Time-Frequency Analysis

Ron Wilson

February 7, 2020

Abstract

There are two parts in this write-up. The first part explores Gabor transforms and how they relate to the analysis of the time-frequency signature of Handel's Messiah. The second part uses Gabor transforms to generate the score of the song *Mary had a little lamb* from a small section of the song played on both a piano and a recorder.

1 Introduction and Overview

Gabor transforms allow you to analyze the time-frequency signature of audio files. In part 1, using a portion of Handel's Messiah (see Figure 1), I will explore the relationship between Gabor transforms and their parameters to understand how effectively these transforms are able to analyze time-frequency signatures. In part 2, I will use two portions of *Mary had a little lamb* (see Figures 2 and 3), one on piano and one on recorder, and reproduce the musical score by studying the time-frequency signatures using Gabor transforms.

2 Theoretical Background

In a previous project, we used the Fast Fourier Transform (FFT) and its inverse to shift between time and frequency domains. However, the main issue with the FFT is that it is only capable of retaining information about the frequencies and loses all of the data that corresponds to the time domain, whereas the inverse FFT loses the information relating to the frequencies. To analyze the time-frequency signature of an audio portion we need to be able to keep information about both the time and frequency of the signal. To that end, we will use the Gabor transform instead. A Gabor transform is a modified version of the Fourier transform using the following kernel:

$$g_{t,w}(\tau) = e^{iw\tau} g(\tau - t). \tag{1}$$

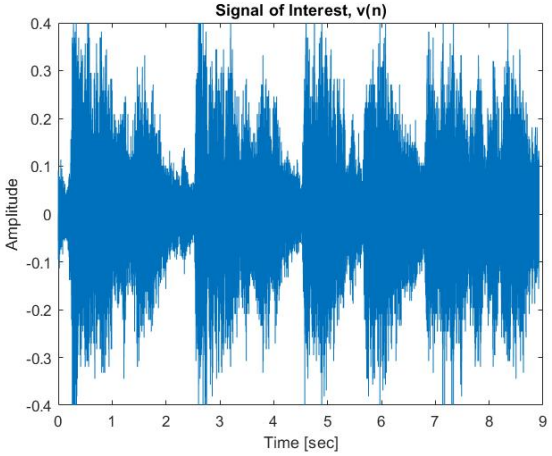

Figure 1: Signal from Handel's Messiah

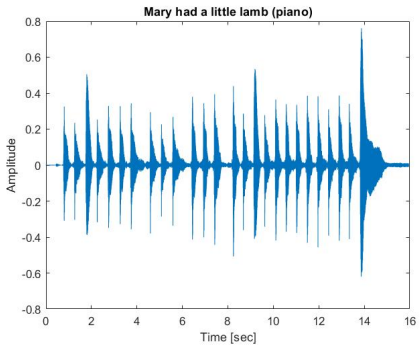

Figure 2: Mary had a little lamb piano signal

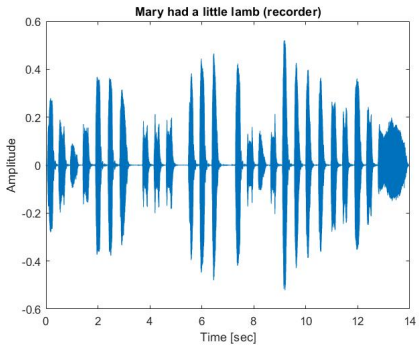

Figure 3: Mary had a little lamb recorder signal

Applying this modified kernel to the equation for the Fourier transform yields the equation for the Gabor transform:

$$\mathcal{G}[f](t,w) = \overline{f}_g(t,w) = \int_{-\infty}^{\infty} f(\tau)\overline{g}(\tau - t)e^{-iw\tau}d\tau = (f, \overline{g}_{t,w}). \tag{2}$$

Note that the bar over the function denotes its complex conjugate. The function $g(\tau - t)$, which I will refer to as the window function, is a filter that localizes the signal of a specific window of time, while the transform is shifting into the frequency domain. Thus, this transform is able to keep information about the time domain while also gaining information about the frequency. We will make use of this for the time-frequency analysis in this write-up.

We will utilize multiple different window functions, with window width w, in this project to compare the effectiveness of different Gabor transforms. The first is a Gaussian function:

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

Next, we will use a Mexican hat wavelet:

$$g(\tau - t) = [1 - (\frac{\tau - t}{w})^2]e^{-\frac{(\frac{\tau - t}{w})^2}{2}}. \tag{4}$$

Finally, we will use a step-function (Shannon) window:

$$g(\tau - t) = \begin{cases} 0 & |\tau - t| > \frac{w}{2} \\ 1 & |\tau - t| \leq \frac{w}{2} \end{cases} \tag{5}$$

3 Algorithm Implementation and Development

Here I will outline the numerical methods used in this write-up. Refer to the MATLAB code in Appendix B to see the implementation of these methods.

3.1 Part 1: Handel's Messiah

In the first part of this project, I am loading a portion of Handel's Messiah to explore with Gabor transforms. Note that I need to transpose the data to be able to apply the Gabor filters to a vector, as seen in equation 2.

3.1.1 Setting the Time and Frequency Domains

1. First, I am using the sampling rate given from loading the Handel signal to scale the time domain and the length of the signal.

2. Then, I use the length of the signal to scale the frequency domain.

(a) Note that the frequency domain is also scaled by 2π because the FFT assumes 2π periodic signals.

3. Finally, I take the FFT of the scaled frequency values to generate the frequency domain.

3.1.2 Creating Gabor Filters

1. The first step in creating the Gabor filter is to set a window width. I preset several different window widths for the sake of comparing them together to see how the size of the window effects the results. See section 4 for the discussion of different window widths.

   (a) Similarly, I am also setting multiple different values for the time-step that the filter will be applied over.

2. Next, I am creating multiple functions that I can call to apply the Gabor filter with. These functions are the Gaussian (equation 3), the Mexican hat wavelet (equation 4), and the step-function (equation 5).

3. Finally, within the loop over the time domain, I am setting each of these 3 functions as 3 separate filters which can be applied to the data at every time-step.

3.1.3 Generating Spectrograms of the Data

1. I apply each of the 3 different filters to the Handel signal using the multiple different windows and time-steps set in section 3.1.1.

2. I then take the FFT of these values and store the absolute value of the shifted FFT data as rows in a matrix for each time-step.

3. Finally, I plot this data as a spectrogram using the pcolor command for the many different choices of window width, time-step, and window function.

3.2 Part 2: Mary Had A Little Lamb

In the second part of this project, I am taking two different portions of *Mary had a little lamb*, one on piano and one on recorder, and using a Gabor transform to reproduce the musical score (viewed as a spectrogram) of the song.

3.2.1 Setting the Parameters

This is very similar to how the parameters were set for part 1. However, I need to use the `audioread` command in MATLAB to load the data and scale the corresponding sampling rate by the time of the recording. Also note that the frequency domain is scaled by the same time of the recording.

3.2.2 Creating the Gabor Filter

Again, the algorithm for creating the Gabor filter is very similar to that of part 1. This time, though, I am not using different windows or time-steps. I am using a Gaussian filter with just one value for the window width and the number of time-steps. The values used were chosen to be the more accurate values from part 1.

3.2.3 Finding the Musical Score

Finally, I apply the Gabor filter, take the FFT, and use the absolute value of the shifted FFT to create a spectrogram of the data for both the piano and recorder audio files. To view the musical score of the song, I use the axis command to zoom in on the data in the spectrogram. Note that to get a good, clean score, I filtered out the overtones generated by the center frequency of each instrument by plotting the log of the spectrogram instead.

4 Computational Results

Here I will discuss the results of the numerical methods explained in section 3.

4.1 Part 1: Handel's Messiah

1. With the number of time-steps set, I have used three different window widths (1000, 100, and 1) of a Gaussian filter to see how the size of the window effects the spectrogram. Notice that the spectrogram with a window width of 1000 (see figure 4), has frequency components which appear and vanish suddenly, while the width of 1 (see figure 5) has indistinguishable frequency components. This suggests that a large window is able to read the frequency signals very well, but loses some of its time-corresponding data (the sudden jumps in frequency values). Meanwhile, the small window is able to get the time data, but is unable to get a readable frequency signature. Note that figure 6 is somewhere in between both of these. This spectrogram has a window width of 100 and is able to retain some time and some frequency data.

2. Now, keeping the window width constant (100), I generated three spectrograms using three different numbers of time-steps (250, 100, and 10). When oversampling (using a large number of time-steps) as shown in figure 7, I get a result very similar to that of using a large window width; I get the frequency data, but have sudden jumps indicating a lack of temporal data. Using a small number of time-steps (undersampling) as in figure 8, I have a blurry spectrogram which contains the temporal data, but is missing most of the frequency components. Again, choosing a value in between the two, I get a spectrogram that has both time and frequency components (figure 6).

3. The spectrogram mentioned in both of the previous parts (figure 6) was generated using a Gaussian filter. However, I also used two other filters: Mexican hat wavelet (see figure 9) and a step-function (see figure 10). I used the same ideal number of time-steps as found in part 2 of this subsection, but I

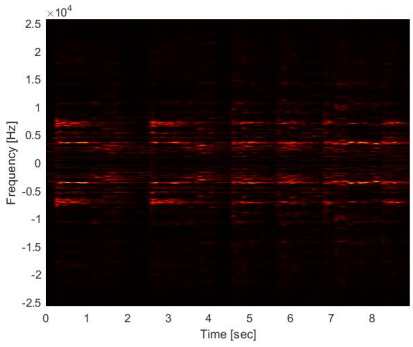

Figure 4: Gaussian (width = 1000; 100 time-steps)

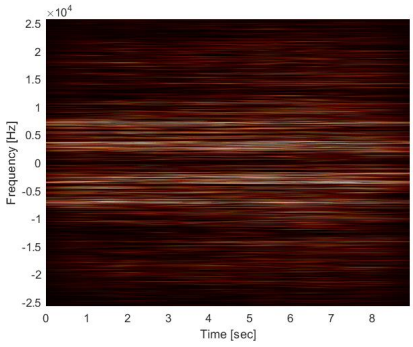

Figure 5: Gaussian (width = 1; 100 time-steps)

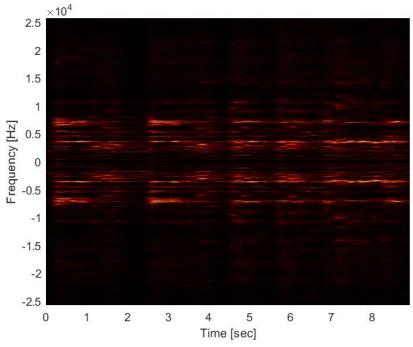

Figure 6: Gaussian (width = 100; 100 time-steps)

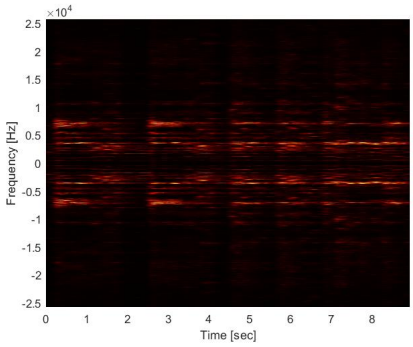

Figure 7: Gaussian (width = 100; 250 time-steps)

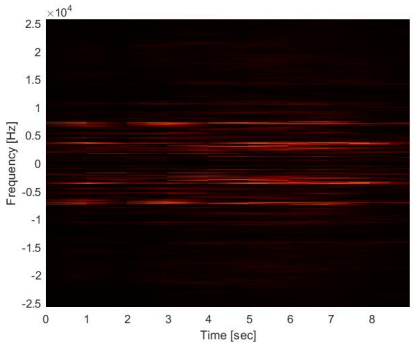

Figure 8: Gaussian (width = 100; 10 time-steps)

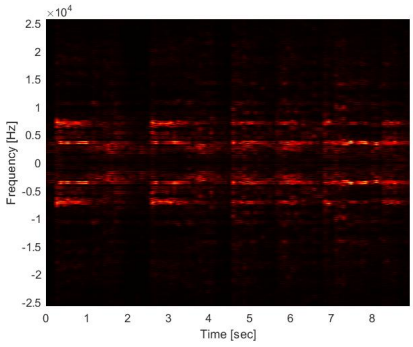

Figure 9: Mexican Hat (width = 0.01; 100 time-steps)

