An Implementation of Real-Time Beat Tracking using Raspberry Pi

Michael Canary, Theodore Nikolaitchik, Nikhil Kandpal, Jonathan Dassoulas and Ganesh Sivaraman*
*University of Maryland, College Park. Team ID: 24765

***Abstract*—Identifying beat times in a musical piece is a difficult task with a wide array of applications in music synchronization, transcription, and improvisation [1]. In this paper we describe an algorithm and associated visual output for beat tracking. Our method improves upon the Ellis algorithm [2] to make it suitable for real-time applications. In addition, we propose a method for adding an adaptive aspect to the algorithm to improve beat tracking in songs with time-varying tempo. This adaptive quality comes from a metric that is used to measure the quality of previous tempo estimates based on the periodicity of a beat scoring function calculated over time.**

I. Introduction

Our goal in this project was to design an algorithm that is able to perceive musical beats in real time. This algorithm was implemented on a Raspberry Pi which communicated with a connected Arduino to give visual output to indicate instances where beats occurred. Beat tracking is an important yet difficult task that is fundamental to understanding music. A person, whether musically inclined or not, is able to immediately identify the beat in most songs. Since beat is such an important characteristic in our perception of music, beat tracking has many applications in fields like automated music synchronization, transcription, and improvisation [1].

In the past, some beat tracking systems have attempted to track certain features which clearly indicate downbeats, such as drum beats [3]. However, this class of algorithms was only applicable to music containing clear drum patterns. Other methods have used MIDI input to determine beat times in songs [4]. The use of MIDI restricts the algorithms use-cases to songs where MIDI data is available, which tends to be simpler songs with fewer instruments. The approach that we chose to emulate was to use a dynamic programming algorithm to determine beat instances based on the beat likelihood at each point in time and the even spacing of beats [2] [5].

To indicate beat times in a clear and creative manner, our team utilized the GPIO capabilities of an Arduino. By separating the implementation of the algorithm (Raspberry Pi) and the implementation of the output (Arduino), we were able to ensure that the algorithm had the necessary computing resources to perform in real time.

II. Real Time Beat Tracking Algorithm

The real-time algorithm described here is an adaptation of the dynamic programming approach to beat tracking described by Daniel Ellis [2]. Dynamic programming (DP) is a suitable algorithmic paradigm for real-time applications for two reasons. First, DP algorithms generally compute functions dependent on past and present inputs. This fits the one main constraint of real time computing that no future inputs are used for the current time steps computation. In addition, determining beats based on past audio as well as the current time steps audio allows for a more robust algorithm than only considering current audio. Second, DP computes recurrences in linear time by storing and using all intermediate recurrence values. Therefore, the algorithms speed in computing the next recurrence value stays constant regardless of the length of the audio input.

The algorithm begins by calculating the onset envelope, a function meant to measure the onset of notes. The onset envelope is calculated from the first-difference of the audios mel spectrogram. The first difference is half wave rectified and then summed over all frequency bands to give the onset envelope. This function essentially captures instances when the frequency composition of the audio increases in one or multiple frequency bands from one time step to the next, therefore acting as a measure of when notes begin. This function is quite simple to calculate in real time. The only previously computed values that the function depends on are the previous time steps mel spectrum coefficients so that the first difference can be taken. In the Ellis paper, this raw onset envelope was smoothed by convolving the signal with a Gaussian. To perform this convolution in real time, each new raw onset value was used to scale a shifted Gaussian 20 milliseconds wide. This scaled Gaussian was then added to the onset envelope rather than the raw onset value.

After calculating the onset envelope at 250 Hz for 3 seconds, a global tempo estimation is calculated by looking for periodicity in the onset envelope. The assumption here is that note onsets are most likely to appear one beat apart from each other. An autocorrelation is performed on the onset envelope and local maxima in the autocorrelation are used to find the songs tempo estimate. For more information on the global tempo estimation, see the Ellis papers section on the Global Tempo Estimate. [2]

After the global tempo estimate is established, each time step can be assigned a cumulative score evaluating whether or not it contains a beat. The current time steps cumulative score is based on two factors: the onset envelope strength at the current time step and the cumulative score from past points in time approximately one beat ago as determined by the global tempo estimate. This is expressed in the following formula

$$C(t) = O(t) + \max_{\tau=0\ldots t}\{\alpha F(t-\tau, \tau_\rho) + C(\tau)\} \qquad (1)$$

