CMPE362

Project

Ahmet Fırat Gamsız – 2020400180

Fatih Demir – 2020400093

ALGORITHM 1

Selected Algorithm: Frequency selected sound energy

Reading Audio File and Preprocessing:

- The code starts by defining the list of song names and their actual BPM.
- Constants such as the FFT size (`N`), sample rate (`Fs`), number of frequency intervals (`num_intervals`), energy history buffer length (`energy_history_length`), and threshold factor for beat detection (`C`) are set.
- The C values are changed according to the song. (hyperparameter optimization)
- The subband frequencies are calculated logarithmically using `logspace`.
- Each song is processed one by one:
    - The audio file is read using `audioread`.
    - The real and imaginary parts of the signal are extracted from both channels.
    - FFT is applied to the complex signal.
    - Energy calculations are performed for each frequency interval.
    - Beat detection is done based on the calculated energy and threshold factor.
    - The results are displayed showing the actual and predicted BPM for each song.

Results:

Song: sevdacicegi.wav, Actual BPM: 114.5, Predicted BPM: 114.0
Song: dudu.wav, Actual BPM: 91.0, Predicted BPM: 88.0
Song: beat.wav, Actual BPM: 60.0, Predicted BPM: 60.0
Song: aleph.wav, Actual BPM: 81.0, Predicted BPM: 80.0

Predicted BPM for Given Songs:

- sevdacicegi.wav:
    - C value = 8
    - Actual BPM: 114.5
    - Predicted BPM: 114.0
- beat.wav:
    - C value = 370

        - Actual BPM: 60.0
        - Predicted BPM: 60.0
- aleph.wav:
    - C value = 122
    - Actual BPM: 81.0
    - Predicted BPM: 80.0
- dudu.wav:
    - C value = 12
    - Actual BPM: 91.0
    - Predicted BPM: 88.0

Explanation of Prediction Method:

- The code segments implement a beat detection algorithm based on frequency domain analysis.
- Each song is divided into frequency intervals using logarithmically spaced subbands.
- The energy of each interval is computed and compared to a threshold to detect beats.
- The threshold (C) is adjusted empirically for optimal beat detection.
- The algorithm identifies beats by analyzing the energy distribution across frequency intervals.

Performance:

- Both algorithms aim to predict BPM based on frequency domain analysis.
- The logarithmically spaced subbands approach provides more flexibility in capturing frequency variations.
- The performance of the algorithm may vary depending on the song type and characteristics.

Accuracy:

- The accuracy of the prediction depends on factors such as threshold selection, frequency interval distribution, and the inherent rhythm complexity of the song.
- Further experimentation and parameter tuning may improve prediction accuracy.

