AMATH 582 Homework 2

Qiangqiang Guo

Due February 10, 2021

Abstract

In this assignment, we will analyze a portion of two of the greatest rock and roll songs. Specifically, we will reproduce the music score for the guitar in the "Sweet Child O' Mine" and the bass in the "Comfortably Numb". In addition, we will isolate the bass and the guitar solo in "Comfortably Numb".

1 Introduction and Overview

Sounds are everywhere in our daily life and songs are the art of sounds. We can enjoy a song in time domain, which is a sound combination of different instruments (and singers sometimes). However, a normal listener usually cannot distinguish different instruments accurately, not mention distinguishing the music scores for different instruments. Luckily, we can analyze the music from spectral domain which will make it much easier to distinguish different instruments as well as music scores. I believe this is one of the key functions included in many music edit software. In this assignment, we will analyze a portion of two of the greatest rock and roll songs by looking into the spectral domain. Specifically, we will reproduce the music score for the guitar in the "Sweet Child O' Mine" and the bass in the "Comfortably Numb". In addition, we will isolate the bass and the guitar solo in "Comfortably Numb".

In the following of this report, we will first introduce the theoretical background used in this assignment (Section 2). Then, we show how to design and implement the analysis algorithm (Section 3). We show the computation results in Section 4 and conclude this report in Section 5.

2 Theoretical Background

The given songs are in time domain. In order to distinguish different instruments and reproduce the music scores, we need to transform them to spectral domain. When it comes to the transformation between time and spectral domains, Fourier Transform is the first and natural choice. The Fourier transform and its inverse are defined as

$$F(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikt} f(t) dt \tag{1}$$

$$f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikt} F(k) dk \tag{2}$$

In MATLAB, we use the Fast Fourier transform (FFT) to perform the forward and backward Fourier transforms.

However, FFT will generate the spectral information for the whole data set but will not tell us when they occur. That is, we will lose all time domain information when we take FFT on the whole data set. For a song, different instruments and different notes are played at different times. Pure FFT can not find when a note is played. To remedy this problem, we can use Gabor transforms instead of FFT. The key idea of Gabor transforms is to transform local time-domain information to frequency domain and do such transforms around different time points. The Gabor transform is

$$\tilde{f}_g(t,k) = \int_{-\infty}^{\infty} f(\tau) \bar{g}(\tau - t) e^{-ik\tau} d\tau \tag{3}$$

where $g(\tau-t)$ is a new term introduced to the Fourier kernel e^{ikx}, aiming to localize both time and frequency. The bar denotes the complex conjugate of the function. t is the time point that we want to localize. There are many types of the function $g(\tau-t)$, e.g., Gaussian, Mexican Hat, and Step function. We will use Gaussian in this assignment.

There is a trade-off between the time and frequency information that we can extract using Gabor transform. When we apply filter around a specific time point, it eliminates some long waves, which means the frequency range we can extract decreases. The good thing is we have the time information in this way. The smaller the filter window is, the more precise time information we can get, but the more frequency information we will lose. We will explore this trade-off in Section 3.

In addition, in order to get a good clean music score, we will filter out overtones and other instrument frequencies in frequency domain. A Gaussian filter is used

$$\mathcal{F}(k) = e^{-d(k-k_0)^2} \tag{4}$$

where d is the bandwidth of the filter and k_0 is the center frequency.

3 Algorithm Development and Implementation

3.1 Read the audio data and setup

We first read in the audio data and calculate the time in seconds, which defines the time domain. The number of Fourier modes n is the size of the sampled audio data. Since FFT assumes 2π periodic signals, we should re-scale the wavenumbers by $\frac{2\pi}{L}$ where L is the time length calculated before. Due to the same reason (periodic assumptions of FFT), the first and last points should be the same, therefore, only the first n Fourier modes should be considered. In addition, we can shift the original wavenumbers (i.e., $k = (2\pi/L) * [0 : (n/2-1) - n/2 : -1]$) back to its mathematically correct positions.

3.2 Create the Gabor filter

We use a Gaussian filter as the Gabor kernel. We also set up the width of the Gaussian filter in this stage. To do so, we test different widths to find the best one with clear spectrogram features. In addition, the number of time discretization points are also specified.

