Strings Decoding and Encoding

Lojain Abdalrazaq

1190707

Birzeit University

1190707@student.birzeit.edu

Aseel Deek

1190587

Birzeit University

1190587@student.birzeit.edu

Mariam Taweel

1192099

Birzeit University

1192099@student.birzeit.edu

***Abstract*— this project aim is to demonstrate the concept of encoding a sent data (string) by implementing an encoder that gives four different frequencies for each character in the data and how to retrieve the string after encoding it by building a decoder using Fourier Transform and Bandpass Filters to analyze the frequency for each character and retrieve the sent data, using MATLAB programming platform** (*Abstract*).

I. Introduction

In general, encoding and decoding operations have a vital use in communication fields, computing, and data programming. Encoding means the creation of the messages that we want to communicate with another side. On the other hand, the decoding means the listener or the audience of encoded messages, so decoding leads to interpreting the meaning of the message (text messages, audience...). There are many ways to encode and decode the messages, one of the most common ways is using ASCII code, which means that each character will be represented by 8 bits, but this way has many negatives and problems such as changes in the message format because of faces noises.

So, in this project, we are going to use another method in encoding and decoding English characters (upper or lower) that have more immunity to noises which leads to keeping the format and increasing the percentage of the accuracy of the encoded messages. The method that will be used in this project depends on giving each English character three voice-band frequency components in the band 100-4000 Hz (low, middle, and high) and the case of (upper and lower) will be represented by the fourth voice-band frequency. In other words, the encoding operation will be done by representing each character with its corresponding four frequencies and generating its waveform segment. While, in the decoding operation, it will be done using the Fourier transform to analyze the frequencies for each character using the highest four peaks for each signal (character). And using Bandpass filters that find the power for each character after it passes throw the filter and according to the power value, the data will be returned.

II. Problem Specification

The problem that we are trying to solve in this project is encoding and decoding alphabetic English characters. This can be done by representing each character with 4 voice-band frequencies. In the encoding part, the input string will be generated to 40ms waveform segment for each character in the input string and then concatenating all segments together to be sent by the network. On the contrary, the decoding part will read the input waveform file and divide it into 40ms short segments and by using Fourier transform method or Bandpass filters method, each segment will be decoded to its corresponding character.

III. Data

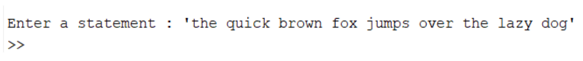
The project was designed to recognize the 26 English letters (upper-case and lower-case format) plus the space. This project does not take the digital numbers and the non-alphabet characters like (- / .*%, etc.).

IV. Evaluation Criteria

To test the program if it is working right or not and finding the accuracy of the expected output, various strings were used. For example, the sentence *" **The quick brown fox jumps over the lazy dog**"* and *" **THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG**"* were used to finds if all the 26 letters can be recovered from the decoder or not, since this sentence contains all the 26 English letter. The next two examples the statements have overlapping upper and lower characters with random spaces ***" BirZeIT UniVeRsi Ty"*** *and* ***"DIGital SiGnal ProCessiNg".***

V. Approach

The first part of this project required creating an encoder that gives each character in the sting four voice-band frequencies. To implement this in code, each signal had four cosines with different frequencies, each character duration is 40ms and a sampling frequency is 8KHz. The first cosine with frequency equals **100 / 200** to differ between the lower letter / upper letter, next is else a **(400, 600, 1000)** to represents a low-frequency component, **(800, 1200, 2000)** for the Middle-frequency component, and finally (**1600, 2400, 4000)** to present high frequency. Each character with it is specified frequencies were stored in an external file called **'*chars.txt*'**, this file was read and split into five columns, the first columns represent the character, the second represents the capital/small, the third represents the low frequencies, the fourth represents the middle frequencies and the last one represents the high frequencies. The ideology was, to compare each character in the entered string with the first column in the file, if there is a match take from the four other columns the value of that index then stores it in a string to concatenate it with the other characters and then convert the entered string to a wave file using audio write function in MATLAB.[1]



```
Enter a statement : 'the quick brown fox jumps over the lazy dog'
>>
```

