

Implementaion of DTMF Decoding with Matlab

Zheng GONG

Department of Electrical and Computer Engineering

University of Maryland

College Park, US

Email: joeygong@termpmail.umd.edu

Abstract—In this paper two method of decoding DTMF(Dual-tone multi-frequency) signaling are accomplished with Matlab. First of them uses the method of FFT and the other one with a filter approaching. Both methods decode an input wave file including multi-digits of code into a readable string of numbers accurately. The decoder can detect every single signal in the input multi-digits wave file then split them so periodic dial signals are not required.

Keywords—DTMF, decoder, Matlab, multi-digits.

I. INTRODUCTION

DTMF is used for telecommunication signaling over analog telephone lines in the voice-frequency band between telephone handsets and other communications devices and the switching center. To make the whole system of telephone communication works, a DTMF decoder plays an important role.

A. DTMF

In DTMF, a 4×4 matrix is formed by each row representing a low frequency, and each column representing a high frequency, as shown in table I. Numbers from 0 to 9, letters from A to D and symbols * and # are represented by a combination of a low frequency and a high frequency. Also, special tone frequencies are used to represent busy signal, dial tone and so on. frequencies used here are all lower than 600 and differ by countries and will not be discussed here.

TABLE I. DTMF KEYPAD FREQUENCIES MATCH

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

B. FFT

Fast Fourier Transform(FFT) is an algorithm to make computation of Discrete Fourier Transform(DFT) faster. By DFT, signals are converted from time domain to frequency domain, which makes it easier to do analysis. In this project, for a single tone of dial, with effect of noise, there should be more than two frequencies having non-zero amplitude after DFT, but two of them should be relatively larger and they are actually the target frequencies.

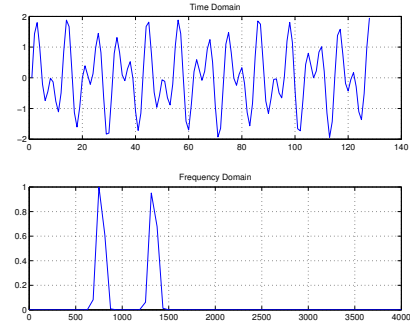


Fig. 1. Example FFT of dialing '5'

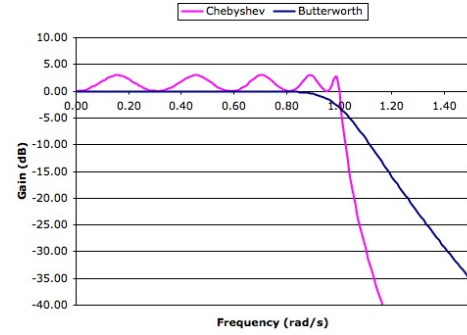


Fig. 2. Compare of Filters

The figure 1 shows an instance of the FFT of signal '5', two of the frequency peak can be easily found. Once we can find the two peaks of the signal, the signal can be identified.

C. Type I Chebyshev Filter

Chebyshev filters(Type I) are filters having a steeper roll-off and more passband ripple than the Butterworth filter(As shown in figure 2). As in this application, there are frequencies need to be filtered but closed to the needed frequency(The highest busy tone is 620Hz and the lowest dial tone is 697Hz, only 80Hz difference). So to minimize the error, we decided to choose Chebyshev filter rather than Butterworth filter in sacrificing of ripple in the passband.

In addition, we only use the filter to identify frequency but not going to playback, the distortion in passband does not matter.

