U N I V E R S I T Y O F C A P E T O W N

Department of Electrical Engineering

EEE4114F – Digital Signal Processing

Dual Tone Multiple Frequency Decoder

Project 2019 Report

Moegamad Ameer Conrad – CNRMOE001

Tareeq Saley – SLYTAR001

13/05/2019

Contents

Aim

The aim of the project is to design and implement a DTMF decoder in software, using MATLAB, and investigate the effect that additive Gaussian white noise and background speech have on the decoder?

Introduction

Dual tone multiple frequency (DTMF) tones are used to represent the digits on the keypad of touch tone phones. Each key on the keypad is represented by a tone which is the combination of two superimposed sinusoidal signals of two different frequency groups (high frequencies and low frequencies) [1]. Each frequency represents either a row or a column on the keypad (as shown in Appendix A) such that each key contains one sinusoid of a frequency from the group of low frequencies and one from the group of frequencies. Low frequencies represent rows and High frequencies represent columns. Some keypads, like the one in Appendix A, have four rows and four columns. However, in this project we will be considering a more traditional keypad with four rows and three columns.

Problem Question

Can digital signal processing be used to detect which button on a touch tone keypad was pressed, from the DTMF signal heard? And what are the effects of noise on the DTMF signal?

Literature Review

The purpose of this literature review is to outline the underlying theory used in this project to develop a dual tone multiple frequency (DTMF) decoder.

ITU (International Telecommunications Union) standard frequency recommendations are 697, 770, 852 and 941 Hz for the low frequency group and 1209, 1336, 1477 and 1633 Hz for the high frequency group [2] (note that we will not be using 1633 Hz since our keypad only has three columns). These frequencies are chosen because they have certain characteristics. They are chosen to allow the signals to be able to pass through telephone lines easily. These frequencies also make it easier to filter the superimposed signals [3]. Also, all the frequencies are in the range that humans are able to hear, none of the frequencies is a multiple of the others and none of the frequencies equals the sum or difference of any of the others. These properties of the DTMF reduce the number of false detections of DTMF tones, when decoding. These specific characteristics of DTMF frequencies also allow DTMF receivers to detect when more than one button is pressed and detect when harmonic energy is present and then reject any tones that contain harmonic energy, such as human speech and noise. Since DTMF signals only contain fundamental tones, if a second harmonic is detected, the signal is not a DTMF tone and can be rejected. Improper detection of DTMF tones due to the presence of speech or noise or both, is known as talk-off error [1].

A standard sampling frequency for DTMF decoders is 8 kHz, for which the best length Discrete Fourier Transform (DFT) has been found to be one of length N=205 [3]. The DFT and Fast Fourier Transform (FFT) are used to extract the frequency information from the given DTMF signal. However, DFT's with high values of N, requires large amounts of computational power. As a result, one of the more popular methods of extracting frequency information from DTMF tones is using the Goertzel Algorithm [1][3].

In this project we will not be implementing the Goertzel Algorithm, as we will be implementing the DTMF decoder in software only, using MATLAB, which means that we will not have the issue of computational power that we would have if we implemented the decoder on an 8-bit microprocessor.

Method and Code Generation

The programming language which has been used to design, simulate and implement the DTMF decoder is MATLAB. The purpose of a DTMF decoder is to compute DFT samples which are closest in frequency to the seven fundamental DTMF tones and their second harmonics. The following list identifies the steps taken towards designing the decoder:

1. Encoding

In order to facilitate DTMF decoding, the respective seven DTMF tones first needed to be developed and encoded. The program encodes data based on a sequence of digits entered by the user. The frequencies that were used for each respective digit corresponds with the standard consumer DTMF keypad and are given below:

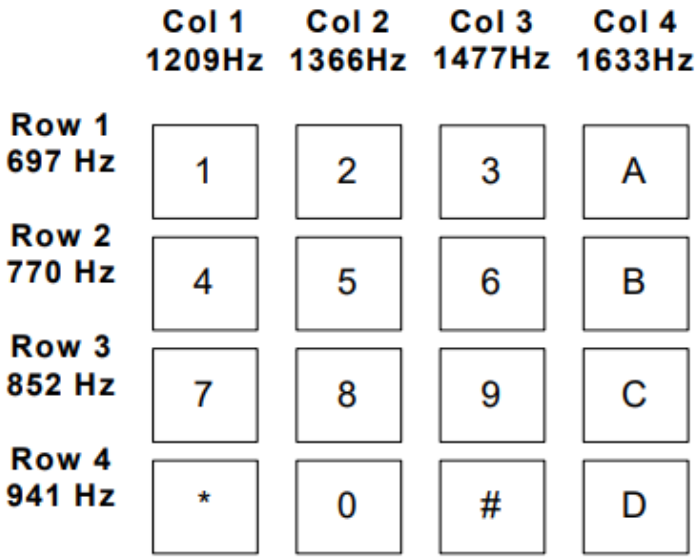

Figure 1: DTF frequency map of touch tone keypad

In order to encode data based on user generated inputs, a MATLAB inbuilt function was used to detect if a button is pressed. If a button happened to be pressed on the keyboard, the program would then find the ASCII code corresponding to that respective button and use it as an input parameter for the developed 'encode' function.