3.3 Create the Gabor spectrogram and plot

We iterate over all time points that defined in the previous step. At each time point, we first apply the Gaussian filter defined in the previous step to the whole data, then take the FFT of the filtered data. Considering that the guitar in "Sweet Child O' Mine" and the bass in "Comfortably Numb" are pretty identifiable, the magnitudes of guitar and bass should dominate the frequency domains. Therefore, to filter out the overtones, we find the frequency with maximum magnitude and apply a Gaussian filter around this center frequency. Finally, we store the filtered frequency data to the spectrogram. After the iteration, we plot the spectrogram by pcolor. Based on the music score map, we can identify the music scores for the corresponding dominant instrument. This process will be conducted for both 'Sweet Child O' Mine" and "Comfortably Numb" so we can find the music scores for both the guitar and bass.

3.4 Isolate the bass in Comfortably Numb

This is done simultaneously with the previous step, where we filter out the overtones using Gaussian filter. Note that several tests are conducted to find the best width of the Gaussian filter (i.e., the width by which the Gaussian filter can eliminate overtones and other instrument frequencies as well as maintain the bass frequencies).

3.5 Extract the guitar solo in Comfortably Numb

The guitar is not dominant in Comfortably Numb. We will first filter out the bass and its overtones, then filter out the drum and its overtones. Finally, we put another filter around the rest center frequencies to make them clear.

4 Computation results

4.1 Sweet Child O' Mine

First, we show the parameter tuning results in Figure 1 and Figure 2. In 1, wt represent the filter width on time domain, which is the Gabor kernel. Note that we do not apply frequency domain filter in this figure. Based on the results, we select wt as 100 since the music scores are pretty clear and concentrated.

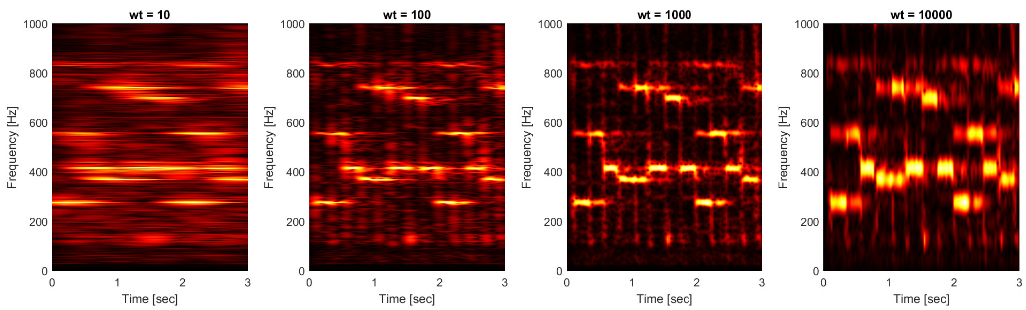

Figure 1: Parameter tuning: width of the filter in time domain

Figure 2 shows the parameter tuning for the width (noted as wq) of the filter in frequency domain, which is applied to the center frequency. Based on the results we select wq as 0.002, since the music scores are clear, concentrated and the details are reserved.

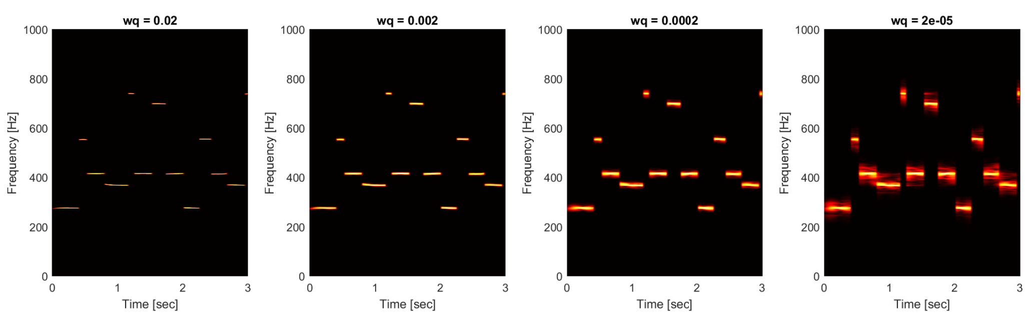

Figure 2: Parameter tuning: width of the filter in frequency domain

Using those parameters, we create the spectrogram of the whole data in Figure 3. Based on this figure and the map of music scores, we can extract the music scores for the guitar in Sweet Child O' Mine: middle C, high C, high A, middle F, high A, high, F, high A, middle C, high C, high G, high F, high A, ...

