TRABAJO FINAL DE GRADO

TÍTULO: Audio fingerprinting para la identificación automática de contenidos audiovisuales

TITULACIÓN: Grado en Ingeniería de Sistemas de Telecomunicación

AUTOR: Enrique Roca Navarro

DIRECTOR: Francesc Tarres

FECHA: 24/10/2017

Título: Audio fingerprinting para la identificación automática de contenidos audiovisuales

Autor: Enrique Roca Navarro

Director: Francesc Tarres

Fecha: 24/10/2017

Resumen

El audiofingerprinting es una tecnología utilizada para identificar archivos de audio basados en el contenido del archivo. Con ellas podemos identificar un patrón de un archivo de audio, para que el patrón pueda ser reconocido desde una base de datos de audio, sin necesidad de disponer información acerca de este. Su uso se basa en la detección de un segmento de audio y posterior envío a una base de datos (BD), para una vez analizada, buscar coincidencias en esta y devolvernos información acerca de la muestra analizada.

Uno de los métodos para poder realizar esta identificación es el método conocido como Hashing.

A partir de esta metodología nosotros podemos escoger un fragmento de canción de una duración determinada y realizar el espectrograma de la canción. A partir de aquí con la ayuda de unos filtros conseguiremos una serie de puntos llamados candidatos. Más tarde se realiza el hashing donde obtendremos una información codificada en bits.

Paralelamente existe una base de datos de capacidad y efectividad razonable para guardar toda la información de las diferentes canciones.
Finalmente se realizará una comparativa entre el resultado del Hashing y lo almacenado en la base de datos para ver coincidencias. En el caso de que las haya, a partir de ese segmento podremos saber que canción es.

En este trabajo haremos un estudio en profundidad de que consiste el audiofingerprining y realizaremos un caso práctico de simulación a través del programa para poder ver todo el funcionamiento.

Títle: Audio fingerprinting para la identificación automática de contenidos audiovisuales

Author: Enrique Roca Navarro

Advisor: Francesc Tarres

Date: 24/10/2017

Overview

Audiofingerprinting is a technology used to identify audio files based on the content of the file.
With them we can identify a pattern of an audio file, so that it can be recognized from an audio database, without having to have information about it. Its use is based on the detection of an audio sample and subsequent sending to a database (BD), once analyzed look for matches in this and return us information about the sample analyzed.

One of the methods to perform this identification is the method known as Hashing.

From this methodology we can choose a song fragment of a certain duration and make the spectrogram of the song. From here with a series of filters we will get a series of points called candidates. Later the hashing is done where we will obtain information encoded in bits.

In parallel there is a database of capacity and reasonable effectiveness to keep all the information of the different songs.

Finally, a comparison will be made between the result of the Hashing and the one stored in the database to see matches. In case you have them from that segment we can know what song it is.

In this work we will do an in-depth study of the audiofingerprining and we will make a practical case of simulation through the program to be able to see the whole operation.

ÍNDICE

Lista de Ilustraciones

Lista de Ecuaciones

INTRODUCCIÓN

El objetivo de este trabajo es el estudio de la tecnología del audio fingerprinting, para la identificación automática de contenidos audiovisuales.

Para ello, el trabajo constara de tres partes:

- En el primer capítulo se podrá encontrar una explicación teórica sobre las diferentes estrategias de fingerprinting audio que existen con sus respectivas ventajas e inconvenientes. Más tarde explicaremos paso por paso a nivel teórico en que consiste la tecnología del audio fingerprinting enfocándonos en la metodología del hashing (empleada por Shazam). Finalmente, también haremos una breve introducción sobre las diferentes aplicaciones en las que se utiliza esta tecnología en la actualidad y explicaremos los principales competidores a la tecnología que me he centrado.
- Seguidamente, haremos un desarrollo, dónde a partir de un entorno de computación numérica y un lenguaje de programación como es el Matlab haremos una demostración del funcionamiento del fingerprinting audio. Escogiendo una canción haremos paso por paso toda la implementación hasta llegar al punto de generar el hash. Más tarde con el lenguaje Python generaremos los diferentes hashes de cada canción.
- Finalmente, explicaremos como se almacena toda esta información en bases de datos y explicaré una herramienta, muy útil para gestionar este tipo de información.

CAPÍTULO 1. FUNDAMENTOS AUDIO FINGERPRINTING

1.1. Introducción

La tecnología del audio fingerprinting para la identificación automática de contenidos audiovisuales, apareció en el año 2000, con la idea de proporcionar un servicio que permitiera reconocer una melodía al completo a partir de unas notas mediante aplicaciones desktop. El algoritmo tendría que ser capaz de reconocer una breve muestra de audio de la música transmitida, mezclado con ruido ambiental a través de un micrófono sometido a la compresión códec de la voz.