The MATLAB function used to implement the encoding takes in a digit represented by its ASCII code and maps the digit with its respective low frequency and high frequency components as suggested by the table above. Two sinusoidal wave signals with arbitrary amplitudes are then generated based on these frequencies. The function returns a variable which represents the sum of those sin waves, after which that same variable is tested through MATLAB to see if the correspond sound that it produces matches up with the sound that it's expected to produce. A sampling frequency of 8 kHz was used for this implementation in order to stay within a safe area of the Nyquist criteria.

The minimum duration of a DTMF signal defined by the ITU standard is 40 ms [2]. Therefore, there are at most 0.04 x 8000 = 320 samples available for estimation and detection. The DTMF decoder needs

to estimate the frequencies contained in these short signals. However, even though having a larger number of samples provides higher resolution in the frequency domain, it takes more computation time and therefore results in tone distortion when a user enters a sequence of digits. For an 8-kHz sampling frequency, the best value of the DFT length N to detect the seven fundamental DTMF tones has been found to be 205. This corresponds to 25.625 ms in the code.

2. Filtering and Data Gathering

As part of the decoding process, the first step was to take the signal representing the entered digit and detect its high frequency and low frequency components. A low pass filter was used to detect the low frequency component and a high pass filter was used to detect the high frequency component. These filters were designed using MATLAB's filterDesigner tool. Given that the difference between the DTMF high frequency and low frequency components is 268Hz, with half of that being 134Hz, the respective stopband and passband frequencies for the low pass filter (941Hz + 134Hz) and high pass filter (1209Hz - 134Hz) was chosen to be 1075Hz. The designed filters were then exported to the MATLAB workspace and used to detect the low and high frequencies of the signal.

After obtaining the plots representing the FFT's of the respective filter results, MATLAB's 'max' function was used to find the maximum value of each resulting graph and its 'find' function was used to find the corresponding frequency (represented on the x-axis) of that maximum value. The frequency of the maximum value in each graph would correspond to one of the two DTMF fundamental tones of the user entered digit. The variables associated with the low frequency and high frequency values are labelled as 'ol' and 'oh' respectively in the code.

3. Decoding

The function that was created to decode the signal takes in variables 'ol' and 'oh' as its input parameters. It then compares these variables to the threshold frequency values associated with the DTMF consumer keypad. These threshold values are, however, not identical to the keypad values because of the trade-off related to frequency resolution and computational time as explained earlier. The function displayed a message indicating what key was pressed in order to affirm that the decoder works as intended.

4. Keypress Sequencing and Noise Addition

To gauge the impact that a sequence of key presses has on the results, and to see if the program can correctly decode a sequence of digits, a 'for' loop was implemented in the code which allows the user to enter 12 digits. 12 digits were then entered with durations of different lengths between them and the effects of those entries were then used for analysis.

Random white noise was also added to the program. This noise was introduced into the program with varying SNR's and the effect that the noise and its respective SNR had on the accuracy of the results was used for analysis.

5. Flowchart Representation of Decoder Code

Actual code snippets for the encoder and decoder are included in the Appendices.

