IMPLEMENTING MUSIC RECOGNITION AND ACOUSTIC FINGERPRINT ALGORITHMS

Kerui Hu
V00851960
kevinqd@me.com

Yiming Sun
V00811496
yimings@uvic.ca

Yajun Wan
V00895974
yajunwan@uvic.ca

ABSTRACT

In this paper, discussions about how acoustic finger algorithm effectively identify an audio sample or quickly locate similar items in an audio database will be presented, then based on existing applications and researches that are relevant to this topic, we developed an app to evaluate the operationality of implementing the fingerprint algorithm from Shazam. In order to evaluate the performance of the model, we adjust the parameters of the input from three aspects: 1) sample length, 2) artificial noise and 3) recording sample in virtual surroundings. And after testing our model under those three circumstances, we concluded that the efficiency and stability of the algorithm is proved.

1. Introduction

Music has become an indispensable part of lives now,it is regarded as a tool to express feelings, to communicate with others. Imagining a scenario that you are attracted by the melody played around, however your phone can't find the entire piece of the song without a given name, therefore, the need of query by vocal or humming is growing. The process of querying a melody can be roughly separated into two parts: extracting the feature of the clip then matching it with the sample from database and in section 2, concrete details will be discussed. Due to the rising amount of audio database, people need to come up with optimized searching algorithms to raise the efficiency of matching and sorting a song, therefore we will develop an application that based on the fingerprinting algorithm that introduced by Shazam, and the general process will be discussed in section 3. And for section 6, we tend to provide a intuitional evaluation of the performances, by 1) adjusting the sample length, 2) adding artificial noise then 3) adding nature noise.

Time Slot	Abstract Goals (to be refined)
Week 1	• Studying required documents and extracting key info • Group discussions
	(credibility of references)
Week 2	• Problem domain analysis and requirements gathering
Week 3	• Architecture design and modelling: Behavioural (external) modelling (use cases)
Week 4	• Architecture design and modelling: Structural (internal) modelling (data flow diagram)
Week 5	• Reimplementation in Python
Week 6	• Validation and presentation

Table 1.Objectives and Timeline

1.1 Prior and Related Work

Fingerprint -- We noticed that the audio fingerprinting algorithm is now the most efficient method to achieve the aim to recognize music. The fingerprint of audio is a kind of short representation of the audio object. This means that a fingerprint function F should map an audio object X, consisting of a large number of bits, to a fingerprint of only a specified number of bits. This allows speeding up the comparison process between two different songs. Instead of comparing all the bits of both songs we can just compare their hash functions in order to identify if the songs are similar or not.

locality-sensitive hashing -- In our project we may also use the locality-sensitive hashing. In computer science, locality-sensitive hashing (LSH) is an algorithmic technique that hashes similar input items into the same "buckets" with high probability.[1] (The number of buckets are much smaller than the universe of possible input items.)[1] Since similar items end up in the same buckets, this technique can be used for data clustering and nearest neighbor search. It differs from conventional

hashing techniques in that hash collisions are maximized, not minimized. Alternatively, the technique can be seen as a way to reduce the dimensionality of high-dimensional data; high-dimensional input items can be reduced to low-dimensional versions while preserving relative distances between items.

1.2 Tools and Datasets

We will initially use Python as our programming language for the implementations, and the outcomes will be presented in Python notebooks, because we have been learning the MIR course with Python, and it comes with a large range of libraries for audio processing and presenting. However, we anticipate that Python has significant disadvantages on runtime when processing large datasets, so we might need to reimplement in Java following the Python code for better performances.

We will mainly use phones to record (resample) music played by speakers, in order to add nature noise and distortions, as well as to clip the piece of music. We can also do such edits by manipulating audio files with Python Numpy array operations. There are some handy online tools such as AudioTrimmer. (There will be more listed for doing more complicated transforms like speed and pitch modifications.)

Since our primary objective is implementing the recognition and fingerprinting algorithms, we will not need to query in a database of millions of songs, and there are copyright limitations. Instead, we compute the scores of similarities in a one-to-one or one-to-several setting. We will download audio files as dataset from Youtube Audio Library[7], which has numerous licensed songs classified by genre, mood, instrument and duration.

2. Evolution of Querying

In this section, we explore the history of mapping songs with the samples in the database and the process of optimizing algorithms.