Así una huella digital de audio buscamos que sea un código único generado a partir de una onda que pueda ser usada para identificar automáticamente una muestra de audio.

En la actualidad ha incrementado mucho el interés científico e industrial en computar audiofingerprinting. El crecimiento queda demostrado por el gran número de compañías que están desarrollando esa técnica. Una de ellas es el Shazam.

La Federación Internacional de la Industria de la Fonografía (IFPI) es otra de las que han mostrado interés y este se ve reflejado en la reciente solicitud de información sobre las técnicas que desarrollan esta tecnología.

Uno de los principales objetivos que tiene esta nueva técnica consiste en conseguir mecanismos más eficientes para establecer una igualdad entre dos contenidos multimedia comparando las huellas digitales audio.

Las grandes ventajas que obtenemos con este uso, frente a utilizar el contenido multimedia directamente son las siguientes:

- Reducir la memoria de almacenamiento ya que los fingerprintings ocupan menos espacio que los contenidos multimedia.
- Comparaciones más eficientes, incluyendo si aparecen distorsiones o ruidos, ya que estas con las técnicas de audiofingerprinting pueden quedar inapreciables.
- Búsqueda más eficiente de datos ya que el tamaño también es menor.

1.1.1. Definición de un sistema basado en fingerprinting audio

Fingerprinting audio es una técnica que se basa en obtener una representación compacta de un objeto de audio, es decir, en obtener una huella digital que resuma un objeto de audio, y trabajar con ella a la hora de identificar el objeto de audio en una base de datos en vez de realizar la búsqueda usando el objeto

de audio original. Por tanto, la función para obtener el fingerprint debe hacer una especie de mapeo a partir del objeto de audio, el cual contiene un gran número de bits, dando como resultado una huella cuya longitud es un número pequeño y limitado de bits.

Un sistema completo basado en audio fingerprint o, como también se le conoce Content-based Identification (CBID), consiste, bajo una descripción muy general, en extraer características perceptuales relevantes de objetos de audio, en nuestro caso canciones, obteniendo los correspondientes fingerprints y almacenarlos en una base de datos.

Una vez que el sistema conforma la base de datos, comienza a trabajar en tiempo real con la señal de audio de entrada. Así, el sistema va recogiendo trozos de una longitud fija de esa señal, calcula la huella digital de ese trozo de audio, y busca en la base de datos el fingerprint más parecido. Como resultado, usando los fingerprints de la base de datos y el que va calculando en tiempo real junto con eficientes algoritmos de búsqueda, podemos identificar, en un periodo de tiempo relativamente corto, qué objeto de audio, o anuncio, se está emitiendo en cada momento

1.1.2. Estrategias de un sistema basado en fingerprinting audio

En este sub-capítulo vamos hacer una descripción un poco más detallada de las diferentes estrategias que se utilizan en el fingerprinting audio con sus ventajas e inconvenientes. Dependiendo si lo queremos para identificar canciones o por ejemplo para identificar anuncios, el sistema presenta algunas diferencias que explicaremos a continuación.

Un sistema de audio fingerprint consiste en dos bloques bastante diferenciables: el primero el bloque de extracción del fingerprint y el segundo bloque que es el de búsqueda y detección referente a la base de datos.

Para anuncios el proceso de extracción se basa en el front-end o cabecera que divide la señal de audio en trozos o tramas y extrae un número considerable de características discriminatorias y robustas a partir de cada trama. Según si es un anuncio o es una canción nos interesará un tamaño u otro de fingerprint.

En el bloque de front-end hay que tener en cuenta que la salida ha de ser robusta ante distorsiones o ruidos de fondo.
El bloque se divide en varias partes definidas y cada una de las partes presenta diferentes opciones. A continuación, se detalla:

- Pre-Procesado: En este primer paso, lo que haremos en nuestro proyecto es convertir la señal a mono. La frecuencia de muestreo utilizada será aproximadamente de 10 KHz. En este punto hay otras alternativas como la realización de un preénfasis, aplicar un filtro paso banda o la utilización de un codificador/decodificador de GSM.

El filtro paso banda o el codificador/decodificador GSM es muy útil en aplicaciones de identificación de conversaciones en la cual la señal llega vía teléfono convencional o móvil.

- Enventanado y Overlap: En nuestro proyecto hemos utilizado un 50% de overlap con una ventana de tipo Hanning. En cambio, para anuncios este 50% no es suficiente ya que el sistema necesita una superposición entre ventanas subyacentes más elevado para asegurar robustez ante desplazamientos.