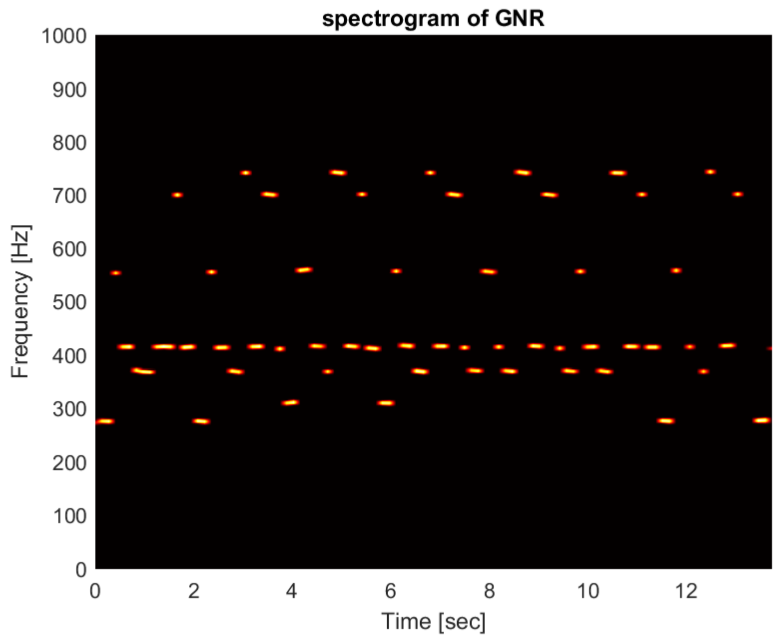

Figure 3: Spectrogram of the Sweet Child O' Mine

4.2 Comfortably Numb

Following the similar process as the previous section, we select the width for time domain (wt) as 100. Before filter in frequency domain, the spectrogram of the whole data is shown in Figure 4 (note that the whole data is clipped to 10 slices to prevent the MATLAB from running out of memory).

