Detection of music tempo with beats per minute (BPM) detector

Pedro Macedo
Faculty of Computer and
Information Science
Ljubljana, Slovenia
Email: pf06708@student.uni-lj.si

***Abstract*—The purpose of this project is to classify the tempo of each audio file by checking the audio Bits Per Minute (BPM). The implementation will be done in Python by using open source libraries specified for sound analysis.**

***Index Terms*—Virtual Reality (*VR*), Augmented Reality (*AR*), Spatial Audio, Location-Guided Audio-Visual Spatial Audio Separation (*LAVSS*), Machine Learning (*ML*), Bits-per-minute (*BPM*)**

I. Problem

The main goal of this project is to develop a sophisticated web application that accurately recognises the tempo of music tracks by analysing their *Beats Per Minute* (*BPM*). This functionality is crucial for a variety of practical applications, from music recommendation systems to fitness applications that use the tempo of music to increase user engagement.

The project employs advanced signal processing techniques and leverages the power of open source sound analysis libraries to ensure high accuracy in BPM detection. By integrating these technologies, the system aims to provide precise BPM values for a variety of music genres, which are essential for correctly determining the tempo of the music.

The aim is to provide a robust and user-friendly tool that:

- Accurately recognise and display the BPM of any music track.
- The BPM data is used to classify the music tempo from a predefined BPM tempo list.

II. Solution

In this section, we demonstrate the implemented solution by presenting the solution's architecture, the system requirements and steps to use this solution.

A. Requirements

TO-DO: Create a more detailed architecture on the *backend* part, and the interaction between *frontend* and *backend*.

As we can see in Fig. 1, this system consists of two different components, *backend* and *frontend* components. The *backend* of the system was developed in *Python* and uses the *FastAPI* library to build an API for our system. The frontend of the system was developed in *React* and uses the *Tailwind CSS* framework for styling our website. It also uses, the *PyDub* library to convert an *MP3* file into a *WAV* file, and the *Librosa* library to load the received audio data and convert them into floating point time series.

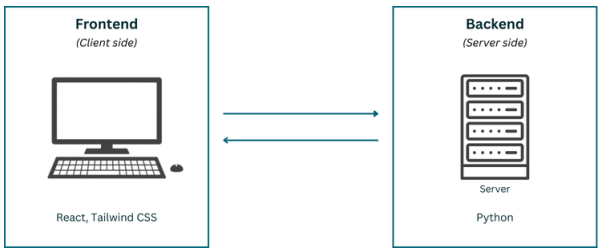

Fig. 1. System Architecture. This figure shows the various components of this system, client and server components.

B. Requirements

To ensure the development of a robust BPM detection system, it is essential to establish a set of requirements. Our system needs to follow these requirements:

- **Functional requirements**:
  - The system should accept multiple file types (*MP3* and *WAV*).
  - The system should allow users to upload a file via the web interface.
  - The solution must detect *BPM* of the uploaded audio file.
  - The solution should handle changes in BPM during the duration of the song.
  - The system should present a real-time *BPM* graph while the song plays in the background.
  - The system should allow users to view their history of previous *BPM* detections.
- **Non-Functional requirements**:
  - The web interface should be intuitive and easy to navigate.
  - The system should be reliable by ensuring consistent *BPM* detection across multiple runs of the same audio file.

C. How to use the solution?

In Fig. 3, you can see the main page of the system, where the user can upload any music file and see the *BPM* values for every second of the song while the song is playing in the background. In addition, the user can view their history by clicking on the button in the top right corner.

Fig. 2. Main page. This figure shows the main page of our system with the various functionalities available.

III. Procedure

In our solution, we implemented two approaches. The initial approach uses energy and peak detection to calculate *BPM* on each sample. The second approach adopts a hybrid multi-band approach that splits the audio frequency ranges into three frequency ranges.

A. Initial approach

In the initial approach, we applied a **bandpass** filter, so that the system only focuses on frequencies that are relevant for the detection of beats (range of musical frequencies). This reduces noise and irrelevant frequencies, improving the accuracy of the system. Then, we divided the signal into 5 second samples. For each sample, we calculated the energy of the wave, and we smooth this energy using a uniform filter over a 100ms window to reduce noise.

