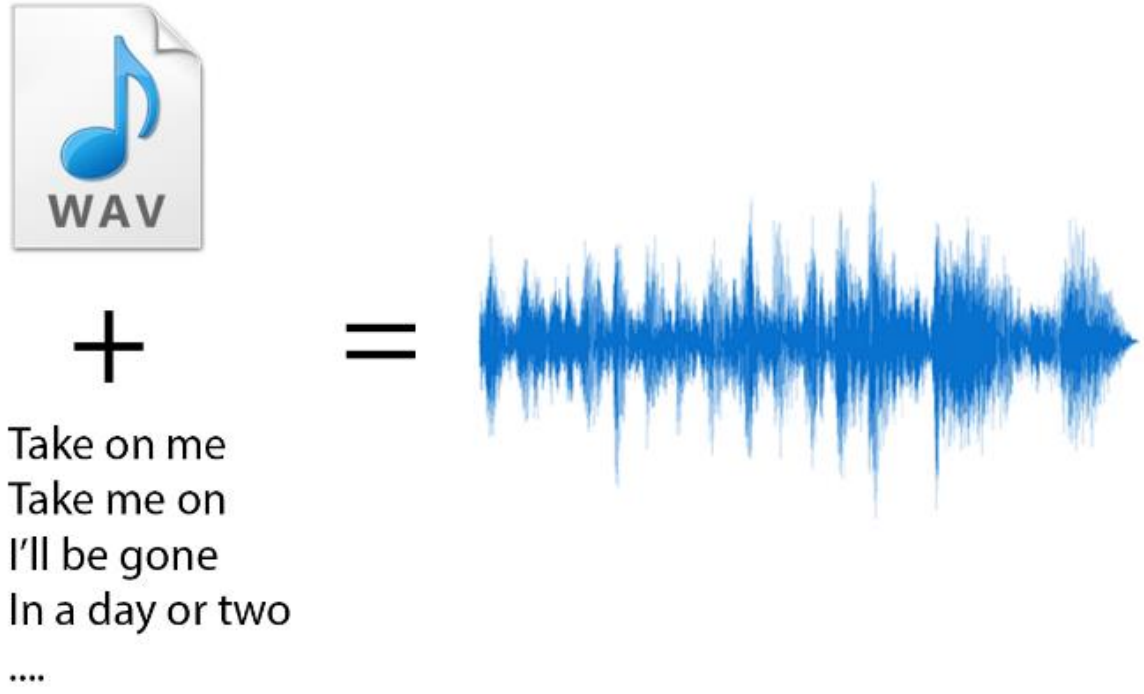
Steganography on Audio



Made by: Jim Vermunt 16417857,
Dirk Brouwers 1631004
Course code: 5LIU0
To: Alexios Balatsoukas Stimming,
Eindhoven University of Technology
Date: 29 January 2021

CONTENTS

1 INTRODUCTION

Steganography comes from the Greek words *steganós* meaning 'covered' and *graphia* meaning 'writing'. Steganography is, therefore, the subject of covering writings, which is commonly innocent-looking for outsiders. Steganography can be used along with cryptography as extra security. This protects the data from being read by an unwanted person in case it is found. This application technique is mainly focused on sending secret messages like a plan to cover operations or to send political espionage information. Historians found hidden messages written in invisible ink or documents that contained messages by reading only the first character of each sentence. In World War II, for example, steganography was useful because anything that looked suspicious or looked like encrypted data from the enemy was immediately destroyed before it could be received by the right person. This made it necessary to apply this technique to the information sent from A to B. This prevented the enemy from the opportunity to destroy the document. Today, the most used method of steganography is to hide files within files on a computer. Steganography is commonly applied to images, videos, and audio files. An example is hiding a text message into a picture where the least significant bit (LSB) of the picture is changed. This way, it is hard to detect the message without the use of a computer. Today there are several steganography tools such as Xia and SSuite Picsel.

Two students at the TU/e, Jim Vermunt and Dirk Brouwers would like to communicate with each other over audio with hidden messages, but this message needed to stay secret. Therefore, they made a software application that uses steganography over audio, that encodes a hidden text message into the audio file.

The goal of this report is to describe how well the application works that hides a text message in an audio file. To describe the progress and results, the following components are discussed in corresponding order: (1) Problem Specification, (2) Data used to evaluate the problem, (3) Evaluation Criteria, (4) The Setup of the Application, (5) The Approach, (6) Results and Analysis, (7) Conclusion and Recommendation.

2 Problem Specification

In this section, a clear and concise description is given of the problem. The general formulations are given into a well-defined technical goal for encoding messages and decoding messages.

2.1 Encoding Message

The application first needs to import an audio file. This file consists of an array of samples, represented in double variables. These values need to be represented into a matrix form with a bit format. The matrix columns are defined by the corresponding bit depth. This formatting needs to be done correctly by the encoder. Also, the decoder needs to use the same structure, otherwise, it is impossible to decode the message.

The sample count is the length of the sample array. This length is calculated by *Equation 1*. Where f_s is the sampling frequency in hertz and t_{audio} the length of the audio in seconds.

$$Sample\ count\ =\ 1/f_s\ *\ t_{audio}$$

Equation 1: Sample count

The last column of the audio matrix has the least significant bit (LSB). The LSB of the audio is used to hide the text message because this is the most precise bit of the audio sample and the only place where the change of data will be the least significant. To change the LSB, some data processing is needed for the audio.

