GLASGOW COLLEGE, UESTC

Digital Signal Processing Course Project Generation and Detection of DTMF Signals

Yimin Tian 2017200603010

April 19, 2020

1. INTRODUCTION

In digital communication system, engineers usually need to tackle transmitting and recognising dial number. Generally, a signalling system, between communication devices and switching centers, is used to transmit dial number. In the early days, the telephone system uses a series of intermittent pulses signal to transmit the called number. However, pulse dealing requires operators in the telecommunications office to manually complete long-distance connections. To automatically complete long distance calls, dual-tone multi-frequency signalling (DTMF), a telecommunication singling system, is introduced to use voice-frequency band over telephone lines. The DTMF system uses a set of eight audio frequencies transmitted in pairs to represent 16 signals, represented by the ten digits, the letters A to D, and the symbols # and *. As the signals are audible tones in the voice frequency range, they can be transmitted through electrical repeaters and amplifiers, and over radio and microwave links, thus eliminating the need for intermediate operators on long-distance circuits.

PROBLEM FORMULATION In this report, project is divided into 3 parts: DTMF generation, DTMF detection, MATLAB GUI. MATLAB GUI is designed firstly, as I prefer to complete the overall operation of the whole system, the I design each individual block. Secondly, DTMF generation and detection use digital oscillator and Goertzel algorithm respectively. Third, simulation result will be given for different test dial number to examine whether the design is correct.

2. DTMF SIGNALS GENERATION

The encoder portion and tone generation part of DTMF encode process is based on two oscillators, one for the row the other one for the column tone. By storing (table 2) column frequency group into 4x1 matrix and row frequency group into 1x4 matrix, **tone=filter([0 sin(2*pi*dtmf.rowTones(r,c)/fs)],[1 -2*cos(2*pi*dtmf.rowTones(r,c)/fs) 1],x) + filter([0 sin(2*pi*dtmf.colTones(r,c)/fs],[1 -2*cos(2*pi*dtmf.colTones(r,c)/fs) 1],x)**, **(figure 1)** produce tone for each key input. As typical DTMF frequencies rang from approx. 700 Hz to 1700 Hz, a sampling rate of 8 kHz for this implementation puts us in a safe area of the Nyquist criteria.

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	[1]	[2]	[3]	[A]
770 Hz	[4]	[5]	[6]	[B]
852 Hz	[7]	[8]	[9]	[C]
941 Hz	[*]	[0]	[#]	[D]