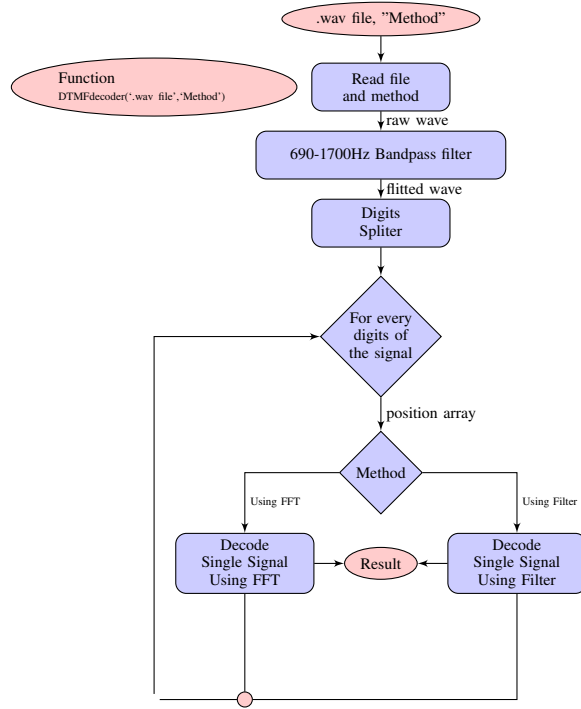


Fig. 3. Flow Chart for the decoder

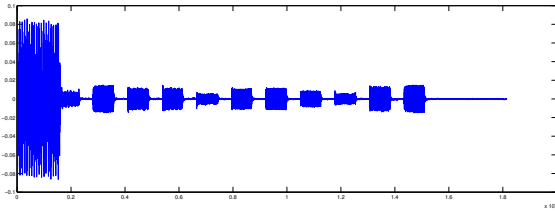


Fig. 4. Waveform of Raw Input Wave

II. PROPOSED APPROACH

The decoder is accomplished following the flow chart in figure 3.

A. Read file

To read the wave file using Matlab, we use the commend:

```
1 [RawWave,Fs]=wavread(WaveFile);%where ...
   WaveFile equals to the string of file name
```

Then we get the wave as a array in RawWave. In our occasion, we tested a real recored wave file 'realrec1.wav' and get its wave form in figure 4

B. Pre-filter

From figure 4 we find that there exits lots of wave with large amplitude that we don't what and can cause trouble to wave

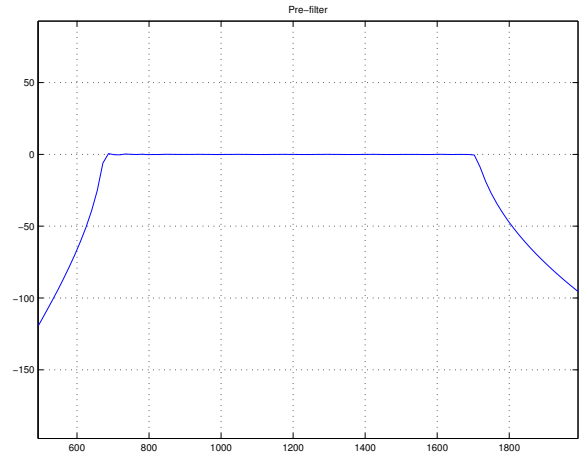


Fig. 5. Raw Wave After the Pre-filter

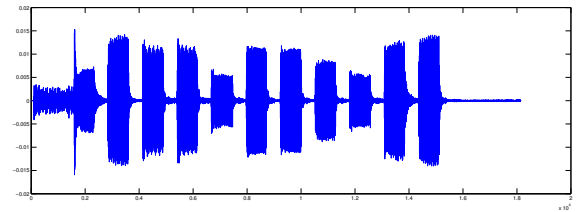


Fig. 6. Pre-filter Frequency Response

analyze. So we need a band pass filter to get ride of them. Using the Matlab code:

```
1 f1=680;f3=1700;
2 fsl=640;fsh=1740;
3 rp=0.1;rs=20;
4 wpl=2*pi*f1/Fs;
5 wp3=2*pi*f3/Fs;
6 wsl=2*pi*fsl/Fs;
7 wsh=2*pi*fsh/Fs;
8 wp=[wpl wp3];
9 ws=[wsl wsh];
10 [n,wn]=cheblord(ws/pi,wp/pi,rp,rs);
11 [bz1,az1]=cheby1(n,rp,wp/pi);
12 FiltedWave=filter(bz1,az1,RawWave);
```

we get the Chebyshev filter in figure 5. Then the filtered wave will be cut off on the low frequency and the unnecessary high frequency, as shown in figure 6.

