Homework 2: Gábor transforms

Jiayi Wang

Abstract

In this report, we will perform experiments on Gábor Transform to see how different time filtering window widths and translations size affect the resolution in time and frequency components. We will also do the comparison between different Gábor windows. At last, we will see an application of Gábor Transform in real life by processing the song *Mary had a little lamb* on both the piano and recorder.

Introduction and Overview

In practice, the Gábor transform is computed by discretizing the time and frequency domain. The width of time filtering window and the size of translation play an important role in the time and frequency resolution. In this report, we will first perform experiments on Gábor Transform to see how different time filtering window widths and translations size can affect the resolution in time and frequency components. We will explore the different Gábor windows, for example: Mexican hat wavelet and Shannon window. At last, we will see an application of Gábor Transform by reproducing the music score of the song *Mary had a little lamb* on both the piano and recorder and then compare.

Theoretical Background

- **Gábor Transform (Short-time Fourier Transform)**

  The Gábor method, which is also known as short-time Fourier transform (STFT), is named after The Hungarian physicist/mathematician/electrial engineer Gábor Dénes (Physics Nobel Prize in 1971 for the discovery of holography in 1947). He was first to propose a formal method for localizing both time and frequency. His method involved a simple modification of the Fourier transform kernel. Thus, Gábor introduced the kernel:

$$g_{t,\omega}(\tau) = e^{i\omega\tau} g(\tau - t) \tag{1}$$

  where the new term to the Fourier kernel $g(\tau - t)$ was introduced with the aim of localizing both time and frequency. The Gábor transform, also known as the short-time Fourier transform (STFT) is then defined as the following:

$$\mathcal{G}[f](t,\omega) = \tilde{f}_g(t,\omega) = \int_{-\infty}^{\infty} f(\tau)\bar{g}(\tau - t)e^{-i\omega\tau} d\tau = (f, \bar{g}_{t,\omega}) \tag{2}$$

where the bar on G denotes the complex conjugate of function and the time window is centered on τ and with the width a.[1] The shorter the time filtering window, the less information there is concerning the frequency content. In contrast, longer windows retain more frequency components, but this comes at the expense of losing the time resolution of the signal.[2]

- **Wavelets**

  The term wavelet means little wave and originates from the fact that the scaling window extracts out smaller and smaller pieces of waves from the larger signal. Wavelet analysis begins with the consideration of a function known as the mother wavelet. In principle, the mother wavelet is designed to have certain properties that are somehow beneficial for a given problem. Thus, depending upon the application, different mother wavelets may be selected.[3]

  - *Gaussian*

    The simplest Gábor window to implement is a Gaussian time-filter. The following code constructs the windowed Fourier transform with the Gábor filtering function:

    $$g(t) = e^{-a(t-\tau)^2} \tag{3}$$

    where a is the width parameter and τ is the translation parameter.[4]

  - *Mexican Hat*

    The Mexican Hat Wavelet. One of the more common wavelets is the Mexican hat wavelet. This wavelet is essentially a second moment of a Gaussian in the frequency domain. The Mexican hat wavelet has excellent localization properties in both time and frequency due to the minimal time-bandwidth product of the Gaussian function. The function is defined as:

    $$g(t) = \frac{2}{\sqrt{3a}\pi^{\frac{1}{4}}}\left(1 - \left(\frac{t-\tau}{a}\right)^2\right)e^{-\frac{(t-\tau)^2}{2a^2}} \tag{4}$$

  - *Shannon*

    The Shannon filter is a step function with value of 1 within the transmitted band and 0 everywhere else. It suppresses all frequencies outside of a given filter box.

    $$g(t) = \begin{cases} 1, & if\ |t-\tau| \le a \\ 0, & if\ otherwise \end{cases} \tag{5}$$

[1] Page 333 from the course notes.
[2] Page 337 from the course notes.
[3] Page 339 from the course notes.
[4] Page 354 from the course notes.