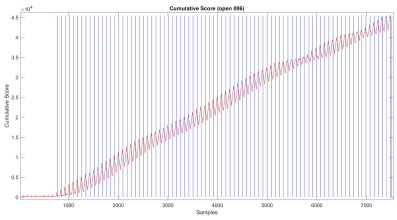

Fig. 1. Characteristic shape of the cumulative score function with an accurate initial tempo estimate

where $C(t)$ is the cumulative score, $O(t)$ is the onset envelope, τ_ρ is the tempo estimate, and $F(t-\tau,\tau_\rho) = -(log(\frac{t-\tau}{\tau_\rho}))^2$. The function $F(t)$ serves as a penalty term for using previous values of $C(t)$ that are far from one beat prior to the current time. The parameter α is a tightness parameter that can make the penalty term more or less important in the previous beat search. Due to the inclusion of the penalty term, the search for the time step that maximizes the expression $\alpha F(t-\tau,\tau_\rho)+C(\tau)$ can be restricted to $\frac{1}{2}\tau_\rho$ to $2\tau_\rho$ time steps prior to the current time, since it is unlikely that $\alpha F(t-\tau,\tau_\rho)+C(\tau)$ will be large for τ outside of that range.

Beats were identified in real time from the cumulative score function. There were three criteria that needed to be met for a time step to be considered a beat. First, the cumulative score must have seen a local maximum. Second, the value at that local maximum must have been greater than any of the previously seen local maxima. Third, the cumulative score function must decrease for at least β time steps after the local maximum. β acts as a threshold that needs to be large enough so that high frequency noise in the cumulative score function would not spuriously indicate beats. At the same time, β can not be too large since β determines how long after a peak the algorithm must wait before signaling that a beat occurred. The β value that was settled upon in the final implementation was 8 time steps which equates to a delay of 32 milliseconds.

A. Adaptation of Tempo Estimate

For some test songs in the training set, the first 3 seconds of audio yield a poor global tempo estimate. To fix cases like this, at the 10 second point, a tempo accuracy metric is calculated. This metric is based off of the fact that a cumulative score function with a good tempo estimate has a characteristic shape. This shape can be seen in Figure 1 showing the graph of the cumulative score generated from song open006 in the training set.

In a cumulative score function that uses an accurate tempo estimate, there are clear peaks and troughs and each peak is roughly one tempo estimate apart. To identify this shape, the metric is calculated by taking the first difference of the cumulative score. In cases where the tempo estimate is good, this yields a shape similar to a square wave with period equal to the tempo estimate. Then an autocorrelation is done on this first difference. If the largest local maximum is located within 10 samples of the tempo estimate, then the tempo estimate is considered to be good. However, in cases where the tempo estimate is poor and the cumulative score does not have this characteristic shape, the tempo estimate is recalculated using 10 seconds of the onset envelope. For the remaining 20 seconds, the second tempo estimate is used. A prime example of when this tempo re-estimate improved the algorithms performance can be seen in Figure 2 cumulative score function generated from song closed011 in the training set.

The tempo re-estimate not only compensates for poor initial tempo estimates, but also adds an adaptive capability to the algorithm. With this feature, the algorithm is able to more ably track beats in songs with variable tempo.

III. Embedded Implementation Using Raspberry Pi

The embedded hardware platform we chose to implement our beat tracking algorithm on was a Raspberry Pi. Raspberry Pi provided a small and portable, yet still powerful platform to implement our algorithm on. With 1 GB of RAM, and a quad core ARM CPU clocked at 1.2 GHz, the Pi provided more than enough computing power for this application. In fact, benchmarking tests showed that the Raspberry Pi was able to compute FFTs faster than any some of our laptops, which had more powerful hardware than the Pi. We attributed this unexpected outcome to the lack of background processes running on the Pi compared to our laptops, and also to the lightweight operating system we installed on the Pi: a distribution of Linux called Raspbian Jesse Lite.

The team selected the most lightweight of the common Linux distributions for Raspberry Pi to maximize the performance of our real-time beat tracking software. In fact, the operating system selected lacked a GUI; all navigation within the Raspberry Pi had to be accomplished through the use of a terminal. An advantage of Raspberry Pi for this application is that it comes with native support for Python, the language we chose to implement our real time beat tracking algorithm in. Python proved to be a versatile language, and we were able to find many tools for signal processing and audio capture which were useful in developing our real time beat tracker. One of the disadvantages of the Raspberry Pi platform was the lack of an audio input jack on the board. In order to solve this problem, the team purchased a small USB sound card dongle, which enabled us to use one of the four USB ports on the Raspberry Pi as an audio jack, with both an input and output. Through a combination of this dongle, an old desktop microphone, and some USB audio drivers for the Raspberry Pi, the team was able to successfully capture audio with the Pi. Within the algorithm itself a Python library called PyAudio was used to sample audio for the beat tracking algorithm to process.

