# A Background on Automatic Music Transcription

## Introduction

The end-goal here is the build a usable automatic music transcriber – a piece of software that is able to notate sheet music for a piece of audio. Such a piece of software would be a very valuable tool for musicians, as it would allow one to quickly transcribe an improvised jazz solo, for example.

## High-Level Overview

The processes of transcribing a piece of music is a complicated one, especially one to implement in software. Such a task requires that we extract the following information from a single time-domain signal:

- Tempo
- Key
- Time signature
- Pitch
- Rhythm

Extracting this information is already hard enough. But after we know all of this, we will also need a piece of software that will take this information, and notate it on sheet music (which is another fairly complicated task).

In order to transcribe a note, the following must be done:

1. Recognize that a note is being played
    a. The beginning of a note is called an *onset*, and the processing of recognizing an onset is called *onset detection*
2. Identify the pitch of that note
    a. The pitch of a note is its fundamental frequency $f_0$, and the process of recognizing $f_0$ is called *pitch detection*

This doesn't even include factors like rhythm or tempo. We will focus on these problems for now.

## MIDI

MIDI (musical instrument digital interface) is a standard that can be used to describe audio. It can be visualized as such, where the x-axis is time, and the y axis is frequency:

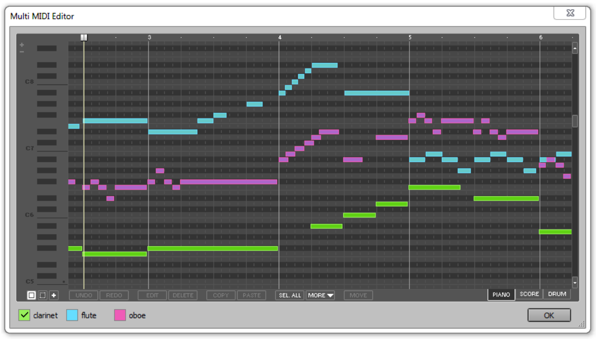


Figure 1 – MIDI Data Visualized in a Digital Audio Workstation

MIDI is important, since our problem of transcribing an audio signal can be reframed as a problem of converting an audio signal into MIDI data (which is slightly easier to solve).

## Pitch Detection

Pitch detection consists of two steps: computing the Discrete Fourier Transform on a time-series dataset, then running a pitch detection routine to determine the fundamental frequency. The Discrete Fourier Transform is documented very well on the internet. As such, we will be focusing on the pitch detection routine.

The pitch detection routine that we will be using is called the *Harmonic Product Spectrum*. It is known that instruments do not give off pure sine waves, but a sum of sine waves. This sum of sine waves consists of sines with frequencies that are scalar multiples of each other. These frequencies are called *harmonics*. With this in mind, a signal coming from any instrument can be expressed as such:

$$x(t) = \sum_{n=1}^{\infty} C_n \sin\big((\omega n)t\big)$$

In frequency domain, $X(\omega)$ might look like this:

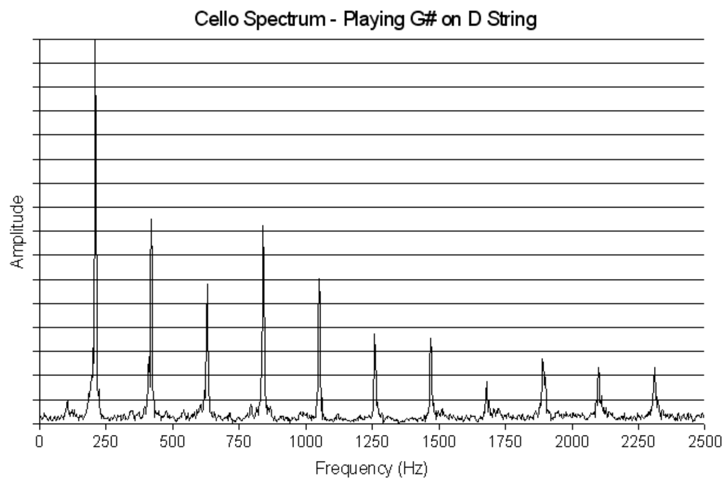


**Figure 2 – Harmonic Spectrum of a Cello**

Now, the goal of the harmonic product spectrum is to identify the lowest frequency $f_0$ that is present in the frequency spectrum. The way that it does this is by computing the following product

$$Y(\omega) = \prod_{n=1}^{N} |X(\omega n)|$$

where $N$ is the number of harmonics being considered.

This product can be visualized as such:

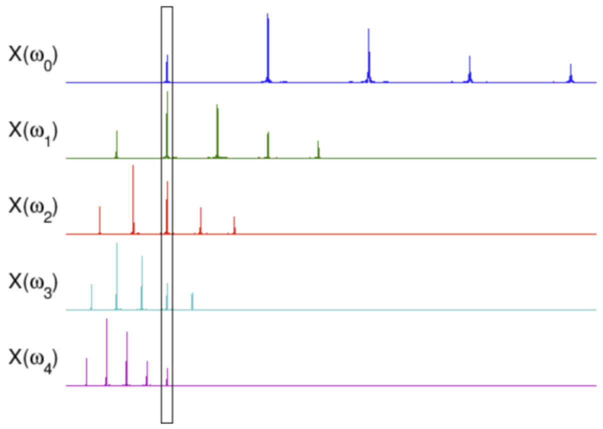


**Figure 3 – Harmonic Product Spectrum Visualized**

As shown in figure 3, the harmonic product spectrum will accumulate the harmonics at the fundamental frequency. Thus, the fundamental frequency can be found by taking $\max(Y(\omega))$.

