Oxford High School

A Music Identification Algorithm which Utilizes the Fourier Analysis of Audio Signals

Joseph Edmond DeChicchis

March 25, 2016

Candidate Number: 006832-0004

Table of Contents

Introduction

The goal of this investigation is to create a small scale system for identifying music through the mathematical analysis of audio signals. Currently there are several smartphone applications which identity music such as Shazam, SoundHound, and Musixmatch. The process Shazam uses to identify a song is outlined in a paper by one of the founders of Shazam Entertainment, Ltd.[1] Although this paper was instrumental in setting the path for this investigation, there was not much detail as to how Shazam implemented a robust music identification system because Shazam's technology is proprietary. Thus, the music identification algorithm had to be designed from scratch.

In particular, the music identification algorithm consists of the following parts:

1. Analyzing audio signals using the Fourier Transform.
2. "Sequencing" music by identifying key frequencies.
3. Calculating a unique Hash-Time vector for each song.
4. Assembling these Hash-Time vector files into a database.
5. Running steps 1 through 3 on a sample audio signal.
6. Matching the sample audio Hash-Time vector to the correct song from the database.

Throughout this investigation "audio signal" will refer to any audio data, whether it be a music file or live recording from a microphone.

The Fourier Transform

The Fourier Transform is a method of converting a function in the time domain into a function in the frequency domain. This is commonly written as

$$F(v) = \int_{-\infty}^{\infty} f(t)e^{-2\pi ivt}dt$$

where $F(v)$ is a function in the frequency domain, $f(t)$ is a function in the time domain, v is frequency and t is time.[2] On the other hand, the Inverse Fourier Transform is a method of converting a function in the frequency domain into a function in the time domain. This is commonly written as

$$f(t) = \int_{-\infty}^{\infty} F(v)e^{2\pi ivt}dv$$.[3]

[1] Avery Li-Chun Wang, "An Industrial-Strength Audio Search Algorithm,", accessed March 13, 2016, http://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf.

[2] Adapted from "Fourier Transform," Wolfram MathWorld, accessed March 13, 2016, http://mathworld.wolfram.com/FourierTransform.html.

[3] Adapted from "Fourier Transform," Wolfram MathWorld, accessed March 13, 2016, http://mathworld.wolfram.com/FourierTransform.html.

While the continuous transforms above work perfectly when dealing with a continuous function, the Discrete Fourier Transform and Inverse Discrete Fourier Transforms are used when dealing with discrete functions. The Discrete Fourier Transform commonly written as

$$X_v = \sum_{k=0}^{N-1} x_t e^{-2\pi i t v/N},$$

and the Discrete Inverse Fourier Transform is written as

$$x_t = \frac{1}{N} \sum_{n=0}^{N-1} X_v e^{2\pi i t v/N}$$

where X_v is a discrete function in the frequency domain, x_t is a discrete function in the time domain, v is frequency t is time, and N is is the number of points in the transform.[4]

This investigation will use the Discrete Fourier Transform to analyze audio signals because audio data is stored digitally with discrete values. (See next section.) By running the transform, the magnitude of the frequencies of the audio signal can be analyzed at various time intervals. It is important to note that using the Discrete Fourier Transform results in a symmetry around $0.5 f_s$ known as the Nyquist which add redundant information.[5] (See next section.)

Analyzing Audio Signals

Before audio signals can be analyzed, it is important to understand how audio signals are digitally stored. There are many file formats for encoding audio signals, but in essence, an audio file consists of a sample rate measured in Hertz (data points per second) and a matrix of entries whose rows are the amplitude of the audio wave at time t, measured from the beginning of the song ($t = 0$). The columns represent the left and right "channel," or audio signal for the left and right ear.[6] If an audio file has a sample rate of 44100 Hz, which is a common sample rate for audio files, each row of the matrix of amplitudes would represent a

$$\frac{1}{44100} \approx 0.00002267573$$

second interval. Thus, the first entry would be the amplitude at time $t = 0$ seconds, the second at $t = 0.00002267573$ seconds, the third at $t = 0.00004535147$ seconds, and so on.

S_{Dual}, on the next page, is an example audio signal represented as a matrix. The sample rate is 44100 Hz.

---

[4] Adapted from "Discrete Fourier Transform," Wolfram MathWorld, accessed March 13, 2016, http://mathworld.wolfram.com/DiscreteFourierTransform.html.

