EECS126 Final Report

Gabe Bertero, James Eigenbrood, Pranav Madanahalli

December 2019

1 Introduction

The goal of this project was to write a program such that given two computers, speaker and receiver, speaker can convert a text file to a series of sounds that receiver records and converts back to the original text file. This process must account for interference produced by any sound present in the environment the sound is being emitted and recorded in as well as be able to communicate at a rate of at least one hundred bits per second. The basic summary of the steps taken by the project is as follows: load the text and convert it to a binary representation of the data, generate an audio file by assigning each bit in this sequence a sample, play said sequence to the receiver, the receiver then converts these sounds back into bits, and decodes the bits into the original text file. This summary, simplifies the problem and does not acknowledge exactly what a sample is and what a sample consists of.

Understanding the sampling required us to learn about the Wave Form Audio File Format(WAV).The format is a type of Resource Interchange File Format (RIFF) and as such consists of a three contiguous chunks of data- a header chunk, a format chunk, and a data chunk. The header chunk indicates that the file is a RIFF file and further that it is an instance of a WAV file, the header file contains meta data regarding the data chunk, and the data chunk is the meat of the file that and contains an array of signed values that consists of the audio when interpreted in the context of the given format chunk. Sound itself is represented as a wave that propagates in all directions effecting its environment by causing pressure changes in its surroundings. When these waves reach a microphone or receiver, these changes are recorded. Samples indicate the signed values of a wave (i.e the amplitude of the wave) at some point in time and are the most basic unit in an audio file. The ordered sequence of samples create a wave stored in what is referred to as a channel. In many cases there is more than one channel that each represent the wave of a different sound;however, in this project only a single channel is used. In the presence of many channels , a frame contains the sample values of multiple channels at a given point in time, but as there is only one channel, the presence of a frame would simply reflect the value of the channel at that point. Thus when a noise or sound is assigned to a piece of data in the encoding of the array representing the text file, what is occurring is a sequence of samples are being appended to the array representing text file as an audio file.

2 Program Break Down

The project is structured into three files. The generator (generator.py), the receiver (2reader.py), and a processor (wave_to_freq.py). The program is run by running a the generator from the command line with two arguments samplerate and file.txt. At the same time the receiver must also be run (it requires no inputs) so that it is prepared to record any sound that gets played by the generator.

Getting generator to function as expected required us to figure out how one goes from digital data to analog sound. As explained above a sample is the most basic unit of sound processing and represents the amplitudes of the wave at some period in time. The amplitude of the wave dictates the level of sound of the wave not the sound itself. The sound produced by a wave is dictated by the frequency at which these samples are played. Since we were concerned with only transmitting ones and zeros we settled on the premise that we only required to frequencies. Further the sample rate dictates the number of samples that get played per second. Thus given a bit array with the pyaudio import,sample rate,a baseline amplitude, and these two frequencies we could generate the appropriate audio. We defined these frequencies as global variables freq0 = 2000 hertz and freq1 = 600 hertz in lines immediately following imports and had the sample rate from the command line. This left only the task of encoding the text file from a string to a string of ones and zeros representing the data in the file.

The encoding of the file is handled in lines nineteen to twenty one of the generator function.After having loaded the file into a string, the string was converted into bytes then bits. The bit array required two stages of pruning. Firstly, the leading two characters in a bit array where always "0b" and as such where to be removed as we did not intend to encode b and as these two where standard we could easily handle that in processing. Secondly, the array had to be of a length equal to a multiple of eight (the size of a byte). using the zfill function we were able to easily pad the string with enough zeros so as to ensure the aforementioned condition was met.

While this data is all that is needed there is still some processing required to make things work. Originally we attempted to simply emit two set amplitudes for each one and zero at each frequency;however, this led to the quick realization that for a sound to be emitted, many (on the order of thousands of samples) must be played at a given frequency, thus two sequences were defined. These two sequences were defined as:

$$sin(\frac{2*\pi*freq_j*n}{1200})*amplitude$$

Here j is either one or zero corresponding to whether or not zero or one was passed and amplitude is a scaling factor.With these sequences a sound is played for each one or zero.Using the pyaudio library we could create a stream by passing calling the open function and passing in the sampling rate ,the number of channels (1), and setting output to True (indicating that the stream would collect sequences and play once the stream closed).Once the stream was initialized playing the sound was simply a matter of iterating through the bit array and writing each sequence to the array appropriately. Once done iterating the stream is closed and the sound plays. Up until this point we had minimal issues and the program functions as intended. Our problems began to arise when dealing with the receiver and processor.

The first problem we encountered was handling when to start and stop recording. Ultimately, we were not able to figure out how to do this and settled on simply setting a global variable RECORD_Seconds to indicate how long the receiver should record for. For testing purposes we set this value knowing preemptively how long everything we tested ran for by running generator once before and setting the time appropriately after. In the receiver (the 2read file) the same global variables where defined as they where in generator, and a pyaudio stream was again initialized with the same parameters as before except this time keeping output as false and setting input to true (implying that this stream was to listen to the surrounding sounds). With the information we picked up we wrote a wavfile using the Scipy.io.wavfile library. This was relatively easy to handle and simply required passing in the data from the recording and the declaring the number of channels,rate,samplewidth,and the sequence of frames. Since there was only one channel the sequence of frames was really just all the polled amplitudes and the samplewidth indicates the number of bits available to represent each sample as a float (in our case this value was 32 bits). While the reader works as intended, on all three group member machines the plots of the data did not provide variation between the regions that where ones and zeros and made processing much more difficult. We did however try using an iphone app that allowed us to record the sound from generator as a WAV file and send it over. In this instance the distinction was much clearer and we were able to do some processing with it.

The scipy library allows for the reading of WAV file into a numpy array and a rate using its read function. The array is very large ;however, upon inspection (by plotting it) we realized that until the generator played sound the values where very small. To handle this we sampled randomly selected the absolute values from the first 44100, this value was chosen as its the rate the app was recorded. (we used this file when processing the and took their mean. While this mean was below a certain value we would remove these first set of values. This allowed us to reduce the array into just the information we had truly intended to receive.Once this condition was met (the mean was greater than the given value). We then removed the first "rate" (as returned from the scipy read command when we first established the array) and took the mean of points in it. If these points where greater then $.4 * 10^9$ the points represented a one and otherwise they were a zero. We chose this value to compare to from trial and error and inspecting the plots we had done before. Using the WAV recorded by the iphone this was an effective method to record array correctly;however, using our own recorded WAV file this did not work.

3 Conclusion

Ultimately, we had trouble with our reader getting to have a significant enough difference between ones and zeros and starting and ending the recording. However the generator works entirely as expected and from this project we learned not only about how analog sound is represented digitally,but also how it is handled using the pyaudio and scipy library.