2.1 Extracting features of audio

The definition of extracting features of audio is highlighting the most dominating characteristics of the audio. As shown in Figure 1 below, In the late 1950s, the simplest way to recognize the piece of the audio is to extract information under time domain, to be

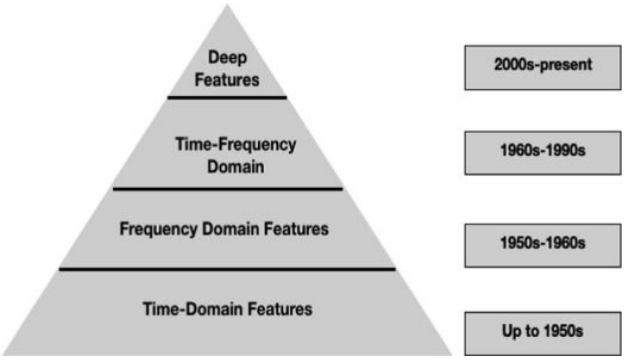

Figure 1.Evolution of Audio Feature Extraction

more specifically, the pitch of the audio needs to be detected constantly and separately due to the instantaneity of the pitches. Therefore, the method of first dividing the original audio signal into small windows then use zero-crossing algorithm to determine the cycles evolves

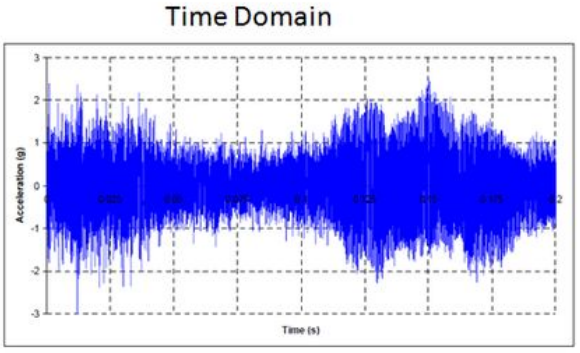

Figure 2(a) Signal shown in time domain

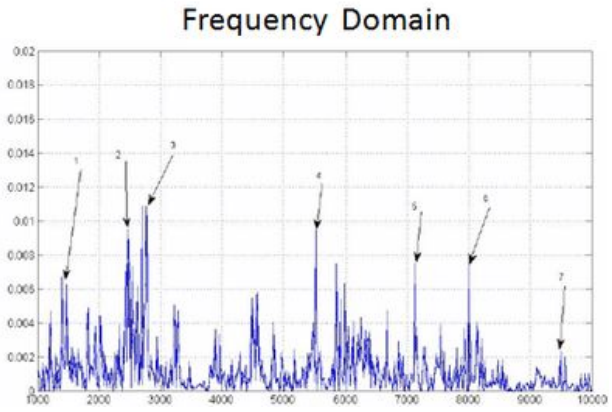

Figure 2(b) Signal shown in frequency domain

up to solve the problem above. However, when querying by humming, the accuracy of pitch detection may not be ideal due to the existence of noise or the unstable volume. The evolution of quantizing signals in frequency domain has optimized the performance of identifying the components of the original audio signal, for example,we can tell that after applying Fourier Transform, the specific features of the original signal shown in Figure 2(b) has been apart from the noise in low frequencies. Therefore, the “landmark” fingerprinting algorithm has been evolved based on the result of transforming from time domain to frequency domain. The audio fingerprint matching results

between two channels are visualized in Fig. 3 [3], where four examples of matched audio landmarks in a short segment. The detailed implementation will be discussed in Section 3.

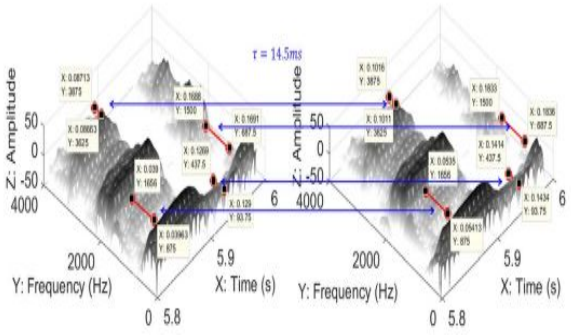

Figure 3. Visualization of audio fingerprint matching between two audio channels with a time offset of 0.0145s . Four examples of matched landmarks are drawn on the corresponding spectrograms. Each pair of matched landmarks shows a time delay of 0.0145s.