C. Split multi-digits

After we get the wave that only contains the frequencies components we want, we need to cut it into small, monotonous(or should be 'bitonous' here) pieces so that we can analyze them. To achieve this, we need a function 'WaveSplitter', who takes in the raw wave and gives out an $n \times 2$ array that tell us where the dial tone starts and where

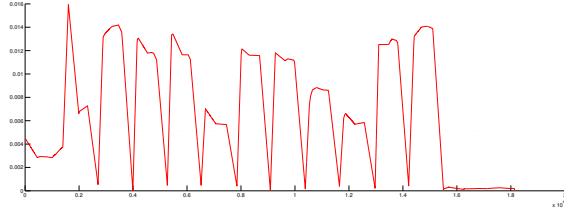


Fig. 7. Envelope of Filtered Raw Wave

it ends.

$$PositionArray = \begin{pmatrix} 1st\ dial & 1st\ dial \\ begin\ point & end\ point \\ \vdots & \vdots \\ \vdots & \vdots \\ nth\ dial & nth\ dial \\ begin\ point & end\ point \end{pmatrix}$$

To get this, we first find the envelope of the signals, as shown in figure 7. Then we calculate the mean of the amplitude. Then scale it with proper value, here we divide it by 1.5, then take it like a threshold value. Find every point on the envelope that cross the threshold value, record it as the beginning or end point of the dialing tone.

D. Decode Single Digits Using FFT

Since we know where the monotonous signal begins and ends, we can analyze them through different ways. The first way of approaching is to use FFT.

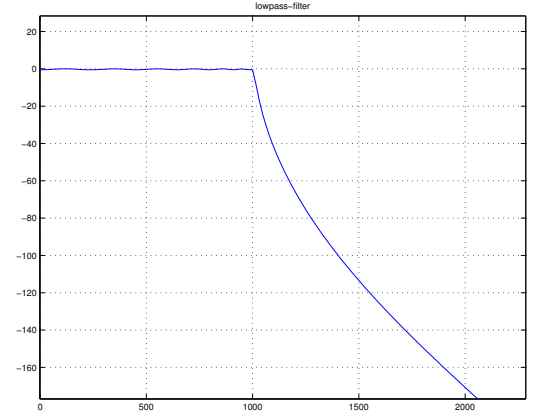
The FFT of the signal will have two peaks, to identify them, we first need to filter the signal through two different filters, lowpass filter at 1000Hz and conjugating high pass filter, as shown in figure 8. After this, we do FFT to the two signals, and now each signal in the frequency domain should have only one peak. Now we can find the maximum of the function to find out the peak value and identify the signal.

E. Decode Single Digits Using Filter Bank

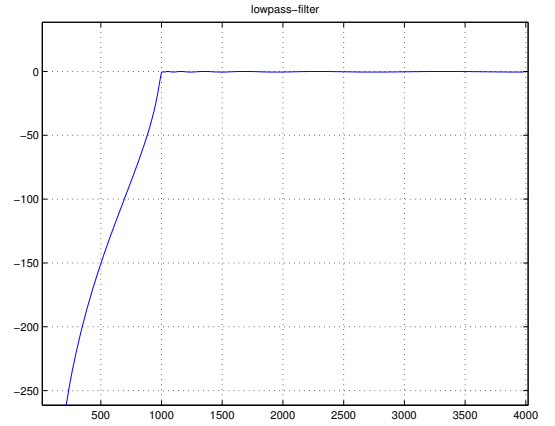
For the filter bank approach, we test the wave through two bandpass filters separately. There are 8 kinds of filters, each of them can filter only one frequency of the DTMF and there are 4×4 combinations. After the signal has passed through the two filters, we add the maximum of the two result signals. At last we will get 16 groups of maximum values, and the maximum of these maximums is the dial tone we want to decide. Figure 9 shows the occasion using the filter band method to identify signal '5'.

