Gábor transforms

Jonathan Zhao

Feb 3rd, 2020

Abstract

This report discusses a method, Gábor transforms, to conduct time-frequency analysis. We have three portions of music and our goal is to extract both frequency and time information at the same time by using Gábor transforms. To explore this method, we will produce different spectrograms of the music through adjusting the window widths, using different time translations, and applying different Gábor windows. And then we will analyze a simple piece of music to reproduce the music score.

Sec I: Introduction and Overview

The Fourier transform, as we discussed in the previous report, is one of the most powerful methods to solve problems related to time-frequency analysis and is widely used in signal processing and analyzing. However, it is also acknowledged that the transform translates the signal in time domain to frequency domain and so that fails to extract information in time domain. That is, we miss the information of the changes of the signal as time goes by. Thus, we will introduce an approach as a modification of the Fourier transform, Gábor transforms, to solve problems related with both time and frequency analysis.

In this report, we will first introduce some theoretical backgrounds in the next section, and then make analysis on three pieces of music. For part one, we will analyze a portion of Handel's Messiah by producing spectrograms of the piece of work, and explore how the factors in the Gábor transform, such as the window width and time translations, impacts on the the spectrograms. For part two, we will process two pieces of the song *Mary had a little lamb* on both the recorder and piano, reproducing the music score and comparing the differences between the two.

Sec II: Theoretical Background

2.1 Drawback of Fourier Transform

For $x \in [-\infty, \infty]$, the Fourier transform (in one dimension) is defined as:

$$F(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x) dx \tag{2.1.1}$$

The Fourier transform is used to decompose a time defined function into a frequency-amplitude image. And we will use the fast Fourier transform (FFT) to perform the Fourier transform. Since the Fourier transform converts the time domain to frequency domain and depicts stationary signals, it eliminates all time-domain information about variance of frequency. Say that there exists a piece of music and we apply FFT on it. What we get is an image containing information about frequency domain but nothing about time domain. We can not check whether it is producing high notes, middle notes, or low notes at a specific moment. Thus, we need a new method to analyze both time and frequency domains.

2.2 Gábor Transforms

The Gábor transform, also known as the short-time Fourier transform, was first proposed by the Hungarian physicist/mathematician/electrical engineer Gábor Dénes. To make time-frequency analysis with various signals changing as the time goes by, the Gábor transform decomposes the time domain into separate time frames. And for each time frame, or window, the Fourier transform is applied to depict the frequencies during that time frame, with the remaining time frames zeroed out.

The Gábor transform, or short-time Fourier transform, is defined as the following:

$$\mathcal{G}[f](t, \omega) = \tilde{f}_g(t, \omega) = \int_{-\infty}^{\infty} f(\tau) \bar{g}(\tau - t) e^{-i\omega\tau} d\tau = (f, \bar{g}_{t,\omega}) \tag{2.2.1}$$

We can see that the integral kernel $exp(ikx)$ for the Fourier transform is replaced with the kernel $e^{i\omega\tau} g(\tau - t)$ for Gábor transform. The function $g(\tau - t)$ localizes a piece of signal over a specific window of time by filtering out the remaining windows of time. The window is centered at τ with a width a. As τ increases, the Gábor filter slides down the entire signal and extracts frequency information for each time frame.

In practice, the Gábor transform is realized by discretizing the time and frequency domains. Thus we need to consider the discrete version with sample points. The Gábor Transform becomes:

$$\tilde{f}(m,n) = \int_{-\infty}^{\infty} f(t) e^{i2\pi m\omega_0 t} g\left(t - nt_0\right) dt \tag{2.2.2}$$

with $\nu = m\omega_0$ and $\tau = nt_0$, where m and n are integers and ω_0, $t_0 > 0$ are constants.

With a finite number of window of time, the Gábor transform trades away some measure of accuracy in both the time and frequency domains. We will discuss this property in the following.

2.3 Gábor Windows / Filters

The Gaussian filter, as one of the most common used time filters, is defined as:

$$\mathcal{F}(k) = \exp\left(-\tau\left(k - k_0\right)^2\right) \tag{2.3.1}$$

It is used to extract the signal pattern within a window in time domain, and so that we can analyze the frequency content at this specific moment. We will also apply other Gábor windows such as Mexican hat wavelet and a step-function (Shannon).

