Project 1: Speech Synthesis and Perception with Envelope Cue

Author	Student Name & ID: 万芃（12011615）、王卓扬（12112907）、刘谦益（12011812）、冯彦捷（12010825）

Introduction

The speech waveform can be decomposed into the product of an *envelope* and a carrier (*fine-structure*), which are shown in *Figure 1*. This process is called *Hilbert Transformation*.

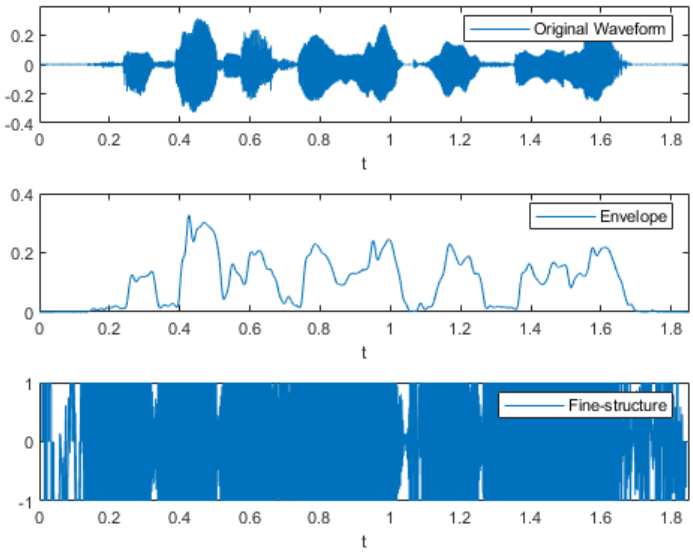

Figure 1: Hilbert Transformation by MATLAB

Studies have shown that, most of the effective information of speech is contained in the envelope waveform. Moreover, the closer the energy frequency distribution of the fine-structure is to the characteristics of the speech spectrum, the more human-voice-like and more clearly recognizable the restored audio is.

Common communication devices have their own unique transmit frequencies when transmitting signals, and they usually cannot ideally handle signals with continuous spectrum. However, the energy distribution of the human voice waveform in the frequency domain is relatively continuous. So directly transmitting without any processing will inevitably lead to distortion.

To handle this problem is the objective of our project. The basic idea to construct a digital speech synthesizer, which is shown in *Figure 2*.

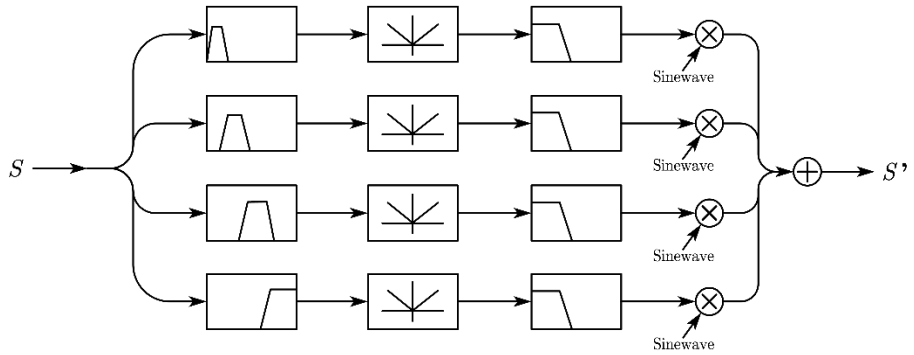

Figure 2: The Speech Synthesizer System

The system uses several band-pass filters to cut the frequency spectrum of the sound signal into segments and takes the midpoint frequency of each segment as the sub-carrier (fine-structure) frequency. Then it finally synthesizes all the sub-waveforms by using sinusoidal carriers. Abstractly speaking, *it is a finite sampling of the human voice spectrum*.

In this way, the components of the signal can be transmitted through different communication devices that are stable in separated frequency bands.

Let's discuss the general processing flow (*Figure 2*).

Firstly, the 200 ~ 7000Hz frequency band is sampled at equal distances according to the relationship between the distance and frequency in the cochlea (formula below), and is divided into N segments. Using the function *butter*, the corresponding band-pass filter parameters are generated according to the sampling points, and the signals of each frequency band are filtered out.

$$f = 165.4 \times (10^{0.06 \cdot d} - 1)$$

Next, the envelope of the speech signal is extracted. First, the function *abs* is used to perform full-wave rectification on each signal component, and then the low-pass filter constructed by the function *butter* is used to remove the high-frequency carrier component, leaving the envelope waveform containing the speech information.

Finally, each envelope waveform is multiplied by an ideal sinusoidal carrier, and the summation is obtained to obtain a synthesized speech signal. Finally, energy normalization is required to make the energy of the final signal equal to the input signal.

In the latter two tasks, it is also necessary to superimpose a piece of speech-shaped noise on the original signal. We normalize the white noise according to the frequency energy distribution characteristics of the human voice to obtain the speech-shaped noise signal and superimpose it on the original signal according to the specified signal-to-noise ratio, and then perform the above process.

Here comes the specific implementation. See the appendix section for the completed code.

Hilbert Transformation

The voice spectrum information is obtained by taking the envelope, retaining the time characteristics and amplitude characteristics, discarding the unnecessary sound spectrum

information, and obtaining the more concise and recognizable speech signal of the sound spectrum.