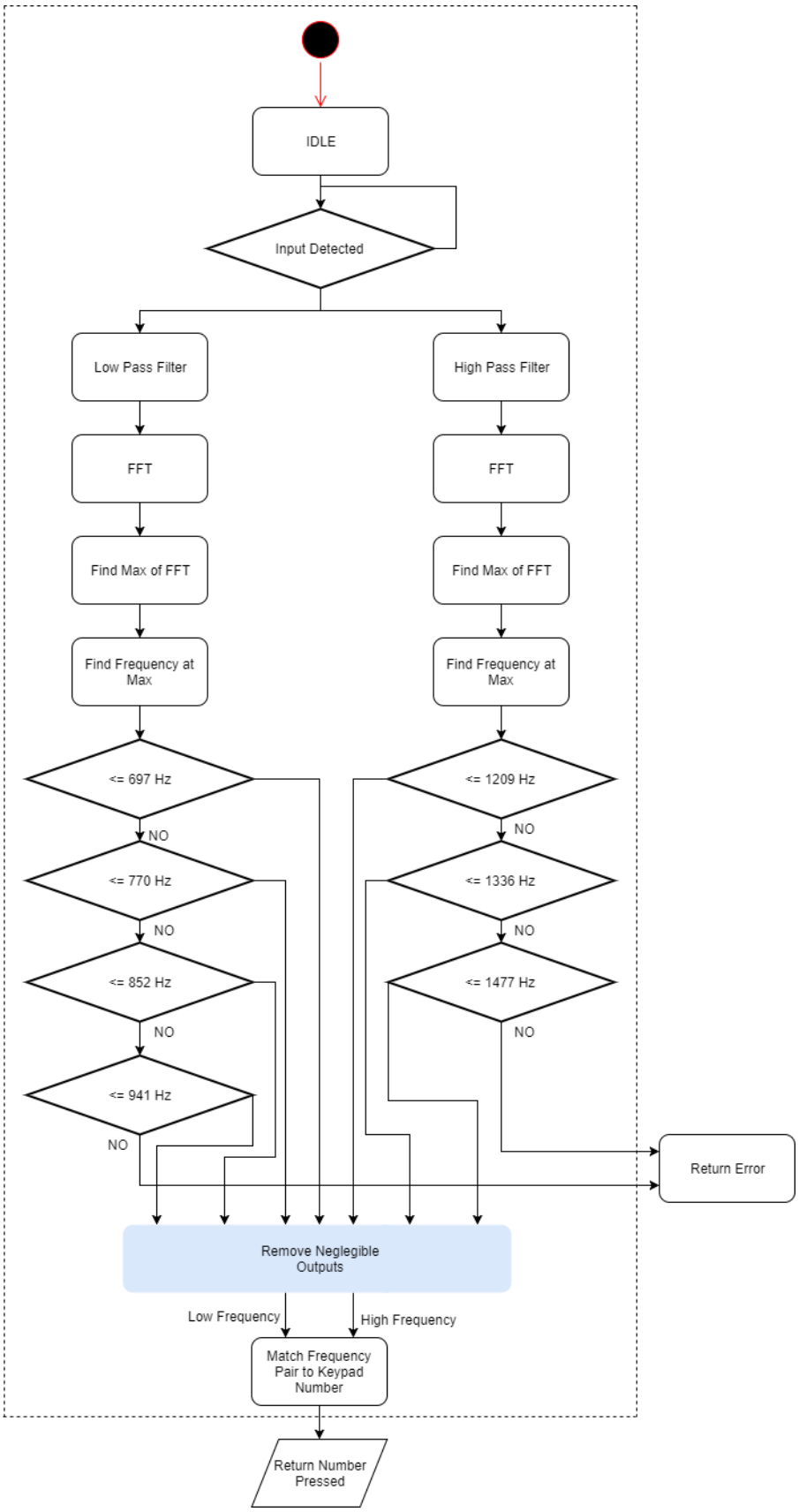

Figure 2: Flow Chart of Decoding Process

Results and Testing

Once we acquired conclusive results that our decoder worked for the pure DTMF tones we generated using MATLAB, we tested its robustness by trying to decode a pure DTMF tone with additive gaussian white noise and also testing a DTMF tone recorded from an actual cell phone in an environment with ambient noise (including talking).

DTFM Decoder Analysis

The DTFM decoder, without the addition of gaussian or ambient noise, works as expected and gives the correct result 100% of the time. It does so regardless of the length of the sequence and inter-tone spacing between key presses. It is virtually impossible to press two digits simultaneously on the keyboard as the program always manage to detect the key that's pressed first and hence the effect of two digits being pressed simultaneously could not be detected.

White Gaussian Noise Addition Analysis

An analysis was performed to determine the effects that additive white gaussian noise has on the decoder and whether or not it significantly affected the accuracy of the decoder result. Different SNR ratios were used for analysis, and each digit was pressed three times for each respective SNR ratio. The following results were obtained

Table 1: Table of results for additive gaussian white noise

SNR RATIO	NO. CORRECT RESULT	NO. INCORRECT RESULT	INCORRECT DATA	% Accuracy
8	34	2	5,8	94.44%
7	33	3	5,8,0	91.67%
5	33	3	2,5,8	91.67%
5	32	4	2,8,8,#	88.89%
4	28	8	2,4,5,5,8,0,0,#	77.78%
3	31	5	2,6,8,8,0	86.11%
2	26	10	2,2,4,4,5,5,6,8,0,*	72.22%
1	24	12	2,2,2,3,3,5,6,7,8,8,0,*	66.66%

Table 1 above indicates that the most common digits which are usually mistaken for other digits are the middle most digits, 5 and 8, in the standard consumer DTMF keypad. This is expected as these two digits have neighbouring digits that have frequencies which are both lower and higher than their fundamental tone frequency values and have the most neighbouring digits around them (4).

The reason why noise causes inaccuracy in the decoding process is because the noise distorts the detection of the fundamental tones in two ways; it either shifts the first harmonic a little to the right in the frequency domain and as such confuses the respective digit with its neighbouring digit to the right of it (figure 1), or the noise is so significant that it creates a new first harmonic altogether somewhere other than where the first harmonic of the unaltered signal is (figure 2). The following two plots highlight these effects for when digit '5' is pressed. The blue line represents the FFT of the unaltered signal and the orange line represents the FFT of the signal with noise added to it.

