Gabor Filter Analysis of Audio Signals

Ashley Batchelor

Abstract

Gabor filters were used to generate spectrograms of audio data. For the first part, I analyzed 9 seconds of the opening of Handel's messiah using various window widths, window types, and translation step sizes and compared the results. For the second part, I analyzed a recording of the song "Mary Had a Little Lamb" as played by piano and as played by a recorder. For each recording, I generated a spectrogram and used the data from the spectrogram to derive a sheet music representation.

Sec. I. Introduction and Overview

Gabor filtering provides a useful means to generate time varying spectrograms of time varying signals such as audio signals. The Gabor uncertainty limit demonstrates the tradeoff between high resolution in the time domain and high resolution in the frequency domain.

Sec. II. Theoretical Background

A time varying signal f(t) may be transformed by a variation of a Fourier transform which uses the Gabor kernel:

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

With this kernel, the Gabor transform of the signal f(t) is defined as:

$$G[f](t,\omega) = \widetilde{f_g}(t,\omega) = \int_{-\infty}^{\infty} f(\tau) g(\tau - t) e^{i\omega\tau} d\tau$$

Various types of Gabor windows $g(\tau - t)$ defined by a window width a. For example, a Gaussian window:

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

a Mexican hat window:

$$g(\tau - t) = (1 - \frac{a(\tau - t)^2}{2}) e^{-a(\tau - t)^2}$$

or a rectangular window:

$$g(\tau - t) = \begin{cases} 1, for\ 0 < \tau - t < 1/2a \\ 0, otherwise \end{cases}$$

For a discrete digital signal f(t), sampled on a time and frequency lattice:

$$\nu = m\omega_0\ ;\ \tau = nt_0$$

A discrete kernel may be defined as:

$$g_{m,n}(t) = e^{i2\pi m\omega_0 t} g(t - nt_0)$$

A discrete Gabor transform is then defined as:

$$\tilde{f}(m,n) = \int_{-\infty}^{\infty} f(t)\bar{g}_{m,n}(t)dt$$

For the case where $0 < t, \omega_0 < 1$, the signal f(t) is oversampled and for the case $t, \omega_0 > 1$, the signal is undersampled.[1]

There is a tradeoff for high resolution in either the frequency domain or the time domain. The frequency variance and the time variance are constrained by the relation[2]:

$$\sigma_\omega \sigma_t \geq \frac{1}{4\pi}$$

This is a more general form of the well-known Heisenberg Uncertainty Principle that applies to any signal.

Sec. III. Algorithm Implementation and Development

Part 1

The initial data was a MATLAB file in the form of a one-dimensional array of 73,133 values sampled at a rate of 8,192 Hz over a total duration of 8.928 s. I defined set of n=73,133 Fourier modes. I scaled the wavenumbers by a factor of $2\pi/L$ for the FFT calculations, where L was then length of the audio data in seconds. For each spectrogram, I defined a Gaussian window, a Mexican hat window, and a rectangular window, and for each window, I generated a spectrogram for three values of the translation step.

Part 2

The initial data included two wav files, one with a piano sampled at a rate of 43,840 Hz over a total duration of 16 s and one with a recorder sampled at a rate of 44,837 Hz over 14 s. For each audio file, I generated spectrograms using a Gabor transform with a Gaussian window to analyze the frequency content to identify the fundamental frequencies of each instrument for each note of the song, and from this information, I generated a sheet music representation of the song.

Sec. IV. Computational Results

Part 1

[1] Kutz, J. Nathan. Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data, Oxford University Press, 2013: p 318

[2] Gabor, D. "Theory of Communication", J. IEE (London), vol.93, pp.429-457, 1946.

I generated spectrograms for a window width of 5000 s^{-2}, and translation step values of 1, 0.1, and 0.01 s. The Gaussian window. The Mexican hat window provided the fastest calculations of the three windows and also showed the finest detail in the frequency domain. All three windows showed distinct demarcation between the vocal words “Hallelujah” for the moderately sampled and oversampled cases. The rectangular window showed more of the higher harmonics than the other two windows. Strangely, it computed very slowly, perhaps a characteristic of the built-in rectangular pulse function in Matlab, such that I did not compute a spectrogram for an oversampled translation step. Running the code overnight was not long enough! Perhaps there is a better alternative code for applying a rectangular window.