Algorithm Implementation and Development

In Part I, we first load *handel* and all the codes that were provided to Matlab. Then we set *L* to be the length of the music which is 9 seconds and *n* to be *length(v)* which is the total sample points: 73113. Then I generate a vector *t2* which contains $n + 1$ points between 0 to *L* with spacing between the points is $\frac{L}{n}$, and let *t* be the first *n* points of *t2*, since the data is periodic. Then I rescale the frequencies *k* by $\frac{1}{L}$. The reason we use $\frac{1}{L}$ instead of $\frac{2\pi}{L}$ here is because we want the y axis to have the unit *Hz*. Since $\omega = 2\pi f$, then $f = \frac{\omega}{2\pi} = \frac{2\pi}{L} \cdot \frac{1}{2\pi} = \frac{1}{L}$. And the reason we use $[0: \frac{n-1}{2} \ -\frac{n-1}{2}: -1]$ is because *k* is an odd number here and we want it to match up with the ordering that FFT naturally does .

To produce the spectrograms of the piece of work, we set the width of the window *a* to be 1 and create a variable *tslide=0:0.1:L*, which indicated the positions of where the windows center at over the time span. Then we build a Gaussian filter and assign it to the variable *g* and use a for loop to extract the frequency information by multiplying the signal data *v* to the filter function *g* and use *fft* command to apply the fast Fourier transform on the signal. Then we use the *fftshift* command to shift the zero-frequency component to the center and save the frequency information into a matrix we create called *Sgt_spec*. Finally, with the help of the command: *pcolor*, we are able to produce the spectrogram.

In order to see how the window width and translation size will affect the spectrogram, we simply create an array *a_vec=[0.1 2 20]* which contains three different window widths and *trans=[0.05 0.5 1]*, which contains three different translation sizes. Then with the help with a for loop, we are able to generate the spectrogram with different window width and translation size, then compare the results.

To explore the different Gábor windows, such as Mexican hat and a step-function (Shannon) window, we simply change the expression of variable *g*. Then we can produce the spectrograms and do the comparison.

In Part II, to reproduce the music score, we basically use the same strategy we used in Part I to create the spectrograms. However, we change *L* to be the length of each song, and *n* to be the number of sample points of each music, that is *length(y)*. We will use $k = \frac{1}{L} * [0:\frac{n}{2} - 1 \ \ -\frac{n}{2}: -1]$ since *k* is an even number in both music pieces now and we want it to match up with the ordering that FFT naturally does. To get a good clean score, we will filter out overtones by using Gábor filtering with Gaussian window. Then we will adjust the window size, translation size, and range of the y-axis to produce the ideal spectrogram. After we obtain the

spectrogram with y-axis has the unit *Hz*, we are able to compare the music scale figure that was provided to reproduce the music scores and find the difference between a recorder and piano.

Computational Results

By setting the window width to be 20 and translation to be 0.1, we obtain the left most spectrogram in figure 1 which is okay in time resolution and frequency resolution, but both are not ideal. When we fix the translation size and increase the window width by decreasing *a* value (since larger *a* resulting in smaller *g* in equation 3), we can see increased frequency resolution is gained at the expense of the worse time resolution in figure 1.

In figure 2, when we fix the length of the filtering window and increase the translation size, we first notice that when the translation is 0.03 and 0.05, the spectrograms do not look that much different. This is a good illustration of "oversampling." When the translation is 1, we obtain a spectrogram with poor resolution in both time and frequency since we only can extract data from very few windows, and this is the consequence of "undersampling."

To explore different Gábor windows, we generate three spectrograms in figure 3 for Gaussian, Mexican Hat, and Shannon respectively with window width 40 and translation 0.1. One thing we notice is Gaussian window gives better time resolution, but worse frequency resolution compared to the other two. Mexican Hat and Shannon gives really good frequency resolution but really bad time resolution in this case.

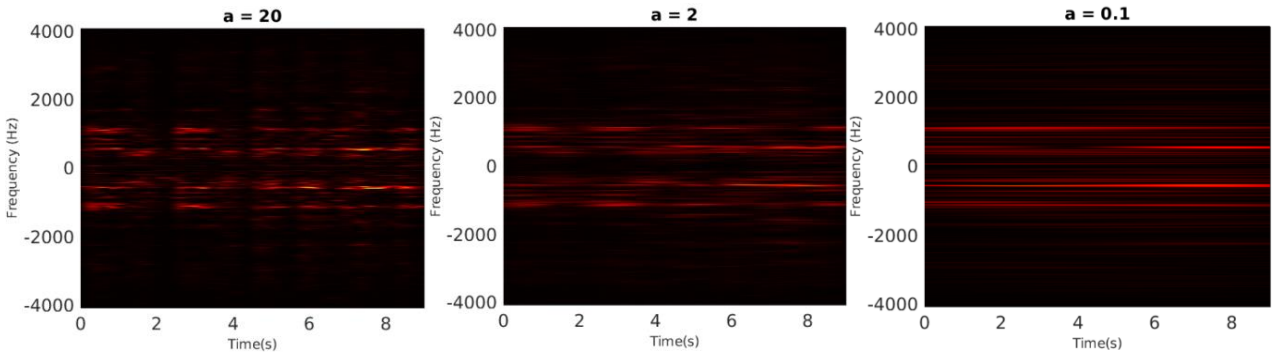

Figure 1: Spectrogram by Gaussian filtering with fixed translation size 0.1 and varying window width

