ECE 132A Final Project

Dual Tone Multi Frequency Signaling (python)

Tait Kaminski, 904964310

Abstract – In this project we look to prove that it is possible to exchange files between two laptops over the air using a Dual Tone Multi Frequency Signaling (DTMF) scheme as done by Mortiniera Thevie and associated group members[1]. In specific, we will implement a transmitter that reads a text file and plays sounds from a computers speakers that is then received by an alternate laptops microphone that reproduces the text file.

Introduction

Dual Tone Multi Frequency Signaling (DTMF) is a system used to telecommunicate signals to other communication devices using the voice-frequency band. MF signaling uses a mixture of two pure sine wave sounds following protocols developed by Bell Labs back in the mid to late-1900s. The general formulation for a pure DTMF signal is characterized by the following equation:

$$x(t) = A_M cos(2\pi f_L T + \theta) + A_M cos(2\pi f_H T + \theta)$$

where A_M is the amplitude, $f_L \; and \; f_H$ are the low and high frequencies, T is the duration of the signal based on number of samples and θ is the phase shift.

DTMF works by assigning eight different audio frequencies to rows and columns of the keypad. These can then be further divided into two groups based on column and row, "low-group frequency" (697-941 Hz) and "high-group frequency" (1209-1633 Hz) respectively. In the equation above, this can be seen by the $f_L \; and \; f_H$ terms. DTMF signaling replaced rotary dials on telephones with the 0-9, A-D, *, # keypad similar to what we still see today with the additional A-D column. When the user presses a number, the phone generates a tone using the signal pair corresponding to that number, which is then transmitted for exchange where the two signals are decoded to determine the number sequence dialed.

		High-Group Frequencies			
		1209Hz	1336Hz	1477Hz	1633Hz
Low-Group Frequencies	697Hz	1	ABC 2	DEF 3	A
	770Hz	GHI 4	JKL 5	MNO 6	B
	852Hz	PRS 7	TUV 8	WXY 9	C
	941Hz	*	OPER 0	#	D

Figure 1: Dtmf keypad

DTMF is still relevant today. One of its primary applications is voice mail dials. When a user presses a number to connect them to a different line or whatever the case may be, the machine uses the DTMF signal to recognize the number. This project attempts to act as a proof of concept for this idea.

Part 1: Signal Generation

Step one is to transmit the data. We can define the input to be a 160-character long string containing a random mix of upper and lower-case alphabetical letters. The first thing we do is convert the string into double, which is then converted into binary. The string is then padded with the barker codes (see appendix for explanation of the algorithm). From a high-level overview, we then encode and create the waveform which is played from our first computer. I choose to test 6 different input strings to compare results. Here are the amplitude frequency plots of the emitter for each of the 6 strings containing no noise obtained by taking the Fast Fourier Transform (FFT).

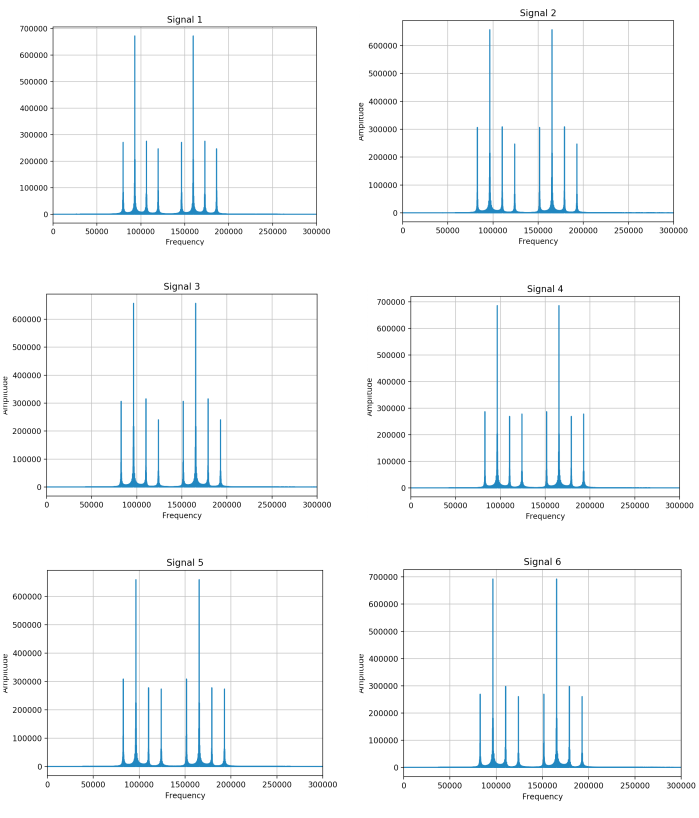

Figure 2: FFT plots for the 6 messages containing no noise

Then in order to demonstrate the impact of noise during transmission, we can use the two white Gaussian noise wavs provided. One containing 1-2 kHz noise and the other containing 2-3 kHz noise. This noise was played via a third source during emission and transmission (a picture of the setup is shown in appendix). The noise signals make transmission outside of the given region difficult and thus impact the received message. For each of the 6 signals, I tested the impact of the 1-2 and 2-3 kHz noise on the output. The results are shown in the next section.