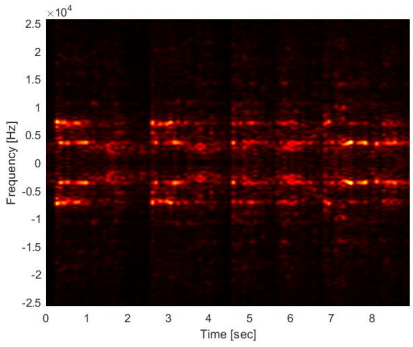

Figure 10: Shannon (width = 0.01; 100 time-steps)

needed to use a much smaller window width because these filters are different from a Gaussian and needed a smaller window to produce a clear image. The three different filters produced very similar spectrograms. The Mexican hat wavelet appears to capture slightly more frequency information than the Gaussian did, while the step-function captures even more frequency components. However, they are both missing a little temporal information as they both have those sudden jumps in frequency components. Because the Gaussian contains information of both time and frequency components when using the ideal window widths and number of time-steps found previously, I used this filter in part 2 of the project.

4.2 Part 2: Mary Had A Little Lamb

Using a Gaussian filter with a window width of 100 and 100 time-steps, I reproduced the musical score of *Mary had a little lamb* on both the piano (see figure 11) and recorder (see figure 12). Notice that the score on the piano has a lot more noise than that of the recorder. This is because the piano generates a lot more overtones than the recorder does (the timbre of the two instruments are very different). This is what makes the sound of the piano a lot fuller than the sound of the recorder.

5 Summary and Conclusions

By exploring the relationships between Gabor transforms and their parameters I found that taking a medium window width and sampling size produced the clearer image, and I used that ideal Gabor filter to reproduce the musical score of *Mary had a little lamb* on two different instruments, being able to easily see the difference between the two instruments.

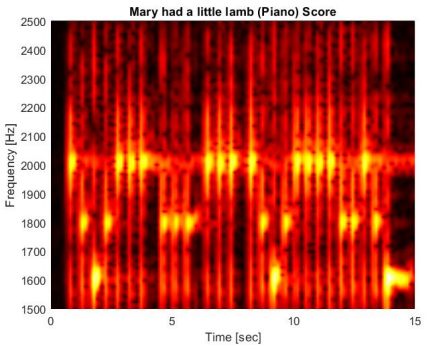

Figure 11: *Mary had a little lamb* score on piano

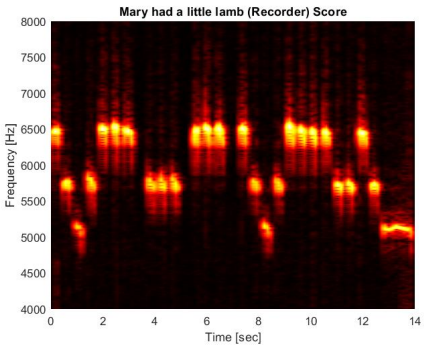

Figure 12: *Mary had a little lamb* score on recorder

There are so many choices that can be made when using a Gabor transform that they work very well in different scenarios of time-frequency analysis. From the results of this write-up, it is clear that experimenting with the parameters of a Gabor transform can really help to find a good, clean image.

Appendix A MATLAB Functions

The following is a list of the important MATLAB functions that were used:

- `plot(X,Y)` creates a 2-D line plot of the data in `Y` versus the corresponding values in `X`.
- `player = audioplayer(Y,Fs)` creates an audioplayer object for signal `Y`, using sample rate `Fs`. The function returns the audio player object, player.
- `playblocking(playerObj)` plays the audio associated with audioplayer object playerObj from beginning to end. playblocking does not return control until playback completes.
- `Y = fftshift(X)` rearranges a Fourier transform `X` by shifting the zero-frequency component to the center of the array.
- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `X = zeros(sz1,...,szN)` returns an `sz1-by-...-by-szN` array of zeros where `sz1,...,szN` indicate the size of each dimension.
- `Y = fft(X)` computes the discrete Fourier transform (DFT) of `X` using a fast Fourier transform (FFT) algorithm.
- `pcolor(X,Y,C)` creates a pseudocolor plot using the values in matrix C. `X` and `Y` 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.
- `[y,Fs] = audioread(filename)` reads data from the file named filename, and returns sampled data, `y`, and a sample rate for that data, `Fs`.

Appendix B MATLAB Code

The MATLAB code used to generate the results of this write-up (`Time-Frequency_Analysis.m`) can be found at the following GitHub repository: `https://github.com/ronwilson016/Data-Science-Projects/tree/master/Time-Frequency%20Analysis/MATLAB%20Code`