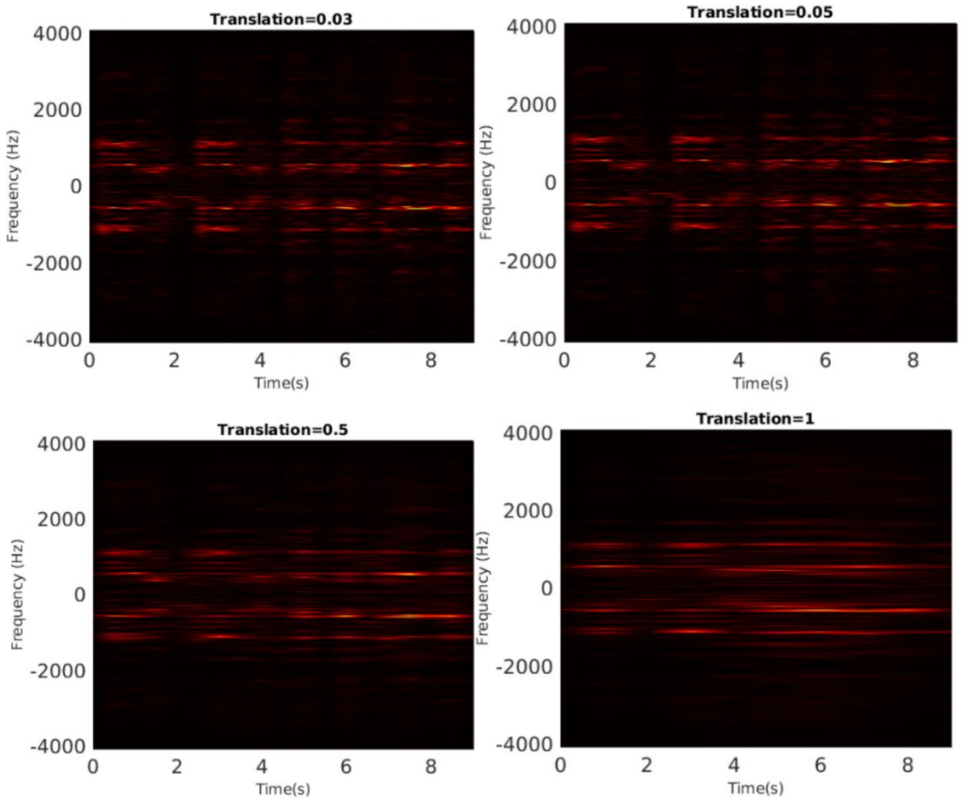

Figure 2: Spectrogram by Gaussian filtering with fixed window width 30 and varying translation size