In order to obtain the overall *BPM* of the uploaded song, the following procedure was established:

1) **File conversion**: Firstly, the file was converted from an *MP3* file type into a *WAV* file type by using the *Fourier-Fast Transform* (*FFT*) method.

$$x(t) = \int_{-\infty}^{\infty} x(t) * e^{-j2\pi ft}\, dt \tag{1}$$

2) **Audio division**: Subsequently, the audio was splitted into 1 second samples. By splitting the audio into 1-second samples, it allows to process the audio and detect the music tempo more accurately.
3) **Smooth audio sample**: For each segment, the audio segment was smooth by calculating the its energy, to ensure that the *BPM* detection is based on genuine patterns rather than spontaneous artifacts or noise.

$$E(t) = |A|^2 \tag{2}$$

where A is the amplitude of the signal at time t.

4) **Peaks detection**: Moreover, we detected significant audio peaks (by defining a peak height threshold and a minimum distance between the detected peaks).
5) ***BPM* calculation**: If it was detect at least 2 peaks, the *BPM* is calculated based on the average interval between peaks.

$$BPM = \frac{60}{average\_interval} \tag{3}$$

If not at least two peaks were detected, we apply the interpolation process between the two nearest neighbours.

6) **Overall song BPM**: Furthermore, we applied the *DBSCAN* clustering algorithm to determine the overall BPM of a song based on the *BPM* values detected for each sample. This clustering algorithm was used since it handles noise and outliers in the audio data. This clustering algorithm forms clusters based on the data density, which helps to detect the most dominant cluster in the song. And, subsequently, to detect the song's BPM by calculating the mean of the BPM values of the most dominant cluster.

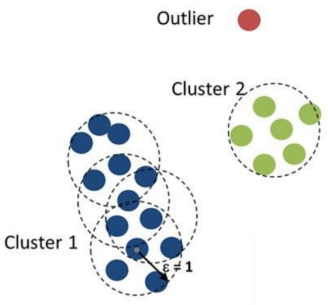

Fig. 3. *DBSCAN* algorithm. Divides the values into clusters and removes outliers.

Equation 4 is used to calculate the centroid of the cluster. The centroid value is the mean of all *BPM* values that belong to the cluster.

$$\text{cluster\_centroid\_C} = \frac{\sum_{i=1}^{n} \text{BPM}_i}{n} \tag{4}$$

7) **Detect song *tempo***: After detecting the song *BPM*, the song tempo was determine by a predefined list of *BPM* ranges.

TABLE I
Musical Tempo Ranges Based on BPM

Tempo Category	BPM Range
Very Slow (Below Largo)	0-39
Largo (Very Slow)	40-60
Adagio (Slow and Stately)	61-76
Andante (Walking Pace)	77-108
Moderato (Moderate)	109-120
Allegro (Fast and Lively)	121-156
Vivace (Lively and Fast)	157-176
Presto (Very Fast)	177-200
Prestissimo (Extremely Fast)	>201

B. Hybrid multi-band approach

In this approach, we decided to replicate the approach studied in *"Survey paper on music beat tracking "* by Makarand Velankar. In the hybrid multi-band approach, the incoming audio is splitted into three frequency ranges. In order to obtain the overall *BPM* of the uploaded song, this method took the following procedure:

1) **File conversion**: Firstly, we needed to convert the uploaded file using the *FFT*, similar to the initial approach.
2) **Bandpass filter**: Then, the signal was divided into three frequency ranges.
   - **Low-frequency range**: From 0 to 200 Hz (capturing percussive instruments).
   - **Medium-frequency range**: From 200 to 5000 Hz (capturing melody).
   - **High-frequency range**: Above 5000 Hz (capturing transients).
3) **Onset (*SC*) and Transient (*TD*) detection**: Detect onsets and transients in each frequency range to identify relevant rhythmic peaks.
   - **Low-band frequencies**: We used spectral-onset detections. This method was identified as a very suitable representation for tempo extraction as it tracks energy changes in the magnitude spectrum. It is effective in identifying slow onsets and rhythmic patterns.
   - **Mid-band frequencies**: For this frequency range, we used a transient detection method. This method counts the number of bins that represent localized energy spikes while ignoring lower-amplitude noises.
   - **High-band frequencies**: For this frequency range, we also used a transient detection method. So, even if the energies of the bins of a transient signal are low, this method will track a new occurrence if the transient spreads over this frequency range.

