Spring 2021 / 520.214 – Signals & Systems
Project 2
Due: 4/27/2023, 11:59pm

DTMF Decoding

Background

You have been recruited by a law enforcement agency, and one of your tasks is to eavesdrop on phone numbers that people are dialing using built-in microphones on your computer or from pre-recorded audio. The idea -*down the road*- is that once you show your project is working, you can implement it as an app or in hardware then import it onto a portable device and maybe sell it. The spy store sells one for $200 or you call sell it on the app store. Our goal in this project is to first develop a DTMF decoder that works in MATLAB!

What is DTMF technology? It stands for 'Dual Tone Multi-Frequency', also known as "touch-tone". It is used as a signaling method to indicate a phone number without requiring a telephone operator. Any time you press a button on your mobile phone keypad, that button generates a sound that contains two frequencies (as shown in the Figure 1). For example, the 1 key produces a superposition of a 697Hz low tone and a 1209Hz high tone. This combination of frequencies was chosen as it is nearly impossible to reproduce by a human voice.

Figure 1: Frequency encoding of phone keypad.

The specific duration of DTMF sounds varies depending on the system used. For instance, a standard Motorola device uses tones that are 250msec long while other systems can be as short as 20msec. Newer IOS releases on iPhones are also using very short tones. The amplitude of the frequencies can also be variable. Many systems make the high frequencies a little louder than lower-frequencies in order to compensate for the roll-off of voice systems in higher frequency ranges.

In order to achieve our goal of eavesdropping on phone numbers, we will develop a system that gets an audio recording of a phone number being dialed and needs to guess what that number is from the sounds the keypad is producing. The algorithm has to be flexible enough to decode any mobile manufacturer (i.e. any duration and weighting of DTMF tones), any dialing speed (i.e. a sequence of numbers with unknown onset times), and any noisy environment (e.g. we record someone dialing their phone from across a room at a loud party).

Project assignment

The project consists of 4 parts. There are bonus parts that are completely voluntary.

1. You should work on the *project in groups of 2 students* for the main part of the project. All students in the group will get the same grade for the main part. If you chose to work on the bonus section, please do so individually. A separate report section is expected for the bonus part. *Each student* will submit their full report and code (group portion for the main project and bonus portion if you want to).
2. Please submit all your codes as well as a report (in .doc or .pdf format) where you describe your approach, your design decisions, any theoretical formulations, as well as show any graphics and analyses of your code. Do NOT include code as part of your report.
3. Your submission should include your MATLAB functions following the parameters defined in each part of the project. The code submitted should be fully debugged and only use built-in MATLAB functions (no third-party software or special MATLAB toolboxes.) Keep in mind your code will be tested using our own generated sequences. So, it should be fully automated and ready to analyze any unknown sequences of phone numbers. There are some implementations of DTMF available online, but they don't cover many aspects of our project. Make sure you develop your own code. *Plagiarism will be severely punished.*
4. Submit your project files on **canvas**. Each student should submit a copy of their repot.

Part 1 – Encoding DTMF

First, we get ourselves more familiar with DTMF sounds. Using the frequencies given in the figure above, write a function **DTMFencode**.m with the following parameters:

[x,fs] = DTMFencode(key,duration,weight,fs)		
Output	*x*	output sound waveform generated by pressing the corresponding key
	fs	sampling rate in Hz
Input	*key*	a character corresponding to one of twelve possible keys (Figure 1)
	duration	[optional argument] desired duration of the signal (default 200ms)
	weight	[optional argument] 1x2 vector with desired weight of low and high frequency component (default [1 1])
	fs	[optional argument] sampling rate in Hz (default 8000Hz). Note: values below 3000Hz should not be accepted

1.1 Write and submit the code for DTMFencode

1.2 Generate the twelve possible digits with your choice of parameters and save each digits as an audio file named digitX.wav (X=1,2,…)

1.3 Provide in your report a plot of both the time-domain signals and Fourier transform magnitude of each digit. Make sure all axes are labeled properly.