The Mexican hat wavelet is defined as:

$$\psi(t) = \frac{2}{\sqrt{3\sigma}\pi^{1/4}}\left(1 - \left(\frac{t}{\sigma}\right)^2\right) e^{-\frac{t^2}{2\sigma^2}} \tag{2.3.2}$$

And the step-function (Shannon) is defined as:

$$f(t) = \begin{cases} 1 \text{ when } \tau - width/2 < t < \tau + width/2 \\ 0, \text{ elsewhere} \end{cases} \tag{2.3.3}$$

Sec III: Algorithm Implementation & Development

Part 1: Exploration of time-frequency properties

First we load a portion of Handel's Messiah in matlab by using *load handel* in MATLAB, which gives us the vector v representing the music and the sampling frequency Fs. Dividing the length of the vector v by the sampling frequency, we gain the length of the music L = 8.925 seconds. During the process defining the time domain and frequency domain, we can find that the length of signal vector is an odd number, and so that we need to create discrete modes on $[0 : (n-1)/2 \ -(n-1)/2 :- 1]$, as different from the discrete modes on $[0 : n/2 - 1 \ -n/2 :- 1]$ for a signal vector with an even number of length. We then create the *tslide* to record the time frame to be processed, and three spectrograms for saving the results with normal window, narrower window, and wider window filtering. In a for-loop to slide the center down to the entire signal, we multiply the Gaussian filters with the signal and apply the FFT on it. After applying

fftshift on the absolute values of the results, we save them into the spectrograms at the relevant time frame. We have to rescale the frequency domain by dividing $(2 * pi)$ before plotting. By using *pcolor*, we get to see three spectrograms with different window widths.

And then we explore the idea of oversampling and undersampling, respectively caused by using very small and very large translations of the Gábor Windows. To obtain a result with oversampling, we make the interval between two adjacent centers of window of time 0.01, and so that we have 901 time translations. Then we set the interval to 1 and get 10 time translations. We create two for-loops and respectively do the same process of Gábor transform with the adjusted *tslide* values.

We have introduced three Gábor windows in the background sections but have only used one of them. At this time, we will use different Gábor windows and examine how the results are affected with the Mexican hat wavelet and the step-function filter. In a for-loop, we create the functions of the two windows and apply each of them on the vector v.

Part 2: Reproduce music notes & timbre examination

At this part we load two audio files to get the vectors representing a piece of music (*Mary had a little lamb*) and sample frequencies for the song played on both the recorder and piano. Dividing the length of the vector v by the sampling frequency, we gain the length of the music played on the piano and that played on the recorder. The two lengths are relatively 15.906s and 14.234s, and we set the interval of *tslide* to 0.2. Then we apply the Gábor transform on each of the signals as before. In order to filter out overtones and get a clean score, we look for the maximum value of the transformed data at each iteration of time translations, and save the indices of them to an array called *notes_piano* or *notes_rec*. Finally, we can produce the spectrograms of the results with the rescaled frequency domain, and the images of music notes with the arrays saving the strongest signals.

Sec IV: Computational Results & Conclusions

Part 1

By assigning the **parameter of the Gaussian filter** a with 1, 100, and 1000, respectively, we change the widths of the window and get three different spectrograms in **Figure1**. We can see that as we use the narrow window, good time resolution is gained but some frequency content is lost; and as we use the wide window, much more various frequency information is presented but the time resolution is blurred.

Figure 2 shows the resulting spectrograms of applying different Gábor windows. Their performances are almost the same, while the Shannon window filter shows a bit better time resolution at an expense of a little lost in time accuracy.

By changing the sampling numbers of time frame to 901 and 10, we explore the idea of oversampling and undersampling in **Figure 3**. We can see that the performance of the Gábor transform with very small translations (oversampling) is similar to that with normal translations.

However, oversampling makes the runtime pretty longer than the normal conditions. For undersampling, there is little accuracy that can be spoken from the spectrogram.