Table 2.1: DTMF keypad frequencies

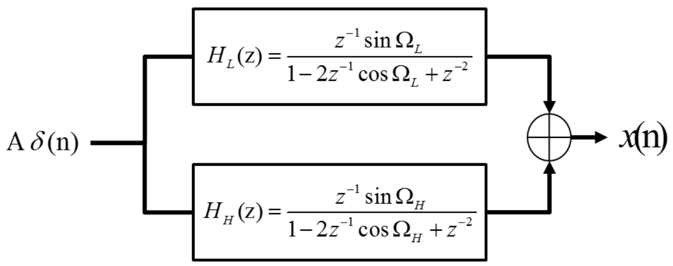

Figure 2.1: encode algorithm

```
global keyNames tone_all h1 h2
fs=8000;
t=(0:1:204*5)/fs;
x=zeros(1,length(t));
x(1)=1;

dtmf.keys = ...
    ['1','2','3','A';
     '4','5','6','B';
     '7','8','9','C';
     '*','0','#','D'];

dtmf.colTones = ones(4,1)*[1209,1336,1477,1633];
dtmf.rowTones = [697;770;852;941]*ones(1,4);

keyName = keyNames(length(keyNames));
[r,c] = find(dtmf.keys==keyName); % find row and col for keyname
```

```
tone=filter([0 sin(2*pi*dtmf.rowTones(r,c)/fs) ],[1 -2*cos(2*pi*dtmf.rowTones(r,c)/fs) 1],x) + filter([0 sin(2*pi*dtmf.colTones(r,c)/fs) ],[1 -2*cos(2*pi*dtmf.colTones(r,c)/fs) 1],x);

soundsc(tone,fs);
tone_all=[tone_all,zeros(1,400),tone];

h1=subplot(2,3,2);plot(t,tone);grid on;
title('Signal tone');
ylabel('Amplitude');
xlabel('time (second)');
axis([0 0.035 -2 2]);

Ak=2*abs(fft(tone))/length(tone);Ak(1)=Ak(1)/2;
f=[0:1:(length(tone)-1)/2]*fs/length(tone);
h2=subplot(2,3,5);plot(f,Ak(1:(length(tone)+1)/2));grid on
title('Spectrum for tone');
ylabel('Amplitude');
xlabel('frequency (Hz)');
axis([500 2000 0 1]);
```

3. DTMF Signal Detection

The task to detect DTMF tones in a incoming signal and convert them into actual digits is certainly more complex than the encoding process. The decoding process is by its nature a continuous process, meaning it needs to search an ongoing incoming data stream for the presence of DTMF tones continually.

Goertzel Algorithm The Goertzel algorithm is the basis of the DTMF detector. This method is a very effective and fast way to extract spectral information from an input signal. This algorithm essentially utilizes two-pole IIR type filters to effectively compute DFT values. It thereby is a recursive structure always operating on one incoming sample at a time, as compared to the DFT (or FFT) which needs a block of data before being able to start processing. The IIR structure for the Goertzel filter incorporates two complex-conjugate poles and facilitates the computation of the difference equation by having only one real coefficient. For the actual tone detection the magnitude (here squared magnitude) information of the DFT is sufficient. After a certain number of samples N (equivalent to a DFT block size) the Goertzel filter output converges towards a pseudo DFT value vk(n), which can then be used to determine the squared magnitude.

Goertzel Algorithm in short:

1. Recursively compute for n = 0 .. N

$$v_k(n) = 2\cos\left(\frac{2\pi}{N}k\right)\cdot v_k(n-1) - v_k(n-2) + x(n)$$

$$where \quad v_k(-1) = 0 \quad v_k(-2) = 0$$

$$x(n) = input$$

2. Compute once every N

$$|X(k)|^2 = y_k(N)y^*_k(n)$$

$$= v^2{}_k(n) + v^2{}_k(N-1) - 2\cos(2\pi f_k/f_s)v^2{}_k(N)\, v^2{}_k(N-1)$$

Figure 3.1: Goerzel algorithm in short

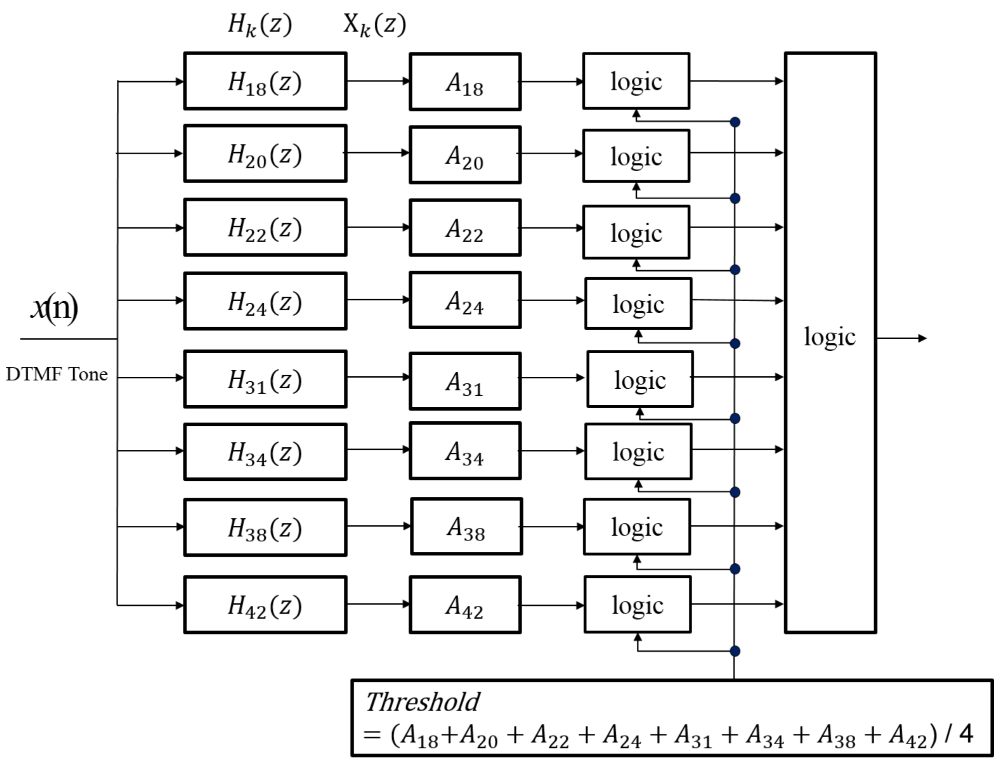

Figure 3.2: Goertzel Algorithm

DTMF Frequency (Hz)	Frequency Bin : k
697	18
770	20
852	22
941	24
1209	31
1336	34
1477	38
1633	42

Table 3.1: DTMF Frequency (Hz) & Frequency Bin : k

Table 3.1 contains a list of frequencies and filter coefficients. Each filter is tuned to most accurately coincide with the actual DTMF frequencies. This is also true for corresponding 2nd harmonics. The exception is the fundamental column frequencies. Each column frequency has two frequency bins attached, which deviate $+/-9$Hz from center (see Table 3.1).

