Beat Detection for Automated Music Transcription:

An exploration of Onset Detection Algorithms.

Adam Hess,
M.S. Electrical Engineering
B.S. Electrical Engineering
B.A. Music

Advisor: Dr. Mark Fowler

May 08, 2011

Submitted in partial fulfillment of the requirements of Masters of Science in Electrical Engineering
Thomas J. Watson School of Engineering and Applied Science
State University of New York at Binghamton

Table of Contents

Table of Figures

1. Abstract:

This project attempts to answer part of the question: Can music be reconstructed into sheet music from a recording through signal processing? This is a multifaceted problem that entails instrument identification, instrument isolation, pitch detection, dynamics etc. This project will explore a subset of this area, i.e., the fascinating world of beat detection, more commonly called onset detection.

While there are many methods that can isolate onsets, this project focuses only frequency domain detection functions. There is, however, some discussion of time domain methods to give a taste of insight to frequency domain methods. The discussion is concluded with the results of several onset detection algorithms to give an empirical comparison between algorithms.

2. Introduction to Onset Detection:

"To beat or not to beat" that is the fundamental question when trying to transcribe music. Many musicians study how to transcribe music in college; it requires a lot of time and mental energy to complete the task. Automation can tremendously speed up transcription process, especially for untrained musicians; beat/rhythm detection, also known as onset detection, is the most important, element in any music transcription process.

The onset is the start of a signal event. In the case of music, the event is a change in pitch or the start of a note. Figure 1 shows a sample of a highly emphasized onset, such as the beating of a drum. The onset is very moment where the start of an abrupt change in amplitude in a signal occurs. The attack is the very moment after the onset; it is the time after the onset but before the signal reaches its peak. The decay is the exponentially decreasing amplitude of the signal after it has reached its peak. Imagine playing a steady beat on a large drum, the moment that you strike it is when you anticipate your ears hear the sound, the onset, not when you've already pressed the drum all the way down, i.e., the peak, but rather the moment before the attack.

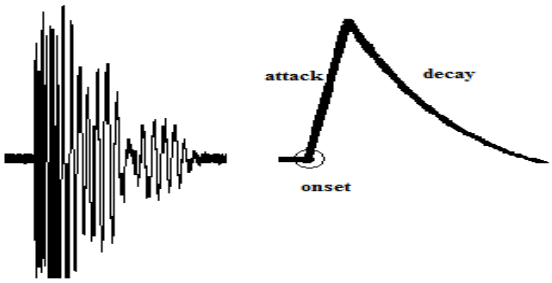

Figure 1 Music sample of an onset

The onset detection algorithms examined in this paper are those primarily proposed by [1]. They examine sound from both a signal perspective and psychoacoustic understanding of how humans interpret sound. The algorithms examine onset features based on energy, spectral content, phase deviation and high frequency content.

3. Detection Functions

The goal of a detection function is to indicate when a specific feature is present in a signal. For example, suppose we would like to know when a light switch is flipped on, if the signal is passed through an appropriate detection function, a peak will occur when the light switch is turned on. There are many detection functions that can set off the very same event. To choose the features to look at, we need to take a look at the signal and develop and understand of what happens to the signal during our desired event.

Let's take a closer look at a music signal. The signal in Figure 1 is a sample of music during a strong onset. The signal could come from a highly percussive instrument, such as marimba, glockenspiel, timpani etc. The main thing to notice is that the signal spikes quickly from being relatively flat and then the exponentially decays. The time-frequency properties of this signal can be exploited into several detection functions.

3.1.1. Time Domain Methods

For a highly percussive signal, such as in Figure 1, the outline of the signal, i.e., the reduced signal on the right, can be used directly as an onset function. [1] suggests rectifying and smoothing the signal by taking the 6 absolute value of the signal and using a moving average filter to generate a detection function. This can result in a noisy signal and smoothing can widen the peak of the signal reducing the accuracy of the time localization.

Another possible onset function is chopping the music signal into small time and calculating the average energy of each section, where **N** is the window size and **x**[n] is our signal of interest.

Another possible onset function is chopping the music signal into small time and calculating the average energy of each section, where **N** is the window size and **x[n]** is our signal of interest:

$$energy(n) = \frac{1}{N}\sum_{\frac{N}{2}}^{\frac{N}{2}-1} x^2[n]$$

The averaging of the energy leads to a sharp and highly peaked function that reduces the time resolution of the signal to reduce the amount of spurious onsets. This detection function can be further improved by measuring the change in energy by taking the derivative. This idea gets explored by [2], by using a Remez FIR filter to implement differentiation. Time domain functions are really only useful for highly percussive signals, changing from one pitch to another while keeping the energy constant would result in a smooth signal transition and the onset would go unnoticed using time domain methods..

3.2. Frequency Domain Methods

These time domain methods, according to [1], result in very noisy detection functions. Time domain signals are highly variable and their features are difficult to use as an accurate detection function. By changing domains, that is, switching from the time domain to the frequency domain, we can exploit features of the signal to obtain better detection functions than time domain methods.

Consider the Short-Time Fourier Transform (STFT) [1]:

$$X_k(n) = \sum_{m=-\frac{N}{2}}^{\frac{N}{2}-1} x(nh+m)w(m)e^{-\frac{2\pi mk}{N}}$$

Where **n** is the time location, **h** is the hop size, i.e. , the number of points to jump ahead for each STFT window, **w**(m) is a windowing function, such as Hamming, Kaiser or Hann, which reduces spectral smudging and the **k** frequency bin. When using the STFT, we do not use a sliding window, instead we hop ahead to create successive STFTS by doing so the ability to localize an onset in time becomes difficult. If we use windows smaller enough, the accuracy of the detection function's time localization is negligible and we can define the center of each window as the time instance of the STFT. Regardless of this shortcoming, the STFT is still a valid and capable tool for onset detection.