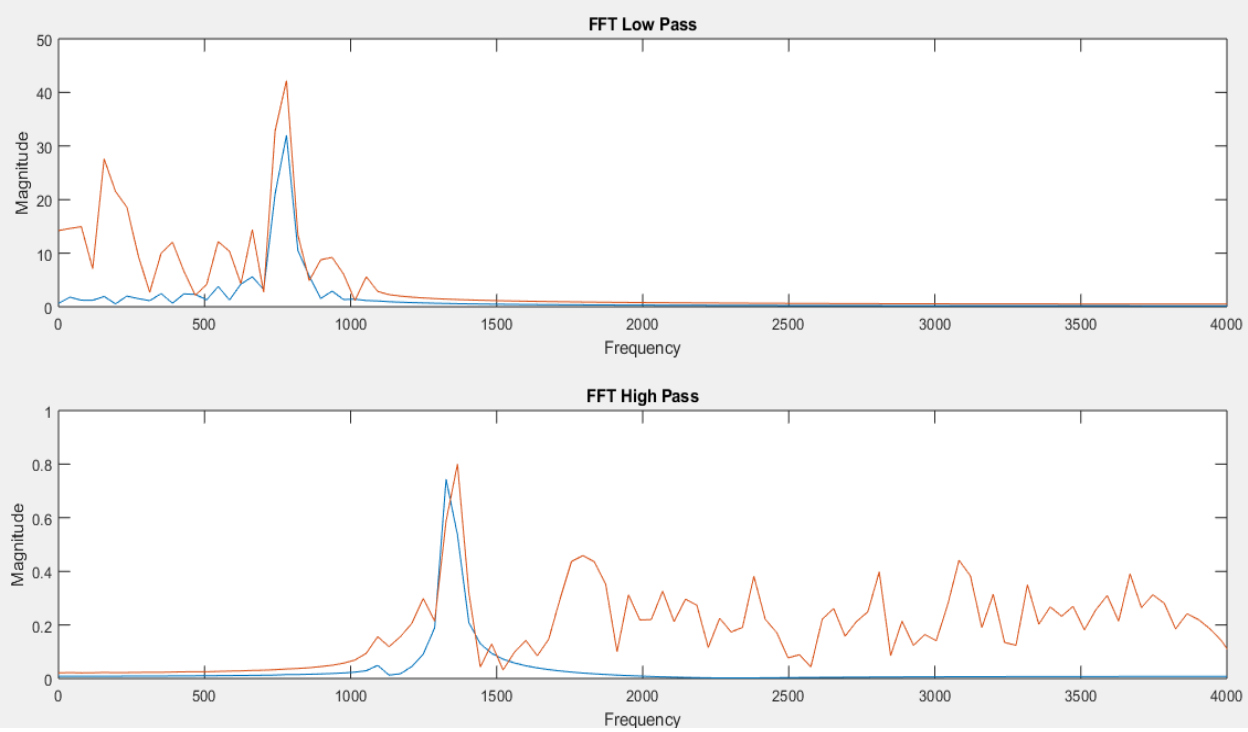

Figure 3: Noisy signal with the high fundamental frequency shifted to the right, thus causing the decoder to confuse 5 with 6.

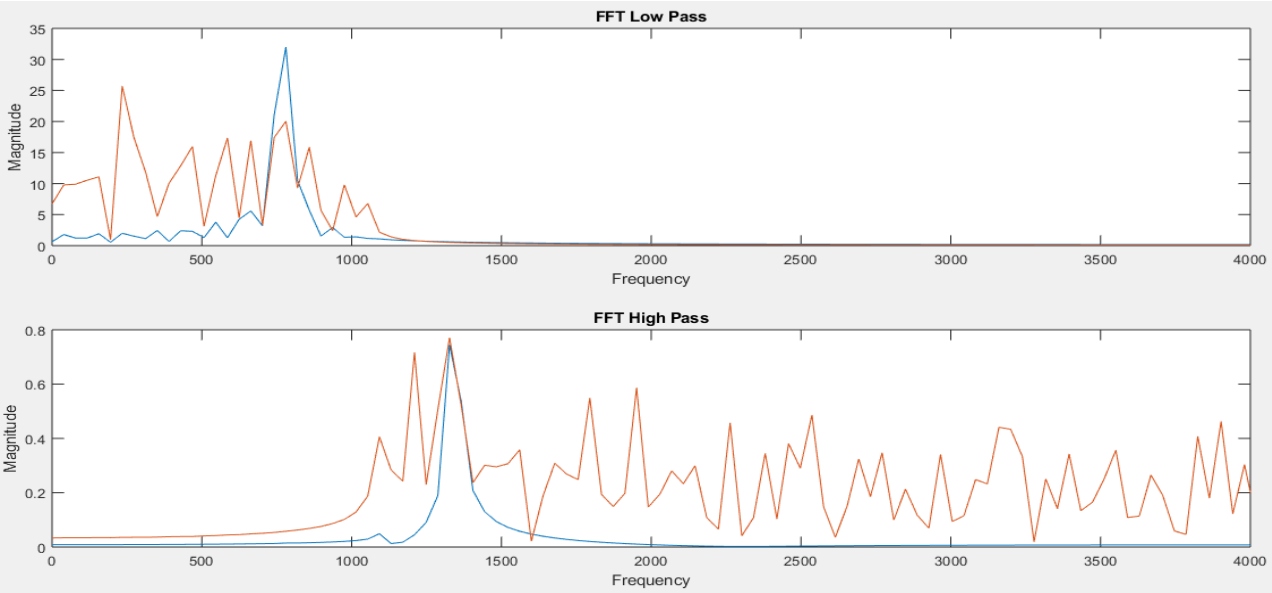

Figure 4: Noisy signal with the low fundamental frequency assuming a completely random location in the frequency domain, in this case causing the decoder to confuse 5 with 2.

