Advances in Science, Technology and Engineering Systems Journal
Vol. 6, No. 2, 846-859 (2021)
www.astesj.com
Special Issue on Multidisciplinary Sciences and Engineering

ASTES Journal
ISSN: 2415-6698

Application-Programming Interface (API) for Song Recognition Systems

Murtadha Arif Bin Sahbudin[*],Chakib Chaouch, Salvatore Serrano, Marco Scarpa

Department of Engineering, University of Messina, Messina, 98166, Italy

ARTICLE INFO

Article history:
Received: 25 December, 2020
Accepted: 09 February, 2021
Online: 04 April, 2021

Keywords:
Song Recognition
Mobile API
Signal Processing
Database Clustering
IoT

ABSTRACT

The main contribution of this paper is the framework of Application Programming Interface (API) to be integrated on a smartphone app. The integration with algorithm that generates fingerprints from the method ST-PSD with several parameter configurations (Windows size, threshold, and sub-score linear combination coefficient). An approach capable of recognizing an audio piece of music with an accuracy equal to 90% was further tested based on this result. In addition the implementation is done by algorithm using Java's programming language, executed through an application developed in the Android operating system. Also, capturing the audio from the smartphone, which is subsequently compared with fingerprints, those present in a database.

1 Introduction

An audio representation includes a recording of a musical piece's output. Digital sound recordings are based on the analog audio signal being sampled. Sampling is achieved by capturing the signal amplitude at a specified sampling rate and storing them in binary format. In terms of recording efficiency, the sampling rate and the bit rate (number of bits used to store each sample) are the two most important variables. Audio CDs use a 44.1 kHz sampling rate or 44,100 samples per second, and each sample uses 16 bits, which mainly an industry standard.

Common audio streaming sites host millions of audio files, and thousands of broadcast stations transmit audio content at any given time. The ever-increasing amount of audio material, whether online or on personal devices, generates tremendous interest in the ability to recognize audio material. It achieves this by using identification technology that seeks to work at the highest degree of accuracy and specificity.

Songs recognition identifies a song segment either from a digital or an analog audio source. Song rankings are based on radio / TV broadcasting or streaming; copyright protection for songs or automatic recognition of songs that a person wishes to identify while listening to them are different applications of such a system. Important information such as song title, artist name, and album title can be provided instantly. To create detailed lists of the particular content played at any given time, the industry uses audio fingerprinting systems to monitor radio and TV broadcast networks. Through automatic fingerprinting devices, royalties' processing relies on the broadcasters who are required to produce accurate lists of content being played.

Given the high demand for an application, several approaches have already been studied based on song fingerprinting recognition, such as [1]–[4] and [5]. Nowadays, the state of the art of recognition techniques are those developed by Shazam [6], [7] and SoundHound [8], and the detection system by [9]. These services are widely known for their mobile device applications.

Audio streams of many broadcast channels or recordings of different events are typically analyzed using fingerprint systems for media monitoring. As these systems work on massive quantities of data, the data models involved should be as small as possible, while the systems need to efficiently run on massive and growing reference databases. Besides, high robustness criteria are determined by the application for media monitoring. Although the sensitivity to noise may not be the primary concern for this use case, the systems need to identify audio content that different effects may have changed.

Android is the most popular mobile operating system globally, despite the presence on the world market of the likes of Apple iOS. It is mainly used for smartphones and tablets, but thanks to its characteristics, it is also extended to other devices, such as laptops, cameras, and IoT devices. Developing applications requires Android Software Development Kit (SDK), which contains all the tools to create and run new software, such as debuggers, libraries,

[*]Corresponding Author: Murtadha Arif Bin Sahbudin, Department of Engineering, University of Messina, Email: msahbudin@unime.it

and emulators. The integrated development environment (IDE) officially supported for this purpose is Android Studio, released and available for free from the official developer site for Android and the SDK. Applications are Java-based and are distributed through self-installing packages, i.e., Android application package (APK) files (a variant of the format JAR), which contain all the components and resources of the created software, including source code, XML, images, and binary files.

In particular, this research deals with the process and the theoretical notions that lead to the generation of linekeys previously mentioned in [10], and [11]. Music recognition is a method of identifying a segment of an audio signal from a digital or analog source; this process uses the power spectral density (PSD) to process the acquired data to obtain complete information about the audio signal source. The fingerprint generation takes place on the client-side, in this case, the Android mobile application. The communication protocol, shown in Fig. 1, depicts the process of communication and interfacing with the recognition server with database collection.

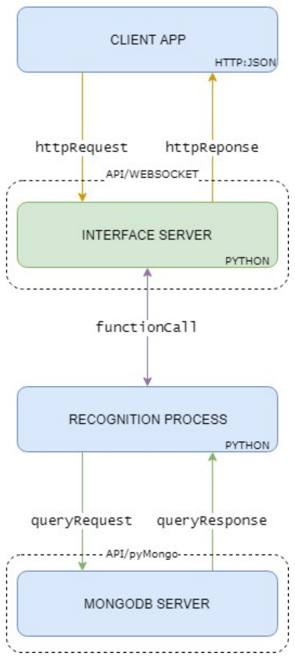


Figure 1: Client and server protocol interface

2 Challenges

The number of songs in the music industry has recently increased significantly, according to a report in [12]. With massive databases, the management and identification of songs using a conventional relational database management system have become more difficult. For large datasets, a common linear search technique that checks the existence of any fingerprint in an array one at a time has a noticeable decrease in efficiency [13]. The stored information, therefore, needs a scalable database system to meet the execution time, memory use, and computing resources for recovery purposes, which is suggested in [14].

Song recognition systems usually operate on vast amounts of data and are expected to meet several robustness requirements depending on the actual use case. Robustness to different kinds of lossy audio compression and a certain degree of noise would seem to be the minimum requirement. Systems designed to detect short audio segments' microphone recordings involve high background noise robustness, such as noise and distortion, or even multiple songs played in the surrounding.

It is crucial to have robust and quick recognition for effective song information retrieval. Major consumers need details about trending tracks, airtime schedules, and song versions, such as music labels, manufacturers, promoters, and radio stations. They, therefore, demand an application that is capable of generating information that is fast and precise.

In the field of real-time song recognition, the entertainment industry, particularly in music, the extensive collection of digital collections, and the commercial interest are opening new doors to research. In 2017, the global digital music industry expanded by 8.1 percent, with total revenues of US$ 17.3 billion, according to IFPI 's Global Music Report 2018 [15]. For the first time in the same survey, 54 percent of the revenue alone comes from digital music revenue.

However, the most challenging application for bringing new songs to listenership is still the FM frequency radio station. The FM frequency channel for music broadcasting in European countries is still actively reliant on radio stations [16]. Radio stations and music companies have been working to advance music industry data analytics by creating ways of analyzing broadcast songs through new services and platforms.

It is an interest to broadcasters and advertisers to measure radio audience size and listening patterns over a broadcast radio station to achieve a source of revenue [17]. However, based on demographics and psychographics (psychological criteria) of the target audience of the station, variations in the region of station promotional material can be predicted [18].

In addition to robustness criteria, the seriousness or effect of incorrect results must be considered, and the necessary performance recognition characteristics of fingerprinting systems must be taken into account. For instance, if a song recognition system is used, an unidentified match is missing, the user can waste storage space. However, on the flip side, a specific song recognized as false match systems that report false positives should be avoided.

Most critically, false positives are expensive for large-scale media monitoring; revenue might be attributed to the wrong artist. False negatives, another form of error, may lead to hours of unidentified material that will have to be checked with manual effort. Any form of error would increase the maintenance cost of a system.

To overcome it, we have proposed a more scalable big data framework using fingerprint clustering. Besides, a new recognition algorithm also was required for the new clustered collection. We also compared the performance from both the legacy system (non-clustered) and the new clustered database.

We define extensions of scale modifications that are likely to be encountered when developing a framework for monitoring FM radio broadcasting stations—investigating our dataset of the reported output of radio segments through the percentage of accuracy. This estimation will serve as the appropriate gold standard throughout this study, i.e., a device should be robust to at least this range of scale noise but may be needed to cope with even more severe distortions.

3 Problem Statement

Fingerprint databases, recognition process instances, and FM transceivers are the legacy system in commercial use. The problems of an audio recognition system in normal use can be described from the following aspects:

a) Near Similarity: This occurs when virtually the same audio fingerprints are produced by two or more perceptually different audio recordings, leading to serious problems in the recognition process. Therefore, the key goals when developing an audio fingerprinting algorithm are to keep the probability of collision as minimal as possible.

b) In-variance to noises and spectral or temporal distortions: Audio signal is usually degraded to some degree due to some kinds of sounds and vibrations when captured or playing in actual environments. The audio fingerprints of the damaged audio signal can be the same as those of the original signal. Important features are still unchanged. In cases of this fingerprinting technique, high robustness must be obtained.

c) Minimum length of song track needed for identification: Due mainly to time and storage limitations, making the entire of an unknown audio track in real-time music recognition is still impractical. Nevertheless, it is ideal that only a few seconds of the track is required to find the unknown audio.

d) Retrieval speed and computing load: Recognition results can be provided in a few seconds in most real-time applications. However, with the increase in song recordings in the audio reference database, locating the matching object correctly in real-time becomes very difficult.

A fingerprint is a type of distinct digital representation of the waveform of a song. A fingerprint can be obtained by collecting significant characteristics from the various audio properties. Without sacrificing its signature, the created fingerprint may also be segmented into several parts. Moreover, fingerprints can be processed at a much smaller scale relative to the audio waveform's initial form.

The items below are several criteria that should take into consideration for a robust audio fingerprint:

- Consistent Feature Extraction: The key feature of fingerprint generation is that it can replicate an audio fingerprint identical to that of a music section.

- Fingerprint size: Fingerprint file size has to be small enough so that more music collections can be stored in the database. In reality, a lightweight fingerprint offers effective memory allocation during processing.

- Robustness: Even if external signal noise has affected the source audio, fingerprints may be used for identification.

4 Related Works

We provided our outstanding contributions to the academic literature that satisfy all the success criteria listed above. We show that, despite our large comparison sets, our method is efficient and that there is an extensive search problem caused by the invariances of the hashes and their robustness in signal modifications. Mainly, we designed the proposed device for low-cost hardware, demonstrated its capabilities, and avoided costly CPU processing.

Despite studies on the identification of songs and fingerprints by other researchers such as Shazam [7] and SoundHound [8]. We understand the clustering design using K-means for an experiment in the real fingerprint database with a set of 2.4 billion fingerprints provided by the company as datasets. We want to emphasize that a database of this scale seldom appears in the research literature, but there is a chance that it exists. The next critical aspect was the audio recognition system. Although initial K-means computing for compilation is resource-intensive, we achieved significant speed efficiency at the end of the day [19]. Also, the proposed architecture and algorithm will lead to a new insight into song recognition.

Moreover, we had introduced an IoT-based solution to song recognition in a cloud environment in this study. We have developed a recognition system to integrate audio streams from remote FM Radio stations [20]. We conducted a song recognition technique based on the K-modes clustered cloud database of MongoDB [11]. We supported various collections of fingerprint length tests to ensure the best accuracy and reliability of the test.

The new fingerprint extraction technique's significant findings focused on Short Time Power Spectral Density (ST-PSD) was also implemented [10]. Later of which binary encoding group's attributes lead to the reliability of K-modes. Besides, this study clarified the identification methodology primarily through hamming distance measure in the predetermined cluster table. The findings were given by sampling 400 random 5-second queries from the initial song set in the experiment. Using this method, the optimal chosen combination of parameters is the identification ratio of 90%.

We have already introduced extracting fingerprints from audio in our previous works based on the ST-PSD calculation. We introduce several significant and remarkable improvements to the previously proposed algorithm to enhance the robustness of the linekeys to temporal shift and the consistency of the fingerprints for perceptually distinct audio signals in this paper. The proposed fingerprints are based on calculating the audio signal's short time spectral power density ST-PSD obtained on the Mel frequency scale.

According to our tests and performance measurements, the framework can be used specifically for different areas of fingerprinting applications. In addition to typical fingerprint applications, these are, for instance, the identification of audio copies and media tracking, copyright identification of songs.

5 K-means Clustering in MongoDB Database Methodology

K-means clustering is used for non-classified results, which performs an unsupervised algorithm for many data. The fingerprints are grouped into subgroups by the K-means classification. As such, objects in the same category (clusters) are more similar to each other. Whereas K-means is often used for high-dimensional data classification, we take advantage of the centroid value as a distance point when executing the nearest computation in this experiment. The basis for using K-means is that we need to create relatively uniform dataset-size clusters.

The K-Means clustering is method of partitioning n data, into k

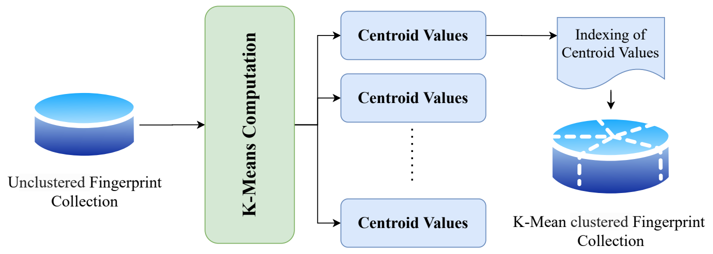
clusters in which each data is associated with the cluster with the nearest mean (intra-cluster distance). Thus, several different distinct clusters are formed. Therefore, the primary goal of K-means is to minimize intra-cluster distances. This is done by determining the J index as Equation (1) follows:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n_j} ||x_i^{(j)} - c_j||^2 , \tag{1}$$

where c_j is the mean value of the j-th cluster with $i \leq j \leq n$, and $x_i^{(j)}$ represents a fingerprint that falls into the j cluster. The c_j is usually called *centroid*s. Sets of k clusters, $k+1$ boundaries and k centroids are discovered by the K-means algorithm, reducing the J optimization index. More precisely, a fingerprint partition set of $S = \{S_1, S_2, \dots, S_k\}$ resulting from fingerprint partitions is computed from:

$$\underset{S}{\operatorname{argmin}}\, J \tag{2}$$

As reference points, the c_j centroids of the clusters are used. Nevertheless, we are expected to specify the number of k clusters subsequently computed. Figure 2 provides an overview of the clustering phase in this implementation, where the original fingerprint collection stored in MongoDB was initiated.



Unclustered Fingerprint Collection
K-Means Computation
Centroid Values
Centroid Values
Centroid Values
Indexing of Centroid Values
K-Mean clustered Fingerprint Collection

Figure 2: K-means clustering implementation overview

A reduced data sample of $2 \cdot 10^6$ (random segment) was selected from the overall selection based on the fingerprint distribution analysis. There is no particular data restriction that the K-means can carry out. However, this depends on computational power, time constraints, and other hardware specification.

Furthermore, once the calculation is completed, the centroids' values are generated and represent a key value for the MongoDB cluster array. Next, the nearest distance fingerprints were determined to the centroid value and transferred the data to the subgroup collections.

5.1 *Stepwise Implementation of K-means*

The experimentation aimed to produce several $k = 10,000$ clusters with key centroids values as reference. In the next phase, song fingerprints are distributed into the new clustered collections.

Step 1: Batch Data Processing

The original approach was to separate the first instances into blocks by introducing a segmentation of data while handling many data for K-mean computation. Next, using the *pickle* data structure in python, the trained model was stored from the computation. Subsequently, once all fingerprint datasets were collected, we used the previous model values and incrementally updated them. Code Listing 1 shows the steps of the python *scikit-learn* library [21] implementation algorithm for K-means. Below describe the steps of the data training sequence:

1. As the first elements of *n_cluster* centres, the algorithm selects 10,000 points.
2. Then, each 100,000 of linekeys that were loaded into memory and loaded to the K-means calculation using these data. Each cluster center was recomputed as the average of the points in that cluster.
3. Finally, the function at item 2 is repeated until the clusters converge. If there is no further change in the assignment of the fingerprints to clusters, the algorithm converges.

```
def dumpFitKmean(n_clusters,i,chunk,chunksize):
if i==1:
X = chunk
mbk = KMeans(init='K-means++',n_clusters=int(
    n_clusters), n_init=1, random_state=42)
else:
X = chunk
mbk = getFitKmean(i-1)
mbk.fit(X)
pickle.dump(mbk,open("pickelDump"+str(i)+".p", "wb"))

def getFitKmean(i):
dumpFile = "pickelDump"+str(i)+".p"
mbk = pickle.load(open(dumpFile, "rb"))
return mbk

if __name__ == "__main__":
i=1
chunksize = 100000
for chunk in pd.read_csv(filename, chunksize=chunksize
    , header=None):
dumpFitKmean(n_clusters,i,chunk,chunksize)
i=i+1
```

Listing 1: A Python implementation of K-means computation

Step 2: Building Cluster Segmentation in MongoDB

Once the centroid values were formed, the distance to each centroid was calculated for each fingerprint $f^{(i)}$ in order to find its partition affiliation by Equation (3)

$$p^{(i)} = \underset{j=1}{\overset{k}{\operatorname{argmin}}}\, ||f^{(i)} - C_j|| \tag{3}$$

In order to obtain the distance for the fingerprint $f^{(i)}$, the Euclidean distance was performed between $f^{(i)}$ and any centroid value C_j. From the distance function list, argmin was used to get the minimum distance. As a result, acquired the fingerprint association $p^{(i)}$ with its corresponding cluster $S_{p^{(i)}}$.

The code in Listing 2 demonstrates the steps of the algorithm implemented in python to obtain an aggregation of fingerprint-clusters.

```
def centroid_linekeys_distance(self,linekeys,
    indexCentroids,centroids):
distance = abs((int(linekeys)-int(centroids)))
```

```
result = distance ,centroids ,indexCentroids
return result

def get_CentroidsDB(self):
result = self.collection_centroids.find()
result = [[document['_id'],document['centroids']] for
     document in result]
result = sorted(result,key=lambda x: x[1])
return result

def update_CentroidsDB(self,sortedDistance,songNo,
     linekeys,linekeysPosition):
self.collection_updateCentroids = self.db['
     zentroid2song'+str(centroidIndex)]
self.collection_updateCentroids.insert({'SongNo':
     songNo, 'linekeys':linekeys, 'linekeysPos':
     linekeysPosition })
```

Listing 2: Python code fingerprint-cluster association

5.2 Song Recognition and Information Retrieval

During the recognition phase, the fingerprint sequence in the clustered fingerprints database must be identified. The algorithm performs a sequential search window in two stages. First, allocate each fingerprint in the query sequence to the nearest centroid value. Then, perform an in-depth search within the corresponding cluster values and obtain a set of candidates. Here, by applying a similar algorithm to the clustering method to ensure the accuracy of the result.

The benefit of the implemented cluster framework is that a binary search technique can be used according to [22]. The first step is to find the position of a specific fingerprint query value within the sorted array. This method examines and searches the nearest key value for the median centroid key value of the selected fingerprint in each step.

5.2.1 Real-Time Slide Window

Figure 3 illustrates the method used to classify a song based on an input fingerprint source. As seen, the fingerprint query stream was chunked into significant portions of the windows. This sliding window approach is a common information retrieval technique used to detect matching sequences. Each window represents a part of the time in the current album. As mentioned above, each fingerprint represents an instant δ of the audio source. Configurable fingerprints window sizes, e.g. 500, 1000, 2000, 3000, 5000 or 6000 which translate into the actual time section, are essential. Here, it is suggested that the window size does not exceed 6000 fingerprints, as this represents around 60 seconds of actual audio.

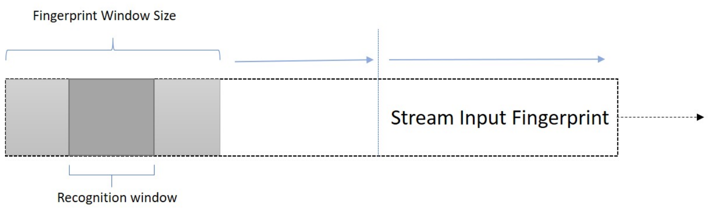


Figure 3: Fingerprint Windows Slides Recognition

Each fingerprint is not unique, so that it could appear in some songs, and then a set of song candidates will be provided. However, in the quest for a result, we can exclude those candidates and obtain a winner. It was next, segmenting a sub-window of 3/4 from the fingerprints used in the window. This sub-segment window aims to reduce the time for recognition. Also, the selected fraction size weight is appropriate to reflect the full fingerprint in the window.

5.2.2 Fingerprint Cluster Identification and Recognition

There are two stages of the recognition process for each fingerprint query for the search's effective result. Firstly, to determine the cluster $p^{(i)}$ representing the nearest centroid as in Equation (4) . The value of the fingerprints $q^{(i)}$ is compared to the value of each centroid C_j. Therefore, returns the cluster $p^{(i)}$ from the set of partitions to get the nearest reference.

$$p^{(i)} = \operatorname*{argmin}_{j=1}^{k} ||q^{(i)} - C_j|| \tag{4}$$

Finally, Equation (5) returns the minimum value *argmin* between the fingerprint query $q^{(i)}$ with the fingerprint set $f_j^{p^{(i)}}$ within the defined cluster $p^{(i)}$. This results in a single result of $wf^{(i)}$ of the song details (title).

$$wf^{(i)} = \operatorname*{argmin}_{j} ||q^{(i)} - f_j^{p^{(i)}}|| \tag{5}$$

Each winner of the fingerprint $wf^{(i)}$ will be associated with a set of songs stored in the collection containing the $wf^{(i)}$ in their sequence. For these songs, the column list entry is set to "1" with an entry for each song to be recorded. For this reason, the following subsection 5.2.3 will be clarified by maintaining a record of N columns forming a matrix called **songs**.

Code Listing 3 shows the steps implemented in python for fingerprint recognition.

```
def centroid_linekeys_distance(linekeys,indexCentroids
     ,centroids):
distance = abs((linekeys-centroids))

def get_linekeys_match(centroidIndex,linekeys):
collection_centroidsValues = db['zentroid2song'+str(
     centroidIndex)]
result =  collection_centroidsValues.find({'linekeys':
     linekeys})

if __name__ == "__main__":
for lk in inputLinekeys:
sortedDistance = min(distance)
```

Listing 3: Python fingerprint recognition

5.2.3 Candidates Scoring

From the previous step, the information for each song was obtained from a single fingerprint question. However, it is crucial to assess the overall result inside the fingerprint sequence in the defined window. Therefore, using the Equation (6) to determine the frequency of $F_k^{(n)}$ of the song results in a N row fingerprint window. The highest score value of the song would be as winning candidates for the window section.

$$F_k^{(n)} = \sum_{i=1}^{N} \textbf{songs}_{k,i} \tag{6}$$

where n is the index of the current window. Thus evaluating the winner song using Equation (7).

$$ws^{(n)} = \underset{k}{\operatorname{argmax}}\, F_k^{(n)} \tag{7}$$

As an initiation a counter is set to the value 1.Each winning song that is $ws^{(n)} = ws^{(n-1)}$ will increase the counter value. Therefore, as soon as the winning song $ws^{(n)} \neq ws^{(n-1)}$ evaluates the W number of subsequent windows for the same winner it will reset the counter back to 1. Therefore the length of the airtime song is evaluated as $d = N \cdot W \cdot \delta$.

6 IoT Based Song Recognition

As shown in Figure 4, the system framework is planned to access the FM Frequency source at a different location by deploying a low-cost IoT system receiver. This FM receiver streams the audio to a cloud instance that converts it to an audio file that is then processed for recognition. Each module of software and hardware configuration components must be integrated to achieve this implementation. Each of these components is discussed in the next section, which includes the IoT system 6.1, the communication protocol 6.2, the recognition server 6.3, and the clustered database.

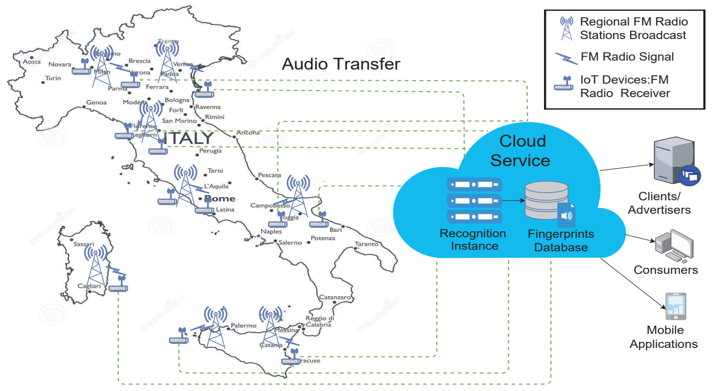


Figure 4: Overview of IoT based Song Recognition Framework

6.1 IoT Device Design

Software-defined radio (SDR) offers a level interface that allows access to filters, mixers, amplifiers, modulators/demodulators on software, meaning on computer embedded systems. This implementation uses RTL-SDR USB dongles based on the RTL2832U chipsets [23] that can read frequencies between 24 and 1,766MHz. The Raspberry Pi is a single-board computer made on a single board and, therefore, small and cheap. It can be used for light programming or, as in this project, to create devices dedicated to the Internet of Things or home automation with various sensors. The operating system used in the project is Raspbian, an official distribution of Raspberry Pi, based on Debian Linux and suitably adapted to the Raspberry Pi. The operating system was downloaded via NOOBS to a 16GB MicroSD.

The FM radio receiver is built using a Raspberry Pi and a dongle, which converts the analog audio signal into a digital audio stream. Python libraries were used for the application, more specifically, the library to launch RTL-FM. The python syntax with the options required to play an FM radio station is shown in Listing 4.

```
import subprocess, signal, os
def newstation(station):
global process, stnum

part1 = "rtl_fm -f "
part2 = "e6 -M wbfm -s 200000 -r 44100 | aplay -r
    44100 -f S16_LE"
cmd = part1 + station + part2
print 'Playing station :', station

# kill the old fm connection
if process != 0:
process = int(subprocess.check_output(["pidof","rtl_fm
    "] ))
print "Process pid = ", process
os.kill(process,signal.SIGINT)

# start the new fm connection
print cmd
process = subprocess.Popen(cmd, shell=True)

def setvolume(thevolume):
os.system('amixer sset "PCM" ' + thevolume)
print 'volume = ' , thevolume

process = 0

while True:
answer = raw_input("Enter a radio station (i.e. 107.9)
     or volume (i.e. 50%): ")
if answer.find('%') > 0:
setvolume(answer)
else:
newstation(answer)
```

Listing 4: Python RTL-FM Frequency Configuration

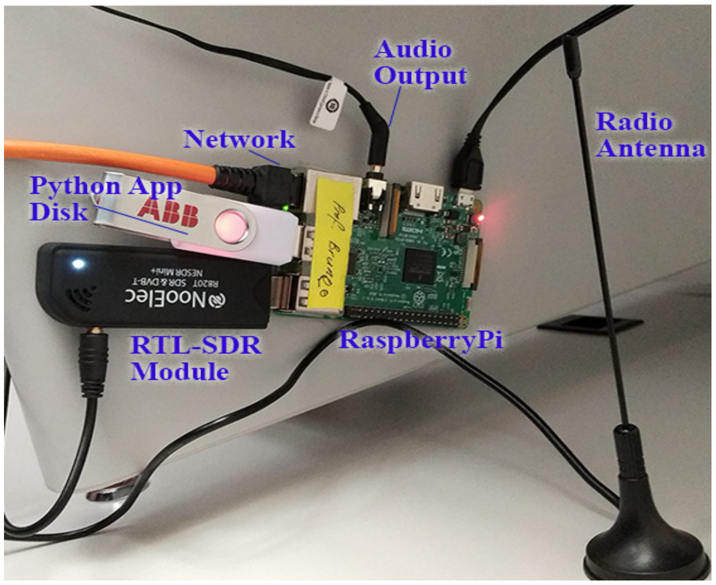


Figure 5: Raspberry Pi with RTL-DSR module Hardware

Figure 5 shows the actual physical Raspberry Pi device configured with the RTL-SDR hardware module.

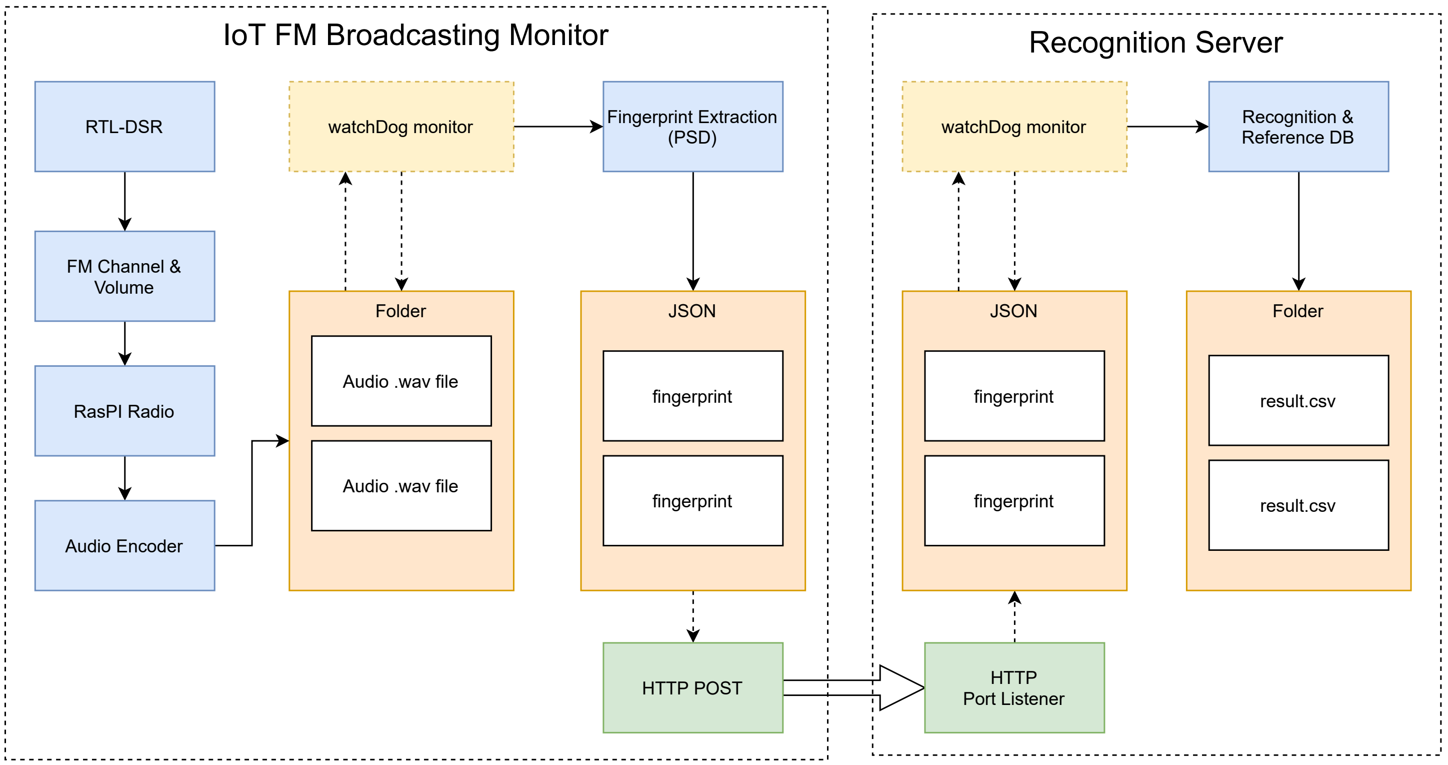


Figure 6: Overview of Framework of IoT device protocol to Recognition Server

6.2 Communication Protocol

Once established the FM frequency channel, we stream the Raspberry Pi output using Real-Time Streaming Protocol (RTSP) with a specific TCP/IP port to maintain end-to-end connection using a python RTSP libraries [24]. The RTSP server will process incoming requests for streams.

Figure 6 shows the component of the Raspberry Pi IoT device and the recognition server communication protocol. The IoT server's final packets consisted of 5 seconds of fingerprint conversion, which is encapsulated as a JSON string as a packet. Additionally, the audio conversion fingerprint algorithm is embedded inside the IoT device; thus, a smaller size packet of string could be transferred much faster. Therefore, the recognition of the fingerprint will be done on the server instance once receiving the packets. The listening port is open as a service for both sides with a directory observer to monitor any incoming packets as demonstrated in code Listing 5.

```
def postJson(linekeys,newFilename):
print "posting"
data = {}
data['fingerprintFile'] = newFilename
data['linekeys'] = map(str,linekeys.tolist())
print data

req = urllib2.Request('http://localhost:8008')
req.add_header('Content-Type', 'application/json')
response = urllib2.urlopen(req, json.dumps(data))

class Watcher:
DIRECTORY_TO_WATCH = "/home/pi/Desktop/PiRadio/
    inputfile/"
```

```
def __init__(self):
self.observer = Observer()

def run(self):
event_handler = Handler()
self.observer.schedule(event_handler, self.
    DIRECTORY_TO_WATCH, recursive=True)
self.observer.start()
try:
while True:
time.sleep(5)
except:
self.observer.stop()
print "Error"
self.observer.join()
```

Listing 5: HTTP Protocol of Fingerprint Exchange

6.3 Recognition Server

A fingerprint is a digital vector that identifies a piece of song signal and can outline the content of that piece of the song. The fingerprint is achieved by removing the main characteristics from audio by choosing distinctive features. Also, fingerprints for storage will minimize the size of the original song with the standard structure.

In this phase, as shown in Listing 6, the streaming audio is converted to *.wav* file in such a way it could be further converted into a sequence of fingerprints.

```
part1 = "rtl_fm -f "
part2 = "e6 -M wbfm -s 200000 -r 44100 | sox -t raw -e
     signed -c 1 -b 16 -r 44100 - inputfile/record"+
    str(numberCount)+".wav"
cmd = part1 + station + part2
```

```
def getArrayKey(filename, noOfSplit, windowlength, step):
fftsize=128
yFinal=[]
D, fs = sf.read(filename)
r = D.size
c = 0
if c > 1:
D = multiToMono(D)
D = np.array_split(D, noOfSplit)
```

Listing 6: Conversion Recording Stream to Audio .wav

The features below are the qualities of an audio fingerprint that we take into consideration:

- Consistent Feature Extraction: The essential part of the fingerprint generation is that it can reproduce a similar audio fingerprint given only a segment of the audio.
- Fingerprint size: The fingerprint's size has to be small enough, so a more exhaustive song collection can be archived in the database. Furthermore, a well-compressed fingerprint uses efficient memory allocation during processing.
- Robustness: Fingerprint can be used for recognition even if the source audio has been affected by external signal noise.

7 Mobile Music Recognition App

The Android application detects and captures audio through the device's microphone, the audio signals and then converts them into fingerprints. The application was developed and compatible with API version 19 (KitKat) and earlier version. It comprises five object classes that includes *FFT, PSD, Fingerprint, MainActivity, RecordAudio, and ToolsAudio*. The following will discussed on class *MainActivity* which provides the function *RecordAudio*, as in the UML Figure 7

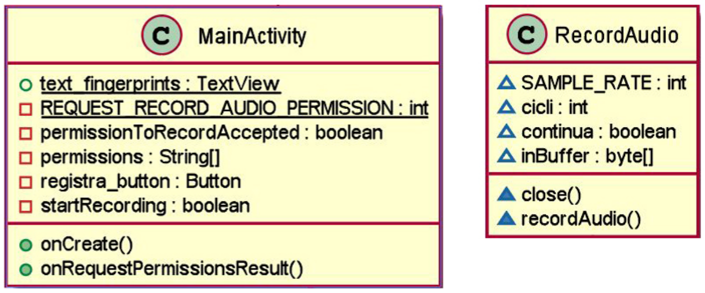


Figure 7: Class MainActivity for RecordAudio

The class *MainActivity* is identified as *Activity*, which is essentially a window containing the application user interface, consisting of a file XML relative to the layout displayed from a class.

Java programming is used to define their behavior; its purpose is to interact with users and goes through various standard functionality cycles, as in Figure 8. When an activity runs, three methods are invoked to interact directly with the user: *onCreate, onStart, onResume*; when instead Android puts the activity to rest, the methods invoked are *onPause, onStop, onDestroy*.

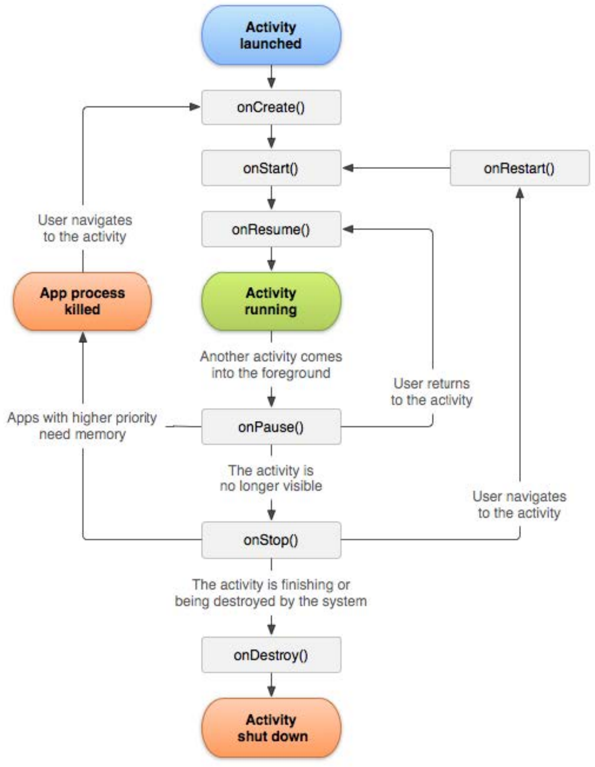

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 8: Android Java classes function cycle

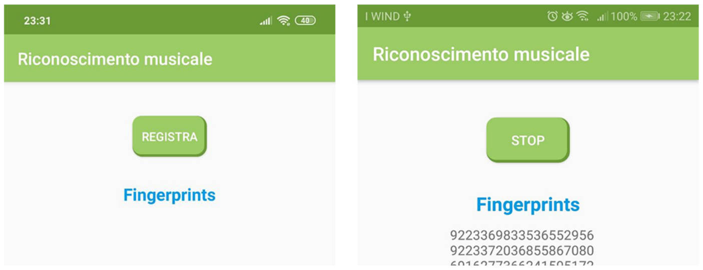
Therefore, the graphic interface interacts with the class provided by *MainActivity*; which overrides its default method internally, *onCreate* is invoked when the activity is created.

To manage the user's interaction with the button, it requires to associate a listener method defined as *onClickListener* , through the method *setOnClickListener*.The class *onClickListener* is an abstract however once an instance of it has been created, the method has been redefined through *onClick*.

The *onClick* as in Listing 7, will provide instructions to be executed when the button is clicked. The process flow implemented is as follows: when the "Register" button is press, as shown in Figure 9, the variable is checked *startRecording* to check whether the registration service is started or not. Therefore, the method will be start *recordAudio* of the class *RecordAudio* to record audio signals. Next, the label "Stop" button initiates the stop service. This basic function is almost similar to Shazam's mobile app.

```
register_button.setOnClickListener (new View.
    OnClickListener(){
 public void onClick(View v){
  if (startRecording) {
   au.recordAudio();
   register_button.setText ("Stop");}
  else {
   au.close();
   register_button.setText("Register");
  }
  startRecording =! startRecording;}
});
```

Listing 7: Java button.setOnClickListener



9223369833536552956
9223372036855867080

Figure 9: User interface for Start and Stop recoding Audio

The final method implemented was the *requestPermissions*, which allows the application to request user permission for using the internal microphone. This method is related to *onRequestPermissionResult*, which manages the results granted the use of the physical module from the mobile.

7.1 Audio Recording

The implementation of audio detection is done by the class RecordAudio. For this purpose, the use of libraries is particularly important:

- *android.media.MediaRecorder*, for recording an audio stream;
- *android.media.AudioRecord*, to manage audio resources;
- *android.media.AudioFormat*, to access audio formats and channel configurations.

The method *recordAudio* as shown in Listing 8, it is invoked by the *MainActivity* as previously described. The invocation will create a thread that initially does the following:

- the priority of the thread for the acquisition of audio signals is set;
- the bufferSize, the maximum size of bytes that the thread can detect at each cycle;
- an object *AudioRecord* is created, which sets different specifications, such as the audio source data (DEFAULT that is the microphone), the sampling frequency (44100), the number of channels (MONO), the coding PCM (16 bit), and the bufferSize;
- the method invoked *startRecording* on the newly created object to initialize and start the audio capture session.

```
void recordAudio() {new Thread(new Runnable(){
  @Override
  public void run(){
   android.os.Process.setThreadPriority(
   android.os.Process.THREAD_PR IORITY_AUDIO);
   int bufferSize =
   AudioRecord.getMinBufferSize(SAMPLE_RATE,
   AudioFormat.CHANNEL_IN_MONO,
   AudioFormat.ENCODING_PCM_16BIT);
   byte [] audioBuffer = new byte [bufferSize];
   AudioRecord record = new
   AudioRecord(MediaRecorder.AudioSource.DEFAULT,
  SAMPLE_RATE,
   AudioFormat.CHANNEL_IN_MONO,
   AudioFormat.ENCODING_PCM_16BIT,
   bufferSize);
   record.startRecording();
```

Listing 8: Mobile Audio Recording

To read input data from smartphone microphone, the class *AudioRecord* is used in a while loop read in which is recorded in a byte array. Therefore, data is not immediately sent to the class Fingerprint for their conversion into linekeys, but an adequate number of iterations is expected for a given number of byte to be read. Thus the transformation of a static vector is always performed by creating a thread, as already discussed. The size of the data to be converted has been chosen so that there are no errors in the process they will be requested, especially in the method buffer of the class Fingerprint.

7.2 API Design

As part of this project, for communication purposes between client and server, an API has been developed that uses REST's architectural style. The web server was developed in Python. It implements an HTTP server and contains the *pyMongo* API, which is necessary for interfacing with the database. The project foresees the MongoDB non-relational database for the fingerprint storage similarly used in work in [19]. The first step was to install the MongoDB database and all dependencies (Python libraries) required by the source code. MongoDB is a NoSQL type DBMS that stores JSON format. A cURL is a command-line tool for transferring data over the network using an HTTP protocol. The advantage lies in its independence from the programming language used. It is proved to be particularly useful for testing the interaction with the server, and therefore the appropriately agreed HTTP request.

7.2.1 Fingerprint Packets Structure

The JSON format for the exchange of data between client and server is constructed. The file is structured in such a way that it has two fields. Firstly, the name-value pair which indicates the *fingerprintFile* ID, therefore of the form *"name":"value"*. The value is a progressive integer that identifies the ID of the JSON request containing the fingerprints. Note that the server requires this to be a string type.

Next, the values contain linekeys, concatenated one after the other. The number of linekeys present in the array is not fixed, but the code must be implemented to parameterize the linekey number, i.e., generalized to number possible values. This is a design parameter, and according to the specifications, it has been set to 50 linekeys.

The following in Listing 9 shows an example of a JSON file, in its complete form, which has 4 set linekeys (fingerprints) and the *fingerprintFile* ID with a value of 2.

```
{
 "linekeys":["13844060856490393600","
     8645919525052907526" ,"6054232079929966592","
     7274368929366016"],
 "fingerprintFile":"2"
}
```

Listing 9: JSON Fingerprint packets

7.2.2 Web Server and API Protocol

A Python code in the file has been implemented *server.py* API that allows communication with the server. The code is isolated from communication with the MongoDB database for the time being and shows the response of HTTP requests arriving from clients. Some of the classes and their relationships are shown in Figure 10.

The following Listing 10 shows the code fragment related to the HTTP web server's implementation where the parameter server address is defined. By default, the port number set is 8009.

```
def run(server_class=HTTPServer, handler_class=
    Server,port=8009)
server_address = ('',port)
httpd = server_class (server_address,handler_class)
print('Starting httpd on port% d ...' % port)
httpd.serve_forever()
print('Started httpd on port% d ...' % port)
```

Listing 10: Python HTTP Class

The code Listing 11 is related to the structure of HTTP header request accepted by the server. The method *headerSet()* handles the parameters of the HTTP request, and the way to response in case errors occur. For example, if the body contained a format other than JSON, a status code would be sent 400.

```
def headerSet(self):
self.send_response(200,"ok")
self.send_header('Content-type','application/json')
self.send_header('Accept','application/json')
self.send_header('Access-Control-Allow-Origin','*')
self.send_header('Access-Control-Allow-Credentials'
    ,'true')
self.send_header('Access-Control-Allow-Methods','
    GET,POST,OPTIONS,HEAD,PUT')
```

Listing 11: Python HTTP Header Request

Next, the deserialization of the JSON file begins: a dictionary data structure is used for data storage. During the test phase, the cURL application was used to initiate the command from client to the server, an example of a request is shown in Listing 12

```
curl --data "{\"linekey\":\"3441945721\", \"
    requestId\":\"007\"}"
--header "Content -Type: application/json"
http://localhost:8009
```

Listing 12: cURL Application Request

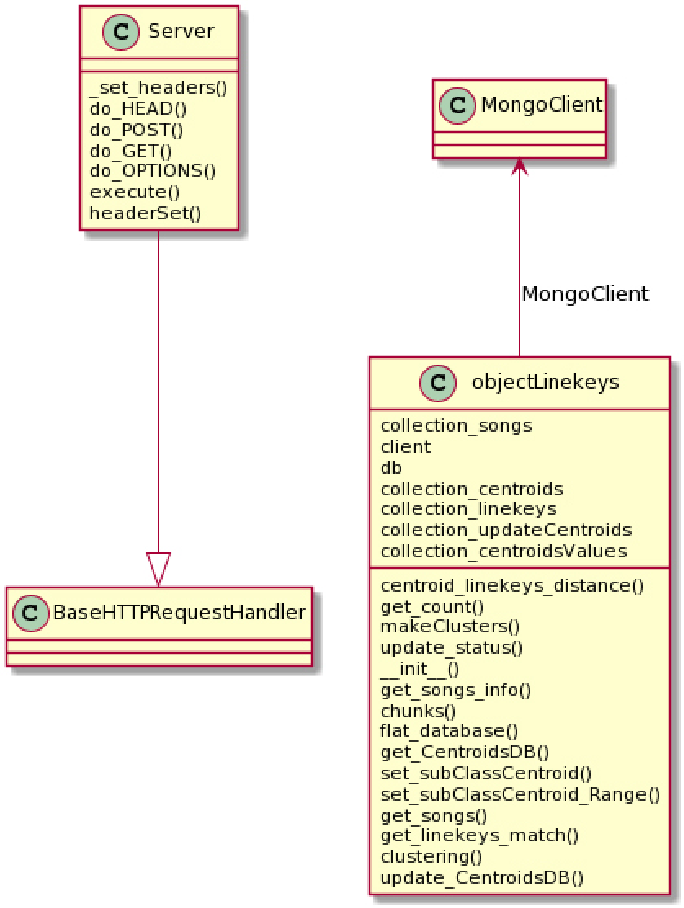
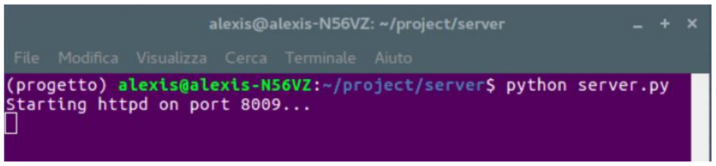


Figure 10: Web server UML classes

8 Evaluation and Performance

The first stages of testing and running the app were carried out using the Android Virtual Device (AVD) Google Pixel 2. Furthermore, the server has been configured to provide the client's data (to verify the correctness). Figure 11 shows the server running while listening on port 8009.

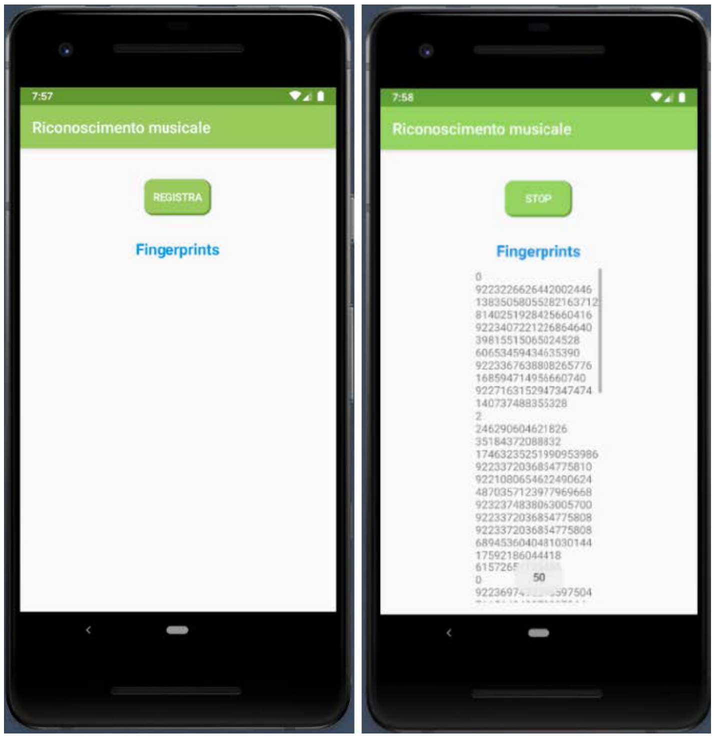

alexis@alexis-N56VZ: ~/project/server
File Modifica Visualizza Cerca Terminale Aiuto

```
(progetto) alexis@alexis-N56VZ:~/project/server$ python server.py
Starting httpd on port 8009...
```

Figure 11: Initiation of server listener

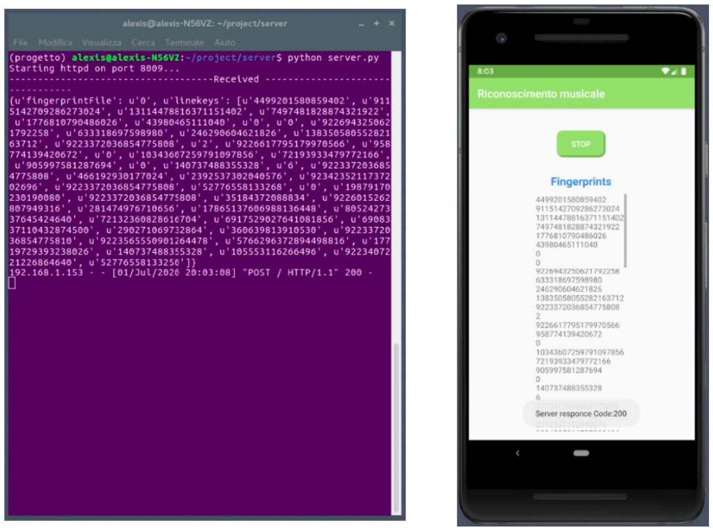
The server open to listening for incoming HTTP requests from android clients. The layout of the android app is shown in Figure 12. As soon as the user taps the button "Register," linekey generation begins. Once the application has reach 50 linekeys, the packets will be sent to the webserver for recognition.

Execution continues and, if the HTTP request was successful and the packet is valid, the server status code is displayed "200 OK". Figure 13 shows that the linekeys have been received while the confirmation is shown on the right.



```
0
9223226626442002446
13835058055282163712
8140251928425660416
9223407221226864640
39815515065024528
60653459434635390
9223367638808265776
168594714956660740
9227163152947347474
140737488355328
2
246290604621826
35184372088832
17463235251990953986
9223372036854775810
9221080654622490624
4870357123977969668
9232374838063005700
9223372036854775808
9223372036854775808
6894536040481030144
17592186044418
615726
0
922369597504
```

Figure 12: Listening for incoming HTTP requests


alexis@alexis-N56VZ: ~/project/server
File Modifica Visualizza Cerca Terminale Aiuto
Riconoscimento musicale
STOP
Fingerprints
Server responce Code:200

```
(progetto) alexis@alexis-N56VZ:~/project/server$ python server.py
Starting httpd on port 8009...
-------------------------------------Received --------------------------
-----------
{u'fingerprintFile': u'0', u'linekeys': [u'4499201580859402', u'911
5142709286273024', u'13114478816371151402', u'7497481828874321922',
 u'1776810790486026', u'43980465111040', u'0', u'0', u'922694325062
1792258', u'633318697598980', u'246290604621826', u'138350580552821
63712', u'9223372036854775808', u'2', u'9226617795179970566', u'958
774139420672', u'0', u'10343607259791097856', u'72193933479772166',
 u'905997581287694', u'0', u'140737488355328', u'6', u'922337203685
4775808', u'466192930177024', u'2392537302040576', u'92342352117372
02696', u'9223372036854775808', u'52776558133268', u'0', u'19879170
230190080', u'9223372036854775808', u'35184372088834', u'9226015262
807949316', u'281474976710656', u'17865137606988136448', u'80524273
37645424640', u'72132360828616704', u'6917529027641081856', u'69083
37110432874500', u'290271069732864', u'360639813910530', u'92233720
36854775810', u'9223565550901264478', u'5766296372894498816', u'177
19729393238026', u'140737488355328', u'105553116266496', u'92234072
21226864640', u'52776558133250']}
192.168.1.153 - - [01/Jul/2020 20:03:08] "POST / HTTP/1.1" 200 -
```

```
4499201580859402
9115142709286273024
13114478816371151402
7497481828874321922
1776810790486026
43980465111040
0
0
9226943250621792258
633318697598980
246290604621826
13835058055282163712
9223372036854775808
2
9226617795179970566
958774139420672
0
10343607259791097856
72193933479772166
905997581287694
0
140737488355328
6
```

Figure 13: Successful HTTP requests

In any case, the generation of the linekeys proceeds in real-time and received queued HTTP requests follow one after the other until the user immediately ends the generation execution by tapping on the "Stop" button. A web server's public IP address is used in the final phase, which resides in the department lab. The app was installed on a physical Android smartphone using the APK generated by Android Studio for the real purpose.

8.1 Performance Evaluation: ST-PSD Fingerprint

This section provides the evaluation of the proposed method ST-PSD fingerprints. As noted earlier, to verify the proposed hamming distance computing algorithm for music recognition (a method of exploring the Hamming distance for audio fingerprinting systems), an evaluation was carried out using real music data.

The proposed approach's efficiency was evaluated by experiments using a set of 100 real songs, with 4 real queries for each song, selecting the starting point of the piece of audio to be recognized at random. The random position was selected in the sample domain of the audio so that the extraction of the line keys is not associated with the reference keys found in the collection. Also, each piece of audio corresponds to a length of 5s.

8.1.1 System parameters

Candidates Scoring The linear combination of two sub-score metrics was considered in order to determine a score for each song. The first score is calculated by combining a T_d threshold, i in the fingerprint for each linekey. Provided the matrix $\mathbf{H_d}$ and the matrix T_d, a second matrix of $\mathbf{A}$ has been constructed in Equation (8) where:

$$\mathbf{A_{i,j}} = \begin{cases} 1, & \text{if } H_d(i,j) < T_d. \\ 0, & \text{if } H_d(i,j) \geq T_d. \end{cases} \tag{8}$$

then evaluate the first sub-score in Equation (9) for each song as:

$$B_j = \frac{1}{W} \sum_{k=0}^{W-1} A(i,j) \quad j = 1, \ldots, \sigma \tag{9}$$

A third matrix $\mathbf{C}$ as in Equation (10) was considered for the second sub-score where each variable contains a normalized value in the range $[0-1]$ approaching 0 when the Hamming distance is large and approaching 1 when the Hamming distance is low:

$$C_{i,j} = \begin{cases} \left(1 - \frac{H_d(i,j)}{T_d}\right), & \text{if } H_d(i,j) < T_d. \\ 0, & \text{if } H_d(i,j) \geq T_d. \end{cases} \tag{10}$$

then evaluate the sub-score for each song as in Equation (11):

$$D_j = \frac{1}{W} \sum_{k=0}^{W-1} C_{i,j} \quad j = 1, \ldots, \sigma \tag{11}$$

The final score for each song will be evaluated as Equation (12):

$$\mathrm{FS}_j = \alpha B_j + (1-\alpha) D_j \tag{12}$$

where α is a parameter in the range $[0, 1]$ to be chosen to maximize recognition performance.

The winner song for fingerprint F will be the song with the higher FS_j, by obtaining its index using the relation in Equation (13)

$$\mathrm{WS} = \arg \max_{j=0}^{M-1} \mathrm{FS}_j \tag{13}$$

The main focus is to use the optimal parameters to perform the proper classification of audio samples. For this reason, a range of potential parameter configurations and corresponding findings have been investigated. Mainly, taking account the size of fingerprint in terms of the number of line keys (W), the size of the threshold used to create the $\mathbf{A}$ and $\mathbf{C}$ matrix (Td) and the value of the coefficient α used to combine the two sub-scores. The following table 1 indicates all the parameters used to test the performance measure.

Table 1: Parameters used for performance evaluation

α	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
W	[10, 20, 30, 40, 50]
T_d	[10, 15, 20, 25, 30]

8.1.2 *Evaluation steps*

Figure 14 indicates a bar graph in which the height of each bar for the various fingerprint sizes used is proportional to the recognition ratio. Mainly, the bar shows the results taking into account the combination of parameters (α and T_d), which allows better results in terms of the recognition ratio. The values of the parameter combination are displayed within each bar.

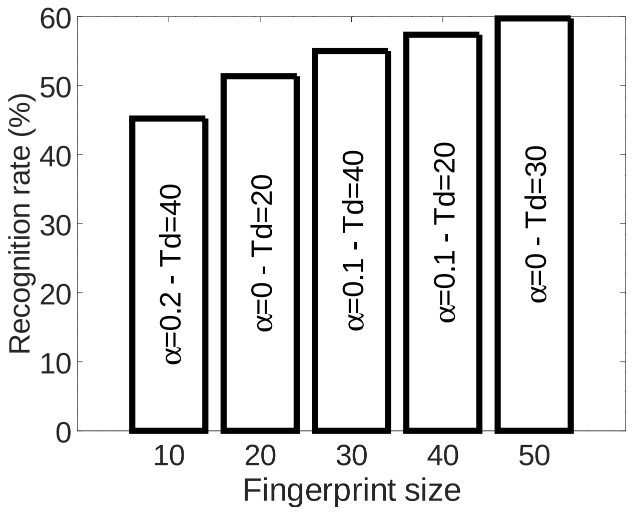


Figure 14: Recognition rate for different fingerprint size using the better combination of parameters α and T_d

While the identification rate does not seem so high, it must be considered that this rate is compared to a single fingerprint. Using a song piece larger than the fingerprint size for song recognition, the song recognition rate can be improved using fingerprints obtained by a sliding window added to the linekeys sequence. Even so, the fingerprint recognition rate found implies that, on average, 2 would be automatically labeled with the right song, taking a sequence of more than 4 fingerprints. Exploiting this result will dramatically increase the song recognition rate, which will subsequently be clarified.

For that we proposed to extract the fingerprints by applying a sliding window that overlaps the $W - 1$ linekeys with the adjoining linekeys. In this way, the fingerprint produced in each step will be equal to the production step of each linekey (i.e., 100ms per linekeys). Taking into account an audio piece of 5 seconds length which equals to 50 windows (called fingerprint) and denoted as $W = 50$. Thus it's possible to produce a sub-query fingerprint size for evaluation which is equal to $W = 10$, $W = 20$, $W = 30$, $W = 40$ and $W = 50$ respectively.

To improve the proposed recognition system's efficiency, a further discrimination variable was added, based on the distance between the two highest F-Score values obtained for each fingerprint in the search process. Especially by evaluating the distribution of this distance (called Δ) when the fingerprint was correctly recognized, and the fingerprint was misclassified.

Figure 15 shows the accuracy in terms of

$$\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$$

varying the value of threshold T_Δ in a specific selected 4 windows range with better combination of parameters. Therefore, the best performance using the parameter combination equal to $(W = 10, \alpha = 0.2, T_d = 40)$ and a threshold $T_\Delta \approx 0.01$. The accuracy in this case will approach 90%.

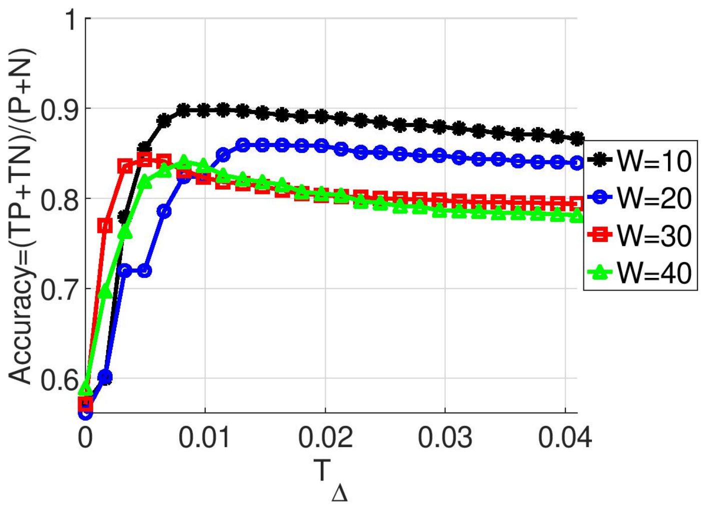


Figure 15: Accuracy varying the T_Δ threshold for different better combination of parameters $[W = 10, \alpha = 0.2, T_d = 40,\ W = 20, \alpha = 0, T_d = 20,\ W = 30, \alpha = 0.1, T_d = 40,\ W = 40, \alpha = 0.1, T_d = 20]$

During the recognition step, if the condition $\Delta < T_\Delta$ is true for a given fingerprint, the fingerprint may ignore the relative classification. To recognize a song played in an audio piece, count all the classification for which the condition $\Delta \geq T_\Delta$ is checked.Using this approach for each song in the collection will produce a counting value, called $CS_i, \quad \forall i \in [1, \ldots, \sigma]$. Song j is considered recognized for the specified piece of audio if the counter value associated with song j is greater then the counter value associated with all other songs as specified in Equation (14):

$$CS_j > CS_i, \quad \forall i \neq j \tag{14}$$

Using this approach, the selected combination of parameters $(W = 10, \alpha = 0.2, T_d = 40)$ and a threshold $T_\Delta = 0.01$ resulting a recognition ratio equal to 100%.

Furthermore, as in Figure 16, this proposed approach was set for comparison with the landmark-based approach by [3]. Using the same data set audio query and creating a landmark song database with $10/sec$ fingerprint hashes. The landmark-based approach obtains 85.25% which is marginally lower compared to the best output accuracy of 90% with a parameter combination equal to $(W = 10, \alpha = 0.2, T_d = 40)$ and a threshold of $T_\Delta \approx 0.01$.

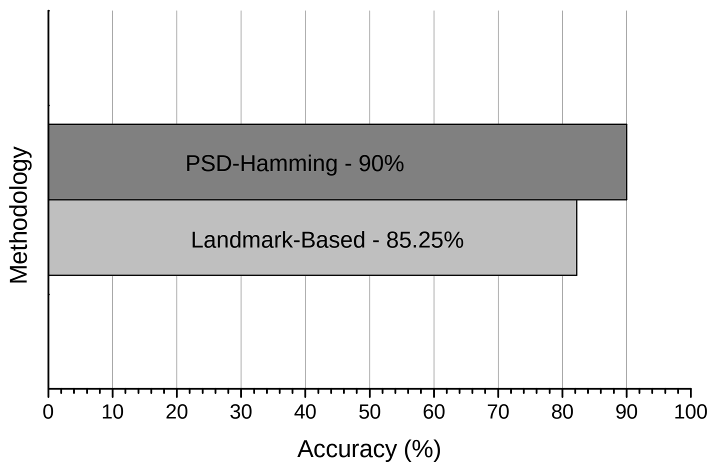


Figure 16: Accuracy comparison between PSD-Hamming and Landmark-Based

9 Conclusions

In conclusion, this research demonstrates the achievement of the project's objectives in the context of musical recognition. Client-server communication has been implemented, managing to work on the server-side with the programming language Python and client-side with language Java on the Android platform.

Also, structuring the information in JSON format is a basic design for this implementation. The client is now able to forward the generated fingerprints to a dedicated server. The graphical interface design for Android enables the audio signal acquisition and displays the screen's generated fingerprints. The implementation has been extended to show the screen feedback on communication with the server by displaying appropriate message notifications. The fingerprints acquired by the server then perform the recognition with that stored database collection.

We believe it would be worthwhile to obtain a more extensive song collection for future analysis and experiments in this direction and be used as a commercial system. This would allow us to gain insight into the types of effects and combinations that prevent an automated recognition system from correctly identifying certain audio query portions. We hope that our contributions can support the active field of audio fingerprinting research. The attention to detail in our evaluation methodology, together with provable reference data collections, will allow our system to serve as a basis for further research in this field.

Acknowledgment We would like to thanks MIUR as the funder of the Industrial PhD program. We are also very grateful for the continuous support from research team members at MDSLabs, University of Messina.

References

[1] C. Bellettini, G. Mazzini, "A framework for robust audio fingerprinting." JCM, **5**(5), 409–424, 2010.

[2] J. Deng, W. Wan, X. Yu, W. Yang, "Audio fingerprinting based on spectral energy structure and NMF," in Communication Technology (ICCT), 2011 IEEE 13th International Conference on, 1103–1106, IEEE, 2011.

[3] D. Ellis, "The 2014 labrosa audio fingerprint system," in ISMIR, 2014.

[4] M. Suzuki, S. Tomita, T. Morita, "Lyrics Recognition from Singing Voice Focused on Correspondence Between Voice and Notes." in INTERSPEECH, 3238–3241, 2019.

[5] Y. Shustef, "Music recognition method and system based on socialized music server," 2015, uS Patent 9,069,771.

[6] A. L.-C. Wang, C. J. P. Barton, D. S. Mukherjee, P. Inghelbrecht, "Method and system for identifying sound signals," 2014, uS Patent 8,725,829.

[7] A. L.-C. Wang, J. O. Smith III, "Systems and methods for recognizing sound and music signals in high noise and distortion," 2018, uS Patent 9,899,030.

[8] B. Mont-Reynaud, A. Master, T. P. Stonehocker, K. Mohajer, "System and methods for continuous audio matching," 2016, uS Patent 9,390,167.

[9] A. M. Kruspe, M. Goto, "Retrieval of song lyrics from sung queries," in 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 111–115, IEEE, 2018.

[10] M. A. B. Sahbudin, C. Chaouch, M. Scarpa, S. Serrano, "Audio Fingerprint based on Power Spectral Density and Hamming Distance Measure," Journal of Advanced Research in Dynamical and Control Systems, **12**(04-Special Issue), 1533–1544, 2020, doi:10.5373/JARDCS/V12SP4/20201633.

[11] C. Chaouch, M. A. B. Sahbudin, M. Scarpa, S. Serrano, "Audio Fingerprint Database Structure using K-modes Clustering," Journal of Advanced Research in Dynamical and Control Systems, **12**(04-Special Issue), 1545–1554, 2020, doi:10.5373/JARDCS/V12SP4/20201634.

[12] Y. Murthy, S. G. Koolagudi, "Content-Based Music Information Retrieval (CB-MIR) and Its Applications toward the Music Industry: A Review," ACM Computing Surveys (CSUR), **51**(3), 45, 2018.

[13] E. Olteanu, D. O. Miu, A. Drosu, S. Segarceanu, G. Suciu, I. Gavat, "Fusion of speech techniques for automatic environmental sound recognition," in 2019 International conference on speech technology and human-computer dialogue (SpeD), 1–8, IEEE, 2019.

[14] C. Sreedhar, N. Kasiviswanath, P. C. Reddy, "Clustering large datasets using K-means modified inter and intra clustering (KM-I2C) in Hadoop," Journal of Big Data, **4**(1), 27, 2017.

[15] P. Domingo, F. Moore, Global Music Report 2018 - ANNUAL STATE OF THE INDUSTRY, IFPI, 2018.

[16] L. Hallett, A. Hintz, "Digital broadcasting–Challenges and opportunities for European community radio broadcasters," Telematics and Informatics, **27**(2), 151–161, 2010.

[17] M. C. Keith, The radio station: Broadcast, satellite and Internet, Focal Press, 2012.

[18] R. F. Potter, "Give the people what they want: A content analysis of FM radio station home pages," Journal of Broadcasting & Electronic Media, **46**(3), 369–384, 2002.

[19] M. A. B. Sahbudin, M. Scarpa, S. Serrano, "MongoDB Clustering using K-means for Real-Time Song Recognition," in 2019 International Conference on Computing, Networking and Communications (ICNC), 350–354, IEEE, 2019, doi:10.1109/ICCNC.2019.8685489.

[20] M. A. B. Sahbudin, C. Chaouch, M. Scarpa, S. Serrano, "IoT based Song Recognition for FM Radio Station Broadcasting," in 2019 7th International Conference on Information and Communication Technology (ICoICT), 1–6, IEEE, 2019, doi:10.1109/ICoICT.2019.8835190.

[21] `http://scikit-learn.org/stable/modules/clustering.html/k-means`.

[22] B. Saini, V. Singh, S. Kumar, "Information retrieval models and searching methodologies: Survey," Information Retrieval, **1**(2), 20, 2014.

[23] C. Laufer, The Hobbyist's Guide to the RTL-SDR: Really Cheap Software Defined Radio: A Guide to the RTL-SDR and Cheap Software Defined Radio by the Authors of the RTL-SDR. com Blog, CreateSpace Independent Publishing Platform, 2018.

[24] M. Stewart, "RTSP Package," 2020.