In R.V.Shannon's article on analyzing speech recognition, proposed methods for processing speech signals by extracting time-domain features to discard unnecessary spectral information. In the past, good results have been obtained by reducing the compression amplitude or the spectral information, but Shannon pointed out that these two treatments still cannot be streamlined for the stimulus causes of the complex time spectrum. He used the frequency information instead of the acoustic spectrum information, which can study the characteristics of the acoustic spectrum while retaining the temporal characteristics and the amplitude characteristics, and thus provides a theoretical model of the cochlear implant.

Now let's have a look at the implementation of the *Hilbert transform*.

This part is divided into two steps, we write the function *extractEnvelope* to implement it (s and fs represent the input signal with a sampling frequency):

```
function S = extractEnvelope(s, fs, fcut, order)
```

First pass the signal through a rectifier, here using full-wave rectification:

```
s0 = abs(s);
```

Next, construct a low-pass Butterworth filter, given the cutoff frequency as *fcut*, and the order of the filter as *order*

```
[b_low, a_low] = butter(order, fcut / (fs / 2));
```

Then use it to filter out the high frequency carrier signal in the signal:

```
S = filter(b_low, a_low, s0);
```

That's how we obtain the result envelope waveform *S*.

Speech Synthesis

Speech synthesis is a relatively complete functional module. We write a function synthesis to realize it, and the number of divided frequency bands *bandNum* and the cut-off frequency *fcut* of the low-pass filter can be changed through parameters (*snr* is the signal-to-noise ratio when the noise is superimposed on the latter two tasks):

```
function S_res = synthesis(s, fs, bandNum, snr, fcut)
```

First we sort out and define the basic quantities (see notes for definition):

```
N = bandNum;
lowF = 200; % The minimum frequency of the bands
highF = 7000; % The maximum frequency of the bands
bandPassOrder = 3; % The order of band-pass filters
lowPassOrder = 2; % The order of the low-pass filter
SNR = snr; % Signal-noise ratio in dB
cutF = fcut; % The cut-off frequency of the low-pass filter
L = length(s); % The length of the signal vector
```

Then use the *getNoisedS* function to superimpose the speech-shaped noise (see the next section, temporarily omitted):

```
S = getNoisedS(s, fs, SNR);
```

Before processing each frequency band, we first write the *getFpByCl* function to solve the frequency sampling points under the same cochlear spacing (Seen in *Appendix*).

Then there is the processing of each frequency band. According to the above process, construct and apply a band-pass filter in turn, find the center point of the frequency band, extract the envelope waveform, and finally multiply the sinusoidal signal corresponding to the center frequency in the time domain (Seen in *Appendix*).

Finally, superposition of wavelets and normalization of energy are performed (The splicing of all-zero rows is to prevent function sum from summing all columns when the number of bands is 1):

```
S_res = sum([S_rescomp; zeros(1, L)]);
S_res = S_res / norm(S_res) * norm(S);
```

Speech-shaped Noise

Finally, there is the generation of speech-shaped noise. See *Lab5* for the specific principle, which will not be repeated here.

First, the frequency energy distribution of the vocal signal sample is obtained, and the corresponding filter is constructed accordingly:

```
% Power Spectral Density
[Pxx, w] = pwelch(repmat(s, 1, 10), [], [], 512, fs);

% Generate coefficients for the filter
b_ssn = fir2(3000, w / (fs / 2), sqrt(Pxx / max(Pxx)));
```

Generate white noise and feed it into a filter to get speech-shaped noise (The truncation operation is to eliminate the effect of filter delay on the result):

```
whiteNoise = 1 - 2 * rand(1, length(s) + length(b_ssn) - 1);
ssn = filter(b_ssn, 1, whiteNoise);
ssn = ssn(length(b_ssn) : end);
```

Finally, modulate its energy according to the given signal-to-noise ratio and add it to the original signal:

```
ssn = ssn / norm(ssn) * norm(s) * 10 ^ (-snr / 20);
S = s + ssn;
```

Results and Analysis

About the Recognition Experiment

When estimating the effect of speech restoration, the task requires us to look at the number of words we can hear. However, considering that being too familiar with voices will affect our judgment, and changing to an unfamiliar voice every time will destroy the principle of controlling variables, we consider using *Google Translator*'s voice recognition function to conduct recognition experiments.

That is, using the same volume on the same machine, playing each audio segment three times, using *Google Translator* to recognize, and recording the number of recognized words, as well as the number of words recognized but incorrectly recognized, and drawn into a chart for judging the effect of speech synthesis. We think this method has less error.