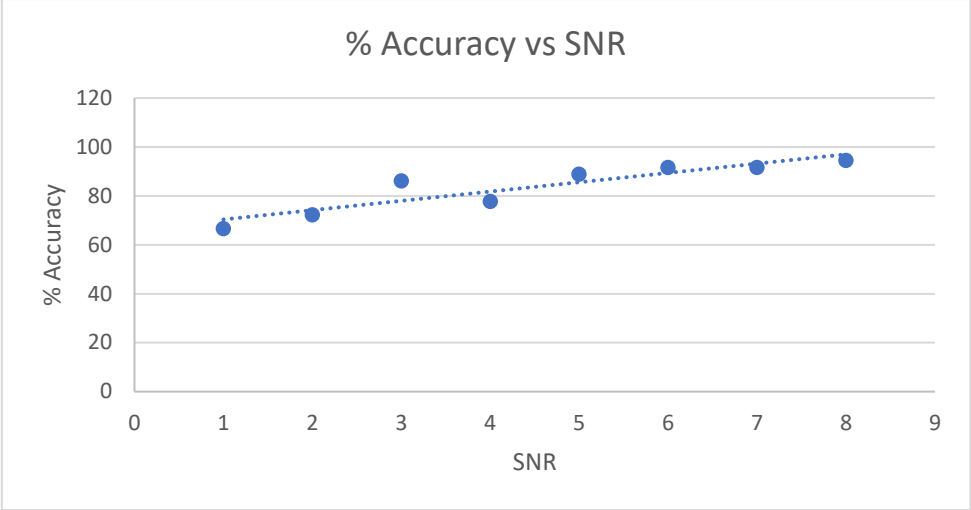

Figure 5: Plot of Accuracy vs Signal to Noise Ratio

According to Figure 5 above, there is a direct correlation between the SNR and the % accuracy. The accuracy of the results is directly proportional to the signal-to-noise ratio.

Ambient Noise Analysis

The DTMF sound clip is recorded at a sampling frequency of 44.1 kHz. This recorded signal is then down sampled to 8 kHz to be passed into the designed decoder. Recording was done using the stereo audio recording on a cell phone. Recording the audio in stereo means that there are two audio channels.

It was noticed that, when decoding the signal from channel two (right side), the decoder decoded the signal correctly, 100% of the time, and when plotting the channel two recorded tone signal and the MATLAB generated tone signal side by side (Figure 6 below), it can be seen that they are almost identical. Therefore, it makes sense that the signal is always decoded correctly.

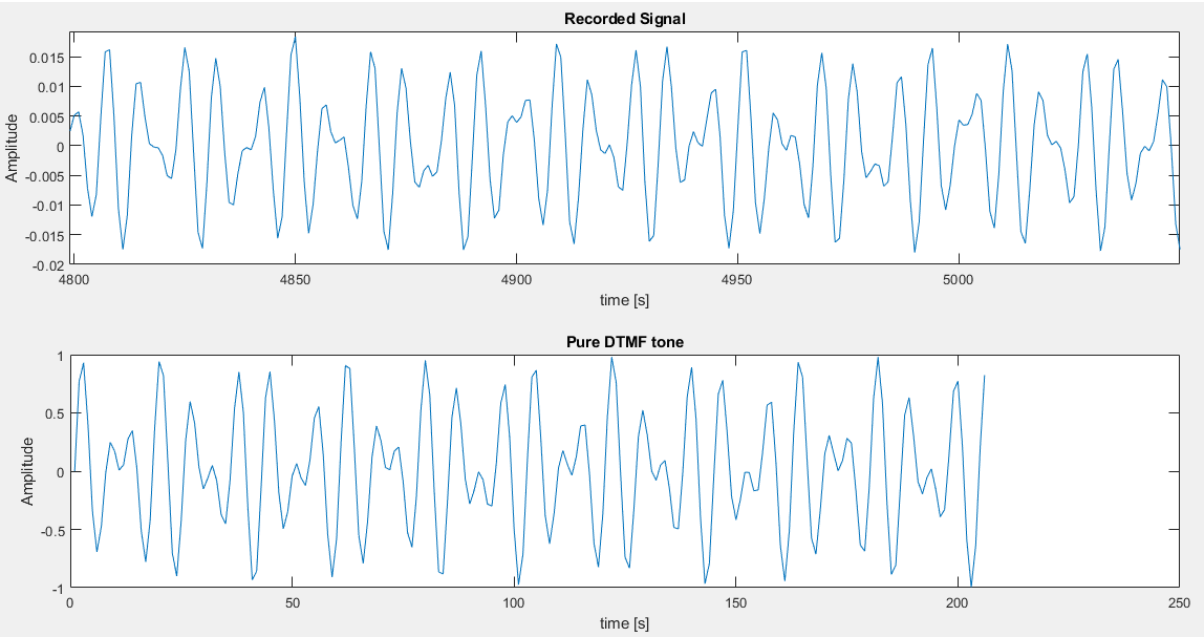

Figure 6: Channel 2 Recorded Signal for zero key and MATLAB generated DTMF tone for zero key