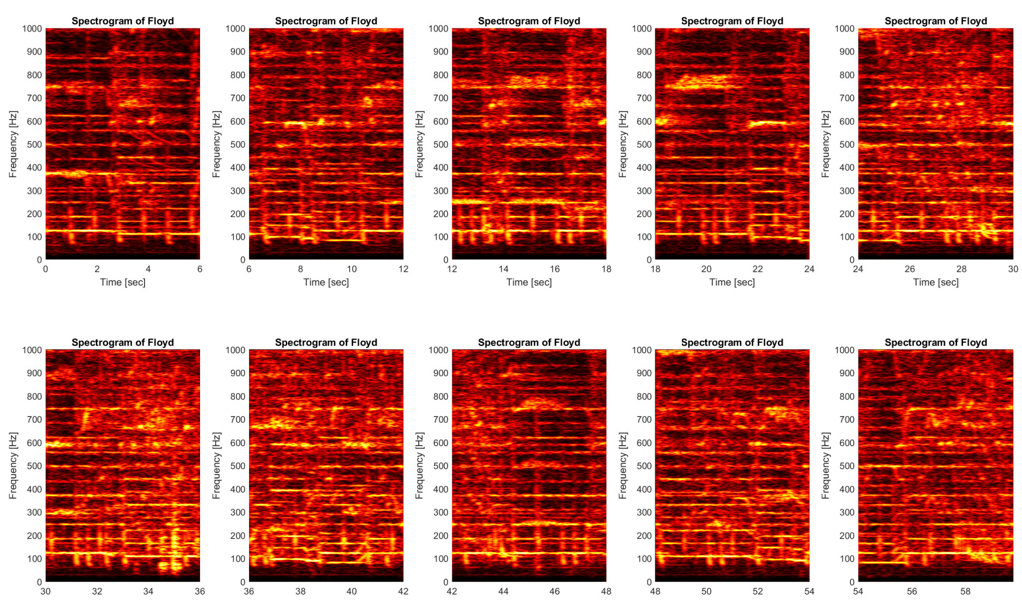

Figure 4: Spectrogram of Comfortably Numb before filter in frequency domain

It can be shown that there are lots of overtones. Considering that the base frequencies are always less than 150Hz, we find the maximum frequencies under 150Hz and filter around these center frequencies (using a Gaussian filter with 0.002 width) to filter out the overtones. The results are shown in Figure 5. The music scores can be extracted based on the music score map: low B, low A, ... Details are omitted here for brevity.

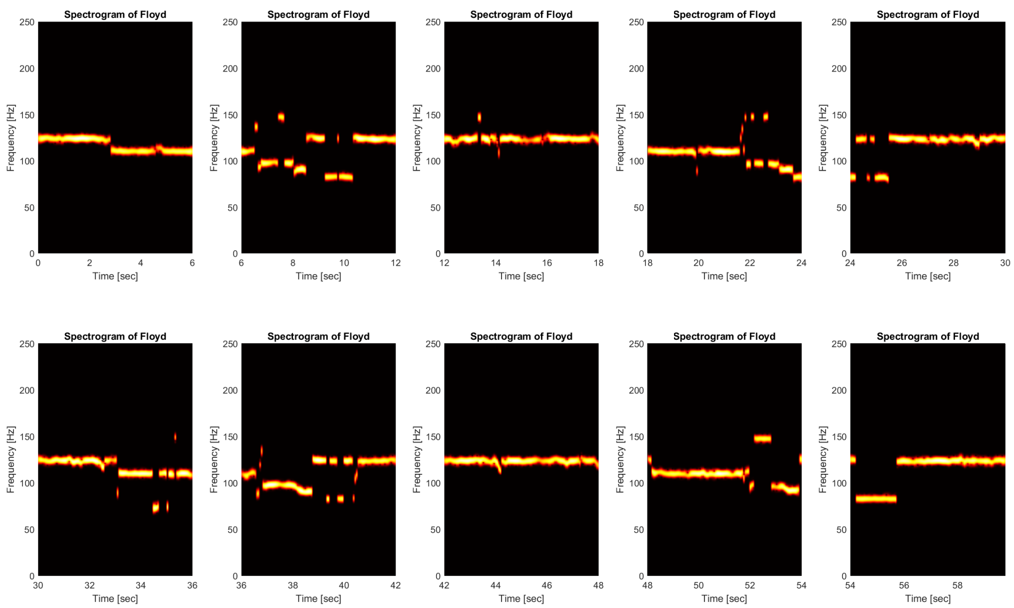

Figure 5: Spectrogram of Comfortably Numb after filter in frequency domain

4.3 Guitar solo in Comfortably Numb

Since bass dominates the Comfortably Numb, we need to filter out the bass in order to get the guitar solo. Considering there are lots of overtones (as shown in Figure 4), we use the base frequency shown in Figure 5 and create multiple filters at the frequencies which are multiple of the base frequency. For example, if the base frequency is 100Hz, we will create other 9 filters centered at $[200, 300, 400, 500, 600, 700, 800, 900, 1000]$ and add them up to make a new filter. This integrated filter can be used to filter out the overtones. Based on our observation, there are still low-frequency peaks after such processing. We think those might be the drum frequencies since the drum is more obvious than guitar when we listen to the music. A quick search shows that the drum frequency is usually lower than 250Hz. Therefore, we apply the same filter process to filter out base (under 250Hz) drum frequencies and the related overtones. In addition, we put another filter around the center frequencies to make the rest center frequencies clear. After these steps, the spectrogram is shown in Figure 6. It is shown that there are lots of noises, but the music scores are relatively clear. The details of mapping Figure 6 to music scores are omitted here for brevity.