Figure 1: *the entered string*

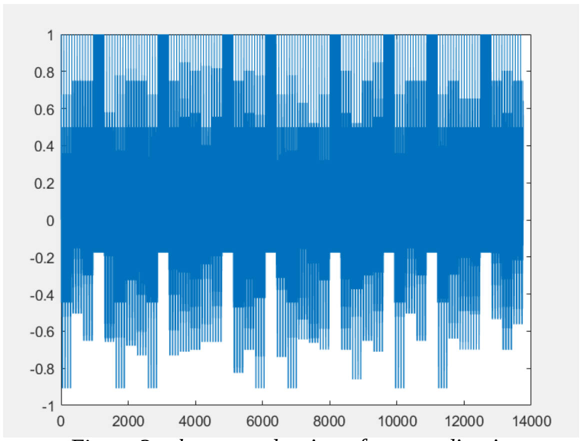


Figure 2: *the entered string after encoding it.*

In the second part which is decoding, it was required to design and implement a decoder that takes an input waveform from the user and recovers the text string from the encoded multi-frequency signal stored in the ***.wav file***. This operation has done by using first, the frequency analysis ***(Fourier transform method)*** that finds the highest four amplitudes for each 40ms segment, then the locations for these amplitudes were found by using the ***find peaks*** function. And finally, the value of each 4 voice-band frequencies (upper or lower frequency, low-frequency component, middle frequency component, and highest frequency component) was found. To display the English text stored in the waveform, the four voice-band frequencies were compared with the values stored in the "***char.tx****t*" in every iteration until the text ended, and when the sequence of frequencies matches the sequence of frequencies stored "***char.txt***", the character will be stored in a defined string until the whole signal finished to be printed later in the screen.[2]

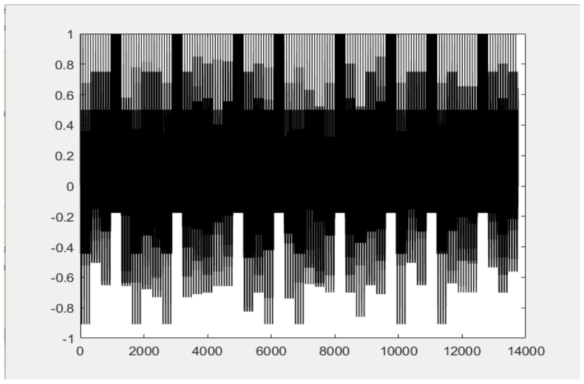
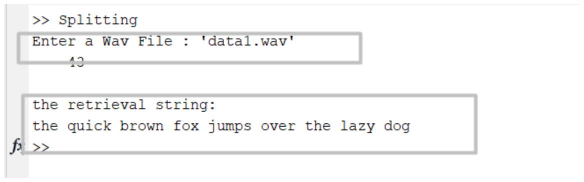


Figure 3: *the encoded data in FTT decoder*



```
>> Splitting
Enter a Wav File : 'data1.wav'
    43

the retrieval string:
the quick brown fox jumps over the lazy dog
>>
```

Figure 4: *the retrieval string using FTT*

Second method is by using the Band pass filters. The decoding is done by creating the filter for 11 different frequencies. To design a filter we need two things, the first is **butter(),** with which we can pass the low and high cutoff of each frequency to create the bandpass, and the second is **filter().**After creating the filter, we calculated the power for each sample through rms. If the value of the power is very small and close to zero, this means that there is no frequency for this sample for the "letter" at the filter, but in case the power is greater than 0.0061, this letter has a frequency. Finally, to display the English text stored in the waveform, the four voice-band frequencies were compared with the values stored in the "char.txt" file in each iteration until the text ended, and when the sequence of frequencies matches the sequence of frequencies stored in "char.txt," the character was stored in a defined string until the entire signal finished to be printed later in the screen.