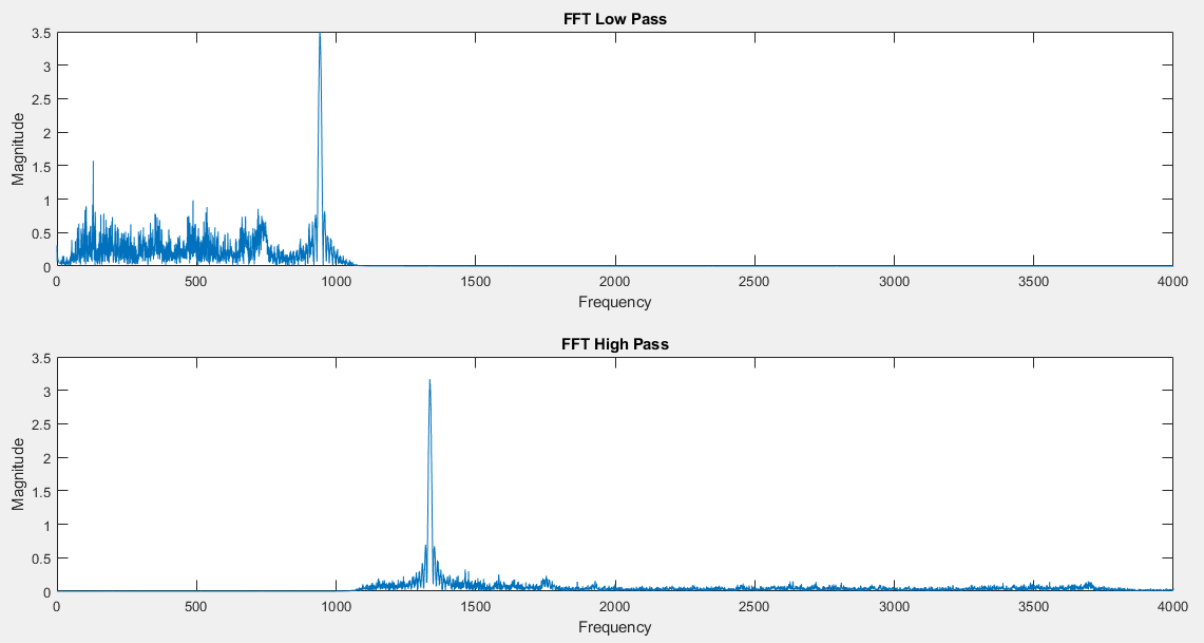

Figure 7: FFT of Recorded zero DTMF tone (Channel 2)

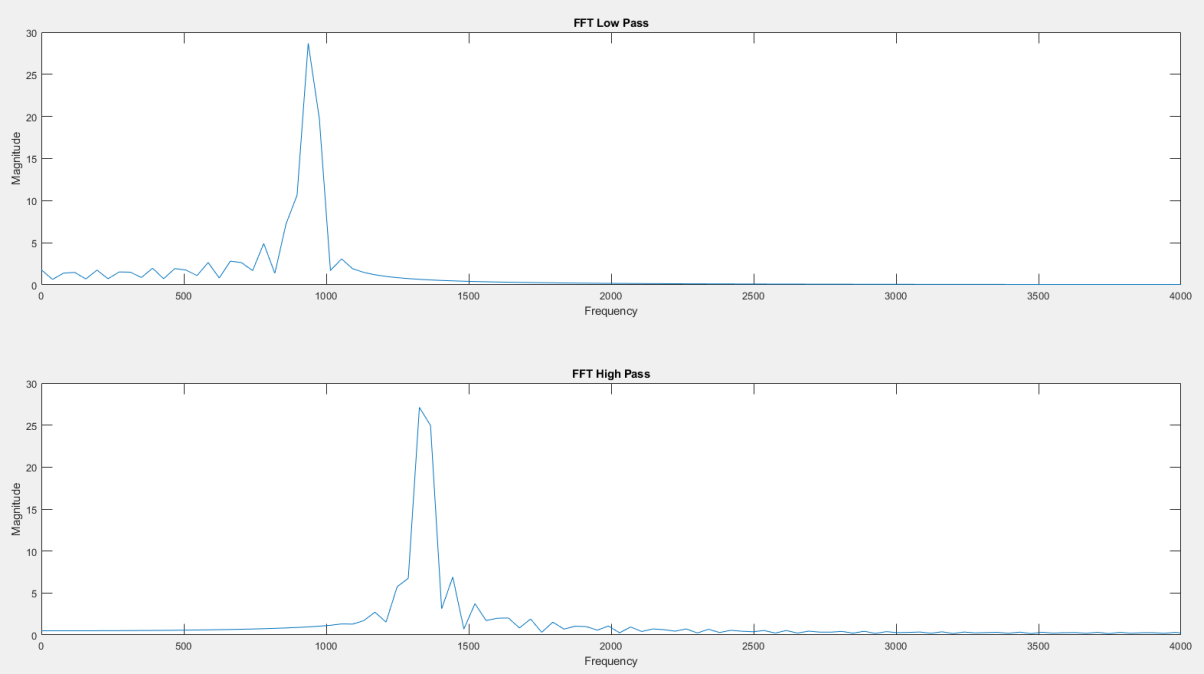

Figure 8: FFT of MATLAB Generated zero DTMF tone

Figure 7 and 8 above shows the maximum values of the LPF FFT plots and the HPF FFT plots of the recorded tone to be roughly at the same frequency as that of the MATLAB generated tone, which again supports our observation that they both return that the Zero key has been pressed.