The text message that needs to be put into the audio also needs to be converted into a bit array. This text bit array needs to be put over the inputted sound bit array. After this overlapping, it is possible for the audio to be corrupted or to be transformed into unrecognizable sounds. That is the art to prevent this from happening. A graphical representation of adding the text message over the audio is given in *Figure 1*.

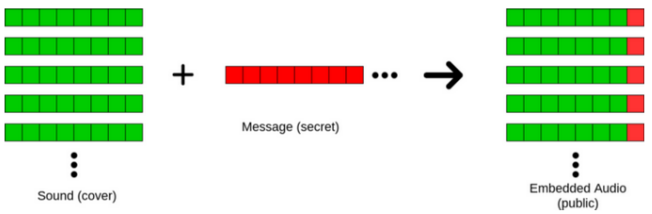


Figure 1: Scheme for encoding audio

2.2 Decoding Messages

The application needs to process the audio file to extract the hidden message. First, the encoder needs to know what the length is of the text bit array, so it decodes only the message. The decoder needs to check every LSB that is changed by the encoder. These bits are gathered into a bit array that represents the text. This bit array needs to be converted to the actual text and is then printed on the console. *Figure 2* gives a schematic overview of how the message is extracted by the decoder.

Figure 2: Scheme for decoding message

3 Data

In this section, it is explained what the real-world and synthetic signals are that are used to develop and evaluate the application. The following components are evaluated in the following sections about the absolute threshold of hearing, the sampling frequency of audio, bit depth, input audio, and input text.

3.1 Absolute Threshold of Hearing

To define non-hearable audio degradation, it is necessary to define what the minimum human hearing difference is. There is a wide variety of variables that can impact the audio threshold value of a human's hearing. This report assumes the values with the defined parameters in *Table 1.* Note that when the situation differs the threshold also variates.

Table 1: Situation description

Frequency	$1000\ Hz$
Sound intensity	$0.98\ \frac{pW}{m^2}$
Pressure	$1\ atm$
Temperature	$25\ °C$
Person description	A young human being, without hearing loss

When the situation is following the table, the threshold of hearing is 20 micro Pascals. This is also the defined reference pressure in the report, so this is 0dB [1].

3.2 Sampling frequency of audio

The standard sample rate of audio is 44.1 kHz. This frequency is standard for most common audio systems. This sample rate is used to convert continuous-time signals into digital audio. The frequency of 44.1 kHz is derived from the Nyquist theorem, the spectrum of human hearing, and some added free space for low pass filtering. Assuming the human ears can hear between 20 Hz and 20kHz, by the Nyquist ratio it comes down to 40 kHz. The leftover 4.1 kHz is for the low pass filter [3] because a filter is never ideal. A filter cuts the power directly not to zero, therefore the filter needs some space to not reduce the power of the wanted frequencies or letting too much noise power go through the filter.

3.3 Bit depth of the Audio

To define the bit depth for the quantization of the input audio, the most common bit depth uses the value of 16. By default, MATLAB functions are set on 16 bits, also CDs use the 16-bit depth. For this project, the 16-bit depth is used also chosen for portability purposes [7].

3.4 Audio input format

The audio format is in the form of a waveform audio file (WAV). This format is standardized by IBM and Microsoft for storing audio in bitstreams on PCs. This audio file has a minimum length and a maximum length, this depends on the SNR and hearable audio threshold. The audio needs to have specific properties, where the sample rate needs to be at least 44.1 kHz. The audio needs to have a minimum duration of $8.8 \cdot 10^{-4}$ seconds. Also, the bit depth needs to have a minimum of 16 bits per sample. These files can be generated by an online tool where an MP3 or MP4 file is converted to a wav file.

3.5 Text Input

The text is inputted by the keyboard on the pc that runs the MATLAB code, where the user can define the text file path. This text length will have a maximum of characters and will depend on the criteria above. After inputting these components, steganography is applied to the audio file that will be added to the working folder. When the user inputs it incorrectly, there will be a notification why steganography cannot be applied to the audio file. A text message that is too long in ratio with the audio file length, is a good example of an invalid input.

4 Evaluation Criteria

In this section, the evaluation criteria defined how well the application performs. These criteria are the signal to noise ratio and the characters per second.

4.1 Signal to Noise ratio (SNR)

There are two options to hide the text while staying undetected. The first option is to add information under the threshold of the hearable sound. Which is under the 0 dB defined by the situation in *Table 1***.** The other way to hide the noise is by masking it with a louder signal to suppress the attention. To create these situations, the correct SNR needs to be defined.

When adding a hidden message to the audio, noise is created on the input audio. This noise cannot be too high, otherwise, the audio is recognizably changed. To prevent this from happening the audio loudness must be significantly higher than the noise. This is expressed with the SNR. In *Equation 2* the formula is expressed where *P* is expressed as average power in watts.

$$SNR = \frac{P_{signal}}{P_{noise}}$$

Equation 2: SNR formula

Because the range of power in signals varies a lot from scale, the sound is often expressed as a logarithmic scale. These calculations are shown in *Equation 3.* $P_{signal,dB}$ is the power of the signal expressed in dB.

$$P_{Signal,dB} = 10log_{10}(P_{signal})$$

$$P_{noise,dB} = 10log_{10}(P_{noise})$$

Equation 3: Signals and Noise expressed in dB

The following step is to define at which ratio the audio signal masks the noise. For reference, the Microphone module AN12590 from NXP in 16-bit mode, records audio with an SNR of 58.2 *dB* [6]. This microphone

module is a microelectromechanical system (MEMS) used for consumer electronics. The SNR of the AN12590 is used as the ratio that produces an audio signal without recognizable noise detection of human hearing. The SNR of 58.2 dB is also the evaluated criteria of the steganographic application.

