$ to protect the fourth chunk, and an R-S $(255, 225)$ to protect the sixth chunk. (If the length of the data exceeds 225 symbols, it is chopped up into segments of 225 symbols and a remainder.)

The number of Symbols protected against corruption during transmission can be calculated as:

$$t = \left\lfloor \frac{n - k}{2} \right\rfloor \tag{1.5}$$

where $\lfloor x \rfloor$ is the largest integer not exceeding x.

The inner workings of Reed-Solomon codes are mathematically very complex, and could be intuitively best understood through matrix multiplication, where the entry data is cleverly divided into matrices (A and B) which are than multiplied and the product serves as the error-correction symbols:

$$AB = E \tag{1.6}$$

If any one of the three matrices were to be corrupted during the transmission, it could be later retrieved thanks to the remaining two:

$$A = EB^{-1}$$
$$B = A^{-1}E \tag{1.7}$$
$$E = AB$$

The implementational segment uses pre-written library to achieve this, hence an intuitive understating will suffice, thorough explanation is given in a paper by Bernard Sklar [3].

1.5   Sound Processing

Sound recorded by the receiver contains ambient noise and inaccuracies caused by both the microphone and the speakers, thus it becomes imperative to use signal processing techniques to extract comprehensive data. Discrete Fourier Transform is used to isolate the amplitude of the carrier frequency f from the data. Amplitude X of the frequency f on the sequence x of the length N is, according to Julius O. Smith[4], calculated in this manner:

$$X(f) = \left| \sum_{n=0}^{N-1} x(n) e^{-j2\pi n f/f_s} \right| \tag{1.8}$$

If the length of the sequence x is set equal to the length of the impulse, and it is precisely aligned with the impulses in the recorded sound, it can than be determined whether the amplitude is high enough to represent a binary one or low enough to represent binary zero.

2. Implementation

The transmission of data occurs between two devices each of which runs independently its own program, these respective programs require certain functionalities which are listed below, furthermore, the programming language of choice for both programs is Python as none of the functionalities are computationally expensive, and thus real-time processing can be achieved even with a less speed-friendly language such as Python.

Transmitter

- Data input
- Encoding / Creating error-correction symbols
- Creating desired frequency impulse
- Translating binary data into audio data
- Playing aloud the audio data

Receiver

- Recording audio and saving it to a buffer
- Computing amplitude of a certain frequency within the buffer
- Locating the precise beginning of the transmission
- Decoding / reconstructing the data with the help of error-correction symbols
- Displaying the result

In addition, both devices need to be synchronized when it comes to the carrier frequency, the impulse length and the number of error-correction symbols.

2.1 Transmitter

For simplicity's sake an input from console serves as the data which is to be transmitted. As mentioned in section 1.3, the most straight forward approach to the transmission of textual data is to choose a single symbol to be synonymous with a single byte. Complementary error-correction symbols are than obtained thanks to a Python library called Reedsolo which can be installed through the console command: `pip install reedsolo`. These symbols are coupled with remaining chunks, as shown in the transmission protocol (table 1). Resulting binary sequence is than converted into a `wav` audio file as illustrated by figure 1, and played aloud.

The last two steps are unfortunately hardware-dependent and are not guaranteed to work on all devices, whilst the quality of speakers is crucial when choosing the values of δ, κ and the carrier frequency. Speakers of lower quality require lower values of both δ and κ which makes the impulse weaker and harder to intercept; meanwhile, the frequency response of the speakers might dampen sounds on frequencies above 20 kHz. This leaves no other option but to determine those values experimentally and separately for each device (section 2.3).

2.2   Receiver

The receiver is split into two threads, one continuously records audio and saves it to a buffer, and the other one goes through the buffer and looks for the data. The split is necessary because otherwise gaps might occur in the recording.

