Time-Frequency Analysis: Applications of the Gabor Transform

Eli Miller

March 15, 2019

Abstract

In this report, we explore time-frequency analysis beyond the Fourier Transform by looking more closely into how frequency signatures change over time. We use the Gabor Transform as a tool to explore how audio signatures of musical works. We are able to leverage this analysis to extract musical score data from two different recordings of the same piece, and look into the differences in musical signatures of two different instruments

1 Introduction and Overview

In this report I first review theoretical background on the Gabor Transform. I then discuss it's implementation and use for numerical computation. Then, using data from audio files, I construct spectrograms looking at how filter width and sampling frequencies play a role in the amount of detail we can capture in time and frequency domains. Topics of over and under-sampling are explored in relation to this method of analysis.

The second part of this analysis looks into backing out musical notes of a piece by using this spectral analysis. With some knowledge of the systems in question (piano and recorder), we can use knowledge of where we expect to find frequency signatures to help filter our data and inform our decisions of where to look further.

In discussing the timbre of instruments, spectrograms provide a very nice, visual representation of the relative strengths of frequency components over time. As a result, we observe the differing levels of overtones produced by these instruments and discuss how these signatures are instrument-specific, rather than note-specific.

2 Theoretical Background

Lets begin with some signal in time $S(t)$. In previous analysis, we simply took the Fourier Transform $\hat{S}(t)$ to transform our signal to the frequency domain. While useful for signals stationary in

the frequency domain, the Fourier Transform can be seen as "too blunt of an instrument" for most time-frequency analysis. The FFT gives *all* frequencies present in a signal, but with no information about where they occur ed in time. As a result we employ the Gabor Transform, a windowed Fourier Transform, to resolve frequency information about a signal as it develops over time. Lets dive into the math.

The basic idea of the Gabor Transform is to take a sliding window in time, take the FFT within each window of our signal, and stitch together the frequency components of each window to create a spectrogram. The functional form is as follows

$$G[f](t,\omega) = \tilde{f}_g(t,\omega) = \int_{-\infty}^{\infty} f(\tau)g(\tau - t)e^{-i\omega\tau}d\tau \tag{1}$$

A more involved derivation and explanation of the properties of the Gabor Transform can be found in *Data-driven modeling & scientific computation: methods for complex systems & big data*.

Like Discreet Fourier Transforms, while the transform is defined over the entire real line, for computation we will only look at a finite domain $t \in [0, L]$. Likewise, in the frequency domain, we limit the frequency domain $k \in [-\frac{n\pi}{2L}, \frac{n\pi}{2L}]$, which has been scaled by π/L because of how the FFT algorithm expects a 2π periodic domain.

This definition of the Gabor Transform allows for many applications of different window shapes and widths. A few such windows are the Gaussian Window

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

Another useful window in signal processing is the Ricker Wavelet, or Mexican Hat Wavelet

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

A final filter window is the simple step function. This can be expressed as piece-wise function with height 1, centered at τ

$$g(x) = \begin{cases} 0 & 0 \le t < \tau - \sigma/2 \\ 1 & \tau - \sigma/2 \le t \le \tau + \sigma/2 \\ 0 & \tau + \sigma/2 < t \le L \end{cases} \tag{4}$$

By changing the window type and width, we explore the different resulting spectrograms

3 Algorithmic Implementation and Development

3.1 Data Preparation

To understand the data, we need to understand what the time series measurements represent. With knowledge of the sampling frequency F_s, we can find the length and rate of our audio samples. The length of the signal can be given

$$L = n/F_s \tag{5}$$

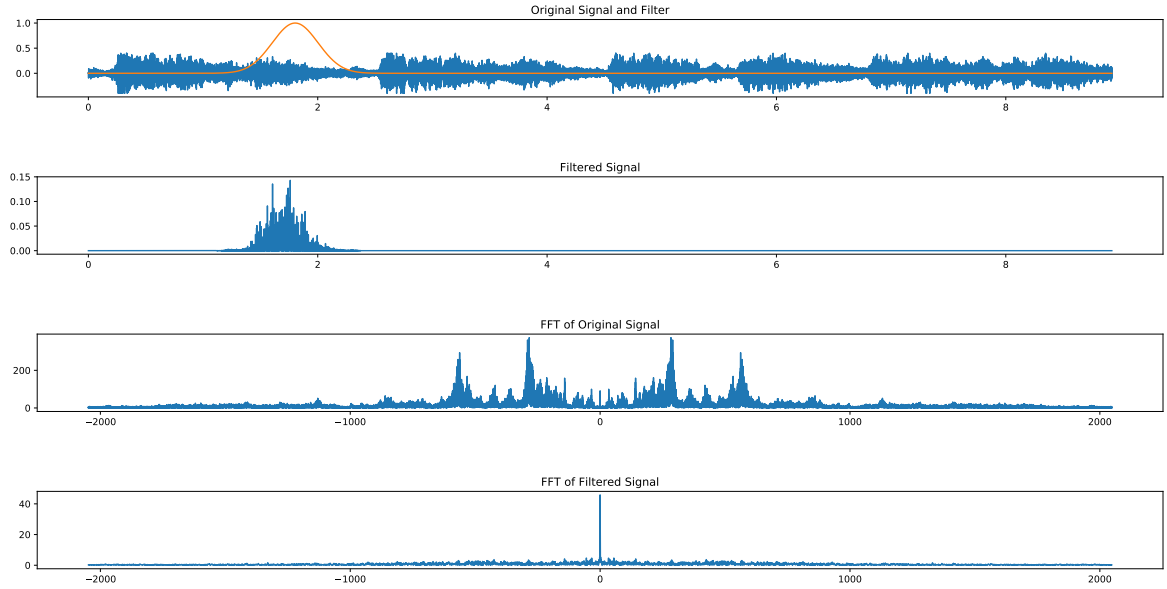


Figure 1: Visualization of Gabor Transform at $\tau = 1.785$

where n is the number of points in our measurement, and L is the length of signal in seconds.

