DT2470 Lab 03: Shazam Wow!

by Bob L. T. Sturm

In this lab you will build and test a music fingerprinting system, similar to how Shazam does it – or at least as described in 2003: A. Wang, "An industrial strength audio search algorithm," in Proc. Int. Soc. Music Info. Retrieval, Oct. 2003. Our approach is illustrated below. First we will take an audio signal and compute its time-frequency representation, or sonogram. Then we will strategically find several points of interest in that representation. For each of those points we will construct tuples using other points of interest in the sonogram. These will become hashes that describe the fingerprint of the audio signal.

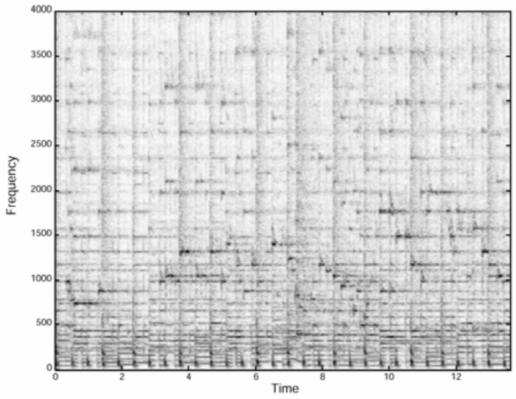


Fig. 1A - Spectrogram

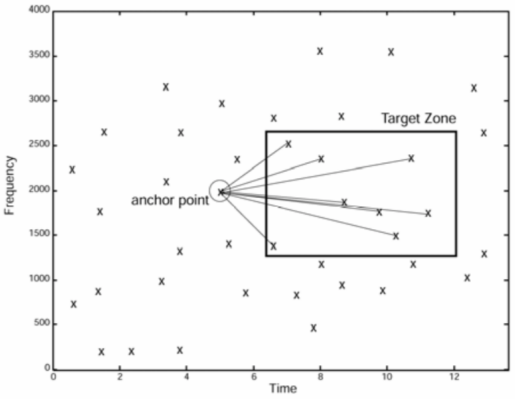


Fig. 1C - Combinatorial Hash Generation

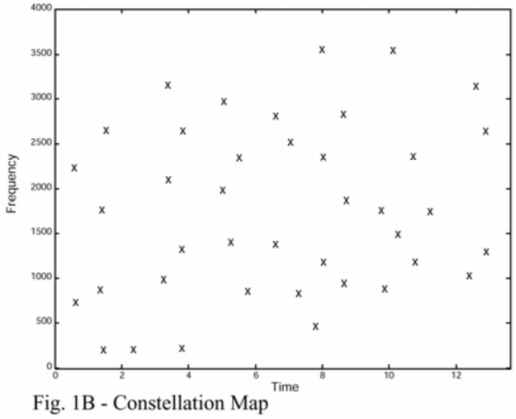


Fig. 1B - Constellation Map

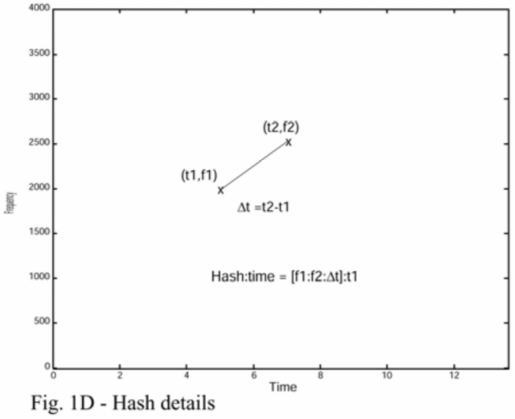


Fig. 1D - Hash details

In the first part, you will gradually build said fingerprint creation system.

In the second part, you will build an algorithm that compares fingerprints.

The lab report you submit should be a testament to your intelligence, as well as a reflection of your willingness to be a part of this module. You are free to use whatever software you want, e.g., MATLAB, Processing, etc. But below I use python. See the first lab and its solutions for assistance.