3.2.1. Magnitude Based Methods

From signal processing theory, as a signal becomes large in amplitude and short in time, its frequency spectrum widens into higher frequency spectrum. Figure 1 shows that that an onset in a piece of music can be treated as a large narrow peak, a Dirac delta function. This means that the frequency content in the higher end of the spectrum increases during an onset. The high frequencies can be emphasized by weighting them by and measuring the signal energy by:

$$HFC(n) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} |W_k X_k(n)|^2$$

If W_k is k,i.e., we are linearly weighting each frequency; this is equivalent to taking an instantaneous derivative of the signal. This is much more accurate because there is no need for an approximation for a derivative via Remez FIR filter.

While the HFC detects when there is an increase in high frequency content, we can also use the signal processing theory assumption to look at the change in spectral content. This is useful for detecting changes in pitch such as a singer transitioning from one note to another. [1] defines the spectral difference as:

$$SD(n) = \frac{1}{N} \sum_{k=\frac{N}{2}}^{\frac{N}{2}-1} \{H(|X_k(n)| - |X_k(n-1)|)\}^2$$

Where H(x) is zero for negative arguments and equal to the result for positive arguments. This is calculated by $H(x) = (x + |x|)/2$ this is to emphasize an increase in spectral content as opposed to a decrease.

3.2.2. Phase Method

An STFT assumes that the signal is a slow changing sum of sinusoids. When an onset occurs there is a discontinuity of amplitude which causes an abrupt change in the phase of these slow varying sinusoids. [1] describes that slow changing sinusoids have a continuous linear phase between frames such that $\varphi_k(n) = \omega_k t$ where ω_k is the kth frequency of interest and t is time. This is extended by [1] to:

$$\varphi_k(n) - \varphi_k(n-1) = \varphi_k(n-1) - \varphi_k(n-2) = \omega_k$$

$\varphi_k(n)$ is the phase at the **n** time of the **k** frequency bin. The assumption of linear phase can be taken a step farther by looking at the phase deviation, which is the difference in the difference of phase, i.e., the second derivative. This should be zero between successive windows; therefore, the phase deviation can be used as a detection function as:

$$\Delta\varphi_k(n) = \varphi_k(n) - 2\varphi_k(n-1) + \varphi_k(n-2) = 0$$

Putting all of this together the absolute average phase deviation is:

$$\zeta_p = \frac{1}{N} \sum_{k=\frac{N}{2}}^{\frac{N}{2}-1} |\Delta\varphi_k(n)|$$

[The absolute phase deviation is generally a poor detection function. [1] indicates large distortions occur because frequencies with little or no energy have equal phase importance. This method is useful for soft onsets. For example, a singer changing notes at a constant volume on an open vowel. The change from one to the other will be undetectable when measuring signal energy; however, it is much more noticeable in the phase deviation of the signal.

3.2.3. Combined Complex Domain Method

Energy based methods are useful for strong onsets, such as percussive instruments, the phase method is good for soft onsets. Both methods extend into opposite ends of detection problems. [3] proposes that a combined energy and phase method can be combined to create a more successful detection function. [3] uses a Euclidean distance measurement between successive frames:

$$\Gamma_k(n) = \left\{|X_k(n-1)|^2 + |X_k(n)|^2 - 2|X_k(n-1)||X_k(n)|\cos\left(\Delta\varphi_k(n)\right)\right\}^{\frac{1}{2}}$$

As with the other detection problems we are interested in the average distance between all frequency bins which is a complex domain detection function:

$$\zeta(n) = \frac{1}{N}\sum_{k=1}^{N}\Gamma_k(n)$$

4. Implementation

The almost all of the algorithms previously described rely on the same preprocessing steps with the exception of the detection function. First, the music signal is divided into overlapping sections. A windowing function, such as Hamming or Hann windows, is applied to reduce the frequency "smudging" caused by the native square, Dirichlet, window. A Discrete Fourier Transform (DFT) is taken, via FFT algorithm, with zeros padded to the signal to increase the clarity, but not the resolution, of the curve. The detection function is applied to the STFT. Finally, the peaks are picked off of the detection function and localized to a time.

4.1. Short-Time Fourier Transform

The main component to each and every detection function is the Short-Time Fourier Transform. A STFT function was not found until very late into the project, so the FFT algorithm was used in conjunction with additional coding to create an STFT function. The process is as follows: A wav file is loaded into Matlab. If it has multiple channels they are summed into a single channel. The imported sound is then normalized by dividing by the largest absolute value. Then the music signal is cut into overlapping sections. A windowing function is applied; in this case, the algorithm uses a hamming window. Apply the FFT algorithm with some zero padding to the windowed, overlapping signal. Then move the next frame of music and repeat this procedure until all parts of the signal have been processed. While the general procedure of the STFT is straight forward, the STFT does have tunable two tunable parameters that need to be appropriately chosen: the percentage of overlap between successive frames and the time length of each window.

The windowing function is used to reduce smudging of adjacent frequency components. This is done by multiplying the time domain signal by the windowing function. Using a windowing function comes at a cost. The windowing function will decimate the amplitude of the signal, throwing away information. To solve this, each STFT window has some percentage of overlap between its current window and its previous windows so that if an onset is located in a weak part of one window, it is caught by the next window; unless otherwise stated, an overlap of 50% is used in all cases.