As soon as the recording starts, an average amplitude $\overline{X}$ of the carrier frequency within the ambient noise is measured. In practice, amplitude is measured on the first ten sequences of the length l and than averaged out. From that point on a rolling average X^o over 3 sequences of the length l is measured, if the condition (2.1) is satisfied the first chunk is sure to have been intercepted.

$$X^o > 10 \cdot \overline{X} \tag{2.1}$$

The number ten has been picked manually and does not seem to have substantial influence on the ability of the program to recognise the first chunk. Any value from the interval $\langle 5 \, ; \, 20 \rangle$ works just as fine.

Now the receiver is receiving a constant stream of binary ones, impulse after impulse. At this point a frequency amplitude is measured on sequences of length $\frac{l}{2}$ everywhere within the length of one impulse. This measurement peaks when the respective sequence overlays the center of the received impulse, thus the exact position of the impulse can be determined, and by extension also the exact position of all the following impulses. Thanks to this knowledge, the frequency amplitude measurement can now be done on sequences exactly aligned with the received impulses, enabling the execution of all the procedures described in the section 1.2.

Next 20 impulses are used to measure X_i, the average frequency amplitude of an impulse. Based on this value, the coming bits are determined to be one or a zero. Due to the noise within the data a threshold has to be set up so that the maximum number of bits can be determined correctly. An Impulse is considered to carry binary one, if the amplitude $X(f)$ on the impulse satisfies following condition, where t stands for threshold.

$$X(f) > X_i(1 - t) \tag{2.2}$$

The optimal value of t is obtained experimentally in the section 2.3. Microphones similarly to speakers have different frequency responses, lower-quality microphones might dampen higher frequencies, and hence the experimental value is not universal.

Once the program is able to distinguish between ones and zeros, the following steps are straight forward.

The threshold is likely to be hardware-dependent as microphones exhibit different degrees of hardware induced noise, therefore the experimental value is not universal.

2.3 Determining the values of the parameters δ, κ, t, f, l

Let the function $g(\delta, \kappa, t, f, l)$ be defined as the number of correctly transmitted bits divided by the overall number of bits. The values for which g peaks are the optimal values. Unfortunately no mathematical expression is at disposal, thus the only option is to measure g for different sets of parameters. The number of measurements rises exponentially with the fifth power of the resolution of the parameters, if 25 sample values per parameter were to be selected, almost 10 million measurement would be needed which would take several months to complete. Obviously the measurement needs to be simplified under certain assumptions:

First, let it be assumed that g is not a function of l.

$$\delta_0, \kappa_0, t_0, f_0 \in \mathbb{R}$$
$$\forall\ \delta, \kappa, t, f \in \mathbb{R}: \quad g(\delta_0, \kappa_0, t_0, f_0) \geqslant g(\delta, \kappa, t, f) \tag{2.3}$$

Second, let it be assumed that the following two expressions hold true.

$$\forall\ \delta, \kappa, t, f \in \mathbb{R}: \quad g(\delta_0, \kappa_0, t, f) \geqslant g(\delta, \kappa, t, f) \tag{2.4}$$

$$\forall\ \delta, \kappa, t, f \in \mathbb{R}: \quad g(\delta, \kappa, t_0, f_0) \geqslant g(\delta, \kappa, t, f) \tag{2.5}$$

Expression (2.4) says that δ_0 and κ_0 can be determined by choosing t and f to be random constants c_1 and c_2, and finding where the function $g(\delta, \kappa, c_1, c_2)$ peaks. Similarly, expression (2.5) says that t_0 and f_0 can be determined by choosing δ and κ to be random constants c_3 and c_4, and finding where the function $g(c_3, c_4, t, f)$ peaks. Under these assumptions the 2D chart becomes feasible as the needed time is significantly shortened and rises just with the second power of the resolution of the parameters.