- Transformación lineal: En nuestro proyecto lo que utilizaremos es el uso de espectrograma. Con el espectrograma realizaremos la transformación lineal a partir de encontrar la FFT y más tarde pasaremos de FFT(compleja) a espectro. Otras alternativas que se pueden utilizar son la Hadamard. La DFT es generalmente menos sensibles a cambios pero esta tiene un coste computacional menor.

- Extracción de característica: Una vez que tenemos la representación tiempo-frecuencia, se aplican transformaciones adicionales para generar el vector final. En este paso, nos encontramos con una gran variedad de algoritmos. El objetivo de todos ellos es obtener una reducción en la dimensión del vector de salida del bloque “front-end” y, al mismo tiempo, incrementar la invariancia ante distorsiones. En nuestro proyecto hemos empleado el sistema Hashing que estará explicado en el punto 1.3 con sus ventajas e inconvenientes respecto a otros sistemas.

- Post-Procesado: Es muy común aplicar una resolución muy baja por lo que respecta a la cuantización de las características escogidas para crear el fingerprint: ternario o binario. El propósito de la cuantización es ganar robustez contra las distorsiones, normalizar, facilitar la implementación en hardware y reducir los requisitos de memoria.

El segundo bloque importante es el de búsqueda. Este bloque es fundamental para la utilidad para ver la eficiencia comparativa entre el audio fingerprint correspondiente al audio desconocido y los fingerprint que hay registrados en la base de datos.

Un algoritmo que compute estas semejanzas puede hacer de nuestro sistema un sistema muy poco eficiente, pues el tiempo para encontrar el fingerprint de la base de datos más parecido al calculado a partir del audio desconocido es proporcional a:

$$N \cdot c(d(\,)) + E$$

Ecuación 1: Ecuación de Proporcionalidad Tiempo de búsqueda

Donde N es el número de huellas que contiene la base de datos, la c(d()) es el tiempo necesario para encontrar la semejanza del fingerprint y E modela un cierto tiempo CPU adicional.

Algunos métodos de búsqueda que optimizan este tiempo:

- Distancias pre-computadas
- Filtrado de candidatos no probables con medidas simples
- Índices invertidos a ficheros: Muy útil para todo el sistema basado en recuperación de conversaciones.
- Reducción de Candidatos
- División de la base de datos: Muy útil para todo el sistema de audio fingerprint basado en anuncios

En el punto 1.4 explicaremos con detalle el procedimiento de búsqueda que aplicamos para identificar audios de canciones y explicaremos sus ventajas e inconvenientes.

1.1.3. Aplicaciones que utilizan audio fingerprint

En este apartado comentaremos diferentes aplicaciones en las que se hace uso de los sistemas audio fingerprinting.

El audio fingerprinting en la actualidad como se puede observar es un sistema muy utilizado, ya que aparte de identificar canciones a partir de un fragmento de audio, se utiliza para recuperar grabaciones de conversaciones gravemente degradadas realizadas desde teléfonos móviles.

La mayoría de ellas se basan en la capacidad de reconocer diferentes audios a partir de una búsqueda en la base de datos sin importar en que formato viene el audio.

Alguna de las más importantes son:

- Supervisión o monitorización de contenido audio: Por lo que respecta a la supervisión del distribuidor final, los distribuidores pueden necesitar saber si tienen los derechos para difundir el contenido de audio a los consumidores. Fingerprinting puede ayudar a identificar objetos de audio sin etiquetar en bases de datos de canales de TV y radio.
  Con ese sistema se puede también identificar el contenido audio no identificado recuperado de distribuidores que hacen discos y de esta forma ir en contra de la piratería.

- Supervisión del canal de transmisión: En muchos países, las estaciones de radio deben pagar los derechos sobre la música que emiten. Los encargados de mantener estos derechos necesitan supervisar las transmisiones de radio para verificar si los derechos se están pagando correctamente. Incluso en los países en donde las estaciones de radio pueden emitir música libremente, los encargados de los derechos están interesados en la supervisión de las transmisiones de radio con propósito estadísticos. Los publicistas también necesitan supervisar las transmisiones de radio y de TV para verificar si los anuncios están

siendo difundidos según lo convenido. Igual ocurre para las difusiones vía Web. Otras aplicaciones incluyen las compilaciones de gráficos para el análisis estadístico del material del programa que se está emitiendo.