Some Figures for Supplementary

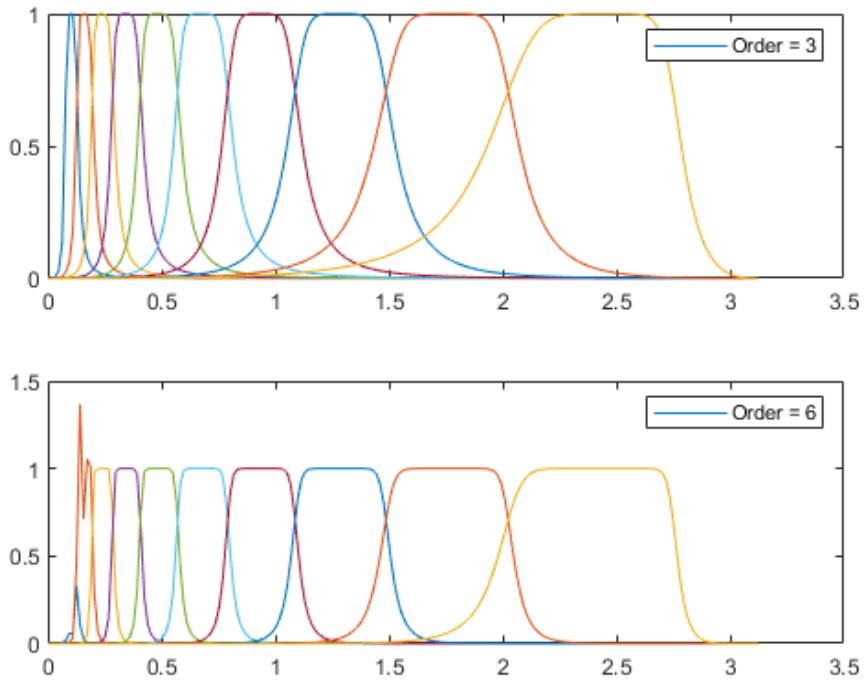

Figure 3: Band-pass Filters' Frequency Responses When N = 10, Order = 3, 6

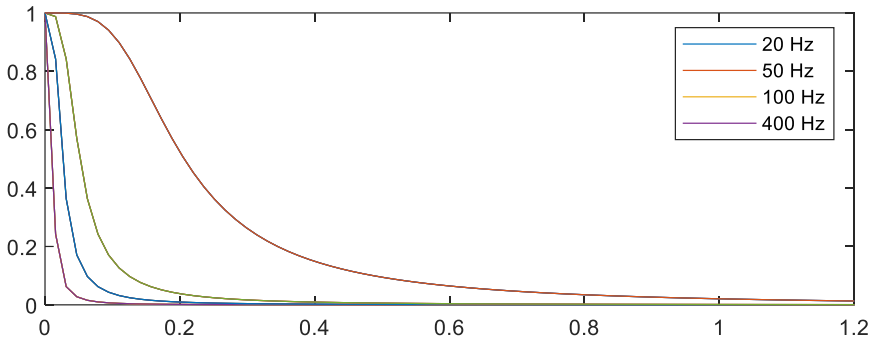

Figure 4: Low-pass Filters' Frequency Responses

To examine the filters, we plotted their frequency response. And we won't show it in tasks again.

Task 1

Restatement

1. Set LPF cut-off frequency to 50 Hz.
2. Implement tone-vocoder by changing the number of bands to N = 1, 2, 4, 6 and 8.
3. Save the wave files for these conditions, and describe how the number of bands affects the intelligibility (i.e., how many words can be understood) of synthesized sentence.

Analysis

Results are shown in *Figure 5* and *Figure 6*. In *Figure 5* is the spectrum of the results in both time domain and frequency domain when *N* = 1, 2, 4, 6, 8. In *Figure 6* is the result of the recognizing experiment, which is explained before.

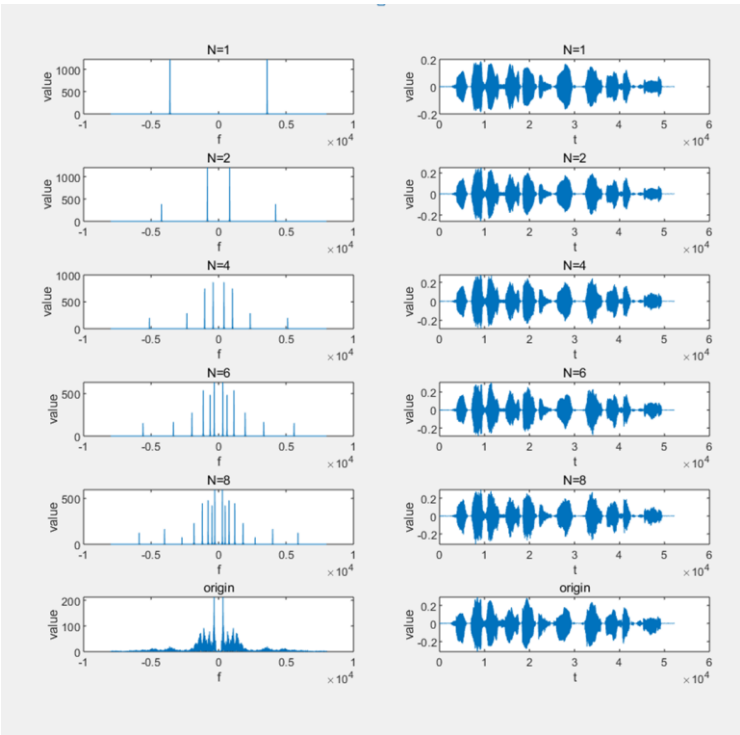

Figure 5: Task1 - Spectrum in Time Domain and Frequency Domain, N = 1, 2, 4, 6, 8

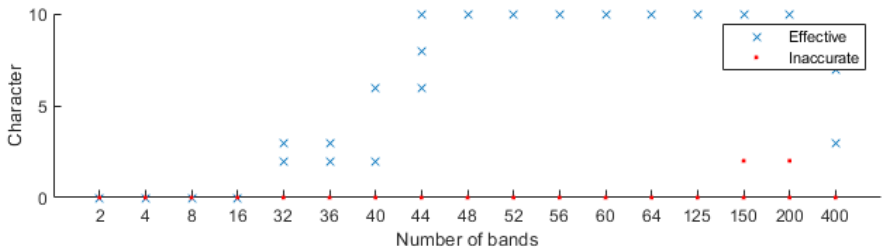

Figure 6: Task1 - The Result of the Recognizing Experiment, N = 2, …, 400

It can be seen that the larger the value of *N*, the closer the image and the original audio signal image, and the same is true, when we play the processed signal, it is obvious that the larger the *N* value, the clearer the sound heard.

Task 2

Restatement

1. Set the number of bands N = 4.

2. Implement tone-vocoder by changing the LPF cut-off frequency to 20 Hz, 50 Hz, 100 Hz, and 400 Hz.

3. Describe how the LPF cut-off frequency affects the intelligibility of synthesized sentence.

Analysis

Results are shown in *Figure 7*. In *Figure 7* is the spectrum of the results in both time domain and frequency domain when *fcut* = 20, 50, 100, 400 Hz. It is easier to hear the speech when *f* = 100 or 400 Hz. When frequency increase, we are easier to identify the speech. From the plot we can also see that higher the frequency, more similar the plot and the origin.

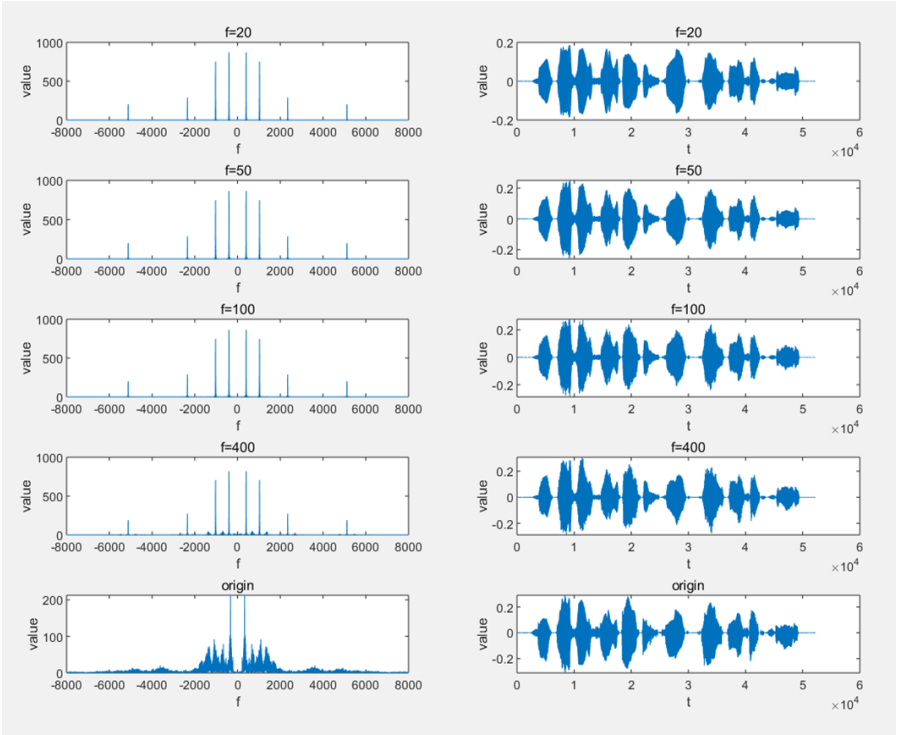

Figure 7: Task2 - Spectrum in Time Domain and Frequency Domain, fcut = 20, 50, 100, 200 Hz

Task 3

Restatement

1. Generate a noisy signal (summing clean sentence and SSN) at SNR -5 dB.

2. Set LPF cut-off frequency to 50 Hz.

3. Implement tone-vocoder by changing the number of bands to N = 2, 4, 6, 8 and 16.

4. Describe how the number of bands affects the intelligibility of synthesized sentence, and compare findings with those obtained in *task 1*.

Analysis

Results are shown in *Figure 8* and *Figure 9*. In *Figure 8* is the spectrum of the results in both time domain and frequency domain when *N* = 2, 4, 6, 8, 16. In *Figure 9* is the result of the recognizing experiment.

And from the result, we can find that, first, in a horizontal comparison, the signal after adding noise is more difficult to hear, and a higher number of frequency bands is required to achieve the effect that the lower number of frequency bands in *task 1* can achieve.

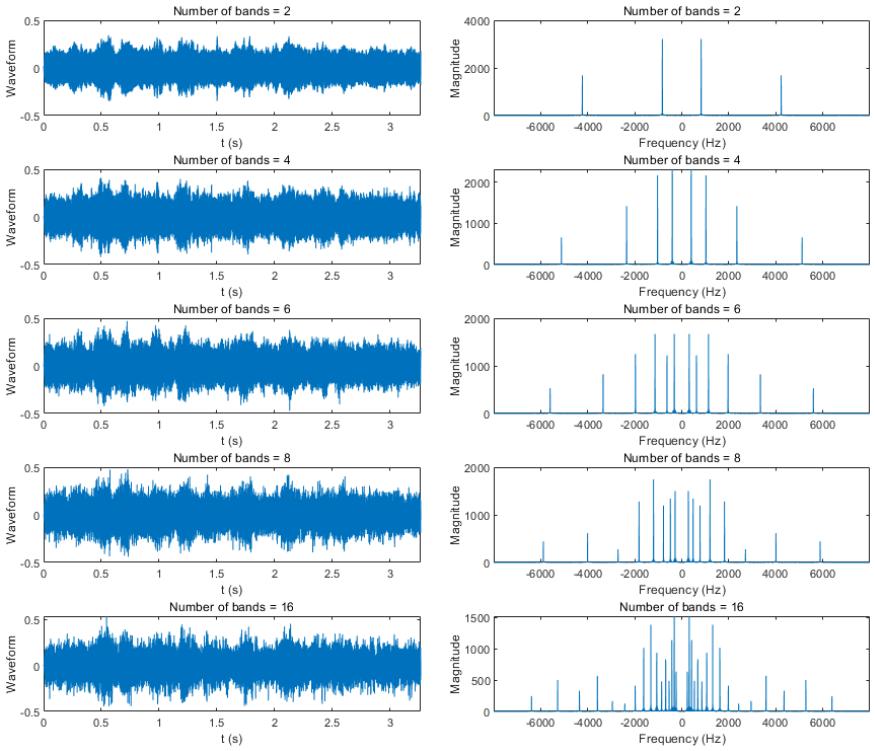

Figure 8: Task3 - Spectrum in Time Domain and Frequency Domain, N = 2, 4, 6, 8, 16

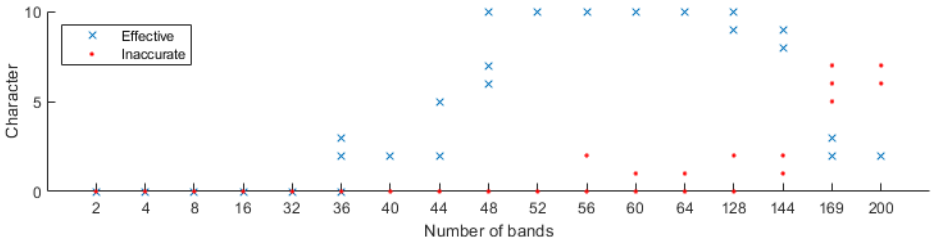

Figure 9: Task3 - The Result of the Recognizing Experiment, N =2, …, 200

Second, we can see from the recognition experiment that when the number of frequency bands is small, the more frequency bands, the better the restoration effect; but when the number of frequency bands is large, the number of words recognized by the computer does not decrease too much, but the error rate increases rapidly. After manual audition, we found that it was the noise that showed a great interference effect at high frequency bands, which became very strange and affected the judgment of the machine, but people can still clearly hear the text information behind it.

Task 4

Restatement

1. Generate a noisy signal (summing clean sentence and SSN) at SNR -5 dB.
2. Set the number of bands to N = 6.
3. Implement tone-vocoder by changing the LPF cut-off frequency to 20 Hz, 50 Hz, 100 Hz, and 400 Hz.
4. Describe how the LPF cut-off frequency affects the intelligibility of synthesized sentence.

Analysis

Results are shown in *Figure 10*. In *Figure 10* is the spectrum of the results in both time domain and frequency domain when *fcut* = 20, 50, 100, 400 Hz. When frequency increase, we are easier to identify the speech. From the plot we can also see that higher the frequency, more similar the plot and the origin.

But due to the existence of the speech-shaped noise, which matters too much that all the results are hard to identify comparing to tasks before.

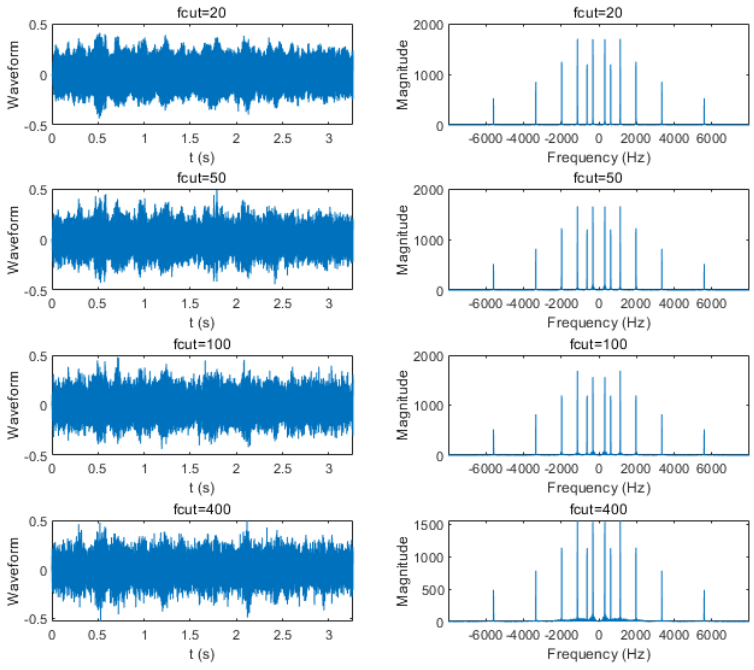

Figure 10: Task4 - Spectrum in Time Domain and Frequency Domain, fcut = 20,50,100,200 Hz

Conclusion

1. Changing *N*.

Too low *N* will lead to the result from which we can hear nothing; Appropriate *N* led to better performance, and when *N* is increasing the sound also becomes clearer; Too high *N* will lead to a weird noise which can influence our judgement.