## Onset Detection

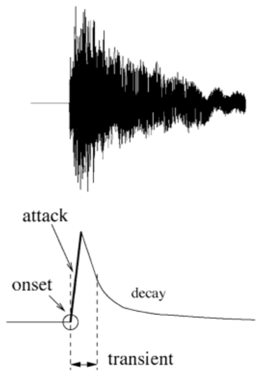
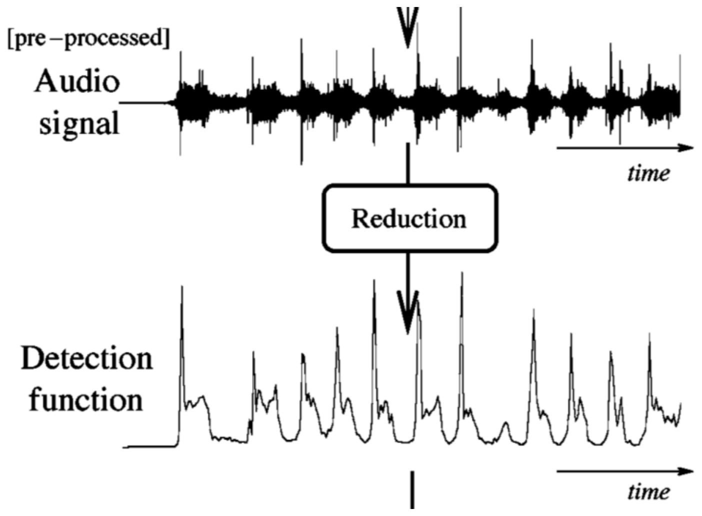


**Figure 4 – An example of a musical onset**

Figure 1 shows what an actual onset might look like in the top image, and what an ideal onset looks like on the bottom image. The goal here is to be able to detect one of these.

One should note that the reliability of onset detection varies by instrument. The onset of an instrument that has a very sharp attack such as a snare drum will be much easier to detect than one with a wider attack, such as a bowed violin.

A broader picture of what we are trying to achieve is shown below:



What's happening here, is we are trying to create a signal in time domain that has transients corresponding to the onsets in the original signal.

Our application will use a frequency domain technique of doing this, since it is simple, and required for pitch detection anyways.

If you are not already familiar with the Fourier Transform, the Discrete Fourier Transform, and the Short-Time Fourier Transform, go do it now please.

First, we consider the Short-Time Fourier Transform:

$$X_m[k] = \sum_{n=1}^{N-1} x[n + mh]e^{\frac{-j2\pi kn}{N}} \cdot w[n]$$

Where $n$ represents time-domain bins $k$ represents frequency-domain bins, $m$ is number of windows that have been taken, $w(n)$ is a window function, and $h$ is the hop size.

In frequency domain, onset transients appear across the entire frequency spectrum, although they are more noticeable at higher frequencies. As such, we can perform a weighted energy summation, with higher frequencies being given more weight than lower frequencies. Such a summation could be used to detect transients in time domain:

$$\tilde{E}[m] = \frac{1}{N}\sum_{k=1}^{N-1} W[k] \cdot |X_m[k]|^2$$

where $\tilde{E}[m]$ is a representation of the $m$th window's energy in frequency domain, and $W[k]$ is a frequency dependent weighting function. Common weighting functions include $W[k] = k^2$ and $W[k] = |k|$.

Now that we have our detection function, all that we need to do is to be able to detect the peaks of this function. Such a process is called *peak picking*.

## Peak-Picking

Peak picking can be divided into 3 steps: post-processing, thresholding, and a final decision process.

### Post-Processing

This is an optional step that can increase the robustness of peak picking. The purpose of this is to reduce the effects of “noise” by smoothing the signal. A common way of doing this include computing a moving average on the signal, which acts as a low-pass filter. This is probably what we will implement in our program.

### Thresholding

Even after post-processing, there will be a number of local maxima that do not correspond to an onset. Therefore, we need a threshold to differentiate between the two. There are two types of thresholding: fixed thresholding, and adaptive thresholding. We will likely need to implement adaptive thresholding, since music can have a wide range of dynamics that make fixed thresholding insufficient.

A simple adaptive threshold $\delta[n]$ can be defined as a version of the detection function, which can be achieved via a low-pass filter:

$$\delta[n] = \delta_0 + lowpass(d[n])$$

where $d[n]$ is the detection function, and $\delta_0$ is a predetermined fixed threshold.

### Final Decision Process

The final decision process consists of a local maxima detection routine, which is extensively documented elsewhere.

## Sources

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.332.989&rep=rep1&type=pdf

http://musicweb.ucsd.edu/~trsmyth/analysis/Harmonic_Product_Spectrum.html