$$k = f/f_s \times N, N = 205$$

The parameter N defines the number of recursive iterations and also provides a means to tune for frequency resolution. The above relationship maps N to the width of a frequency bin mainlobe and thereby frequency resolution.

```
global h1 h3 h4 Decode_output
Decode_output=[];
output=[];
tone_all_2=audioread('tone_all.wav'); % load
tone_all_2=(tone_all_2')*2;
fs=8000;
%% Filter Bank Design
a697=[1 -2*cos(2*pi*18/205) 1];
a770=[1 -2*cos(2*pi*20/205) 1];
a852=[1 -2*cos(2*pi*22/205) 1];
a941=[1 -2*cos(2*pi*24/205) 1];
a1209=[1 -2*cos(2*pi*31/205) 1];
a1336=[1 -2*cos(2*pi*34/205) 1];
a1477=[1 -2*cos(2*pi*38/205) 1];
a1633=[1 -2*cos(2*pi*42/205) 1];

[w1, f]=freqz([1 -exp(-2*pi*18/205)],a697,512,fs);
[w2, f]=freqz([1 -exp(-2*pi*20/205)],a770,512,fs);
[w3, f]=freqz([1 -exp(-2*pi*22/205)],a852,512,fs);
[w4, f]=freqz([1 -exp(-2*pi*24/205)],a941,512,fs);
[w5, f]=freqz([1 -exp(-2*pi*31/205)],a1209,512,fs);
[w6, f]=freqz([1 -exp(-2*pi*34/205)],a1336,512,fs);
[w7, f]=freqz([1 -exp(-2*pi*38/205)],a1477,512,fs);
[w8, f]=freqz([1 -exp(-2*pi*42/205)],a1633,512,fs);
%    [H,F] = freqz(...,N,Fs) and [H,F] = freqz(...,N,'whole',Fs)
    return
```

```
%    frequency vector F (in Hz), where Fs is the sampling frequency (in Hz).

t=(0:length(tone_all_2)-1)/fs;
h1=subplot(2,3,2);plot(t,tone_all_2);grid on;
title('Signal␣tone');
ylabel('Amplitude');
xlabel('time␣(second)');

h3=subplot(2,3,3);plot(f,abs(w1)/1000,f,abs(w2)/1000,f,abs(w3)/1000,f,abs(w4)/1000,f,abs(w5)/1000,f,abs(w6)/1000,f,abs(w7)/1000,f,abs(w8)/1000);grid on
title('BPF␣frequency␣responses');
xlabel('Frequency␣(Hz)');
ylabel('Amplitude');
axis([500 2000 0 1]);
legend('697','770','852','941','1209','1336','1477','1633');
%% Decode
for ii=0:(length(tone_all_2)/1421-1)
    tone=tone_all_2(1+1421*ii:1421*(ii+1));
    tone=tone(401:end);

    yDTMF=[tone 0];
    y697=filter(1,a697,yDTMF);
    y770=filter(1,a770,yDTMF);
    y852=filter(1,a852,yDTMF);
    y941=filter(1,a941,yDTMF);
    y1209=filter(1,a1209,yDTMF);
    y1336=filter(1,a1336,yDTMF);
    y1477=filter(1,a1477,yDTMF);
    y1633=filter(1,a1633,yDTMF);
%   y = filter(b,a,x) filters the input data x using a rational transfer function
%   defined by the numerator and denominator coefficients b and a.

    m(1)=sqrt(y697(206)^2+y697(205)^2-2*cos(2*pi*18/205)*y697(206)*y697(205));
    m(2)=sqrt(y770(206)^2+y770(205)^2-2*cos(2*pi*20/205)*y770(206)*y770(205));
    m(3)=sqrt(y852(206)^2+y852(205)^2-2*cos(2*pi*22/205)*y852(206)*y852(205));
    m(4)=sqrt(y941(206)^2+y941(205)^2-2*cos(2*pi*24/205)*y941(206)*y941(205));
    m(5)=sqrt(y1209(206)^2+y1209(205)^2-2*cos(2*pi*31/205)*y1209(206)*y1209(205));
    m(6)=sqrt(y1336(206)^2+y1336(205)^2-2*cos(2*pi*34/205)*y1336(206)*y1336(205));
    m(7)=sqrt(y1477(206)^2+y1477(205)^2-2*cos(2*pi*38/205)*y1477(206)*y1477(205));
    m(8)=sqrt(y1633(206)^2+y1633(205)^2-2*cos(2*pi*42/205)*y1633(206)*y1633(205));
    m=2*m/205;
    th=sum(m)/4;  % based on empirical measurement
    f=[697 770 852 941 1209 1336 1477 1633];
```

```
    f1=[0  4000];
    th=[th th];

    idx=find(m>th(1));
    Determination=f(idx);
    switch Determination(1)
            case {697}
                switch Determination(2)
                    case {1209}
                        output='1';
                    case {1336}
                        output='2';
                    case {1477}
                        output='3';
                    case {1633}
                        output='A';
                end
            case {770}
                switch Determination(2)
                    case {1209}
                        output='4';
                    case {1336}
                        output='5';
                    case {1477}
                        output='6';
                    case {1633}
                        output='B';
                end
            case {852}
                switch Determination(2)
                    case {1209}
                        output='7';
                    case {1336}
                        output='8';
                    case {1477}
                        output='9';
                    case {1633}
                        output='C';
                end
            case {941}
                switch Determination(2)
                    case {1209}
                        output='*';
                    case {1336}
                        output='0';
                    case {1477}
                        output='#';
                    case {1633}
                        output='D';
                end
    end

    Decode_output=[Decode_output,output];

```

```
    h4=subplot(2,3,6);stem(f,m);grid on
    hold on;
    plot(f1,th); % Threshold
    title('Decode Spectrum');
    xlabel('Frequency (Hz)');
    ylabel('Amplitude');
    axis([500 2000 0 1]);
    clear th
    hold off

    set(Display2,'String',Decode_output); % Property
    soundsc(tone,fs);
    pause(0.25);
end % LOOP
```

4. Numerical Experiments

This project includes a MATLAB GUI panel, which can collect input messages, shows signal tone, BPF frequency response, spectrum of tone and decode spectrum. To examine project can transmit dialled number in DTMF way, a test number would be input and its output signal frequency response and spectrum will be shown.