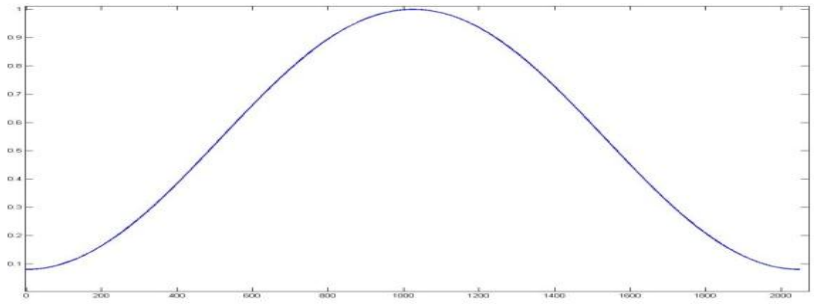

Figure 2 Hamming Window

The window time length determines the minimum onset length that we can detect, that is, the shortest note we can detect whether, it is a quarter note, half note or sixteenth note. The larger a window the smoother our detection functions becomes and the easier it is to pick the peaks off it; however, if the window gets too large, onsets get blurred together, as can be seen in Figure 3.

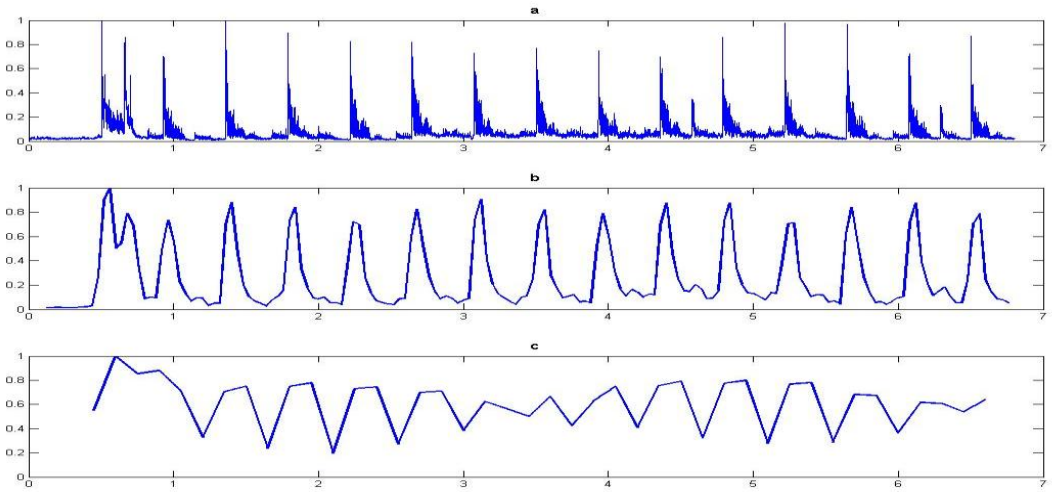

Figure 3 Euclidean Distance detection function for various window times. (a) Window Time= 0.001s (b) Window Time = 0.08s (c) Window Time = 0.3s

On the other hand, If the window is too short the processing misses areas of significant energy. On top of that, the detection function, as seen in Figure 3, becomes noisy and makes accurate peak picking difficult.

The best selection of a window time is to think of the window time in terms of the actual music. Let's assume that a piece of music has a fast tempo of 180 quarternote beats per minute (Bpm) which translates to 3 beats per second. lets say that the smallest note value time, in this unplayably fast tempo, is an eighthnote; this means that there are 12 notes a second being played, a quarternote is four eigth notes. The time period required for each of these notes is 83.3 milliseconds per note. If the sixthteenth note is the smallest note of consideration, at 180 Bpm, each sixteenth note takes about 20.3m.

An alternative consideration is a psychoacoustic aproach to hearing, if the rate of a beat pulsates on the order of human hearing the impulse will be more of a pitched frequency. Since the lowe end of the human hearing is spectrum is 20 Hz a 50ms can be seen as the mimium time allowed for the window length. When testing the algorithms times were selected between 30ms and 50s this allowed the signal to have more "rests" inbetween onsets as giving more points to work with for each onset for the peak picking algorithm.

4.2. Detection Function

After getting the series of spectrographs, from the STFT algorithm, the detection function is applied to generate a series of peaks, such as the one seen in figure 3. Each one is easily implemented in Matlab as described above.

There is some care that needs to be taken with the phase deviation and HFC functions. The phase calculations needs to keep the phase continuous $\pm\pi$; the arctangent function which is limited to $\pm\frac{\pi}{2}$ should not be used without additional logic to place the angle in the correct quadrant. In addition, the HFC function requires the amount of zero padding and sampling frequency to multiply the frequency bins by the appropriate frequencies

4.3. Peak Picking

From calculus, the initial peaks are first selected by determining the zero crossing point in differentiation, for the numerical data this was done by detecting elements that were larger than the element before and after it. If the onset signal were ideal this would be all the processing that would be needed. Looking at the detection function in Figure 3, the smaller the window size the more spurious, or false, peaks would be noted from the detection function. Noisy elements of the music, such as a high-hat resonating for several frames, can interfere with accurate detection an additional step is required to eliminate these spurious peaks.

Every paper thus far, namely [1-4,7], recommend using an adaptive threshold filter around the initially selected peaks, where the center point of the median filter is the possible peak noted by the differentiation stage of the peak picking algorithm. [1] suggests:

$$\delta(m) = \alpha + \beta * median\big(DetectionFn(k_m)\big) \;\; k_m \in \left[m - \frac{H}{2}, m + \frac{H}{2}\right]$$

Where $\boldsymbol{m}$ is the center of the window, i.e., the potential peak time found from differentiation, $\boldsymbol{H}$ is the number of points in the median filter, **α** is an absolute minimum threshold and **β** is the scaling factor of the median filter. Often $\beta = 1$ and α is not used. α, if used at all, is usually swept through to generate curves to compare various detection functions.

After finding potential peaks, if these potential peaks are above the adaptive median threshold they are still potential onsets. Looking at Figure 3a, the first peak has additional components that increase and decreases near a main peak; even though it may pass the median filter threshold it is still a spurious onset. To deal with problem we introduce a refractory period. Once an onset is detected another one cannot detected until the refractory time period has passed. From a psycho-acoustic approach, [5] says that the human ear can only perceive pitch when a signal is at least 15-20ms long, this is on the order of the window length so it must be larger to be of substantial use. [1] Recommends using 100ms refractory period but because it can cause the loss of several onsets in such a large period of time; 50ms was selected as the refractory period.

The short coming of including a refractory period is that it gives earlier onsets preferential treatment in that they are selected first. A much better way to implement the refractory period is to compare all peaks that fall

within the 50-100ms refractory period and select the peak with the largest value from the detection function; this was not implemented for simplicity, though would be suggested for future iterations of this project

5. Results

The algorithms were tested against a database of 3102 hand labeled onset times and corresponding audio files. They were provided by Music and Audio Research Laboratory (MARL) at NYU and Prosemus project at Universitat Pompeu Fabra. The databases consist of a mixed set of music files ranging from commercial to open source and from classical to pop music. The quantitative measurement of performance is to use a Receiver Operating Characteristic curve (ROC). The Y-axis represents the percentage of true positive, i.e., the total number of correctly discovered onsets versus the total number of onsets, and they X-axis represents the percentage of false positives, i.e., the total number of falsely labeled onsets versus the total number of detected onsets. The more accurate an algorithm is the more correctly determined onsets versus false positives, i.e., the plot should appear up and to the left. For generating each of the false positive versus true positives plots all variables but one was left constant. To illustrate the difference in performance between functions, while keeping all other parameters constant, the absolute minimum threshold, from the median filter; the median filter length, and the window time were swept through.

Figure 4, shows the result of sweeping through the window length and keeping all other parameters constant. The Spectral Difference function is the best performing under these conditions compared to all of the other functions.

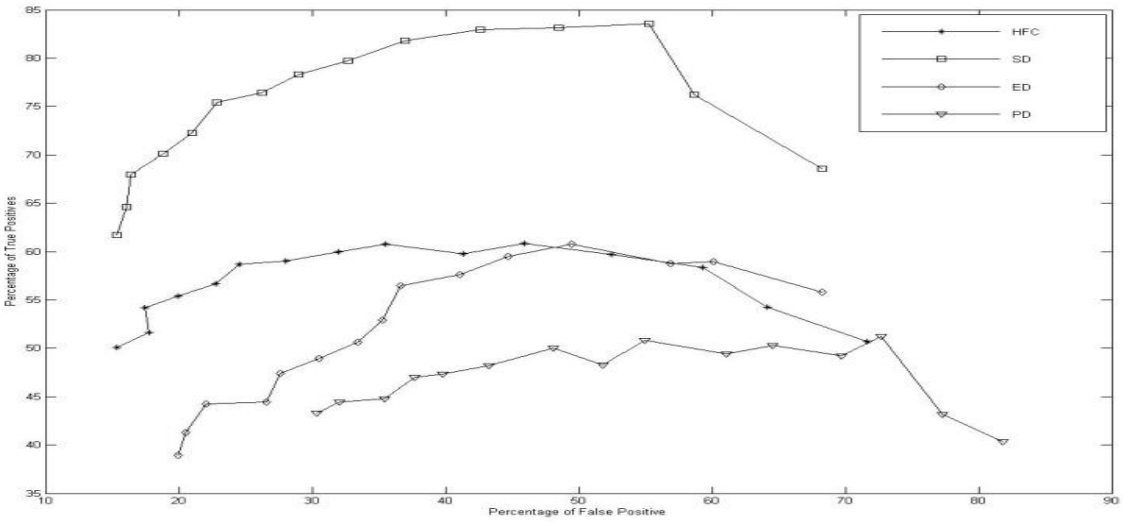

Figure 4 Sweeping through window time length (HFC = High Frequency Content, SD = Spectral Difference, ED = Euclidean Distance and PD = Phase Deviation)

As the window length increases the number of false positives decreases, i.e., the plot traces leftward. As the window length gets smaller, the detection functions approaches an instantaneous value. As the window gets larger, the instantaneous values get averaged out and features begin to approach a smoother function with well defined peaks. The error tolerance of the peak picking algorithm increases with the window length. The peak detection function will mark an onset as being correct if the absolute difference between the detected onset and the actual onset time is within half the window time. If the window becomes too large, peaks in the detection

functions will become blurred together, seen in in Figure 3 , and the true positive detection rate will decrease, as seen in Figure 4.

Figure 5 shows the effects of changing the median filter length. As the filter length increases the number of false positive decreases. Eventually the performance takes a small dip in performance because the window length becomes too large; the number of peaks within the window increases which raises the median filter height. The best performing during this sweep is the Euclidean Distance. For phase deviation, the change in filter length has no effect on its results; it's also the worst performing function.