2.3.1 Determining δ_0 and κ_0

Each parameter has been given 25 sample values, each pair of sample values has been measured ten times and averaged out, thus bringing the overall number of measurements to 6250. Given the duration of a single measurement which is roughly 1.6 seconds, the entire procedure took just under three hours.
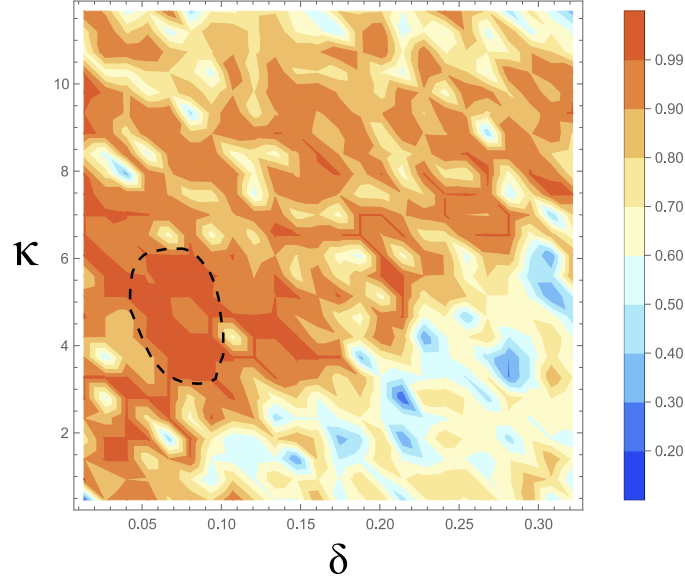

figure 4. Function $g(\delta, \kappa, c_1, c_2)$

The chart of the function $g(\delta, \kappa, c_1, c_2)$ definitely rules out the bottom right corner as well as the very top of the chart. The rest of the field seems to be even, with the notable exception of the encircled area, where g has even surpassed the 0.99 mark, therefore the parameters are selected to be:

$$\delta_0 = 0.07 \qquad \kappa_0 = 4 \qquad\qquad (2.6)$$

2.3.2 Determining t_0 and f_0

Identically to 2.3.1 each parameter has been given 25 sample values, each pair of sample values has been measured ten times and averaged out, thus bringing the overall number of measurements to 6250. Given the duration of a single measurement which is roughly 1.6 seconds, the entire procedure took just under three hours.
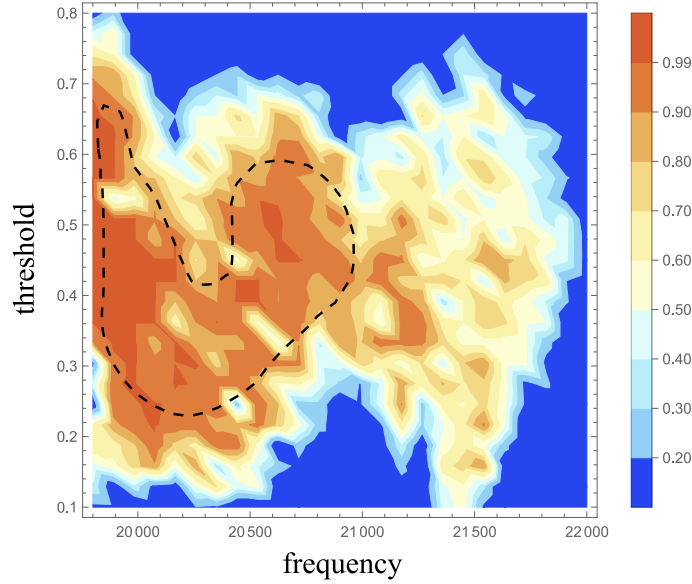

figure 5. Function $g(c_3, c_4, t, f)$

The chart of the function $g(c_3, c_4, t, f)$ shows just one acceptable area, which has been encircled by the dashed line. Unfortunately some young and healthy ears are able to notice frequencies up to 20200 Hz, so the frequency has to be chosen from the right-hand side of the encircled area. One thing to notice is that the transmission becomes increasingly problematic in the frequency range 21 kHz - 22 kHz. That is more than likely caused by the frequency response of both the speakers and the microphone, which noticeably dampen the higher frequencies. The parameters are selected to be:

$$t_0 = 0.45 \qquad f_0 = 20700\text{Hz} \tag{2.7}$$

2.3.3 Determining l_0

By assumption g does not depend on l, nonetheless, l needs to be somehow set. The shorter the length of an impulse is, the faster is the data transmission. Therefore it is imperative to make l as small as possible. The smallest value of l which does not produce any sound seems to be 49 (whilst using $\delta_0, \kappa_0, t_0, f_0$ as transmission parameters). The reason why short impulses become audible is because the shorter they become, the more sudden is the amplitude change; i.e., they cause crackling sound. The parameter is than selected to be:

$$l_0 = 49 \tag{2.8}$$

This impulse length sets the data transmission speed to 900 bits per second, if arranged for the error-correction symbols which have to be transmitted too, the number falls down to 794 bits of data per second.

3. Testing

This chapter discusses the range at which the transmission is effective, and associated security implications. The range is measured differently than in section 2.3, whilst the value of g is not important in this instance, the crucial metric is the probability of a successful transmission with regards to r, the distance between speakers and the microphone.

The chart of the probability $p(r)$ is based on 50 transmissions at the distance r with the parameters $\delta_0, \kappa_0, t_0, f_0, l_0$.
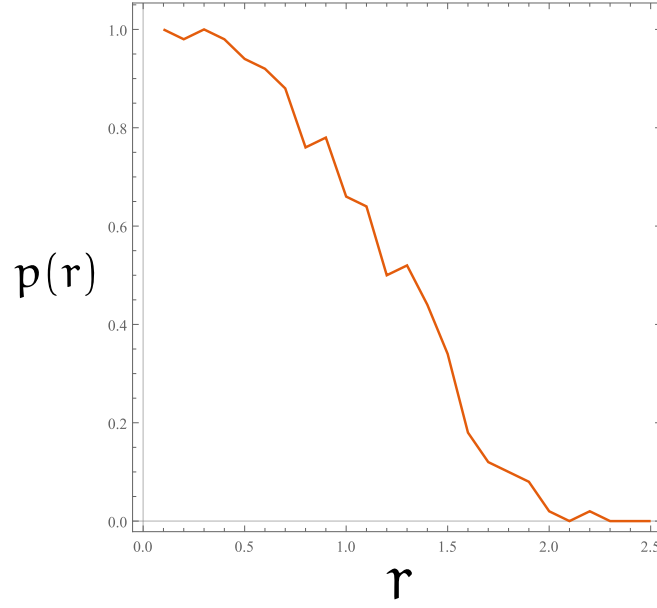

figure 6. Probability of successful transmission

As the figure 6 illustrates, after multiple attempts, the data can be transmitted up to the distance of 2 meters, though reliable transmission reaches just to the distance of 0.4 meters.

As mentioned in the introduction this technology carries with it certain security risks, for example data could be leaked this way from a device that is purposefully disconnected from the internet. An obvious protection would be to use only hardware without the capability to play any audio; meanwhile, another option exists, at least theoretically. Area of interest could be equipped with microphones which are constantly searching for sustained change in the ultra-sonic frequency spectrum. To do this, all that is needed is to modify the Receiver. In Condition (2.1) the constant would be lowered and the rolling average would be taken over larger amount of sequences. Constant equal to 2 with average taken over 800 sequences enables detection of the transmission at the distance s and the angle ψ (figure 7, table 2).

