Audio fingerprinting

Escola Tècnica Superior d'Enginyeria

Universitat Rovira i Virgili

Student: **Eduard Bel**

Table of Contents

Structure of the project

The project is structured in the following way:

Project_folder:

- library/
  - Contains all the songs of the moodle's music library (parts 1 and 2)
- samples/
  - clean_samples/
    - Contains the clean samples.
  - filtered_samples/
    - Contains the filtered samples.
  - noisy_filtered_samples/
    - Contains the noisy filtered samples.
  - noisy_samples/
    - Contains the noisy samples.
- benchmark.sh: a script that iterates through all the samples and returns the precision rate for all differently distorted samples (also the global precision).
- builddb.py
- fingerprint.py: contains the fingerprinting functionalities, used in both main programs.
- identify.py

User manual

The dependencies of the system are:

- sys: to obtain the input arguments from the terminal.
- os: to remove the database file if it already exists. To search for songs in the directory.
- scipy: to read the .wav files.
- sqlite3: to handle the database.
- warnings: to avoid printing a warning when reading some songs (the package can't handle metadata).
- time: to keep track of the execution time.
- matplotlib: to obtain the spectrogram.
- numpy: for data analysis.
- hashlib: to perform the hashes for the fingerprint.

The execution should be done in the following way:

- `$ python3 builddb.py -i songs_folder -o database_file`
- `$ python3 identify.py -d database_file -i song_name`

To perform the benchmark you should first gain execution permissions:

- `$ chmod u+x benchmark.sh`

To execute it, the folders should be arranged as specified in the User manual section.

- `$ ./benchmark.sh`

Design of the utilities

The programming language of choice has been python, since I am used to it and found all the libraries I could possibly need for the project development. I have also built a bash script to perform the tests because it is the fastest and easiest way to do so. The code is also highly commented.

Parameters choice

When the code was working and I was able to perform the test on the whole sample files, I started tweaking some parameters with the aim of reducing incorrect identifications while keeping the identification process in a low running time. The final set of tweakable parameters is:

- WINDOW_SIZE: 4096. Window size for the Fourier Transform, should be a power of 2.
- TARGET_ZONE: 8. The number of constellation points in the target zone for an anchor point. It could be increased, but the number of correct identifications doesn't increase that much. The size of the database and the time of recognition also increase.
- BLOCK_SIZE: 30. We will iterate through the spectrogram in blocks of 30x30 and get the highest amplitude point of the block.

I have selected these parameter values after comparing the single identification time and the identification precision using the bash script.

Database schemas

The database used is the one we can create with the package *sqlite3*. It holds all different hashes returned by the fingerprint generation. The structure of the songs table is:

- hash: TEXT
- offset: REAL
- song_name: TEXT

When we want to store the hashes of a song's fingerprint, we do it in batch. This is to reduce the overhead that we have if we do single additions. The time difference of database creation with arbitrary parameters between adding single items or adding them in batch is approximately 2 minutes.

Finally, when we have generated the fingerprint for the sample song and want to obtain the hash matches from the database, we also do it in batch. The identification time is also notoriously reduced.

Design choices

Since the followed steps for the algorithm are those from the Shazam paper, I won't dwell on that.

When identifying the samples, I found that I had no way of correlating an offset (from the sample) with a match hash retrieved from the database. I decided to use a dictionary hash-offset to easily set the relation again and prepare the final identification step.

Other decisions were some already mentioned above, doing batch insertions and batch recollection in the database and, creating a file for the fingerprinting process.

For the builddb and identify programs I used a function to check if the input parameters are given in the correct format.

Detection metrics

The results obtained after the execution of the benchmark script using the parameters specified in the parameters choice section are the following:

Set	Correct	Incorrect	Precision
Clean	200	0	100%
Filtered	200	0	100%
Noisy filtered	181	19	90.5%
Noisy	192	8	96%
Total	**773**	**27**	**96.625%**

The execution time for the creation of the database is: 46.92 seconds.
The execution time for a song identification is around 2 seconds.