III. CONCLUSION

Finally we accomplished the function 'DTMFdecoder'. It has two parameters, the input wave file name and the method



(a) Lowpass Filter



(b) Highpass Filter

Fig. 8. Filter Before FFT

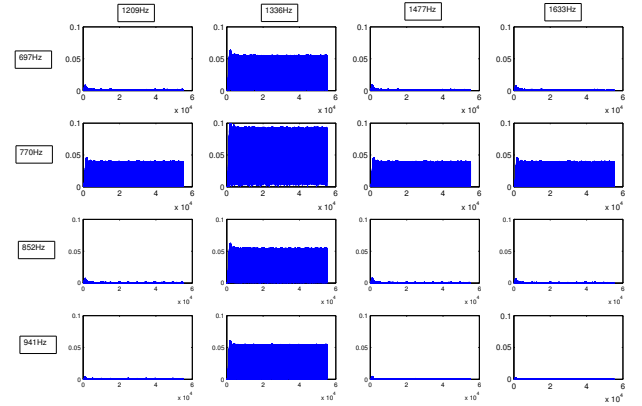


Fig. 9. Filter Band Identify Signal 5

string. The function can be used as:

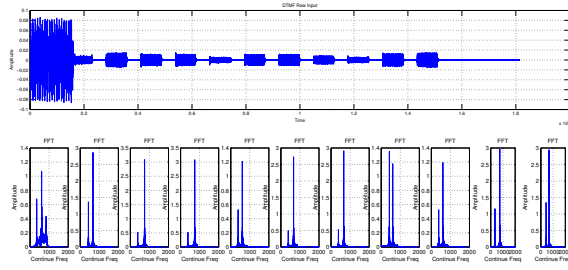


Fig. 10. Filter Band Identify Signal 5

```

1 >>DTMFdecoder('realrec1.wav','FFT'); %for ...
    using FFT method to decode file realrec1.wav
2 >>DTMFdecoder('realrec1.wav','Filter'); %for ...
    using Filter method to decode file ...
    realrec1.wav

```

and the out put will be like:

```

1 >> DTMFdecoder('realrec1.wav','FFT');%for ...
    using FFT method to decode file realrec1.wav
2 The code is 18664557438
3 >>DTMFdecoder('realrec1.wav','Filter'); %for ...
    using Filter method to decode file ...
    realrec1.wav
4 The code is 18664557438

```

Further, if the option 'graph' sets to 'gryphz_on', the input wave will be played back and the waveform and the FFT of every digits will be shown, see figure 10.

```

1 >> DTMFdecoder('realrec1.wav','FFT','graph_on');
2 %for using FFT method to decode file ...
    realrec1.wav with graph on
3 The code is 18664557438

```

APPENDIX A CODE FOR THE WHOLE PORJECT

A. DTMFdecoder.m

```

1 function result = DTMFdecoder(WaveFile,Method,Graph)
2 %read file
3 if nargin==1,Method='FFT';Graph='graph_off';end
4 if nargin==2,Graph='graph_off';end
5 [RawWave,Fs,Bits]=wavread(WaveFile);
6 assert(strcmp(Method,'FFT')||strcmp(Method,'Filter'),...
7 'ERROR: invalid method! The method should be 'FFT' or 'Filter'.');
8 assert(strcmp(Graph,'graph_on')||strcmp(Graph,'graph_off'),...
9 'ERROR: invalid Graph option! it should be 'graph_on' or ...
    'graph_off'.');
10
11 if strcmp(Graph,'graph_on'),sound(RawWave,Fs,Bits);end
12
13 % Prefilterer
14 f1=690;f3=1700;
15 fsl=630;fsh=1800;
16 rp=1;rs=15;
17 wpl=2*pi*f1/Fs;
18 wp3=2*pi*f3/Fs;
19 wsl=2*pi*fsl/Fs;
20 wsh=2*pi*fsh/Fs;
21 wp=[wpl wp3];
22 ws=[wsl wsh];
23 [n,wn]=cheb1ord(ws/pi,wp/pi,rp,rs);