1.4 Describe in your code how you checked that your code is running properly.

Part 2 – Decoding DTMF keys

Next, we focus on decoding an audio signal into the corresponding digit. Here, we *assume our audio recording contains only one digit*. Write a function **DTMFdecode**.m with the following parameters:

[key,fs] = DTMFdecode(filename)		
Output	*key*	a character corresponding to one of twelve possible keys (Figure 1)
	fs	The sampling rate in Hz
Input	*filename*	*.wav file that contains a single DTMF signal

2.1 Write and submit the code for DTMFdecode.
- Your code should work with any sampling rate above 3000Hz, any signal duration and any weighting of low and high frequencies.
- You should think of the best strategy to determine the DTMF values (time-domain, frequency-domain, energy over small frequency bins or a combination).
- Keep in mind that your code for Part 2 will be used in Part 4; it should be able to work well in noisy conditions. You can add other sounds (off the web) to your test signals to generate noisy versions of dialed keys in order to make sure your decoder is robust to noise.
- Generate audio files from Part 1 (with varying parameters) to test your DTMF decoder.

2.2 Describe in your report what strategy or strategies you use to decode the key. Choose for your report an example test-file and plots that describes how your decoder works on this example.

Part 3 – Decoding DTMF sequences

Next, we will deal with parsing a continuous recording of a person dialing a phone number. Here, the unknowns are the onset and offset of each key press as well as how many key presses there are. Write a function DTMFsequence.m with the following characteristics:

[seq,fs] = DTMFsequence(filename)		
Output	*seq*	a string of characters 1 corresponding to one of twelve possible keys (Figure 1)
	fs	The sampling rate in Hz
Input	*filename*	*.wav file that contains a sequence of key presses of unknown length

3.1. Write and submit the code for DTMFsequence
- You should generate a variety of test signals using your code from Part 1.
- To appreciate the range of difficulty in real-life applications, you are provided with a list of 20 real-life recordings recorded in a real office (some of these are noisy and may include additional short sections of dialtone or ringtone). You should also generate your own test signals with varying degree of distortions.

3.2. Use the report to explain your strategy for finding the key presses in the sequence. Use an example and plots (both time and frequency domain with proper axes) to walk us through how an example works.

Part 4 – Decoding DTMF sequences in open environments (This part is optional)

Now, we would like to understand the limits of your algorithm (so we know what settings we can market it for!) How much noise can it tolerate? You are provided a noise recording referred to as babble noise. It corresponds to a recording from a group of people talking at the same time, as would happen in a cafeteria or at a party. If a person is dialing their phone at this party, how loud can the party be before your algorithm stops reliably decoding the phone number? To estimate that, follow these steps:

a. Generate 50 random phone sequences (random numbers and random length) and encode them using your code from Part 1 (with varying DTMF parameters across sequences: frequency weighting and tone duration). You can assume that the weighting and duration is constant *within* each sequence.
b. Normalize each audio signal $s[n]$ to an absolute maximum of 1
c. Normalize the background noise distortion $d[n]$ to an absolute maximum of 1
d. Combine each sequence $s[n]$ with the distortion $d[n]$ using the equation:

$$y[n] = s[n] + \alpha\, d[n]$$

e. Gradually vary the degree of noise (α) from 0 to 3 in steps of 0.1. Obviously, $\alpha = 0$ represents your clean sequence without distortion.
f. For each level of α, decode your sequence and check your error rate (how correctly are you decoding the phone number?).
g. Repeat this procedure 50 times with each of the 50 sequences and provide an average accuracy for each noise level α
h. Include a plot of average accuracy vs. α in your report.
i. Examine a few examples in the time and frequency domain and comment on why you think your algorithm deteriorated at a certain level α.

Part 5 – Bonus section (This part is optional)

5.1. Repeat the procedure in Part 4. Instead of using additive noise $(y[n] = s[n] + \alpha\, d[n])$, we want to analyze effects of room reverberation, i.e. the reflections of the sound through all surfaces in the room. In this case, the distortion can be modeled as a convolution: $y[n] = s[n] * d_r[n]$ where $d_r[n]$ is the impulse response of the room. You are provided a MATLAB function addreverb.m which simulates reverberation with different echo characteristics. The parameter reverbTime in the MATLAB function represents that echo characteristics: the higher that value, the more echo in the room. Repeat the procedure in Part 4, but this time testing accuracy of the algorithm as a function of reverberation time.

5.2. Revisit your analysis of Part 4. Repeat your analysis, but this time estimate how the encoding of DTMF tones affects your accuracy? Do longer or shorter tones give you better accuracy, or is your accuracy independent of tone duration? How about the weighting of low and high frequencies?

Relevant MATLAB functions

help or *doc*	Find out more details about each function (e.g. type help fft)
audioread	Read an audio signal into MATLAB.
audiowrite	save a signal as an audio file
fft	compute the Fourier transform of a signal
abs	take the absolute value of a number (or magnitude, for a complex number)
sound	play an audio signal
plot	Display a signal. Carefully adjust the parameters in order to properly label x and y axes