Enmao Diao ECE 4271
Project 1

Dual-Tone Multi-Frequency (DTMF) Signal Decoding

ALGORITHM

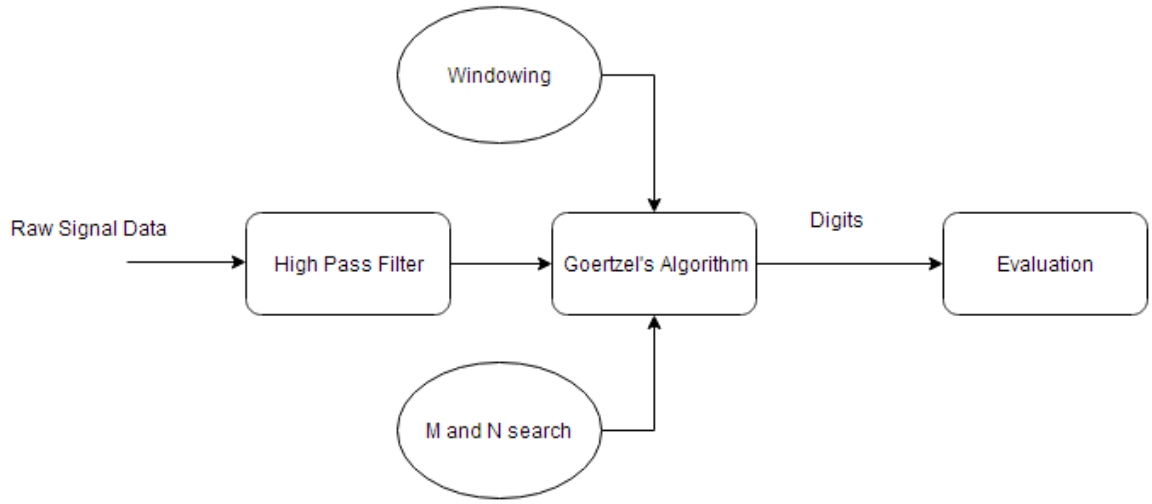

Figure 1. Block diagram of my DTMF signal decoding algorithm

High Pass filter is not my original plan for this project. Initially, I planned to use Band Pass filter to filter only the ranges including these eight tones. Later I realize that since the tone does not have harmonics then there is no need to use band pass filter.

So I directly dig into the DFT problem. There are two important parameters I have to decide. The first one is the data length M that fed into DFT. Due to the smallest tone difference, the frequency resolution has T >= 13.7ms which is M >= 110 samples. Meanwhile, interruptions of no more than 10 ms should not be interpreted as two separate signals, which implies that at least 10 ms (80 samples) interruption needed between digits. Therefore I choose M as 110. Considering that tone duration ranges from 80ms to 400ms, we only need 5 windows to cover the shortest tones.

Next, we need to determine the DFT size N. In order to satisfy the 1.5% and 3.5% accuracy specifications, I use a combination of the approaches discussed in the class. Instead of worrying about the upper bound limitation from 3.5% specs, I allow that DFT samples between the DFT samples corresponding to adjacent DTMF tones. Meanwhile, I also have to maintain the location error lower than 1.5%. Therefore I run a trial code to search optimal choices for N and K. I split the problem into lower band and higher band because the higher band has a larger frequency spacing and therefore it does not require very large N size. The formula is as follows:

Frequency_spacing = FS/N
K = Tone_frequencies/Frequenc_spacing
Location Error = (K*Frequency_spacing-Tone_frequencies)Tone_frequencies < 1.5%
For K values differences greater than 2

I start from N=M to a brute force parameter 500 to bound computation. After that, I found out that I have many N values candidates. For each N value I have four K values with respect of the tone frequencies. It is not surprising that the smallest N candidate for higher band is much smaller than that for the lower band since the tone frequencies are much closer in the lower band. After doing some simple tests with the three test files provided, I found out that even with the lowest N value pairs, all test files passed.

The way I use gfft is very straight forward. Each K and N value pair will run with gfft once. Each operation yields the magnitude of each tone frequency. For either lower frequency group or higher frequency group, I extract the highest magnitude of them and its index to figure out the row and column number. After that we can just map the row and column number to decode the digits.

The way I use windowing is rather interesting. For each window, I retrieve the maximum magnitude of both frequency groups and its index. I also store the index values of the previous window so that I could count the number of window which has the same index value in a row. Since each tone does not vary its own frequency, there should be at least 5 windows in a row to cover each tone. Therefore when the counter reaches the window_per_tone parameter, I would output one digit. If the next window is the same tone, then the counter grows but nothing else changes. If the next window is different, then the counter resumes to zero. This approach, although seems naïve, works extremely well if the tone length is not very small, for example 40 ms. As we know that noise is random and for the majority of time it cannot last 5 windows with exactly the same index result, otherwise it is definitely not noise. Since the result varies, the counter can never reach up to the threshold to output a digit. In the "quiet" session, there are actually many noises and this scheme works just as expected. If the tone duration is small, then we have to shrink the data size M to have enough number of windows to keep it away from noises. However, we cannot have M smaller than 80 otherwise interruptions are not taken into account. One way to deal with the shorter tone duration is to overlapping the data samples and windowing with smaller size. We do not actually need this for this project as the shortest duration is 80ms and it actually yields very accurate result.

Then, I build an evaluation system by modifying tt_create.m file. In the test.m file, it runs 1000 iterations to test the error rate of my algorithm. This file uses My_tt_create.m file to generate 8 signal data with different dB levels from 50 dB to 0 dB.

After several trials, I found out that the biggest issue of my algorithm is that the dial tone keeps

interfering the first few digits. At this point the error rate is only 0.3%. From the references, we know that dial tone along with other telephone tones have frequencies no more than 620 Hz. Therefore it adds up to the magnitude of the smallest tone frequency 697 Hz. To fix that, I implement a Cheby1 High Pass filter with passband frequency at 650 Hz. This helps dropping the error rate to 0.05%.

Finally, I test the DFT size N in terms of error rate. I found out even with the lowest N value pair, the error rate is still around 0.1%. I just choose the values in the middle to represent a trade of between computation and accuracy. The values I choose are as follows:

N_lower = 387, N_higher = 329
K_lower = [34,37,41,46], K_higher = [50,55,61,67]

FILE DESCRIPTION

./EDiao

- /Project Report.pdf
- tt.decode.m % Main algorithm, dependent on gfft.m
- My_tt_create.m % Generate test signals
- test.m % Test and report error rate
- gfft.m % Goertzel's Algorithm