AMATH 582 Homework 2: Time-Frequency Analysis of an Unlikely Couple- G.F. Handel and Mary the Shepherdess

Gavin M. Chase

February 7, 2020

Abstract

This report explores time-frequency analysis for a segment of G.F. Handel's *Messiah* and several renditions of *Mary had a Little Lamb* using the concept of short-time Fourier transforms (STFT) or the Gábor Transform with Gaussian, Mexican hat, and step-function windows. Resultant plots of this analysis were then used to determine the score for *Mary had a Little Lamb*.

1 Introduction and Overview

Fourier transforms, as a stand alone method, are exceptional at determining frequency content of a signal, but are unable to describe where those frequencies occur in time. In 1946, Gábor built upon the Fourier transform and developed a method that extracts frequency content for various windowed moments in time using filters of various widths to integrate time analysis and frequency analysis into a single coherent construct[2]. The Gábor transform accepts multiple filter types to take advantage of their unique filtering properties when analyzing data[4].

Music by definition is a series of frequencies ordered in time. Thus, it serves as an excellent domain to demonstrate the utility of Gábor transforms. I applied multiple filter types to a segment of Handel's *Messiah* and varied the widow widths and sampling rates to explore each filter's properties and how the Heisenberg uncertainty principle limits time-frequency analysis. I also used Gábor filtering on two renditions of *Mary had a Little Lamb* played on piano and recorder, respectively.

2 Theoretical Background

2.1 The Gábor Transform and Filter Types

The Fourier series, FFT, and Gaussian filters are discussed at length in the Homework 1 report and will not be discussed here, even though they serve as the foundation for time-frequency analysis[1].

The Gábor Transform is defined as

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

where the function $g(\tau - t)$ acts as a sliding time filter applied to the function $f(\tau)$ to localize the signal at a specific window in time[4]. The difference between this transform and the Fourier transform is shown in Figure 1. Fourier transforms are limited in that they show frequency content across an entire data set. Adding a sliding filter in time increases the usefulness of the transform by localizing the results in time.

For computational purposes, a discretized version of the Gábor transform is used. In this case $g_{t,w}$ becomes

$$g_{m,n}(t) = e^{i2\pi m\omega_0 t}g(t - nt_0) \tag{2}$$

where m and n are integers and $w_0, t_0 > 0$ are constants. It is worth noting that to get sufficient localization in time and frequency the Gábor windows must overlap. This is referred to as oversampling and occurs when $0 < w_0, t_0 < 1$.

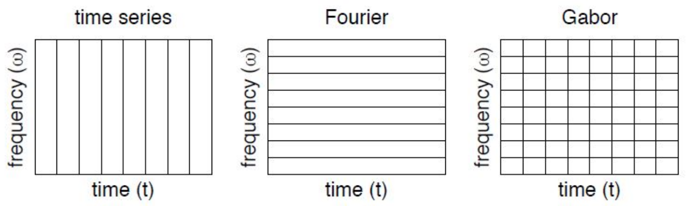


Figure 1: A cartoon representation of how Gábor transforms integrate Fourier and time domains into a more useful construct. Note that the size and shapes of the boxes in the Gábor frame are limited by the Heisenberg uncertainty principle[4].

The biggest drawback to the Gábor transform is its limitation by the Heisenberg uncertainty principle. Narrow filters produce excellent time resolution at the expense of capturing frequency content with a wavelength longer than the filter is wide. Conversely, wide filters catch more frequency content, but at the cost of not knowing precisely when the signal occurred in time. This rule exists regardless of filter type.

The Mexican hat wavelet is a second moment of a Gaussian filter and is defined as

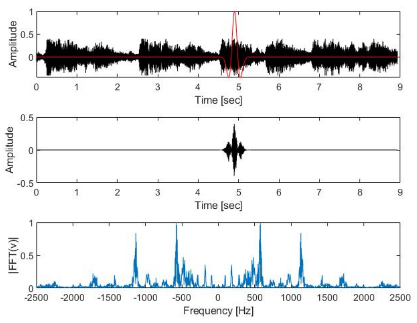
$$\psi(t) = (1 - \tau^2(t - t_0)^2)e^{-\frac{\tau^2(t-t_0)^2}{2}} \tag{3}$$

where τ determines the width of the filter and t_0 shifts the filter in time. An example of what this filter looks like, and how frequency data looks when this wavelet is applied can be found in Figure 2a.