Test number: 18628025606

4.1. Encode Examination

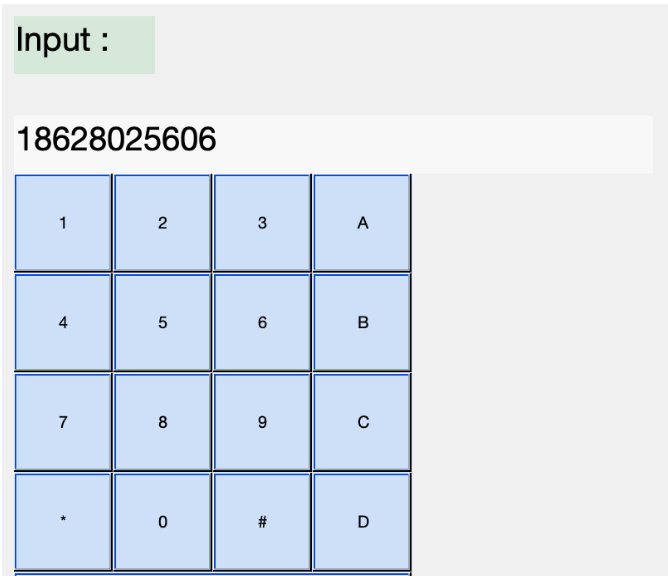

Figure 4.1: input dial number

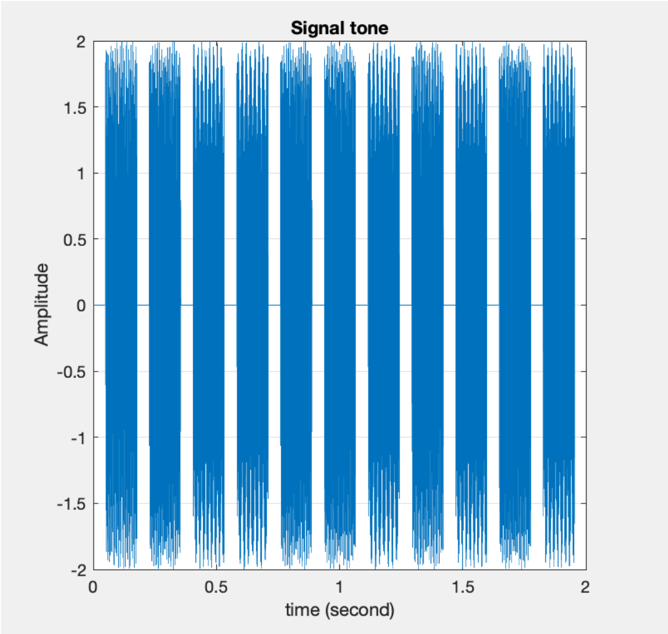

Figure 4.2: input all tone in time domain

4.2. Decode Examination

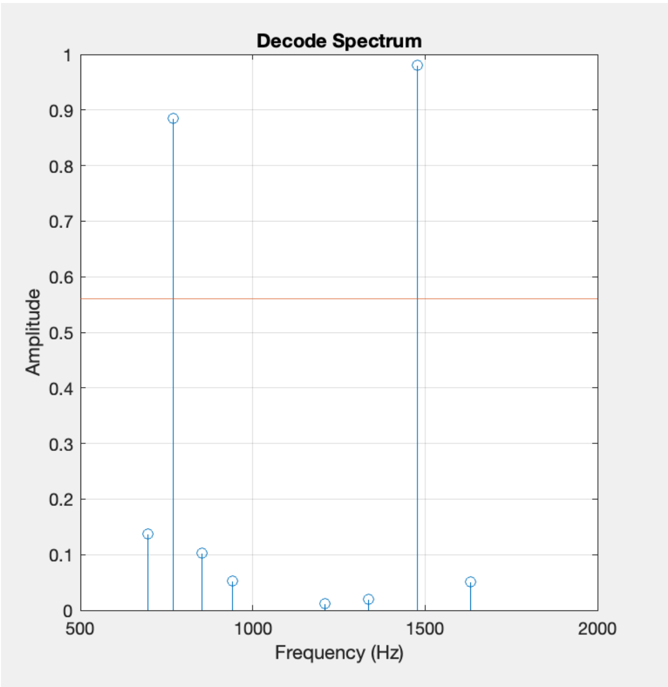

Figure 4.3: decode spectrum

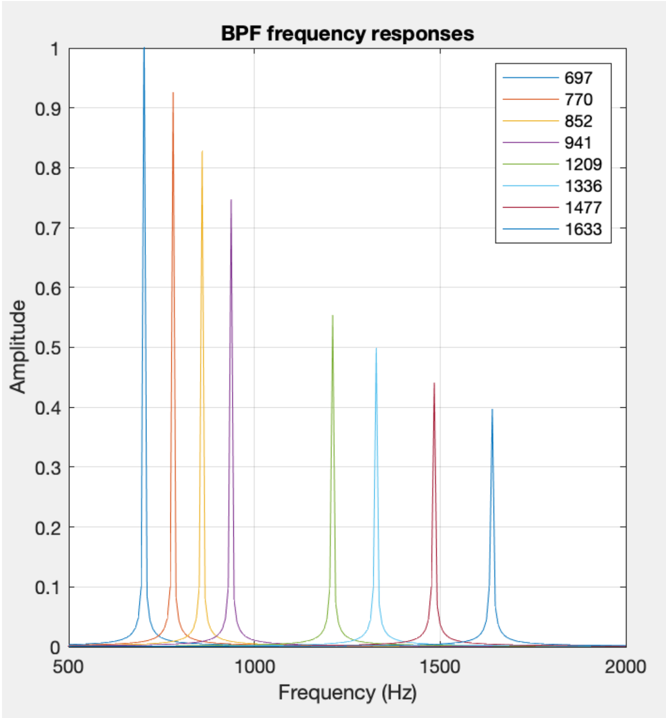

Figure 4.4: BPF frequency responses

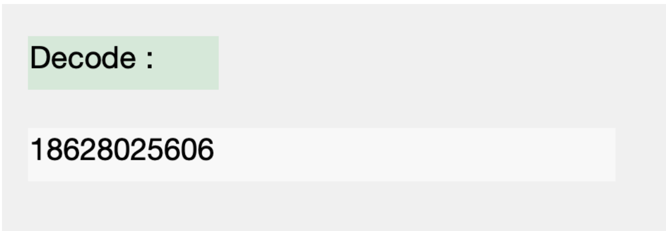

Figure 4.5: decode result

5. CONCLUSION

DTMF tone encoding and decoding concepts and algorithms were described here in some detail. Further theoretical background is provided in the appendix. The DTMF encoder and decoder implementations were explained and the associated speed and memory requirements were presented. The DTMF tone decoder has been tested according to the MITEL and BELLCORE test specifications and the results are documented. It is important to note that the encoder and decoder was implemented as reentrant, C-callable functions, which facilitate setting up a multichannel DTMF decoder system. The code is modular and easy to integrate into any given telephony application. The decoder algorithm was greatly optimised to meet the test specifications as well as offer a very attractive MIPS count of around 1.1 MIPS per channel of generation and detection.

