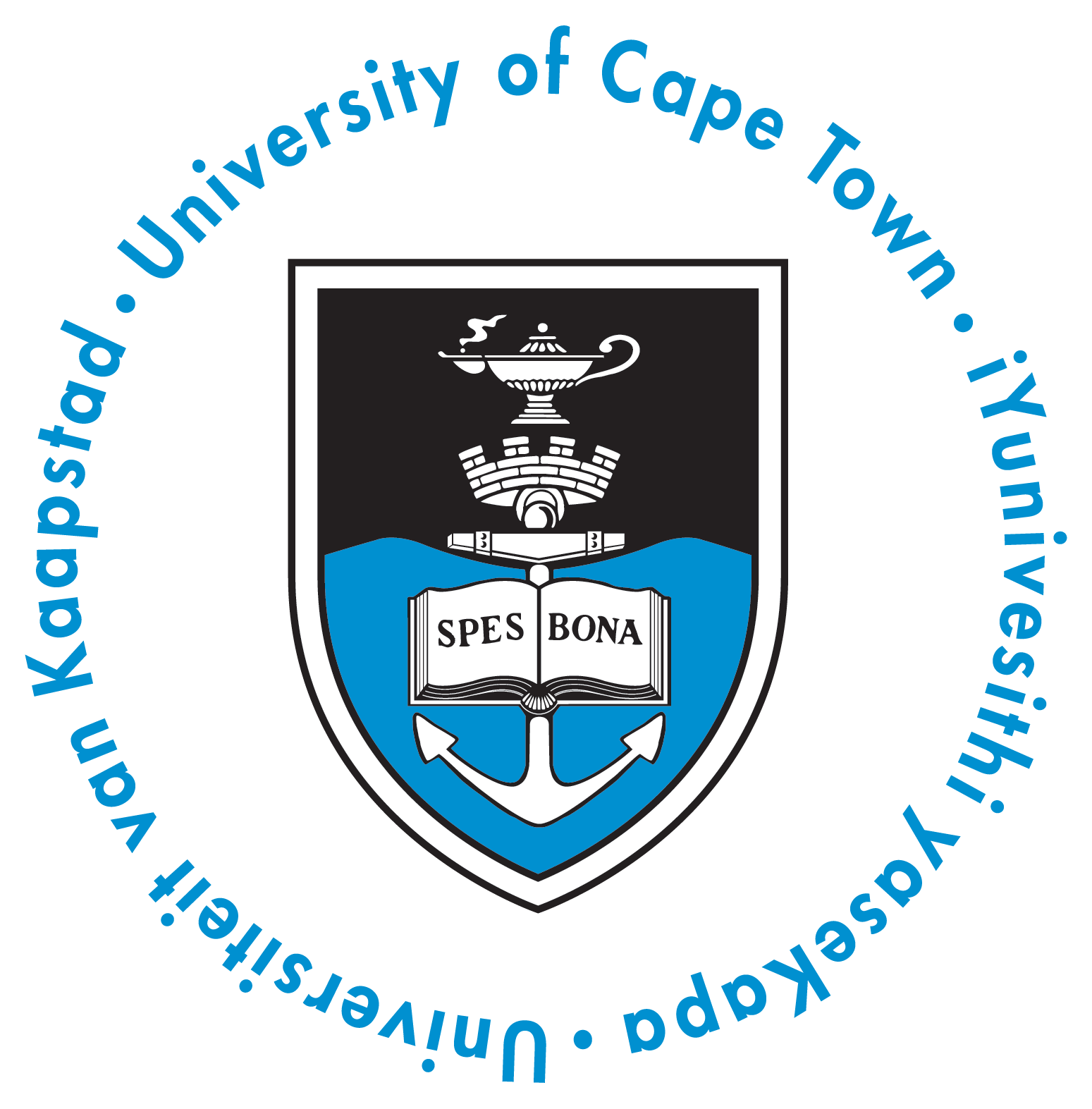
**EEE4114F: DSP Project Report**

**DTMF Decoder**

Andrew Olivier OLVAND008

Hannes Beukes BKSJOH009  
University of Cape Town

April 3, 2018



**Contents**

[1. Introduction 3](#_Toc512252924)

[2. Literature Review 4](#_Toc512252925)

[2.1 Background of DTMF Decoders 4](#_Toc512252926)

[2.2 Goertzel’s Algorithm Method 4](#_Toc512252927)

[2.3 ITU Recommendations 5](#_Toc512252928)

[2.4 Goerzel’s Algorithm 5](#_Toc512252929)

[3.2.1 Precomputed constants: 6](#_Toc512252930)

[4. Implementation of Goertzel’s Algorithm in MATLAB 7](#_Toc512252931)