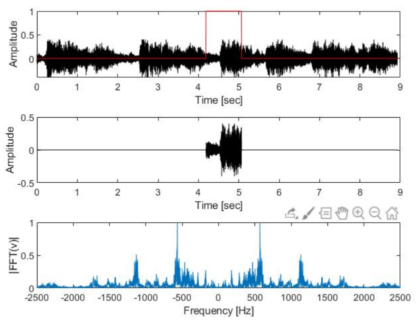
The step-function, which is also known as a Shannon filter, is a square wave set to a single amplitude across the plateau of the wave and is zero elsewhere. It is defined mathematically as

$$\psi(t) = \begin{cases} 1 & \text{if t} \in \mathcal{X} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where $\mathcal{X}$ is the width of time you want to isolate for filtering. Sliding $\mathcal{X}$ along the time domain sequentially filters the entire signal set. Because of the step-function's sharp edges, wide filters retain sharp cutoffs in time for a each frequency. The filter's shape and effects are shown in Figure 2b.



(a) Mexican hat filter.



(b) Step-function: note sharp edges in the middle frame.

Figure 2: A snapshot of each filter applied to data used in this study. The top frame shows the shape of the filter, the middle frame shows what remains of the signal content after applying the filter, and the bottom frame shows the resultant frequency content at the filtered moment in time.

2.2 Characteristics of the Music (Data)

Because music is simply a set of frequencies ordered in time it can be analyzed in the same manner as other signals using a spectrogram. The musical notes played by performers correspond to specific frequencies of

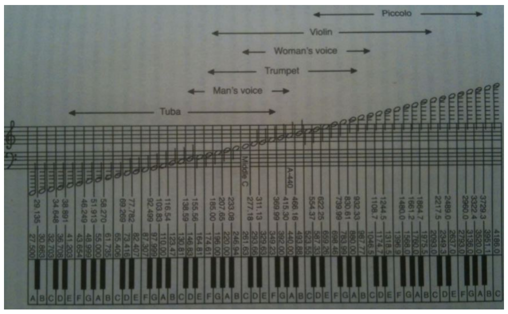


Figure 3: Musical notes, their associate frequencies and several instruments ranges.

sound waves shown in Figure 3. Thus, when establishing Fourier modes for FFT calculations the following relationship is useful[3]

$$\omega = \frac{2\pi}{T} = 2\pi f \tag{5}$$

where ω is the angular frequency or wavenumber, T is the period, and f is the frequency measured in Hertz [Hz].

Simple pieces of music, such as *Mary had a Little Lamb*, are easy to evaluate, especially when played by only a single instrument. This kind of analysis is useful for understanding a particular instrument's qualities such as its frequency range and timbre. More complicated pieces such as Handel's *Messiah*, which has a choir, violins, trumpets, oboes, a viola, and a timpani, produce cluttered spectrograms, but can be filtered for a particular instrument's input if the instrument's frequency range is already known.

3 Algorithm Implementation and Development

No matter what music sample is being analyzed there are three generic steps that are always followed. The first step, shown in Algorithm 1 and Appendix B, Listing 1, includes initial modifications of the music data and establishing the frequency and time domains.

Algorithm 1: Load Music File and Set Temporal and Frequency Domains

```
    Create vector $y$ as the audioread music file filename.wav
    Set the time domain $L$ to length of song in seconds
    Set Fourier modes $n$ to length of $y$
    Find the sampling rate $Fs$ as $\frac{n}{L}$
    Discretize the time domain $t$ from 0 to $L$ with $n$ points ensuring to account for boundary conditions
    If $n$ is even, set frequency domain $k$ on a $2\pi$ periodic domain using $k = \frac{2\pi}{L} * [0 : (\frac{n}{2} - 1) - \frac{n}{2} : -1]$
    If $n$ is odd, set frequency domain $k$ on a $2\pi$ periodic domain using $k = \frac{2\pi}{L} * [0 : (\frac{n}{2}) - \frac{n}{2} : -1]$
    Set new vector $ks$ as the FFT shifted $k$
    Plot vector $y$ with respect to time
    Optional: play music with audioplayer
```

The second step is to create a filter and apply it to the data. Because each filter is slightly different, the creations of the Gaussian, Mexican hat, and step-function filters are described separately in Algorithms 2, 3, and 4. The algorithms are designed to apply a single filter at a single predetermined sampling rate tied to the step size in `tslide` and at a single width τ. Thus, they need to be rerun for every desired adjustment of those two variables. The actual application of each filter is generically the same regardless of shape. Thus, the process is only described in great detail in Algorithm 2. The final step is to produce the spectrogram. MATLAB has several colormaps available to display spectrograms, and these can be tailored in size to focus on a particular band of frequencies or time.