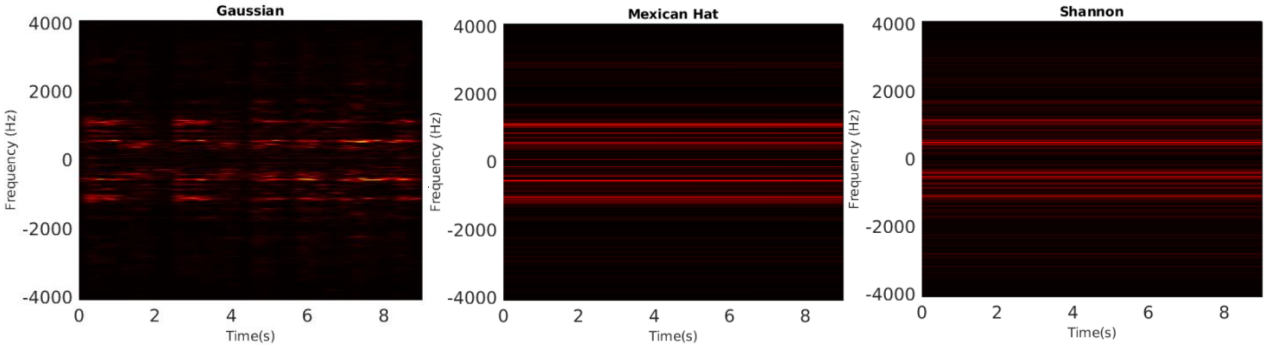

Figure 3: Spectrogram with different Gábor windows

After filtering out the overtones in given music on piano and recorder by using Gaussian window with window width: 40 and translation: 0.1, then adjust the window size, translation size, and range of the y-axis we obtain the spectrograms in figure 4. After reading off the frequency in Hz and comparing with the music scale, we obtain the music score for piano as: E D C D E E E D D D E E E E D C D E E E E D D E D C and for recorder as: B A G A B B B A A A B B B B A G A B B B B A A B A G. One thing we can notice from the spectrograms is, the shapes look similar. The recorder fundamental frequencies are a multiple of the piano fundamental frequencies. One difference from the two spectrograms is the

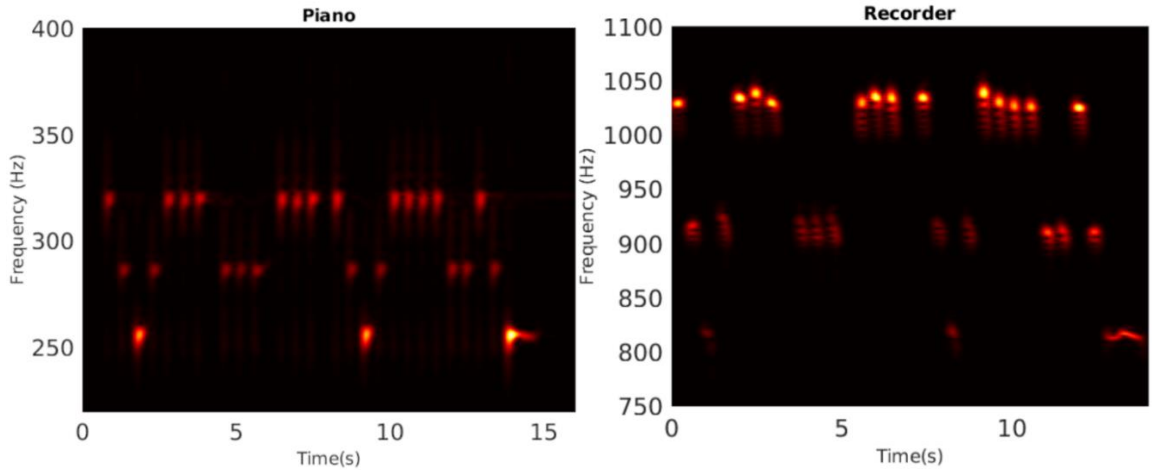

Figure 4: Spectrogram by piano and recorder with both window widths: 40 and translations: 0.1

notes at lowest frequency from piano are the brightest and the notes at highest frequency from recorder are the brightest in their own spectrograms respectively. So, if people play exactly the same notes on both the piano and recorder, although the overtones would occur at the same frequencies, people still can hear the difference between two instruments and identify what type of instrument is being played.

Summary and Conclusions

When doing Gábor transform, we saw the importance of choosing the right filter. Besides that, finding the best time filtering window width and translation size to avoid "oversampling" and "undersampling" are also really important in producing the ideal spectrogram so we can extract out both time and frequency information from the signal.

Appendix A. MATLAB functions used and brief implementation explanation

- *Y = fftshift(X)* rearranges a Fourier transform X by shifting the zero-frequency component to the center of the array.
- *Y = fft(X)* computes the discrete Fourier transform (DFT) of X using a fast Fourier transform (FFT) algorithm.
- *pcolor(X,Y,C)* specifies the *x*- and *y*-coordinates for the vertices. The size of C must match the size of the *x*-*y* coordinate grid. For example, if X and Y define an *m*-by-*n* grid, then C must be an *m*-by-*n* matrix.
- *colormap(map)* sets the colormap for the current figure to the colormap specified by map.
- *shading interp* varies the color in each line segment and face by interpolating the colormap index or true color value across the line or face.

Appendix B. MATLAB codes

```
% clear all; close all; clc
load handel
v = y';
t=(1:length(v))/Fs;
%figure(1)
% plot(t,v);
% xlabel('Time [sec]');
% ylabel('Amplitude');
% title('Signal of Interest, v(n)');
L=round(max(t)); n=length(v);
t2=linspace(0,L,n+1); t=t2(1:n);
k=(1/L)*[0:(n-1)/2 -(n-1)/2:-1];
ks=fftshift(k);

%%Spectrograms for varying window sizes
a_vec = [20 2 0.1];
for jj = 1:length(a_vec)
    a = a_vec(jj);
    tslide=0:0.1:L;
    Sgt_spec = zeros(length(tslide),n);
    Sgt_spec = [];
    for j=1:length(tslide)
        g=exp(-a*(t-tslide(j)).^2);
        Sg=g.*v;
        Sgt=fft(Sg);
        Sgt_spec(j,:) = fftshift(abs(Sgt));
    end
    figure(jj)
    pcolor(tslide,ks,Sgt_spec.'),
    shading interp
    set(gca,'Fontsize',16)
    colormap(hot)
    title(['a = ',num2str(a)],'Fontsize',15)
    xlabel('Time(s)','FontSize', 12), ylabel('Frequency (Hz)','FontSize', 12)
    pause(2)
end

%% Spectrograms for varying translations
trans=[0.03 0.05 0.5 1];
for jj=1:length(trans)
    a = 30;
    tslide=0:trans(jj):L;
    Sgt_spec = zeros(length(tslide),n);
    Sgt_spec = [];
    for j=1:length(tslide)
        g=exp(-a*(t-tslide(j)).^2);
        Sg=g.*v;
        Sgt=fft(Sg);
```

```
        Sgt_spec(j,:) = fftshift(abs(Sgt));
    end
figure(jj)
pcolor(tslide,ks,Sgt_spec.'),
shading interp
set(gca,'Fontsize',16)
colormap(hot)
title(['Translation=',num2str(trans(jj))],'Fontsize',12)
xlabel('Time(s)','FontSize', 12), ylabel('Frequency (Hz)','FontSize', 12)
end

%% Different Gabor windows
a = 40;
tslide=0:0.2:L;
Sgt_spec = zeros(length(tslide),n);
Sgt_spec = [];
for j=1:length(tslide)
% Gaussian
g=exp(-a*(t-tslide(j)).^2);
% Mexican Hat
%g=2/(sqrt(3*a).*pi^(1/4)).*(1-((t-tslide(j))/a).^2).*exp(-(t-tslide(j)^2/(2.*a.^2)));
% Step-function(Shannon)
%g=abs(t-tslide(j))<=a;
Sg=g.*v;
Sgt=fft(Sg);
Sgt_spec(j,:) = fftshift(abs(Sgt));
end
figure(1)
pcolor(tslide,ks,Sgt_spec.'),
shading interp
set(gca,'Fontsize',16)
colormap(hot)
title(['Gaussian'],'Fontsize',12)
%title(['Mexican Hat'],'Fontsize',12)
%title(['Shannon'],'Fontsize',12)
xlabel('Time(s)','FontSize', 12), ylabel('Frequency (Hz)','FontSize', 12)

%% Part 2
clear all; close all;clc
opengl('save', 'software')
[y,Fs] = audioread('music1.wav');
tr_piano=length(y)/Fs; % record time in seconds
% plot((1:length(y))/Fs,y);
% xlabel('Time [sec]'); ylabel('Amplitude');
% title('Mary had a little lamb (piano)');
% p8 = audioplayer(y,Fs); playblocking(p8);
v=y';
L=round(tr_piano);
n=length(y);
t2=linspace(0,L,n+1); t=t2(1:n);
```

```
k=(1/L)*[0:n/2-1 -n/2:-1];
ks=fftshift(k);
a =40;
tslide=0:0.25:L;
Sgt_spec = zeros(length(tslide),n);
Sgt_spec = [];
for j=1:length(tslide)
g=exp(-a*(t-tslide(j)).^2);
Sg=g.*v;
Sgt=fft(Sg);
Sgt_spec(j,:) = fftshift(abs(Sgt));
end
figure(1)
pcolor(tslide,ks,Sgt_spec.'),
shading interp
set(gca,'Fontsize',16)
colormap(hot)
xlabel('Time(s)','FontSize', 12), ylabel('Frequency (Hz)','FontSize', 12)
title(['Piano'],'Fontsize',12)
ylim([500 4000])
```