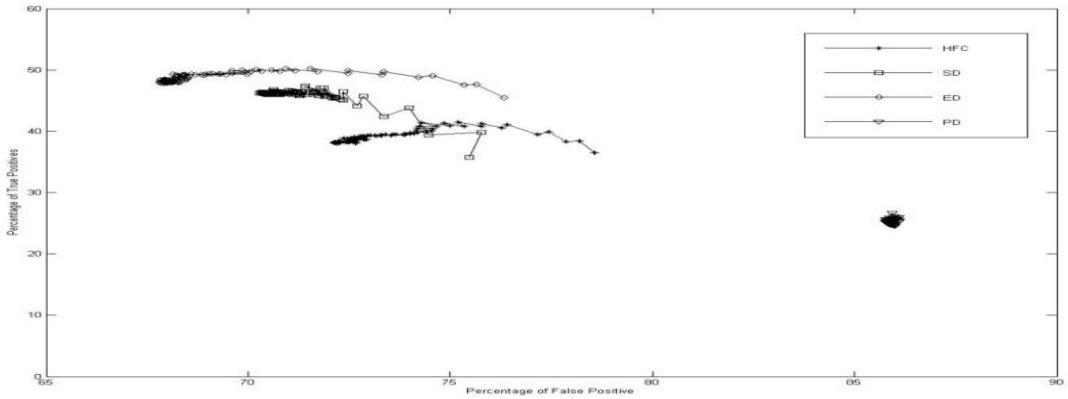

Figure 5 sweeping through median filter length

Figure 6 shows the most widely referenced sweep, the minimum threshold term of the median filter; virtually all papers on onset detection use it as the benchmark of onset detection algorithms; see [1-4].

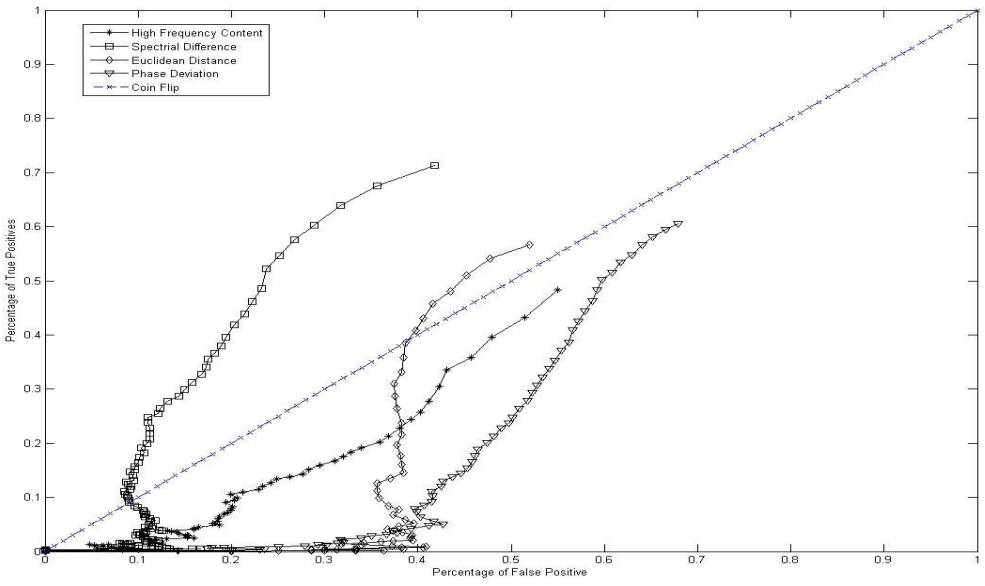

Figure 6 sweeping through constant median filter term

As seen in Figure 6, as the constant part of the filter is raised, the threshold for a peak to be marked as correct peak increases; therefore, the decrease in both true and false positives is to be expected. As the threshold is lowered, the number of true positives increases faster than the number of false positives. Eventually, there is no significant improvement in the performance and the number of spurious onsets slowly increases. While not shown on the previous plots, ROC curves have a line that goes from (0,0) to (1,1) called coin-flip line which is present in Figure 6. The coin-flip line represents the probability of guessing if an event is onset or not as If it were a coin flip. If a plot lies above this line the function performs better than flipping a coin and guessing if the event is an onset. If a function is below the coin-flip line, the function is better at assuming the opposite, the event is not an onset. [1]

6. Comments on Results:

Detection functions are highly variable and dependent on the tuning parameters associated with the peak detection function, that is, the median filter. With fine tuning, [1] has shown correct detection rates as high as 96.7% with no false positives using the MARL database and the HFC function. Even though according to Figure 6 that the HFC function is worse than flipping a coin, the effectiveness of a specific function is limited to the type of music signal that is being analyzed. [1] recommends four classifications of onset types they are: Pitched non-percussive, such as a flute, singer or clarinet; pitched percussive, such as a piano or glockenspiel; non-pitched percussive, such as tam-tam or congas; and complex mixed, such as an orchestra, jazz combo or rock band.

[1] indicates that energy based methods, such as the High Frequency Content and the time domain energy, work best for any highly percussive onset. Energy methods are computationally quick and easy to calculate; the need to take a Discrete Fourier Transform DFT can be eliminated if the signal is passed through an FIR differentiation filter which can dramatically improve computational performance time. [2]

[1] explains that complex music sets are best performed by the Spectrial Difference. The Prosemus onset database is nearly three times the size of the MARL database. The Prosemus library consists of more complexes pitched percussive and non-percussive which skews the results of the onset detection algorithm in favor the spectral difference; however, spectral difference is not without its advantages. It combines both the benefits of detecting a change in pitch as well as a change in energy. This is the best first estimate of any of the onset detection functions for general purpose beat detection.