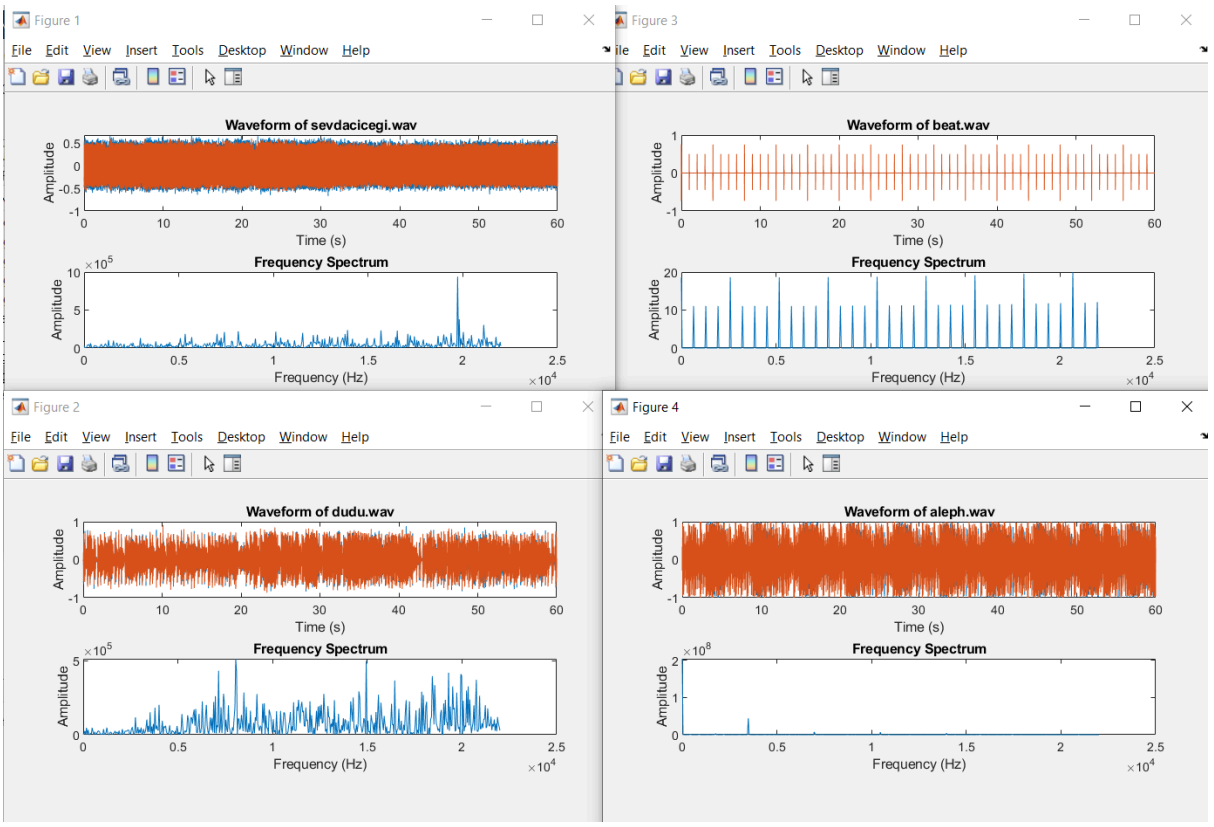
Conclusion:

Feedback:

- The code implements beat detection algorithms based on frequency domain analysis.
- The logarithmically spaced subbands approach enhances flexibility and adaptability to different song types.
- Experimenting with different threshold values (c) and subband distributions may further improve prediction accuracy.
- Robust testing with diverse song samples and rigorous evaluation can provide insights into algorithm performance and potential enhancements.

Potential Improvements:

- Explore dynamic threshold adjustment techniques for adaptive beat detection.
- Implement machine learning algorithms for pattern recognition to enhance BPM prediction accuracy.
- Optimize code efficiency for faster processing, especially for large audio files.



Algorithm 2

Frequency selected processing combfilters algorithm #1

1. Code Explanation

a. filterbank

The filterbank function divides a time-domain signal into separate frequency bands for beat detection. It first calculates the FFT of the input signal to transform it into the frequency domain. The function then determines the index range for each frequency band based on the provided bandlimits (logarithmically separated) and maxfreq. For each band, it isolates the corresponding frequency components by zeroing out frequencies outside the band range, ensuring symmetrical bands for real signals by including negative frequencies. Finally, it assembles the band-limited signals into the output matrix, with each column representing a different frequency band. This prepares the signal for further processing in the beat detection algorithm.

b. hwindow

The hwindow function rectifies a signal and convolves it with a half Hanning window to smooth its envelope. It starts by setting default values for winlength, bandlimits, and maxfreq if they are not provided, and calculates the Hanning window length. The function creates a half Hanning window and transforms the input signal from the frequency domain to the time domain using the inverse FFT. It then performs full-wave rectification on the time-domain signals to emphasize positive changes, and transforms them back to the frequency domain. The rectified signals are multiplied by the FFT of the half Hanning window, corresponding to convolution in the time domain. Finally, the smoothed signals are transformed back to the time domain, producing an output where each column represents a smoothed, frequency-banded signal, ready for further beat detection processing.

