EE473
Semester Project Report

BOĞAZİÇİ ÜNİVERSİTESİ

Music Recognition
Using Audio Fingerprint

Submitted by:

Yağız Gençer

Aybars Manav

Ümran Yungucu

1   Introduction

In this project, we created a **Shazam-like** app, called **GiMY − Give Me the Song**. The program is designed to recognize any song that is present in its database when any 10-second interval of the song is played in the environment. Our database consists of 285 different songs. Ideally, programs such as Shazam [**?**] search in their databases using multiple servers with great computational power which we are not able to satisfy employing our personal computers. However, one should note that the number of songs can be varied without any loss of generality since the algorithm itself remains the same independent of the size of the database. Our program is designed via MATLAB and is also supported by an interactive GUI application.

2   Algorithm

The algorithm can be divided into four essential operations.

2.1   Creating Constellations

Each song is identified and compared with each other for a possible match based on their **fingerprints**. The fingerprint of each song consists of a set of **hashes**. The concept of the hash will be explained in the later stages of the report. However, at this point, it is important to note that we need to have the frequency components of the song with peak amplitudes to create its hashes. For this purpose, we get the spectrogram of each audio. The spectrogram of any audio shows the PSD amplitudes of the frequency components present in the audio at each time instant by taking the DFT of time-overlapping windows. The window we have chosen contains $W = F_s \times w_t$ samples where w_t is the time duration of the window. The overlap is chosen as $0.6W$, this means we have points on the spectrogram at each time instant $t = 0.4 \times k$ until the end of the audio file where $k \in \mathbb{N}$. One of the example spectrograms can be seen in Fig.1.
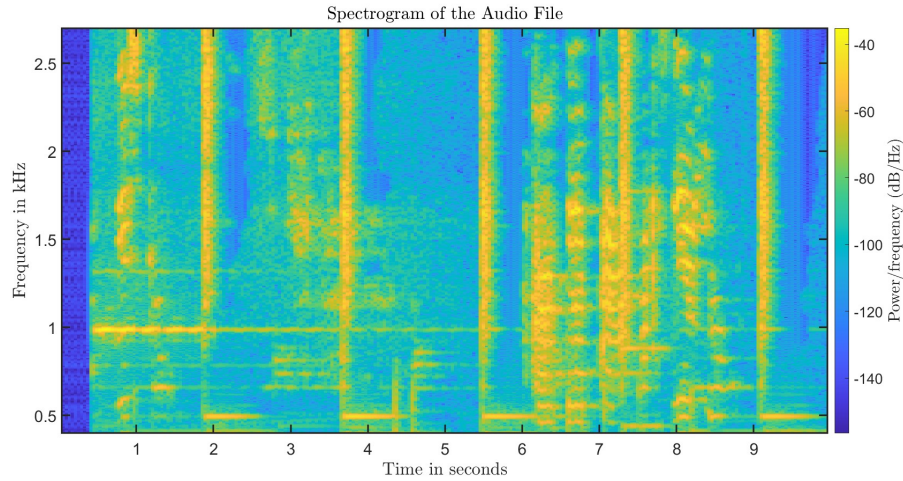

Figure 1: Spectrogram of the 10-second audio

For our purposes, we only need the magnitudes of the spectrogram. As a result, we have a spectrogram matrix of the audio that represents the time axis along its columns and frequencies along its rows and therefore any cell of the matrix holds the magnitude of the corresponding frequency component at that specific time. Another important point is that for improved efficiency, we should only consider the relevant frequencies which are the most likely to come up in songs. Based on our observation during tests, it is unusual for a song to have frequency components larger than 2700 Hz. Furthermore, the low-frequency components are heavily affected by the noise and therefore such components

in audio are often unreliable for recognition and detection purposes. Therefore, we have only considered the frequencies from 400 Hz to 2700 Hz while creating the fingerprints.

Now, it is time to select the peaks of the spectrogram matrix as the next step of creating fingerprints of audio. We conclude that a point in the matrix is a peak if and only if it is larger than its neighbors. For this, we use local windows of size 15×15 (225 elements) such that each element of the spectrogram matrix has its own local window centered at the element itself. *Note that these local windows are different than the window used in spectrogram which meant the number of samples during a DFT operation.* Then, an element is a peak if it is larger than all the other elements encapsulated by its local window. For comparing each of the elements to their neighbors, an efficient way is to make use of the circular shifts. For each element in the matrix, we start comparing starting from the bottom right up to the top left of the local window. At each iteration, we update the elements of a matrix called "mask_peaks" which has the size of our spectrogram matrix and is initialized with ones at each element. At the end of circular shift operations, any element of the "mask_peaks" is one if and only if it is larger than all of its neighbors otherwise, it is set to zero. When we multiply this matrix with the spectrogram magnitude matrix, we effectively select the peak element at each window. We call this collection of peak points as a "constellation" and therefore this above-described operation of selecting the peak values is done in the file "getConstellation.m". The constellation of the same audio file given in Fig.1 is shown in Fig.2.
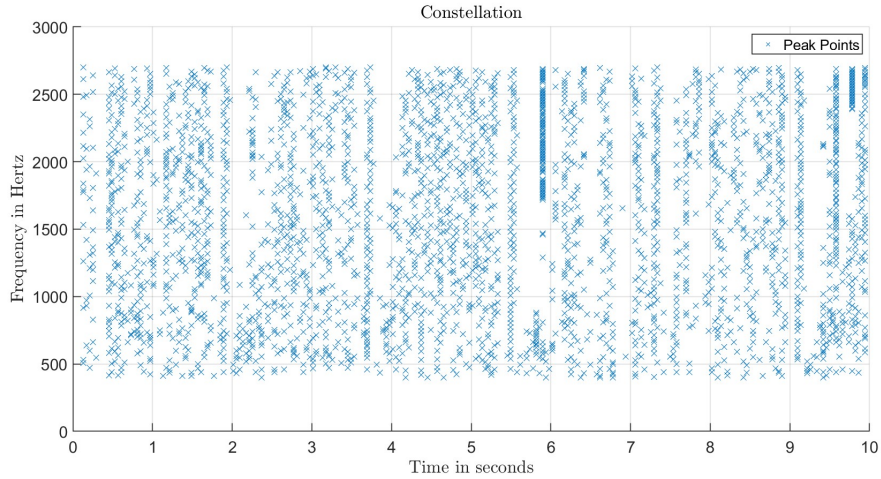