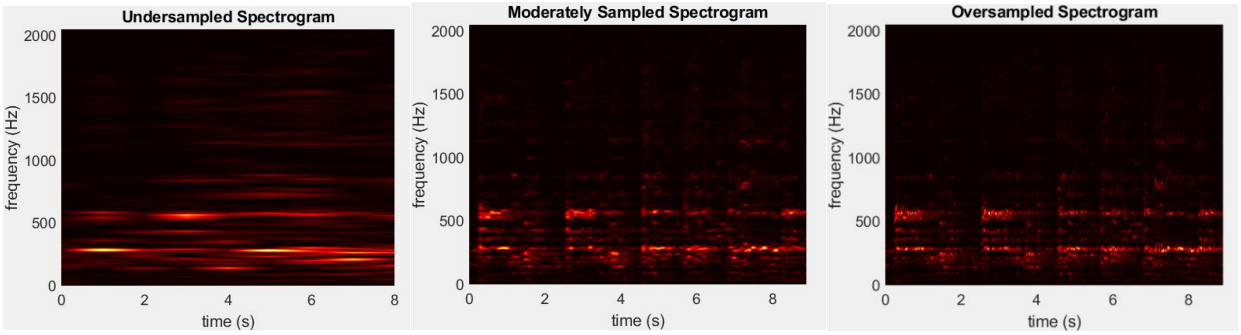

Figure 1 a), b), c), Gaussian window spectrograms for translation steps of 1, 0.1, and 0.01 s

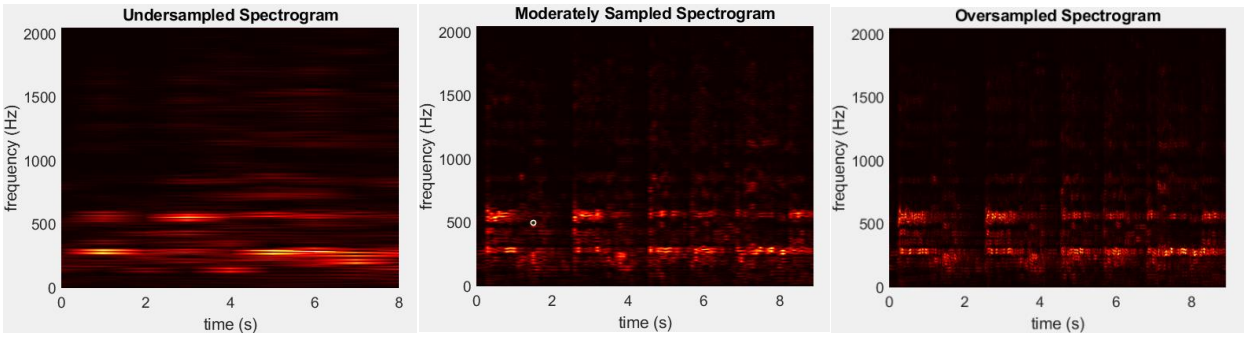

Figure 2 a), b), c), Mexican hat window spectrograms for translation steps of 1, 0.1, and 0.01 s

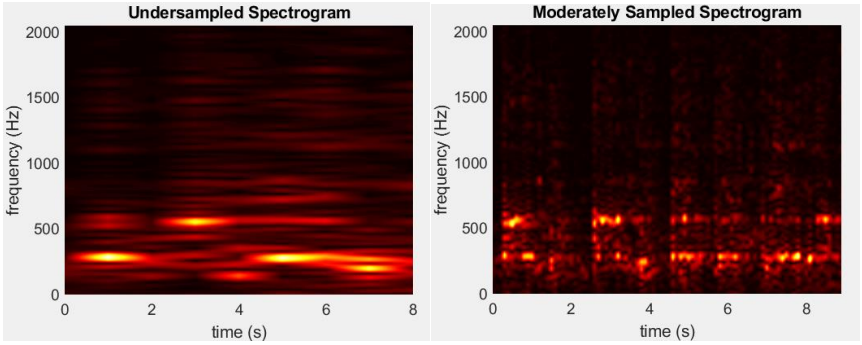

Figure 3 a), b), rectangular window spectrograms for translation steps of 1 and 0.1 s

Part 2

I had frequent memory overflows when running my code, so I downsampled the data by a factor of four for each wav file to save memory (10,960 Hz for the piano and 11,209 Hz for the recorder).

I used a translation step value of 0.1 s, and a Gaussian window width of 5,000 s^{-2}, which provided an appropriate balance of time and frequency resolution.

The recorder showed a weak third harmonic, but no other significant harmonics. Most of the spectral energy was in the fundamental. The piano on the other hand, showed second, third, and fourth harmonics with values near a tenth of the fundamental.