When we use the Fourier Transform, it expects a 2π periodic domain, and outputs frequencies in Radians/ seconds. To convert to musical notes, we use a scale in Hertz. As a result, we need to re-scale the frequency components by dividing by 2π to get units of Hz. Thus our shifted Frequency components are given $k \in [-\frac{n}{2L}, -\frac{n}{2L}]$.

3.2 Gabor Transform

Lets look at a window location $\tau \in [0, t]$. Lets call the number of time-steps m. For each value of τ, we will filter our time signal with a filter $g(t = \tau)$, and compute the filtered signal

$$S_{filt} = S(t) * g(t = \tau) \tag{6}$$

We then take the Fourier Transform of this filtered signal S_{filt}. After looping through each time-step, we will have an array of size(n, m), where n is the number of measurements in $S(t)$ and m is the number of time-steps.

As discussed in the *Assignment 1*, we continue to take care with shifting the FFT and taking the absolute value. An example visualization of the Gabor Process is seen in Figure 1.

4 Computational Results

The first piece used for exploration is a snippet of Handle's "Hallelujah" . We will use it to explore creating spectrograms with the Gabor Transform. The second pieces used are two recordings of the

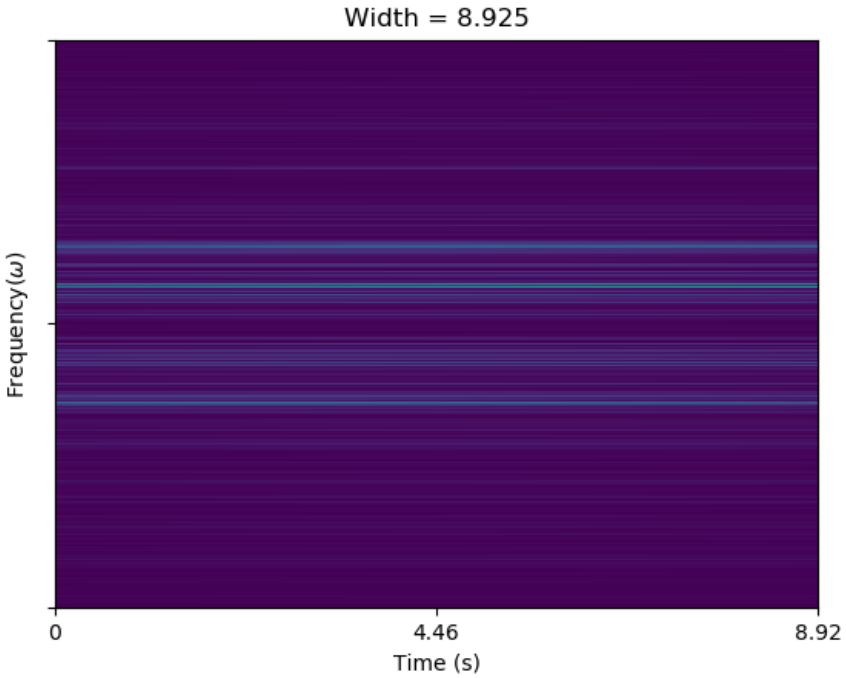


Figure 2: Wide Filter: very detailed in the frequency domain, but provides almost no information in the time domain

hit classic ”Mary Had a Little Lamb” performed by the critically acclaimed J. Nathan Kutz where we use the spectrogram to extract the music score and discuss instrument timbre

4.1 Exploration of Filter Width

One of the parameters that we have direct control over when preforming time-frequency analysis is filter width. The Gabor Transform holds value over the Fourier Transform because we are able to gain insight into the frequency components of our signal at different points in time, thus allowing us to analyze signals that are non-stationary in time. For the following analysis, we will use a fixed number of 100 time-steps using a Gaussian filter shape, and explore using a range of different widths of filters and how these changes affect the corresponding spectrograms.

In these three figures, we see how filter width is tied directly to time-and frequency resolution. With a filter as wide as our sample, we can see that the Gabor Transform almost looks like the FFT of our entire sample; we do not get any new information about when specific frequencies are present. As our filter gets narrower, we see information in the time domain. One side-effect of narrowing our filter is that we are not able to pick up low frequency signatures because the wavelength of these low frequencies is too long to ”fit” in the narrow window. This can be seen by the dark band in 4. The low frequencies that we see in 2 are not present as a result of using such a narrow window.

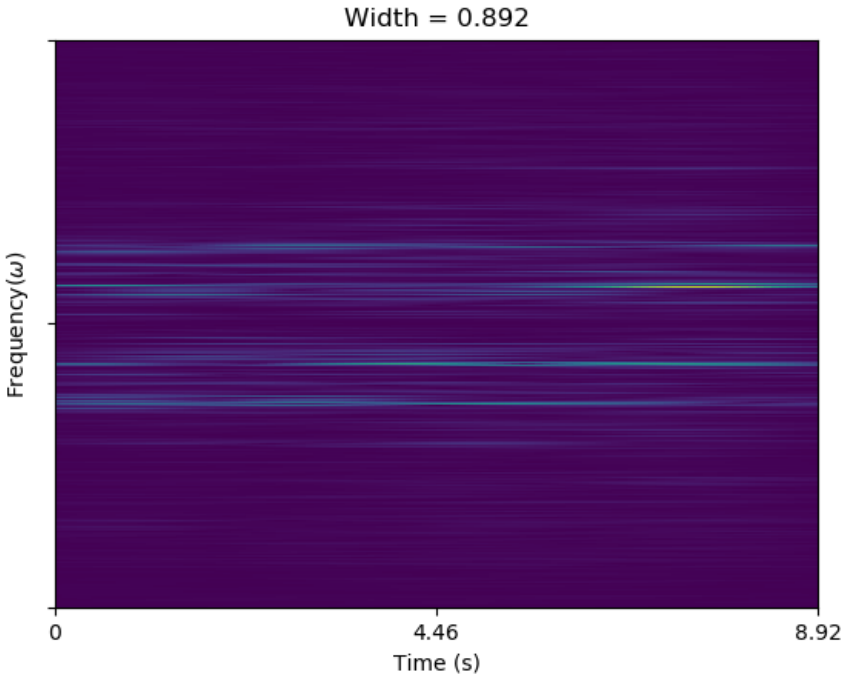


Figure 3: Mid-Width Filter: we see blurring in the frequency domain, and more detail in the time domain

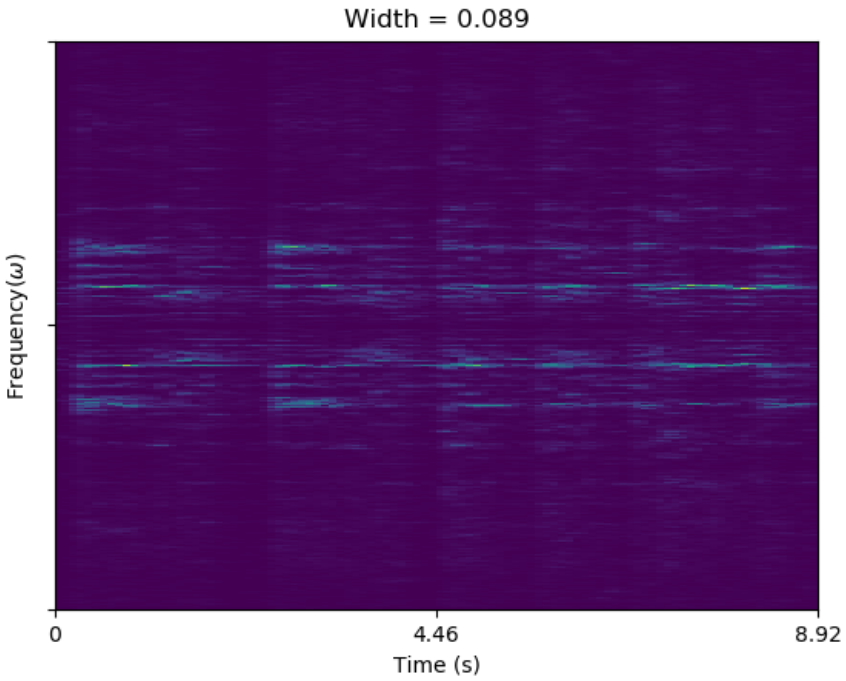
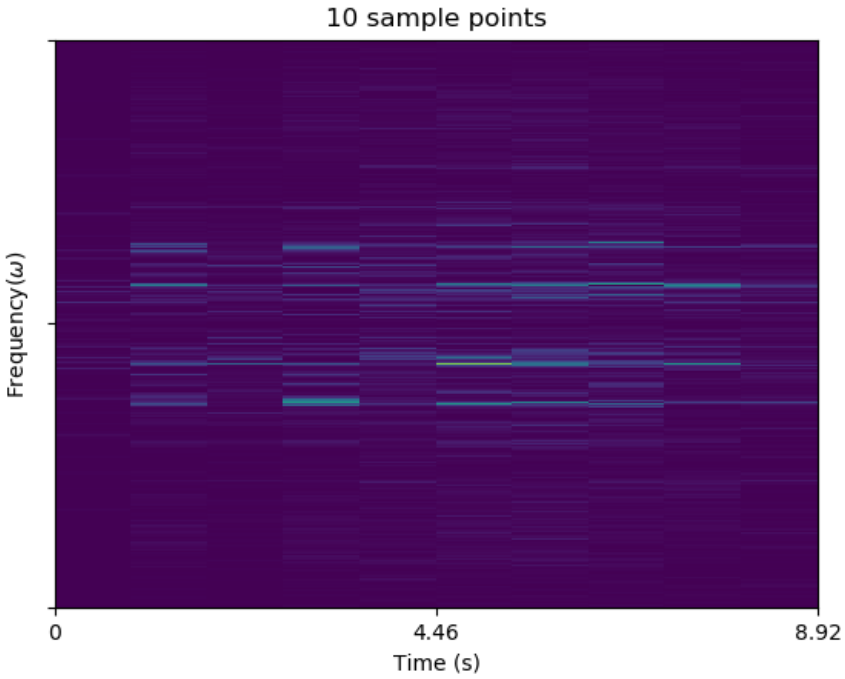


Figure 4: Narrow Filter: we get usable data in the time domain with some blurring in the frequency domain.

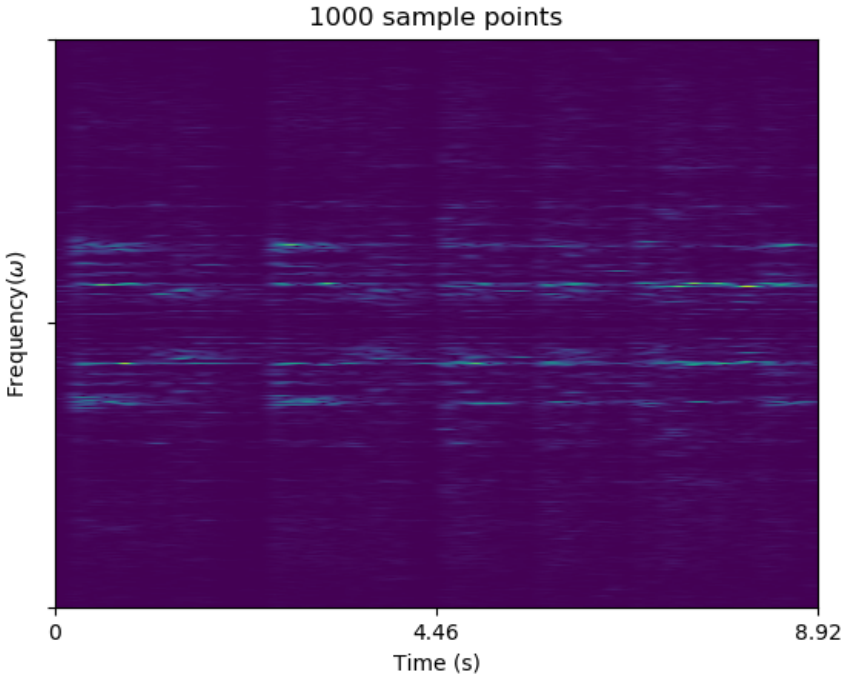
4.2 Over and Under Sampling

Whereas we used a fixed number of time-steps to explore filter width, we will use a fixed filter width to explore over/under sampling of our data. We will use the width from Figure 4 of $\sigma = \frac{L}{100} = 0.0892$. An example of under-sampling can be seen in Figure 5 When we only use 10 time-steps (a bit faster than one a second) we see that our output is very coarse in the time domain and does not represent the amount of information that we could extract from our data. Because we did not use a wider filter, we did not get any usable benefit of under-sampling. In contrast, we are able to take more time-steps than necessary and over-sample our data. This can be seen in Figure 6. While our time resolution appears very good, we are getting a lot of redundant data and are not getting any benefit of taking such a small time-step. The reason that this happens is because of how our time-frequency analysis is bounded by the Heisenberg Uncertainty Principle; essentially, we cannot gain any new information from the data by sampling more often.



points.png

Figure 5: Under-sampling in the time domain



points.png

Figure 6: Over-sampling in the time domain

4.3 Exploration of Filter Window

Because the Gabor Transform is defined for any generic filter window $g(\tau - t)$, we can adjust our filter shape for different applications in our analysis.

4.3.1 Gaussian

The window used for all of the previous discussion has been the Gaussian window as defined in Equation 2. Because of its gradually decreasing shape, id does a good job at smoothing between time-steps

4.3.2 Mexican Hat

Another useful window is the Ricker Wavelet or Mexican Hat Wavelet as defined in Equation 3. We can see that for the same filter "width" (in terms of standard deviation) the Mexican Hat window does a much better job of retaining resolution in time while not blurring the signal in the frequency domain.

4.3.3 Step

The Step window subjects our spectrogram to the effects of truncation of periodic signals. Because of the hard edges of the window, if our signal's frequency of interest does not align with the window at each time-step, we get some delay and coarseness in the time domain. When sampled at a critical rate, the step function can be useful, however it appears very sensitive to over and under-sampling as a function of its shape.

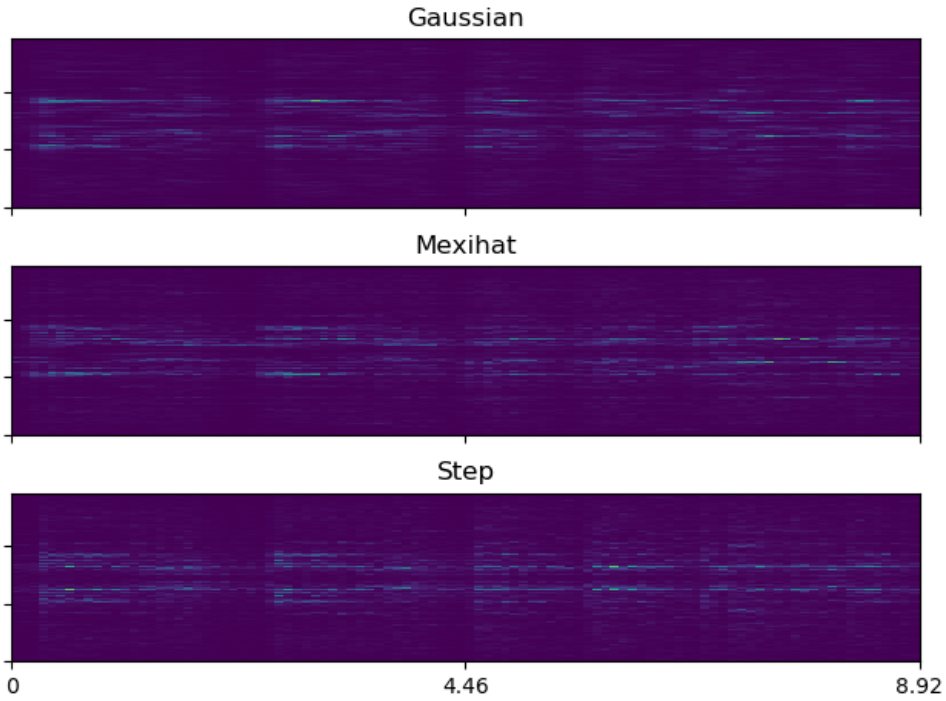


Figure 7: Comparison of different window shapes

4.4 Music Score

We can use our knowledge of the system in question (recorder and piano) to aid in our analysis of the musical compositions. Specifically, we know that recorders and pianos are only capable of producing notes within a specific range. We can use this fact to trim our spectrograms. For the piano, after trimming, we still need to zoom in by hand to find the correct portion of the spectogram. We are looking for the lowest frequency signature (the fundamental frequencies ω_0) associated with our melody. We can repeat this process with the recorder recording. Let's assume that the piece was played on either an alto or soprano recorder. This limits our possible notes from $[F_4, G_7]$ which corresponds to a frequency range of approx. $[349, 3136]$Hz. Again, zooming in manually and looking for the fundamentals, we can determine the notes played in these scores. Plots of the resultant parts of the spectrograms are see in Figure 8.

In determining the music scores, we relied on finding the closest approximate note associated with our determined frequency. These scores are not perfect matches when played in tune, but rather a close reconstruction of the pieces of interest.

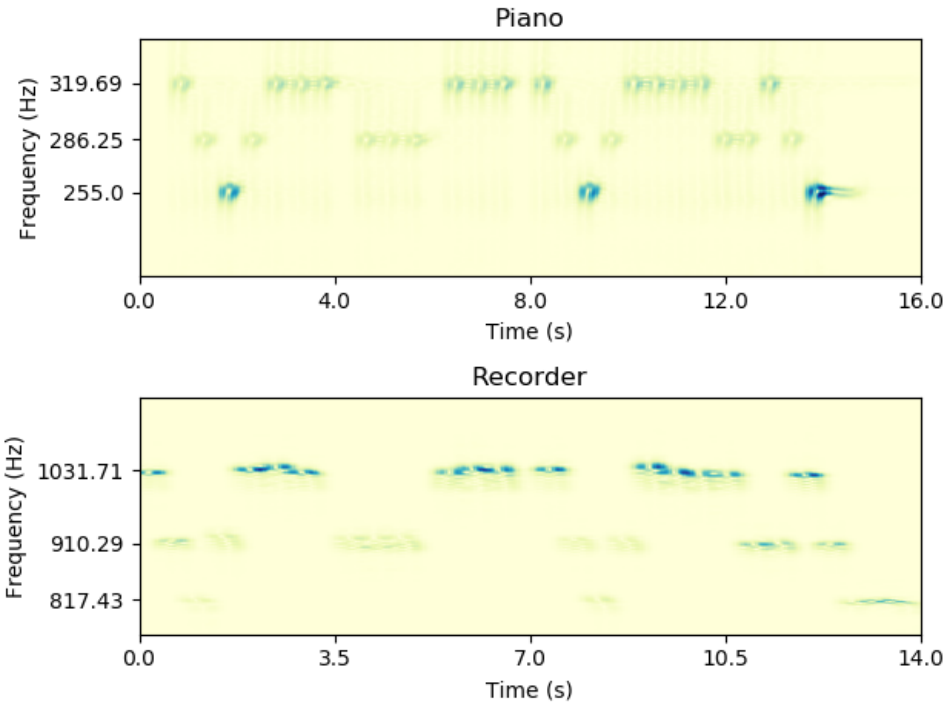


Figure 8: Frequency values for notes played on each instrument

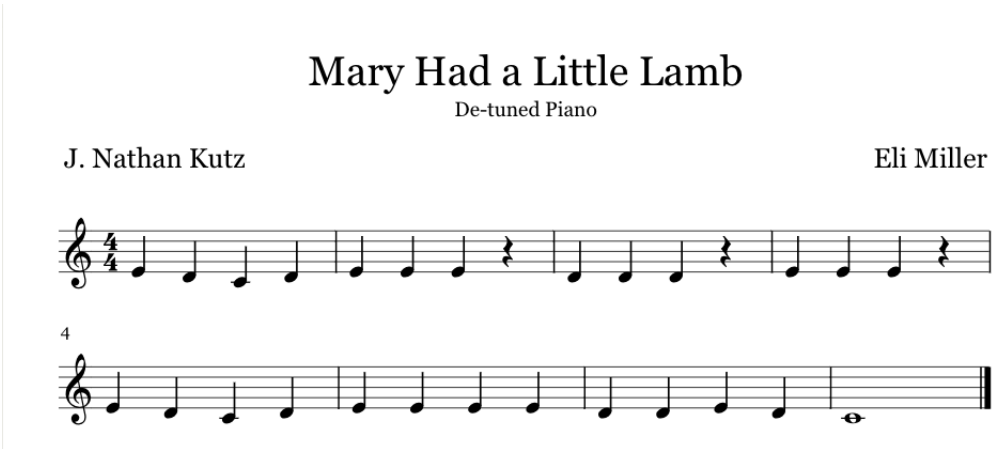


Figure 9: Formal piano score from spectrogram

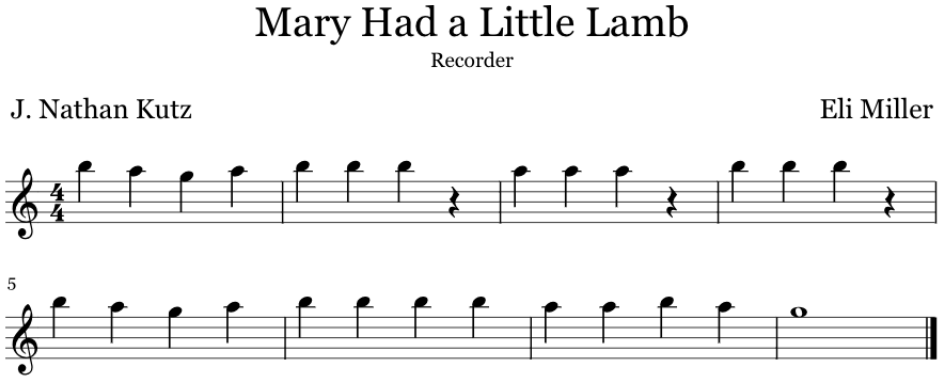


Figure 10: Formal recorder score from spectrogram

4.5 Instrumental Differences

The difference between recorder and piano is the relative amount of frequencies of the overtones present in their signals. When an instrument plays a note of frequency ω_0 we get overtones at $2\omega_0, 3\omega_0, etc.$ This can be seen in the spectrogram of each instrument.

The Piano has a very ”messy” frequency signature. Because of the way that a piano makes sound (hammers striking strings), we get lots of overtone content close to the fundamental frequency. In a grand piano, each hammer strikes either 2 or 3 strings all tuned to the same note. As the strings age, they each drift from producing the precise frequency, contributing to the dirtier sound of the piano.

In contrast, the Recorder has a ”cleaner” frequency signature. This can be seen in how there are less frequency components near the fundamental frequency. This makes sense physically, as recorder can be thought of as producing standing waves through resonance. As a result, the corresponding frequencies are wider spread and do not have the fundamental dissonance inherent to the multiple strings of the piano.

5 Summary and Conclusions

In Summary, we looked at how the Gabor Transform can be useful in creating spectrograms for time-frequency analysis. The parameters that are important in this analysis is the filter width and shape, and the number of time-steps over which we are sampling. We explored how filter width and shape effect the corresponding spectrograms and traps to watch out for when trying to increase resolution. In particular, by narrowing the window width, we limit the low frequencies that our spectrogram is able to ”see”. Additionally, we looked at each window type and applications / limitations of each in the context of spectrogram generation. For a final application, we used the spectrogram to identify the nores played in a musical composition by two different instruments, and use our output to gain insight into the idea of timbre.

Next steps may include a more robust way of determining the frequencies present in a signal. At present, we are using interactive plots to determine the frequencies present in a signal, and each set of frequencies was determined essentially by hand. In addition, it would be useful to define a better Object-Oriented way to creating spectrograms for a given input and set of parameters of interest. Perhaps this could include wavelet superposition of different window lengths.

6 References

J. N. Kutz, *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

7 Appendix A: Functions Used

For this analysis, I relied heavily on the numpy and matplotlib libraries for Python, and a bit on the scipy.io module. Below are some specific functions used.

`np.fft.fft`: returns the Fast Fourier Transform

`np.fft.fftshift`: returns a zero-centered frequency domain representation of our data.

`scipy.io.loadmat`: loads matlab files into a dictionary with relevant information. The key 'data' extracts the necessary variables.

`matplotlib.pyplot.pcolormesh`: similar to pcolor in Matlab

`matplotlib.pyplot.ioff()`: turns off interactive mode for pyplot. This is useful for speed of plot creation in loops

`matplotlib.pyplot.clf()`: clears the current figure. Useful for garbage collecting associated with the pyplot module storing data when not needed.

8 Appendix B: Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Feb  4 13:29:53 2019

@author: elimiller
"""
import numpy as np
from numpy import pi
import matplotlib.pyplot as plt
```

```
import scipy.io
plt.ioff()
#plt.ion()
#%%

def fft2flip(args):
        ReturnThis = np.abs(np.fft.fftshift(np.fft.fft(args)))
        return ReturnThis

def gaussian(unfiltered, t, location, width=.2):
    #scipy.signal.gaussian
    sigma = width
    coeff = (1/(sigma*np.sqrt(2* pi)))
    denom = 2 * sigma**2
    gauss_filter = coeff * np.exp(-(t-location)**2 / denom)
    return gauss_filter * unfiltered

def mexihat(unfiltered, t, location, width=.2):
    sigma = width
    t_instant = t - location
    psi = 2/(np.sqrt(3*sigma)*pi**(1/4)) * (
            1 - (t_instant)/(sigma)**2) * np.exp(
                    -(t_instant**2)/(2*sigma**2))
    return psi * unfiltered

def step(unfiltered, t, location, width=.2):
    bool_array = np.abs(t-location) < width/2
    stepfilt = np.array(bool_array * 1, dtype='float64')
    return stepfilt * unfiltered

def spectrogram(
        v, fs, width, samplemod=1,
        window='gaussian', makefig=False, numpts='100'):

    window_dict = {'gaussian': gaussian, 'mexihat': mexihat, 'step': step}
    L = len(v)/fs
    t = np.arange(0, len(v))/fs
    tshift = np.linspace(0, L, numpts)
    n = len(v)
    k = pi/L * np.concatenate((np.arange(0, (n/2)), np.arange(-n/2, 0)))
    ks = np.fft.fftshift(k)/(2*pi)
    gabor = np.zeros((len(t),len(tshift)))

    for j in range(len(tshift)):
        v_gabor = window_dict[str(window)](v, t, tshift[j], width=width)
        v_gabor_trans_shift = fft2flip(v_gabor)
        gabor[:,j] = v_gabor_trans_shift
        if makefig:
```

```
            #make figure for looking at how gabor works
            if j == 20:
                fig, axs = plt.subplots(4,1)
                titles = [
                        'Original Signal and Filter',
                        'Filtered Signal',
                        'FFT of Original Signal',
                        'FFT of Filtered Signal']
                axs[0].plot(t, v)
                axs[0].plot(
                        t, gaussian(np.ones_like(v), t, tshift[j])/np.amax
                        gaussian(np.ones_like(v), t, tshift[j])
                        )
        )
                axs[1].plot(t, v * gaussian(v, t, tshift[j]))
                axs[2].plot(ks, fft2flip(v))
                axs[3].plot(ks, fft2flip(v*gaussian(v, t, tshift[j])))

#                plt.suptitle('Visualization of Gabor Transform')

                count = int(0)
                for title in titles:
                    axs[count].set_title(title)
                    count += 1
                plt.tight_layout()
    return gabor
#%%
load_dict = scipy.io.loadmat('handel')
v = np.reshape(load_dict['y']/2, len(load_dict['y']))
v = v[:-1]
#k = pi/L * np.concatenate((np.arange(0, (n/2)), np.arange(-n/2, 0)))
#ks = np.fft.fftshift(k)
fs = 8192
L = len(v)/fs
n = len(v)
k = pi/L * np.concatenate((np.arange(0, (n/2)), np.arange(-n/2, 0)))
ks = np.fft.fftshift(k)

ratio = np.array([1, .5, .1, .01])
widthvals = len(v)/fs * ratio

#%%
# Filter Width

for width in widthvals:
    gabor_width = spectrogram(
            v, fs, width, samplemod=1.5, window='gaussian')
    fig = plt.figure()
```

```
    plt.pcolormesh(gabor_width)
    plt.xlabel('Time (s)')
    plt.xticks([0, 50, 100], [0, 4.46, 8.92])
    plt.ylabel('Frequency($\omega$)')
    plt.yticks([0, len(v)/2, len(v)],[] )
    plt.title('Width = %.3f' %width)
    pathname = '%.3fwidth' %width
    pathname = pathname.replace('.', 'point')
    plt.savefig(pathname)
    plt.close(fig)
#%%
#tsample = [10, 100, 1000]
tsample = [1000]
width = widthvals[-1]

for numpts in tsample:
    gabor_sample = spectrogram(
            v, fs, width, samplemod=1.5, window='gaussian', numpts=numpts)
    fig = plt.figure()
    plt.pcolormesh(gabor_sample)
    plt.xlabel('Time (s)')
    plt.xticks([0, numpts/2, numpts], [0, 4.46, 8.92])
    plt.ylabel('Frequency($\omega$)')
    plt.yticks([0, len(v)/2, len(v)],[] )
    plt.title('%d sample points' %numpts)
    plt.savefig('%d points' %numpts)
    plt.clf()
#     plt.show()

#%%
#Filter Type
width = widthvals[-1]
numpts=100
gabor_gauss = spectrogram(
        v, fs, width, samplemod=1.5, window='gaussian')
gabor_mexihat = spectrogram(
        v, fs, width, samplemod=1.5, window='mexihat')
gabor_step = spectrogram(
        v, fs, width, samplemod=1.5, window='step')

fig, axs =  plt.subplots(3, 1, sharex=True, sharey=True)
titles = ['Gaussian', 'Mexihat', 'Step']
cmap = 'viridis'
if True:
    axs[0].pcolormesh(gabor_gauss, cmap=cmap)
    axs[0].set_title(titles[0])
    axs[1].pcolormesh(gabor_mexihat, cmap=cmap)
    axs[1].set_title(titles[1])
```

```
    axs[2].pcolormesh(gabor_step, cmap=cmap)
    axs[2].set_title(titles[2])
    plt.setp(axs,
             xticks=[0, numpts/2, numpts],
             xticklabels=[0, 4.46, 8.92],
             yticklabels=[])
    plt.tight_layout()
    plt.show()
    plt.savefig('window')
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Feb  4 13:29:53 2019