Figure 2: The constellation of the 10-second audio

2.2 Reducing the Number of Peaks in a Constellation

However, the total number of peaks in the constellation is still too high and it is unpractical to work with such a high number of peaks. Furthermore, the distribution of peaks is non-uniform in its present form, and therefore, without a uniform distribution of peaks, each time interval of the audio would not be given the same importance when creating its fingerprint. Therefore, we take the constellation that we have just created and divide it into many time slices. At each slice, we find the highest 30 peaks and leave them as they are except for the last slice. Since the last slice can be shorter than the other slices, the number of surviving peaks are scaled with $30 \times \frac{\text{length of the last slice}}{\text{slice length}}$. All the other peaks at each slice are set to zero. This way, we ensure the uniformity of the peaks and also avoid having more hashes than needed per audio by creating an audio fingerprint. This described operation is done in the file "getReducedConstellation.m" and the matrix we obtain at the end that includes fewer peaks than the original constellation matrix is the reduced constellation matrix as shown in Fig.3.
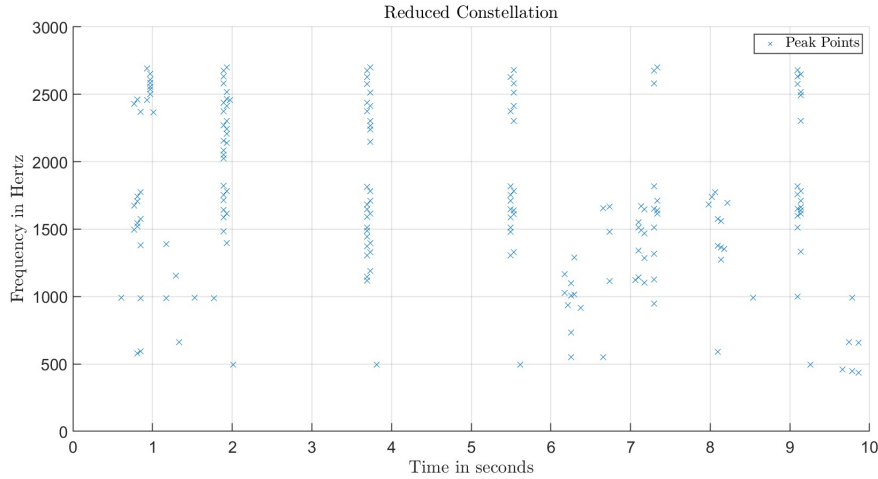
Figure 3: The reduced constellation of the 10-second audio

2.3 Creating Hashes

As mentioned, the fingerprint of a song consists of a series of hashes. A hash is a vector, which describes the relative position of two peaks in the *reduced* constellation matrix. For each hash, one of these points is the *anchor point*, relative to which many other hashes will/may be calculated. We have selected all the peaks in the reduced constellation as anchor points, one by one. Each anchor point has its own *target zone* and from this anchor point, multiple hashes will be created. A particular hash corresponding to this anchor point will be formed for each peak present in the target zone, if there are any.

For instance, let us say that one of our peaks is located at the location (i, j) in the reduced constellation matrix and we want to treat it as an anchor point to form the hashes of this anchor point. Furthermore, assume that the point (i, j) corresponds to the frequency f_1 and time t_1 respectively in our reduced constellation matrix. We now have to consider the target zone of this anchor point, in which we will search for the other peaks. We defined the width of the target zone as 40 and its height as 50. In addition, the target zone begins at 5 elements to the right of the anchor point. Therefore, the target zone of an anchor point at the (i, j) of the reduced constellation is a rectangle, the four corners of which are located at $(i + 25, j + 5)$, $(i + 25, j + 45)$, $(i - 25, j + 5)$ and $(i - 25, j + 45)$. We look for non-zero elements (in other words, peaks) in this zone. Let us assume that an arbitrary peak in the target zone is located at a point that corresponds to time t_2 and frequency f_2 of the reduced constellation. Then, the **hash** indicating the relative position of the anchor point and this peak in its target zone in frequency and time is defined and saved as $[f_1; f_2; t_2 - t_1; t_1]$. In other words, each hash stores the frequency of its anchor point, the frequency of the specific peak in its target zone, the time difference between the two, and the absolute time of the anchor point. The purpose of storing the absolute time of the anchor point for each hash will be explained in a later section of the report. The procedure of hash formation is illustrated in Fig.4.
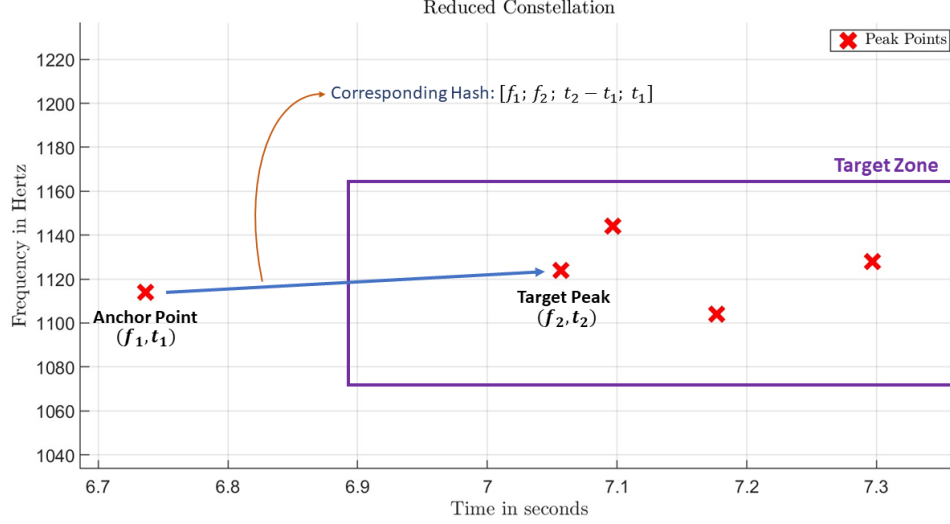
3

Figure 4: Creating a single hash

2.4   Searching

At this point, we must have generated a list of hashes for each song in the database. The data is kept as a $(5 \times N_h)$ matrix in a text file where N_h is the total number of hashes. The rows indicate the values for [Track ID; f_1; f_2; Δt; t_1] respectively. A sample song of duration 10 seconds is given as input to the GUI. A constellation is created for the sample, then it is reduced and the its hashes are created. An efficient searching algorithm is taking each hash of the sample song and subtracting it from the data matrix. The second, third, and fourth rows of the difference matrix corresponding to $[f_1 - f_1'; f_2 - f_2'; \Delta t - \Delta t']$ are compared to three threshold values separately. If all the difference values are below their respective threshold values, the hashes are matching.

We stored the matching hashes and from now on, we only care about the absolute time differences and track IDs of the matching hashes. If the sample audio belongs to a particular song in the database, we must observe a prominent value for the absolute time differences among their matching hashes. Even though the audio and the corresponding song in the database have the same hashes at possibly different absolute time positions, overall matching hashes will still have the same positions relative to each other. For instance, let us assume we have two matching hashes from the sample audio H_1' and H_2' with absolute time values t_1' and t_2'. Also assume that we have the corresponding hashes from the correct song H_1 and H_2 with absolute time values t_1 and t_2. Then, the absolute time differences must satisfy $t_1 - t_1' = t_2 - t_2' = C$ where C is a constant. This result can be expanded to any number of absolute time differences.

Using a histogram for the absolute time differences is feasible to detect the prominent constant C in this case. Even though the sample audio will have matching hashes with all other songs in the database, the absolute time differences will be mostly arbitrary. Consequently, we will only observe a prominent time difference among others if the recorded audio indeed belongs to a particular song in the database. The histograms of time absolute time differences of matching hashes are shown in Fig.5.

3   Application

The app has a "RECORD" button and when the button is pressed, the program listens to the environment through a microphone and records the audio for 10 seconds. The "RECORD" button is accompanied by a lamp that is white (non-active) in standby mode. When the button is pressed and the program is recording the audio, it flashes
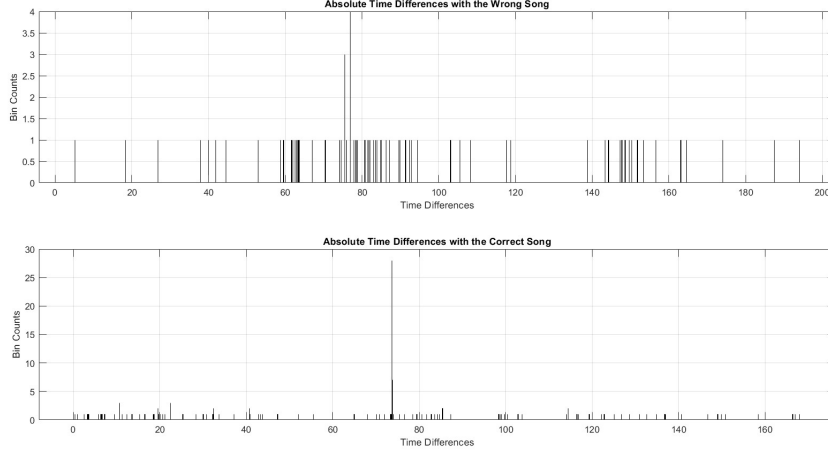
Figure 5: Comparison of absolute time differences of matching hashes of the actual song and the second-best predicted song in the database

green. When the recording phase is over and the program is searching in the database to find the matching songs, it flashes red, signaling that the recording is over, but the button is not yet available for another push. After the search is over and the results are listed, it turns white again as the program is now in standby mode and recording/searching for another song is possible. The app also has a "Progress" text bar that informs the user about the state and/or result of the program. When in standby mode, it displays: "Waiting for a song". During the recording and searching, it informs the user by displaying "Recording. . ." and "Processing. . ." respectively. When the search is over and there is no song in the database similar to the recording, it displays: "The song does not have a match in the database.". Lastly, when there is at least one song similar to the recording, it displays: "The song is successfully identified.". In such a case, the app also displays the names of the five most similar songs and their corresponding probabilities of being the correct match of the recorded song.

4 Difficulties and Solutions

4.1 Finding a Peak in a Local Neighborhood

To check whether an element is the largest in its local window, we have come up with the idea of utilizing circular shifts both in horizontal and vertical axes to compare with each element of the window. This method worked better than nesting for loops and reduced the computational overhead.

4.2 Slow Search

Instead of looking at every song one by one to find a match with the hashes in the audio, we hold hashes of all songs in a huge matrix and perform subtraction operations on it, and then compare the differences with thresholds to find matches. This difference operation also helped us during the detection of the prominent absolute time difference value of the matching hashes.

5   Numerical Results

5.1   Accuracy

We have achieved 92.63% accuracy in a database consisting of 285 songs from various genres. We used our microphones to record a 10-second segment of the songs. These audio samples were recorded in EE Lounge which is always crowded and noisy as it is a place for group studies.

5.2   Rejecting the Negative Samples

We used 40 sample audios that were not in the database. Expectedly, GiMY did not find any match for the songs and inform the user about the absence of the song in its database. We set a threshold of 30% match percentage and if the matching percentage of the best matching song that GiMY found is below this threshold, it is concluded that this song is not from the data. The match percentage is calculated using the top 5 songs returned by GiMY. The **match percentage** is defined as *the ratio of the number of time-matching hashes of a song to the total number of the time-matching hashes of the top 5 matching songs.*

5.3   Performance in a Noisy Environment

While recording the sample audio we have also talked next to the audio source. If the chat sounds overwhelmingly suppress the sample audio, even though matches are found with the true song, the matching percentage mostly drops to values under the threshold. However, if the song is distinctive enough, we have observed that GiMY still finds the song, but of course with a decrease in the match percentage.