6. Appendix

A. Code: DTMF tone generator

```
global keyNames tone_all h1 h2
fs=8000;
t=(0:1:204*5)/fs;
x=zeros(1,length(t));
x(1)=1;

dtmf.keys = ...
   ['1','2','3','A';
    '4','5','6','B';
    '7','8','9','C';
    '*','0','#','D'];

dtmf.colTones = ones(4,1)*[1209,1336,1477,1633];
dtmf.rowTones = [697;770;852;941]*ones(1,4);

keyName = keyNames(length(keyNames));
[r,c] = find(dtmf.keys==keyName); % find row and col for keyname
tone=filter([0 sin(2*pi*dtmf.rowTones(r,c)/fs) ],[1 -2*cos(2*pi*
    dtmf.rowTones(r,c)/fs) 1],x) + filter([0 sin(2*pi*dtmf.colTones
    (r,c)/fs) ],[1 -2*cos(2*pi*dtmf.colTones(r,c)/fs) 1],x);

soundsc(tone,fs);
tone_all=[tone_all,zeros(1,400),tone];

h1=subplot(2,3,2);plot(t,tone);grid on;
title('Signal tone');
ylabel('Amplitude');
xlabel('time (second)');
axis([0 0.035 -2 2]);

Ak=2*abs(fft(tone))/length(tone);Ak(1)=Ak(1)/2;
f=[0:1:(length(tone)-1)/2]*fs/length(tone);
h2=subplot(2,3,5);plot(f,Ak(1:(length(tone)+1)/2));grid on
title('Spectrum for tone');
ylabel('Amplitude');
xlabel('frequency (Hz)');
axis([500 2000 0 1]);
```