It was also noticed that, when decoding the signal from channel one, the decoder always decoded the high frequency component correctly but always decoded the low frequency to be the lowest frequency in each respective column for that column. For example, when

inputting the recorded channel one DTMF tone for 2, 5, 8 and 0, the decoder decodes them all to be 2. The same is true for each of the columns.

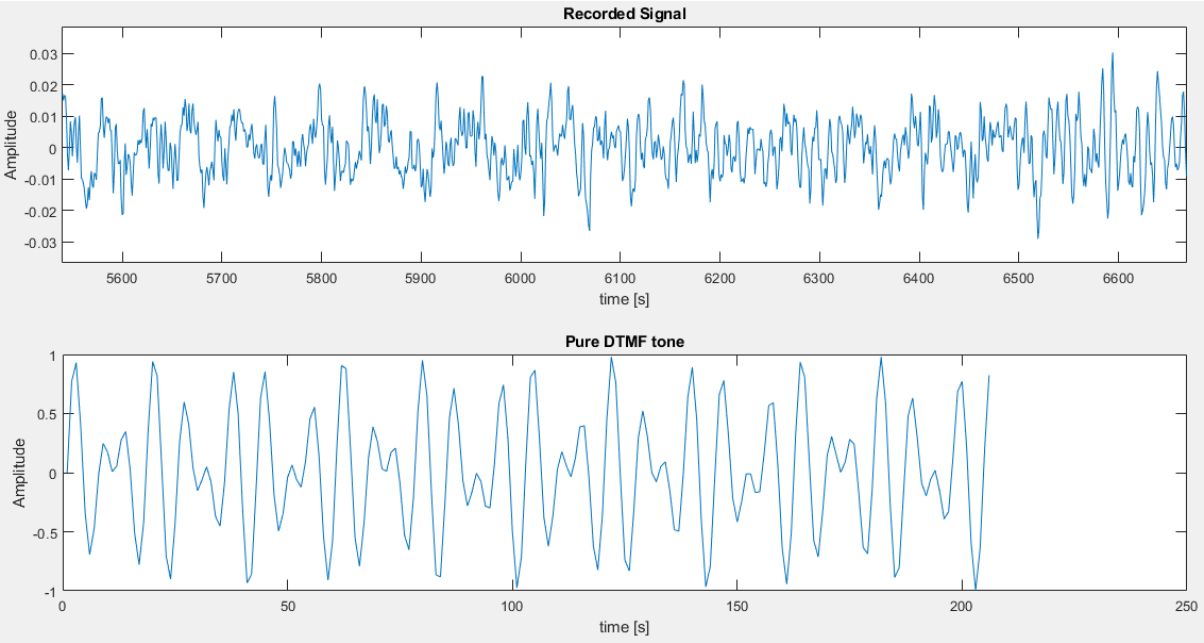

Figure 9: Channel 1 Recorded Signal for zero key and MATLAB generated DTMF tone for zero key

It is clear, from Figure 6 above, that the recorded tone is much different to the MATLAB generated tone.

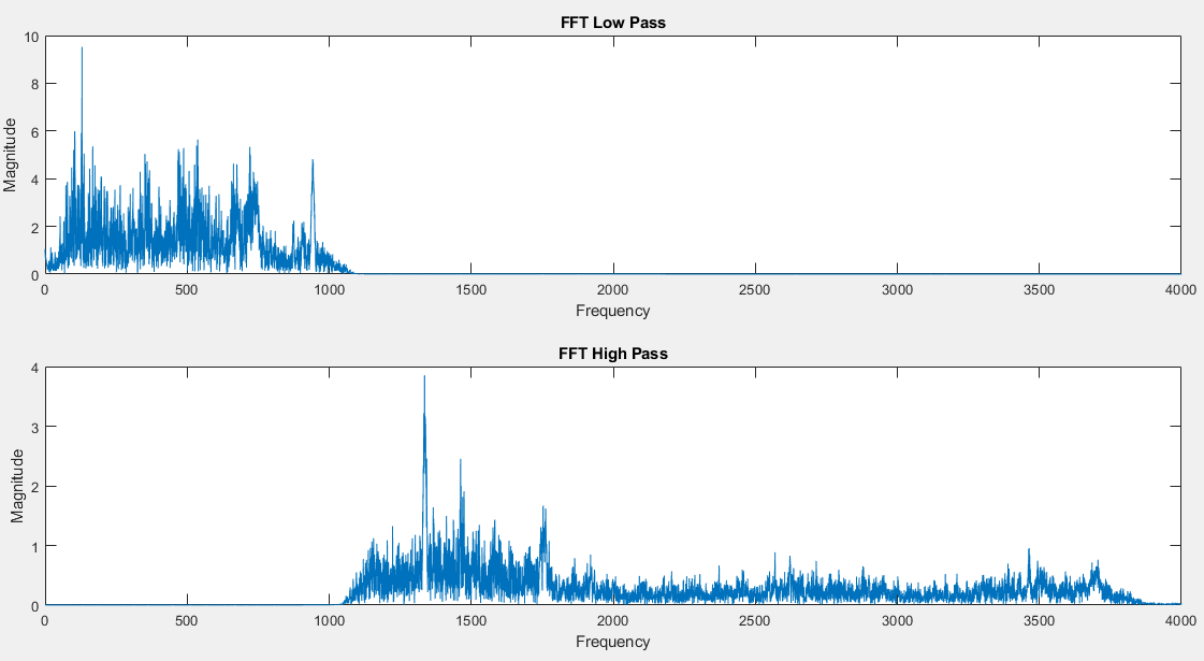

Figure 10: FFT of Recorded zero DTMF tone (Channel 1)

When comparing Figure 7 to Figure 5 it can be seen that, while the FFT is a lot noisier, the maximum value of the FFT of the high frequency signal is still in the right place and while there is a maximum point at the right frequency of the FFT of the low frequency signal, it is not the absolute maximum.