5.4   Speed

5.4.1   Finding a Match

The algorithm returns the most matching songs in 5 seconds on average. We observed that the inference time was around 4.5 seconds when the data consisted of 100 songs. Increasing the number of songs did not result in excessive overhead such that the operation time increases slower than a linear fashion with respect to the song number in the database.

5.4.2   Creating the Data

Since the songs in the database must be stored with their corresponding hashes, all the songs must go through the same process as the sample audio. This process takes a lot of time as the number of songs increases. However, this is a one-time process that we do not repeat again and again unless we change the parameters of the hashing process and want to create a new database accordingly.

6   Conclusion

We have created GiMY as our term project for EE 473 course. GiMY decides which song is played by using the fingerprints of the database and the sample audio. GiMY also comes with a graphical user interface. The user can record a sample audio using the GUI and GiMY returns the matching song after listening to any 10-second interval of the song. It takes only 5 seconds for GiMY to give you the top matches if the actual song is in the database.

Appendix A   Code

A.1   Main

```
1  %% Creating the database if Data.txt is not in the folder
2  %createDatabase();
3  %% Reading the .txt files
4  global data
5  global songNames
6  [data,songNames] = loadData();
7  %% Running the GUI
8  run('GiMY_app.m');
```

A.2   createData

```
1  function [] = createData()
2  % Generates the hashes of the songs in the .mp3 format. The hashes are then
3  % written into 'Data.txt' and names of the songs are written into
4  % 'SongNames.txt'
5
6      % Gets names of all .mp3 files in current directory
7      tic
8      audioInDir = dir('*.mp3');
9      audioNames = {audioInDir.name};
10     hashes = []; % To hold all the hashes of stored songs
11     filenames = cell(1,length(audioNames));
12     if(isempty(audioNames))
13         disp 'No Audio Files in Current Directory'
14     else
15         %Audio files are present in the directory
16         for i=1:length(audioNames)
17
18             filename = char(audioNames(i));
19
20             hash = hashing(filename);
21             % Append song ID at the beginning.
22             hashes = [hashes, [repmat(i,[1,size(hash,2)]);hash]];
23             filenames{i} = filename;
24         end
25         % Write the songs whose hashes extracted to a single matrix.
26         writematrix(hashes,'Data.txt','Delimiter',' ');
27         % Keep the name of the songs in a txt file.
28         writecell(filenames,'SongNames.txt','Delimiter',',');
29     end
30     toc
31 end
```

A.3   loadData

```
1  function [data,songNames] = loadData()
2  % Loads Data.txt in which the hashes for the whole songs in the database
3  % are located and SongNames.txt in which the names of the songs are located
   .
4      tic
5      global data;
6      global songNames;
7      datafile = 'Data.txt';
8      namefile = 'SongNames.txt';
9      data = readmatrix(datafile,'Delimiter',' ');
10     songNames = readcell(namefile,'Delimiter',',');
11     toc
12 end
```

A.4   getConstellation

```
1  function [peak_magnitudes, spec_f, spec_t, new_fs, window] =
       getConstellation(song_file)
2  % Takes a .mp3 or .wav file and returns the peak spectrum points.
3  % Inputs:
```

```matlab
 4 % song_file              An audio file containing the song.
 5 % Outputs:
 6 % peak_magnitudes        Spectrogram matrix containing peak magnitude in
 7 %                        each local neighbourhood.
 8 % spec_f                 Frequencies
 9 % spec_t                 Time instants
10 % new_fs                 New sampling rate
11 % window                 Number of samples per window of spectrogram
12
13
14
15 new_fs = 8192; % 8192 is new sampling rate of signal. Corresponds to the
      highest note on a piano.
16
17
18 [song, Fs] = audioread(song_file);
19 % Since the songs read are stereo, we average the two columns and get a one
20 % dimensional column vector, we later convert it to a row vector.
21 if length(song(1,:))>1
22     song_mono = ((song(:,1)+song(:,2))./2)';
23 else
24     song_mono=song';
25 end
26 song_mono = song_mono - mean(song_mono); % Remove DC component
27
28 % Resampling the mono song so that the sequence length is decreased and
29 % high frequencies are excluded. This results in not taking high
30 % frequencies into account and might reduce the probability of detecting
31 % a song correctly. However, we also reduce time and memory complexity.
32 song_rs = resample(song_mono, new_fs,Fs);
33
34 % We also do not want to account for low frequencies since it reduces
35 % reliablity.
36 frequencies = 400:2:2700; % We can represent up to newfs/2 frequency.
37
38 timePerWindow = 0.1; % Window time duration for spectrogram
39
40 window = round(timePerWindow*new_fs); % Number of samples per window
41 nOverlap = round(0.6*window); % Number of samples overlapping while sliding
      the window
42
43 % Spectrogram takes window-point dft at every slice of the song.
44 % spec_f is the corresponding frequencies to row indices of the spec_song
45 % spec_t is the corresponding time instants to column indices of the
46 % spec_song
47 [spect_song,spec_f,spec_t] = spectrogram(song_rs,window,nOverlap,
      frequencies,new_fs,'yaxis');
48
49 spec_magnitude = abs(spect_song); % Magnitude of the spectrogram
50
51 %% Finding Peak Magnitudes
52
53 % One side length of the local window. The local window will be a
54 % (len_localWindow + 1) x (len_localWindow + 1) window. Each element in
55 % spec_magnitude will be compared with (len_localWindow + 1)^2 - 1 elements
56 % around it. If a specific element is greater than all of others, the
57 % mask_peaks matrix will still contain 1 in the corresponding index,
58 % otherwise we update it to 0.
59 len_localWindow = 7;
60
61 mask_peaks = ones(size(spec_magnitude)); % The boolean mask containing ones
      , will be updated.
62
63 % For comparing each of the element to neighbors, an efficent way is to
64 % make use of the circular shifts. For each element in the spec_magnitude
65 % we start comparing starting from the bottom right up to top left of the
66 % local window and update mask_peaks.
67 for shift_hor = -len_localWindow:len_localWindow
68     for shift_ver = -len_localWindow:len_localWindow
69         if(shift_ver ~= 0 || shift_hor ~= 0) % Avoid comparing to self
70             mask_peaks = mask_peaks.*( spec_magnitude > circshift(
```

```
        spec_magnitude ,[ shift_hor , shift_ver ]) );
71        end
72    end
73 end
74 % The actual values of peak magnitudes will be used for choosing local
75 % maximum peaks. The peaks will be further reduced.
76 peak_magnitudes = mask_peaks .* spec_magnitude ;
```

A.5   getReducedConstellation

```
 1 function [ reduced_constellation , spec_t , fs] = getReducedConstellation (
       peak_magnitudes , spec_f , spec_t , fs , window )
 2 % Among the peak_magnitudes , this function selects top N peaks from each
 3 % interval . The peak_magnitudes matrix is further updated to have 0 if
 4 % peak magnitudes are not in top N.
 5 % Inputs :
 6 % peak_magnitudes              The peak magnitude matrix obtained from
 7 %                              getConstellation function .
 8 %
 9 % spec_t                       The corresponding time instants to
10 %                              peak_magnitudes columns .
11 %
12 % spec_f                       The corresponding frequency components to
13 %                              peak_magnitudes rows .
14 %
15 % fs                           The sampling rate of the spectrogram
16 %
17 % window                       Number of samples in a window of
18 %                              spectrogram .
19 % Outputs :
20 % reduced_constellation        Constellation with reduced the number of
21 %                              peaks in peak_magnitudes that is more
       uniform .
22 %
23 % spec_t                       The corresponding time instants to
24 %                              peak_magnitudes columns .
25 %
26 % fs                           The sampling rate of the spectrogram
27
28
29 % Number of top peaks to survive .
30 NtopPeaks = 30; % 30;
31 % Updated and more uniform spectrogram , initialization with zero
32 reduced_constellation = zeros ( length ( spec_f ), length ( spec_t ));
33
34 % Interval length
35 index_interval = floor (( fs /( window /4))*1);
36
37 for i = 0: ceil ( length ( spec_t )/ index_interval ) -1
38
39     % At the end of the time array we must get whatever is left regardless
40     % of the interval length
41     if i == ceil ( length ( spec_t )/ index_interval ) -1
42         % Choosing the last sub interval of the peak spectrogram
43         relevant_spect = peak_magnitudes (: , i* index_interval +1: length ( spec_t
    ));
44         % Update the NtopPeaks according to the size of the last interval
45         % updated_NtopPeaks is scaled according to last_interval / interval
46         updated_NtopPeaks = ceil ( size ( relevant_spect ,2)/ index_interval *
    NtopPeaks );
47         % Choosing the maximum updated_NtopPeaks number of values to later
48         % use the last one as a value for threshold
49         temp = maxk ( relevant_spect (:) , updated_NtopPeaks );
50     else
51         % Choosing the given interval of the peak spectrogram
52         relevant_spect = peak_magnitudes (: , i* index_interval +1:( i+1)*
    index_interval );
53         % Choosing the maximum NtopPeaks to later use as a threshold
54         temp = maxk ( relevant_spect (:) , NtopPeaks );
55     end
56     relevant_spect ( relevant_spect < temp ( end )) = 0;
```

```
57
58      % Assign values to the reduced_spect
59      if i == ceil(length(spec_t)/index_interval)-1
60          reduced_constellation(:,i*index_interval+1:length(spec_t)) =
        relevant_spect;
61      else
62          reduced_constellation(:,i*index_interval+1:(i+1)*index_interval) =
        relevant_spect;
63      end
64  end
65  end
```

A.6   getHashes

```
1  function [hashes] = getHashes(reduced_constellation, spec_t, spec_f)
2  % For each anchor point in the reduced constellation getHashes function
3  % returns the hashes. Hashes are created by pairing each anchor point with
4  % points in its target zone. The hashes are translation invariant. A hash
5  % consist of frequency of the anchor point, f1; frequency of target point,
6  % f2; time difference of these two points, delta t. We use each nonzero
7  % constellation point (peak) as anchor points.
8
9  [size_rowsCons, size_colsCons] = size(reduced_constellation); % Dimensions
       of constellation
10 [i_rowPeak, i_colPeak] = find(reduced_constellation); % Indices of nonzero
       constellation points (peaks)
11 nPeaks = nnz(reduced_constellation); % Number of peaks
12
13
14 width_TargetZone = 40; % Width of the target zone in terms of time instant
       indices, right of the anchor point
15 height_TargetZone = 50; % Height of the target zone in terms of frequency
       indices. Must be an even integer.
16 initial_HorDist = 5; % Initial horizontal distance from anchor point to
       target zone.
17
18 nMaxPairs = 5; % Number of points in target zone to pair for each anchor
       point.
19
20
21 f1=[];
22 f2=[];
23 delta_t=[];
24 t1 = [];
25
26
27 % Each peak is considered an anchor point so we will loop over them.
28 for i = 1:nPeaks
29      % The frequency and time indices of the current anchor point
30      i_row_anchor = i_rowPeak(i);
31      i_col_anchor = i_colPeak(i);
32
33      % If the target zone right end is outside of the indices, update the
34      % target zone right limit
35      if i_col_anchor + width_TargetZone + initial_HorDist > size_colsCons
36          target_right_end = size_colsCons;
37      % Otherwise assign the target zone right end
38      else
39          target_right_end = i_col_anchor + width_TargetZone +
        initial_HorDist;
40      end
41
42      % If the target zone left end is outside of the indices update the left
43      % target zone limit
44      if i_col_anchor + initial_HorDist > size_colsCons
45          target_left_end = size_colsCons;
46      % Otherwise assign the target zone left end
47      else
48          target_left_end = i_col_anchor + initial_HorDist;
49      end
50
```

```matlab
51      % If the target zone top end is outside of the indices , update the
52      % target zone top limit
53      if i_row_anchor - height_TargetZone /2 < 1
54          target_top_end = 1; % Row 1
55      % Otherwise assign the target zone top end
56      else
57          target_top_end = i_row_anchor - height_TargetZone /2;
58      end
59
60      % If the target zone bottom end is outside of the indices , update the
61      % target zone bottom limit
62      if i_row_anchor + height_TargetZone /2 > size_rowsCons
63          target_bottom_end = size_rowsCons ;
64      % Otherwise assign the target zone bottom end
65      else
66          target_bottom_end = i_row_anchor + height_TargetZone /2;
67      end
68
69      nPairings = 0; % Number of current pairs for the anchor points
70
71      % Find the peaks inside of the target zone
72      target_zone = reduced_constellation ( target_top_end : target_bottom_end
        ,...
73          target_left_end : target_right_end );
74      [j_row_targets , j_col_targets] = find( target_zone );
75      for j = 1: length ( j_row_targets )
76              % Complying with the nMaxPairs constraint and not pairing with
77              % anchor point itself
78              if ( nPairings < nMaxPairs && spec_t ( j_col_targets (j) +
        target_left_end - 1) - spec_t ( i_col_anchor ) > 0)
79                  f1 = [f1 , spec_f ( i_row_anchor ) '];
80                  f2 = [f2 , spec_f ( j_row_targets (j) + target_top_end - 1) '];
81                  delta_t = [delta_t , spec_t ( j_col_targets (j) +
        target_left_end - 1) - spec_t ( i_col_anchor )];
82                  t1 = [t1 , spec_t ( i_col_anchor )];
83                  nPairings = nPairings + 1;
84              end
85      end
86
87
88 end
89 hashes = [f1; f2; delta_t ; t1];
90 end
```

A.7   hashing

```matlab
1 function [ hashes ] = hashing ( filename )
2 % A function just combining getConstellation , getReducedConstellation and
3 % getHashes for readibility .
4 [ peak_magnitudes , spec_f , spec_t , fs , window ] = getConstellation ( filename );
5 [ reduced_constellation , spec_t , ~] = getReducedConstellation (
     peak_magnitudes , spec_f , spec_t , fs , window );
6 [ hashes ] = getHashes ( reduced_constellation , spec_t , spec_f );
7 end
```

A.8   GiMY_app

```matlab
1 classdef GiMY_app < matlab . apps . AppBase
2
3      % Properties that correspond to app components
4      properties ( Access = public )
5          UIFigure                matlab .ui. Figure
6          Image                   matlab .ui. control . Image
7          Lamp                    matlab .ui. control . Lamp
8          EditStatus              matlab .ui. control . EditField
9          ProgressLabel           matlab .ui. control . Label
10         Percentage5             matlab .ui. control . NumericEditField
11         Percentage4             matlab .ui. control . NumericEditField
12         Percentage3             matlab .ui. control . NumericEditField
13         Percentage2             matlab .ui. control . NumericEditField
14         Percentage1             matlab .ui. control . NumericEditField
```

```matlab
15          SongName5                matlab.ui.control.EditField
16          Label_5                  matlab.ui.control.Label
17          SongName4                matlab.ui.control.EditField
18          Label_4                  matlab.ui.control.Label
19          SongName3                matlab.ui.control.EditField
20          Label_3                  matlab.ui.control.Label
21          SongName2                matlab.ui.control.EditField
22          Label_2                  matlab.ui.control.Label
23          SongName1                matlab.ui.control.EditField
24          Label                    matlab.ui.control.Label
25          SongNameLabel            matlab.ui.control.Label
26          PercentageofMatchLabel   matlab.ui.control.Label
27          RECORDButton             matlab.ui.control.Button
28          GiMYGivemetheSongLabel   matlab.ui.control.Label
29      end
30
31      % Callbacks that handle component events
32      methods (Access = private)
33
34          % Button pushed function: RECORDButton
35          function RECORDButtonPushed(app, event)
36
37              % Clear the song name and percentage value boxes each time the
38              % "RECORD" button is pushed to avoid confusion.
39              app.Image.Visible = "off";
40              app.Lamp.Color = 'g';
41              app.RECORDButton.Enable = "off"; %Make the button unavailable
    while in process.
42              app.SongName1.Value = ' ';
43              app.SongName2.Value = ' ';
44              app.SongName3.Value = ' ';
45              app.SongName4.Value = ' ';
46              app.SongName5.Value = ' ';
47              app.Percentage1.Value = 0;
48              app.Percentage2.Value = 0;
49              app.Percentage3.Value = 0;
50              app.Percentage4.Value = 0;
51              app.Percentage5.Value = 0;
52
53
54              %Start recording the song when the "RECORD" button is pushed.
55              app.EditStatus.Value = sprintf('Recording...');
56              fs = 44100; % Recording sampling frequency
57              recordingTime = 10; % Record for 10 seconds
58
59
60  %          Record the audio by using the "audiorecorder" function. Here,
61  %          we use sampling frequency of fs = 44100Hz, 16bit to represent
62  %          each sample and we record a mono audio since the songs in the
63  %          database are also represented as mono audios.
64
65              RECORDING = audiorecorder(fs,16,1); %Recording the audio
66              recordblocking(RECORDING,recordingTime); %Record via microphone
     for 15 seconds.
67              RECORDED = getaudiodata(RECORDING); %Save the audio data in "
    RECORDED" variable.
68
69
70  %          Convert the data in a '.wav' file. Note that "audiowrite"
    function does
71  %          not support ".mp3" format since r2015b version.
72
73              audiowrite('mic.wav',RECORDED,fs);
74
75              % Create constellation for the recording
76              hashMic = hashing('mic.wav');
77              % Appending the track ID as 0
78              hashMic = [zeros(1,size(hashMic,2)); hashMic];
79
80
81  %          We must access the song database and the names of the songs
```

```matlab
82 %               to compare the recording with the database and display the
      name
83 %               of the matched songs in the app. Therefore, we declare these
84 %               variables here as global variables to use them for the rest
      of
85 %               the algorithm.
86
87
88          global data;
89          global songNames;
90
91          app.Lamp.Color = 'r'; %"RECORD" button is not available while
      processing.
92          app.EditStatus.Value = 'Processing...';
93          drawnow()
94
95          % Defining the match thresholds
96          th_f1 = 3;
97          th_f2 = 3;
98          th_delta_t = 0.050;
99          tic
100         % For each recorded hash make matrix subtractions to find
101         % matching hashes
102         matches_diff = []; % Matching hashes
103         % Search for the matching hashes
104         for i = 1:size(hashMic,2)
105             % Take the difference of each hash in the recording with
106             % the who data.
107             difference = abs(data - hashMic(:,i));
108             % Logical index = 1 if the difference satisfies threshold
109             % conditions. The indices represent the indices of the
110             % matching hashes inside the difference matrix.
111             indices = difference(1,:) > 0 & difference(2,:) < th_f1 &
      ...
112                 difference(3,:) < th_f2 & difference(4,:) < th_delta_t;
113
114             % Hold the values of time diffs and track ids