There are two main reasons that make the fingerprinting algorithm to be more reliable than the previous method:

I. **The pitches are proved to be unique and identical to the audio.** The peaks of the signal represent the most typical information of the audio, when we listen to a segment of the melody, we not only remember the pitches nor the rhythms but also the timbres. The fingerprinting algorithm tend to find the most representative peaks for each period of the audio, therefore, when comparing the sample piece with the desired audio by the consistency of the peaks from the same specific period, if the accuracy is relatively high, the two audios are considered to be the matching pairs.

II. **High resistance on background noise.** Because the spectrogram prominent peaks that this algorithm chose to form the landmark pairs are most likely to survive the noise and distortions [3]. The details will be discussed in section 3.

2.2 Mapping sample audio

As mentioned before, fingerprinting algorithm will first need to extract key values from by selecting points with peak amplitudes. After that, each highlighted points will be regarded as anchor points[4] and will be allocated with a target zone[4], the reference retrieval database will be generated by recording the frequencies for both anchor points, points in target zones as well as the time difference between the two points. More details will be discussed in Section 3 below.

3. Algorithm Explanation

During our research on the algorithms to achieve our goals, we found that there are several methods might be helpful on our project.

The very first we are going to discuss is the algorithm that really cut the edge in today's audio search field, the audio fingerprinting algorithm. Similar to our human fingerprints, a piece of audio also has its own fingerprint. The main idea is to find the peaks after we do Short Fourier Transformation to the clip of music under testing.

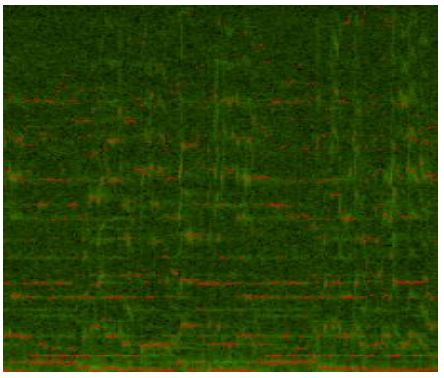

Figure 4. Array visualized and colored[6]

As shown in Figure 4 above, those highlighted red dots are selected to be the fingerprints points. Those are the peak energy of the spectrum which represents the frequencies with the highest amplitude, the reason to choose this group of points to generate the constellation map in Figure 5 below is that they are said to hold the most probability to survive the noise distortions.

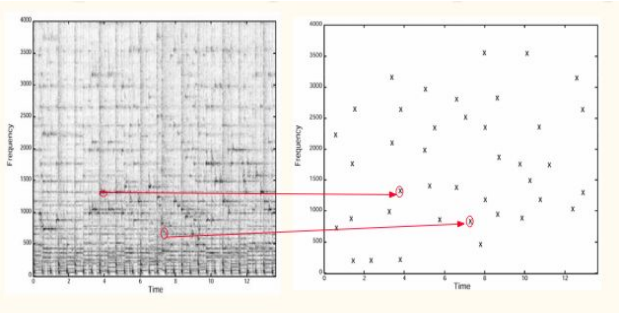

Figure 5. Spectrogram and Constellation Map [4]

Figure 5 above shows how to transfer from spectrogram to Constellation Map[4] on the right part, it is simply by maintaining the points with more powerful frequencies. This is not the final look for fingerprint database, and in the following part, we will discuss how to define and store the parameters of fingerprint pairs.

3.1 Anchor Point and Target Zone

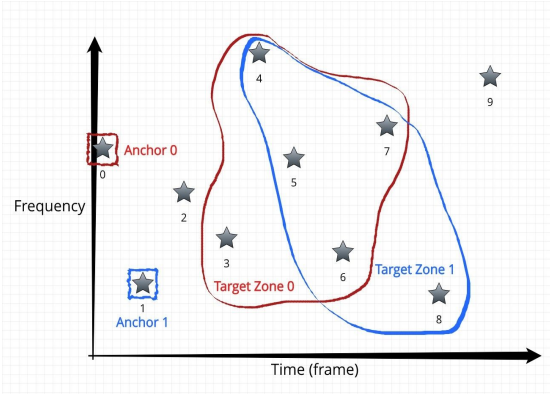

Figure 6. Visualized anchor point and target zone

Based on the Constellation Map we get and set the anchor distance(>5) and fan-out factor(>10). We start to do the following steps:

1. Set the first point in the constellation map as the anchor point and find points which have a distance greater or equal to the anchor distance away from the anchor point on the positive direction of the x-axis. The number of the points we need is equal to the fan-out factor we set in advance.
2. We take each point in the target zone in order and set a 3-element tuple, save the anchor point frequency, the frequency of the point we took, and the difference of the time of these two points. The tuple has the example as (F anchor, F target1, delta-T). Save the tuple into the list. Repeat step 2 until all the points in the target zone has been retrieved.
3. Find the next point in the constellation map as the new anchor point, repeat the steps above until all the points have been retrieved.