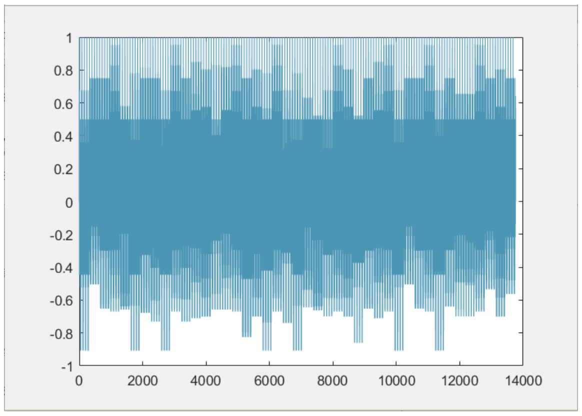
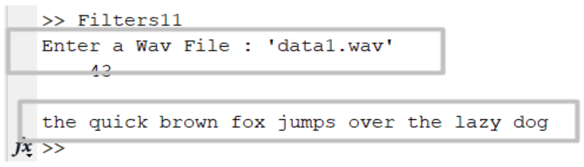


Figure 5: *the encoded data in the filter decoder*

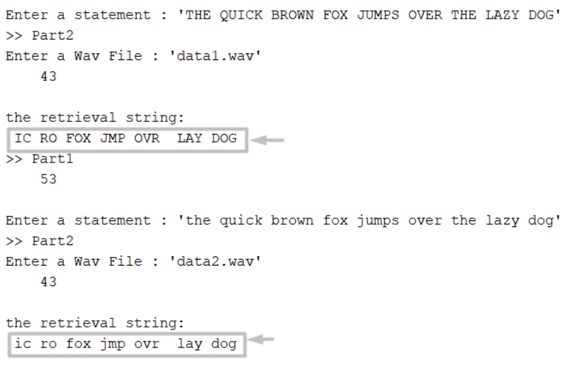


```
>> Filters11
Enter a Wav File : 'data1.wav'
    43

the quick brown fox jumps over the lazy dog
>>
```

Figure 6: *the retrieval string using Bandpass Filters*

VI. Results and Analysis

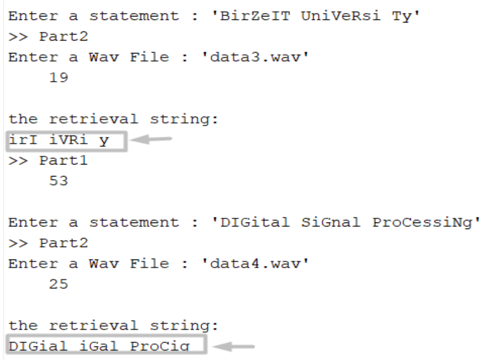
To test the program, and to evaluate the percentage of the accuracy of the retrieved strings, many sentences with different formats were used to check the program with both parts encoding and decoding working correctly or not. In the first decoding part which is using frequency analysis (***Fourier Transform***), the first two sentences that have been tested contained all upper English alphabetic characters ***"THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG"*** and ***"the quick brown fox jumps over the lazy dog"***, the percentage of the accuracy was 62.7% for both sentences.



```
Enter a statement : 'THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG'
>> Part2
Enter a Wav File : 'data1.wav'
    43

the retrieval string:
IC RO FOX JMP OVR  LAY DOG
>> Part1
    53

Enter a statement : 'the quick brown fox jumps over the lazy dog'
>> Part2
Enter a Wav File : 'data2.wav'
    43

the retrieval string:
ic ro fox jmp ovr  lay dog
```

Figure 7: *the decoded first two sentences using FFT*

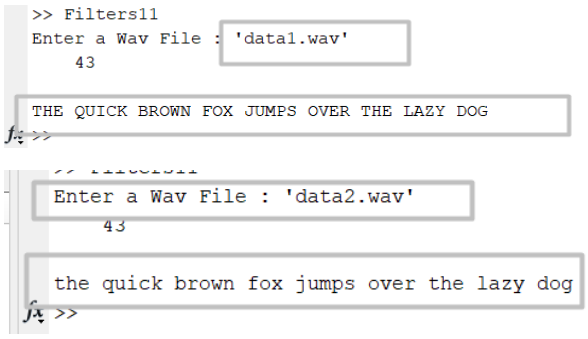
While the next two sentences have been tested in ***Fourier Transform*** decoding part to cover the situation of the overlapping upper and lower characters with spaces ***"BirZeIT UniVeRsi Ty"*** and ***"DIGital SiGnal ProCessiNg"***, the results of the percentage of accuracy were **52.6%** and **72%** respectively.



```
Enter a statement : 'BirZeIT UniVeRsi Ty'
>> Part2
Enter a Wav File : 'data3.wav'
    19

the retrieval string:
irI iVRi y
>> Part1
    53

Enter a statement : 'DIGital SiGnal ProCessiNg'
>> Part2
Enter a Wav File : 'data4.wav'
    25

the retrieval string:
DIGial iGal ProCig
```

Figure 8: *the decoded last two sentences using FFT*

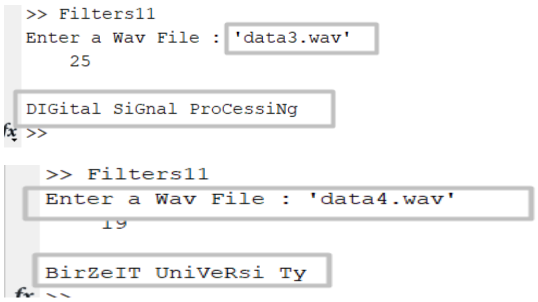
For the filter part, the same files were used, and the accuracy was **100%** for all the testing cases we have used as an evaluation Criteria. The below figures shows that the retrieved string is the same as the original one and the percentage of losing data was **0%.**



```
>> Filters11
Enter a Wav File : 'data1.wav'
    43

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
>>
```

```
Enter a Wav File : 'data2.wav'
    43

the quick brown fox jumps over the lazy dog
>>
```

Figure 9: *the decoded first two sentences using Filters*



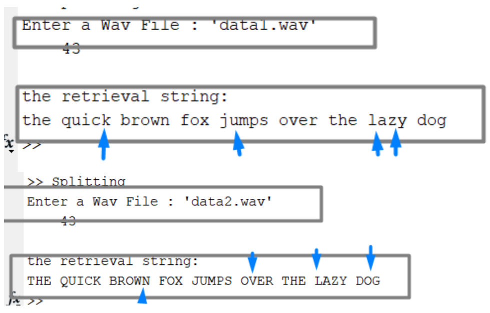
```
>> Filters11
Enter a Wav File : 'data3.wav'
    25

DIGital SiGnal ProCessiNg
>>
```

```
>> Filters11
Enter a Wav File : 'data4.wav'
    19

BirZeIT UniVeRsi Ty
```

Figure 10: *the decoded last two sentences using Filters*

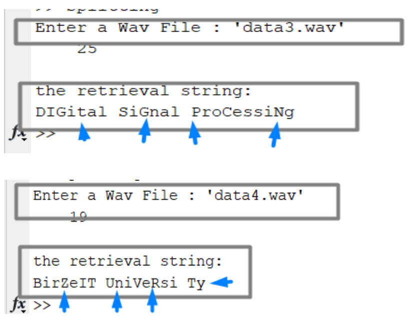
VII. Development

From the previous part, the similarities between the original string and the retrieved string in the Fourier Transform part, are not close and the occurrence is from **(60-70) %.** So, to make the system more accurate some cases were handled. For example, in the case of these letters **(H, h, k, K, u, Q, l, L, B)** sometimes they did not appear because of their last frequency, most of the cases it equals **(3000 -3800)**, not **4000** as specified so, an if – statement was implemented to catch this error and handle it. The same is done for the letter **(K)**. The last frequency should be **2400** but, in the program, sometimes it equals **1900** also, the letters (**E, e, Z, z)** their medium frequency should be **2000** as given but, the decoder gives **1900,** therefore an if statement where set to handle the error. And the last case is the space case, since the space character consists of three frequencies, the first frequency will be **1000** as specified, so any signal with the first peak equals **1000,** translate it to a space character. The figures below show some test cases after improvements with accuracy equals **100%.**



```
Enter a Wav File : 'data1.wav'
     43

the retrieval string:
the quick brown fox jumps over the lazy dog
>>
>> Splitting
Enter a Wav File : 'data2.wav'
     43

the retrieval string:
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
>>
```

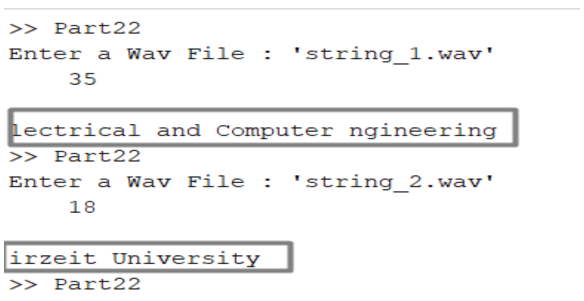
Figure 11*: the first two strings after improving the FTT*



```
Enter a Wav File : 'data3.wav'
     25

the retrieval string:
DIGital SiGnal ProCessiNg
>>

Enter a Wav File : 'data4.wav'
     19

the retrieval string:
BirZeIT UniVeRsi Ty
>>
```

Figure 12*: the last two strings after improving the FTT*

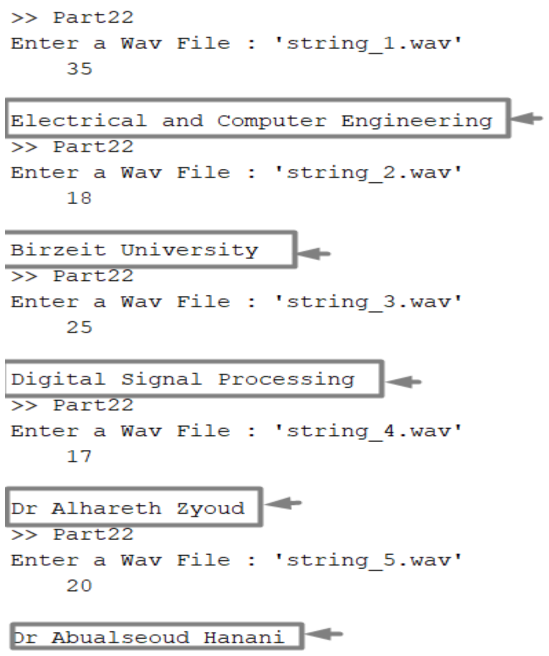
Finally, we have tested the waveform files that the instructor has sent, we noticed that there are some cases have noises in their frequencies that need to be handled in the decoding part. In another word, in the case of the following **letters (B, E, D, S, P, A, Z)**, they have not been printed on the screen in decoding operation. For example, when testing the first waveform file **"string_1.wav",** the output was "lectrical and Computer ngineering" ***instead of*** "Electrical and Computer Engineering", and the output of the second file "string_2.wav" was "irzeit University" ***instead of*** "Birzeit University" and so on. The following figure shows the output of some testing files before handling the cases that have noises:



```
>> Part22
Enter a Wav File : 'string_1.wav'
    35

lectrical and Computer ngineering
>> Part22
Enter a Wav File : 'string_2.wav'
    18

irzeit University
>> Part22
```

Figure 13: *the first two files before handling the noise*

After handling the noise for the following **letters (B, E, D, S, P, A, Z)**, the output of testing files becomes working correctly. The figure below shows all test files after handing the noise of all letters.



```
>> Part22
Enter a Wav File : 'string_1.wav'
    35

Electrical and Computer Engineering
>> Part22
Enter a Wav File : 'string_2.wav'
    18

Birzeit University
>> Part22
Enter a Wav File : 'string_3.wav'
    25

Digital Signal Processing
>> Part22
Enter a Wav File : 'string_4.wav'
    17

Dr Alhareth Zyoud
>> Part22
Enter a Wav File : 'string_5.wav'
    20

Dr Abualseoud Hanani
```

Figure 14: *testing all files after handling the noise*

VIII. Conclusion

This project was a great experience for us, many challenges and problems were faced, but with hard and teamwork they were solved successfully. Finally, the purpose and the main concept of encoding and decoding input strings were implemented, and it was working clearly as explained before.

IX. References

[1] . *Fileid*. Read data from text file - MATLAB. (2021, August 11). Retrieved January 1, 2022, from https://www.mathworks.com/help/matlab/ref/fscanf.html

[2]. How do I prevent peak clipping when using audiowrite? How do I prevent peak clipping when using audiowrite? -. (2014, July 28). Retrieved December 28, 2021, from https://www.mathworks.com/matlabcentral/answers/143620-how-do-i-prevent-peak-clipping-when-using-audiowrite