115             matches_diff = [matches_diff, difference([1,5],indices)];
116         end
117
118         % Among the matching hashes we will look into the absolute time
119         % differences, we expect to observe a constant absolute time
      difference
120         % between the mathing hashes if the song is detected correctly.
121
122         % We will hold the maximum bin. We expect a bin to be highly
      prominent.
123         max_bins = zeros(1,length(songNames));
124         for i = 1:length(songNames)
125             % Select the matches from matches_diff from each track ID.
126             time_diff_track_i = matches_diff(2,matches_diff(1,:) == i);
127             % Group the time diffs into bins. Each time diff is shifted
       by a little
128             % amount so that we wont have values at the edges of the
      bins.
129             [bin_counts, edges] = histcounts(time_diff_track_i,'
      BinWidth',0.040);
130             max_bins(i) = sum(maxk(bin_counts,2));
131         end
132
133         % Among the max_bins choose the most prominent bins.
134         [max_max_bins, track_ids] = maxk(max_bins,5);
135         toc
136         %Find location (index) of the five songs with most matches and
137         %their corresponding matches.
138
139         % Calculate the probabilites
140         percentages = max_max_bins/sum(max_max_bins)*100;
141
142
143 %           If none of the songs in the database has more than 1 percent
```

```matlab
144 %              similarity to the recording, than we conclude that the
145 %              recording does not correspond to any song of the database.
146
147              if max(percentages) < 30
148                  app.EditStatus.Value = sprintf('The song does not have a
     match in the database.');
149                  app.Image.Visible = "off";
150
151
152 %              If at least one song in the database has more than 1 percent
153 %              similarity to the recording, than we conclude that the
154 %              recording is indeed similar enough to some songs in the
155 %              database. In that case, we display the names of the most 5
156 %              similar songs and their corresponding similarity percentages.
157
158              else
159                  app.SongName1.Value = (songNames{track_ids(1)}(1:end-4));
160                  app.SongName2.Value = (songNames{track_ids(2)}(1:end-4));
161                  app.SongName3.Value = (songNames{track_ids(3)}(1:end-4));
162                  app.SongName4.Value = (songNames{track_ids(4)}(1:end-4));
163                  app.SongName5.Value = (songNames{track_ids(5)}(1:end-4));
164                  app.Percentage1.Value = percentages(1);
165                  app.Percentage2.Value = percentages(2);
166                  app.Percentage3.Value = percentages(3);
167                  app.Percentage4.Value = percentages(4);
168                  app.Percentage5.Value = percentages(5);
169                  app.EditStatus.Value = sprintf('The song is successfully
     identified.');
170                  app.Image.Visible = "on";
171              end
172
173              app.Lamp.Color = 'w'; %" RECORD" button returns to standby mode
     .
174              app.RECORDButton.Enable = "on"; %Make the button available
     again.
175          end
176      end
177
178      % Component initialization
179      methods (Access = private)
180
181          % Create UIFigure and components
182          function createComponents(app)
183
184              % Get the file path for locating images
185              pathToMLAPP = fileparts(mfilename('fullpath'));
186
187              % Create UIFigure and hide until all components are created
188              app.UIFigure = uifigure('Visible', 'off');
189              app.UIFigure.Color = [1 1 1];
190              app.UIFigure.Position = [100 100 818 564];
191              app.UIFigure.Name = 'MATLAB App';
192
193              % Create GiMYGivemetheSongLabel
194              app.GiMYGivemetheSongLabel = uilabel(app.UIFigure);
195              app.GiMYGivemetheSongLabel.BackgroundColor = [1 1 1];
196              app.GiMYGivemetheSongLabel.HorizontalAlignment = 'center';
197              app.GiMYGivemetheSongLabel.FontName = 'Calibri';
198              app.GiMYGivemetheSongLabel.FontSize = 48;
199              app.GiMYGivemetheSongLabel.FontColor = [0.0157 0.3373 0.549];
200              app.GiMYGivemetheSongLabel.Position = [161 481 499 67];
201              app.GiMYGivemetheSongLabel.Text = 'GiMY - Give me the Song';
202
203              % Create RECORDButton
204              app.RECORDButton = uibutton(app.UIFigure, 'push');
205              app.RECORDButton.ButtonPushedFcn = createCallbackFcn(app,
     @RECORDButtonPushed, true);
206              app.RECORDButton.BackgroundColor = [1 1 1];
207              app.RECORDButton.FontName = 'Calibri';
208              app.RECORDButton.FontSize = 24;
209              app.RECORDButton.FontWeight = 'bold';
```

```matlab
210            app.RECORDButton.FontColor = [1 0 0];
211            app.RECORDButton.Position = [331 411 158 38];
212            app.RECORDButton.Text = 'RECORD';
213
214            % Create PercentageofMatchLabel
215            app.PercentageofMatchLabel = uilabel(app.UIFigure);
216            app.PercentageofMatchLabel.HorizontalAlignment = 'center';
217            app.PercentageofMatchLabel.FontName = 'Calibri';
218            app.PercentageofMatchLabel.FontSize = 20;
219            app.PercentageofMatchLabel.FontWeight = 'bold';
220            app.PercentageofMatchLabel.FontAngle = 'italic';
221            app.PercentageofMatchLabel.FontColor = [0.0157 0.3373 0.549];
222            app.PercentageofMatchLabel.Position = [608 304 178 40];
223            app.PercentageofMatchLabel.Text = 'Percentage of Match';
224
225            % Create SongNameLabel
226            app.SongNameLabel = uilabel(app.UIFigure);
227            app.SongNameLabel.HorizontalAlignment = 'center';
228            app.SongNameLabel.FontName = 'Calibri';
229            app.SongNameLabel.FontSize = 20;
230            app.SongNameLabel.FontWeight = 'bold';
231            app.SongNameLabel.FontAngle = 'italic';
232            app.SongNameLabel.FontColor = [0.0157 0.3373 0.549];
233            app.SongNameLabel.Position = [285 303 101 40];
234            app.SongNameLabel.Text = 'Song Name';
235
236            % Create Label
237            app.Label = uilabel(app.UIFigure);
238            app.Label.BackgroundColor = [1 1 1];
239            app.Label.HorizontalAlignment = 'right';
240            app.Label.FontName = 'Calibri';
241            app.Label.FontSize = 18;
242            app.Label.FontWeight = 'bold';
243            app.Label.Position = [59 277 25 24];
244            app.Label.Text = '1.';
245
246            % Create SongName1
247            app.SongName1 = uieditfield(app.UIFigure, 'text');
248            app.SongName1.FontSize = 16;
249            app.SongName1.Position = [99 279 473 25];
250
251            % Create Label_2
252            app.Label_2 = uilabel(app.UIFigure);
253            app.Label_2.HorizontalAlignment = 'right';
254            app.Label_2.FontName = 'Calibri';
255            app.Label_2.FontSize = 18;
256            app.Label_2.FontWeight = 'bold';
257            app.Label_2.Position = [59 235 25 24];
258            app.Label_2.Text = '2.';
259
260            % Create SongName2
261            app.SongName2 = uieditfield(app.UIFigure, 'text');
262            app.SongName2.FontSize = 16;
263            app.SongName2.Position = [99 237 473 26];
264
265            % Create Label_3
266            app.Label_3 = uilabel(app.UIFigure);
267            app.Label_3.HorizontalAlignment = 'right';
268            app.Label_3.FontName = 'Calibri';
269            app.Label_3.FontSize = 18;
270            app.Label_3.FontWeight = 'bold';
271            app.Label_3.Position = [59 192 25 24];
272            app.Label_3.Text = '3.';
273
274            % Create SongName3
275            app.SongName3 = uieditfield(app.UIFigure, 'text');
276            app.SongName3.FontSize = 16;
277            app.SongName3.Position = [99 194 473 27];
278
279            % Create Label_4
280            app.Label_4 = uilabel(app.UIFigure);
```

```matlab
281             app.Label_4.HorizontalAlignment = 'right';
282             app.Label_4.FontName = 'Calibri';
283             app.Label_4.FontSize = 18;
284             app.Label_4.FontWeight = 'bold';
285             app.Label_4.Position = [59 153 25 24];
286             app.Label_4.Text = '4.';
287
288             % Create SongName4
289             app.SongName4 = uieditfield(app.UIFigure, 'text');
290             app.SongName4.FontSize = 16;
291             app.SongName4.Position = [99 152 473 25];
292
293             % Create Label_5
294             app.Label_5 = uilabel(app.UIFigure);
295             app.Label_5.HorizontalAlignment = 'right';
296             app.Label_5.FontName = 'Calibri';
297             app.Label_5.FontSize = 18;
298             app.Label_5.FontWeight = 'bold';
299             app.Label_5.Position = [59 111 25 24];
300             app.Label_5.Text = '5.';
301
302             % Create SongName5
303             app.SongName5 = uieditfield(app.UIFigure, 'text');
304             app.SongName5.FontSize = 16;
305             app.SongName5.Position = [99 110 473 25];
306
307             % Create Percentage1
308             app.Percentage1 = uieditfield(app.UIFigure, 'numeric');
309             app.Percentage1.FontSize = 16;
310             app.Percentage1.Position = [647 283 100 22];
311
312             % Create Percentage2
313             app.Percentage2 = uieditfield(app.UIFigure, 'numeric');
314             app.Percentage2.FontSize = 16;
315             app.Percentage2.Position = [647 241 100 22];
316
317             % Create Percentage3
318             app.Percentage3 = uieditfield(app.UIFigure, 'numeric');
319             app.Percentage3.FontSize = 16;
320             app.Percentage3.Position = [647 199 100 22];
321
322             % Create Percentage4
323             app.Percentage4 = uieditfield(app.UIFigure, 'numeric');
324             app.Percentage4.FontSize = 16;
325             app.Percentage4.Position = [647 154 100 22];
326
327             % Create Percentage5
328             app.Percentage5 = uieditfield(app.UIFigure, 'numeric');
329             app.Percentage5.FontSize = 16;
330             app.Percentage5.Position = [647 112 100 22];
331
332             % Create ProgressLabel
333             app.ProgressLabel = uilabel(app.UIFigure);
334             app.ProgressLabel.HorizontalAlignment = 'right';
335             app.ProgressLabel.FontName = 'Calibri';
336             app.ProgressLabel.FontSize = 18;
337             app.ProgressLabel.FontWeight = 'bold';
338             app.ProgressLabel.FontAngle = 'italic';
339             app.ProgressLabel.FontColor = [0.0157 0.3373 0.549];
340             app.ProgressLabel.Position = [143 50 75 24];
341             app.ProgressLabel.Text = 'Progress:';
342
343             % Create EditStatus
344             app.EditStatus = uieditfield(app.UIFigure, 'text');
345             app.EditStatus.HorizontalAlignment = 'center';
346             app.EditStatus.FontName = 'Calibri';
347             app.EditStatus.FontSize = 18;
348             app.EditStatus.Position = [246 47 430 27];
349             app.EditStatus.Value = 'Waiting for a song';
350
351             % Create Lamp
```

```matlab
            app.Lamp = uilamp(app.UIFigure);
            app.Lamp.Position = [513 415 31 31];
            app.Lamp.Color = [1 1 1];

            % Create Image
            app.Image = uiimage(app.UIFigure);
            app.Image.Visible = 'off';
            app.Image.Position = [19 273 42 37];
            app.Image.ImageSource = fullfile(pathToMLAPP, 'Hand-drawn-gold-
    crown-Clipart-PNG.png');

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = GiMY_app

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```