After we finished the steps above, we will have a list with all the tuples saved inside. And that is the fingerprint of the audio.

3.2 Matching Between Fingerprints and Scoring

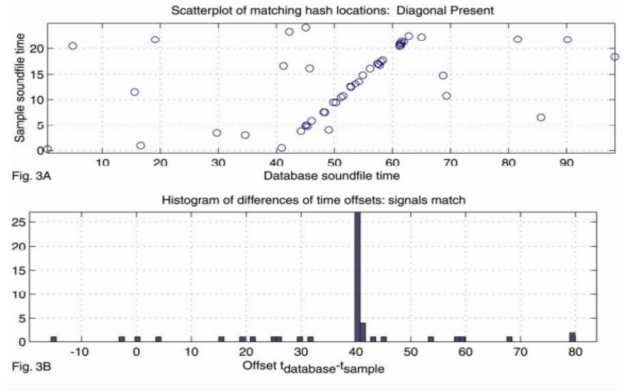

Figure 7. Scatterplot of matching has locations and histogram of differences of time

Since we have the fingerprint database and the fingerprint of the clip of the target audio, we need to match with each fingerprint in the database and score each match. We collect the same tuples and plot them into the scatterplot. We can see from the Figure 7 that some points arranged in a straight line, that means we got a matched clip. Mathematically, we prove this matched clip by using the differences of time offsets. We calculate the difference of x-axis and y-axis of each point and plot the result of the number of each difference into a histogram as we see in the Figure7. And the score of each match is the number of times that the difference of offset occurs most.

4. Expected Results (Inherited from Progress Report)

From what we have discussed in Section 2 and 3, at this point, we have explored the evaluation of music feature extraction, and it provided the essential background knowledge and design requirements of the project; then, we have decided to focus on processing the frequency domain features, and found the ideally optimal algorithms for fingerprint generation and hashing.

Our primary objective so far is implementing the basic algorithms in Python, which will take about a week. In the meanwhile, we will note down the possible alternatives (e.g. having different number of frequency ranges that the points with the highest magnitudes are taken from). The demo program will be tested with a small dataset will about 100 songs, followed by an evaluation process. We are inspired by the evaluation metrics introduced in Haitsma and Kalker's (2002) paper, which are Robustness (can an audio clip still be identified after severe signal degradation?), Reliability (how often is a song incorrectly identified?), Fingerprint size (how much storage is needed for a fingerprint?), Granularity (how many seconds of audio is needed to identify an audio clip?) and Search speed & scalability (how long does it take to find a fingerprint in a fingerprint database?) [1]. Typically for Robustness, as mentioned before, we would like to compare the performances by querying with original clips, clips resampled by a microphone, and recordings of humming.

After completing the primary objective, we would do some extended works with time permission: one option is re-implementation in Java or C for performance increase, and evaluate with a significantly larger dataset (if access is allowed); another option is trying the design alternatives we noted, even adding new features beside the spectrum peaks in the frequency ranges, and discussing why the new designs are better or worse in terms of the evaluation metrics.

5. Implementation Approaches

The final implementation is solely written in Python, because the program already has a satisfying performance in terms of runtimes. We used Sqlite 3 library for maintaining the database. The database has tables as follows:

1. **Parameters** is used to store the parameters sample rate, window size, anchor distance and fan-out factor when fingerprinting the dataset. The purpose is ensuring the same parameters are used when fingerprinting the samples.
2. **Songs** has the filename in the dataset, and a unique identification for each song (tinyint, maximum 256 songs). The purpose is to reduce the size of the fingerprints.
3. **Fingerprints** has the identification, the time (frame number) of the anchor point (smallint, maximum 32768 frames in a song), and the fingerprint (int and indexed, explained below).

Recall that each fingerprint is a tuple of three: frequency of the anchor point, frequency of the target point, and their time (frame) difference. In this implementation, the frequencies are divided into 1024 bins of 10 Hz, and the maximum frequency is 10230 Hz (frequencies above are trimmed to the maximum). Therefore, the three-tuple can be hashed into 30 bits (10 bits for each element), and fit into a 32-bit int. Necessarily, each integer value of fingerprint is mapped into an index, in order to let each search be done in O(1).

In the query, the fingerprints of the song are matched against the fingerprints of all the songs in the database, rather than against one by one. A map from song identifications to lists of time offsets is maintained during the query.

Overall, the time complexities of fingerprinting and query are both O(n). The compression ratio (size of dataset to size of database) is 4% with the default parameters (example in 6.1).

6. Experimental Results

6.1 Dataset and parameters

We build a sample audio database and fingerprint database test our algorithm on its accuracy, performance of anti-noise and find out any potential problems.

Audio Database: The audio of our database are from Youtube Audio Library [7]. We downloaded 99 songs in classical genre, the size was 402.3Mb.

Parameters: sample rate: 22050, window size: 4096, anchor distance: 5,fan-out factor: 10

Fingerprint Database: The size of fingerprint database is 17 Mb, which is 4.23% of the original audio database. To generate the fingerprint database, our algorithm cost 457 seconds.

Sample: We use the way of control variables to test the effectiveness of this algorithm. We intercepted the segments from the same song and the same starting time. Since our algorithm is based on the amplitude peaks in the audio, so we chose the intro part of the song(00:05-00:15). The reason is that in the intro part, there is not much heavy notes, and in this case we can test the algorithm under an extreme conditions.

6.2 Effects of Sample Length

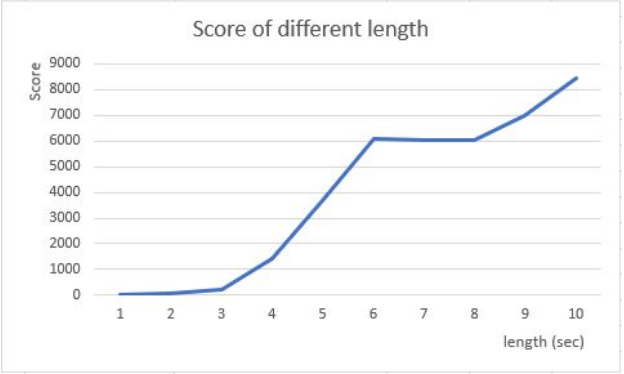

Figure 8. Line chart of Score of difference length

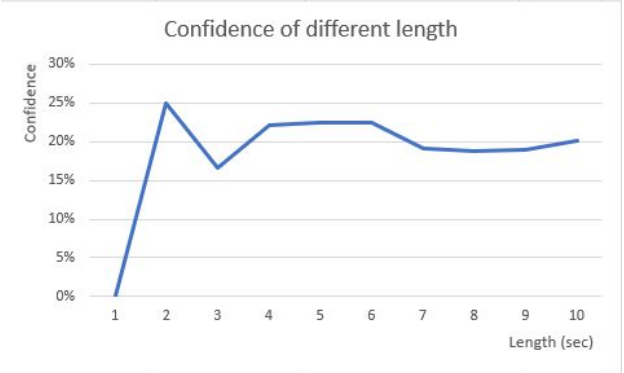

Figure 9. Line chart of Confidence of difference length

We used the same audio to match with the same database. We intercept the segment from the 00:05 of the audio and each time we extend the length of 1 second and test for 10 times. Except the 0 second segment, we got the expected matching results. The score is on the uptrend with the maximum of 8453. The Confidence (score in overall result percentage) hovering between 16% to 25%. The running time of this algorithm is O(1), so it rise from 0.04 to 0.22.

6.3 Effects of Artificial Noise

We also test for the audio with artificial noise. We generate the noise based on normal(Gaussian) distribution with the ratio of standard deviation of the

maximum amplitude of the audio. Mixed the noise with the 10 second segment together and test for the results.

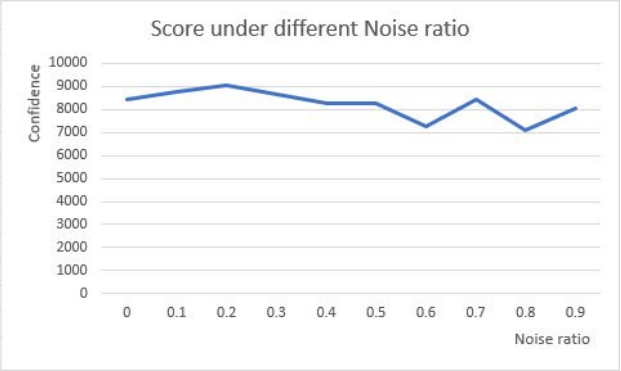

Figure 10. Line chart of Score under difference Noise ratio

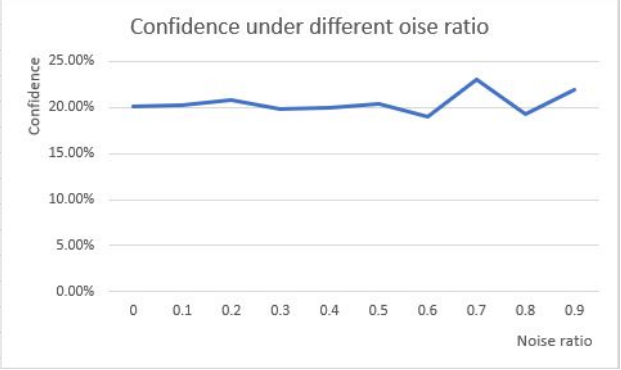

Figure 11. Line chart of Confidence under difference Noise ratio

We got accurate matching results for all 10 matchings. The artificial noise does not work as expected in this algorithm. Since we use the ratio no more than 0.9 so the noise would not cover the amplitude peaks of the original audio. As a result, the noise would influence the final results. But this test still shows the anti noise performance of this algorithm.

6.3 Effects of Nature Noise

We record the same audio in different environments to test the performance of anti-noise of this algorithm. The results are shown in the table:

Environment	Score	Confidence	Note
Silent room	3926	15.88%	Correct
Mixed with disco	5177	21.55%	Correct
Running car	553	5.92%	False, at rank 6, scored 31.62% of top result
Restaurant	369	4.25%	False, at rank 7, scored 53.95% of top result

Table 2. Evaluation under different circumstances

The record in the silent room does not contain much noise, the result only shows the algorithm works for the phone recording qualification. And then we mixed the audio with the disco audio, and it still work means that if the amplitude of the noise is not higher than most of the amplitude peaks of the original audio, then the algorithm still works. But in the environments of running car and restaurant, there are multiple noises contained, the performance of anti-noise is not excellent. In the running car environment, there are engine sound, background music and the raining noise, the score and confidence is significant lower than in a silent room. But the correct matching will still occur our top 10 recommendations(at the rank 6). And in the restaurant record, there are background music, people's talking and some other noise contained, the result is almost the same as in the running car(at the rank 7). In these situations, the amplitude and volume of the noise covered those of the original audio, and it will influence the result of this algorithm. But generally, the algorithm can still match with the desired audio and the rank will be in top10. We will keep finding ways to solve these issues.

7. Conclusion

First, our implementation tends to maintain high compression ratio by transforming default audio database to fingerprint database, therefore, the retrieving speed will be improved by that. Second, it only requires a short-length sample audio to find the original piece of the song. And also, the result shows that our model proved to have a certain degree of anti-noise ability however it doesn't reach a desired accuracy when testing under noisy circumstances. Overall, the performance of this implementation is stable.

8. Reference

[1] Haitsma, Jaap & Kalker, Ton. 2002. A Highly Robust Audio Fingerprinting System. Proc Int Symp Music Info Retrieval. 32.

[2] Cheng Yang, "MACS: music audio characteristic sequence indexing for similarity retrieval," *Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No.01TH8575)*, New Platz, NY, USA, 2001, pp. 123-126. doi: 10.1109/ASPAA.2001.969558

[3] T. Hon, L. Wang, J. D. Reiss and A. Cavallaro, "Audio Fingerprinting for Multi-Device Self-Localization," in *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 10, pp. 1623-1636, Oct. 2015.

doi: 10.1109/TASLP.2015.2442417

[4] Christophe, " How does Shazam work". Aug.6, 2015. [online]. Available:http://coding-geek.com/how-shazam-works/. [Accessed on Nov.30,2019]

[5] Avery Li-Chun Wang, "An Industrial-Strength Audio Search Algorithm". [online]. Available:http://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf. [Accessed on Dec.2, 2019]

[6] Roy van Rijn, "Creating Shazam in Java". June.1, 2010. [online]. Available:https://royvanrijn.com/blog/2010/06/creating-shazam-in-java/. [Accessed on Nov.13,2019]

[7] Youtube Audio Library. [online]. Available: https://www.youtube.com/audiolibrary/music [Accessed on Dec.9,2019]