For each set of spectrogram data, I plotted the relative volume around the peak frequencies of the notes over the time sequence to locate the maxima which corresponded to the time each note was played. Then, I put the indices of these maxima into an array and wrote a for loop to iterate over this array and find the frequency max for each array. From each of these maximum frequencies I used the lookup table of notes from even temperament tuning with A defined as 440 Hz, to determine the musical notes.

The song "Mary Had a Little Lamb" only uses three different notes. The piano notes corresponded most closely to E3, D3, and C3 with respective average measured frequencies of 318.9, 286.6, and 255.8 Hz and respective standard deviations of 1.1, 0.9, and 0.9 Hz (12, 11, and 13 cents). The difference between the measured frequencies and even temperament (A4=440Hz) notes were respectively 57, 42, and 39 cents. The piano probably needs some tuning! The recorder notes corresponded most closely to C5, Bb4, and Ab4 with respective average measured frequencies of 1032, 912, and 816 Hz and respective standard deviations of 2.7, 1.8, and 1.8 Hz (9, 7, and 8 cents). The difference between the measured frequencies and even temperament (A4=440Hz) notes were respectively 24, 38, and 30 cents.

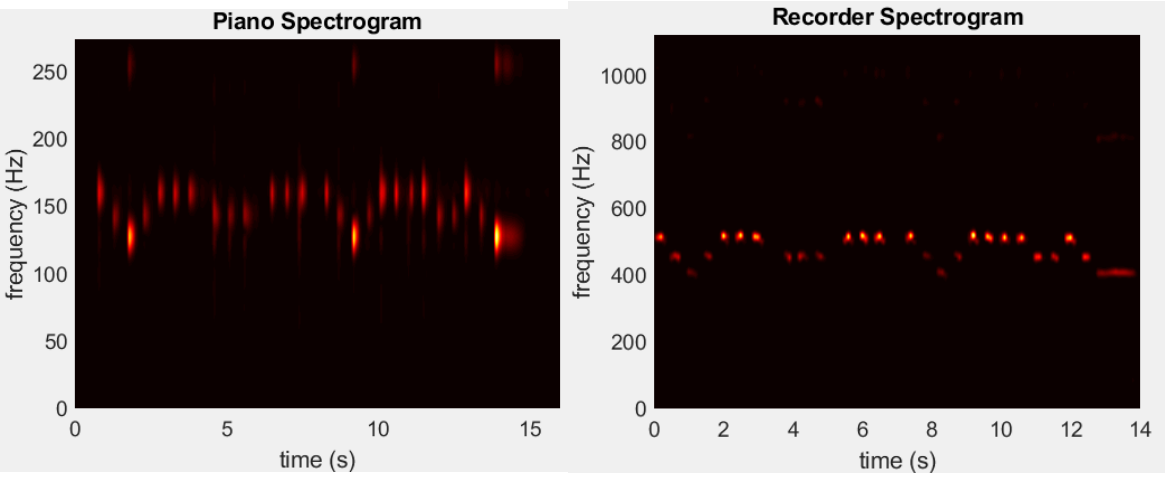

Figure 4 a), b), spectrograms for piano and recorder

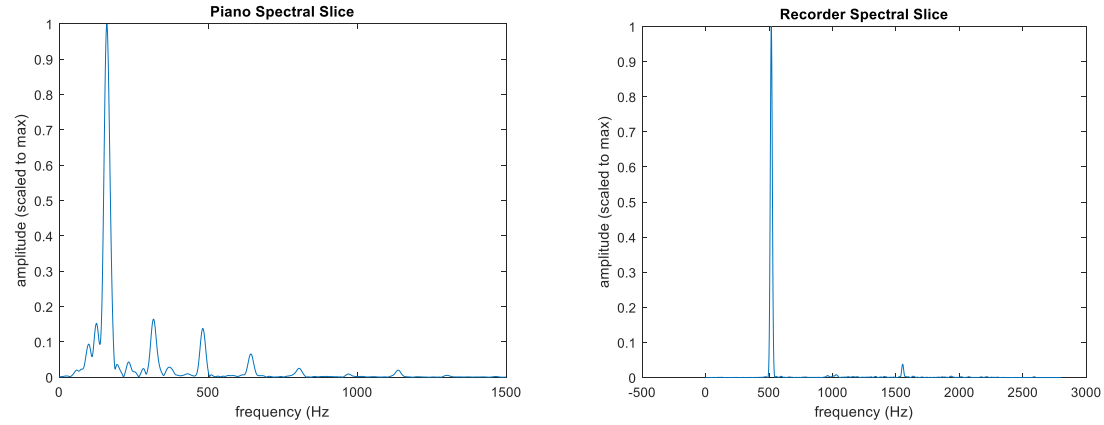

Figure 5 a), b), comparison of piano and recorder timbre

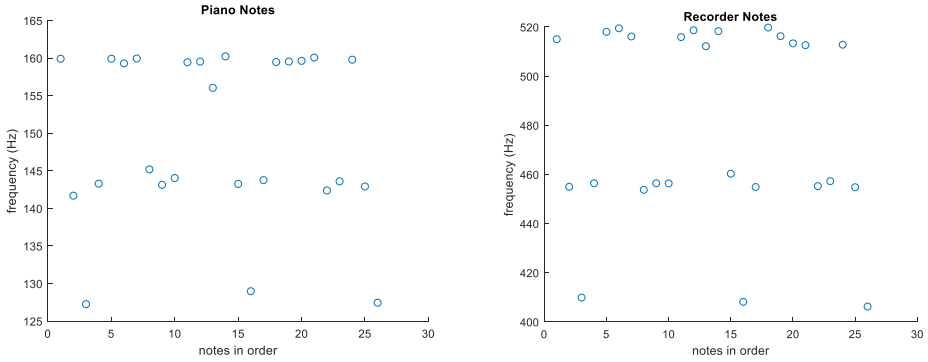

Figure 6 a), b), peak frequencies for each note for piano and recorder

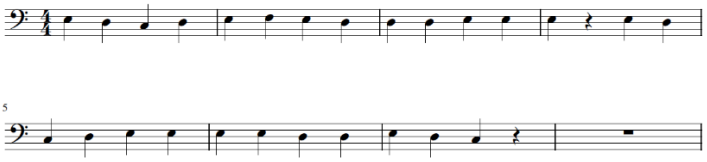

Figure 7 piano sheet music score

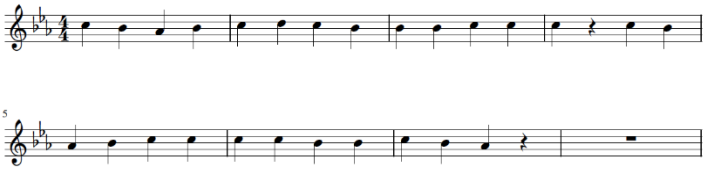

Figure 8 recorder sheet music score

Sec. V. Summary and Conclusions

Part 1

