The process

The process of recognizing a song takes a few steps to accomplish.
The fist step is creating the database, then we would need to use said database to actively recognize the song.

Step 1: Creating the database

Part 1 - Recording the audio

This is the first part of the process of creating the database.
In order to recognize a song we first need to record it.
To accomplish that I will use the raspberry pi with a USB microphone.
Once we have the recording we can start to analyze it.

Part 2 - Analyzing the recording

Now we have come to the tricky part, we need to analyze the sound we have just recorded.
This analyzation takes a few steps, which in the end will give us the "fingerprint" of the song, along with some other data we are going to need in order to completing the task at hand.
The way we are doing to do this is as such:
We are going to define two windows, a big one and a small one.
The big window defines the piece of song for which we generate a line-key, the small window defines the length at which we perform the calculations (such as FFT).

- definition of lengths

The first step into analyzing the recorded audio is to define the sizes of the windows that "run" on the recorded data.
We have two such windows and a sum of three lengths, the two lengths of the windows and the length at which the big window advances.

- W_σ - this is the big window at length $L_\sigma = 300_{ms}$
- W_S - this is the small window at length $\frac{2}{3}L_\sigma = 200_{ms}$
- L_δ - the rate at which the big window advances (hops) $= \frac{1}{3}L_\sigma = 100_{ms}$

This way we will generate an overall number of $\frac{L_A}{L_\sigma}$ line-keys for song (L_A - length of song A).

- generating PSDs

The next step is to divide the recorded signal to windows W_σ with length L_σ.
For each window, the first step is to calculate the PSD (power spectral density) using Welch's method.

- Welch's method: this method is used to estimate the PSD of a signal by slicing it to smaller pieces and on each piece (window), divide it into smaller overlapping pieces, for each smaller piece perform a window function for smoothing the edges, calculating it's FFT and the FFT's square amplitude - this is called a periodogram!

Now for our implementation of Welch's method:

- First we need to slice up the signal to windows (W_σ) with length L_σ. each such window houses a smaller window (W_S) with length $L_S = \frac{2}{3}L_\sigma$. We would have K overlapping windows (of the smaller kind).
- Now that we have determined the window sizes and all, we need to start performing the calculations. The first one is the Hamming window, essentially, for every sub window (W_S) we perform this window function, for smoothing the edges of the cut signal eliminating any discontinuous in the function.
- The next step, the main one for analyzing the signal, we perform the FFT, getting as a result the raw spectral components of the signal (the frequencies that make it).
- Finally, to get the Power of it we square the FFT results, This is the periodogram of the sub window (W_S).
- The final step (for reals this time) is to actually estimate the PSD. To do that we average on all periodograms. By averaging these multiple, slightly different views of the signal's spectrum, the random noise tends to cancel out, while the signal's true spectral components are reinforced. The result is a much smoother and more reliable PSD estimate.

- Mel filter bank

The next stage transforms the linear frequency spectrum into a more relevant representation. The linear Herz scale does not align with human pitch perception and so we use the Mel scale. To implement this we use an algorithm that applies a Mel filter bank to the computed power spectrum. This consists of a series of triangular filters that are narrow and dense at lower frequencies but become wider and more spread out at higher frequencies. This process effectively groups frequency bins and "blurs" the spectrum in a way that models the frequency resolution of the human ear. The center frequencies of the B (64) triangular filters are spaced linearly on the Mel scale, not the Hertz scale, which means we get an "evenly distributed Mel filter bank". Below is the specifications of said algorithm:

- Define frequency range: I have chosen a minimum (f_{min}) of $300Hz$ and a maximum (f_{max}) of $8000Hz$.

- Convert to Mel scale: the conversion f_{min} and f_{max} to the Mel scale is by the formula: $m = 2595 \cdot log_{10}(1 + \frac{f}{700})$.
- Create linear Mel points: we now need to create $B + 2$ points that linearly spaces between mel_{min} and mel_{max}. B is the number of filters (64).
- Convert back to Herz: the conversion back to the Herz scale is by the reverse formula: $f = 700 \cdot (10^{\frac{m}{2595}} - 1)$.
- Round the FFT bins: now we need to map these Herz values to the closest FFT bins indices using the formula: $bin = \lfloor (N_{FFT} + 1) \times \frac{f}{f_s} \rfloor$.
- Create triangular filters: For each filter m (from 1 to B), we define a triangular shape that starts at bin $m - 1$, peaks at bin m, and ends at bin $m + 1$.

An important note is that the bins that correspond to m and $m + 1$ are not adjasent and spread more and more as the frequencies get higher.
At the end of it, these B filters would represent the B bits in the resulted "linekey"!
Now we need to decide the polarity of said bit, see the next part for explanation.

- Binarization through adaptive exponential thresholding

After the previous stage, we have a list of B continuous energy values. The final stage of calculating the "linekey" is to determine for each value, with a threshold, if its a $'1'$ or a $'0'$., essentially converting this vector into a binary string.
This process involves several steps:

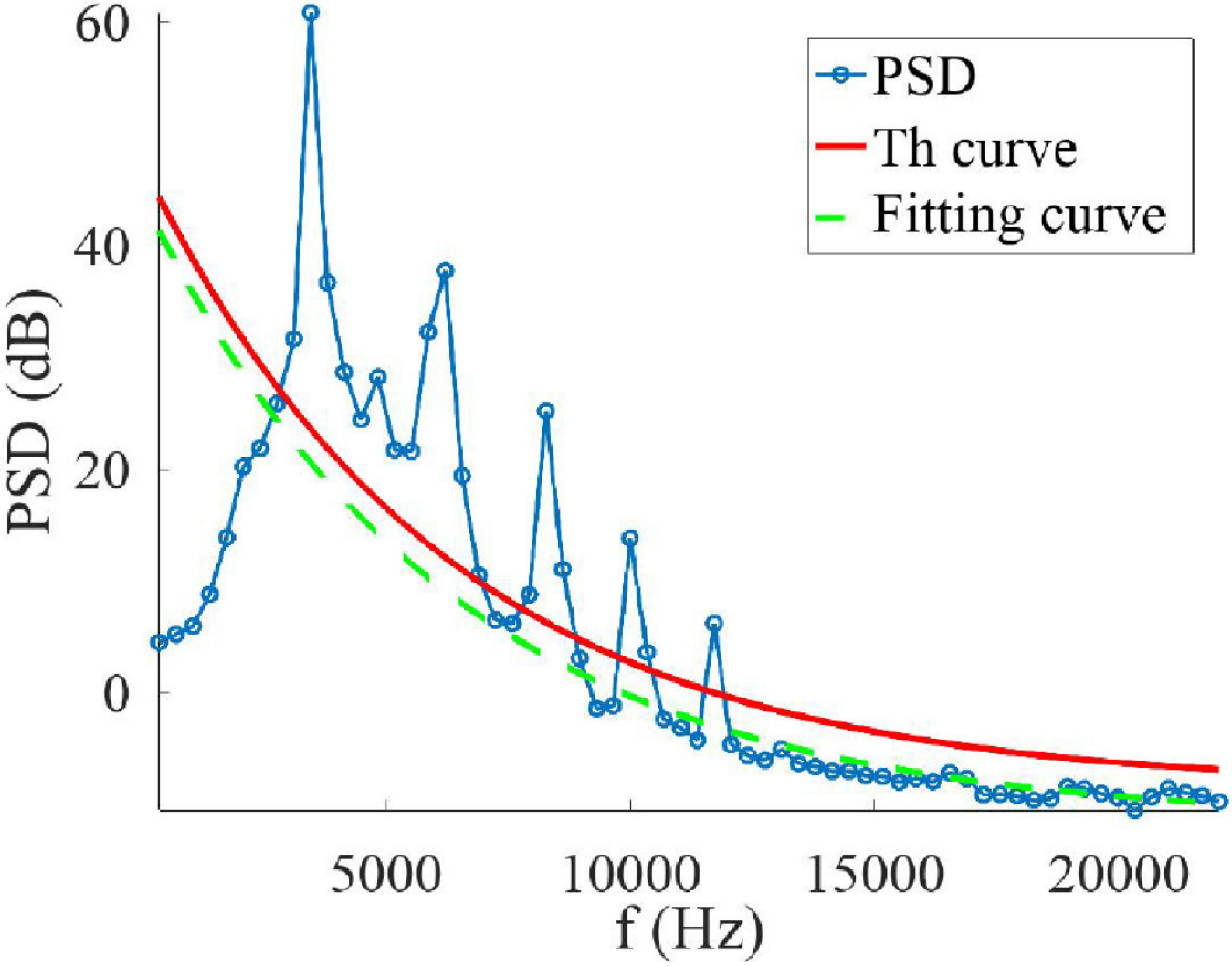
- The first step is preparing the data. In order to get a good representation of the human hearing, we convert the PSD values to be in the logarithmic decibel (dB) scalse by the formula: $P(t_n)[i] = 10log_{10} \sum_{k=1}^{K} \varepsilon_k(t_n)[i]$ where ε_k is the result of the Mel filter and $i \in 1, \ldots, B$.
- Now that we have the data prepared to the decibel scale we calculate the fitting curve.
  A direct non-linear fit to an exponential curve is computationally complex, and so the standard approach is to linearize the problem:

  - original exponential equation: $y = a \cdot e^{bx}$
  - taking the natural Log of both sides: $ln(y) = ln(a \cdot e^{bx})$
  - using logarithmic properties: $ln(y) = ln(a) + ln(e^{bx})$
  - finally the linear equation: $ln(y) = ln(a) + bx$

  From this we get to the equations presented in the paper for $a(t_n)$ and $b(t_n)$:
  $a(t_n) = \frac{\sum_{i=1}^{B} ln(P(t_n)[i]) \sum_{i=1}^{B} (i-1)^2 - \sum_{i=1}^{B} (i-1) ln(P(t_n)[i])}{B \sum_{i=1}^{B} (i-1)^2 - (\sum_{i=1}^{B} (i-1))^2}$.
  $b(t_n) = \frac{\sum_{i=1}^{B} (i-1) ln(P(t_n)[i]) - \sum_{i=1}^{B} ln(P(t_n)[i]) \sum_{i=1}^{B} (i-1)^2}{B \sum_{i=1}^{B} (i-1)^2 - (\sum_{i=1}^{B} (i-1))^2}$.

- Now that we have the fitting curve $y[i](t_n) = a(t_n) \cdot e^{b(t_n) \cdot i}$, $\forall i = 1, \dots B$, We add a margin m value to the curve $y[i]$ and derive the threshold values: $T(t_n)[i] = y[i] + m$, $\forall i = 1, \dots B$.
- To get the final binary string which is the "linekey" we apply the simple ruling of $'1'$ for values above the threshold and $'0'$ below: $l(t_n)[i] = \begin{cases} 1, & \text{if } P(t_n)[i] > T(t_n)[i] \\ 0, & \text{otherwise} \end{cases}, \quad \forall i = 1, \dots, B.$

attached is a representation of how such an exponential curve would look:



Now that we have the "linekeys" all we have to do is define the recognition algorithm, build the database and have fun :)

Database information

After the line-keys are produced, the songs are stored in the database as a list of said line-keys. In addition to that there is some more information stored, to be exact a list and a square matrix:

- L is a list of all the unique line-keys produced (for all songs in the database).
- C is a list that is defined by: for each L_i there is a C_i that is a list of couples (id, p) such that:
  - id is the index of the song in which this line-key (L_i) was found.

- p is the position in said song that the line-key was found in (idx in L_{id}).

That way all the information lies in three data structures:

- A list of line-key lists -> all the songs.
- A list of all the unique line-keys -> L.
- A list of lists each representing the information about each unique line-key -> C.

Identifying a song