[5] "FFT Tutorial," University of Rhode Island Department of Electrical and Computer Engineering - ELE 436: Communication Systems, accessed March 21, 2016, http://www.phys.nsu.ru/cherk/fft.pdf.

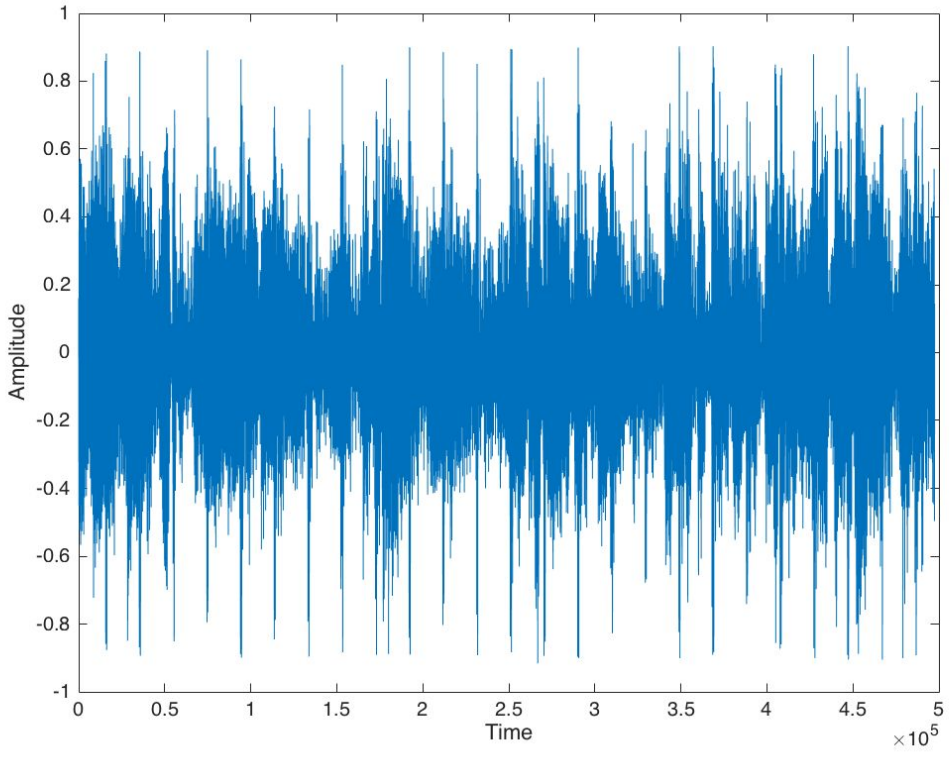
[6] These characteristics about audio files were deduced by analyzing various audio file formats using MATLAB.

$$S_{Dual} = \begin{bmatrix} -0.0420532226562500 & 0.0169982910156250 \\ -0.0574951171875000 & -0.0713195800781250 \\ -0.136291503906250 & -0.146820068359375 \\ -0.107666015625000 & -0.0498962402343750 \\ \vdots & \vdots \end{bmatrix}$$

In order to simplify the analysis process, the two channel signal can be converted to a single channel signal, or mono signal, by treating the first and second column of the S_{Dual} matrix as two separate vectors, adding the two vectors, then averaging the value by dividing it by two. Therefore,

$$S_{Mono} = \frac{1}{2}\left(\begin{bmatrix} -0.0420532226562500 \\ -0.0574951171875000 \\ -0.136291503906250 \\ -0.107666015625000 \\ \vdots \end{bmatrix} + \begin{bmatrix} 0.0169982910156250 \\ -0.0713195800781250 \\ -0.146820068359375 \\ -0.0498962402343750 \\ \vdots \end{bmatrix}\right) = \begin{bmatrix} -0.0125274658203125 \\ -0.0644073486328125 \\ -0.141555786132813 \\ -0.0787811279296875 \\ \vdots \end{bmatrix}$$

Graph 1 is a plot of S_{Mono} where the horizontal axis is the time axis, and the vertical the amplitude.



Graph 1: Plot of audio signal.