This means that the ambient noise introduced a frequency component to the recorded tone that was greater in magnitude than that of the DTMF tone and at a lower frequency than any of the DTMF tone frequencies. As a result of this the DTMF decoder decodes key presses of 1, 2 and 3 correctly 100% of the time but decodes all other keys incorrectly 100% of the time

The inconsistencies between the two stereo recording channels could be due to the direction of the stereo microphones relative to the directions of the incoming sounds (DTMF tone and ambient noise). If one sound was originating closer to one of the microphones it would be more prominent in that channel signal (a plot of the two channels for the recorded tone of zero is included in Appendix C).

Improvements

The scope of the project could have been increased to create a DTMF decoder that works most of the time is not 100% of the time, including in the real environment where the signal being decoded will not be a pure DTMF tone (noise-free). As discussed in the literature review at the beginning of this project, this can be done by implementing a prefilter or prefilters that detect whether a signal component contains a second harmonic, which would mean that it is speech and not a DTMF tone, since DTMF tones only contain the fundamental tones.

Conclusion

To answer the initial problem question, it is possible to using digital signal processing techniques to decode a DTMF signal and additive white gaussian noise and ambient noise both have a negative effect on the accuracy of the decoder.

In conclusion DTMF tones can be decoded using various signal processing techniques including, most importantly, filtering and down sampling and noise does in fact negatively affect the DTMF decoder's ability to accurately decode a DTMF tone. The extent of the noise's effect is proportional to the signal to noise ratio. While not included in the scope of this project there are ways to greatly decrease this effect of noise. One of the most common methods, as discussed in the literature review and Improvements section above, is to filter out any tones which include a second harmonic.

References

[1] S. Laboritories, *DTMF DECODER REFERENCE DESIGN,* Austin, 2005.

[2] I. T. Union, “ITU-T Recommendation Q.24,” in *Fascicle VI.1*, 1993, p. 3.

[3] A. Zhang and H. Shao, “DTMF Decoder,” Department of Electrical Engineering Columbia University, New York.

Appendices

Appendix A: Encoder Function

```
function [y, noisy] = encode(number)

 switch(number)
    case 49
        f1 = 697; f2 = 1209;
    case 50
        f1 = 697; f2 = 1336;
    case 51
        f1 = 697; f2 = 1477;
    case 52
        f1 = 770; f2 = 1209;
    case 53
        f1 = 770; f2 = 1336;
    case 54
        f1 = 770; f2 = 1477;
    case 55
        f1 = 852; f2 = 1209;
    case 56
        f1 = 852; f2 = 1336;
    case 57
        f1 = 852; f2 = 1477;
    case 42
        f1 = 941; f2 = 1209;
    case 48
        f1 = 941; f2 = 1336;
    case 35
        f1 = 941; f2 = 1477;
    otherwise
        display("wtf?");
end
t=[0:0.000125:0.025625];
fs=8000;
y1=.5*sin(2*pi*f1*t);
y2=.5*sin(2*pi*f2*t);
y=y1+y2;
noisy = awgn(y,1.5);
soundsc(noisy,fs)
%end
end
```

Appendix B: Decoder Function

```
function decode (ol,oh)
 if (ol <= 737.9 & oh <= 1242.8)
     display ("Key Pressed is 1");
         elseif (ol <= 737.9 & oh <= 1359.3);
     display ("Key Pressed is 2");
         elseif (ol <= 737.9 & oh <= 1514.7);
     display ("Key Pressed is 3");
         elseif (ol <= 815.6 & oh <= 1242.8);
     display ("Key Pressed is 4");
         elseif (ol <= 815.6 & oh <= 1359.3);
     display ("Key Pressed is 5");
         elseif (ol <= 815.6 & oh <= 1514.7);
     display ("Key Pressed is 6");
         elseif (ol <= 893.3 & oh <= 1242.8);
     display ("Key Pressed is 7");
         elseif (ol <= 893.3 & oh <= 1359.3);
     display ("Key Pressed is 8");
         elseif (ol <= 893.3 & oh <= 1514.7);
     display ("Key Pressed is 9");
         elseif (ol <= 970.9 & oh <= 1242.8);
     display ("Key Pressed is *");
         elseif (ol <= 970.9 & oh <= 1359.3);
     display ("Key Pressed is 0");
         elseif (ol <= 970.9 && oh <= 1514.7);
     display ("Key Pressed is #");
 end

end
```

Appendix C: Channel 1 vs Channel 2 for DTMF Tone of Zero