1.1.4. Optimización de los sistemas basado en audio fingerprint (Anuncios)

Para tener un buen sistema de audio fingerprinting, nos interesa optimizar al máximo el sistema sobre todo en anuncios de televisión que contienen un gran contenido multimedia. Con la optimización se pretende conseguir aspectos como:

- Reducción del tiempo necesario para extraer una huella
- Reducir el tamaño de una huella manteniendo la probabilidad de error y el grado de robustez del sistema en unos valores aceptables.

Para todos los parámetros que tenemos en el sistema de audiofingerprinting explicado anteriormente la optimización se debe hacer en tres fases:

Fase 1: Modelar mediante funciones de densidad de probabilidad de error conocidas. Existen dos muy útiles: la probabilidad de falsa alarma que es la probabilidad de que se produzca una detección de una canción sin que debiera producirse. Y la probabilidad de no alarma que es la probabilidad de que no se produzca una detección cuando debería producirse.

Fase 2: Encontrar los valores de los parámetros óptimos para que el tamaño del fingerprinting sea el menor posible ya que eso nos mejora a nivel computacional y también nos requiere menos memoria (explicado más adelante en la parte procedimental con más detalle)

CAPÍTULO 2. FUNDAMENTOS PARA RECUPERAR MELODÍAS A PARTIR DE UN TROZO DE AUDIO.

2.1 Análisis de la canción basada en el espectrograma

Tanto los archivos de "base de datos" como los registros obtenidos durante el proceso se someten al mismo análisis.
Primero de todo seleccionamos una canción y lo que hacemos es escoger un fragmento de ella. Esta se divide en diferentes muestras. Una vez las tenemos se realiza una comparación entre las que tenemos registradas en la base de datos de música y las que hemos seleccionado en el fragmento de audio. A continuación, se evalúan las coincidencias de candidatos con respecto a la exactitud de la coincidencia.

El análisis al que se somete la canción es el siguiente:

Primero de todo, se realiza el espectrograma al fragmento de audio seleccionado. En él obtenemos como resultado una serie de puntos que destacan por tener un pico de energía superior al resto. Pero, no podemos considerarlo como definitivos si no tienen un mayor contenido de energía que los puntos que se sitúan alrededor. Los picos se eligen de acuerdo con un criterio de densidad, para asegurar que en cualquier parte del tiempo tiene una amplitud considerable. Cuanta más amplitud tienen, más probabilidad tienen de aguantar a distorsiones considerables. Por tanto, a partir de un espectrograma inicial como el que podría ser el de la ilustración 1, se podría reducir a una imagen con los puntos definitivos como el de la ilustración 2.

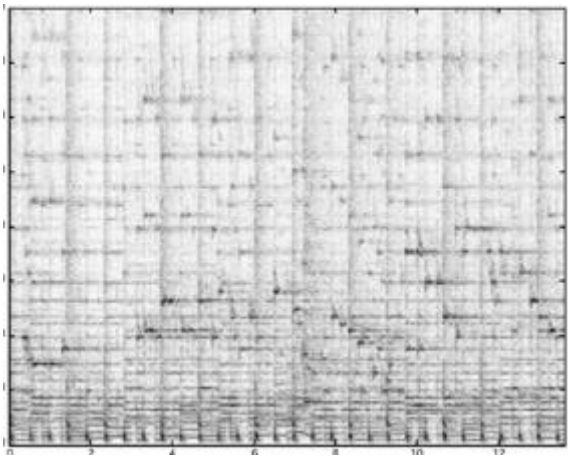


Ilustración 1: Ejemplo de espectrograma

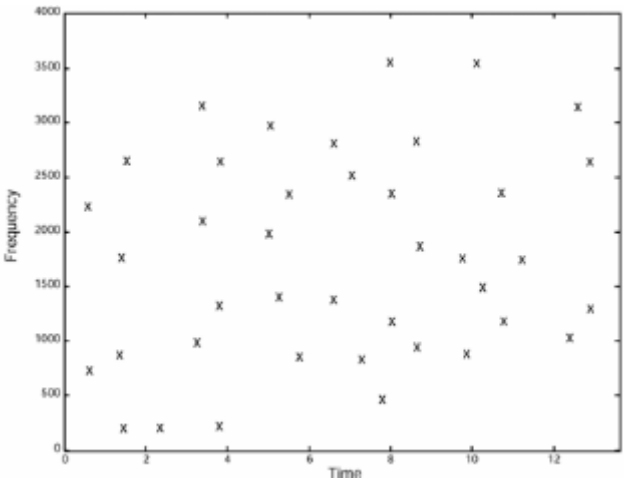


Ilustración 2: Ejemplo de puntos candidatos

Esto se realiza mediante una serie de procedimientos. Primero de todo tenemos que encontrar los máximos locales. Para ello utilizamos primeramente un filtro de dilatación o un filtro máximo y finalmente se utiliza un filtro de erosión binaria que explicaremos a continuación y que detallaremos como se realiza en la parte procedimental.