Although audio files store the time and amplitude data of audio signals, the magnitude of various frequencies of the audio signal are not stored. A Fourier Analysis must be performed to extract this frequency-magnitude data. Because the audio signal data is discrete, the Fast Fourier Transform (FFT) algorithm in MATLAB (on optimised implementation of the Discrete Fourier Transform) was used to analyze audio signals.[7] The FFT function takes the form $FFT(x,N)$ where x is the signal and N the number of frequency points. It is important that N be at least as large as the sample rate ($44100\ Hz$ for this investigation).[8]

F is a vector with 44100 elements, which is the result of the FFT of S_{Mono}.

$$F = FFT(S_{Mono}, 44100) = \begin{bmatrix} -11.7084197998047 + 0.00000000000000i \\ -11.6080348127239 - 0.964594419737194i \\ -13.3451041284403 - 0.344030055338166i \\ -9.96484769916224 - 2.46013776630596i \\ \vdots \end{bmatrix}$$

Each row, $n = 1, 2, 3, 4, \ldots, 44100$, contains the complex number f_n, where n is the index. The whole number n is the frequency, and the magnitude of f_n is the magnitude of the frequency n.[9]

$$|f_n| = \sqrt{Re(f_n)^2 + Im(f_n)^2}$$

is the magnitude of that frequency n, where $Re(f_n)$ is the real part of f_n, and $Im(f_n)$ is the imaginary part of f_n. For example, the magnitude of the first row, which is the magnitude of $1\ Hz$ is

$$\sqrt{(-11.708419799804688)^2 + (0.000000000000000)^2} = 11.708419799804688.$$

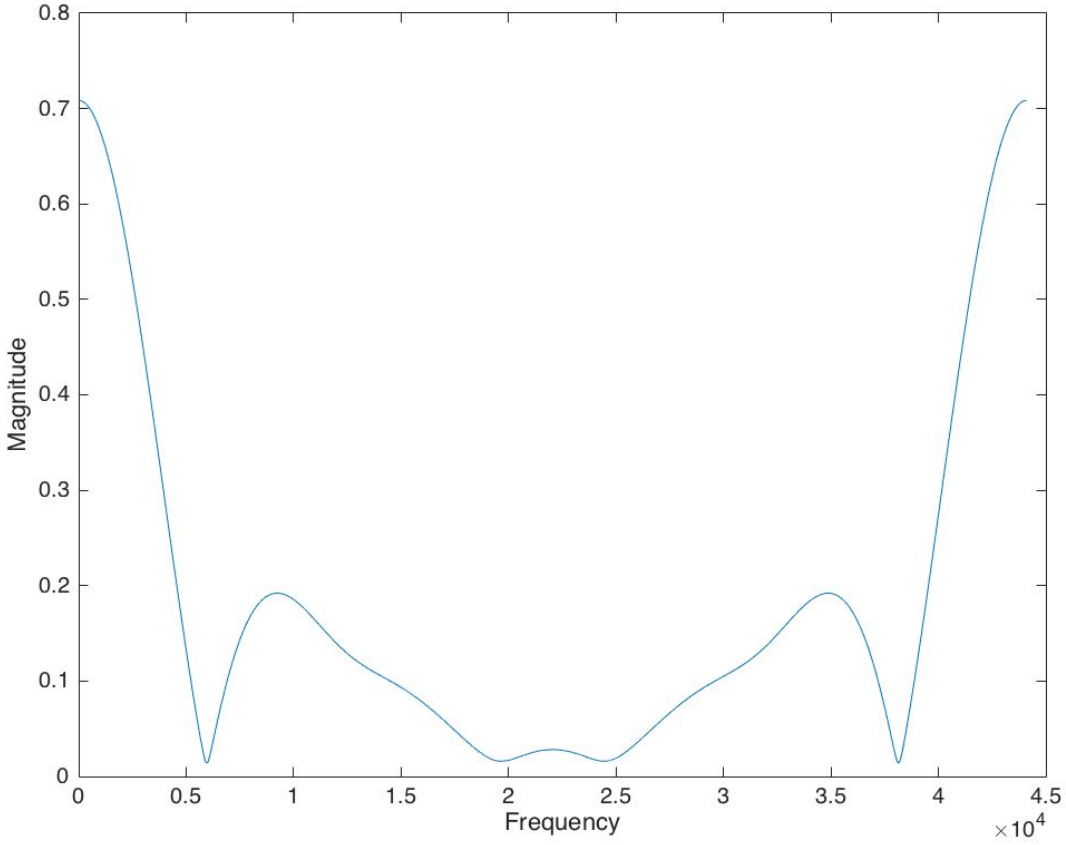
M is a vector whose elements are the magnitudes of the frequencies. The index n is the frequency.