[4.1 Software Implementation 7](#_Toc512252932)

[5. Testing and Analysis 9](#_Toc512252933)

[5.1 Plotting of Individual Tones 9](#_Toc512252934)

[5.2 Table of Decoded Tones 11](#_Toc512252935)

[6. Conclusion 12](#_Toc512252936)

[6.1 Review of Work Completed 12](#_Toc512252937)

[6.2 Possible Improvements to the Experiment 12](#_Toc512252938)

[7. References 13](#_Toc512252939)

[Appendix A: Matlab Code 14](#_Toc512252940)

[DTMF\_Decoder.m 14](#_Toc512252941)

[createBlocks.m 14](#_Toc512252942)

[DectectTones.m 14](#_Toc512252943)

[getValues.m 15](#_Toc512252944)

# **1. Introduction**

Dual-tone Multi-Frequency otherwise known as DTMF signals are a standard in telecommunications systems. They correspond to touch tone telephones as well as various keystroke-based systems. This is where keystrokes from a keypad are translated to dual tone signals. The DTMF dialing system traces its roots to a technique AT&T developed in the 1950s called MF (Multi-Frequency) which was deployed within the AT&T telephone network to direct calls between switching facilities using in-band signaling. In the early 1960s, a derivative technique was offered by AT&T as a "modern" to place calls. The consumer product was marketed by AT&T under the registered trade name Touch-Tone. Other vendors of compatible telephone equipment used other names such as "Tel-Touch."

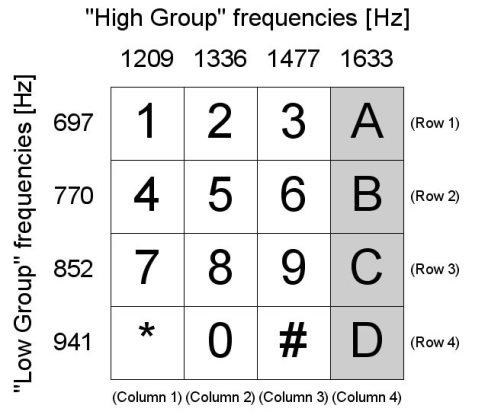
DTMF is used to represent 16 keys ranging from 0-9 and including \*, # and A, B, C and D. Each of these keys are represented by 2 frequencies. Each coming from a separate bin one which holds the lower frequencies and the other the higher frequencies. The combination of the two frequencies produce a unique tone that cannot be imitated.

Figure 1: Image Representing the 16 DTMF keys and the high group and low group frequencies associated with each of them. [4]

The aim of this project is to produce a DTMF decoder with the ability to take in various input wave files with varying levels of noise and use of different tones. The type of DTMF decoder that was chosen to be implemented was the Goertzel’s Algorithm Method.

# **2. Literature Review**

## **2.1 Background of DTMF Decoders**

DTMF detection is the process whereby DTMF signals are identified in sound waveforms in the presence of noise, speech and various dial tones. Upon research was identified that the majority of implementations of DTMF decoders, use the Goertzel's Algorithm Method of DTMF decoding. The Goertzel Algorithm method will be expanded upon further below.

## **2.2 Goertzel’s Algorithm Method**

Goertzel’s algorithm was first described by Gerald Goertzel in 1958. The algorithm is a technique in Digital Signal Processing (DSP) that provides an efficient means to evaluate individual terms of the Discrete Fourier transform (DFT) [7]. Goertzel’s algorithm reduces the number of real-valued multiplications by almost a factor of two relative to direct computation via the DFT calculation. This makes it particularly useful in applications such DTMF decoding.

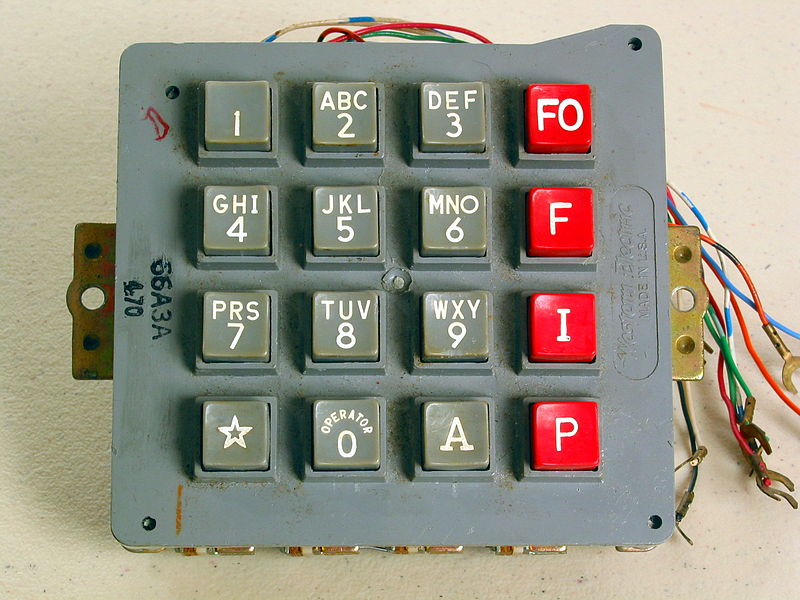


Figure 2: Example of a keypad that produces DTMF tones. [1]