5 Summary and Conclusions

In this assignment, we designed algorithms to explore the two of the greatest rock and roll songs from frequency domain. We used Fourier transform to transform the time domain data to frequency domain. In order to reserve time information, e.g., what the specific guitar note is at a specific time point, we used Gabor transform to analyze the data. In addition, to eliminate the overtones, we detected the center frequencies in spectral domain and filtered out all related overtones. For Sweet Child O' Mine, we clearly identified the guitar scores. For Comfortably Numb, we successfully identified the bass scores and have tried to isolate the

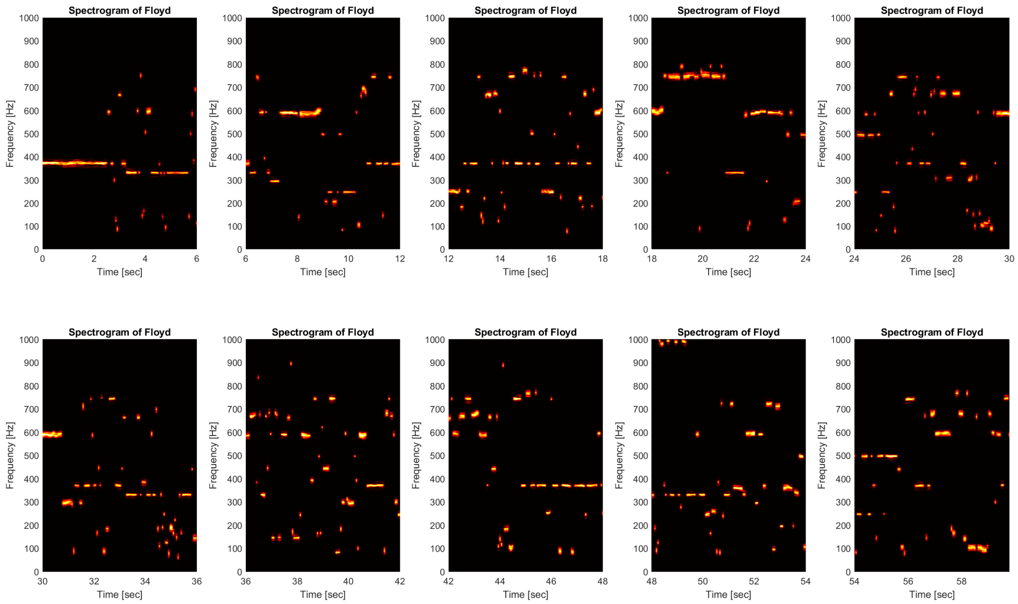

Figure 6: Spectrogram of Comfortably Numb: find the guitar

guitar solo. During this assignment, we have learned the power of Fourier transform and Gabor transform in time-frequency analysis.

Appendix A GitHub repository

`https://github.com/Guoqq17/UW-AMATH-582`

Appendix B Key MATLAB Functions

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `[y,Fs] = audioread(filename,samples)` reads the selected range of audio samples in the file, where samples is a vector of the form `[start,finish]`.
- `Y = fft(X)` computes the discrete Fourier transform (DFT) of `X` using a fast Fourier transform (FFT) algorithm.
- `[row,col] = ind2sub(sz,ind)` returns the arrays row and col containing the equivalent row and column subscripts corresponding to the linear indices ind for a matrix of size `sz`. Here `sz` is a vector with two elements, where `sz(1)` specifies the number of rows and `sz(2)` specifies the number of columns.