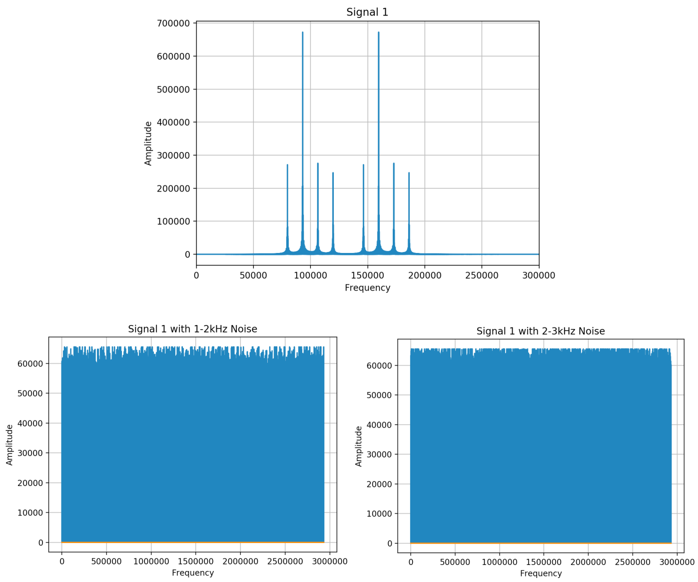

Figure 3: Example of the effect of noise on just the output of the emitter containing 1-2 and 2-3 kHz noise. Only signal 1 plots are shown here, but the same was done for all 6 signals. These again are not the plots post reception; I am simply displaying the effect of noise on the FFT of the signal.

PART 2: SIGNAL RECEPTION

Upon transmitting the signal through a noisy channel, we then turn to the receiver to load in the wav file in order to decode the signal. From a high-level overview, in this portion we filter and then decode.

The outputs for each of 3 noise scenarios given each of the 6 input strings are given here.

Noise file used	Output string	Observation
none	abcDEFghiKLMnopQRStuvWXYzabcDEFghiKLMnopQRSt uvWXYzabcDEFghiKLMnopQRStuvWXYzabcDEFghiKLM nopQRStuvWXYzabcDEFghiKLMnopQRStuvWXYzabcDE FghiKLMnopQRStuvWXYzKART	Fully recovered
1-2khz 2-3khz	Full output Full output	Fully recovered Fully recovered
none	XTLlQyFVgdEAWHnWsuvejJfJKTcKHbReFaLgmcuHBKlP xCCVPzHVZlFPSJnuxmlDzSsUyZkobSrUqIZcClnnzuhJKM bwBtWhROoDCPlxohsXOnXVlXYWBmBZApfkUnlUifDG ddXVWBBaMAtxoEigwlIDWAHLJXpH	Fully recovered
1-2khz	Full output	Fully recovered
2-3khz	XTLlQyFUWdEAWHmWsuvejJfJKTcKHbReFaLgmcuHBK lPtCC_PzDVZlPSJnuxmlDzSsUuZg_bSrUqIYcClnnyuhF[M RwBuWhQOoDCPlx_hsXOnXVlXYWBmBZApfkU^lUifD GddXVWBBaMAtxoEigw\IDWAHLJXpH	Partially recovered
none	npCQuBAtKhhJEQuoMnEgdJveDegZnDpGVIOKFAEhWlG TrVIarkFvBhTrlWTmMydiWbbkNBJsDphytqHppeVildlENk jLucrVDkkYOIUeRitwNoOshjrFRbKhYaplQzzWptTLxBfm CEPgbdMBWSpRjkPXDhjDnsdi	Fully recovered
1-2khz 2-3khz	Full output Full output	Fully recovered Fully recovered
None	AMdlKxYXMZeVFpuMlqQPVSnQzEPveLNDlYpsRbdjyA UEuITlVtwoMCiBOosDOuGVvKzNDghhJYXSiwRMQaGy ksGsyxtuzhLccULNRtchIkCrKevlwRKNtzizwobTSqCDbEgI WvQELuJKfwjUTifuNjLACTAcNerx	Fully recovered
1-2khz 2-3khz	Full output Full output	Fully recovered Fully recovered
None	cktrJWFFaiqGiPAnGhMVGoDvUrAyejHHAioJuQrlMpGvc NgZRqvAaruYMdGspYLqXUsSwKcJFcHaNLqsoowdHnnH BOmCKtCNSDLEhdcfbkGSxiqedlvKEtkVhWRqgTcmtSEX sUQcYvrhInHaGqnklZdPRpCgxaFQ	Fully recovered
1-2khz 2-3khz	Full output cktrZWFFeiqWiTEnGk]VWoDvUrAyejIHAioZuUvlMpWvg ^gZRqvQeruYMdGspYLqXUsSwKcJFcHaNLqsoowdInnHR OmCKtC^SDLEhdcfbkGSyiqedlvKEtkVhWVqgTcmtSEXsU QgYvvhInXaGqnkmZdPVpCgxaFU	Fully recovered Partially recovered
none	ERYSvTgRFPGtoVChGCeTGPWKOPEkvDcfancTkEnIHof VSVFnvjDbuCoTSjEheTMqJzoFtvqsMklTRjqfATPyvnRPA glZerRPOVFgPOtJGiascwlGOSToVQvvsDgqEPkZpzbpfgkcl GZQBueKIyklWitdkRjVylch	Fully recovered
1-2khz 2-3khz	Full output Full output	Fully recovered Fully recovered

