EEE4120F SSDC Project: Milestone 2

1 May 2019

Stefan Schröder (SCHSTE054) and Elle Mouton (MTNELL004)

Introduction

This report contains an outline of the theory that we will be using to implement the transmitter and receiver for this project as well as descriptions for how they will be implemented.

Encoding the and transmitting the data

Encoding a message:

In order to include the message into the wave sounds it must first be converted into binary format. We have decided to use the Huffman Tree Encoding technique in order to achieve this. A Huffman tree is made by creating a binary tree of the letters of the alphabet as well as any other necessary characters and assigning each letters unique string of bits. The more frequent the letter appears in words, the higher up it will be in the binary tree and the shorter it's uniquely assigned bit string will be. Figure 1 below shows the relative letter frequency and Figure 2 shows an example of what a Huffman tree looks like.

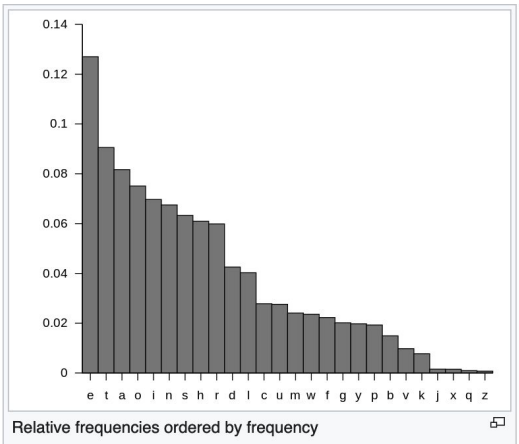

Figure 1: Relative frequencies of different letters [2]

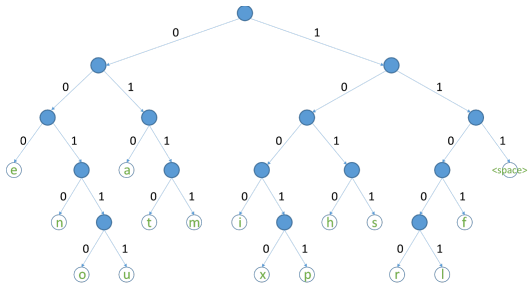

Figure 2: An example of a Huffman binary tree [1]

By using the Huffman encoding technique, the number of bits per letter that needs to be encoded and transmitted becomes drastically less.

For example, if the message being sent is *“testing”* then then the Huffman Encoding of this message using the letter frequencies shown in Figure x is the following string of 36 bits:

"010110110100000010110010000010001011"

If ASCII encoding was used then for each character in the string being sent, 8 bits would be required and so 56 bits would have been required to send the *“testing”* message.

Modulating and sending the encoded data:

As discovered in the research done for Milestone 1, wave sounds resemble a white noise distribution in the frequency domain.

We decided to encode the data by splitting the string of encoded bits into sections of 4 bits each and then using ½ a second to send each section of 4 bits.

To encode each section of 4 bits, we first convert the huffman encoding output into 4 waves, each at their own frequency. Each of these waves have half the amplitude of the previous (lowest frequency being the highest), this is done to emulate the sound of the ocean better. These waves will be encoded as; full amplitude being considered a 1 and 1/3 amplitude being considered a 0. Each wave will encode the huffman output by splitting it into groups of 4, and then giveing each wave frequency one of the bits. After these waves each have their amplitudes encoded, they are added together and made into array. After this the wave frequencies are convolved with noise “bubbles”, which is a band of noise. This is so that each encoded peak sounds similar to the ocean.

Since we are sampling at 16kHz we need to ensure that the data is only created with in the frequency band of ½ of 16kHz (according to nyquist). So all of the waves and their “noise bubbles” need to be less than 8kHz.

Figure 3 below shows the process of creating the audio section that would encode the bitstream ‘1010’.