```

```

24 [bz1,az1]=cheby1(n,rp/wp/pi);
25 FiltedWave=filter(bz1,az1,RawWave);
26 FiltedWave(1:100)=0;
27
28 %position signal
29 HalfSampleLength=1024;
30 EffPosition = WaveSplitter(FiltedWave);
31 a=size(EffPosition);
32
33 %draw raw wave
34 if strcmp(Graph,'graph_on')
35
36 set(gcf,'Position',[0 20 1200 500]);
37 set(gca,'Position',[0 0 1 1]);
38 figure_FonSize=8;
39 set(get(gca,'XLabel'),'FontSize',figure_FonSize,'Vertical','top');
40 set(get(gca,'YLabel'),'FontSize',figure_FonSize,'Vertical','middle');
41 set(findobj('FontSize',10),'FontSize',figure_FonSize);
42 set(findobj(get(gca,'Children'),'LineWidth',0.5),'LineWidth',2);
43
44 subplot(2,a(1),1:a(1))
45 plot(RawWave);
46 title('DTMF Raw Input');xlabel('Time');
47 ylabel('Amplitude');grid;
48 end
49
50 for i=1:a(1)
51 %decode single
52 MidSig = ceil((EffPosition(i,1)+EffPosition(i,2))/2);
53 SigBegin = MidSig-ceil((EffPosition(i,2)-EffPosition(i,1))*3/10);
54 SigEnd = MidSig+ceil((EffPosition(i,2)-EffPosition(i,1))*3/10);
55
56 if strcmp(Method,'FFT')
57 result(i) = DTMFdecoder_single_FFT(...
58 FiltedWave(SigBegin:SigEnd),Fs);
59 elseif strcmp(Method,'Filter')
60 result(i) = DTMFdecoder_single_Filter(...
61 FiltedWave(SigBegin:SigEnd),Fs);
62 end
63 %do fft and draw
64 if strcmp(Graph,'graph_on')
65 FFTWave=abs(fft(FiltedWave...
66 (SigBegin:SigEnd),2*HalfSampleLength));
67 subplot(2,a(1),a(1)+i)
68 plot(Fs/length(FFTWave)/2.*(1:length(FFTWave)/2)...
69 ,FFTWave(1:length(FFTWave)/2));
70 title('FFT');xlabel('Continue Freq');
71 ylabel('Amplitude');grid;
72 end
73 end
74
75 disp(['The code is ' result])
76 end

```

B. WaveSplitter.m

```

1 function Result = WaveSplitter(RawWave)
2
3
4 Env = envelope([1:length(RawWave)],abs(RawWave),400,'top');
5 Avr = mean(abs(RawWave))/1.5;
6 ResultRowIndex=1;ResultColIndex2=1;
7 IsDailing = 0;
8 for WaveIndex=1:length(RawWave)
9 if Env(WaveIndex)>Avr
10 if IsDailing==0
11 Result(ResultRowIndex,1)=WaveIndex;
12 IsDailing = 1;
13 end
14
15 else
16 if IsDailing==1
17 Result(ResultRowIndex,2)=WaveIndex;
18 IsDailing=0;
19 ResultRowIndex=ResultRowIndex+1;
20 end
21
22 end
23 end
24 % wave end
25 if IsDailing==1
26 Result(ResultRowIndex,2)=length(RawWave);
27 end
28 end

```

C. envelope.m

```

1 function [env] = envelope(x_data, y_data, view, side)
2 % Function call: env_secant(x_data, y_data, view, side)
3 % Calculates the top envelope of data <y_data> over <x_data>.
4 % Method used: 'secant-method'
5 % env_secant() observes the max. slope of about <view> points,
6 % and joints them to the resulting envelope.