Algorithm 2: Creation and Application of the Gaussian Window

```
Create an empty matrix $yspec$
Set width value $\tau$
Create a vector tslide as a set of evenly spaced times between 0 and $L$
for $j = 1$ :length of tslide do
    Create filter centered at time $j$ using $g = e^{\tau(t-j)^2}$
    **START APPLICATION PROCESS GENERIC TO ALL FILTER TYPES**
    Apply filter at position $j$ to music vector $y$
    Perform FFT on filtered vector $y$
    Add the absolute value of the FFT shifted and filtered vector $y$ as a row in $yspec$
    plot filter $g$ and original music vector $y$ with respect to time $t$
    plot filtered vector $y$ with respect to time $t$
    plot the normalized and transformed vector $y$ with respect to frequency $\frac{ks}{2\pi}$
end for
Select colormap style
Set band of frequencies you are interested in displaying
Use pcolor to plot the $yspec$ matrix produced at the end of filtering with time on the $x$ axis and frequencies $\frac{ks}{2\pi}$ on the $y$ axis
**END APPLICATION PROCESS GENERIC TO ALL FILTER TYPES**
```

Algorithm 3: Creation and Application of the Mexican Hat Wavelet

```
Create an empty matrix $yspec$
Set width value $\tau$
Create a vector tslide as a set of evenly spaced times between 0 and $L$
for $j = 1$ :length of tslide do
    Create filter centered at time $j$ based on Equation 3
    **Follow lines 6-18 of Algorithm 2 for application process**
end for
```

Algorithm 4: Creation and Application of the Step-function Window

```
Create an empty matrix $yspec$
Set width value $\tau$
Create a vector of zeros called step equal in length to the length of $n$
Create a vector of ones that is $2\tau + 1$ long
Create a vector tslide as a set of evenly spaced Fourier modes between $\tau$ and the max value of $n$
for $j = 1$ :length of tslide do
    Create filter centered at time $j$ based on Equation 4
    **Follow lines 6-18 of Algorithm 2 for application process**
end for
```

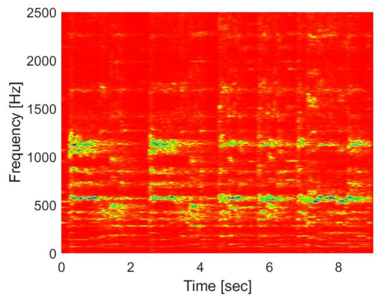
4 Computational Results

4.1 *Messiah*

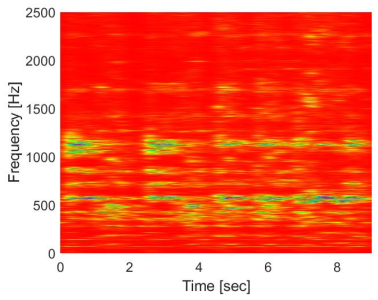
I applied Gaussian, step-function, and Mexican hat windows to Handel's *Messiah*, and the results are shown in Figures 4, 6, and 7 respectively. In each of these trials, I applied two filters: one narrow and one wide. The narrow filters clearly produce greater time resolution, but the wider bands of frequencies indicate a loss of frequency resolution. Conversely, each of the wide filter spectrograms produce excellent frequency resolution, even if the time localization stretched over almost a full second. These effects are more obvious when viewing a narrower segment of time such as that shown in Figure 8.

The spectrograms indicate significant frequency content between 250-500 Hz and 1kHz-1.1kHz. These ranges correspond to the male and female voices in the choir as well as the trumpets and violins accompanying the choir.

Figure 5 explores the importance of oversampling. This figure is the exact same as the trial used to plot Figure 4 but with much fewer time translations. There is a clear loss of signal content due to the lack of overlapping Gábor windows. The signal is not completely lost, but there are still obvious gaps.