The results have indicated that absolute phase deviation is the worst performing of all the detection functions. Only by lowering the threshold will more true positives be detected. The phase deviation is a noisy feature because elements with no significant energy still have an equal effect on the average phase as frequencies with significant energy; the Euclidean distance function will outperform phase deviation in all instances because it takes energy into consideration on top of the phase. [1, 6] proposes that a statistical approach, such as using the kurtosis, that is, “peakedness”, of the phase distribution to, combined with the inner quartile range [6] of the distribution is more effective as a detection function than using the phase directly. Further research lead to [4] which suggests giving significance to each phase by weighting the phase by its corresponding frequency amplitude, i.e., Weighted Phase Deviation (WPD):

$$WPD(n) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} |X_k(n)\Delta\varphi_k(n)|$$

From this discussion, the effectiveness of an onset detection algorithm depends on the type of music that being analyized. Even the worst performing onset detection functions can have acceptable levels of performance if

the proper type of instrument is used. These algorithms can benefit from more *a priori* knowledge about the music whether through source separation or instrument identification preprocessing algorithms to choose an appropriate onset detection scheme.

7. Conclusion:

Time localization of the start of notes is possible. The detection methods discussed here are the tip of the iceberg of possible functions. Every year the MIR community produces more and varied detection functions based on wavelet transform theory, statistical analysis, detecting changes in fundamental frequency and more. Because of the nature of onset detection algorithms, it is a multidisciplinary field; anything that relies on the detection of the start of a signal relies on it, e.g., sonar, speech, EKGs, movement detectors— virtually the whole field of signal processing stands to gain from onset detection.

8. Acknowledgement:

I would like to thank Professor Mark Fowler for his help and guidance through this project. Professor Stephen Zahorian for directing me to applications of onset detection in speech. Juan Bello for graciously providing me with the same data set used in [1] which was fundamental for developing the project and the algorithms; and the International Society of Music Information Retrieval (ISMIR) community, which has been an essential source of ideas, papers and discussion about the current state of MIR.

9. References:

[1] J.P.Bello and L.Daudet and S.Abdallah and C.Duxbury and M.Davies and M.B.Sandler " A Tutorial on Onset Detection in Music Signals," *Transactions on Speech and Audio Processing*, Vol.13,No.5,2005.

[2] M. A. Alonso, B. David, and G. Richard, Tempo and beat estimation of musical signals," *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*, Barcelona, Spain, 2004.

[3] J. P. Bello, C. Duxbury,M. Davies, and M. Sandler, "On the use of phaseand energy for musical onset detection in the complex domain," *IEEE Signal Processing Lett*., vol. 11, no. 6, pp. 553–556, Jun. 2004.

[4] Simon Dixon, "Onset detection revisited," *in Proceedings of the 9th International Conference on Digital Audio Effects (DAFx-06)*, Montreal, Canada, September 18-20, 2006.

[5] John J Bray. "Lecture Notes on Human Physiology" Blackwell Publishing. 1999.pg 173-174.

[6] J. P. Bello and M. Sandler, "Phase-based note onset detection for music signals," in *Proceedings of IEEE International Conference of Acoustics, Speech, and Signal Processing (ICASSP-03)*, Hong Kong, 2003, pp. 49–52.

Appendix A

A.1. Sample Code

```
%This is an example how to use the included code
% Define variables
% take STFT
%  Apply Detection Functions
%  Pick Peaks
%  Calculate Error

clc;
clear all;
%% Define Variables
%load music file
[music Fs] = wavread('music_bello_1.wav'); %import 10 second sound file
%load onsets depending on format
load('onsets_bello_1.mat');
onsets = T;
% Define STFT parameters
%window Time
wTime = .05;
%2^ZP_exp additional zero padding
ZP_exp = 1;
%percentage of overlap (written as a number between 1 and 100)
Percent_overlap = 50;
%median filter Parameters
%scaling parameter
gamma = 1;
%filter length
filt_len = 5;
%constant threshold
C = 0;
%peak picking refractory period
refractory = .05;

%% Detection Algorithm
%take STFT
[STFT_in times_in N_padding] = STFT_Music(music, Fs, wTime, Percent_overlap,ZP_exp);
% Apply Detection Function
```

```
%High frequency content
[detection_function_HFC times_HFC] = HFC(STFT_in,times_in,Fs, N_padding);
%Optional Normalization
detection_function_HFC = (detection_function_HFC)/max(abs(detection_function_HFC));
%pick out peaks in detection function
[peak_time_HFC] = pickpeaks(detection_function_HFC, times_HFC, filt_len, gamma, C, refractory);
%Calculate Correct and false positive rate
[correct_HFC false_pos_HFC] = peak_error(peak_time_HFC, onsets, wTime);
```

A.2. STFT_Music Function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% STFT_music Function
%  [STFT_out times N_padding] = STFT_Music(music, Fs, wTime,  Percent_overlap,ZP_exp)
%   Computes the STFT for a music signal using a hamming window for each
%   set of samples taken.
%   Outputs:
%   STFT_out is the complex output for the STFT in the range +/- Pi
%   times corresponds to the times associated with the center of each
%   window
%   N_padding is the number of points of zero padding used
%   Inputs:
%   music is them music signal whose windows are to be taken
%   Fs is the sampling rate of the windows
%   wTime is the time range of the stft windows (number of points are
%   rounded down)
%   Percent_overlap is the percentage overlap between stft windows (written as an integar)
%   ZP_exp is the power of 2 of zero padding desired for FFT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function  [STFT_out times N_padding] = STFT_Music(music, Fs, wTime, Percent_overlap,ZP_exp)
%number of STFT samples per STFT slice
N_window = floor(wTime*Fs);
%Number of overlaping points
window_overlap =floor(N_window* (Percent_overlap/100));
wTime = N_window/Fs;