To produce an output to show when the algorithm detects a beat, we chose to use a blinking effect on two RGB LEDs. One LED turns on or off every time a beat is detected, and the other LED blinks a random color each time a beat is detected. This output was controlled using an Arduino; moving the processing required to blink LEDs off of the Pi let the Pi

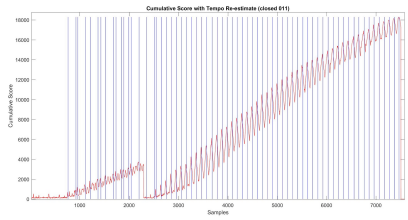

Fig. 2. Example of song where the initial tempo estimate was poor and the adaptive tempo re-estimation at 10 seconds improved performance

focus all of its computing power on the actual beat tracking, and put the load of maintaining 3 PWM outputs and generating a random color on the Arduino. The two microcontrollers are connected via a common ground and one digital pin: an output on the Pis side and an input on the Arduinos side: whenever the Raspberry Pi detects a beat it changes the state of this pin, and the Arduino responds accordingly by producing an output on the LEDs.

A demonstration of our implementation can be viewed by https://youtu.be/kSKw3m-RxaQ

IV. Conclusion

Overall, the results from the implementation of our beat tracking algorithm were good on most of the songs except a few difficult ones. On most pieces, the algorithm performed well, and on many it performed nearly perfectly with only a few missed beats. However, on a small subset of the training pieces, the algorithm produced poor results. The songs on which the algorithm struggled break down into two main categories. The first is songs whose tempos are derived from transitions between notes that have similar frequency composition. The second is songs with prolonged periods of silence during the periods critical for the tempo estimation.

The class of songs whose tempos are derived from onsets of similar sounding notes were difficult to track because transitions between notes with similar frequency compositions do not show up prominently in the onset envelope. Since the onset envelope was calculated from the mel-spectrogram, notes that had similar enough frequency compositions would map to similar mel spectrum coefficients. As a result, the first difference of the mel-spectrogram during a transition between two similar notes would be very small. A solution to this problem would be to increase the number of mel coefficients calculated for the spectrogram as this would increase frequency resolution. However, using too many mel spectrum coefficients could also be detrimental. If the frequency resolution were too fine-grained, then distortions introduced by the microphone would have a larger effect on the onset envelope.

The class of songs with prolonged periods of silence were difficult to track for obvious reasons. We observed that the most critical part of the algorithm was the tempo estimation because a poor tempo estimate could make the algorithm miss all of the beats. Songs with silent portions usually do not yield good tempo estimates since the tempo estimate is derived from note onsets. We attempted to address this weakness by adding the adaptive tempo estimation, but for some algorithms even the second estimation was corrupted by periods of silence.

In future, we would like to explore gammatone filterbanks instead of Mel filterbanks as gammatone banks have shown superior performance in several audio enhancement algorithms as compared to Mel filterbanks.

Acknowledgment

We'd like to thank Professor Carol Espy-Wilson at the University of Maryland for putting out the call to organize this team. We would also like to Ganesh Sivaraman for his consistent guidance and support. Without him this team would not have gotten as far as it has.

References

[1] P. E. Allen and R. B. Dannenberg, "Tracking musical beats in real time," in *Proceedings of the International Computer Music Conference*, vol. 1990. International Computer Music Association San Francisco, CA, 1990, pp. 140–3.

[2] D. P. Ellis, "Beat tracking by dynamic programming," *Journal of New Music Research*, vol. 36, no. 1, pp. 51–60, 2007.

[3] M. Goto and Y. Muraoka, "A real-time beat tracking system for audio signals," in *Proceedings of the international computer music conference*. San Francisco: International Computer Music Association, 1995, pp. 171–174.

[4] S. Dixon and E. Cambouropoulos, "Beat tracking with musical knowledge," in *ECAI*, 2000, pp. 626–630.

[5] J. Laroche, "Efficient tempo and beat tracking in audio recordings," *Journal of the Audio Engineering Society*, vol. 51, no. 4, pp. 226–233, 2003.