B. Code: DTMF decode

```
global h1 h3 h4 Decode_output
```

```
Decode_output=[];
output=[];
tone_all_2=audioread('tone_all.wav'); % load
tone_all_2=(tone_all_2')*2;
fs=8000;
%% Filter Bank Design
a697=[1 -2*cos(2*pi*18/205) 1];
a770=[1 -2*cos(2*pi*20/205) 1];
a852=[1 -2*cos(2*pi*22/205) 1];
a941=[1 -2*cos(2*pi*24/205) 1];
a1209=[1 -2*cos(2*pi*31/205) 1];
a1336=[1 -2*cos(2*pi*34/205) 1];
a1477=[1 -2*cos(2*pi*38/205) 1];
a1633=[1 -2*cos(2*pi*42/205) 1];

[w1, f]=freqz([1 -exp(-2*pi*18/205)],a697,512,fs);
[w2, f]=freqz([1 -exp(-2*pi*20/205)],a770,512,fs);
[w3, f]=freqz([1 -exp(-2*pi*22/205)],a852,512,fs);
[w4, f]=freqz([1 -exp(-2*pi*24/205)],a941,512,fs);
[w5, f]=freqz([1 -exp(-2*pi*31/205)],a1209,512,fs);
[w6, f]=freqz([1 -exp(-2*pi*34/205)],a1336,512,fs);
[w7, f]=freqz([1 -exp(-2*pi*38/205)],a1477,512,fs);
[w8, f]=freqz([1 -exp(-2*pi*42/205)],a1633,512,fs);
%    [H,F] = freqz(...,N,Fs) and [H,F] = freqz(...,N,'whole',Fs) return
%    frequency vector F (in Hz), where Fs is the sampling frequency (in Hz).

t=(0:length(tone_all_2)-1)/fs;
h1=subplot(2,3,2);plot(t,tone_all_2);grid on;
title('Signal tone');
ylabel('Amplitude');
xlabel('time (second)');

h3=subplot(2,3,3);plot(f,abs(w1)/1000,f,abs(w2)/1000,f,abs(w3)/1000,f,abs(w4)/1000,f,abs(w5)/1000,f,abs(w6)/1000,f,abs(w7)/1000,f,abs(w8)/1000);grid on
title('BPF frequency responses');
xlabel('Frequency (Hz)');
ylabel('Amplitude');
axis([500 2000 0 1]);
legend('697','770','852','941','1209','1336','1477','1633');
%% Decode
for ii=0:(length(tone_all_2)/1421-1)
    tone=tone_all_2(1+1421*ii:1421*(ii+1));
    tone=tone(401:end);

    yDTMF=[tone 0];
    y697=filter(1,a697,yDTMF);
    y770=filter(1,a770,yDTMF);
    y852=filter(1,a852,yDTMF);
    y941=filter(1,a941,yDTMF);
    y1209=filter(1,a1209,yDTMF);
    y1336=filter(1,a1336,yDTMF);
```

```
    y1477=filter(1,a1477,yDTMF);
    y1633=filter(1,a1633,yDTMF);
%   y = filter(b,a,x) filters the input data x using a rational transfer function
%   defined by the numerator and denominator coefficients b and a.

    m(1)=sqrt(y697(206)^2+y697(205)^2-2*cos(2*pi*18/205)*y697(206)*
        y697(205));
    m(2)=sqrt(y770(206)^2+y770(205)^2-2*cos(2*pi*20/205)*y770(206)*
        y770(205));
    m(3)=sqrt(y852(206)^2+y852(205)^2-2*cos(2*pi*22/205)*y852(206)*
        y852(205));
    m(4)=sqrt(y941(206)^2+y941(205)^2-2*cos(2*pi*24/205)*y941(206)*
        y941(205));
    m(5)=sqrt(y1209(206)^2+y1209(205)^2-2*cos(2*pi*31/205)*y1209
        (206)*y1209(205));
    m(6)=sqrt(y1336(206)^2+y1336(205)^2-2*cos(2*pi*34/205)*y1336
        (206)*y1336(205));
    m(7)=sqrt(y1477(206)^2+y1477(205)^2-2*cos(2*pi*38/205)*y1477
        (206)*y1477(205));
    m(8)=sqrt(y1633(206)^2+y1633(205)^2-2*cos(2*pi*42/205)*y1633
        (206)*y1633(205));
    m=2*m/205;
    th=sum(m)/4;  % based on empirical measurement
    f=[697 770 852 941 1209 1336 1477 1633];
    f1=[0  4000];
    th=[th th];

    idx=find(m>th(1));
    Determination=f(idx);
    switch Determination(1)
            case {697}
                switch Determination(2)
                    case {1209}
                        output='1';
                    case {1336}
                        output='2';
                    case {1477}
                        output='3';
                    case {1633}
                        output='A';
                end
            case {770}
                switch Determination(2)
                    case {1209}
                        output='4';
                    case {1336}
                        output='5';
                    case {1477}
                        output='6';
                    case {1633}
                        output='B';
                end
            case {852}
```

```
                switch Determination(2)
                    case {1209}
                        output='7';
                    case {1336}
                        output='8';
                    case {1477}
                        output='9';
                    case {1633}
                        output='C';
                end
            case {941}
                switch Determination(2)
                    case {1209}
                        output='*';
                    case {1336}
                        output='0';
                    case {1477}
                        output='#';
                    case {1633}
                        output='D';
                end
    end

    Decode_output=[Decode_output,output];

    h4=subplot(2,3,6);stem(f,m);grid on
    hold on;
    plot(f1,th); % Threshold
    title('Decode Spectrum');
    xlabel('Frequency (Hz)');
    ylabel('Amplitude');
    axis([500 2000 0 1]);
    clear th
    hold off

    set(Display2,'String',Decode_output); % Property
    soundsc(tone,fs);
    pause(0.25);
end % LOOP
```

References

[1] Asnjit K. Mitra. *Digital Signal Processing: A Computer-Based Approach, Fourth Edition.* EIP, 2011.

[2] Gunter Schmer, MTSA. [*DTMF Tone Generation and Detection: An Implementation Using the TMS320C54x*]. SC Group Technical Marketing, SPRA096A, May 2000.

[3] WIKIPEDIA: Dual-tone multi-frequency signaling, `https://en.wikipedia.org/wiki/Dual-tone_multi-frequency_signaling`