Musical Signal Processing

Our project is divided into two parts, first part is called Karaoke Making, and the second part is called Notes Recognition.

1.Vocal Elimination is the first step of the Karaoke Making process. Typically, when recording the album, we are taking the following way: first record the vocal to a mono track and then insert it into a stereo track, and then we have a complete song, noticed that when they mix-down recording, they usually average vocal accompaniment to both track of the song, that is, acoustic waveform vocal songs in the two channels are the same or similar, so we can take two channels by subtraction to remove the stereo vocal songs.

In this way, we can see that there is a unavoidable drawback of this method, which is if the producer of this songs intentionally add some reverberation to both track to make the sound more magnetic, these kind of vocal sound cannot be removed using this method, which is why you can still hear some vocal sound after vocal elimination.

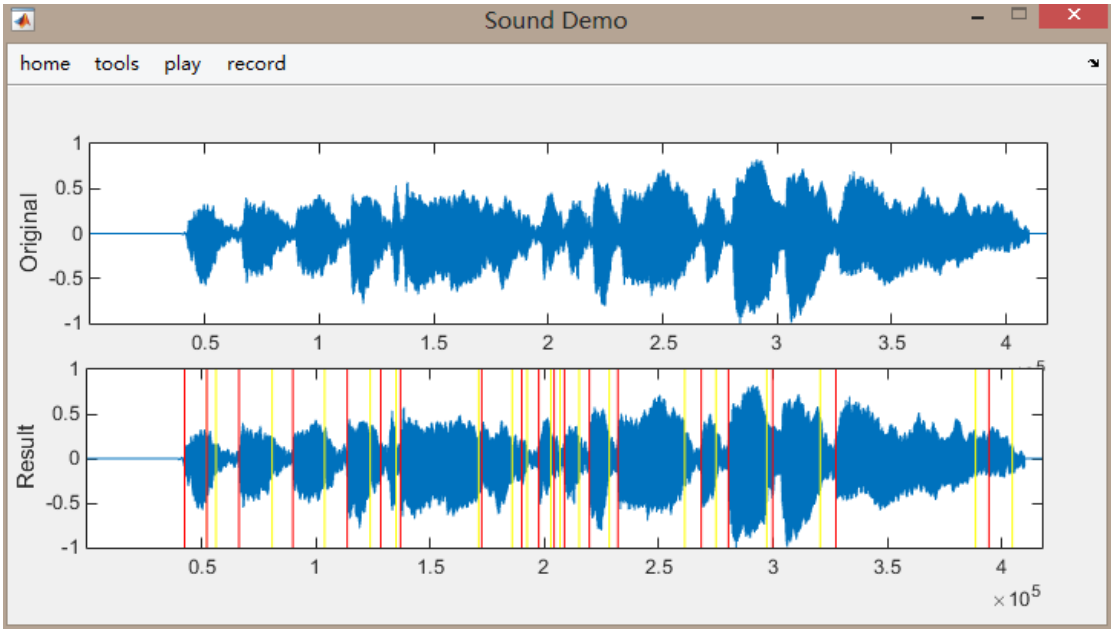
2.Pitch Adjusting. In this part we use an outside library called A Phase Vocoder by D.P.W.Ellis from Columbia University, we used some of their code and did some revision, we used a method called pvoc(x, r, n) which Time-scale a signal to r times faster with the input signal x using STFT(short-time Fourier transform). We first read the file using wavread() function to get the signal d and sample rate of the wave file, and then we use pvoc() method from the library to time-scale the music to r time faster and then we use the resample function from matlab to resample the music to original length.

We use the Equal Temperament to calculate the r and the scale of the resample function, in the Equal Temperament, each note is increasing by 2^(1/12) in frequency rate.
To have better result and less distortion from the pitch changing process; we confine the range of the pitch change within 4notes in total(major 3rd).