$$M = \begin{bmatrix} 11.7084197998047 \\ 11.6480433811005 \\ 13.3495378525958 \\ 10.2640375825843 \\ \vdots \end{bmatrix}$$

The values of M have no unit. However, they can be easily converted to the decibel scale using the formula

$$M_{Decibel} = 20\log_{10}(M).$$[10]

[7] "Matlab," MathWorks, accessed March 13, 2016, http://www.mathworks.com/help/matlab/.
[8] "FFT Tutorial," University of Rhode Island Department of Electrical and Computer Engineering - ELE 436: Communication Systems, accessed March 21, 2016, http://www.phys.nsu.ru/cherk/fft.pdf.
[9] Brian Douglas, "Introduction to the Fourier Transform (Part 1 and 2)" (video), Jan 10, 2013, accessed March 13, 2016, https://www.youtube.com/watch?v=1JnayXHhjlg, https://www.youtube.com/watch?v=kKu6JDqNma8.
[10] "Matlab," MathWorks, accessed March 13, 2016, http://www.mathworks.com/help/matlab/.

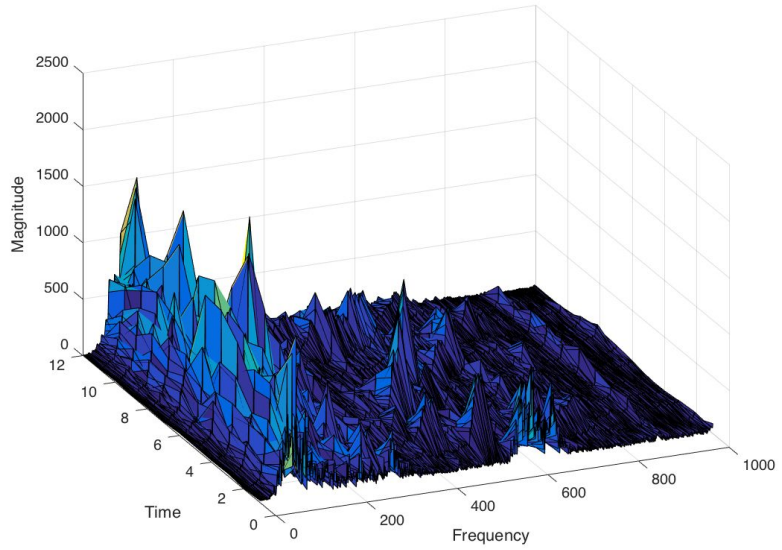


Graph 2: Plot of frequencies and magnitudes of an audio signal.

Graph 2 is a plot of M of an audio signal where the horizontal axis is frequency (rows of the vector), and the vertical axis the magnitude (elements of the vector). From *Graph 2*, it is clear that the FFT contains a symmetric duplicate of information.[11] In this case, the line of symmetry is at $22050\ Hz$. Thus, when analyzing the FFT of an audio signal, half of the data can be discarded. It is also important to notice at this point that all time data is lost when a FFT is performed on an audio signal. In order to preserve the time information so that the magnitude of the various frequencies at certain time intervals can be analyzed, the FFT must be run on certain ranges of the audio signal, and reconstructed into a matrix where the rows are ranges of time, the columns are frequencies, and elements magnitude at a specific time and frequency. X, on the next page, is such a matrix with 44100 columns and 12 rows. Because the FFT was performed every 44100 samples, each row represents 1 second ($44100/44100 = 1$).

[11] "FFT Tutorial," University of Rhode Island Department of Electrical and Computer Engineering - ELE 436: Communication Systems, accessed March 21, 2016, http://www.phys.nsu.ru/cherk/fft.pdf.

$$X = \begin{bmatrix} 11.708419799804688 & 11.648043381100502 & \dots & 91.286142264102340 \\ 0.875854492187500 & 1.638081122706419 & \dots & 18.599284062584506 \\ \vdots & \vdots & \ddots & \vdots \\ 4.970230102539063 & 5.103761808444784 & \dots & 14.331400670795963 \end{bmatrix}$$



Graph 3: Plot of audio signal in 3 dimensions.