Appendix C MATLAB Code

```
%% understand the data
figure(1)
[y, Fs] = audioread('Floyd.m4a');
tr_gnr = length(y)/Fs; % record time in seconds
```

```
plot((1:length(y))/Fs,y);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Sweet Child O Mine');
p8 = audioplayer(y,Fs); playblocking(p8);

%% GNR
[y, Fs] = audioread('GNR.m4a', [1, 3*Fs]);      % read first 3 seconds
tr_gnr = length(y)/Fs;                 % record time in seconds
L = tr_gnr;                            % time domin
n = length(y);                         % Fourier modes
t1 = linspace(0, L, n + 1);
t = t1(1:n);
k = (2*pi/L)*[0:n/2-1, -n/2:-1];
ks = fftshift(k);

% create Gabor filter
width_t_all = [10, 100, 1000, 10000];         % width of the filter
width_q_all = [0.02, 0.002, 0.0002, 0.00002];           % width of the filter in frequency
    domain
num_gabor = 100;                               % number of the time points to take
t_gabor = linspace(0, t(end), num_gabor);   % discretize the time
s_gabor = zeros(length(t_gabor), n);         % matrix to store the Gabor transforms

% tune wt
figure(2)
for w_t = 1:1:length(width_t_all)
    for i=1:length(t_gabor)
        gabor = exp(-width_t_all(w_t)*(t - t_gabor(i)).^2);
        gyt = fft(gabor.*y.');
        gyts = abs(fftshift(gyt));
        s_gabor(i,:) = gyts;
    end

    % plot the spectrogram
    subplot(1, length(width_t_all), w_t)
    pcolor(t_gabor, ks/(2*pi), log(s_gabor.' + 1)), shading interp
    colormap('hot'), xlabel('Time [sec]'), ylabel('Frequency [Hz]')
    axis([0, tr_gnr, 0, 1000])
    title(['wt = ', num2str(width_t_all(w_t))])
end

% tune wq
figure(3)
for w_q = 1:1:length(width_q_all)
    for i=1:length(t_gabor)
        gabor = exp(-100*(t - t_gabor(i)).^2);
        gyt = fft(gabor.*y.');
        gyts = abs(fftshift(gyt));
        [val, ind] = max(gyts(n/2:end));
        [a,b] = ind2sub(size(gyts),ind+n/2-1);
        s_filter = exp(-width_q_all(w_q) * ((ks - ks(b)).^2));
        gytf = fftshift(gyt).*s_filter;
        s_gabor(i,:) = abs(gytf);
    end

    % plot the spectrogram
    subplot(1, length(width_q_all), w_q)
    pcolor(t_gabor, ks/(2*pi), log(s_gabor.' + 1)), shading interp
    colormap('hot'), xlabel('Time [sec]'), ylabel('Frequency [Hz]')
    axis([0, tr_gnr, 0, 1000])
    title(['wq = ', num2str(width_q_all(w_q))])
end

% whole data
[y, Fs] = audioread('GNR.m4a');               % read whole audio data
tr_gnr = length(y)/Fs;                 % record time in seconds
L = tr_gnr;                            % time domin
n = length(y);                         % Fourier modes
```

```
t1 = linspace(0, L, n + 1);
t = t1(1:n);
k = (2*pi/L)*[0:n/2-1, -n/2:-1];
ks = fftshift(k);

% create Gabor filter
num_gabor = 100;                               % number of the time points to take
t_gabor = linspace(0, t(end), num_gabor);   % discretize the time
s_gabor = zeros(length(t_gabor), n);        % matrix to store the Gabor transforms
score = zeros(1, length(t_gabor));

for i=1:length(t_gabor)
    gabor = exp(-100*(t - t_gabor(i)).^2);
    gyt = fft(gabor.*y.');
    gyts = abs(fftshift(gyt));
    [val, ind] = max(gyts(n/2:end));
    [a,b] = ind2sub(size(gyts),ind+n/2-1);
    score(i) = ks(b);
    s_filter = exp(-0.002 * ((ks - ks(b)).^2));
    gytf = fftshift(gyt).*s_filter;
    s_gabor(i,:) = abs(gytf);
end

% plot the spectrogram
figure(4)
pcolor(t_gabor, ks/(2*pi), log(s_gabor.' + 1)), shading interp
colormap('hot'), xlabel('Time [sec]'), ylabel('Frequency [Hz]')
axis([0, tr_gnr, 0, 1000])
title('spectrogram of GNR')

%% Floyd
clear; clc;
close all
[y, Fs] = audioread('Floyd.m4a');