4.2 Character per Second

When everything works as it should, the expected outcome would be that the steganographic algorithm can encode 100 characters in a one-minute wave audio file. This equates to $1\ ^{2}/_{3}$ characters per second. It affects the result when more or fewer characters per second are processed in an audio file. When there are more chars processed, it will be more noticeable for the listener in the form of noise. The opposite is also true, when there are processed fewer characters per second, it will be less noticeable for the human ear. This will determine how good the algorithm 'encoding' works.

5 Setup

In this section, it is explained what the hardware and/or software setup is. Also, the kind of signals that are being used is clarified.

This project consists of one software component and one hardware component. The hardware component will consist of a PC that can run the program MATLAB, and the software component is the program itself. The version of MATLAB is R2020b, which is the most recent MATLAB version available. There could occur problems running the application on other versions, due to changes in function calls of every version.

Within MATLAB a toolbox package is available, called Digital Signal Processing (DSP) which will be used for the help of analysing signals from the audio file. The system toolbox includes signal transforms such as fast Fourier transform (FFT).

6 Approach

This section describes how the problem is solved. First encoding the text message into the audio is discussed. After that, the decoding of the message out of the audio is reasoned.

6.1 Encoding Message into Audio

To encode the message into audio, several steps are needed, these are the steps in corresponding order: (1) obtaining audio file, (2) obtaining text file, (3) splitting audio into correct samples, (4) converting the hidden message to binary, (5) converting samples to binary format, (6) encode the message into audio, and (7) write encoded audio to directory. An overview of this process is given in *Figure 3.*

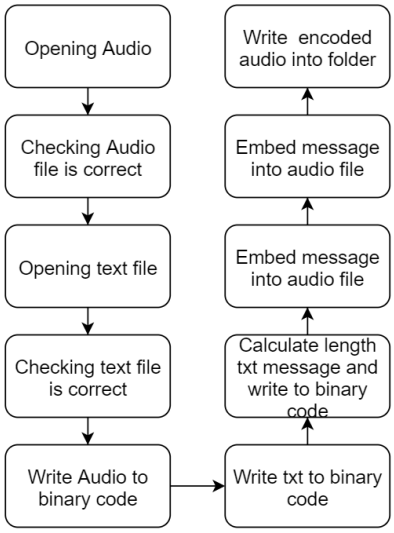


Figure 3: Function block diagram encoding message into Audio

Inputting Audio into the application

In order to import the audio, the function 'audioread('filename')' is used where the filename is the audio file path selected by the user. With the use of MATLAB function 'audioinfo()' the inputted audio file will be checked on correct specifications. With an interface shown beforehand, the user will be informed about the recommended specifications for selecting the audio file. The user gets the corresponding warnings and errors in the cases given in *Table 2* for the correct inputting audio file.

Table 2: Warning table input audio

Case description	Type of notification
Sampling rate is under 44100	Warning: Sampling Rate to Low there could occur noise
Duration is under 9E-4 seconds	Warning: Duration of audio to short in order to put 1 message in the audio file

Inputting .txt into the application

The function 'fileread('filename')' imports the text message that needs to be hidden to the audio file. The text file is checked on its length of bits. When the length of bits is longer than the samples, the user gets an error message, that the text file is too long. This could be resolved by writing data in the lesser significant bits. This approach is mainly focused on programming and not on digital signal processing, therefore this functionality is not added in this phase of the design process, but the influence is tested.

Converting Hidden Message to Binary Format

To make sure the audio is not big enough to hold the message, and the audio sounds as normal as possible, the length of both inputs are checked. There are 7 bits per ASCII letter/character. This means there is a need for a function that converts an ASCII value in the range 0-225, to binary digits. This is done with the function 'de2bi()', where the values are converted to a binary matrix. The function stands for decimal to binary. This results in a matrix where every row represents a character.

Converting Samples into Binary Format

When the audio is imported the audio needs to be converted into a binary format. Every sample of the audio is represented with a value ranging from 0 to 65536. This value is converted to a binary representation with 16 bits. This is done with the function 'dec2bin()'. Only the audio file needs some processing before it can be converted. When there is an audio inputted with a stereo sound, the audio has two different values per sample. This is solved with the function 'single(audiofile)'. The audio is by default represented in the double format that represents values from -1 to 1. Doubles are not suited for conversion from decimal to binary, so the function 'typecast()' converts the doubles into int16 values.

Encode Message Length into Audio

To notify the decoder of the length of the hidden message, it is included in the encoded audio. The first 32 least significant bits of the encoded audio represents the length of the hidden text. First, the length of the hidden message is converted to binary format, in the same way the text is converted. With a for loop, the least significant bit of the audio file is changed, to the corresponding value of the text length in binary format. The reason for implementing the length of the message is reducing execution time and prevention of reading bits that are not the embedded text message.

Encode Message into Audio

To add the message into the audio, the binary format of the message loops through every cell. The corresponding value that is located in the message is added to the least significant audio bit. Figure 4 shows the loop of how the message is added to the audio.

```
%Adding message to audio with LSB
k = 33;
for i = 1:rows
    for j = 1:columns
        put_lsb = InMeB(i,j);

         EnAuB(k,16) = put_lsb;
         k = k+1;
    end
end
```

Figure 4: Code that adds the message to audio with LSB

After the data manipulation, the data frame is as shown in *Figure 5* of the encoded audio.