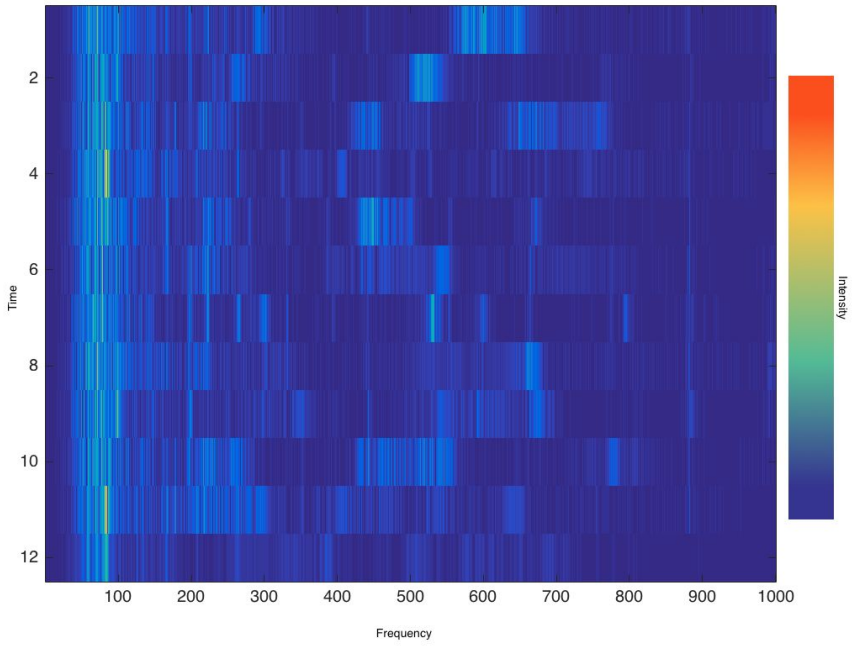


Image 1: Audio signal represented as an image.

Graph 3 is a 3 dimensional representation of X. Alternatively, X can be represented as an image, such as in *Image 1*. *Graph 3* and *Image 1* are often referred to as a spectrogram. The music identification algorithm is centered around effectively analyzing such spectrograms.[12]

Sequencing Music

During the sequencing process, a spectrogram is created by running the FFT on 2205 chunks, and reconstructing a matrix from the results of theses transforms. This results in a matrix similar to X except that each row represents $2205/44100 = 0.05$ second intervals. (All of the audio files were stored in the `.m4a` format, and had sample rates of 44100 Hz.) The resultant matrix was saved for each song sequenced. *Code 1* is a MATLAB program which performs these calculations. The green text are comments and do not affect the program. X_{song} is an example output of the program. Note that only 1103 columns are saved because the FFT is symmetric around 1102.5 Hz.

```
for currentSong=1:657   %Cycle through the audio files
    fileName = ['Song',num2str(currentSong),'.m4a']   %Load the audio file
    [sig,Fs] = audioread(fileName); %Extract the audio signal and sample rate
    monoSig = (sig(:,1)+sig(:,2))/2;      %Convert the signal to a mono signal
    interval = 2205;                      %Set the interval to run the FFT
    extraSpace = mod(length(monoSig),interval);       %Make sure monoSig is
    monoSig = [monoSig;zeros(interval-extraSpace,1)]; %divisible by interval
    currentPos = 1;
    allDB = [];                     %Matrix to store the result of the FFT
    for index=1:length(monoSig)/interval  %Cycle through the audio signal
        tempSig = monoSig(currentPos:currentPos+interval-1);
        Y = fft(tempSig,interval);  %Run the FFT
        magY = abs(Y);              %Find the magnitudes of the FFT result
        magY = magY(1:ceil(interval/2));  %Extract the first half of the data
        allDB = [allDB;reshape(magY,[1,length(magY)])]; %Add magY to allDB
        currentPos = currentPos + interval;
    end
    saveFileName = ['SequencedSong',num2str(currentSong),'.txt']
    dlmwrite(saveFileName,allDB);   %Save allDB as a plain text file
end
```

Code 1: Program to sequence audio file and save the resultant matrix.

[12] "How Shazam Works," Free Won't, January 10, 2009, accessed March 13, 2016, https://laplacian.wordpress.com/2009/01/10/how-shazam-works/.

$$X_{song} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0.0003204 & 0.00032158 & \dots & 1.565e-05 \\ 0.0076599 & 0.0025392 & \dots & 0.00012706 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

Hashing the Sequenced Music

Once an audio signal is sequenced, the frequency with the maximum magnitude for the frequency ranges $40 \leq f < 80$, $80 \leq f < 120$, $120 \leq f < 160$, and $160 \leq f < 200$ are identified.[13] These frequencies are the "key points" of the audio signal. The ranges were chosen because a lot of "activity" occurs in this range, as illustrated in *Graph 3* and *Image 1*. (Several other ranges up to $1000\ Hz$ were tested but the aforementioned ranges worked best.) The four frequencies are subsequently stored in a matrix with four columns. *K* is an example of this matrix.

$$K = \begin{bmatrix} 1 & 41 & 81 & 121 \\ 4 & 41 & 119 & 121 \\ 5 & 42 & 81 & 151 \\ 26 & 51 & 81 & 131 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Once the key frequencies are identified, the four frequency pairs are converted to a vector using a hashing algorithm. The hashing algorithm creates a unique number from 4 other numbers. The hashing algorithm follows the following recursive process:[14]

```
Start with Hash = 4 (any number other than 0 will work)
For each frequency:
      Hash = 17 * Hash * frequency (any prime number will work)
Add the Hash to a vector of hash values
```

Thus, the original audio signal is reduced to a vector of hash values with rows which represent 0.05 second intervals (a Hash-Time vector). *Code 2*, on next page, is a MATLAB program which performs these calculations. The green text are comments and do not affect the program. H is an example output of the program.

$$H = \begin{bmatrix} 352344 \\ 367729 \\ 372315 \\ 478069 \\ \vdots \end{bmatrix}$$

[13] Adapted from Roy van Rijn, "Creating Shazam in Java," June 1, 2010, accessed March 13, 2016, http://royvanrijn.com/blog/2010/06/creating-shazam-in-java/.

[14] "C# hashcode for array of ints," Stack Overflow, August 4, 2010, accessed March 13, 2016, http://stackoverflow.com/questions/3404715/c-sharp-hashcode-for-array-of-ints.

```
for currentSong=1:657          %Cycle through the sequence files of all songs
    fileName = ['FrequencySong',num2str(currentSong),'.txt']%Load the sequence
    allDB = dlmread(fileName);                              %matrix
    plotDB = allDB(1:end,40:200);   %Use only columns 40 to 200 for analysis
    keyPoints = [];            %Matrix to store the result of the key points
    for index=1:size(plotDB,1)  %Cycle through each row of the sequence matrix
        currentSubPos = 1;
        for subIndex=1:4       %Cycle through the 40 Hz ranges
            %Identify and store the key point for a frequency range
            [v,i] = max(plotDB(index,currentSubPos:currentSubPos+39));
            keyPoints(index,subIndex) = i+40*(subIndex-1);
            currentSubPos = currentSubPos + 40;
        end
    end
    hashTable = [];     %Matrix to store the hash values
    for index=1:size(keyPoints,1)   %Cycle through through key points
    hash = 4;           %Initial value of the hash (cannot be zero)
        for subIndex=1:4       %Cycle through the key points
            %Calculate the hash
            hash = 17*hash+keyPoints(index,subIndex);
        end
        hashTable(index,1) = hash;  %Store the hash
    end
    %Save the hash matrix as a plain text file
    saveFileName = ['DataSong',num2str(currentSong),'.txt']
    dlmwrite(saveFileName,hashTable);
end
```

Code 2: Program to hash audio file sequence and save the resultant vector.

Identifying Music

Once a database of Hash-Time vectors for a number of music audio files is compiled, a song can be identified by converting a sample audio signal into a Hash-Time vector and searching the database for a match.

The following are the steps of the music identification process:

1. Record a 1 to 10 second sample of the audio signal through a microphone.
2. Create a sample Hash-Time vector for the audio signal (H_{Sample}).

3 Cycle through the music Hash-Time vectors (H_{Music}).

4. Divide H_{Sample} and H_{Music} by 1000 and round the elements to a whole number.
5. Cycle through the elements of H_{Sample}.
6. Find the indices of each H_{Music} which matches with the element of the H_{Sample}. Store these indices in vector M_{Temp}.