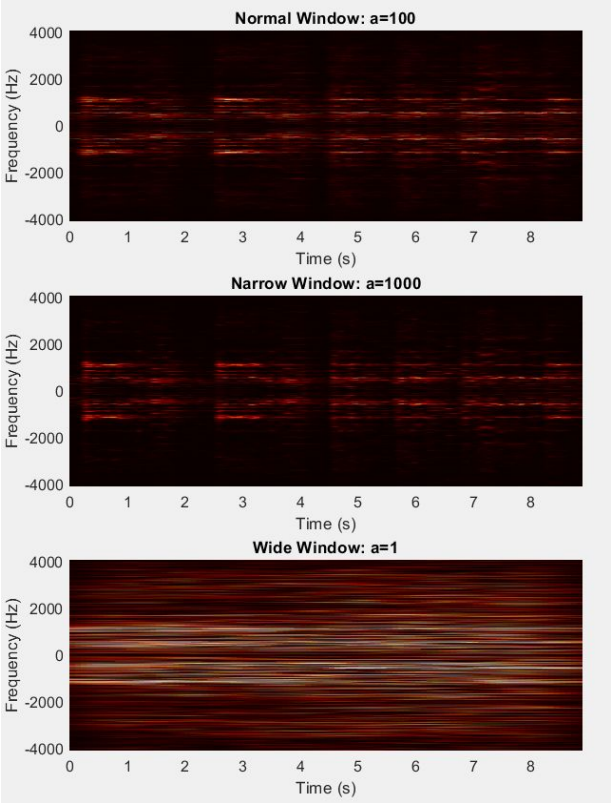

Figure 1: Using different window widths

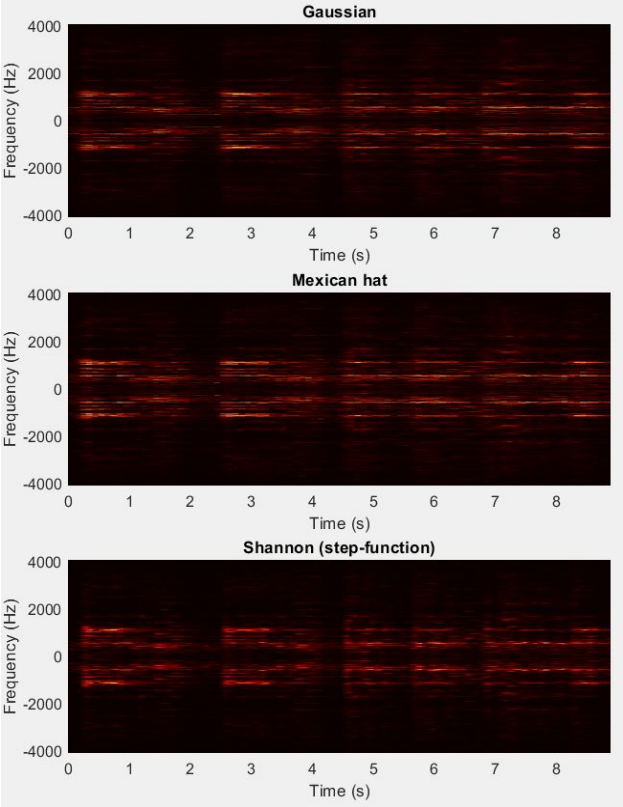

Figure 2: Using different Gábor windows

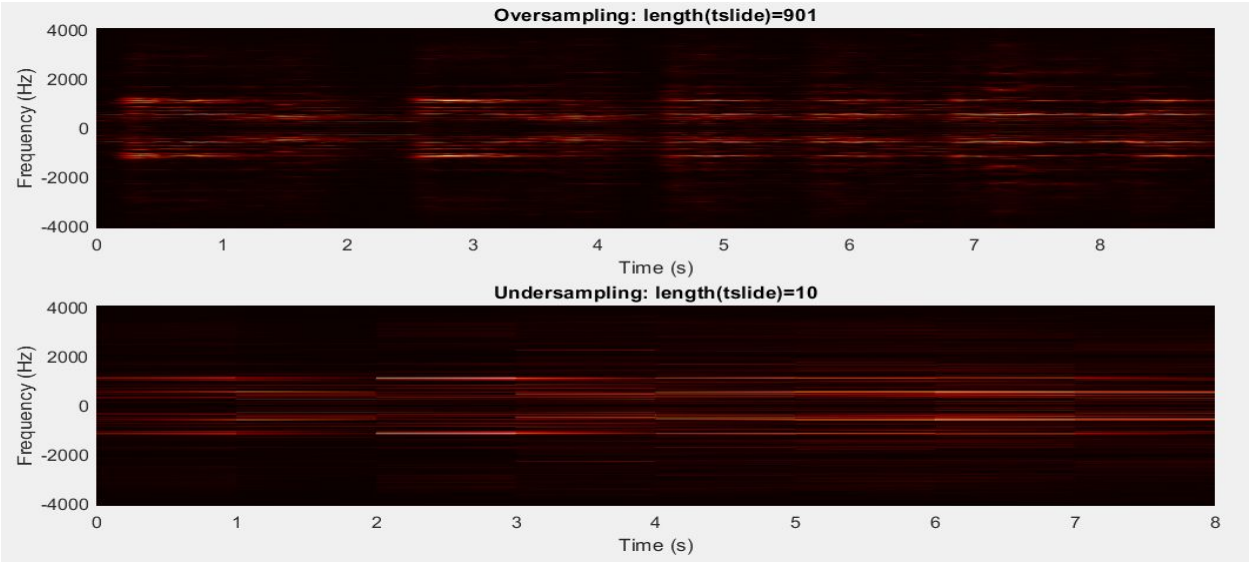

Figure 3: Oversampling & Undersampling

Figure 4: Spectrograms for the music played on the piano and recorder

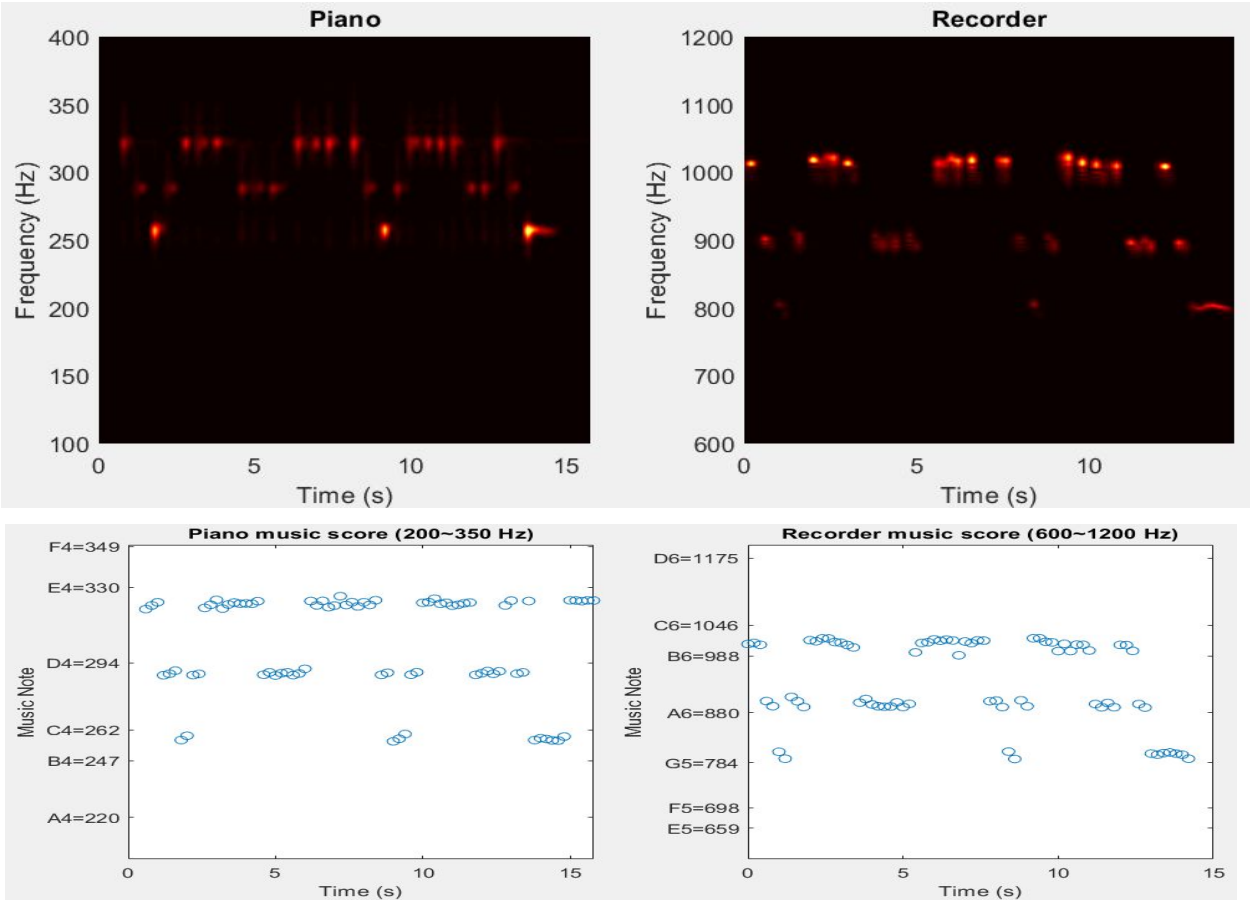

Figure 4: Music notes corresponding to the frequency on the piano and recorder

Part 2

After examining the spectrograms for the music played on the piano and recorder, we notice that the frequency range of the music played on the piano is from 200 to 350 Hz, and the frequency range of the other is from 600 to 1200 Hz. We filtered out the overtones by finding the maximum values of the signal at each time frame. **Figure 5** shows the music notes.

Comparing the two plots in **Figure 4 & 5**, we can see that the the general pattern of the two are the same, but with different frequency ranges, and there seems to be more overtones and undertones on the piano than there are in a recorder, which causes the differences in the timbre.

Appendix A: MATLAB functions used

1. fft: apply the FFT in 2-D
2. fftshift: Rearranges X by shifting the zero-frequency component to the center.
3. pcolor: creates a pseudocolor plot using the values in the object
4. colormap: set the colormap for the current figure to the color specified.
5. yticks: sets the y-axis tick values