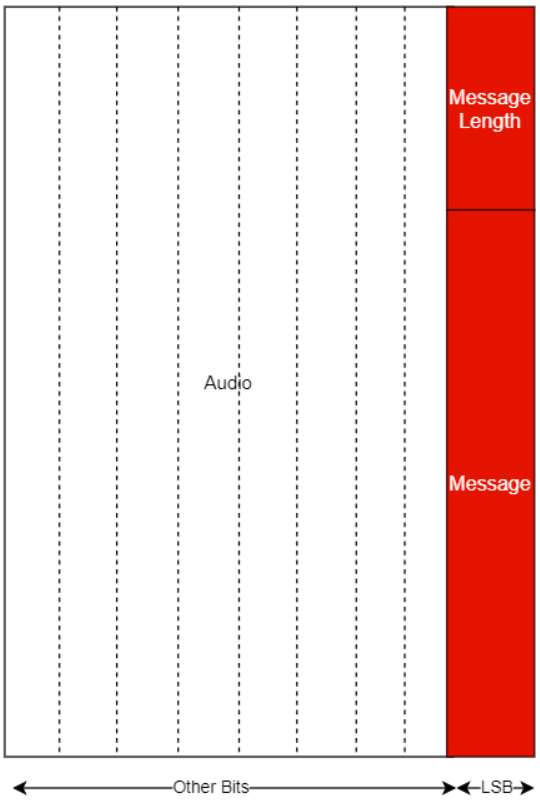


Figure 5: Data frame of encoded audio

Write encoded audio to directory

To obtain the result of the encoded audio, the file is saved with the function 'audiowrite('directory', 'encoded_audio')'.

6.2 Decode Message from Audio

To decode the message, several steps are needed and done in the following order: (1) Convert imported audio into binary format, (2) Getting the length of hidden message, (3) Decode message from encoded audio, and (4) Write a decoded text file in director. *Figure 6* gives an overview of the sequence.

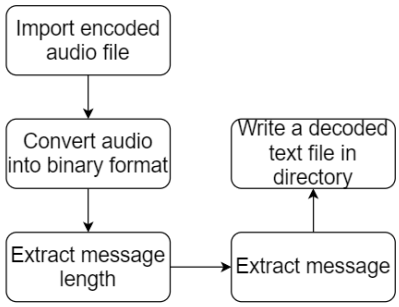


Figure 6: Function block diagram decoder

Convert imported audio into binary format

To decode the text again, the encoded audio file is needed. This is done with the function 'audioread('filename')'. After this is done, the audio needs to be converted to a binary format. This conversion is done in the same way as described in *Converting Samples into Binary Format.*

Getting the length of the hidden message

The decoder needs to know the length of the message that is hidden in the least significant bit part of the audio file. That is why the first 32 bits will return the length of the hidden message. This message length is extracted by a for loop that checks these bits so that the length of the message can be determined.

Decode message from encoded audio

Now the length of the message is known, the binary bits of the characters can be extracted from the encoded audio. The starting point is at the 33rd row at the least significant bit (LSB). The bits from the LSB are extracted until the length of the message is surpassed. This results in a column vector and needs some pre-processing. The vector is first converted to a matrix with 7 columns. After this conversion, the matrix is converted into a char vector. This char vector is then transformed into an ASCII row.

Write a decoded text file in directory

With the 'fopen()' function, it is possible to create or rewrite a new file, which will include the decoded message in string form. After writing and closing the text file, the decoder is finished.

7 Results and Analysis

In this section, the results and analysis of the application are given, for the effects of changing the bits and characters per second ratio. To check if the application meets the evaluation criteria.

7.1 Bit Depth Effects

In *Paragraph 4* the importance of SNR is discussed. To calculate the SNR of the application, *Equation* 4 is used, where S_i is the input audio and S_{en} the encoded audio. N represents the total number of samples.

$$SNR = 10log_{10}(\frac{\sum_{n=1}^{N}|S_i|^2}{\sum_{n=1}^{N}|S_{en} - S_i|^2})$$

Equation 4: SNR formula used in MATLAB

The application that is made is tested. These tests are run with the audio fragment 'Take on Me' with the following properties: sample rate 44.1 kHz, duration 5.878 seconds, and bit depth of 16 bits. As a hidden message, the lyrics of 'Take on Me' is chosen and repeated multiple times with a total length of 72265 characters, and with a bit format of 505855 bits. The results when calculating the SNR are given in *Table* 3. This table also includes the results of applying steganography using different bit depth levels. The SNR of a given bit depth is calculated with *Equation 4.* Also, the audio quality is determined through listening to the encoded audio.

Bit depth:	SNR (dB):	Audio quality
1	0.2238	- - - -
2	-∞	- - -
3	6.3046	- - -
4	6.1030	- - -
5	6.3046	- - -
6	-8.0760	- - - - (volume amplification)
7	-14.5057	- - - - - (volume amplification)
8	-0.6297	- - -
9	8.0069	- -
10	14.8769	-
11	20.9087	-
12	26.9591	+
13	32.9487	+ +
14	39.0341	+ + +
15	45.0270	+ + +
16/LSB	51.03650	+ + +
31% of LSB	58.2	+ + +

Table 3: SNR comparison with different bit depths

The results of *Table 3* show that when only using the LSB, the SNR is still less than the given value of the criteria. The value given in the criteria is an SNR of 58.2 dB. To satisfy the evaluation criteria, less text needs to be inputted. When reducing the characters of the text file until the SNR of 58.2 dB is reached, the text file needs to reduce its size to 31% of the original size.

To validate that the application is user-friendly, the encoding time is calculated. The time it took to embed 72,265 characters into the audio was 2.7 seconds, with a laptop with some decent specifications. This report consists of about 28,000 characters, to give an impression of how much 72,265 characters it is. The time it took was very short, so the user does not have to wait long if the user gives the application a reasonable amount of characters.

Another way to analyse the effects of changing the LSB is with a periodic analysis. The representation of the spectral density of the input audio compared to the encoded audio is given in *Figure 7*. It shows a scaled display of the density of the audio in equivalence to the signal power at each frequency measured with a filter exactly 1 *Hz* width.

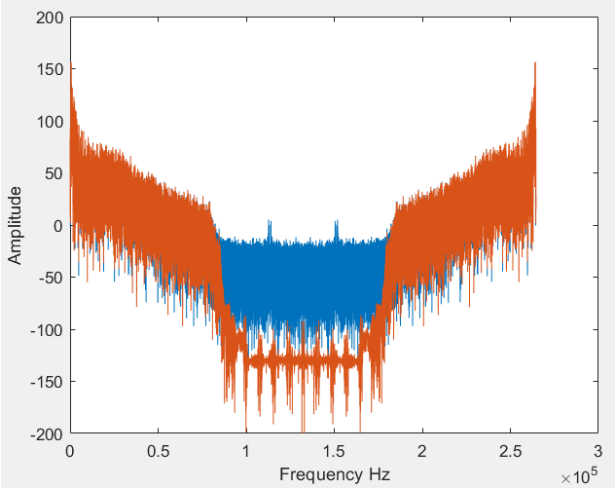


Figure 7: LSB difference of input audio (blue) and encoded audio (orange) with a power spectral density representation.

An absolute periodic representation of the input and encoded audio is given in *Figure 8* by a fast Fourier transform (FFT). It shows that the difference in audio is minimal and can only be shown when zoomed in.

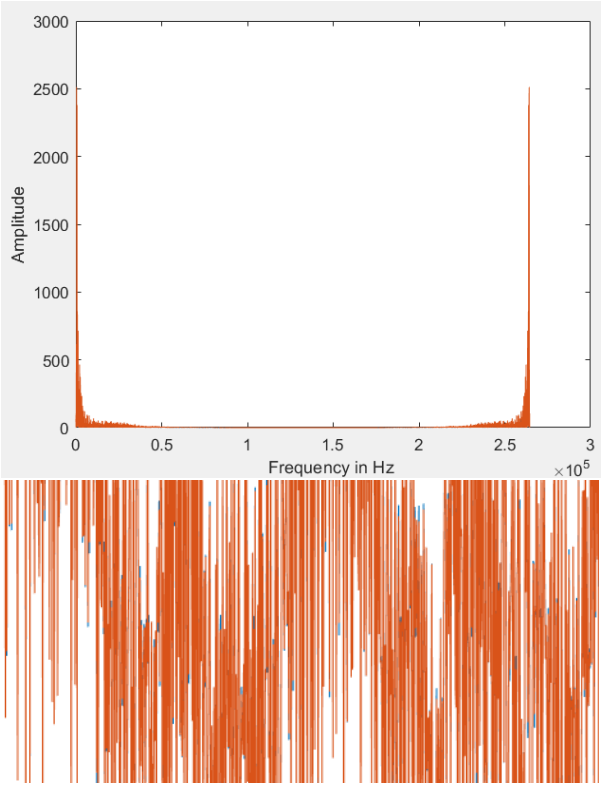


Figure 8: Periodic representation 'input audio' (orange) and 'difference encoded audio' (blue) - Normal representation (above) - Zoomed representation (below)

After the 7th bit depth, the SNR becomes an unreliable measurement technique to define the amount of noise in the audio, this is shown in Table 3. Due to unknown reasons, the SNR increases again, which indicates that it is misleading to use. This only for specific bit depth

conditions lower than the 7th-bit depth. The hypothesis of increasing the significance of the bits, was that the SNR would decrease. In contrast, the SNR increases when going from the 7th-bit depth to the 3rd. The SNR at the second bit jumps to infinite. When transitioning to the first bit the SNR increases again.

A better representation of what is happening with the audio signal can be given by other figures in the appendices and by listening to the audio by ear. In *Appendix A & Appendix B* periodic representations of other bit depth values are given. In *Appendix A* the absolute logarithmic representation with a fast Fourier transform is shown. In *Appendix B* the power spectral representation is given. In these figures, it can be clearly shown what the effects are of changing the bit depth of the audio.

7.2 Character per Second

In the evaluation criteria, it is stated that the steganographic algorithm can encode 100 characters in a one-minute wave, which is equated to $1\,{}^{2}/_{3}$ characters per second.

The result shows that it is possible to have a sampling frequency of 44,100 Hz. This means 44,100 samples per second where it is possible to change the least significant bit (LSB) in binary form. Because a character of the hidden message is of 7 bits: $\frac{44100\ (samples\ per\ sec)}{7\ (bits)} = 6300\ char/seconds$. 6,300 char per seconds suffices only when the full range of the LSB is used. However, when using this full range, the application does not suffice the evaluation criteria of the SNR, so the characters per second need to be reduced to 1,982 characters per second. When using this rate, the SNR is 58.2 dB and uses 31% of the full range shown in *Table 3.* This rate is still more than the expected $1\ {}^{2}/_{3}$ characters per second from the Evaluation Criteria chapter.

8 Conclusion & Recommendation

The report shows how the application is approached from code and how well it performs. The application that is made consists of two components: the encoder and the decoder.

The encoder imports the audio and text file and convert it to binary data matrixes. It writes the bits of the text to the least significant bits (LSB) of the audio file. Also, at the beginning of the audio file, the length of the message is embedded in the audio, for performance and correct output. After encoding, the encoded message is exported, so it can be shared without people knowing that the audio is changed.

The decoder imports the encoded audio file and writes it to a binary matrix so the LSB can be extracted from the hidden messages. First, the length of the audio is extracted, then the decoder reads until it has read the whole length of the audio of the file. After the message is obtained, the extracted text can be exported.

When using the least significant bit (LSB), the application performs well in hearable audio quality. Therefore, the application still operates over the full range of the LSBs, to increase the characters per second rate. Because the audio quality difference is still inaudibly changed using the LSB. It can be concluded that the evaluation criteria of the SNR of 58.2 dB are chosen too strict. Using an SNR of 39 dB would also satisfy when assumed by only listening. When using the full range of the LSB the characters per second rate is 6,300.

Calculating the SNR below the bit depth of 7 gives unexpected results. For example, at the second-bit depth and the transition from the 7th to the 6th bit. These effects remain unknown and could be addressed furthermore in the future.

REFERENCES

[1] Sangita, "Audio Steganography Using LSB Encoding Technique with Increased Capacity and Bit Error Rate Optimization," Bihar.

[2] S. A. Gelfand, "Hearing: An introduction to psychological and physiological acoustics," 1990.

[3] S. M. W. Dale Purves, "Neuroscience 2nd Revised Edition," 2001.

[4] C. Mongomery, "24/192 Music Downloads ... and why they make no sense," 2021. [Online]. Available: https://web.archive.org/web/20130707161555/http://xiph.org/~xiphmont/demo/neil-young.html .

[5] Dewesoft, "Sound Power Level Measurement on a Notebook Computer," Dewesoft, 2019. [Online]. Available: https://dewesoft.com/application-notes/sound-power-level-measurement-on-a-notebook-computer.

[6] N. Semiconductors, "AN12590 i.MXRT600 PDM MEMS Microphone Audio Path Optimal Settings," November 2019. [Online]. Available: https://www.nxp.com/docs/en/application-note/AN12590.pdf.

[7] Mathworks, "sound," Mathworks, 2020. [Online]. Available: https://www.mathworks.com/help/matlab/ref/sound.html. [Accessed December 2020].

APPENDIX A. ABSOLUTE LOGARITHMIC REPRESENTATION WITH FFT

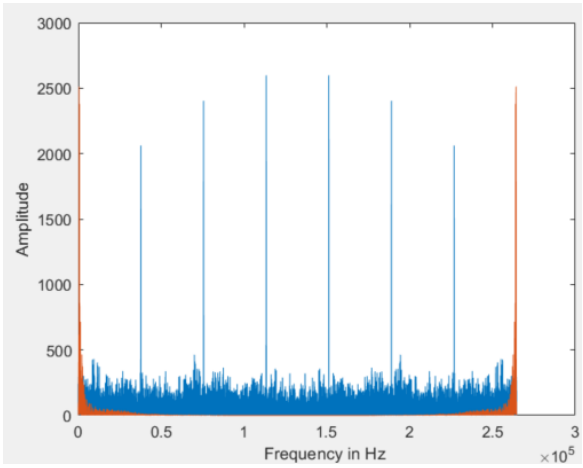


Figure 9: Bit depth 1- Input audio (orange) and encoded audio (blue) as Absolute Logarithmic Representation with FFT

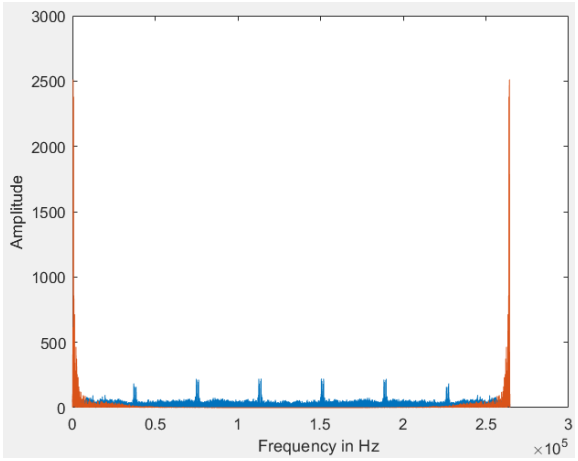


Figure 10: Bit depth 4 - Input audio (orange) and encoded audio (blue) as Absolute Logarithmic Representation with FFT

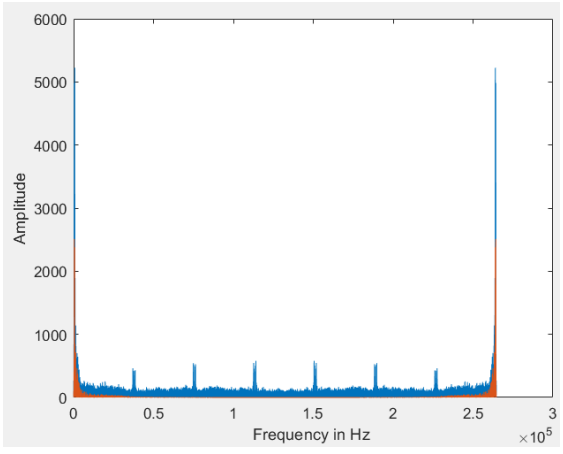
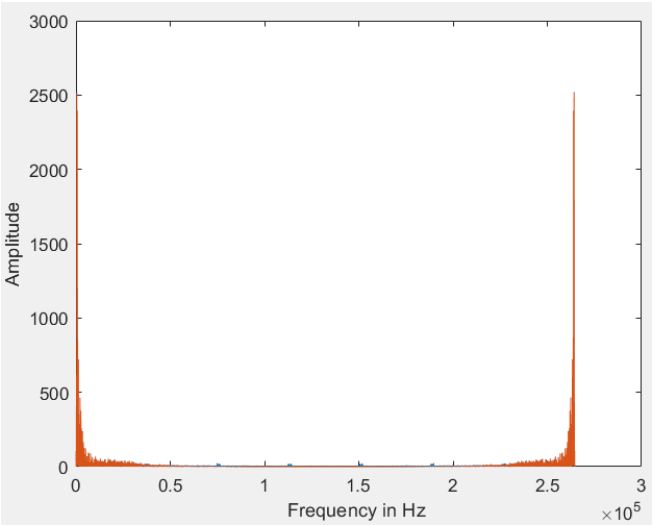


Figure 11: Bit depth 8 - Input audio (orange) and encoded audio (blue) as Absolute Logarithmic Representation with FFT



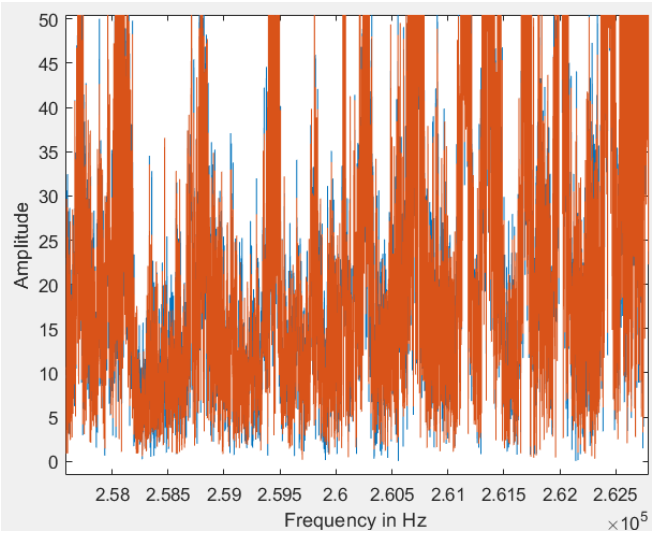
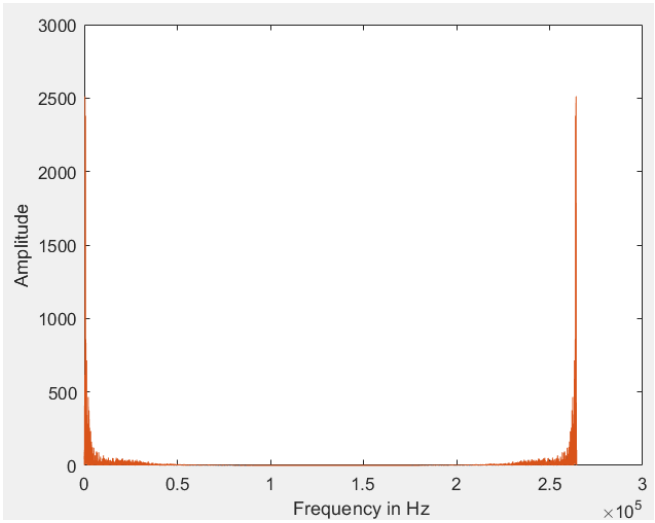


Figure 12: Bit depth 12 - Input audio (orange) and encoded audio (blue) as Absolute Logarithmic Representation with FFT - Normal representation (above) - Zoomed representation (below)



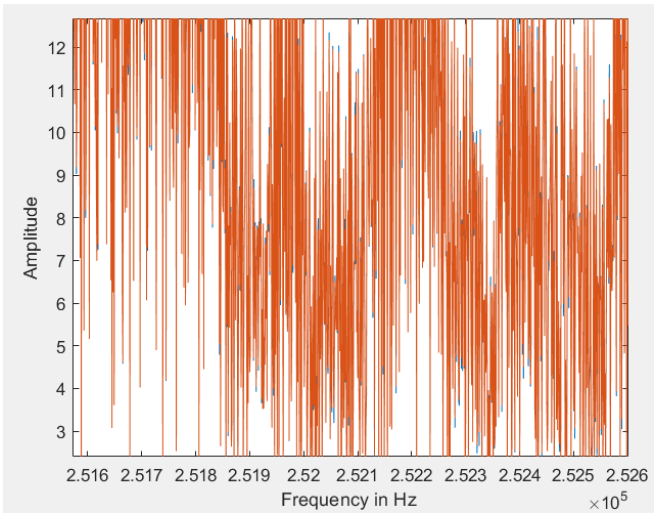


Figure 13: LSB/ Bit depth 16 - Input audio (orange) and encoded audio (blue) as Absolute Logarithmic Representation with FFT - Normal representation (above) - Zoomed representation (below)

APPENDIX B. ABSOLUTE FFT REPRESENTATION OF THE FFT

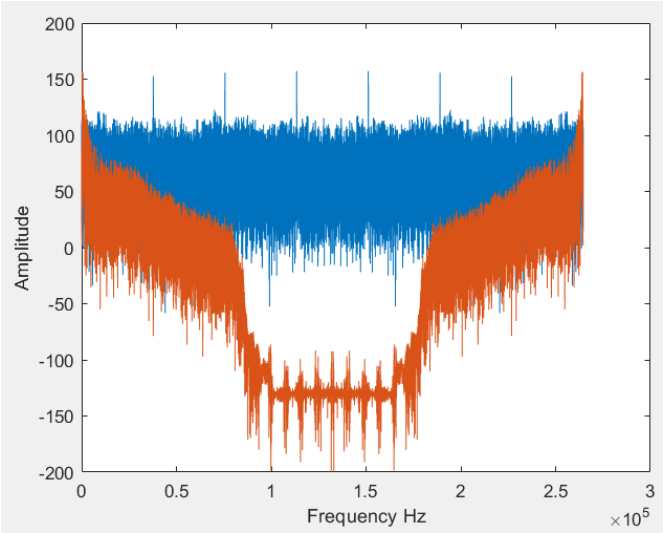


Figure 14: Bit depth 1 - Difference of input audio (blue) and encoded audio (orange) with a power spectral density representation.