## **2.3 ITU Recommendations**

When looking to build the DTMF decoder the first step was to look at the guidelines that are defined by the ITU [1] and ITU-I Recommendations [2], as these recommendations are defined for the use of DTMF signals and therefore any DTMF decoder should be able to decode any signal adhering to the recommendations set out by these documents. These recommendations entail:

**Signal Frequencies:**

* Low Band (Hz): 697, 770, 852, 941
* High Band (Hz): 1209, 1336, 1477, 1633

**Frequency Tolerances:**

* Maximum accepted frequency offset: <=1.5% to 1.8%
* Minimum rejected frequency offset: >= 3.5% to 7.5%

**Power Levels per Frequency:**

* Power range: -27dBm to 0dBm
* Maximum Power difference between the two frequencies: 5 to 10 dBm

**Signal Timing:**

* Minimum accepted tone duration: 40ms
* Minimum pause (silence between tones): 30ms to 70ms
* Maximum rejected tone duration: 20ms
* Maximum signal interruption: 10ms

The test data that was used to test the implementation of our DTMF decoder was taken from sound examples site recommended by the description of the project these .wav files included the full range of 16 available DTMF characters. All of these example sound files adhered to the recommendations stated above.

## **2.4 Goerzel’s Algorithm**

When implementing the Goertzel Algorithm, some preliminary calculation must be completed:

* Decide on the sampling rate.
* Choose the block size, **N**.
* Precompute one cosine and one sine term.
* Precompute one coefficient.

**Sampling rate:** The sampling rate, in our case was predetermined to be 8kHz, which is standard for telecommunication applications.

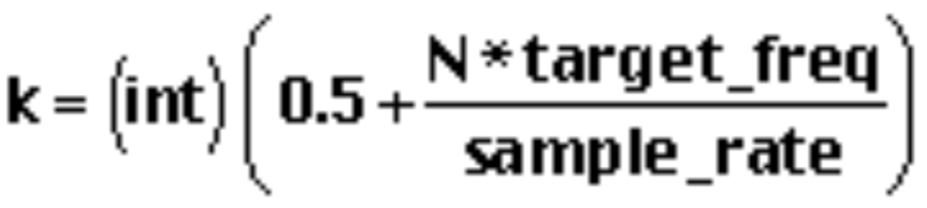
**Block size:** This variable, **N**, controls the frequency resolution (also called bin width).

For example, if your sampling rate is 8kHz and **N** is 100, then your bin width is 80Hz.

However, the larger **N** is the longer it takes to detect a tone since you have to wait longer for all the samples to be computed. So, the tradeoff with choosing **N** is frequency resolution vs time.

Lastly you want your target frequencies to be integer multiples of **sample\_rate/N** so that your target frequencies are centered in their respective bins.

### **3.2.1 Precomputed constants:**



***w* = (2\*π/N)\**k***

***cosine* = cos *w***

***sine* = sin *w***

***coeff* = 2 \* *cosine***

For the per-sample processing you're going to need three variables. Let's call them ***Q0****,* ***Q1****,* and ***Q2****.*

***Q1*** is just the value of ***Q0*** *last time.* ***Q2*** is just the value of ***Q0*** *two times ago* (or **Q1** last time).

***Q1*** and ***Q2*** must be initialized to zero at the beginning of each block of samples. For every *sample,* you need to run the following three equations:

***Q0 = coeff \* Q1 - Q2 + sample***

***Q2 = Q1***

***Q1 = Q0***

After running the per-sample equations N times, it's time to see if the tone is present or not.

***real = (Q1 - Q2 \* cosine)***

***imag = (Q2 \* sine)***

***magnitude2 = real2 + imag2***

A simple threshold test of the magnitude will tell you if the tone was present or not. Reset *Q2* and *Q1* to zero and start the next block.

# **4. Implementation of Goertzel’s Algorithm in MATLAB**

It was decided that the implementation of the DTMF decoder would be done in MATLAB, as all the required signal processing tools are available. MATLAB also allows for the visualization of the inputs and outputs thus allowing for a validated solution.