%% preprocessing manupulation
%makes sure the music signal is only one channel
music = sum(music,2);
%normalize music signal
music = music/max(abs(music));

%% Size Checking
%make sure there are an integer number of windows, if not zero pad until
%they are
```

```
[music_length mus_c] = size(music);
number_of_windows = floor((music_length-N_window)/(N_window-window_overlap));%determins the number of times -1 that the overlaping window will fit the music length
number_of_points_left =  music_length - (N_window + number_of_windows*(N_window-window_overlap));
%determin the remainder
number_of_padding = (N_window-window_overlap)-number_of_points_left; %calculate the number of points neede to pad
music = cat(1 ,music,zeros(number_of_padding,1)); %append the zeros to the end of the signal
clear number_of_windows number_of_points_left number_of_padding

number_of_windows = floor((music_length-N_window)/(N_window-window_overlap)) +1;


%% STFT
%create hamming window
windowing = hamming(N_window);
%calculate the amount of desired zero padding
N_padding = 2^(nextpow2(N_window)+ZP_exp);
parfor k = 1:number_of_windows
    %define the starting and ending indicies for each window of STFT
  starting = (k-1)*(N_window -window_overlap) +1;
   ending = starting+N_window-1;
   %Define the Time of the window, i.e., the center of window
   times(k) = (starting + ceil(N_window/2))/Fs;
   %apply windowing function
   frame_sample = music(starting:ending).*windowing;
   %take FFT of sample and apply zero padding
   F_trans = fft(frame_sample,N_padding);
   %store FFT data for later
   STFT_out(:,k) = F_trans;
end
end
```

A.3 Detection Functions

A.3.1 High Frequency Content

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%[detection_function times] = HFC(STFT_in, times_in, Fs, N_padding)
%  Outputs:
%  detection_function returns the values of the detection function applied
%  at times.
%  times are the times correspoinding to the detection_function features
%      times is a modified form of times_in, multiple initial points are
%      lost due to the initial conditions needed for the detection
%      algorithm.
%  Inputs:
% STFT_in is the complexted domain STFT of the music signal
%      Rows represent the kth frequency bin
%      Columns represent the order in time the STFT was taken
% times_in are the corresponding times of the center of the STFT window
% Fs is the sampling frequency of the music
% N_padding is the total number of points used for the STFT
%      (actual points + Zero padding) = N_padding
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [detection_function times] = HFC(STFT_in, times_in, Fs, N_padding)
[nr number_of_transforms] = size(STFT_in);
%% Linearly Frequency Weighted Spectral Energy

f_plot = -Fs/2:Fs/N_padding:Fs/2 - (Fs/N_padding);
f_plot = f_plot';
parfor k= 1:number_of_transforms
F_weight(:,k) = STFT_in(:,k).*f_plot;
end
%% Alternate Power Calculation Option
% numeric integration vs summation
% power_weighted = cumtrapz(f_plot,abs(F_weight).^2,1);
% power_weighted = power_weighted(end,:);
% energy = sum(abs(F_weight).^2,1);
% detection_function = energy;
%% Calculate Power by summing all amplitudes squared
detection_function = sum(abs(F_weight).^2,1);
times = times_in;
end
```

A.3.2. Spectral Difference

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% [detection_function times] = SD(STFT_in, times_in)
%  Outputs:
%  detection_function returns the values of the detection function applied
%  at times.
%  times are the times correspoinding to the detection_function features
%      times is a modified form of times_in, multiple initial points are
%      lost due to the initial conditions needed for the detection
%      algorithm.
% Inputs:
% STFT_in is the complexted domain STFT of the music signal
%      Rows represent the kth frequency bin
%      Columns represent the order in time the STFT was taken
% times_in are the corresponding times of the center of the STFT window
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [detection_function times] = SD(STFT_in, times_in)
[nr nc ] = size(STFT_in);

%% Spectrial Difference
F_transform = abs(STFT_in);
times = times_in(2:end);
parfor k = 1:nc-1
   Spect_Diff_temp = F_transform(:,k+1) - F_transform(:,k);
   Spect_Diff_temp = ((Spect_Diff_temp + abs(Spect_Diff_temp))/2).^2;
   detection_function(k) = sum(Spect_Diff_temp,1);
end
end
```

A.3.3. Phase Deviation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
%[detection_function times] = Phase_deviation(STFT_in, times_in)
% Outputs:
%  detection_function returns the values of the detection function applied
%  at times.
%  times are the times correspoinding to the detection_function features
%      times is a modified form of times_in, multiple initial points are
%      lost due to the initial conditions needed for the detection
%      algorithm.
% Inputs:
% STFT_in is the complexted domain STFT of the music signal
%      Rows represent the kth frequency bin
%      Columns represent the order in time the STFT was taken
% times_in are the corresponding times of the center of the STFT window
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [detection_function times] = Phase_Deviation(STFT_in, times_in)


[nr nc] = size(STFT_in);
F_phase = 0;
parfor k = 1:nc
   F_Phase(:,k) =  angle(STFT_in(:,k));%find phase of k FFT points for later use;
   %store all data in sample by sample chunks
end

%% phase deviation
%simple phase calculation by averaging the phase deviation across all
% frequency bins
for k = 3:nc
   phi_k = F_Phase(:,k) - 2*F_Phase(:,k-1)+F_Phase(:,k-2);
   detection_function(k-2) = sum(abs(phi_k))/nr;
end
times = times_in(3:end);
end
```