[y, Fs] = audioread('Floyd.m4a', [1, 10*Fs]);      % read first 3 seconds
tr_floyd = length(y)/Fs;               % record time in seconds
L = tr_floyd;                          % time domin
n = length(y);                       % Fourier modes
t1 = linspace(0, L, n + 1);
t = t1(1:n);
k = (2*pi/L)*[0:n/2-1, -n/2:-1];
ks = fftshift(k);

% create Gabor filter
width_t_all = [10, 100, 1000, 10000];       % width of the filter
width_q_all = [0.02, 0.002, 0.0002, 0.00002];           % width of the filter in frequency
    domain
num_gabor = 100;                               % number of the time points to take
t_gabor = linspace(0, t(end), num_gabor);   % discretize the time
s_gabor = zeros(length(t_gabor), n);        % matrix to store the Gabor transforms

%% tune wt
figure(4)
for w_t = 1:1:length(width_t_all)
    for i=1:length(t_gabor)
        gabor = exp(-width_t_all(w_t)*(t - t_gabor(i)).^2);
        gyt = fft(gabor.*y.');
        gyts = abs(fftshift(gyt));
        s_gabor(i,:) = gyts;
    end

    % plot the spectrogram
    subplot(1, length(width_t_all), w_t)
    pcolor(t_gabor, ks/(2*pi), log(s_gabor.' + 1)), shading interp
    colormap('hot'), xlabel('Time [sec]'), ylabel('Frequency [Hz]')
    axis([0, tr_floyd, 0, 1000])
    title(['wt = ', num2str(width_t_all(w_t))])
end
```

```
%% all data before wq
[y_all, Fs] = audioread('Floyd.m4a');
step = 6;                                % time step to cut the whole audio to slices
num_gabor = 100;                         % number of the time points to take at each slice

figure(5)
for s_i=1:length(y_all)/(Fs*step)+1
    if s_i*step*Fs < length(y_all)
        [y, Fs] = audioread('Floyd.m4a', [(s_i-1)*step*Fs+1, s_i*step*Fs]);
    else
        [y, Fs] = audioread('Floyd.m4a', [(s_i-1)*step*Fs+1, length(y_all)-1]);
    end

    tr_floyd = length(y)/Fs;                     % record time in seconds
    L = tr_floyd;                                % time domin
    n = length(y);                             % Fourier modes
    t1 = linspace(0, L, n + 1);
    t = t1(1:n);
    k = (2*pi/L)*[0:n/2-1, -n/2:-1];
    ks = fftshift(k);

    % create Gabor filter
    width = 100;                                  % width of the filter
    t_gabor = linspace(0, t(end), num_gabor);    % discretize the time
    s_gabor = zeros(length(t_gabor), n);

    % create the spectrogram
    for i=1:length(t_gabor)
        gabor = exp(-width*(t - t_gabor(i)).^2);
        gyt = fft(gabor.*y.');
        gyts = abs(fftshift(gyt));
        s_gabor(i,:) = gyts;
    end

    subplot(2,5,s_i)
    pcolor(t_gabor + step*(s_i-1), ks/(2*pi), log(s_gabor.' + 1)), shading interp
    colormap('hot'), xlabel('Time [sec]'), ylabel('Frequency [Hz]')
    axis([step*(s_i-1), step*(s_i-1) + tr_floyd, 0, 1000])
    title('Spectrogram of Floyd')
    drawnow
end

%% all data after aplly frequency domain filter (under 150Hz)
[y_all, Fs] = audioread('Floyd.m4a');
step = 6;                                % time step to cut the whole audio to slices
num_gabor = 100;                         % number of the time points to take at each slice

figure(5)
for s_i=1:length(y_all)/(Fs*step)+1
    if s_i*step*Fs < length(y_all)
        [y, Fs] = audioread('Floyd.m4a', [(s_i-1)*step*Fs+1, s_i*step*Fs]);
    else
        [y, Fs] = audioread('Floyd.m4a', [(s_i-1)*step*Fs+1, length(y_all)-1]);
    end

    tr_floyd = length(y)/Fs;                      % record time in seconds
    L = tr_floyd;                                 % time domin
    n = length(y);                                % Fourier modes
    t1 = linspace(0, L, n + 1);