```

```

7 % An interpolation over original x-values is done finally.
8 % <side> ('top' or 'bottom') defines which side to evolve.
9 % Author: Andreas Martin, Volkswagen AG, Germany
10
11 side = strcmpi( {'top','bottom'}, side ) * [ 1 ; -1 ];
12
13 assert( view > 1, ...
14 'Parameter <view> too small!' );
15 assert( ndims( x_data ) == 2, ...
16 'Parameter <x_data> has to be vector type!' );
17 assert( size( x_data, 1 ) == 1 || size( x_data, 2 ) == 1, ...
18 'Parameter <x_data> has to be vector type (Nx1)!' );
19 assert( ndims( y_data ) == 2, ...
20 'Parameter <y_data> has to be vector type (Nx1)!' );
21 assert( size( y_data, 1 ) == 1 || size( y_data, 2 ) == 1, ...
22 'Parameter <y_data> has to be vector type (Nx1)!' );
23 assert( length( x_data ) == length( y_data ), ...
24 'Parameters <x_data> and <y_data> must have same length!' );
25 assert( side ~= 0, ...
26 'Parameter <side> must be ''top'' or ''bottom''' );
27
28 y_data = y_data(:);
29 data_len = length( y_data );
30 x_new = [];
31 y_new = [];
32
33 i = 1;
34 while i < data_len;
35     ii = i+1:min( i + view, data_len );
36     [ m, idx ] = max( ( y_data(ii) - y_data(i) ) ./ (ii-i)' .* side );
37
38     % Equidistant x_data assumed! Use next row instead, if not:
39     % [ m, idx ] = max( ( y_data(ii) - y_data(i) ) ./ ( x_data(ii) - ...
40         x_data(i) ) * side );
41
42     % New max. slope: store new "observation point"
43     i = i + idx;
44     x_new = [ x_new x_data(i) ];
45     y_new = [ y_new y_data(i) ];
46 end;
47
48 env = interp1( x_new, y_new, x_data, 'linear', 'extrap' );

```

D. DTMFdecoder_single_FFT.m

```

1 function ResultCode = DTMFdecoder_single_FFT(PureWave,Fs)
2 Rp=0.5;
3 As=25;
4 Fpass = 1000;
5 Fstop = 1100;
6 wp = 2*pi*Fpass/Fs;
7 ws = 2*pi*Fstop/Fs;
8 [N,Wn]=cheblord(wp/pi,ws/pi,Rp,As);
9
10 [b1,a1]=cheby1(N,Rp,Wn);
11 RawWaveFilt_L=filter(b1,a1,PureWave);
12 FFTWave_L=fft(RawWaveFilt_L,length(PureWave));
13 FFTWave_L_Mag=abs(FFTWave_L(:cell(length(PureWave)/2)));
14
15 [bh,ah]=cheby1(N,Rp,Wn,'high');
16 RawWaveFilt_H=filter(bh,ah,PureWave);
17 FFTWave_H=fft(RawWaveFilt_H,length(PureWave));
18 FFTWave_H_Mag=abs(FFTWave_H(:cell(length(PureWave)/2)));
19
20 m=max(abs(FFTWave_L_Mag)); n=max(abs(FFTWave_H_Mag));
21 o=find(m==FFTWave_L_Mag); p=find(n==FFTWave_H_Mag);
22 j=((o-1)*Fs)/length(PureWave);
23 k=((p-1)*Fs)/length(PureWave);
24
25 ResultCode = '';
26 if j<732.59 & k<1270.91;
27     ResultCode = '1';
28 elseif j<732.59 & k<1404.73;
29     ResultCode = '2';
30 elseif j<732.59 & k<1553.04;
31     ResultCode = '3';
32 elseif j<732.59 & k>1553.05;
33     ResultCode = 'A';
34 elseif j<809.96 & k<1270.91;
35     ResultCode = '4';
36 elseif j<809.96 & k<1404.73;
37     ResultCode = '5';
38 elseif j<809.96 & k<1553.04;
39     ResultCode = '6';
40 elseif j<809.96 & k>1553.05;
41     ResultCode = 'B';
42 elseif j<895.39 & k<1270.91;
43     ResultCode = '7';
44 elseif j<895.39 & k<1404.73;
45     ResultCode = '8';
46 elseif j<895.39 & k<1553.04;
47     ResultCode = '9';
48 elseif j<895.39 & k>1553.05;
49     ResultCode = 'C';
50 elseif j>895.40 & k<1270.91;
51     ResultCode = '+';
52 elseif j>895.40 & k<1404.73;
53     ResultCode = '0';