7. Subtract the index of element of the H_{Sample} being matched from M_{Temp}.[15]
8. Add the elements of M_{Temp} (normalized time values) to vector M_{All}.
9. Repeat steps 3 through 8.
10. Find the mode of M_{All}.
11. Find the number of elements n_{mode} in M_{All} with the mode value.
12. Find the ratio

$$r = \frac{n_{mode}}{\text{number of elements in } M_{All}}.$$

13. Add r to a vector R_{All}.
13. Repeat steps 3 through 13.
14. Find the index of R_{All} with the maximum value. This is the ID number of the song which is the closest match to the sample audio signal.

H_{Sample} and H_{Music} are divided by 1000 and the elements are rounded to a whole number in step 3 to compensate for small errors in frequency readings. Testing showed that 1000, which reduces the elements to three significant digits, results in more viable matches and increases the viability of the algorithm.

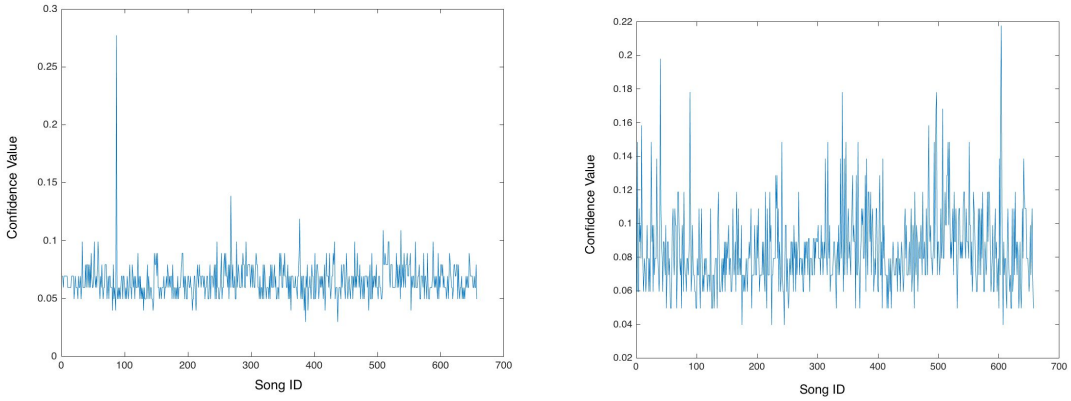
```
Matched Hash Values:     126   760   873   964

Music Time:              100   110   120   130
Sample Time:             0     10    20    30

Normalized Time:         100   100   100   100
(Music Time - Normalized Time)
```

Diagram 1: Calculating the normalized time value.

In addition, the index of element of the H_{Sample} being matched is subtracted from M_{Temp} in step 7 in order to extract the normalized time value. This normalized time value represents the time at which the sample audio signal starts in the song. As *Diagram 1* shows, if the sample audio signal starts at $t = 100$ in the song, the difference of the indices which matched to the sample Hash-Time vector elements and the index of the sample Hash-Time vector elements equal 100. (Note that the index is a time value.) It follows that the most common normalized time value is the position at which the sample audio signal begins in the song (step 11). By taking the ratio of the number of elements with the most common time-shift value and the number of overall matches, the confidence of a match can be determined (r in step 12). Thus, choosing the match with the highest confidence value is the most likely match (step 14).

[15] Adapted from Roy van Rijn, "Creating Shazam in Java," June 1, 2010, accessed March 13, 2016, http://royvanrijn.com/blog/2010/06/creating-shazam-in-java/.

The left plot in *Graph 4* illustrates a match with Song 87. (Each song was assigned a number from 1 to 657.) There is a clear peak in confidence value at 87. On the other hand, the right plot in *Graph 4* illustrates an instance of no unique match being found. There are many peaks but no single unique one.



Graph 4: Plots of the confidence values for a match and inconclusive match.

Code 3 is a MATLAB program which performs this matching process. The green text are comments and do not affect the program. Once the sample audio signal is recorded, the matching process is almost instantaneous.

```
%sampleHashTable is the Hash-Time vector of the sample audio signal
%Normalize the Hash-Time vector of the sample audio signal
sampleHashTable = sampleHashTable / 1000;
sampleHashTable = round(sampleHashTable);
allRatios = [];   %Vector to store all of the ratio values
for currentSong=1:657   %Cycle through all of the audio file Hash-Time vectors
    fileName = ['DataSong',num2str(currentSong),'.txt']; %Load the Hash-Time
    song = dlmread(fileName);                             %vector
    %Normalize the Hash-Time vector of the audio file
    song = song / 1000;
    song = round(song);
    matches = []; %Vector to store all of the match times
    for index=1:length(sampleHashTable)   %Cycle through the sample audio
                                          %Hash-Time vector and find matches
        %Turn the match into normalized time and add it to the matches vector
        matches = [matches;find(song==sampleHashTable(index))-index];
    end
    m = mode(matches);  %Find the mode of the matches
    numberOfMode = sum(matches(:) == m);  %Find the number of mode values
    ratio = numberOfMode/length(sampleHashTable);     %Find the ratio
    allRatios = [allRatios;ratio];                    %Store the ratio
end
[v,i] = max(allRatios);       %Find the maximum ratio
%v is the value and i is the index of the maximum ratio
```

