Alessandro Ragano
Mattia Gabbrielli
Davide Fiocchi

Computer Music project: Multipath Beat Tracker

This work was carried out as a practical implementation of the idea described in the Di Giorgi's paper "Multipath Beat Tracking" [1] to have an easy, portable and flexible tool to visualize some results. The Vamp Plugin was a good option because could be used inside some host program (i.e. SonicVisualizer) to graphically see the result of the computation.

In the Di Giorgi's paper [1] it's written that the experimental results are carried out using the ODF proposed in the reference 25 [2] and the IBI proposed in the reference 26 [3]. We implemented this along with the Di Giorgi's code for the Multipath part.

Contents

Algorithm Principle

The Multipath (as most of the beat tracking algorithms) extract the beat information from the Onset Detection Function (ODF) and the InterBeat Interval (IBI). Here we describe briefly the algorithm we have implemented

ODF

This function gives an estimation on where there is an onset in an audio file. An onset "*refers to the beginning of a musical note or other sound*" [4]. The higher the ODF(n) is, the higher is the probability to have an onset in the n-th frame of the audio input signal.

We implemented the SuperFlux (SF) algorithm described by Bock in [2]. This algorithm works really well, gives a "sharp", very clear ODF. The formulas of the algorithm:

$SF(n) = \sum_{m=1}^{M} H(X(n,m) - X(n-\mu,m))$ where

$X(n,m)$ is the m-th bin of the n-th frequency frame of the input signal and

$\mu = 2$ and

$H(x) = \frac{x+|x|}{2}$

IBI

This function gives an estimation of the interval of time in between two beats, given the ODF.

We implemented the algorithm described by Davies in [3]. The ODF is partitioned into overlapping frames of length $B_f = 512$ samples and hopsize $B_h = 128$ samples, for each frame of the ODF we obtain a value of the IBI. So, for each frame of the ODF, we should estimate the beat period τ_G as the index of the maximum value of the function:

$y_G(\tau) = \sum_{l=1}^{B_f} A(l) \cdot F_G(l,\tau)$ where

$A(l) = \frac{\sum_{m=1}^{B_f} ODF(m) \cdot ODF(m-l)}{|l-B_f|}$ is the normalized autocorrelation function of the ODF (half-way rectified after subtracting its mean) and

$F_G(l,\tau) = w_G(\tau) \cdot \lambda_\tau(l)$ is the shift invariant filterbank (in matrix form), where

$w_G(\tau) = \frac{\tau}{\beta^2} e^{-\frac{\tau}{2\beta^2}} \quad \tau = 1, \dots, B_f$ is the lag attenuation function and

$\lambda_\tau(l) = \sum_{p=1}^{4} \sum_{v=1-p}^{p-1} \frac{\delta(l-\tau p+v)}{2p-1} \quad l = 1, \dots, B_f$ is the comb template.

At the end of this process (when there are no more ODF frames to analyse) we interpolate the IBI values obtained in order to obtain the same rate as the ODF.

Multipath

This part is already extensively described in Sec. 2 of the Di Giorgi's paper [1].

Vamp Plugin Architecture

Vamp is an audio processing plugin system for plugins that extract descriptive information from audio data, typically known as audio features. A Vamp plugin is a binary module that can be loaded up by a host application and fed audio data, just like a VST. It could be run on SonicVisualiser host which allows to display the features extracted by the plugin together with the song. A Vamp plugin needs to make a certain amount of information available to the host such as: the description, the name, the identifier, the names of the authors, the step size and the block size, a list of output descriptors and so on.

The Vamp SDK contains the main class “Plugin” from which all the classes representing the plugin implementations should be derived. The main virtual methods are:

bool initialise(size_t inputChannels, size_t stepSize, size_t blockSize);

Here is the place where you can define some initialisation parameters. The passed parameters in the body function are fixed when this method is called by the host. The method return false when the initialisation is failed.

2) FeatureSet process(const float *const *inputBuffers, RealTime timestamp);

This is the method where the core execution of the plugin lives. Here you need to write the code you want to be executed in real time. Each time process is called, it is passed a single block of audio of size in samples equal to the block size that was passed to initialise. The audio is provided in either time domain (PCM samples) and frequency domain(STFT). In getInputDomain you need to specify it.

FeatureSet objects:

the plugin returns a FeatureSet object. This is an STL map whose key is an output number and whose value is a FeatureList, which is an STL vector of Feature objects. The use of a FeatureList allows the plugin to return features with more than one timestamp from a single process call, or to return all features for the entire audio input in a single FeatureSet from getRemainingFeatures.

Build a Vamp plugin

Tutorial for OS/X: https://code.soundsoftware.ac.uk/projects/vamp-plugin-sdk/wiki/mtp1
Tutorial for Windows: https://code.soundsoftware.ac.uk/projects/vamp-plugin-sdk/wiki/mtp2 (actually, this tutorial is for an older version of the Vamp Plugin SDK, we had some problems)