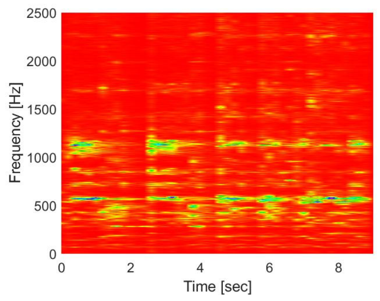


(a) Spectrogram of *Messiah* with $\tau = 250$.

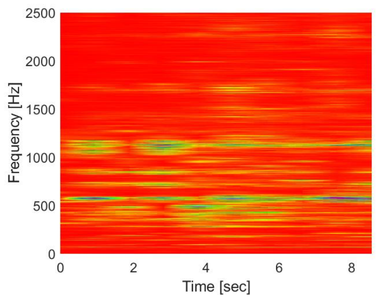


(b) Spectrogram of *Messiah* with $\tau = 20$.

Figure 4: Handel's *Messiah* filtered using a Gaussian window at 33 translations per second



(a) Spectrogram of *Messiah* with $\tau = 250$.

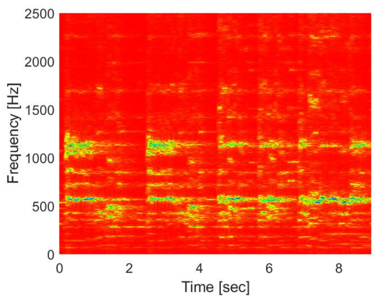


(b) Spectrogram of *Messiah* with $\tau = 20$.

Figure 5: Handel's *Messiah* undersampled at 5 translations per second and 1 translation per second respectively with a Gaussian window. Much of the signal remains, but it is an incomplete representation of the music.

4.2 *Mary Had a Little Lamb*

Figures 9 and 10 are spectrograms of recordings of *Mary Had a Little Lamb* played on piano and recorder respectively. The frequencies are higher on the recorder due to the shrill nature of the instrument, but each instrument follows the same pattern when playing the song. I used Noteflight's composing tool to translate the frequencies from the piano spectrogram into an actual score found in Figure 9b. It is also worth noting


(a) Spectrogram of *Messiah* with $\tau = 500$.

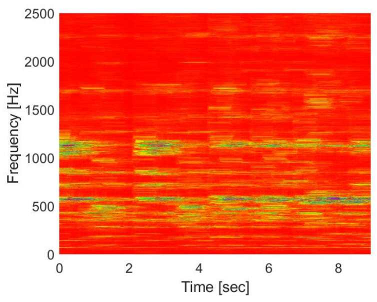
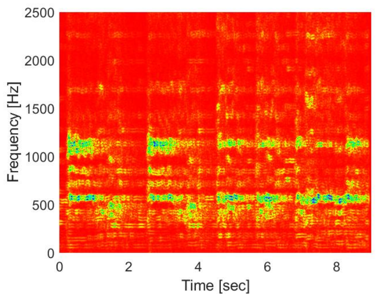

(b) Spectrogram of *Messiah* with $\tau = 2500$.

Figure 6: Handel's *Messiah* filtered using a step-function at 32 translations per second.


(a) Spectrogram of *Messiah* with $\tau = 60$.

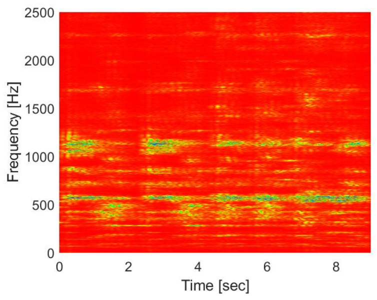
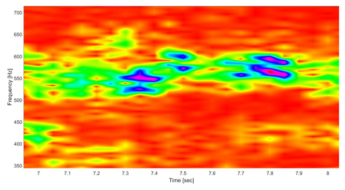

(b) Spectrogram of *Messiah* with $\tau = 10$.

Figure 7: Handel's *Messiah* filtered using a Mexican hat wavelet at 40 translations per second.

the faint overtone signature at higher frequencies on the piano spectrogram. These overtones can also be found on the recorder spectrogram but are much fainter. They correspond to $2\omega_0$, $3\omega_0$, $\ldots$ and so forth where ω_0 is the center frequency played.

5 Summary and Conclusions

The purpose of this exercise was to explore the applications and limitations of time-frequency analysis as well as the properties of a subset of its tools. The reproduction of the score for *Mary had a Little Lamb* is the most obvious demonstration of time-frequency analysis's utility. The importance of oversampling and varying the window widths was also clearly evident. There is, however, room to improve the methods used in this report. First, much of the code is redundant and could be made more efficient if repeated actions are turned into functions. It should also be possible to set more than just a single desired width and sampling


(a) Spectrogram of *Messiah* with $\tau = 60$.

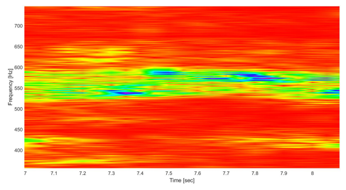
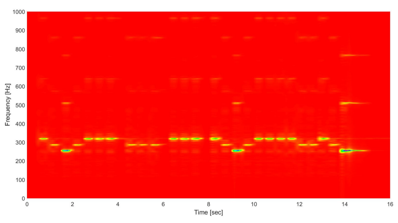
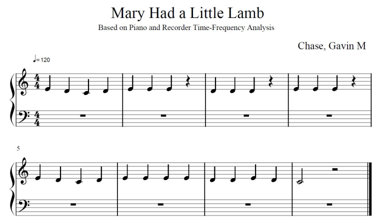

(b) Spectrogram of *Messiah* with $\tau = 10$.

Figure 8: A closer view of the difference between narrow and wide filters. Figure 8a exhibits great time resolution down to tenths of a second but at the cost of certainty in frequency. Conversely, Figure 8b shows much finer frequency resolution at a cost of exactness in time.



(a) Spectrogram of *Mary had a Little Lamb*.



(b) The score of *Mary had a Little Lamb*.

Figure 9: *Mary had a Little Lamb* played on piano. Note the faint overtone signatures between 500-600Hz and 750Hz-1kHz in Figure 9a.

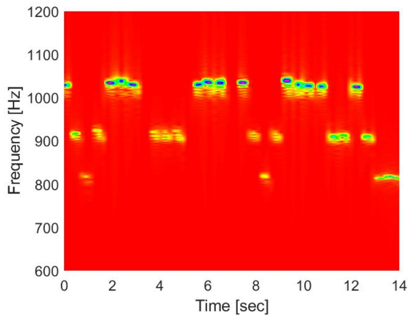


Figure 10: Spectrogram of *Mary had a Little Lamb* played on a recorder.

rate before running the calculations. This would reduce time spent manipulating the scripts if there is a range of desired calculations to be run.

References

[1] Gavin M. Chase. "AMATH 582 Homework 1: FFT Applications Including Filtering by Averaging and Gaussian Filtering". In: *AMATH 582 Portfolio* (2020).

[2] Dennis Gabor. "Theory of communication. Part 1: The analysis of information". In: *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering* 93.26 (1946), pp. 429–441.

[3] David Halliday, Robert Resnick, and Jearl Walker. *Fundamentals of physics*. John Wiley & Sons, 2013.

[4] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

Appendix A MATLAB Functions

What follows is a list of MATLAB functions not covered in the Homework 1 Report[1] as they appear in Appendix B. Much of the text below with additional examples can be found in the Mathworks Documentation Library.

- `audioplayer(y,Fs)` plays signal y, using sample rate Fs.
- `playblocking(x)` plays audio from start to end and prevents script from continuing to run until audio is complete.
- `pcolor(x)` creates a pseudocolor plot using the values in matrix x. A pseudocolor plot displays matrix data as an array of colored cells (known as faces). MATLAB creates this plot as a flat surface in the

$x-y$ plane. The surface is defined by a grid of x- and y-coordinates that correspond to the corners (or vertices) of the faces. The grid covers the region $X = 1 : n$ and $Y = 1 : m$, where `[m,n] = size(x)`.

- `colormap(x)` sets the colormap for the current figure to the colormap specified by `x`. Colormap options can be found at `https://www.mathworks.com/help/matlab/ref/colormap.html`.
- `[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

This appendix includes the MATLAB code used to produce the results in this report. The code can also be found at `https://github.com/gchase15/AMATH-582`

```matlab
%% The Setup

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

v2=y'/2; %y is the data that comes when loading handel

figure(1)
plot((1:length(v2))/Fs,v2); %Fs is the sampling rate that comes when loading handel
xlabel('Time [sec]');
ylabel('Amplitude');
title('Signal of Interest, v(n)');

 p8 = audioplayer(v2,Fs);
 playblocking(p8);

%% Gabor Transform with Gaussian Window
figure(4)
vgt_spec=[];
tslide=0:.15:9;
tau=200;
for j=1:length(tslide)
    g=exp(-tau*(t-tslide(j)).^2); % Filter
    vg=g.*v2;
    vgt=fft(vg);
    vgt_spec=[vgt_spec;
    abs(fftshift(vgt))];
    subplot(3,1,1), plot(t,v2,'k',t,g,'r')
    xlabel('Time [sec]');
    ylabel('Amplitude');
    subplot(3,1,2), plot(t,vg,'k')
    xlabel('Time [sec]');
    ylabel('Amplitude');
    subplot(3,1,3), plot(ks/(2*pi),abs(fftshift(vgt))/max(abs(vgt)))
    xlabel('Frequency [Hz]');
    ylabel('|FFT(v)|')
    axis([-2500 2500 0 1])
    drawnow
    pause(0.02)
end

figure(5)
pcolor(tslide,ks/(2*pi),vgt_spec.'),
shading interp
set(gca,'Ylim',[-0 2500],'Fontsize',[14])
colormap(hsv)
xlabel('Time [sec]');
ylabel('Frequency [Hz]');
```

Listing 1: MATLAB code covering initial setup for analyzing `Messiah`, and Gàbor transform using a Gaussian window.

```
%% STEP FILTERS

figure(8)
width=2500;
step=zeros(1,length(t));
mask=ones(1,2*width+1);
vstept_spec=[];
tslide=(width+1):250:(length(step)-width);
for j=1:length(tslide)
    step=zeros(1,length(t));
    step(tslide(j)-width:1:tslide(j)+width)=mask;
    vstep=step.*v2;
    vstept=fft(vstep);
    vstept_spec=[vstept_spec;
    abs(fftshift(vstept))];
    subplot(3,1,1), plot(t,v2,'k',t,step,'r')
    xlabel('Time [sec]');
    ylabel('Amplitude');
    subplot(3,1,2), plot(t,vstep,'k')
    xlabel('Time [sec]');
    ylabel('Amplitude');
    subplot(3,1,3), plot(ks/(2*pi),abs(fftshift(vstept))/max(abs(vstept)))
    xlabel('Frequency [Hz]');
    ylabel('|FFT(v)|')
    axis([-2500 2500 0 1])
    drawnow
end

tslide=linspace(0,L,length(tslide));
figure(9)
pcolor(tslide,ks/(2*pi),vstept_spec.'),
shading interp
set(gca,'Ylim',[0 2500],'Fontsize',[14])
colormap(hsv)
xlabel('Time [sec]');
ylabel('Frequency [Hz]');
```

Listing 2: MATLAB code covering the Gàbor transform using a step-function window.

```
%% Mexican Hat Filter

figure(11)
width=10;
vmhwt_spec=[];
tslide=0:0.05:9;
for j=1:length(tslide)
    mhw=(1-((t-tslide(j))*width).^2).*exp(-(width^2*(t-tslide(j)).^2)/2);
    vmhw=mhw.*v2;
    vmhwt=fft(vmhw);
    vmhwt_spec=[vmhwt_spec;
    abs(fftshift(vmhwt))];
    subplot(3,1,1), plot(t,v2,'k',t,mhw,'r')
    xlabel('Time [sec]');
    ylabel('Amplitude');
    subplot(3,1,2), plot(t,vmhw,'k')
    xlabel('Time [sec]');
    ylabel('Amplitude');
    subplot(3,1,3), plot(ks/(2*pi),abs(fftshift(vmhwt))/max(abs(vmhwt)))
    xlabel('Frequency [Hz]');
    ylabel('|FFT(v)|')
    axis([-2500 2500 0 1])
    drawnow
end

figure(12)
pcolor(tslide,ks/(2*pi),vmhwt_spec.'),
shading interp
set(gca,'Ylim',[0 2500],'Fontsize',[14])
colormap(hsv)
xlabel('Time [sec]');
ylabel('Frequency [Hz]');
```

Listing 3: MATLAB code covering the Gàbor transform using a Mexican hat window.

```
%% Piano Setup
tr_piano=16; %record time in seconds
yp=audioread('music1.wav');
yp=yp';
Fsp=length(yp)/tr_piano;
Lp=tr_piano;
np=length(yp);
tp2=linspace(0,Lp,np+1);
tp=tp2(1:np);
kp=(2*pi/Lp)*[0:np/2-1 -np/2:-1];
kps=fftshift(kp);
 figure(13)
plot((1:length(yp))/Fsp,yp);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Mary had a little lamb (piano)');
drawnow
   p8 = audioplayer(yp,Fsp);
   playblocking(p8);
%% Piano STEP FILTERS
figure(16)
width=7014;
step=zeros(1,length(tp));
mask=ones(1,2*width+1);
ypstept_spec=[];
tslide=(width+1):7000:(length(step)-width);
for j=1:length(tslide)
    step=zeros(1,length(tp));
    step(tslide(j)-width:1:tslide(j)+width)=mask;
    ypstep=step.*yp;
    ypstept=fft(ypstep);
    ypstept_spec=[ypstept_spec;
    abs(fftshift(ypstept))];
    subplot(3,1,1), plot(tp,yp,'k',tp,step,'r')
    xlabel('Time [sec]');
    ylabel('Amplitude');
    subplot(3,1,2), plot(tp,ypstep,'k')
    xlabel('Time [sec]');
    ylabel('Amplitude');
    subplot(3,1,3), plot(kps/(2*pi),abs(fftshift(ypstept))/max(abs(ypstept)))
    xlabel('Frequency [Hz]');
    ylabel('|FFT(v)|')
    axis([-1200 1200 0 1])
    drawnow
end
tslidep=linspace(0,Lp,length(tslide));
figure(17)
pcolor(tslidep,kps/(2*pi),ypstept_spec.'),
shading interp
set(gca,'Ylim',[0 1000],'Fontsize',[14])
colormap(hsv)
xlabel('Time [sec]');
ylabel('Frequency [Hz]');
```

Listing 4: MATLAB code covering initial setup for *Mary had a Little Lamb* on the piano and the Gàbor transform using a step-function window.

```
%% Recorder
tr_rec=14; % record time in seconds
yrec=audioread('music2.wav');
yrec=yrec';
Fsrec=length(yrec)/tr_rec;
Lrec=tr_rec;
nrec=length(yrec);
trec2=linspace(0,Lrec,nrec+1);
trec=trec2(1:nrec);
krec=(2*pi/Lrec)*[0:nrec/2-1 -nrec/2:-1];
ksrec=fftshift(krec);
figure(18)
plot((1:length(yrec))/Fsrec,yrec);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Mary had a little lamb (recorder)');
drawnow
 p8 = audioplayer(yrec,Fsrec);
 playblocking(p8);
 %% recorder step function
figure(21)
width=7144;
step=zeros(1,length(trec));
mask=ones(1,2*width+1);
yrecstept_spec=[];
tslide=(width+1):8000:(length(step)-width);
for j=1:length(tslide)
    step=zeros(1,length(trec));
    step(tslide(j)-width:1:tslide(j)+width)=mask;
    yrecstep=step.*yrec;
    yrecstept=fft(yrecstep);
    yrecstept_spec=[yrecstept_spec;
    abs(fftshift(yrecstept))];
    subplot(3,1,1), plot(trec,yrec,'k',trec,step,'r')
    xlabel('Time [sec]');
    ylabel('Amplitude');
    subplot(3,1,2), plot(trec,yrecstep,'k')
    xlabel('Time [sec]');
    ylabel('Amplitude');
    subplot(3,1,3), plot(ksrec/(2*pi),abs(fftshift(yrecstept))/max(abs(yrecstept)))
    xlabel('Frequency [Hz]');
    ylabel('|FFT(v)|')
    axis([-1400 1400 0 1])
    drawnow
end
tsliderec=linspace(0,Lrec,length(tslide));
figure(22)
pcolor(tsliderec,ksrec/(2*pi),yrecstept_spec.'),
shading interp
set(gca,'Ylim',[600 2400],'Fontsize',[14])
colormap(hsv)
xlabel('Time [sec]');
ylabel('Frequency [Hz]');
```

Listing 5: MATLAB code covering initial setup for *Mary had a Little Lamb* on the recorder and the Gàbor transform using a step-function window.