I compared spectrograms for three types of windows Gaussian, Mexican hat, and rectangular. The Mexican hat window was fast in computation time and gave similar results to the Gaussian window. The rectangular pulse needs to be programmed with care to avoid processing bottlenecks. The results did not appear to have any advantage over the other two windows. With undersampled translation steps, I

could not discern the rhythm of the music, but it was very apparent in the moderately sampled and oversampled cases.

Part 2

I was able to produce spectrograms of wav files of a piano and a recorder and I was able to easily determine the fundamental frequencies and harmonics of each note. I was able to reproduce a sheet music score for each set of data using moderately sampled spectrograms.

In both Part 1 and Part 2, I found that it was important to consider memory and computation time, especially when using highly sampled data and analysis.

Appendix A MATLAB functions used and brief implementation explanations

audioread() – This opens a wav file as an array.

decimate() – This reduces the sample rate of an array

fft() – This is the Fast Fourier Transform.

fftshift() – This shifts the FFT results such that the zero frequency components are at the center of the array.

max() – This returns the maximum value in an array.

Appendix B MATLAB codes

Part 1

```
clear all; close all; clc

L=73133/8192; n=73113;
t2=linspace(0,L,n+1); t=t2(1:n);
k=(2*pi/L)*[0:n/2 -n/2:-1]; ks=fftshift(k);

windowWidth = 0.01;
translationStep = 0.1;

kmax = n/(4*L);

load handel
v = y'/2;

Vgt_spec=[];
tslide=0:translationStep:L;

for j=1:length(tslide)
    %g=exp(-1*windowWidth*(t-tslide(j)).^2); % Gabor
    %g=(1-2*windowWidth*(t-tslide(j)).^2).*exp(-1*windowWidth*(t-
tslide(j)).^2);
    %Mexican hat
    g=rectangularPulse(-1*windowWidth/2,windowWidth/2,t-tslide(j));
    Vg=g.*v; Vgt=fft(Vg);
    Vgt_spec=[Vgt_spec; abs(fftshift(Vgt))];
end

pcolor(tslide,ks/(4*pi),Vgt_spec.'), shading interp
set(gca,'Ylim',[0 kmax],'Fontsize',[14])
colormap(hot)
xlabel('time (s)')
ylabel('frequency (Hz)')
title('Moderately Sampled Spectrogram')
```