Al realizar el espectrograma de la canción, lo que hacemos es realizar una normalización para poder ver el espectrograma como si fuera una imagen y en él obtenemos uno de los primeros problemas, ya que hay algunos picos que dudamos entre elegirlos como punto candidato o no. En la imagen definimos cada pixel si pertenece a uno de los puntos candidatos o no y eso lo hacemos mediante un proceso donde convertimos la imagen en binarias (un bit para la cuantificación) Blanco y negro.

Una vez obtenida la imagen (matriz de i filas j columnas) binaria (valores 0,1), se ve que quedan pixeles o pequeños grupos de ellos que sabemos que podrían pertenecer a candidatos, pero por diferentes motivos como puede ser el ruido no quedan identificados. También existe la posibilidad contraria, que es que aparezcan pixeles que sabemos que no deberían haber aparecido.

Para incorporar estos pixeles, conviene utilizar un filtro de dilatación con el objetivo en que si alguno de los pixeles vecinos al pixel en estudio pertenece a uno que se considera candidato, entonces el pixel de estudio también pertenece al objeto.

Otra técnica, es utilizar los filtros espaciales no lineales que también operan sobre entornos. En este caso, nos interesa aplicar el filtro de máximos en el cual se selecciona el mayor valor de una ventana ordenada de valores de nivel de gris.

Finalmente, para eliminar algún punto que ha podido aparecer de forma errónea al aplicar el filtro de dilatación utilizamos un filtro de erosión. Este filtro tiene el siguiente funcionamiento: si todos los pixeles vecinos al pixel de estudio son candidatos, entonces el pixel de estudio también lo es. Si alguno

de los pixeles vecinos al pixel de estudio no puede ser candidato, entonces ese pixel de estudio tampoco lo es.

Una vez aplicado los diferentes filtros nos aparecerá una imagen donde su matriz está completamente llena de 0 y 1. Los unos serán los diferentes puntos en los cuales tendremos un máximo y por tanto serán utilizados para la identificación del audio. Una vez los tenemos identificados solo nos faltará conocer la posición de la muestra y la cantidad de energía que contiene.

A partir de este procedimiento pasaremos, de tener el espectrograma de la canción como el de la ilustración 1, al espectrograma de la canción con los puntos candidatos de la ilustración 2.

Una vez tenemos el espectrograma del trozo de canción a analizar, lo que tenemos que realizar es una concatenación para tener todos los puntos candidatos de una canción entera, con el fin de ejecutar a posteriori el procedimiento de Hashing con los puntos candidatos de toda la canción.

2.2 Generación de Información de Hashing

2.2.1 Que es la función Hash

Una función Hash es una función computable mediante un algoritmo tal que:

$$H: U \rightarrow M$$
$$x \rightarrow h(x)$$

Ecuación 2: Ecuaciones función Hash

Tiene como entrada un conjunto de elementos, que suelen ser cadenas y los convierte en un rango de salida finito, normalmente cadenas de longitud fija. Es decir, la función actúa como una proyección del conjunto U sobre el conjunto M [8].
Observar que M puede ser un conjunto definido de enteros. En este caso podemos considerar que la longitud es fija si el conjunto es un rango de números de enteros ya que podemos considerar que la longitud fija es la del número con mayor cifra.
Normalmente el conjunto U tiene un número elevado de elementos y M es un conjunto de cadenas con un número más o menos pequeño de símbolos. La idea básica de un valor Hash es que sirva como una representación compacta de la cadena de entrada.

Esta función es muy útil ya que tiene un coste computacional muy bajo, se puede mapear un dominio con datos de longitud muy grande.

2.2.2 SHA1

El SHA (Secure Hash Algorithm, Algoritmo de Hash Seguro) es una familia de funciones hash de cifrado publicadas por el Instituto Nacional de Estándares y Tecnología (NIST).

SHA-1 es el algoritmo que vamos a utilizar en nuestro proyecto y pueden producir una salida resumen de hasta 160 bits (20 bytes) de un mensaje que puede tener un tamaño máximo de 2^{64} bits. Normalmente es representado de forma hexadecimal.

2.2.3 Utilización del Hashing en el Audio Fingerprinting

El valor del Hash en el audio fingerprinting se forma a partir de los puntos que se han generado a partir del mapa de constelación explicado en el apartado anterior y explicado paso por paso en la parte procedimental.
Una vez tenemos los puntos en tiempo-frecuencia lo que haremos es ir cogiendo estos puntos por parejas. Estas estarán asociadas de forma combinatoria [1]. Otro de los aspectos que se tendrán que tener en cuenta a la hora de hacer una buena elección es la zona objetivo. Todas las parejas tendrán que tener un elemento dentro de la zona elegida.
Por cada pareja se proporciona la información de las dos componentes en frecuencia y la distancia en tiempo entre los dos puntos. Esto se observa en las ilustraciones siguientes:

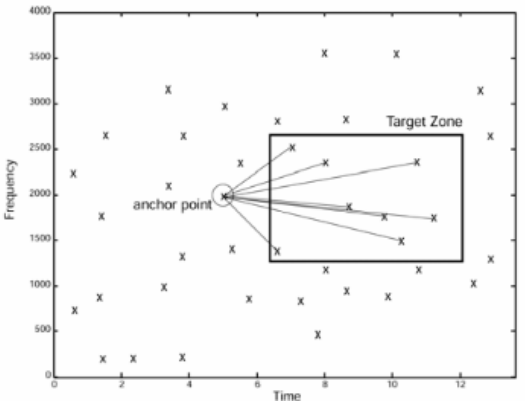


Ilustración 3: Emparejamiento de puntos

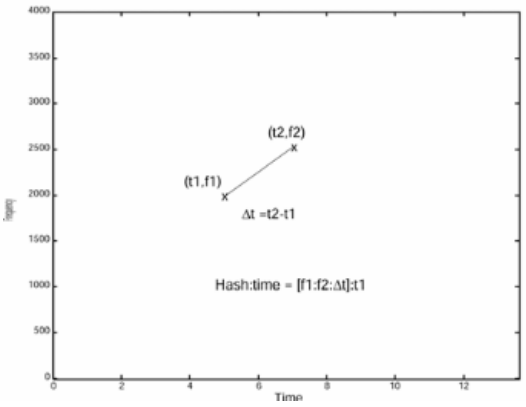


Ilustración 4: Cálculo de Hash-time

Esta metodologia es bastante resistente a la presencia de ruido o pequeños errores en la codificación de voz. Además cada hash se puede empaquetar en numeros naturales de 32 bits. Cada hash también está asociado con el desplazamiento de tiempo desde el comienzo hasta su punto de unión, aunque el tiempo absoluto no es una parte importante en el hashing.

Para crear un índice de base de datos, la operación se lleva en cada una de las canciónes que contiene la base de datos con el fin de generar una lista correspondiente de hashes y tiempos de desplazamientos asociados. Los Ids de las canciones también se pueden añadir a las estructuras de datos pequeñas. Por tanto lo ideal es almacenar información de 64 bits 32 para el procedimiento de hashing y otros 32 para el desplazamiento de tiempo y ID de canción. Estas seran ordenadas con prioridad a los valores de hashing.

$$nº\ \frac{hashes}{s} = densidad\ de\ puntos\ de\ constelación * F\ (factor\ fan - out\ zona\ objetivo)$$

Ecuación 3: Calculo de Hashes por segundo

Por ejemplo, si cada punto de constelación se toma un punto de anclaje y el factor fan-out (F) [3] de la zona objetivo = 20 entonces el número de hashes es 20 veces el número de puntos candidatos de la constelación extraídos. El factor fan-out es un parámetro que se puede ajustar para asegurar una mejor detección con poco coste computacional. Cuanto más grande sea más coste computacional habrá en el proceso, pero a su vez mejorará la detección. Por tanto, el factor fan-out de la zona objetivo coge un papel muy importante para poder tener bien identificada la canción. Dependiendo el tamaño puede influir gravemente en el coste de almacenamiento.

El hecho de trabajar con pares en lugar de trabajar con constelaciones individuales obtenemos una aceleración en el proceso de búsqueda. Si, por ejemplo, tenemos que cada componente de frecuencia es de 10 bits y el componente Δt también es de 10 bits al emparejar un par de puntos candidatos, tendremos 30 bits de información frente a sólo 10 para un único punto. Por tanto, la velocidad de búsqueda para un solo token de hash se acelera aproximadamente un millón de veces debido a estos 20 bits extra de información que tenemos. En la siguiente ilustración se puede ver un resumen de lo explicado.

Parámetro	nºBits
Frecuencia 1	10 bits
Frecuencia 2	10 bits
Variación de tiempo	10 bits

Ilustración 5: Resumen Hashing Información pares de puntos

2.3 Procedimiento de Búsqueda de Coincidencias

Para realizar una búsqueda del audio que se ha capturado por el dispositivo móvil, se realiza el procedimiento de Hashing descrito anteriormente.

Cada hash de los diferentes puntos, se utilizan para buscar en la base de datos los tiempos correspondientes desde el comienzo de la muestra y los archivos de base de datos asociados a los pares de tiempo. Los pares de tiempo como hemos comentado en el procedimiento de Hashing, se distribuyen en bins con el ID de la canción asociada y el hash de base de datos correspondiente [1]. Despues de que todos los hashes de las muestras se han utilizado para buscar en la base de datos pares coincidentes, se analizan los comportamientos para buscar coincidencias entre lo recibido y lo registrado en la base de datos. Dentro de cada bin, el conjunto de pares representa un diagrama de dispersión

de asociación entre los archivos de sonido de la muestra y la base de datos. Si los archivos coinciden, las características coincidentes deben ocurrir en desplazamientos muy similares desde el principio del archivo, es decir tendríamos una secuencia de hashes casi idéntica.

El problema de decidir si se ha encontrado una coincidencia o no se reduce a detectar un grupo significativo de puntos que forman una línea diagonal dentro del diagrama de dispersión. Un ejemplo sería el de la ilustración 6

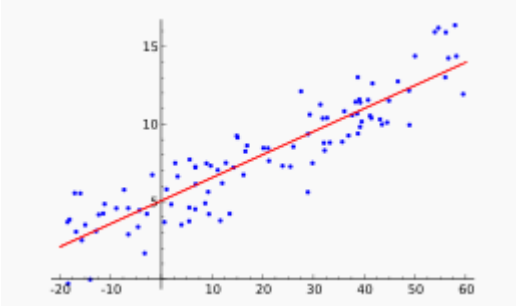


Ilustración 6: Línea Diagonal que garantiza coincidencias

Podemos suponer que la diagonal tiene pendiente 1, entonces los tiempos correspondientes a las muestras coincidentes entre archivos cumplen la siguiente fórmula:

$$t_k' = t_k + \text{offset}.$$

Ecuación 4: Relación de coordenadas temporales DB y muestras

- T_k' = es la coordenada temporal en la base de datos
- T_k = es la coordenada temporal de la muestra de la canción

Por cada T_k' y T_k en el diagrama de dispersión calculamos la diferencia entre la muestra de la base de datos y la muestra de la canción:

$$\delta t_k = t_k' - t_k.$$

Ecuación 5: Cálculo de la diferencia entre coordenadas

Este proceso de búsqueda se realiza en cada una de las pistas de la base de datos hasta que se encuentra una coincidencia significativa en caso de que la haya. Tenga en cuenta que las fases de coincidencia y escaneado no hacen ninguna suposición especial sobre el formato de los hashes. De hecho, los hashes sólo necesitan tener las propiedades de tener entropía suficiente para evitar que se produzcan demasiadas coincidencias falsas, además de ser reproducibles. En la fase de escaneo lo principal es que los hashes coincidentes estén temporalmente alineados.

2.4 Resistencia al ruido

El algoritmo funciona bien con niveles significativos de ruido e incluso distorsión no lineal. Puede identificar correctamente la música en la presencia de voces, ruidos e incluso música mezclada. Una propiedad de la técnica anterior es que las discontinuidades son irrelevantes, lo que permite la inmunidad a los abandonos y el enmascaramiento debido a la interferencia.
Un resultado algo sorprendente es que incluso con una gran base de datos, podemos identificar correctamente cada una de varias pistas mezcladas, incluyendo varias versiones de la misma pieza, una propiedad que llamamos "transparencia".

CAPÍTULO 3. IMPLEMENTACIÓN Y DESARROLLO

3.1 Introducción

La parte procedimental constará de tres partes bastante destacadas:

- Primera parte haremos una explicación y un análisis paso por paso del funcionamiento del audio fingerprinting para la detección de canciones a partir de una canción escogida.
- En la segunda parte haremos una breve implementación de la extracción de los puntos candidatos en Matlab y realizaremos a partir del programa Spyder que admite lenguaje Python, la generación de los Hashes que tendrán toda la información para el reconocimiento de canciones.
- Finalmente haremos una muy breve introducción de como se podría crear una aplicación web con el fin de recoger todos los valores que insertamos en la base de datos para que el usuario tenga una visualización mucho más intuitiva.

3.2 Plataformas

Las principales plataformas que utilizaremos en este trabajo serán Matlab Python y APEX.