## **4.1 Software Implementation**

For the software implementation we started by choosing a **block size** of 370 as it produced the best frame size for an 8khz waveform. But for a differing sampling rate this block size can be varied to produce the best results.

To start the decoding process in the ***DTMF\_Decoder.m*** the input wave file is specified by typing in its target value. The audio file is then read by using the audioread() in MATLAB this data is then passed to ***createBlocks.m.***

In ***createBlocks.m*** the input data (that is a .wav file) is split into a matrix with 370 (This block size is determined by the sampling rate of the audio file and 370 is the produces the best results for a 8kHz audio file) rows and number of blocks = (length(data)/blockSize)\*2 - 1 columns. Every column then is a section of the input data.

This matrix is then passed as an input to ***DetectTones.m.***In ***DetectTones.m***the frequency bin is created. This is all the frequencies that we will want to detect when doing the FFT. This frequency bin included the high and low frequencies contained in our Touch-Tones as well as +/- 1.5% of that frequency to make sure that we adhere to the ITU standards. The Goertzel algorithm is then run on every block of input data. In MATLAB there is a goertzel function that can be used.

Below is part of the documentation provided for the function from MATLAB, this is also the way we implemented it in our code:

***goertzel(X,INDVEC****): computes the discrete Fourier transform (DFT) of X at indices contained in the vector INDVEC, using the second-order Goertzel algorithm. The indices must be integer values from 1 to N where N is the length of the first non-singleton dimension.*

The relevant indices were then chosen and the *goertzel* function run on all of the blocks of data that we had created. The results from running the *goertzel* function were stored in an output matrix, where every column was the equivalent of *goertzel* algorithm that had run on a block of data and the rows represented the 21 different indices.

A matrix is then returned that contains the maximum peaks at the relevant indices with every row in the matrix representing a frequency of interest.

This matrix gets passed to ***getValues.m***where this matrix gets parsed and frames are checked to see if they are a ‘pause’ or ‘silence’. The mean of the frames with DTMF tones is much higher than the mean of "silent" frames, so we found the 3 largest averages in the first 20 frames. A frame was then determined to be silent if it had a mean DFT magnitude less than 70% of the average of the top 3 frames

If the frame was not silent it was checked to see what frequencies present and thus which buttons were was pressed. This was all then appended into a string and given as an output. ***DTMF\_Decoder.m***simply calls all of these functions in the correct order to give the output desired.

# **5. Testing and Analysis**

The algorithm was tested for each of the various sound files found on the sound example page [4]. These sound examples included automatic dialing signals, signals recorded over the network that have been hand-dialed, signals recorded locally that have been hand-dialed and signals recorded locally with wideband coupling.

Each of the various signal inputs the implemented DTMF decoder was able to produce 95% accuracy of detection with regard to the sample files. It is also noted that the file where it was unable to produce the correct output, the output produced was off by one digit at the end and upon investigation the last tone of the sequence is at the slightly lower volume than the rest and this may have played a part in the detection of the last tone.

## **5.1 Plotting of Individual Tones**

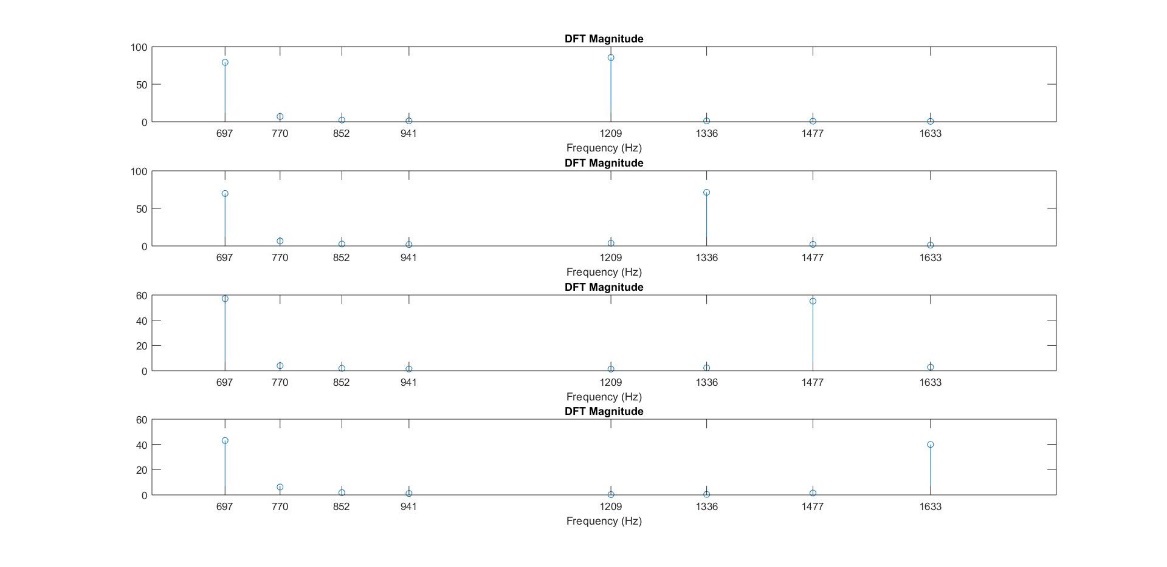
To provide a visual representation of how the decoder is able to determine which tone is being produced by the signal, the following plots have been made where the peak of each frequency as a point in the signal are shown. Therefore, it is clear to see which tone is being represented by the signal at that point in time. By taking the peaks of each of the tones the decoder is able to still decode DMTF tones in the presence of noise and dialing tones that could exist within the signal. This aspect of decoding whether a tone is a pause noise even dial tones is easy to see as the magnitude of the tones is much lower than that of a plot of a signal within the tone region and thus providing verification the method of deeming a tine to be a pause or non-tone if the mean of its magnitudes is less than 70% of the average of the top 3 frames in the whole signal.