Part 2

```
clear all; close all; clc

windowWidth = 5000;
```

```
translationStep = 0.1;

tr_recorder=14; % record time in seconds
y=audioread('music2.wav');
y=decimate(y,4); %Reduce sample rate to save memory
Fs=length(y)/tr_recorder;

L=length(y)/Fs; n=length(y);
t2=linspace(0,L,n+1); t=t2(1:n);
k=(2*pi/L)*[0:n/2-1 -n/2:-1]; ks=fftshift(k);

kmax = n/(10*L);

v = y'/2;

Vgt_spec=[];
tslide=0:translationStep:L;
for j=1:length(tslide)
    g=exp(-1*windowWidth*(t-tslide(j)).^2); % Gabor
    Vg=g.*v; Vgt=fft(Vg);
    Vgt_spec=[Vgt_spec; abs(fftshift(Vgt))];
end

%pcolor(tslide,ks/(4*pi),Vgt_spec.'), shading interp
%set(gca,'Ylim',[0 kmax],'Fontsize',[14])
%colormap(hot)
%xlabel('time (s)')
%ylabel('frequency (Hz)')
%title('Recorder Spectrogram')

notes =
[3,8,11,17,21,26,30,40,43,49,57,61,65,75,79,84,89,93,97,102,107,112,116,121,1
25,130];
frequencyIndices = [];

for j = 1:26
    [C,I]= max(Vgt_spec(notes(j),length(y)/2:length(y)));
    frequencyIndices = [frequencyIndices, I+length(y)/2];
end

%example of a spectral slice
%plot(ks(length(y)/2:length(y))/(4*pi),Vgt_spec(21,length(y)/2:length(y))/max
(Vgt_spec(21),length(y)/2:length(y)))
%xlabel('frequency (Hz)')
%ylabel('amplitude (scaled to max)')
%title('Recorder Spectral Slice')

xaxis = linspace(1,26,26);
scatter(xaxis,ks(frequencyIndices)/(4*pi))
xlabel('notes in order')
ylabel('frequency (Hz)')
title('Recorder Notes')
```

```
clear all; close all; clc

windowWidth = 5000;
translationStep = 0.1;

tr_piano=16; % record time in seconds
y=audioread('music1.wav');
y=decimate(y,4); %Reduce sample rate to save memory
Fs=length(y)/tr_piano;

L=length(y)/Fs; n=length(y);
t2=linspace(0,L,n+1); t=t2(1:n);
k=(2*pi/L)*[0:n/2-1 -n/2:-1]; ks=fftshift(k);

kmax = n/(40*L);

v = y'/2;

Vgt_spec=[];
tslide=0:translationStep:L;
for j=1:length(tslide)
    g=exp(-1*windowWidth*(t-tslide(j)).^2); % Gabor
    Vg=g.*v; Vgt=fft(Vg);
    Vgt_spec=[Vgt_spec; abs(fftshift(Vgt))];
end

pcolor(tslide,ks/(4*pi),Vgt_spec.'), shading interp
set(gca,'Ylim',[0 kmax],'Fontsize',[14])
colormap(hot)
xlabel('time (s)')
ylabel('frequency (Hz)')
title('Piano Spectrogram')

notes =
[9,14,19,24,29,34,39,47,52,57,66,71,75,85,88,94,98,102,107,112,116,121,126,13
0,135,140];
frequencyIndices = [];

for j = 1:26
    [C,I]= max(Vgt_spec(notes(j),length(y)/2:length(y)));
    frequencyIndices = [frequencyIndices, I+length(y)/2];
end

%example of a spectral slice
%plot(ks(87680:175360)/(4*pi),Vgt_spec(9,87680:175360)/max(Vgt_spec(9,87680:1
75360)))
%xlabel('frequency (Hz))
%ylabel('amplitude (scaled to max)')
%title('Piano Spectral Slice')
%xlim([0 1500])

%xaxis = linspace(1,26,26);
%scatter(xaxis,ks(frequencyIndices)/(4*pi))
%xlabel('notes in order')
%ylabel('frequency (Hz)')
%title('Piano Notes')
```