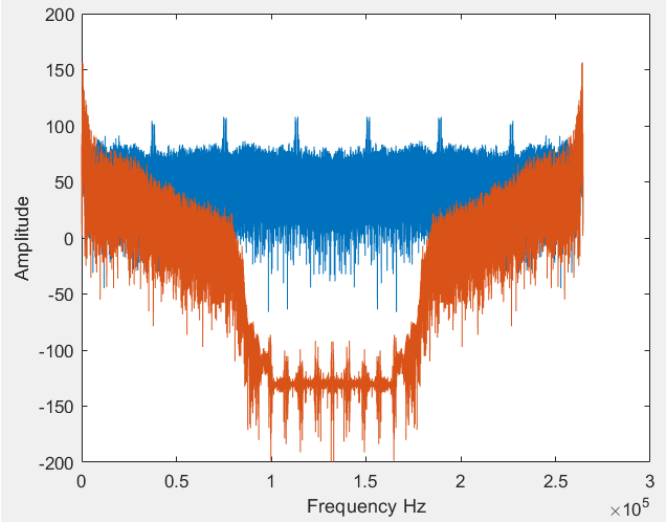


Figure 15: Bit depth 4 - Difference of input audio (blue) and encoded audio (orange) with a power spectral density representation.

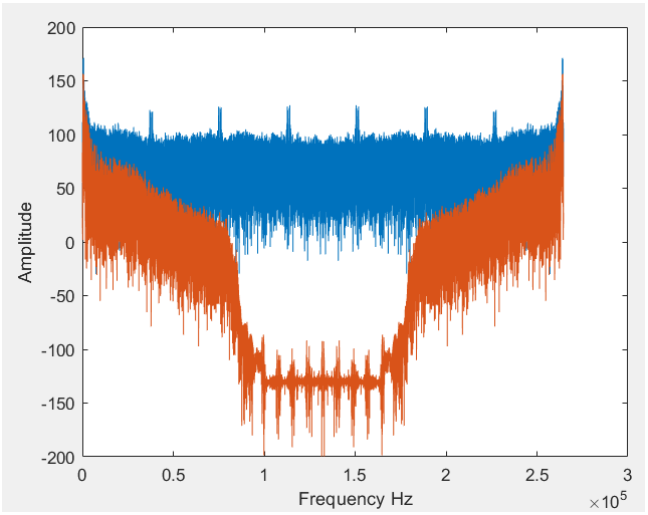


Figure 16: Bit depth 8 - Difference of input audio (blue) and encoded audio (orange) with a power spectral density representation.

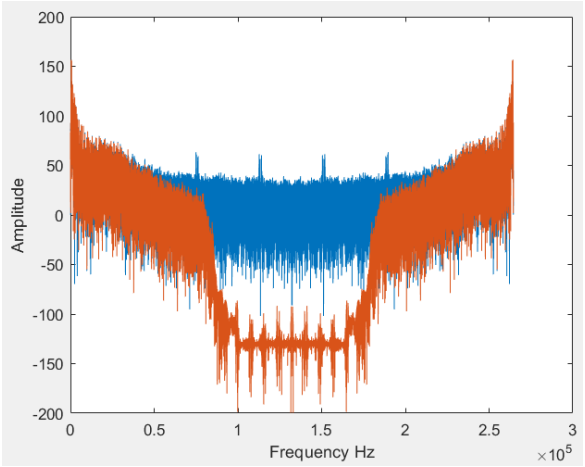


Figure 17: Bit depth 12 - Difference of input audio (blue) and encoded audio (orange) with a power spectral density representation.

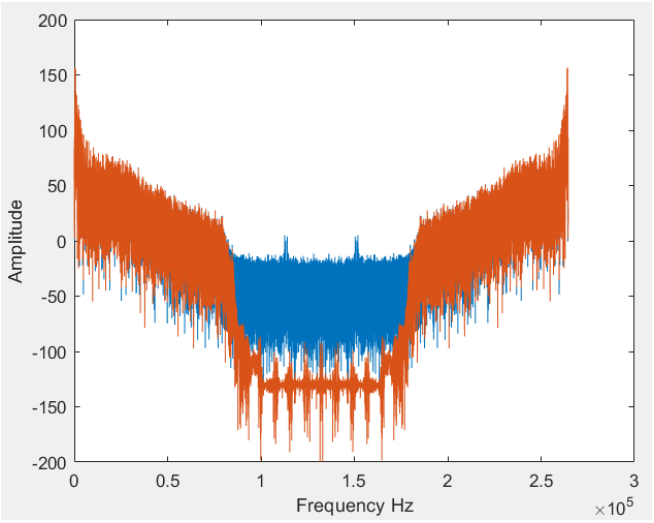


Figure 18: LSB/ Bit depth 16 - Difference of input audio (blue) and encoded audio (orange) with a power spectral density representation.