2. Changing Cut-off Frequency of Low-pass Filters.

Too low cut-off frequency will lead to distortion of the envelope (losing information); Higher *fcut* can make it easier to identify the sound.

3. Adding a Speech-shaped Noise.

The noise affects a lot. -5 dB *SSN* makes that we should use larger *N* to generate the effective results.

Extra Thoughts

1. Changing the order of the band-pass filters.

Lower order makes the result blurrier; Too high order will make it easier to destruct the result especially when N is large too *Figure 11*.

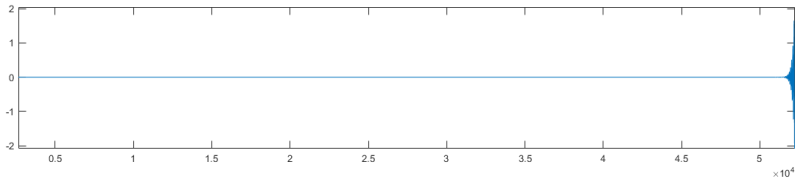

Figure 11: When Band-pass Filters' Order = 6 and N = 40

2. Changing the order of the low-pass filters.

That is a little bit similar to change the cut-off frequency.

Extension

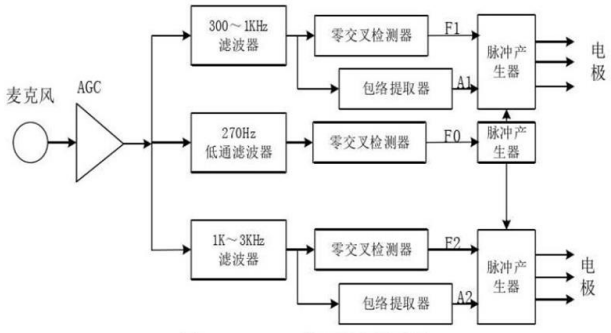

Figure 12: F0/F1/F2 Processing Methods

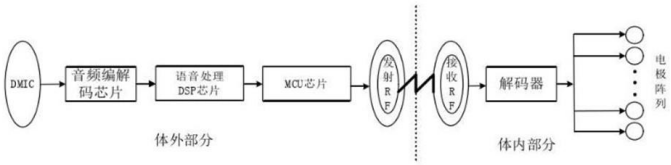

Figure 13: Hardware Sketech

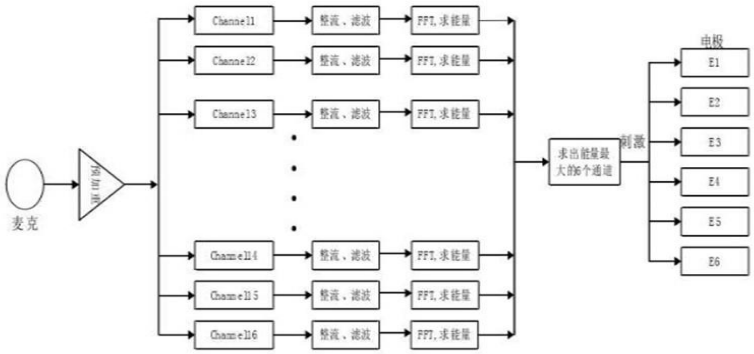

Figure 14: SMSP Processing Method

The principle of artificial cochlea. The audio decoder can transform voice to signal. Then the voice processor will process it and deliver to MCU.

Many different solutions can be used to process the voice signal.[1]

F0 / F1 / F2

Voice process solution based on feature extraction.

Voice signal go through filter of different band, then the output go through a zero-cross detector. Have another resonance peak frequency to extracted, and similar to F0/F2.

MPEAK

Increase the band to 800 ~ 4000Hz, and extract an extra high frequency signal.

Voice process solution based on group of filter.

Using alternating current impulse to stimulate electrode and avoid interference.

A series of band-pass filters are applied to process the high frequency which is separated into different pings. Then using low-pass filter to extract the envelop. Using FFT to transform the frequency band. Then choosing the 6 channels have the biggest energy as the stimulate signal.

SPEAK

Using more channels than SMSP.

[1]王旭. 电子耳蜗音频信号处理系统设计[D]. 西安电子科技大学, 2017.

Experience

Contributors

Task 1 - Liu Qianyi, Task 2 -Wan Peng, Task 3 - Wang Zhuoyang, Task 4 - Feng Yanjie.

Extension - Wan Peng.

Report - Wang Zhuoyang, Liu Qianyi, Wan Peng.

Slides - Wang Zhuoyang, Feng Yanjie, Liu Qianyi.