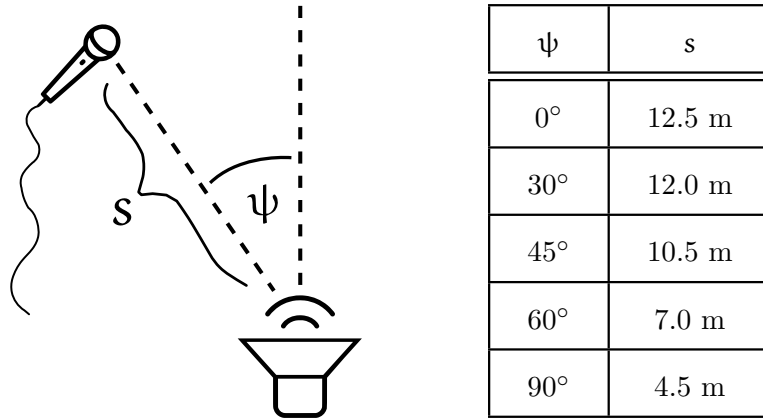| ψ	s
0°	12.5 m
30°	12.0 m
45°	10.5 m
60°	7.0 m
90°	4.5 m

figure 7. & table 2. Detection range

Nonetheless, the problems with detectability can be easily circumvented by putting any object between the speaker and the security microphone, simple slip of paper is capable of blocking the transmission.

During the testing phase it emerged that some animals are able to hear the, for humans inaudible, transmission, namely dogs, cats and horses.

4. Technical Documentation

This paper is accompanied by two Python 3 files [2], one dubbed `Transmitter.py`, and another one dubbed `Receiver.py`. These two programs utilise Python libraries (reedsolo, sounddevice, struct, threading, vlc, wave) which have to be installed beforehand.

`Transmitter.py` has to be run on a device with speakers rated above 20 kHz, likewise, `Receiver.py` has to be run on a device with microphone rated above 20 kHz.

Neither of the devices can use any type of noise suppression software, it would inhibit correct transmission as this software typically filters out nuances such as short impulses. Beware, some computers might use these by default.

Do not use this software in close proximity to animals of any kind as it could make them uncomfortable, and depending on where you live, could also amount to a criminal offence - animal cruelty.

The software is distributed under the MIT License, i.e., it can be used without any restrictions.

Conclusion

The paper has proposed a comprehensive theoretical solution to a problem of data transmission via sound-waves, unfortunately reality seems to meet theory only halfway and does not allow for universal implementation as it demands several parameters to be determined beforehand. Setting these parameters was far from trivial, and if such software were to find a concrete usage, some workaround would have to be found, investigation into this matter could be subject to further research. Aside from these shortcomings, the concept has been proven functional and could be employed at places where other more sophisticated solutions are an overkill or simply are not feasible from the hardware point of view.

Personally I consider the assignment to be fulfilled and to my satisfaction even though the distance at which the transmission is effective has proven underwhelming when compared to initial expectations. Unlike bluetooth or wifi the technology would at this stage require the user to be aware where the speakers and microphones are facing, making it far less convenient than anticipated.

I found the project challenging, whilst in most cases there was no way to test the constituents of the final program individually, thus I was left in the dark until all of the program was written. Beneficial at the least has been the fact that I got to try my hand on finding extremes of multi-variable function, which made me realise how complex can real-world optimization problem become.

References

[1] Spohie Bushwick. "Ultrasonic Attack Device Hacks Phones through Solid Objects". In: *Scientific American* (2020). URL: https://www.scientificamerican.com/article/ultrasonic-attack-device-hacks-phones-through-solid-objects/.

[2] Teodor Machart. "Maturitni_prace". In: *GitHub repository* (2022). URL: https://github.com/MachTe/Maturitni_prace.

[3] Bernard Sklar. "Reed-solomon codes". In: *Downloaded from URL http://www. informit. com/content/images/art. sub.–sklar7. sub.–reed-solomo-n/elementLinks/art. sub.–sklar7. sub.–reed-solomon. pdf* (2001), pp. 1–33.

[4] Julius Smith. *Mathematics of the Discrete Fourier Transform.* Center for Computer Research in Music and Acoustics (CCRMA) Department of Music, Stanford University, 2002, p. 145. URL: https://ccrma.stanford.edu/~jos/mdft/mdft.html.

List of Figures

List of Tables