@author: elimiller
"""

#from spectrogram import spectrogram
from scipy.io import wavfile
import numpy as np
from numpy import pi
import matplotlib.pyplot as plt
plt.ion()
#%%

rate, music1, = wavfile.read(
        '/Users/elimiller/Desktop/AMATH482/HW2/music1.wav')
rate2, music2 = wavfile.read(
         '/Users/elimiller/Desktop/AMATH482/HW2/music2.wav')
tr_piano = 16
tr_rec = 14
t = np.linspace(0, tr_piano, len(music1))
rate_analytical1 = len(music1) / tr_piano
rate_analytical2  = len(music2 / tr_rec)

#k = 1/(L) * np.concatenate((np.arange(0, (n/2)), np.arange(-n/2, 0)))
#ks = np.fft.fftshift(k)


def fft2flip(args):
        ReturnThis = np.abs(np.fft.fftshift(np.fft.fft(args)))
        return ReturnThis

def gaussian(unfiltered, t, location, width=.2):
    #scipy.signal.gaussian
    sigma = width
```

```
    coeff = (1/(sigma*np.sqrt(2* pi)))
    denom = 2 * sigma**2
    gauss_filter = coeff * np.exp(-(t-location)**2 / denom)
    return gauss_filter * unfiltered

def mexihat(unfiltered, t, location, width=.2):
    sigma = width
    t_instant = t - location
    psi = 2/(np.sqrt(3*sigma)*pi**(1/4)) * (
            1 - (t_instant)/(sigma)**2) * np.exp(
                    -(t_instant**2)/(2*sigma**2))
    return psi * unfiltered

def step(unfiltered, t, location, width=.2):
    bool_array = np.abs(t-location) < width/2
    stepfilt = np.array(bool_array * 1, dtype='float64')
    return stepfilt * unfiltered

def spectrogram(v, fs, width, samplemod=1, window='gaussian', numpts='100'

    window_dict = {'gaussian': gaussian, 'mexihat': mexihat, 'step': step}
    L = len(v)/fs
    t = np.arange(0, len(v))/fs
    tshift = np.linspace(0, L, numpts)
    gabor = np.zeros((len(t),len(tshift)))

    for j in range(len(tshift)):
        v_gabor = window_dict[str(window)](v, t, tshift[j], width=width)
        v_gabor_trans_shift = fft2flip(v_gabor)
        gabor[:,j] = v_gabor_trans_shift
    return gabor

def gaussian_freq(width):
    sigma = width
    coeff = (1/(sigma*np.sqrt(2* pi)))
    denom = 2 * sigma**2
    gauss_filter = coeff * np.exp(-(ks)**2 / denom)
    return gauss_filter / np.amax(gauss_filter)

#%%
    numpts = 400
width  = .1
gabor_out1 = spectrogram(
        music1, rate_analytical1, width, samplemod=1, window='mexihat', nu
#piano
cmap = 'YlGnBu'

band = 200000
```

```
#lowbnd = int(len(music1)-band/2)
lowbnd = 354000
#highbnd = int(len(music1)+band/2)
highbnd = 356250
gabor_trim1 = gabor_out1[lowbnd:highbnd, :]
n = len(music1)
L = tr_piano
k = 1/(L) * np.concatenate((np.arange(0, (n/2)), np.arange(-n/2, 0)))
ks = np.fft.fftshift(k)
kstrim1 = ks[lowbnd:highbnd]

#plt.pcolormesh(gabor_trim, norm=matplotlib.colors.LogNorm(vmin=gabor_trim
#plt.imshow(gabor_out, cmap='magma', aspect='auto', origin='lower')
if False:
    plt.subplot(211)
    plt.pcolormesh(gabor_trim1, cmap=cmap)
    note_index1 = [1835, 1300, 800]
    note_freq1 = []
    for notes in note_index1: note_freq1.append(kstrim1[notes].round(2))
    plt.title('Piano')
    plt.yticks(note_index1, note_freq1)
    plt.ylabel('Frequency (Hz)')
    plt.xticks(np.linspace(0, numpts, 5), np.linspace(0, L, 5))
    plt.xlabel('Time (s)')
    #plt.plot(tshift, 1350*np.ones_like(t))

#recorder
n = len(music2)
tr_rec = 14
L = tr_rec
width  = .01*L
rate_analytical2  = len(music2) / tr_rec

gabor_out2 = spectrogram(
        music2, rate_analytical2, width, samplemod=1, window='mexihat', nu



k = 1/(L) * np.concatenate((np.arange(0, (n/2)), np.arange(-n/2, 0)))
ks = np.fft.fftshift(k)
lowbnd = 324500
highbnd = 330000

gabor_trim2 = gabor_out2[lowbnd:highbnd, :]
kstrim2 = ks[lowbnd:highbnd]

if False:
    plt.subplot(212)
```

```
plt.title('Recorder')
plt.pcolormesh(gabor_trim2, cmap=cmap)=
note_index2 = [3800, 2100, 800]
note_freq2 = []
for notes in note_index2: note_freq2.append(kstrim2[notes].round(2))
plt.yticks(note_index2, note_freq2)
plt.ylabel('Frequency (Hz)')
plt.xticks(np.linspace(0, numpts, 5), np.linspace(0, L, 5))
plt.xlabel('Time (s)')

plt.tight_layout()
plt.savefig('part2')
```