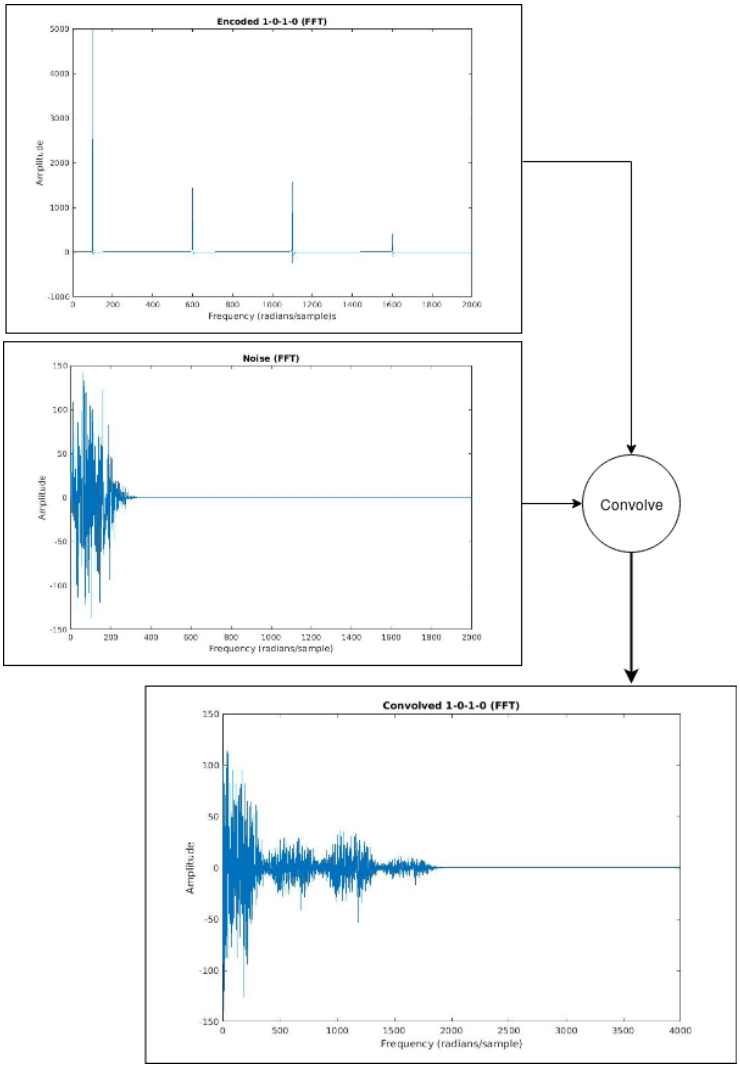

Figure 3: Diagram showing how bandlimited noise is modulated with a data signal to encode the data in wave sounds

Some matlab sample code is shown below, to show how the send signal is generated. This sample code will show the process for one wave centered at frequency of 220Hz

Main send function

```
syms t;
signal(t) = cos(2*3.14159*t);
peakOne = 220;                  % frequency of first  peak
sampleRate = 16000;
bitRate = 0.5;                  % seconds per bit
encode = “01”;

% create wave
rang = linspace(1,bitRate,sampleRate*bitRate);
```

```
waveOne = signal(peakOne*rang);

% create final wave
finalWave = [];
for i:length(encode)
   switch(char(encode)(i))
   Case “0”
      finalWave = finalWave + 0.5*waveOne;
   Case “1”
      finalWave = finalWave + waveOne;
end

% get noise bubble (frequency convolve is time multiply)
noise = noise_bubble(length(finalWave));
finalWave = finalWave .* noise;

soundsc(finalWave, sampleRate);
```

Noise_bubble function

```
function y=noise_bubble(len) % returns band limited noise
        mu=0;
        sigma=1;
        x=sigma*randn(len,1)+mu;

        load("Filters/low16k200.mat"); % loads the impulse response for low pass filter
        y=conv(x,impresp);
        y = y(1:len);

end
```

Receiving and Decoding the data

The receiver will get an audio file. The receiver will have information regarding how the transmitter encoded the data such as: the Huffman Tree, the length of time that each group of bits is sent for (bit-rate), the sample rate used by the transmitter and the frequency bands that it will use to modulate the data. The code used for splitting the audio file up into pieces is shown below:

```
[y,sampleRate]=audioread(filename);
len_bit_piece = sampleRate*bitRate;
num_chunks = length(y)/len_bit_piece;
y_split = reshape(y, len_bit_piece, num_chunks);
```

To demodulate the data, the audio file will be split into pieces (the size of these pieces is based on the bit-rate and the length of the audio file). For each piece, the Fourier Transform can be taken and the band power for each of the predetermined ranges of frequencies can be calculated. For each band of frequencies being examined, the power in the band will be calculated. If the power is above a certain threshold then that corresponds to a ‘1’ otherwise it corresponds to a ‘0’. Figure 4 below shows the FFT of an audio slice that would be decoded as the bits ‘1111’ and figure 5 below shows the FFT of an audio slice that would be decoded as the bits ‘1010’.