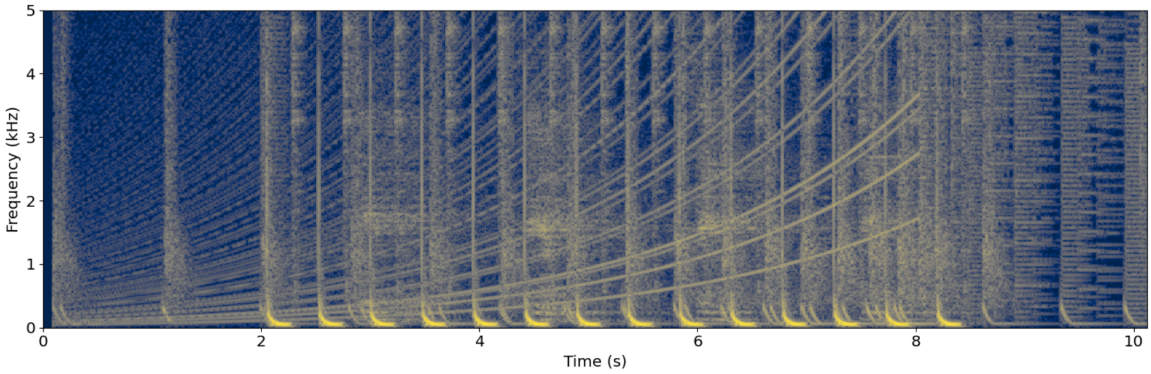
I also include some images so you can confirm whether you are on the right track, or just to have a brief pause to laugh at how far your answer is from being correct.

Part 1: Creating audio fingerprints

1.1

Pick one of the query sound files and compute its dB magnitude short-time Fourier transform using a Hann window of duration 50 ms with a window hopsize of 10 ms, and zeropadding to four times the length of the window. Plot said sonogram, and appropriately label your axes with "Frequency (kHz)" and "Time (s)". The frequency axis must be in kiloHertz, and limited 0 to 5000 Hz. The time axis must be in seconds. Choose a colormap that you feel describes your personality (https://matplotlib.org/3.1.1/tutorials/colors/colormaps.html (https://matplotlib.org/3.1.1/tutorials/colors/colormaps.html)).

Here's what mine looks like:



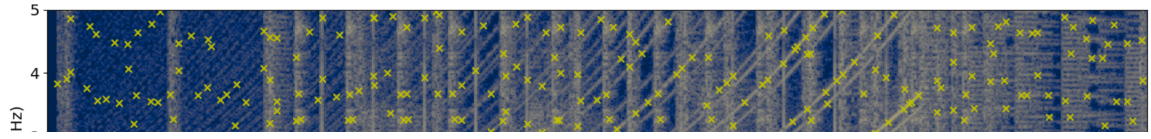
In [ ]:

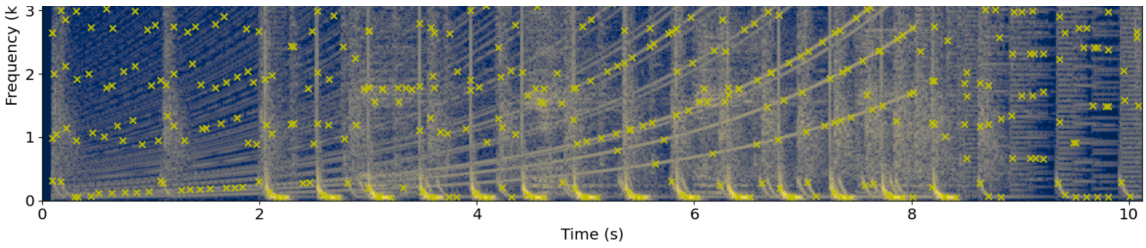
1.2

We are now going to locate in this sonogram "anchors", or points of interest. We will take a rather naive approach: for each continguous ΔTxΔF box, find the time and frequency with the largest energy. Let's make ΔT=100 ms, and ΔF be such that it divides the whole spectrum $[0,F_s/2]$ into 25 bands. For each one of these segments, locate the time and frequency of the largest value, as well as its dB. Do this for the sonogram you created in the last question.

In [ ]:

3. With the anchors you found in the previous question, superimpose them on the sonogram. Write something intelligent about the results. My figure appears below, with each anchor marked by an "x".