Experience

1. Code: We have a deeper understanding of what we learned in class, such as envelope extracting, filtering, cut-off frequency and so on. For example, sample frequency for human's speaking is normally 8000Hz. In addition, understanding the structure and function of the cochlear implant also facilitates us to write the corresponding code.

2. Team work: Obviously, the team project is divided into two parts: code writing for task 1 to 4, and making slides and reports. The first meeting in Week 10 built us a collaboration atmosphere. The following three discussion greatly promoted the progress of the project. We hope to continue this positive communication in the next cooperation.

Score	99

Appendix

Programs – Fundamental

getNoisedS.m

```
% Add Speech-shaped Noise to the Raw Signal
function S = getNoisedS(s, fs, snr)
    % Generate Filter for SSN Generating
    [Pxx, w] = pwelch(repmat(s, 1, 10), [], [], 512, fs); % Power Spectral
Density
    b_ssn = fir2(3000, w / (fs / 2), sqrt(Pxx / max(Pxx))); % Generate
coefficients

    % Obtain SSN
    whiteNoise = 1 - 2 * rand(1, length(s) + length(b_ssn) - 1);
    ssn = filter(b_ssn, 1, whiteNoise);
    ssn = ssn(length(b_ssn) : end);

    % Adjust SNR and Obtain Jittered Signal S
    ssn = ssn / norm(ssn) * norm(s) * 10 ^ (-snr / 20);
    % disp(20 * log10(norm(s) / norm(ssn))); % Test the value
    S = s + ssn;
end
```

getFpByCl.m

```
% Get Frequency Points By Equal Cochlea Lengths
function freqPoints = getFpByCl(bandNum, lowF, highF)
    % Convertors
    getF = @(d) 165.4 .* (10 .^ (0.06 .* d) - 1);
    getD = @(f) log10(f ./ 165.4 + 1) ./ 0.06;
```

```
    % Calculate Distance Distribution
    lowD = getD(lowF);
    highD = getD(highF);
    distPoints = linspace(lowD, highD, bandNum + 1);

    % Return Frequency Distribution
    freqPoints = getF(distPoints);
end
```

extractEnvelope.m

```
% Extract the Envelope of The Signal by Rectifying and Low-pass Filtering
function S = extractEnvelope(s, fs, fcut, order)
    % Full-wave Rectification
    s0 = abs(s);

    % Generate Low-pass Filter
    [b_low, a_low] = butter(order, fcut / (fs / 2));

    % Filtering
    S = filter(b_low, a_low, s0);
end
```

synthesis.m

```
% The Synthesis Procedure
function S_res = synthesis(s, fs, bandNum, snr, fcut)
    % Basic Parameters
    N = bandNum;
    lowF = 200;
    highF = 7000;
    bandPassOrder = 5;
    lowPassOrder = 2;
    SNR = snr; % dB
    cutF = fcut;
    L = length(s);

    % Obtain Noised Signal S
    S = getNoisedS(s, fs, SNR);

    % Band Processing
    Fp = getFpByCl(N, lowF, highF); % Get Frequency Points
    for I = 1 : N
        [b_band, a_band] = butter(bandPassOrder, [Fp(I), Fp(I + 1)] / (fs /
2)); % Generating Band-pass Filters
```

```
        Fm = (Fp(I) + Fp(I + 1)) / 2; % Middle Frequency Points
        S_comp = filter(b_band, a_band, S); % Filtering the Signal
        S_enve = extractEnvelope(S_comp, fs, cutF, lowPassOrder); %
Extracting the Envelope
        S_rescomp(I, :) = S_enve .* sin(2 * pi * Fm .* linspace(0, L / fs,
L)); % Multiplying Sinusoidal Signals
    end

    % Synthesis and Energy Normalization
    S_res = sum([S_rescomp; zeros(1, L)]);
    S_res = S_res / norm(S_res) * norm(S);
end
```

Programs - Tasks

Task 1

```
clc;
clear;
[y,fs]=audioread("C_01_02.wav");
%N=[1,2,4,6,8];
out1=task1(1,y,fs,50);
out2=task1(2,y,fs,50);
out3=task1(4,y,fs,50);
out4=task1(6,y,fs,50);
out5=task1(8,y,fs,50);
out6=y;
subplot(2,3,1),plot(out6),title('origin'),ylabel('value'),xlabel('t');
subplot(2,3,2),plot(out1),title('N=1'),ylabel('value'),xlabel('t');
subplot(2,3,3),plot(out2),title('N=2'),ylabel('value'),xlabel('t');
subplot(2,3,4),plot(out3),title('N=4'),ylabel('value'),xlabel('t');
subplot(2,3,5),plot(out4),title('N=6'),ylabel('value'),xlabel('t');
subplot(2,3,6),plot(out5),title('N=8'),ylabel('value'),xlabel('t');


function output=task1(N,y,fq,cut)
output=0;
[m0,n0]=butter(4,cut/fq*2);
time=1:size(y,1);
[a]=task11(N);
for j=1:N
    [m,n]=butter(4,[a(j) a(j+1)]/(fq/2));
    output=filter(m,n,y);
    mid=abs(output);
    outputmid=filter(m0,n0,mid);
    A=0.5*(a(j)+a(j+1));
```

```
    outputsin=sin(2*pi*A*time*1/fq);
    outputmid=outputmid.*outputsin';
    output=output+outputmid;
end
output=output/norm(output)*norm(y);
end


function [out]=task11(N)% distance to frequency
out=zeros(1,N+1);
lg2=(log10((200/165.4)+1))/0.06;
lg7=(log10((7000/165.4)+1))/0.06;
L1=(lg7-lg2)/N;
out(1)=200;
out(N+1)=7000;
for x=2:N
    out(x)=165.4*(10^(0.06*(L1*(x-1)+lg2))-1);
end
end
```