Appendix B: MATLAB codes

```
%% Part I - Starter Code
clear; close all; clc
load handel  % Fs = the sampling frequency in Hz
             % y = the audio signal amplitude as a single column vector
v = y';
plot((1:length(v))/Fs, v);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Signal of Interest, v(n)');

% p8 = audioplayer(v,Fs);
% playblocking(p8);

%% Part I - Exploring different window widths
L = length(v)/Fs;  % 8.925 seconds
n = length(v);
t2=linspace(0,L,n+1);
t=t2(1:n);
k=((2*pi)/L)*[0:(n-1)/2 -(n-1)/2:-1];  % length of n is odd
ks=fftshift(k);

a = [1 100 1000];  % parameters for Gaussian Filter
tslide = 0:0.1:L;
vgt_spec = zeros(length(tslide), n);
vgst_spec = zeros(length(tslide), n);  % spectrogram w/ small window width
vglt_spec = zeros(length(tslide), n);  % spectrogram w/ large window width
for j = 1:length(tslide)
    g = exp(-a(2)*(t-tslide(j)).^2);
    g_narroww = exp(-a(3)*(t-tslide(j)).^2);  % narrower window
    g_widew = exp(-a(1)*(t-tslide(j)).^2);  % wider window

    vg = g.* v;
    vgt = fft(vg);
    vgs = g_narroww .* v;
    vgst = fft(vgs);
    vgl = g_widew .* v;
    vglt = fft(vgl);

    vgt_spec(j,:) = fftshift(abs(vgt));
    vgst_spec(j,:) = fftshift(abs(vgst));
    vglt_spec(j,:) = fftshift(abs(vglt));
end
```

```
43
44      % Spectrograms with different window widths
45 -    figure(1)
46 -    subplot(3,1,1)
47 -    pcolor(tslide,ks/(2*pi),vgt_spec.'),
48 -    shading interp
49 -    title('Normal Window: a=100')  % The 'a' here is not the width, but the parameter in
50                    % the Gaussian filter to change the width.
51 -    xlabel('Time (s)'), ylabel('Frequency (Hz)')
52 -    colormap(hot)
53
54 -    subplot(3,1,2)
55 -    pcolor(tslide,ks/(2*pi),vgst_spec.'),
56 -    shading interp
57 -    title('Narrow Window: a=1000')
58 -    xlabel('Time (s)'), ylabel('Frequency (Hz)')
59 -    colormap(hot)
60
61 -    subplot(3,1,3)
62 -    pcolor(tslide,ks/(2*pi),vglt_spec.'),
63 -    shading interp
64 -    title('Wide Window: a=1')
65 -    xlabel('Time (s)'), ylabel('Frequency (Hz)')
66 -    colormap(hot)
67
68      %% Part I - Oversampling & Undersampling
69 -    tslide_over = 0:0.01:L;  % using very small translations of the Gabor window
70 -    vgtspec_over = zeros(length(tslide_over), n);
71
72 -    for j = 1:length(tslide_over)
73 -        g = exp(-100*(t-tslide_over(j)).^2);
74 -        vg = g.* v;
75 -        vgt = fft(vg);
76 -        vgtspec_over(j,:) = fftshift(abs(vgt));
77 -    end
78
79 -    tslide_under = 0:1:L;  %  using very course/large translations
80 -    vgtspec_under = zeros(length(tslide_under), n);
81
82 -    for j = 1:length(tslide_under)
83 -        g = exp(-100*(t-tslide_under(j)).^2);
84 -        vg = g.* v;
```

```
 85 -        vgt = fft(vg);
 86 -        vgtspec_under(j,:) = fftshift(abs(vgt));
 87 -    end
 88
 89      % Spectrograms with different time translations
 90 -    figure(2)
 91 -    subplot(3,1,1)
 92 -    pcolor(tslide,ks/(2*pi),vgt_spec.'),
 93 -    shading interp
 94 -    title('length(tslide)=91')
 95 -    xlabel('Time (s)'), ylabel('Frequency (Hz)')
 96 -    colormap(hot)
 97
 98 -    subplot(3,1,2)
 99 -    pcolor(tslide_over,ks/(2*pi),vgtspec_over.'),
100 -    shading interp
101 -    title('Oversampling: length(tslide)=901')
102 -    xlabel('Time (s)'), ylabel('Frequency (Hz)')
103 -    colormap(hot)
104
105 -    subplot(3,1,3)
106 -    pcolor(tslide_under,ks/(2*pi),vgtspec_under.'),
107 -    shading interp
108 -    title('Undersampling: length(tslide)=10')
109 -    xlabel('Time (s)'), ylabel('Frequency (Hz)')
110 -    colormap(hot)
111
112
113     %% Part I - Using Different Gabor windows
114
115 -    vmht_spec = zeros(length(tslide), n);
116 -    vsht_spec = zeros(length(tslide), n);
117 -    for j = 1:length(tslide)
118         %  Mexican hat wavelet
119 -       sigma = 0.05;
120 -       mex_hat = (2/(sqrt(3*sigma)*(pi^0.25))).*(1-((t-tslide(j))/sigma).^2)...
121             .* exp(-((t-tslide(j)).^2)/(2*sigma^2));
122         % mex_hat = (1-((t-tslide(j)).^2)) .* exp(-((t-tslide(j)).^2)/2);
123
124         % Step-function (Shannon) window
125 -       width = 0.05;
126 -       shn = abs(t - tslide(j)) <= width /2;
```

```
126 -       shn = abs(t - tslide(j)) <= width /2;
127
128 -       vmh = mex_hat .* v;
129 -       vmht = fft(vmh);
130
131 -       vsh = shn .* v;
132 -       vsht = fft(vsh);
133
134 -       vmht_spec(j,:) = fftshift(abs(vmht));
135 -       vsht_spec(j,:) = fftshift(abs(vsht));
136 -   end
137
138 -   figure(3)
139 -   subplot(3,1,1)
140 -   pcolor(tslide,ks/(2*pi),vgt_spec.'),
141 -   shading interp
142 -   title('Gaussian')
143 -   xlabel('Time (s)'), ylabel('Frequency (Hz)')
144 -   colormap(hot)
145
146 -   subplot(3,1,2)
147 -   pcolor(tslide,ks/(2*pi),vmht_spec.'),
148 -   shading interp
149 -   title('Mexican hat')
150 -   xlabel('Time (s)'), ylabel('Frequency (Hz)')
151 -   colormap(hot)
152
153 -   subplot(3,1,3)
154 -   pcolor(tslide,ks/(2*pi),vsht_spec.'),
155 -   shading interp
156 -   title('Shannon (step-function)')
157 -   xlabel('Time (s)'), ylabel('Frequency (Hz)')
158 -   colormap(hot)
159
160
161     %% Part II - Starter Code
162 -   clear; close all; clc
163
164 -   [y,Fs] = audioread('music1.wav');
165 -   tr_piano=length(y)/Fs; % record time in seconds
166     % plot((1:length(y))/Fs,y);
167     % xlabel('Time [sec]'); ylabel('Amplitude');
```

```
% title('Mary had a little lamb (piano)');
% p8 = audioplayer(y,Fs); playblocking(p8);


%% Part II - 1

% We first focus on the spectrogram of the song on the piano
% to reproduce the music notes.
v = y';
n = length(v);
t2 = linspace(0,tr_piano,n+1);
t_p = t2(1:n);
k_p = (2*pi/tr_piano)*[0:n/2-1 -n/2:-1];
ks_p = fftshift(k_p);

tslide_p = 0:0.2:tr_piano;
spec_p = zeros(length(tslide_p), n);
notes_piano = zeros(1, length(tslide_p));
for j = 1:length(tslide_p)
    g = exp(-100*(t_p-tslide_p(j)).^2);  % Use Gaussian Filter

    vgp = g .* v;
    vgpt = fft(vgp);
    [M, I] = max(vgpt);  % Get the index with strongest frequency (music score)

    notes_piano(1,j) = abs(k_p(I))/(2*pi);
    spec_p(j,:) = fftshift(abs(vgpt));
end

% The length and the range of frequencies are different between the
% music score on the piano and recorder.
[y,Fs] = audioread('music2.wav');
tr_rec=length(y)/Fs; % record time in seconds
% plot((1:length(y))/Fs,y);
% xlabel('Time [sec]'); ylabel('Amplitude');
% title('Mary had a little lamb (recorder)');
% p8 = audioplayer(y,Fs); playblocking(p8);

v = y';
n = length(v);
t2 = linspace(0,tr_rec,n+1);
t_rec = t2(1:n);
```

```
210 -   k_rec = (2*pi/tr_rec)*[0:n/2-1 -n/2:-1];
211 -   ks_rec = fftshift(k_rec);
212
213 -   tslide_rec = 0:0.2:tr_rec;
214 -   spec_rec = zeros(length(tslide_rec), n);
215 -   notes_rec = zeros(1, length(tslide_rec));
216 -   for j = 1:length(tslide_rec)
217 -       g = exp(-100*(t_rec-tslide_rec(j)).^2);  % Use Gaussian Filter
218
219 -       vgr = g .* v;
220 -       vgrt = fft(vgr);
221 -       [M, I] = max(vgrt);  % Get the index with strongest signal (music score)
222
223 -       notes_rec(1,j) = abs(k_rec(I))/(2*pi);
224 -       spec_rec(j,:) = fftshift(abs(vgrt));
225 -   end
226
227
228 -   figure(4)
229 -   subplot(1,2,1)
230 -   pcolor(tslide_p,(ks_p/(2*pi)), spec_p .'),
231 -   shading interp
232 -   title ("Piano")
233 -   xlabel('Time (s)'), ylabel('Frequency (Hz)')
234 -   ylim ([100 400])
235 -   colormap(hot)
236
237 -   subplot(1,2,2)
238 -   pcolor(tslide_rec,(ks_rec/(2*pi)), spec_rec .'),
239 -   shading interp
240 -   title ("Recorder")
241 -   xlabel('Time (s)'), ylabel('Frequency (Hz)')
242 -   ylim ([600 1200])
243 -   colormap(hot)
244
245     %% Reproduce the music score/note on the piano
246 -   figure(5)
247 -   subplot(1,2,1)
248 -   plot(tslide_p, notes_piano,'o');
249 -   title ("Piano music score (200~350 Hz)")
250 -   xlabel('Time (s)'), ylabel('Music Note')
251 -   ylim ([200 350])
```

```
252 -    yticks([220.00 246.94 261.63 293.66 329.63 349.23])
253 -    yticklabels({'A4=220','B4=247','C4=262','D4=294','E4=330','F4=349'})
254
255 -    subplot(1,2,2)
256 -    plot(tslide_rec, notes_rec,'o');
257 -    title ("Recorder music score (600~1200 Hz)")
258 -    xlabel('Time (s)'), ylabel('Music Note')
259 -    ylim ([600 1200])
260 -    yticks([659.26 698.46 783.99 880.00 987.77 1046.5 1174.7])
261 -    yticklabels({'E5=659','F5=698','G5=784','A6=880','B6=988','C6=1046','D6=1175'})
262
263
```