In [ ]:

1.3

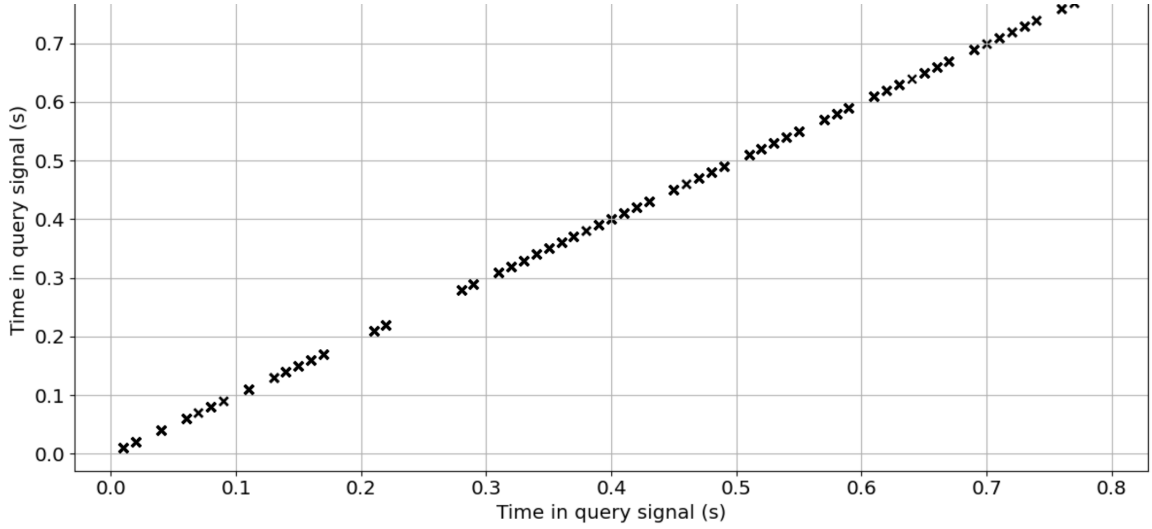
Now it's time to compute hashes from a set of anchors. A *hash* is a tuple describing an anchor at time (t_1) and its relationship to another anchor at time (t_2). A Shazam hash is the tuple $(f_1,f_2,Δt_{12})$, which contains the frequencies of both anchors (f_1,f_2), and the difference in their time positions ($Δt_{12} = t_2-t_1$). The Shazam fingerprint of an audio signal is then the set of hashes extracted from it as well as times at which each hash appears. To create these hashes, Shazam does not look at all pairs of anchors. For a given anchor, it computes hashes using anchors in a "target zone". Define the target zone of an anchor as a time-frequency region 100 ms after t_1, spanning 500 ms, and spanning frequencies one half-octave above and below f_1. For instance, for an anchor with t_1 and f_1, its target zone is located between t_1+100 ms and $t_1+100+500$ ms, and spans a frequency range of $f_12^{-0.5}$ and $f_12^{0.5}$. Compute all the hashes for your choosen query signal, in the form $(f_1,f_2,Δt_{12})$. Your final data structure representing a fingerprint F should be something like a sequence of times paired with hashes $F = ((\tau_1, h_1), (\tau_2, h_2), \ldots)$, where τ_1 is the time at which hash h_1 occurs, etc.

In [ ]:

Part 2: Comparing fingerprints

2.1

Now it's time to create an algorithm for comparing two fingerprints. We will start simple to make sure things work. Extract any 1-second segment of from the audiofile you worked with above and compute its fingerprint F_q as above. ("q" is for "query".) Now for each hash in F_q, find all matching hashes in F_q, and store the times of the matches. To compare two hashes, compute the sum of the magnitude differences between the dimensions of the hashes. For instance, if I have these hashes, (a,b,c) and (d,e,f), I just compute the following value: $d = |(a-d)|+|(b-e)|+|(c-f)|$. If $d=0$ then it's a match! Create a scatter plot of the times of the matching hashes. Write something about what you see, and why you see it. Not why you see it, but *why* you see the properties you see in this particular plot. My plot looks like this:



In [ ]:

In [ ]:

2.2

Now compare F_q with F_d, where F_d is the fingerprint of the original audiofile. ("d" is for "document") Again make a scatter plot of the times of the matches. As above, write something about the results.

In [ ]:

2.3

Finally compare F_q with F_d, where F_d is the fingerprint of a *different* audiofile. Again make a scatter plot of the times of the matches. As above, write something about the results, and in particular compare with the plot produced in the last parts.

In [ ]:

We have not a single matching point! This is NOT the document.

*2.4

Here's a project idea: Write a program to locate the files in the kiki-bouba collection from which all these queries come.

In [ ]: