Dual Tone Multi-Frequency (DTMF) Signaling

Vincent Edwards, Julia Corrales, and Rachel Gossard

June 10, 2025

Contents

1. Encoding Program

DTMFwrite.py

```
# Program to encode a sequence of single digits into a DTMF sound (written to a .wav
file)

import numpy as np
import wave # Necessary for writing the .wav file
import struct # Necessary for writing the .wav file

file_name = "media/TestSignals/TenDigits.wav" # Output file name (must include .wav)

number_list = [0,1,2,3,4,5,6,7,8,9] # List of digits (0-9) to be encoded into sound

sample_rate = 44100
sound_level = 4096
sound_length = 400
pause_length = 200

def create_pure_tone_data(freq):
    return np.array([sound_level/2 * np.sin(2.0 * np.pi * freq * x / sample_rate) for
x in range(0, sample_rate)]).astype(np.int16)

array697 = create_pure_tone_data(697)
array770 = create_pure_tone_data(770)
array852 = create_pure_tone_data(852)
array941 = create_pure_tone_data(941)
array1209 = create_pure_tone_data(1209)
array1336 = create_pure_tone_data(1336)
array1477 = create_pure_tone_data(1477)

tone_list =
[sum([array941,array1336]).tolist(),sum([array697,array1209]).tolist(),sum([array697,array1336]).toli

sound_data = []
for i in range(len(number_list)):
    sound_data += tone_list[number_list[i]][:int(sample_rate*sound_length/1000)]
    sound_data += [0] * int(sample_rate*pause_length/1000)

# Start to write the .wav file
wav_file = wave.open(file_name, "w")

# Parameters for the .wav file
nchannels = 1
sampwidth = 2
framerate = int(sample_rate)
nframes =
(int(sample_rate*sound_length/1000)+int(sample_rate*pause_length/1000))*len(number_list)
comptype = "NONE"
compname = "not compressed"
```

```
wav_file.setparams((nchannels, sampwidth, framerate, nframes,
    comptype, compname))

# Write the data to the file
for s in sound_data:
    wav_file.writeframes(struct.pack('h', int(s)))

wav_file.close() # Finish writing the .wav file

print("Writing " + file_name + " complete!")
```

Output:

```
Writing media/TestSignals/TenDigits.wav complete!
```

2. Decoding Program

DTMFread.py

```
# Program to read in and decode DTMF sound data from a .wav file

from numpy import *
import matplotlib.pyplot as plt # Necessary if you want to plot the waveform
(commented out lines at the end)
import wave # Necessary for reading the .wav file
import struct # Necessary for reading the .wav file

# These first few blocks read in the .wav file to an ordinary integer data list
file_name = "media/TestSignals/TenDigits.wav"
plot_name = "media/TenDigitsPlot.svg"

wavefile = wave.open(file_name, 'r')

length = wavefile.getnframes()
framerate = wavefile.getframerate()
save_data = []
for i in range(0, length):
    wavedata = wavefile.readframes(1)
    data = struct.unpack("<h", wavedata)
    save_data.append(int(data[0]))
# At this point the sound data is saved in the save_data variable

low_frequencies = [697, 770, 852, 941]
high_frequencies = [1209, 1336, 1477]
decode_matrix = [[1,2,3],[4,5,6],[7,8,9],[-1,0,-1]]

def slice_data():
    i = 0
    data_list = []
    streak_length = 2
    while i < length:
```

```
        if not any(save_data[i:i+streak_length]):
            i += 1
        else:
            j = 0
            current_signal = []
            while any(save_data[i+j:i+j+streak_length]):
                current_signal.append(save_data[i+j])
                j += 1
            data_list.append(current_signal)
            i += j + 1
    return data_list

def calculate_coefficient(data_sample, freq):
    a = 0
    b = 0
    N = len(data_sample)
    for i in range(N):
        y = data_sample[i]
        t = i / framerate
        a += y * cos(2 * pi * freq * t)
        b += y * sin(2 * pi * freq * t)
    return 2/N * sqrt(a**2 + b**2)

def decode_freqs(low_freq, high_freq):
    return decode_matrix[low_frequencies.index(low_freq)]
[high_frequencies.index(high_freq)]

sliced_data = slice_data()

for signal in sliced_data:
    low_coeffs = [calculate_coefficient(signal, freq) for freq in low_frequencies]
    high_coeffs = [calculate_coefficient(signal, freq) for freq in high_frequencies]
    low_freq = low_frequencies[argmax(low_coeffs)]
    high_freq = high_frequencies[argmax(high_coeffs)]

    print(decode_freqs(low_freq, high_freq), end="")

print()

fig, ax = plt.subplots()
ax.set(ylabel="$y$", xlabel="$t$ (s)")

time = arange(length) / framerate
ax.plot(time, save_data)

fig.savefig(plot_name)
```

Output:

```
0123456789
```

3. Extension