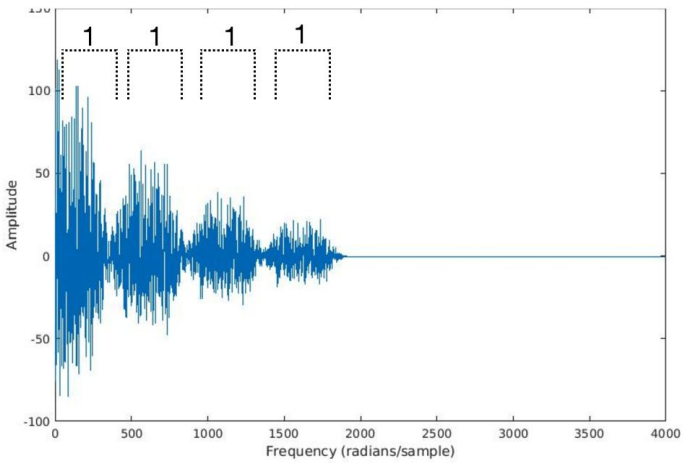

Figure 4: FFT of an audio slice that will be decoded as ‘1111’

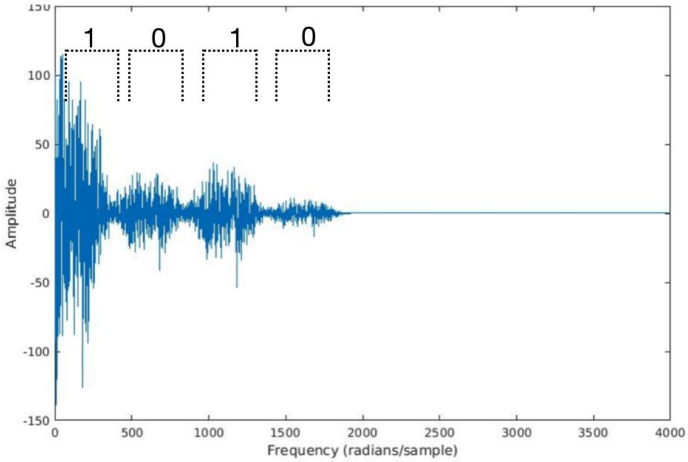

Figure 5: FFT of an audio slice that will be decoded as ‘1010’

The code below shows an example of how the bandpower of a specific frequency range can be calculated and then used to determine whether a ‘0’ or ‘1’ is present. The code that we will use will be similar but will include calculations for 4 different frequency bands.

```
final = “”;
for i = 1:num_chunks
      sample =  y_split(:,i);
      bp1 = bandpower(sample, sampleRate, [170 270]); %for band centered at 220

      if(bp1<=thresh1)
         final = final + "0";
      else
         final = final + "1";
      end
   end
```

This process will be done for each band in each piece of the audio recording and as a result a string of 1's and 0's can be constructed. This string will then be decoded with the Huffman Tree.

Figure 6 below shows a block diagram of the receiver sub-system.

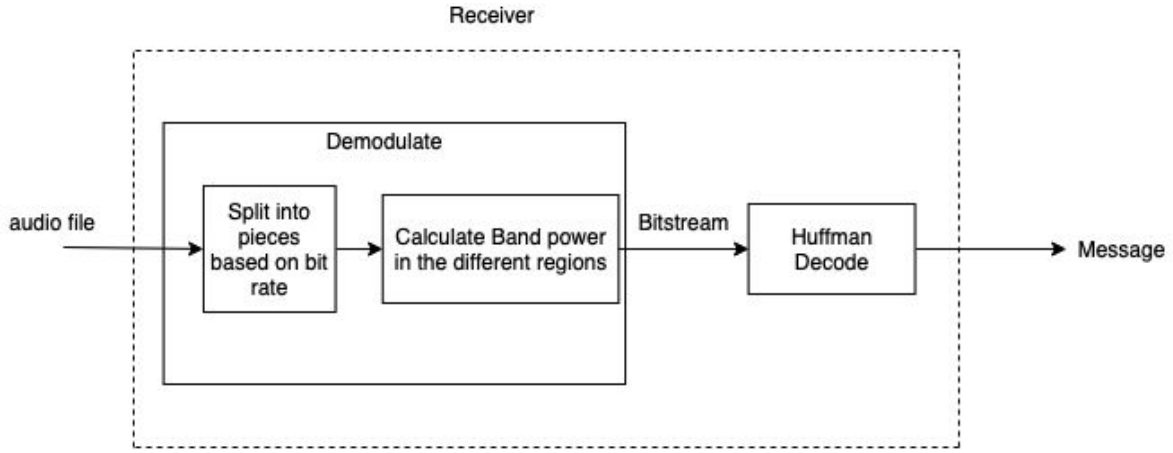

Figure 6: block diagram of the receiver sub-system

References

[1] Ghent University. "Huffman Coding." *Dodona*, dodona.ugent.be/en/exercises/601074743/.
[2]"Letter Frequency." *Wikipedia*, Wikimedia Foundation, 14 Mar. 2019, en.wikipedia.org/wiki/Letter_frequency.