Figure 3: Plot representing the peaks of the frequency in a DTMF signal that has the tones 123A

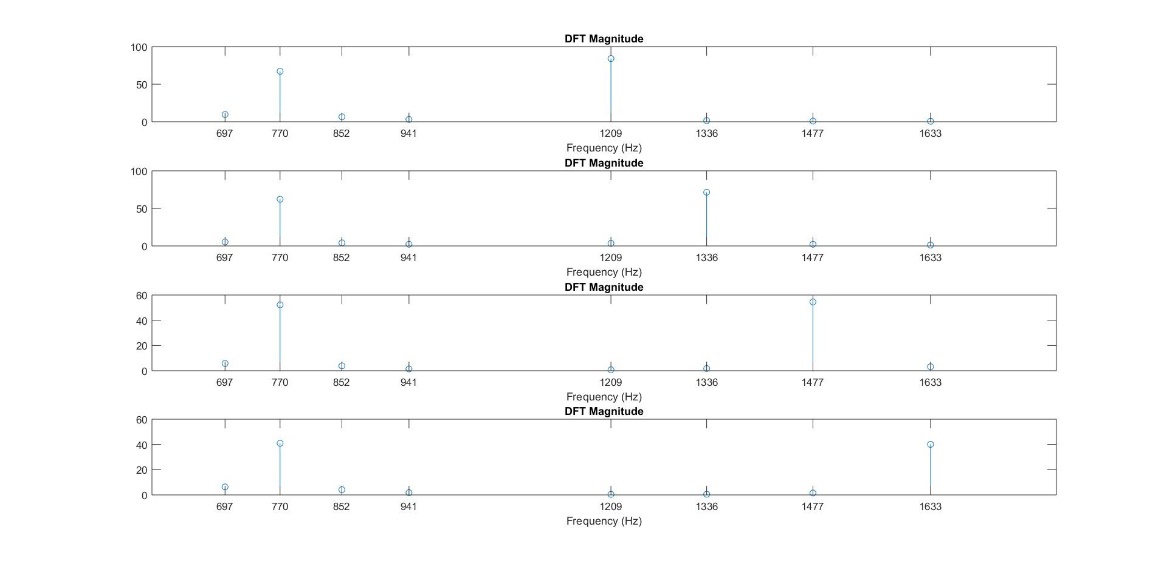
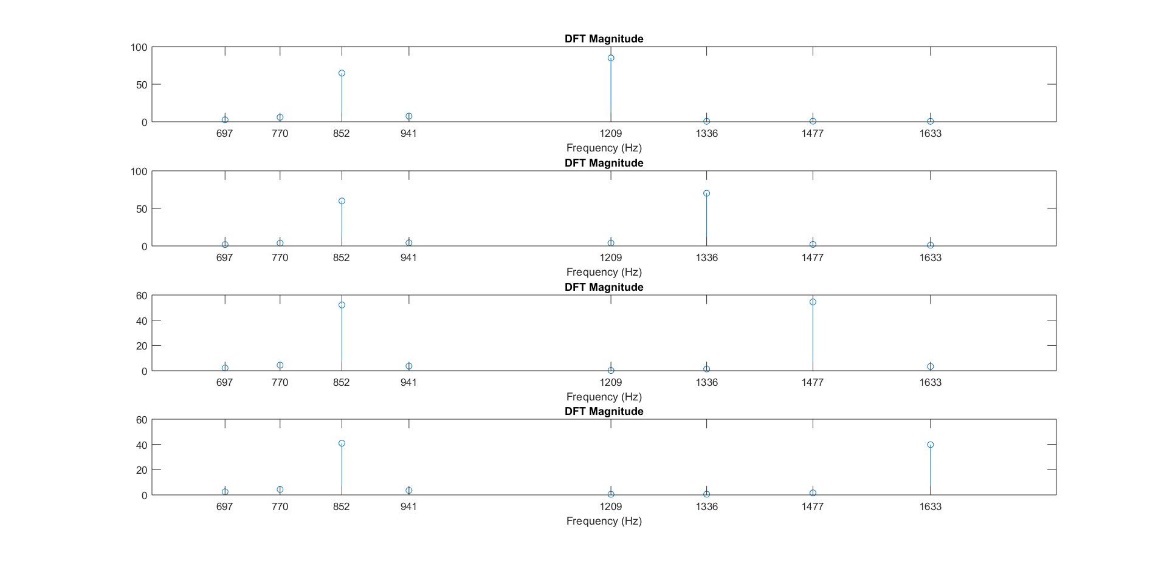
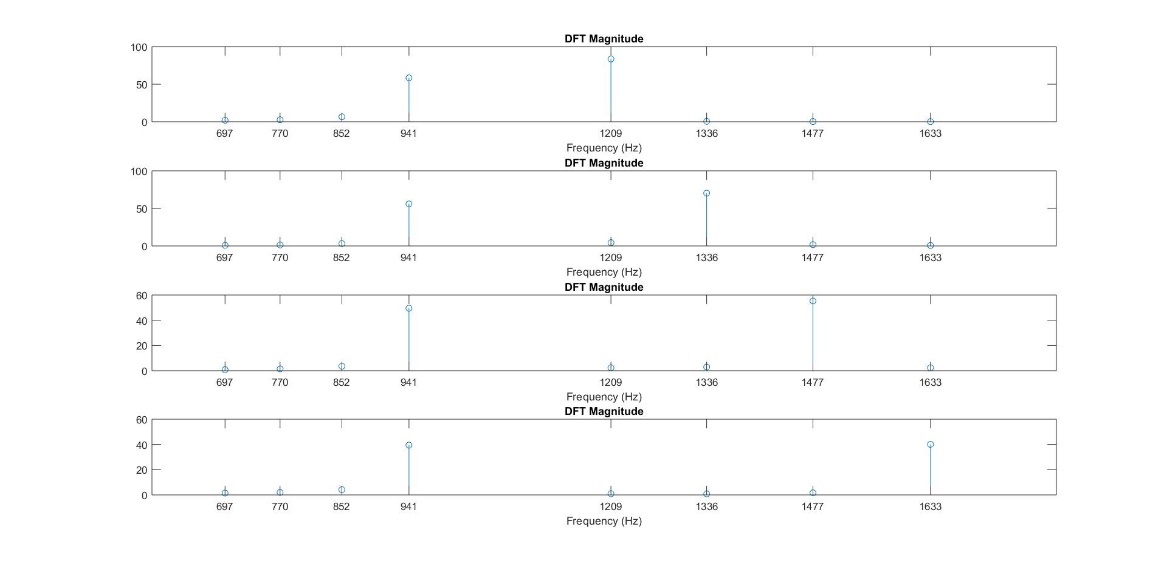


Figure 4: Plot representing the peaks of the frequency in a DTMF signal that has the tones \*0#D

Figure 5: Plot representing the peaks of the frequency in a DTMF signal that has the tones 789C

Figure 6: Plot representing the peaks of the frequency in a DTMF signal that has the tones 456B

## **5.2 Table of Decoded Tones**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Dial ABC Online DTMF Decoder** | **MATLAB Implemented Decoder** |  |
| ***Automatic Dialing*** | | |  |
| Input File | Output Produced | Output Produced | Result |
| dtmfA1.wav | 182846 | 182846 | ✔ |
| dtmfA2.wav | 31415 | 31425 | ✔ |
| dtmfA3.wav | 271827 | 271827 | ✔ |
| dtmfA4.wav | 8548928 | 8548928 | ✔ |
| dtmfA5.wav | 926535 | 926535 | ✔ |
| dtmfA6.wav | 8548928 | 8548928 | ✔ |
| ***Hand Dialed – Locally*** | | |  |
|  | Output Produced | Output Produced |  |
| dtmfM1.wav | 8548928 | 8548928 | ✔ |
| dtmfM2.wav | 12859 | 12859 | ✔ |
| dtmfM3.wav | 6619 | 6619 | ✔ |
| dtmfM4.wav | 12832 | 12832 | ✔ |
| dtmfM5.wav | 20164 | 20164 | ✔ |
| dtmfM6.wav | 196509 | 196509 | ✔ |
| dtmfM7.wav | 110405 | 110405 | ✔ |
| dtmfM8.wav | 20001 | 20001 | ✔ |
| dtmfM9.wav | 10711305 | 10711305 | ✔ |
| ***Hand Dialed – Over the Network*** | | |  |
|  | Output Produced | Output Produced |  |
| dtmfN1.wav | 121285 | 12128 | ✖\* |
| dtmfN2.wav | 48928151 | 48928151 | ✔ |
| dtmfN3.wav | 1965 | 1965 | ✔ |
| ***Local WBC*** | | |  |
|  | Output Produced | Output Produced |  |
| dtmfL1.wav | 8548928 | 8548928 | ✔ |
| dtmfL2.wav | 20001 | 20001 | ✔ |

The wave files were downloaded from the example samples page and run through an online DTMF decoder [6] which has proven capability of decoding DTMF tones at a range of frequencies and lengths as well as in the presence of noise. These results were then compare to the results produced by the DTMF decoder we implemented in MATLAB. These comparisons can be seen in the table below.

* The correct output was produced when the threshold that the mean of the magnitudes the tone must not be less than was set to 50% instead of 70% this was possibly caused by the last tone being softer than the rest.

# **6. Conclusion**

## **6.1 Review of Work Completed**

The decoder that was produced

## **6.2 Possible Improvements to the Experiment**

After a final review of the DTMF decoder there was some aspect that could be improved on for future implementations. Some of these improvements include:

* **Producing a GUI for better user interaction with the DTMF-Decoder.**
* **Allow for real-time decoding.**
* **Dynamic changing of the threshold required of a tone as experienced with the single error in decoding the examples audio files. [6]**
* **Allowing for decoding of various types of audio files.**
* **Identification of the type of file being decoded.**
* **Dynamic changing of the block size based on the sampling rate of the signal**
* Detection of the location of detected tones within the signal.

# **7. References**

1) ITU-T Recommendation Q.23 - Technical Features of Push-Button Telephone Sets. (1988). 1st ed. [ebook] INTERNATIONAL TELECOMMUNICATION UNION. Available at: https://www.itu.int/rec/dologin\_pub.asp?lang=e&id=T-REC-Q.23-198811-I!!PDF-E&type=items [Accessed 6 Apr. 2018].

2) ITU-T Recommendations Q.24 - Multifrequency Push-Button Reception. (1988). 1st ed. [ebook] INTERNATIONAL TELECOMMUNICATION UNION. Available at: https://www.itu.int/rec/dologin\_pub.asp?lang=e&id=T-REC-Q.24-198811-I!!PDF-E&type=items [Accessed 6 Apr. 2018].

3) Anon, (2018). [online] Available at: https://www.researchgate.net/figure/DTMF-Keypad Layout\_fig1\_264782796 [Accessed 6 Apr. 2018].

4) Ee.columbia.edu. (2018). *Sound examples*. [online] Available at: http://www.ee.columbia.edu/~dpwe/sounds/dtmf/ [Accessed 7 Apr. 2018].

5) En.wikipedia.org. (2018). *Dual-tone multi-frequency signaling*. [online] Available at: https://en.wikipedia.org/wiki/Dual-tone\_multi-frequency\_signaling [Accessed 10 Apr. 2018].

6) The Engineering Projects. (2018). *DTMF Decoder using MATLAB - The Engineering Projects*. [online] Available at: https://www.theengineeringprojects.com/2016/05/dtmf-decoder-using-matlab.html [Accessed 9 Apr. 2018].

7) En.wikipedia.org. (2018). *Goertzel algorithm*. [online] Available at: https://en.wikipedia.org/wiki/Goertzel\_algorithm [Accessed 23 Apr. 2018].

# **Appendix A: Matlab Code**

### **DTMF\_Decoder.m**

%Decode audio file with DTMF Tones and return values

[data,Fs] = audioread('789C.wav')

blocks = createBlocks(data);

tones = DetectTones(blocks,Fs);

Values = getValues(tones)

### **createBlocks.m**

function blocks = createBlocks (data)

%Make blocks of size "block size" and return

%a matrix with each of the columns as a seperate block for processing

blockSize = 370;

if (length(data) < blockSize)

blockSize = length(data);

nBlocks = 1;

else

nBlocks = floor(length(data)/blockSize)\*2 - 1;

end

% Preallocate memory for the output

blocks = zeros(blockSize,nBlocks);

col = 1;

for i= 1 : floor(blockSize/2) : length(data)

new = data(i:i+blockSize-1);

blocks(:,col) = new;

if (col == nBlocks) % break when all the blocks have been created

break;

end

col = col+1;

end

end

### **DectectTones.m**

function tones = DetectTones( blocks, Fs )

% Decode a given block matrix and give an output in the form a

% matrix with each column giving the magnitude of the each of the DTMF

% signals in the corresponding block.