Task 2

```
clc;
clear;
[y,fs]=audioread("C_01_01.wav");
%f=[20,50,100,400];
L = length(y);
t = L / fs;
lf = (1 - L) / t / 2;
rf = (L - 1) / t / 2;
out1=fftshift(abs(fft(task1(4,y,fs,20))));
out2=fftshift(abs(fft(task1(4,y,fs,50))));
out3=fftshift(abs(fft(task1(4,y,fs,100))));
out4=fftshift(abs(fft(task1(4,y,fs,400))));
out6=fftshift(abs(fft(y)));

subplot(6,2,1),plot(linspace(lf, rf, L),out1),title('f=20');
ylabel('value'),xlabel('f');
subplot(6,2,2),plot(task1(1,y,fs,50)),title('f=20');
ylabel('value'),xlabel('t');
subplot(6,2,3),plot(linspace(lf, rf, L),out2),title('f=50');
ylabel('value'),xlabel('f');
subplot(6,2,4),plot(task1(2,y,fs,50)),title('f=50');
ylabel('value'),xlabel('t');
```

```
subplot(6,2,5),plot(linspace(lf, rf,
L),out3),title('f=100'),ylabel('value'),xlabel('f');
subplot(6,2,6),plot(task1(4,y,fs,50)),title('f=100');
ylabel('value'),xlabel('t');
subplot(6,2,7),plot(linspace(lf, rf,
L),out4),title('f=400'),ylabel('value'),xlabel('f');
subplot(6,2,8),plot(task1(6,y,fs,50)),title('f=400');
ylabel('value'),xlabel('t');

subplot(6,2,11),plot(linspace(lf, rf,
L),out6),title('origin'),ylabel('value'),xlabel('f');
subplot(6,2,12),plot(y),title('origin'),ylabel('value'),xlabel('t');

function output=task1(N,y,fq,cut)
y=y';
output=zeros(1, length(y));
[m0,n0]=butter(4,cut/fq*2);
time=1:length(y);
[a]=task11(N);
for j=1:N
    [m,n]=butter(4,[a(j) a(j+1)]/(fq/2));
    output1=filter(m,n,y);
    mid=abs(output1);
    outputmid=filter(m0,n0,mid);
    A=0.5*(a(j)+a(j+1));
    outputsin=sin(2*pi*A*time*1/fq);
    outputmid=outputmid.*outputsin;
    output=output+outputmid;
end
output=output./norm(output).*norm(y);
end

function [out]=task11(N)
out=zeros(1,N+1);
lg2=(log10((200/165.4)+1))/0.06;
lg7=(log10((7000/165.4)+1))/0.06;
L1=(lg7-lg2)/N;
out(1)=200;
out(N+1)=7000;
for x=2:N
    out(x)=165.4*(10^(0.06*(L1*(x-1)+lg2))-1);
end
end
```

Task 3

```
clear;

%% Basic Parameters
filename = "C_01_01.wav";

%% Load Raw Signal
[y_wav, fs_wav] = audioread(filename);
s = y_wav';
fs = fs_wav;

L = length(s);
t = L / fs;
lf = (1 - L) / t / 2;
rf = (L - 1) / t / 2;

%% Task 3
range = [2, 4, 6, 8, 16];
for bandNum = range
    S = synthesis(s, fs, bandNum, -5, 50);

    subplot(length(range), 2, find(range == bandNum) * 2 - 1);
    plot(linspace(0, L / fs, L), S);
    xlim([0, L / fs]), xlabel("t (s)"), ylabel("Waveform");
    title("Number of bands = " + num2str(bandNum));
    subplot(length(range), 2, find(range == bandNum) * 2);
    plot(linspace(lf, rf, L), abs(fftshift(fft(S))));
    xlim([lf, rf]), xlabel("Frequency (Hz)"), ylabel("Magnitude");
    title("Number of bands = " + num2str(bandNum));
end
```

Task 4

```
clear;
[y,fs]=audioread("C_01_01.wav");
s=y';
L=length(s);
t=L/fs;
r=[20,50,100,400];
for lowpassF=r
    S = synthesis(s,fs,6,-5,lowpassF);
    subplot(length(r),2,find(r==lowpassF)*2-1);
    plot(linspace(0,L/fs,L), S);
    xlim([0,L/fs]),xlabel("t (s)"), ylabel("Waveform");
    title("fcut="+num2str(lowpassF));
```

```
    subplot(length(r),2,find(r==lowpassF)*2);
    plot(linspace((1-L)/t/2,(L-1)/t/2,L),abs(fftshift(fft(S))));
    xlim([(1-L)/t/2,(L-1)/t/2]), xlabel("Frequency (Hz)"),
ylabel("Magnitude");
    title("fcut="+num2str(lowpassF));
end
```