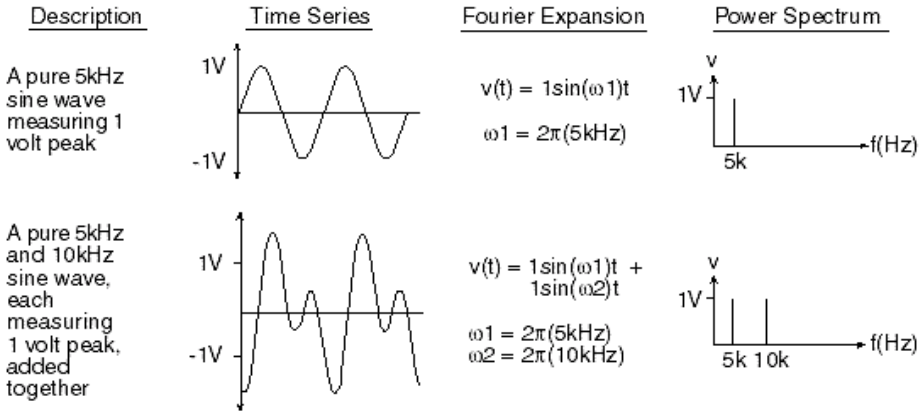
```
[d,sr] = wavread('star.wav');

one_key_down = 10595;
two_key_down = 11225;
three_key_down = 11892;
four_key_down = 12599;

one_key_up = 9439;
two_key_up = 8909;
three_key_up = 8409;
four_key_up = 7937;

p = four_key_up;
q = 10000;
quo = p/q;

e = pvoc(d,quo,1024);
f = resample(e,p,q);
sound(f,sr);
```

The second part of our project is called notes recognition. There are two main obstacles we are facing, the first is how to divide the music into single notes, the second is how to identify the single notes.

1. There are many different ways to do notes split work, such as neural network, entropy-based way and short-time energy recognition, etc. We decided to use short-time energy because it is the most classic and simplest way. When the short-time energy increases rapidly, we think that a new note starts. When the short-time energy cuts by half, we think that this note ends. As the picture below, the red line is the start of a note and the yellow line is the end of a note. We can see that this music can be separated correctly.



2. Once we have divided the music into single notes, we have to identify the every note; we have tried different methods to do this job, and also thought out a lot of heuristics to do a better job. First let's talk about Fourier transform briefly, this process is the foundation of the signal processing, in a word, The Fourier transform decomposes a function of time (a signal) into the frequencies that make it up, similarly to how a musical chord can be expressed as the amplitude (or loudness) of its constituent notes.



as you can see from the picture, the Fourier transform can transform the time series from the left to the right, and we can analyze the frequency in the left graph.

After we did the fft, we tried different ways to infer the fundamental frequency of the note

```
if(spectrum(i) > .1 & spectrum(i) > spectrum(i-1) & spectrum(i) > spectrum(i+1))
```

this is how we get the local maximum of the frequency

```
for(j = 2:numNote)
      if(mod(newFrequency,frequencies(numSegment,j)) < .03*newFrequency | ...
       frequencies(numSegment,j) - mod(newFrequency,frequencies(numSegment,j))  < .03*newFrequency)
```

this is how we figure out if the notes we got is in perfect eighth.

```
function [music,tempo] = noteDuration(head,tail,notes,fs)
    durs = zeros(1,length(head));
    for n = 1:length(durs)
        durs(n)=tail(n)-head(n);
    end
    dist=zeros(1,ceil(max(durs)/1000)*1000);
    for n = 1:length(durs)
        if durs(n) ~= 0
            dist(durs(n))=dist(durs(n))+1;
        end
    end
    N = 60;
    histdist=histc(durs,0:ceil(length(dist)/N):length(dist));

    [val,loc]=max(histdist);
    loc = (loc-.5)*length(dist)/N;
    quarter=avex2(dist(ceil(loc*7/8):min(length(dist),floor(loc*9/8))),ceil(loc*7/8));
    num8=round(2*durs/quarter);
    lengths = cell(length(durs),1);
    for n = 1:length(durs)
        switch(num8(n))
            case 1
                lengths(n) = cellstr('eighth note');
            case 2
                lengths(n) = cellstr('quarter note');
            case 3
                lengths(n) = cellstr('dotted quarter note');
            case 4
                lengths(n) = cellstr('half note');
            case 6
                lengths(n) = cellstr('dotted half note');
            case 8
                lengths(n) = cellstr('whole note');
            otherwise
                lengths(n) = cellstr('unknown length');
        end
    end
```

This is part of the code of how we calculate the duration of the note.

And we also change some part of the code to handle the difference between the vocal recognition and instrumental sound.

3. We used matlab guide to design UI. In our UI, we have inputdlg window, msgbox window, toolbar, axes, etc. We used handler to pass variables between different callback functions.