c. diffrect

The diffrect function processes a signal by differentiating it and then applying half-wave rectification to emphasize increases in sound amplitude, which correspond to beats. It first sets the default number of frequency bands (nbands) to 6 if not provided (but we normally provide 16) and initializes an output matrix of zeros. For each frequency band, the function calculates the difference between consecutive samples to highlight changes in amplitude. Positive differences, which indicate increases in amplitude, are retained while negative differences are discarded, effectively isolating the onset of beats. This prepares the signal for the final tempo analysis step by emphasizing sudden increases in sound intensity.

d. timecomb

The timecomb function determines the tempo of a musical signal divided into frequency bands. It takes in the signal (`sig`), beat resolution (`acc`), minimum and maximum beats per minute (`minbpm` and `maxbpm`), frequency band limits

(`bandlimits`), and maximum frequency (`maxfreq`). Defaults are set if these parameters are not provided. The function operates by iterating over a range of beats per minute (`bpm`) within the specified range and resolution, calculating the energy of the signal after convolution with a comb filter corresponding to the tempo. The comb filter is defined by a specified number of pulses (`npulses`). The tempo with the highest energy (`sbpm`) is chosen as the output, representing the estimated tempo of the musical signal. This final step completes the beat detection sequence by determining the fundamental tempo of the music.
Comb Filter Step Size: It helps determine the step size (nstep) for placing pulses in the comb filter, influencing the accuracy of tempo estimation.
Energy Calculation: It's indirectly involved in calculating the energy after convolution with the comb filter (e), which is crucial for evaluating different tempos and selecting the optimal one based on energy criteria.

e. freqselectedcombfilter

The freqselectedcombfilter function predicts the beats per minute (BPM) of a given song by implementing the Frequency selected processing combfilters algorithm. It takes in the name of a .wav file (`song`) and an optional parameter `maxfreq` used for beat-matching. If `maxfreq` is not provided, it defaults to 44100. The function reads the audio file, extracts a 5-second sample from the middle of the audio, and divides it into frequency bands logarithmically. It then applies a series of processing steps including filter bank, smoothing, differentiation and half wave rectifying, and tempo analysis with comb filters using the `filterbank`, `hwindow`, `diffrect`, and `timecomb` functions. These steps implement the beat detection algorithm described and recursively call `timecomb` with decreasing accuracy to speed up computation. Finally, the function outputs the predicted BPM of the song.

f. algorithm2.m

Algorithm2 just calls the freqselectedcombfilter which runs the algorithm and returns the BPM. For different songs change the song name or uncomment predetermined lines. You can also give a custom maxfreq but it is better to keep the default value 44100.

2. Results

Beat: 60.0 BPM (exact match)
Dudu: 91.0 BPM (exact match)
Sevda cicegi: 229.2 BPM (twice of the given value)
Aleph: 162.0 BPM (twice of the given value)

It creates different comb filters for all BPM values over a range and checks their similarity to the subbands we created. We take the one with biggest energy calculated by the given formula in the algorithm as the closest BPM.

I also implemented Derivation and Combfilter algorithm #1 algorithm but that algorithm never yielded correct results. Also, I tried running Pretty Please by Dua Lipa and get approximately 200 BPM which is not correct.

3. Comments

The algorithm sometimes gave the twice of the actual value for some songs. Accuracy might be increased by increasing the subband counts. Also the accuracy might be increased to detect other pop songs like Pretty Please which has more complex musical structures. If we can speed up the process we can make a fun app that instantly finds the BPM so you wouldn't have to search BPM of a song that you want to play with your instrument.

4. Resources

Algorithm was originally described here:
https://archive.gamedev.net/archive/reference/programming/features/beatdetection/page2.html

We used this website and given files at the end (filterbank.m, hwindow.m, diffrect.m, timecomb.m):
https://www.clear.rice.edu/elec301/Projects01/beat_sync/beatalgo.html

IMPORTANT NOTE: Since we used these files we didn't want to merge them to our code so we have multiple files for algorithm 2. Running algorithm2.m is enough to see the output.