Strings Encoder Decoder

Baraa Atta 1, Ahmad Hamad2, Mohammed Muwafi3

1 Department of Electrical and Computer Engineering

2 Department of Electrical and Computer Engineering

3 Department of Electrical and Computer Engineering

[1180445@student.birzeit.edu](mailto:1180445@student.birzeit.edu), [1180060@student.birzeit.edu](mailto:1180060@student.birzeit.edu), [1180491@student.birzeit.edu](mailto:1180491@student.birzeit.edu)

# Introduction

This project is about designing an English character voice frequency encoder by a combination of three frequencies and a fourth one for the case of the character (capital or small). Also, a decoder is designed to recover the string by applying a .wav file, the decoder is designed using two approaches Fourier transform and Filters.

# Problem Specification

## Encoder

The encoder will take a string consisting of only English characters and spaces. Then, it will encode each character with its frequencies as in Table 1.

Table 1: Encoding Frequencies

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Characters | Capital/Small | Low F | Middle F | High F |
| A/a | 200/100 | 400 | 800 | 1600 |
| B/b | 200/100 | 400 | 800 | 2400 |
| C/c | 200/100 | 400 | 800 | 4000 |
| D/d | 200/100 | 400 | 1200 | 1600 |
| E/e | 200/100 | 400 | 1200 | 2400 |
| F/f | 200/100 | 400 | 1200 | 4000 |
| G/g | 200/100 | 400 | 2000 | 1600 |
| H/h | 200/100 | 400 | 2000 | 2400 |
| I/i | 200/100 | 400 | 2000 | 4000 |
| J/j | 200/100 | 600 | 800 | 1600 |
| K/k | 200/100 | 600 | 800 | 2400 |
| L/l | 200/100 | 600 | 800 | 4000 |
| M/m | 200/100 | 600 | 1200 | 1600 |
| N/n | 200/100 | 600 | 1200 | 2400 |
| O/o | 200/100 | 600 | 1200 | 4000 |
| P/p | 200/100 | 600 | 2000 | 1600 |
| Q/q | 200/100 | 600 | 2000 | 2400 |
| R/r | 200/100 | 600 | 2000 | 4000 |
| S/s | 200/100 | 1000 | 800 | 1600 |
| T/t | 200/100 | 1000 | 800 | 2400 |
| U/u | 200/100 | 1000 | 800 | 4000 |
| V/v | 200/100 | 1000 | 1200 | 1600 |
| W/w | 200/100 | 1000 | 1200 | 2400 |
| X/x | 200/100 | 1000 | 1200 | 4000 |
| Y/y | 200/100 | 1000 | 2000 | 1600 |
| Z/z | 200/100 | 1000 | 2000 | 2400 |
| Space | Don’t Care | 1000 | 2000 | 4000 |

Each letter will be encoded for 40 ms long and with sampling frequency FS = 8 KHz. Finally, the output would be a .wav file that have the concatenation of all waveforms for all letters.

## Decoder

The decoder will take .wav file and recover the string using two ways:

### Fourier Transform

In this way, it will take the Fourier transform of the input signal and take 4 frequencies that have the highest amplitudes in each 40 ms segment to determine the character and its case.

### Bank of Bandpass Filters

Here, it will use a bank of a very narrow bandpass filters that have all 9 frequencies for the letter and 2 frequencies for the case, the decoder will pass each 40 ms segment into these filters and pick the character that matches the highest 4 frequencies got from the bank of the filters.

# Data

The data given to the encoder would be a string consisting of only English words with spaces and during encoding of the string a cosine signal is used. For the decoder, it would be a .wav file. All the frequencies were taken from Table 1, that was given in the project description.

# Evaluation Criteria

For evaluation, the encoder was used to generate a waveform for about 32000 letters, then the waveform was decoded using the three decoders, and all of them recovered the same string.

Since the brute force approach usually gives a correct result, by trying the all possibilities, so we can say that our models gave as an accuracy of 100%.

# Approach

## Encoder

1. The user enters a string.
2. The system will make sure it only consists of English characters and spaces, anything else will be removed.
3. Loop over the string, and for each letter do the following:
4. Find the cosine of each frequency of the letter.
5. Take a segment of 40 ms and take samples from it at sampling frequency 8 KHz.
6. Concatenate the samples for all letters in the string which results in full waveform represents the string.
7. Write the result to a .wav file.

## Decoder

### Fourier transform

1. Reading a .wav file that have the encoded string.
2. Dividing the waveform to 40 ms segments at sampling frequency 8 KHz which means 320 samples for each letter.
3. Finding the Fourier transform of each segment.
4. Taking the 4 frequencies that have the highest amplitudes.
5. Finding each letter represented by the frequencies.
6. Concatenating the letters to get the full string.

### Bank of Bandpass Filters

1. Reading a .wav file that has the encoded string.
2. Dividing the waveform to 40 ms segments at a sampling frequency of 8 KHz which means 320 samples for each letter.
3. Passing each segment into the bank of filters.
4. Finding the maximum amplitude of the resulting output of each filter in the bank from its samples in the time domain.
5. Storing the amplitudes with their corresponding frequencies and sort them based of the value of the amplitude.
6. Taking the highest four amplitudes, and consider them as the frequency components of the current letter.
7. Comparing the four found frequencies with the table given in the project description, and find the letter that has those 4 frequencies and add it to the retrieved string.
8. Concatenating the suitable characters for each segment to construct the final string.

# Results & Analysis

To test our model, we will start by encoding a string and try to retrieve it again by the two models we have. Figure 1 shows the encoding process.

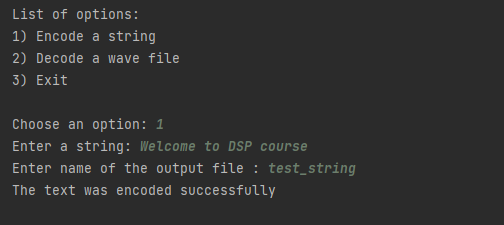


Figure 1:encoding the string

As shown in the previous figure, we generated a wave file called “test\_string” for the string “Welcome to DSP course”.

Figure 2 shows the decoding process.

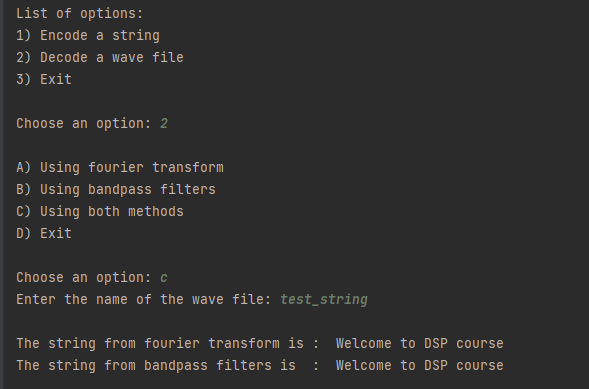


Figure 2:decoding the wave file

As shown in the previous figure, we have decoded the same file that we generated in the encoding stage using the two decoding methods, moreover we can notice that both methods return the same string, which equals the one we encoded previously.

Next, we tried to decode the five wave files, that were sent to us by the instructors. In all the wave files, both methods of decoding gave us the same output, which can be seen in table 2.

Table 2: the output after decoding the five challenging strings

|  |  |  |
| --- | --- | --- |
| Wave file name | FFT method output | Band-pass filters output |
| string\_1.wav | Electrical and Computer Engineering | Electrical and Computer Engineering |
| string\_2.wav | Birzeit University | Birzeit University |
| string\_3.wav | Digital Signal Processing | Digital Signal Processing |
| string\_4.wav | Dr Alhareth Zyoud | Dr Alhareth Zyoud |
| string\_5.wav | Dr Abualseoud Hanani | Dr Abualseoud Hanani |

# Development

In the future work, we could design a very narrow band pass filter which will be able to extract the expected impulses that make the signal and ignore the unwanted frequencies so that the accuracy will be 100% every time. And we can use another way to test the samples instead of using the cross-correlation that takes about O(Duration2). And finally, we could build a user-friendly GUI that allows any user to use the app.

# Conclusion

In this project we implemented a strings encoder and decoder DSP system.

According to the encoder, we started by writing the samples to the wav file in one shot, but then we realized that the output file was very large, so we decided to write it frame by frame instead, which reduced the size of the file to approximately the half, otherwise every thing went on smoothly without any problems to talk about.

For the decoder, we implemented it using the two methods in the project description, which are the Fourier transform and the bank of bandpass filters.

According to the results we got, we tried to generate a very large wav file with about 31000 letter and apply it to both decoders we have, and both of them return the same expected string, so we can say that the accuracy was 100% on all wave files we generated, also we asked some different groups to generate wav file for us, and our three models return the same expected string.

# References

1. Van Rossum G, Drake FL. Python 3 Reference Manual. Scotts Valley, CA: CreateSpace; 2009