There are some interesting results that can be seen from this table. There appear to be two different trends. The first trend is full output for all three schemes. The second trend is full output for no noise and 1-2kHz noise while the output for the 2-3kHz noise is partially recovered, meaning part of the input string remained intact but the rest of the output was random.

For the first trend, signal 1, 3, 4 and 6 follow the pattern. We handle both cases of the 1-2 kHz and 2-3kHz noise with no problem. This result is the expected and proper outcome. The assignment was built with the expectation that the code could handle either noise scheme at random.

As for the second trend, signal 2 and 5 follow the pattern. For example, in signal 2, we see full output for 1-2 kHz, but then for 2-3 kHz we see an output that closely resembles the original string in places but contains mixed up letters throughout most of the string. Below are the two outputs with differences shown in bold for the noisy output.

Original:
XTLlQyFVgdEAWHnWsuvejJfJKTcKHbReFaLgmcuHBKlPxCCVPzHVZlFPS
JnuxmlDzSsUyZkobSrUqIZcClnnzuhJKMbwBtWhROoDCPlxohsXOnXVlXY
WBmBZApfkUnlUifDGddXVWBBaMAtxoEigwlIDWAHLJXpH

Transmitted:
XTLlQyF**UW**dEAWH**m**WsuvejJfJKTcKHbReFaLgmcuHBKlP**t**CC_Pz**D**VZlPSJ
nuxmlDzSsU**u**Z**g**_bSrUqI**Y**cClnn**y**uhF**[**M**R**wB**u**Wh**Q**OoDCPlx_hsXOnXVlXY
WBmBZApfkU^lUifDGddXVWBBaMAtxoEigw\ID**W**AHLJXpH

I believe the error happens as a result of mistakes during the filtration phase. However, exactly what or why the error is occurring is harder to describe. Since they are all relatively minor one letter long mistakes, I believe we could attribute it to environmental error such as background interference, poor reception or audio quality, etc. Signal 5 has a very similar output to this, and I again believe, it could be the same problem with the same source of error.

Discussion

Overall, this project was a fun way to learn about DTMF. I was shocked to learn that this was the technology in a lot of voicemail receptions, and it was cool to make that connection. Taking a 160-character string, transmitting it and receive it all in live time between two totally separate computers was very interesting and never something I would have thought possible with such a relatively small amount of code. It also had really high Accuracy. We then also got to see the effect of noise on our DTMF transmission as well.

Appendix

Explanation of the algorithm/setup

1. The emitter coverts the text to binary, then transmits 0 or 1 DTMF signals. Barker code of length 13 is used for the headers and ending sequence. Barker code aims to improve range resolution for relatively long transmission pulses. They are sequences of +1's and -1's which contain unique sidelobe ratio sounds in dB that seek to meet the condition of autocorrelation as close as possible.

2. On the receiving end, we load the generated wav file containing either the original or distorted signal. The signal is then passed through narrow bandpass filters to extract the 0 and 1 frequency components. Then we run it through a function to decode the letter using a built-in python decode function.