Results

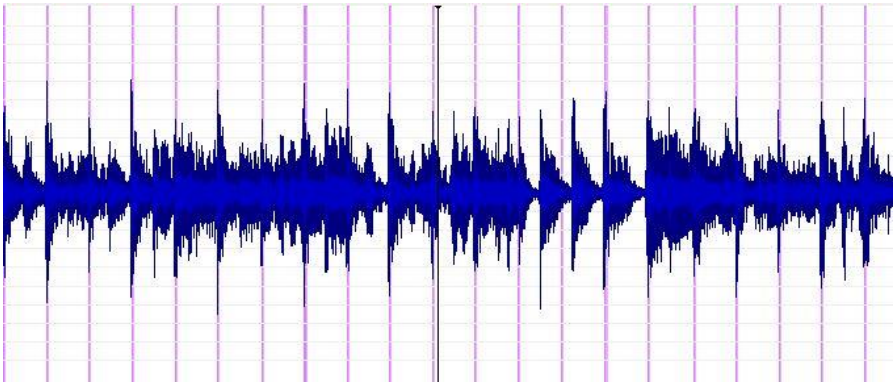

Fig a. Result of a beat tracking evaluation

In order to get a right evaluation we perform some tests on the dataset “The Beatles Annotations” * [7], then we compared the results with the ones computed in the section 3.1 [1]. The parameters are set with these values:

number of Trackers = 18
stability = 40

We used the “Beat Tracking Evaluation Toolbox” by Davies, Stark [6]. It is a toolbox written in python which compares two annotations (the one from the database compared with ours) the and returns a set of measures.

After importing the library, it is just required to run a function
evaluate_db(annotations,beats,measures,doCI)

where the arguments are as follows:

- 'annotations' is a list of numpy arrays containing the beat annotations for the database
- 'beats' is a list of numpy arrays containing the beat estimations for the database
- 'measures' is a list of strings indicating which evaluation measures to use. Setting measures='all' calculates all measures.
- 'doCI' is a boolean indicating whether or not calculate confidence intervals (we used True).

The obtained results are the following (results [confidence interval]):
fMeasure = 69.47 [62.22, 76.46]
cemgilAcc = 48.72 [42.25, 54.61]
gotoAcc = 52.83 [39.62, 66.04]
pScore = 76.78 [69.94, 83.21]
amlT = 73.50 [66.27, 79.69]
infoGain = 2.02 [1.84, 2.21]

Comparing with the ones in [1] section 3.1 we have similar values for both fMeasure and amlT. However, there is a slightly different result for the infoGain which is lower. In general, for all the measures, higher value is better.

In conclusion, we can state that the Beat Tracker has good results depending on some built-in songs aspects. They are: too much compression, bootleg, poor quality recording, etc... All those aspects can affect the result.

* The number of tested songs is 54 because in the dataset we noticed some big differences with our computed beats instants even though we had good results looking them in SonicVisualiser. We decided to don't take into account all the songs affected by these features, supposing that they are different versions or not always exactly "correct" as explained by the authors in http://isophonics.net/content/reference-annotations.

Source Code Organization

Mainly the code is organized in three files:

- MyPlugin
- getODFValue
- BeatPeriod

MyPlugin was a skeleton file to start the development of our plugin. As described above in the Vamp Plugin Architecture, there are two main functions: process and getRemainingFeatures.
In the function process we compute the ODF, we get one value for each audio frame analysed and we store it in a vector to use it later to retrieve the IBI value.
In the function getRemainingFeatures we compute the beats: we start from the calculation of the IBI and then this result is passed to the Multipath code. GetRemainingFeatues works with no time stamp by default, but we need to assign a correct time stamp for each beat (feature) returned. We needed to use some methods from the SDK and to declare the sampleType of the output descriptor as VariableSampleRate. Otherwise you will get a result related to the last time instant of the song. In this way the time of each feature is returned in the timestamp of the feature.

getODFValue has two main functions: getODFValue and normalizeODF.
GetODFValue returns the ODF value returned from the execution of the algorithm explained in the section Algorithm Principle.
NormalizeODF normalizes the vector with all the ODF values and apply the smoothing.

BeatPeriod calculates the IBI vector by applying the algorithm described in the section Algorithm Principle. Here a brief explanation of each function:

Overlapping_ODF: partitions the ODF into overlapping frames, stores the result in a matrix.
Adaptive_moving_mean: calculates the adaptive moving mean from the overlapped and framed ODF.
Hwr: applies the half way rectification in order to discard the non-useful values.
Unbiased_autocorrelation: calculates the autocorrelation function.
Output: calculates $y_G(\tau) = \sum_{l=1}^{B_f} A(l) \cdot F_G(l,\tau)$. Calls the function comb_filterbank and multiplies its output with the autocorrelation function.

- *Comb_filterbank*: calculates the $F_G(l,\tau)$ for each τ or l. Weights (with Rayleigh function) the output of a comb template.
- *Comb_template*: calculates the comb filter output $\lambda_\tau(l)$ as seen in the Fig.2 of [3]

Beat_period: calculates the IBI value as $\tau_G = arg\max_{\tau} y_G(\tau)$.
Interpolate: this function interpolates all the IBI values to have the same rate as the ODF.

References

[1] B. Di Giorgi, M. Zanoni, S. Bock and A. Sarti “Multipath Beat Tracking”

[2] S. Bock and G. Widmer, “Maximum Filter Vibrato Suppression for Onset Detection,” in Proceedings of the 16th International Conference on Digital Audio Effects (DAFx-13), Maynooth, Ireland (2013)

[3] M. E. Davies and M. D. Plumbley, “Context Dependent Beat Tracking of Musical Audio,” Audio, Speech, and Language Processing, IEEE Transactions on, vol. 15, pp. 1009–1020 (2007 Mar.)

[4] Onset definition: https://en.wikipedia.org/wiki/Onset_(audio)

[5] All information about Vamp Plugins: http://www.vamp-plugins.org/

[6] Beat Tracker evaluation toolbox: https://github.com/adamstark/Beat-Tracking-Evaluation-Toolbox

[7] The Beatles Annotations: http://isophonics.net/content/reference-annotations-beatles