Code 3: Program to identify a sample audio signal.

Testing Music Identification

A database of the Hash-Time vector of 657 songs were compiled to test the matching algorithm. It took approximately 15 hours to prepare the database (1 minute and 24 seconds per song).

Overall, the matching algorithm worked very well. The samples were recorded from a microphone in real time and matched against the database. In many cases, the algorithm correctly identified a song with only a one second sample. At most, a ten second sample was sufficient to correctly identify as song.

Conclusion

This investigation has shown that a robust music identification algorithm can be constructed by analyzing audio signals using Fourier Analysis. As the testing illustrated, the algorithm presented in this investigation is just as effective, if not more effective, as commercial music identification systems such as Shazam. Shazam and other companies have had many years to optimize their algorithms for faster and more accurate match results. In fact, when Shazam first launched, a ten second audio sample was needed to identify songs. Surprisingly, the comparably simple algorithm presented in this investigation can identify many songs with only a 1 second audio sample. Moreover, while Shazam uses a much more complex system for calculating hash values which takes into account time,[16] the algorithm presented was effectively able to identify music by using hash values which did not incorporate any time data. This simpler implementation was made possible by the unique method of matching Hash-Time vectors.

Although the music identification algorithm presented is surprisingly robust, it does contain some limitations. First and foremost, only 657 songs were included in the Hash-Time vector database, and the algorithm's effectiveness with a more realistic database of millions of songs is yet to be tested. In addition, not all frequency ranges were fully tested to conclude on the best way to choose key frequencies. Furthermore, there are other sample audio signal ranges which the FFt can be use on which may be effective, and this path was not fully explored. (Only a few sample ranges were tested.)

Overall, this investigation yielded a robust music identification algorithm which can be improved upon to possibly elevate the speed and percentage of positive matches to a commercial level.

[16] Avery Li-Chun Wang, "An Industrial-Strength Audio Search Algorithm,", accessed March 13, 2016, http://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf.

Bibliography

Douglas, Brian. “Introduction to the Fourier Transform (Part 1)” (video). Jan 10, 2013. Accessed March 13, 2016. https://www.youtube.com/watch?v=1JnayXHhjlg.

Douglas, Brian. “Introduction to the Fourier Transform (Part 2)” (video). Jan 19, 2013. Accessed March 13, 2016. https://www.youtube.com/watch?v=kKu6JDqNma8.

Free Won't. “How Shazam Works.” January 10, 2009. Accessed March 13, 2016. https://laplacian.wordpress.com/2009/01/10/how-shazam-works/.

MathWorks. “Matlab.” Accessed March 13, 2016. http://www.mathworks.com/help/matlab/.

Stack Overflow. “C# hashcode for array of ints.” August 4, 2010. Accessed March 13, 2016. http://stackoverflow.com/questions/3404715/c-sharp-hashcode-for-array-of-ints.

University of Rhode Island Department of Electrical and Computer Engineering - ELE 436: Communication Systems. “FFT Tutorial.” Accessed March 21, 2016. http://www.phys.nsu.ru/cherk/fft.pdf.

van Rijn, Roy. “Creating Shazam in Java.” June 1, 2010. Accessed March 13, 2016. http://royvanrijn.com/blog/2010/06/creating-shazam-in-java/.

Wang, Avery Li-Chun. “An Industrial-Strength Audio Search Algorithm.” Accessed March 13, 2016. http://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf.

Wolfram MathWorld. “Discrete Fourier Transform.” Accessed March 13, 2016 .http://mathworld.wolfram.com/DiscreteFourierTransform.html.

Wolfram MathWorld. “Fourier Transform.” Accessed March 13, 2016. http://mathworld.wolfram.com/FourierTransform.html.