A.3.4. Euclidean Distance

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Euclidean Distance
%[detection_function times] = Euclidean_Distance(STFT_in, times_in)
%  Outputs:
%  detection_function returns the values of the detection function applied
%  at times.
%  times are the times correspoinding to the detection_function features
%       times is a modified form of times_in, multiple initial points are
%       lost due to the initial conditions needed for the detection algorithm.
% Inputs:
% STFT_in is the complexted domain STFT of the music signal
%       Rows represent the kth frequency bin
%       Columns represent the order in time the STFT was taken
% times_in are the corresponding times of the center of the STFT window
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [detection_function times] = Euclidean_Distance(STFT_in, times_in)
[nr nc] = size(STFT_in);
parfor k = 3:nc
   X_n = STFT_in(:,k)
   X_n_1 = STFT_in(:,k-1)
   X_n_2 = STFT_in(:,k-2)
   delta_phi = angle(X_n)-2*angle(X_n_1)+angle(X_n_2);
   euclid_dist = sqrt(abs(STFT_in(:,k)).^2 + abs(STFT_in(:,k-1)).^2  - 2*(abs(STFT_in(:,k)).*abs(STFT_in(:,k-
1)).*cos(delta_phi)))
   [nr nc] = size(euclid_dist);
   detection_function(k-2) = sum(euclid_dist)/nr;
end
times = times_in(3:end);
end
```

A.4. Peak Picking Algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%[peak_time filt] = peakpick_music_median(detection_function, times, filt_len, gamma, C)
% returns times of peak as well as median filter used
% Outputs:
% peak_time is the time corresponding to the peaks in Detection_function
% filt is the filter threshold used for all points
%
% Inputs:
% detection_function - function with peaks which you wish to detect
% times - times corresponding to points of the detection function
% filt_len - length of median filter
% gamma - scaling factor for median filter
% C - constant minimum threshold
% refractory - minimum allowable time between peaks
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [peak_time filt] = pickpeaks(detection_function, times_in, filt_len, gamma, C, refractory)

   [fn_len nc] = size(detection_function);
   med_filt = medfilt1(detection_function,filt_len);%median filter function
   filt = gamma*med_filt + C; % thresholding function
  %see where the reduction function is above the threshold
   thresholded = detection_function -filt; %subtract threshold

   [~, loc] = findpeaks(detection_function); %find peaks
   %find only the peak points where it is above the threshold filter
   index = find(thresholded(loc)>=0);
   index = loc(index);%use only these points for peak picking
   %eliminate redundant onsets bey introducing a refractory period.
   [nr nc] = size(index);
   if (nc ~= 0)
     %eliminate spurious peaks by introducing a refractory period
     redundant_peak_times = times_in(index);
      temp_times(1) = redundant_peak_times(1);
      parfor k = 2: length(redundant_peak_times)
      diff_time = redundant_peak_times(k) - redundant_peak_times(k-1);
      if (diff_time < refractory)
         temp_times(k) = -1;
       else
          temp_times(k) = redundant_peak_times(k);
      end
      end
    peak_time = temp_times(find(temp_times>=0));
   else
      peak_time = [];
   end
end
```

A.5. False Positive and True Positive Calculations

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% [correct false_pos num_correct_onset num_FP] = peak_error(onset_times,
% actual_onsets,wTime)
% Returns the percentage error and false positives between onset times
% Outputs:
% correct - percentage of correct onsets
% false_pos - percentage of false positives detected
% num_correct_onset - returns the number of correct onsets
% num_FP - returns the number of false positve
% Inputs:
% onset_times - the time of an onset detected via various algorithms
% actual_onsets - the prenotated onsets that are known before processing
% wTime - the time used for each STFT window
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [correct false_pos num_correct_onset num_FP] = peak_error(onset_times, actual_onsets,wTime)
%initialize matricies
correct_onset_time = [];
false_positive= [];
marked_onsets(1:length(actual_onsets)) = 0;

%for each onset
for k = 1:length(onset_times)
    %find absolute distance between detected onset and onset list
   time_dist = abs(actual_onsets - onset_times(k));
    %find the closest onset
   [close_time Index] =  min(time_dist);
   %if the smallest distance is within one windowlength and hasnt been detected

  if (((wTime/2) >= close_time) && (marked_onsets(Index) == 0))
       %mark it as a correct onset
       correct_onset_time(k) = onset_times(k);
       %mark the onset as being used so double counting does not occur
       marked_onsets(Index) = 1;   else
       %otherwise it is a false positive
       false_positive(k) = onset_times(k);
    end
end

%due to indexing isolate all correct onsets
correct_onset_time = correct_onset_time(find(correct_onset_time>0));
%isolate all false positive
false_positive = false_positive(find(false_positive>0));

%count the number of false positives
num_FP = length(false_positive);
% count the number of detected onsets
num_onset = length(onset_times);
%count the number od actual onsets
num_act_onset = length(actual_onsets);
```

```
%count the number of correct onsets
num_correct_onset = length(correct_onset_time);
%calculate false postive error percentage
false_pos = (num_FP)/num_onset;
%calculate correct onset error percentage
correct = num_correct_onset/num_act_onset;
end
```