3. On one laptop, I ran emitter.py while on my other laptop, I ran reciever.py with my phone placed in between (close to the emitter as directed) playing either the 1-2 kHz or 2-3 kHz noise depending on the trial. Receiver.py outputs the text in the terminal upon completion of the emitter.

Figure 4: Setup, emitter on right and receiver on left, noise source not pictured here (because I was using it to take this picture)

Input strings:

Signal1:
abcDEFghiKLMnopQRStuvWXYzabcDEFghiKLMnopQRStuvWXYzabcDEFghiKLMnopQRStuvWXYzabcDEFghiKLMnopQRStuvWXYzabcDEFghiKLMnopQRStuvWXYzabcDEFghiKLMnopQRStuvWXYzKART

Signal2:
XTLlQyFVgdEAWHnWsuvejJfJKTcKHbReFaLgmcuHBKlPxCCVPzHVZlFPSJnuxmlDzSsUyZkobSrUqIZcClnnzuhJKMbwBtWhROoDCPlxohsXOnXVlXYWBmBZApfkUnlUifDGddXVWBBaMAtxoEigwlIDWAHLJXpH

Signal 3:
npCQuBAtKhhJEQuoMnEgdJveDegZnDpGVIOKFAEhWlGTrVIarkFvBhTrlWTmMydiWbbkNBJsDphytqHppeVildlENkjLucrVDkkYOIUeRitwNoOshjrFRbKhYaplQzzWptTLxBfmCEPgbdMBWSpRjkPXDhjDnsdi

signal 4:
AMdlKxYXMZeVFpuMlqQPVSnQzEPveLNDlYpsRbdjyAUEuITlVtwoMCiBOosDOuGVvKzNDghhJYXSiwRMQaGyksGsyxtuzhLccULNRtchIkCrKevlwRKNtzizwobTSqCDbEgIWvQELuJKfwjUTifuNjLACTAcNerx

Signal 5:
cktrJWFFaiqGiPAnGhMVGoDvUrAyejHHAioJuQrlMpGvcNgZRqvAaruYMdGspYLqXUsSwKcJFcHaNLqsoowdHnnHBOmCKtCNSDLEhdcfbkGSxiqedlvKEtkVhWRqgTcmtSEXsUQcYvrhInHaGqnklZdPRpCgxaFQ

signal 6:
ERYSvTgRFPGtoVChGCeTGPWKOPEkvDcfancTkEnIHofVSVFnvjDbuCoTSjEheTMqJzoFtvqsMklTRjqfATPyvnRPAglZerRPOVFgPOtJGiascwlGOSToVQvvsDgqEPkZpzbpfgkclGZQBueKIyklWitdkRjVylch

*Also note I show a bit of additional code on the next page that I wrote for some plotting.

References

[1] Thevie, Mortiniera, Sinnathamby, Karthigan, Ndoye, Mohamed, Myotte, Frederic, "Text transfert over audio channel", *Github*, 3/5/2021. https://github.com/Mortiniera/text_transfert_over_audio_channel

[2] Various Authors, "Dual-tone multi-frequency signaling," *Wikipedia,* 3/5/2021.

[3] DeBari, Joe, "DTMF Tones and Signaling Explained", *Onsip*, 3/5/2021.

[4] Truong, Dangman, "DTMF", *Github*, 3/5/2021. https://github.com/dangmanhtruong1995/DTMF

[5] Various Authors, "Barker code", *Wikipedia,* 3/5/2021.

[6] Borwein, P, "Wieferich pairs and Barker sequences", *LMS Journal of Computation and Mathematics,* 17(1) S. 24-32, 3/5/21.

[7] Nagi, J, Yap, K, Tiong, S, Ahmed, K, "Intelligent Dection of DTMF Tones using a hybrid Signal Processiong Technique with Support Vector Machines," *ResearchGate*, 3/5/2021.

I basically used the code for the emitter and the receiver as seen exactly in the repository, however this is the simply additional code I wrote for the plots in figure 1 and 2 (this is the code from specifically figure 2, i.e. all just change wav.read file and plt.title…)

```
import matplotlib.pyplot as plt
import numpy as np
import os
import scipy.io.wavfile as wav
from scipy import signal
from scipy.fftpack import fft, ifft
import numpy.random as random

fs, audio = wav.read('emitter2b_w2-3.wav')
plt.plot(audio)
plt.title('Signal 1')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.grid()
plt.show()
plt.show()

yf = np.abs(np.fft.rfft(audio))
plt.plot(yf)
#plt.xlim(0,300000)
plt.title('Signal 1 with 2-3kHz Noise')
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.grid()
plt.show()
```