```

```

54 elseif j>895.40 & k<1553.04;
55     ResultCode = '#';
56 elseif j>895.40 & k>1553.05;
57     ResultCode = 'D';
58 end
59 end

```

E. DTMFdecoder_single_Filter.m

```

1 function ResultCode = DTMFdecoder_single_Filter(PureWave,Fs)
2 DTMFCell=...
3 { ...
4     'Tone1/Tone2' '1209' '1336' '1477' '1633';...
5     '697' '1' '2' '3' 'A';...
6     '770' '4' '5' '6' 'B';...
7     '852' '7' '8' '9' 'C';...
8     '941' '+' '0' '#' 'D'...
9 };
10 Code=DTMFCell( z_end,z_end );
11 Tone1=cellfun(@(x) str2num(x), DTMFCell(1,z_end));
12 Tone2=cellfun(@(x) str2double(x), DTMFCell(z_end,1));
13 [ToneMat1 ToneMat2] =ndgrid(Tone1,Tone2);
14 k=1;
15 MaxAmplitude=zeros(numel(Tone1)*numel(Tone2),1);
16
17 for TonePair=[ToneMat1(:)';ToneMat2(:)']
18     if TonePair(1)~=TonePair(2)
19         zewave = PureWave;
20         fal=TonePair(1)-20;fa3=TonePair(1)+20;%
21         fsal=TonePair(1)-40;fsah=TonePair(1)+40;%
22         rpa=0.1;rsa=20;%
23         wpa1=2*pi*fal/Fs;
24         wpa3=2*pi*fa3/Fs;
25         wsa1=2*pi*fsal/Fs;
26         wsa3=2*pi*fsah/Fs;
27         wpa=[wpa1 wpa3];
28         wsa=[wsa1 wsa3];
29         [na,wna]=cheblord(wsa/pi,wpa/pi,rpa,rsa);
30         [bza1,aza1]=cheby1(na,rpa,wpa/pi);
31         FiltedWave_1=filter(bza1,aza1,zewave);
32
33         fbl=TonePair(2)-20;fb3=TonePair(2)+20;%
34         fsbl=TonePair(2)-40;fsbh=TonePair(2)+40;%
35         rpb=0.1;rsb=20;%
36         wpb1=2*pi*fbl/Fs;
37         wpb3=2*pi*fb3/Fs;
38         wsb1=2*pi*fsbl/Fs;
39         wsbh=2*pi*fsbh/Fs;
40         wpb=[wpb1 wpb3];
41         wsb=[wsb1 wsbh];
42         [nb,wnb]=cheblord(wsb/pi,wpb/pi,rpb,rsb);
43         [bzb1,azb1]=cheby1(nb,rpb,wpb/pi);
44         FiltedWave_2=filter(bzb1,azb1,zewave);
45
46         MaxAmplitude(k)=max(abs(FiltedWave_1))+max(abs(FiltedWave_2));
47     else
48         MaxAmplitude(k)=0;
49     end
50     k=k+1;
51 end
52 [val Indx]=max(MaxAmplitude);
53
54 ResultCode=Code(Indx);
55 end

```

REFERENCES

- [1] A. V. Oppenheim and R. W. Schaffer, *Discrete-Time Signal Processing*, 3rd ed. Pearson Higher Education, Inc., Upper Saddle River: 2010.
- [2] bertrand.removethis.(2008,July,12) *detect frequency of dtmf tone (.wav) file*, url=<http://compgroups.net/comp.soft-sys.matlab/detect-frequency-of-dtmf-tone-wav-file/957018>.
- [3] bertrand.removethis.(2008,July,12) *detect frequency of dtmf tone (.wav) file*, url=<http://compgroups.net/comp.soft-sys.matlab/detect-frequency-of-dtmf-tone-wav-file/957018>.
- [4] Can Yi Tian Shi.(2011,Feb,22) *Sharing Matlab Program — Filter*, url=http://blog.sina.com.cn/s/blog_574d08530100qfrb.html.