    t = t1(1:n);
    k = (2*pi/L)*[0:n/2-1, -n/2:-1];
    ks = fftshift(k);

    % create Gabor filter
    width = 100;                                  % width of the filter
    t_gabor = linspace(0, t(end), num_gabor);    % discretize the time
    s_gabor = zeros(length(t_gabor), n);
```

```

    % create the spectrogram
    for i=1:length(t_gabor)
        gabor = exp(-width*(t - t_gabor(i)).^2);
        gyt = fft(gabor.*y.');
        gyts = abs(fftshift(gyt));
        [val, ind] = max(gyts(n/2:n/2 + 150 * 2*pi));
        [a,b] = ind2sub(size(gyts),ind+n/2-1);
        s_filter = exp(-0.002 * ((ks - ks(b)).^2));
        gytf = fftshift(gyt).*s_filter;
        s_gabor(i,:) = abs(gytf);
    end

    subplot(2,5,s_i)
    pcolor(t_gabor + step*(s_i-1), ks/(2*pi), log(s_gabor.' + 1)), shading interp
    colormap('hot'), xlabel('Time [sec]'), ylabel('Frequency [Hz]')
    axis([step*(s_i-1), step*(s_i-1) + tr_floyd, 0, 250])
    title('Spectrogram of Floyd')
    drawnow
end
%% all data to find guitar
[y_all, Fs] = audioread('Floyd.m4a');
step = 6;                               % time step to cut the whole audio to slices
num_gabor = 100;                        % number of the time points to take at each slice

figure(5)
for s_i=1:length(y_all)/(Fs*step)+1
    if s_i*step*Fs < length(y_all)
        [y, Fs] = audioread('Floyd.m4a', [(s_i-1)*step*Fs+1, s_i*step*Fs]);
    else
        [y, Fs] = audioread('Floyd.m4a', [(s_i-1)*step*Fs+1, length(y_all)-1]);
    end

    tr_floyd = length(y)/Fs;                    % record time in seconds
    L = tr_floyd;                               % time domin
    n = length(y);                              % Fourier modes
    t1 = linspace(0, L, n + 1);
    t = t1(1:n);
    k = (2*pi/L)*[0:n/2-1, -n/2:-1];
    ks = fftshift(k);

    % create Gabor filter
    width = 100;                                % width of the filter
    t_gabor = linspace(0, t(end), num_gabor);   % discretize the time
    s_gabor = zeros(length(t_gabor), n);

    % create the spectrogram
    for i=1:length(t_gabor)
        gabor = exp(-width*(t - t_gabor(i)).^2);
        gyt = fft(gabor.*y.');

        % filter out the bass and overtones
        gyts = abs(fftshift(gyt));
        [val, ind] = max(gyts(n/2:n/2 + 150 * 2*pi));
        [a,b] = ind2sub(size(gyts),ind+n/2-1);
        s_filter = zeros(1, length(ks));
        j = 1;
        c_q = ks(b) * j;
        while c_q <= 1000*2*pi
            s_filter = s_filter + exp(-0.0002 * ((ks - c_q).^2));
            j = j + 1;
            c_q = ks(b) * j;
        end

        % filter out the drum and overtones
        gyts = abs(fftshift(gyt).*(1-s_filter));
        [val, ind] = max(gyts(n/2:n/2 + 250 * 2*pi));
        [a,b] = ind2sub(size(gyts),ind+n/2-1);
```

```
        j = 1;
        c_q = ks(b) * j;
        while c_q <= 1000*2*pi
            s_filter = s_filter + exp(-0.0002 * ((ks - c_q).^2));
            j = j + 1;
            c_q = ks(b) * j;
        end
        s_filter = 1-s_filter;
        gyts = fftshift(gyt).*s_filter;

        % identify the guitar
        [val, ind] = max(gyts(n/2:n/2 + 1000 * 2*pi));
        [a,b] = ind2sub(size(gyts),ind+n/2-1);
        s_filter = exp(-0.0002 * ((ks - ks(b)).^2));
        gytf = gyts.*s_filter;

        s_gabor(i,:) = abs(gytf);
    end

    subplot(2,5,s_i)
    pcolor(t_gabor + step*(s_i-1), ks/(2*pi), log(s_gabor.' + 1)), shading interp
    colormap('hot'), xlabel('Time [sec]'), ylabel('Frequency [Hz]')
    axis([step*(s_i-1), step*(s_i-1) + tr_floyd, 0, 1000])
    title('Spectrogram of Floyd')
    drawnow
end
```