% initialise the output matrix

output = zeros(21,size(blocks,2));

fre = [697, 770, 852, 941, 1209, 1336, 1477, 1633];

freq\_bin = [687,707,758,782,839,865,927,955,1191,1210,1227,1316,1336,1356,1455,1466,1499,1609,1620,1647,1657];

freq\_indices = round(freq\_bin/Fs \* size(blocks,1)) + 1;

for i = 1:size(blocks,2)

output(:,i) = abs(goertzel(blocks(:,i),freq\_indices));

end

%For each block of data the highest spectral peak for each DTMF

%frequency must be chosen.

tones = zeros(8,size(blocks,2));

for f = 1:size(blocks,2) % for each frame

tones(1,f) = max(output(1:2,f)); % 697Hz

tones(2,f) = max(output(3:4,f)); % 770Hz

tones(3,f) = max(output(5:6,f)); % 852Hz

tones(4,f) = max(output(7:8,f)); % 941Hz

tones(5,f) = max(output(9:11,f)); % 1209Hz

tones(6,f) = max(output(12:14,f)); % 1336Hz

tones(7,f) = max(output(15:17,f)); % 1477Hz

tones(8,f) = max(output(18:21,f)); % 1633Hz

end

p=1

for k = 3:9:size(blocks,2)

subplot(4,1,p);

stem(fre,tones(:,k))

ax = gca;

ax.XTick = fre;

xlabel('Frequency (Hz)');

title('DFT Magnitude');

p = p+1;

end

end

### **getValues.m**

function final\_val = getValues(tones)

% Decode each of the blocks to get the number represented by

% the data from the Goertzel Calculation.

% the output "in\_val" will be the number represented by each frame. This

% has the numbers decoded from every frame present. Blocks that represent a

% "silence" will be shows as "\_" strings

init\_val = repmat('~',[1,size(tones,2)]);

%go through each vector and first determine whether it is a silence or

%an actual DTMF. if it is a silent, add '\_' to the output. if it is not

%a silence, get the indicies of the two highest peaks.

% get the top 3 frames from first 20

if (length(tones) >= 50)

f20 = sort(mean(tones(:,1:50)),'descend');

else

f20 = sort(mean(tones),'descend');

end

% remove the frames with an avg of zero and use that array for the

% averaging

if (size(f20,2) < 6)

avg = mean(f20(1));

else

avg = mean(f20(2:6)); % average of the top 5

end

% go through all the frames, decode frames with a mean greater than

% 10% of 'topAvg'

for j = 1 : size(tones,2) % for each decoded frame

%get index of highest DTMF high and low frequencies

if (mean(tones(:,j)) < (0.66 \* avg))

init\_val(j) = '\_';

else

[a,low] = max(tones(1:4,j));

[a,high] = max(tones(5:8,j));

% find the corresponding frequencies

if (low == 1) %low = 697

if (high == 1) %high = 1209

init\_val(j) = '1';

elseif (high == 2) %high = 1336

init\_val(j) = '2';

elseif (high == 3) %high = 1477

init\_val(j) = '3';

elseif (high == 4) %high = 1633

init\_val(j) = 'A';

end

elseif (low == 2) %low = 770

if (high == 1) %high = 1209

init\_val(j) = '4';

elseif (high == 2) %high = 1336

init\_val(j) = '5';

elseif (high == 3) %high = 1477

init\_val(j) = '6';

elseif (high == 4) %high = 1633

init\_val(j) = 'B';

end

elseif (low == 3) %low = 852

if (high == 1) %high = 1209

init\_val(j) = '7';

elseif (high == 2) %high = 1336

init\_val(j) = '8';

elseif (high == 3) %high = 1477

init\_val(j) = '9';

elseif (high == 4) %high = 1633

init\_val(j) = 'C';

end

elseif (low == 4) %low = 941

if (high == 1) %high = 1209

init\_val(j) = '\*';

elseif (high == 2) %high = 1336

init\_val(j) = '0';

elseif (high == 3) %high = 1477

init\_val(j) = '#';

elseif (high == 4) %high = 1633

init\_val(j) = 'D';

end

end

end

end % end of loop through each frame

final\_val = '';

if (init\_val(1) ~= '\_')

final\_val = init\_val(1);

end

for i = 2 : length(init\_val)

if (init\_val(i) == '\_' && init\_val(i-1) ~= '\_')

final\_val = strcat(final\_val,' ');

elseif (init\_val(i) ~= init\_val(i-1))

final\_val = strcat(final\_val,init\_val(i));

end

end

end