   The following formulas help to understand the detection of onsets (Equation 5) and transients (Equation 6 and Equation 7) on the signal.

$$ODF(t) = \max(\frac{\mathrm{d}x(t)}{\mathrm{d}t}) \quad \text{for} \quad t \in [0, \mathrm{T}] \tag{5}$$

where $x(n)$ is the amplitude of the signal at time n, $x(n-1)$ is the amplitude of the signal at the previous time step, and $\mathrm{T}(n)$ represents the magnitude of change (transient) between consecutive samples.

$$\mathrm{T}(n) = |x(n) - x(n-1)| \tag{6}$$

$$\text{Transient}(n) = \begin{cases} 1, & \text{if } T[n] > \text{Threshold}, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

where $x(n)$ is the amplitude of the signal at time n, $x(n-1)$ is the amplitude of the signal at the previous time step, and $\mathrm{T}(n)$ represents the magnitude of change (transient) between consecutive samples.

4) **Periodicity Detection method**: Succeeding the onset and transient detections, existing periodicities are calculated after getting the *PeDF* in each frequency range. This detection used an autocorrelation to identify rhythmic patterns.
5) **Combination of *PeDFs***: After having all the *PeDFs* for the different frequency bands, these ranges are combined into a single unified *PeDF* using weighted contributions, as we can see in Equation 8.

$$\begin{aligned} \text{Combined PeDF} = {} & w_{low} \cdot \text{PeDF}_{low} \quad + \\ & w_{med} \cdot \text{PeDF}_{med} \quad + \\ & w_{high} \cdot \text{PeDF}_{high} \end{aligned} \tag{8}$$

6) **Detection of peaks**: Finally, after calculating the combined *PeDF*, we needed to find relevant peaks (in our study we used the mean of the values of the combined *PeDF* to represent the heigh threshold for peak detection).
7) **BPM calculation**: Similar to the first method, if at least 2 peaks were detected, the *BPM* is calculated based on the average interval between peaks.
8) **Overall song BPM**: Finally, the *tempo* is defined based on the *BPM* obtained from the previous step.

TABLE II
PROPOSED HYBRID MULTI-BAND CONFIGURATION

Low Freq Band	Middle Freq Band	High Freq Band
SC	TD	TD

IV. PROGRAM SOLUTION EVALUATION

In order to evaluate our program solution, we evaluated the transparency, flexibility, reliability, performance and scalability.

A. Transparency

In order to measure the transparency of our solution, we used the *Python* library *Pylint* for the backend *API*, and the library *ESLint* for the frontend. For the backend score, the library *Pylint* was used to check the code quality of the system. The score obtained was 6.36 out of 10. For the frontend score, the library *ESLint* was used to analyse static code. The score obtained was

V. FUTURE WORK

In order to improve the BPM detection pipeline, I want to use the approach described in the paper "Survey paper on music beat tracking" by *Makarand Velankar* [1]. In this approach, they divided the frequency spectrum into 3 ranges (*Low-frequency band*, *Middle-frequency band* and *high frequency band*, while using onset and transient detectors to determine the song's *Periodicity Detection Function* (*PeDF*) and then get the overall song's BPM. This approach will allow for more precise and reliable beat tracking and tempo detection in songs.

VI. Conclusion

This project demonstrates the development and implementation of a sophisticated system that accurately determines the beats per minute (*BPM*) of the uploaded song.

The integration of the *BPM* overtime graph significantly enhanced the user interface, offering a dynamic and interactive experience.

In conclusion, the current state of this project meets the initial proposed plan. The future work described will be done until the end of the project development phase, and some user interface will be enhanced to improve user's experience on our system.

References

[1] M. Velankar, "Survey paper on music beat tracking," *http://www.ijrcct.org/index.php/ojs/article/view/373/pdf*, vol. 2, p. 953, 10 2013.