- Matlab: Con este programa nos encargaremos básicamente de explicar con un ejemplo paso por paso en que consiste todo el procedimiento de audio fingerprinting, desde la lectura de la canción hasta conseguir los diferentes puntos candidatos a realizar el Hash. Cuando tengamos los resultados generaremos un fichero .csv que más tarde será utilizado en Python para generar los Hash.
- Python: En Python nos encargaremos básicamente de extraer los Hashes de la Canción. Primero, recogeremos los datos de Matlab a partir de un fichero csv, más tarde extraeremos los diferentes Hashes de las canciones y generaremos otro .csv que será utilizado más tarde para importarlo en la base de datos.

- APEX: Con este programa crearemos los objetos necesarios en una base de datos para almacenar la información de los Hahes y crearemos una aplicación web para gestionar de forma visual y didáctica el sistema de audiofingerprint.

Ilustración 7: Programas utilizados

3.3 Librerías utilizadas

Uno de los programas utilizado en concreto el Python, para poder tener un buen funcionamiento del programa se requiere importar unas librerías que utilizaremos a continuación:

- Numpy: es una extensión de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices. También se utiliza para escribir ficheros en csv

- Matplotlib: es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación Python y su extensión matemática NumPy. Proporciona una API, pylab, diseñada para recordar a la de MATLAB.

- SciPy: contiene módulos para optimización, álgebra lineal, integración, interpolación, funciones especiales, FFT, procesamiento de señales y de imagen, resolución de ODEs y otras tareas para la ciencia e ingeniería. En nuestro caso hacemos la importación también del wavfile debido a que nuestras canciones están en formato .wav.

- Hashlib: pertenece a la librería estándar y permite realizar cifrados directamente desde Python con diferentes algoritmos como por ejemplo el utilizado en este proyecto el SHA1

- Panda: es una librería open_source que aporta a Python unas estructuras de datos fáciles de usar y de alto comportamiento. En el proyecto lo utilizamos para leer el fichero csv de Matlab.

3.4 Lectura de la Canción

Para poder realizar la lectura de la canción primero de todo lo que hemos hecho es bajar la canción en formato wav. Una vez descargada la canción el programa la tiene que leer para poder comenzar a trabajar con ella. La canción utilitzada como ejemplo en este trabajo es “This One’s for You” Para Matlab

como se puede ver en el código situado en los anexos se realiza con el comando wavread.

Wavread soporta datos multicanal con hasta 32 bits de muestra y admite la lectura de archivos de 24 y 32 bits en .wav.

Una vez ejecutado el comando devuelve la frecuencia de muestreo en Hercios que en nuestro caso es de 44KHz y el número de muestras.

3.5 Espectrograma

El espectrograma es el resultado de calcular el espectro de tramas enventanadas de una señal. El espectrograma es una herramienta básica de representación que se utiliza para el análisis de las señales eléctricas, de comunicaciones, y cualquier señal audiovisual en su contenido frecuencial. Es una representación en tres dimensiones, temporal, frecuencial y amplitud de la distribución de energía de una señal.

El espectrograma se puede interpretar como una proyección en dos dimensiones de una sucesión de Transformadas de Fourier (es una transformación matemática empleada para transformar señales entre el dominio del tiempo (o espacial) y el dominio de la frecuencia) de tramas consecutivas, donde la energía y el contenido frecuencial de la señal va variando a lo largo del tiempo.

El espectrograma consiste en coger un determinado número de muestras por medio de una ventana temporal. A continuación, se hace el cálculo del contenido frecuencial de las muestras puestas en ventana, y se representan en una gráfica en tres dimensiones.

Seguidamente se desplaza la ventana a lo largo del tiempo de la señal, para coger otro número de muestras diferentes, se vuelve a calcular el contenido frecuencial y se vuelve a representar en la misma gráfica que la anterior. Esta operación se repite sucesivamente a lo largo de la señal.

La suma de la representación de las transformadas de Fourier de las ventanas consecutivas aporta información en el dominio frecuencial de la señal, y de la variación de la energía y la frecuencia en función del tiempo.

La gráfica en tres dimensiones puede ser representada de formas diferentes, pero la forma en la que la hemos realizado en este trabajo es la siguiente: el tiempo en el eje de ordenadas, las frecuencias en el eje de ab y una representación de la energía en dB en el plano vertical, está representada con una gama de colores que indican la variación en la energía. Esto se puede observar en la ilustración 7. La energía en dB va de colores rojos a azules. Las partes en que tenemos colores rojos hay mucha energía, en cambio la parte que tenemos colores amarillos o azules no hay tanta energía. Por lo que se observa en la imagen la parte de arriba el color azul oscuro indica que la canción ha acabado porque la energía en esos puntos es prácticamente nula.

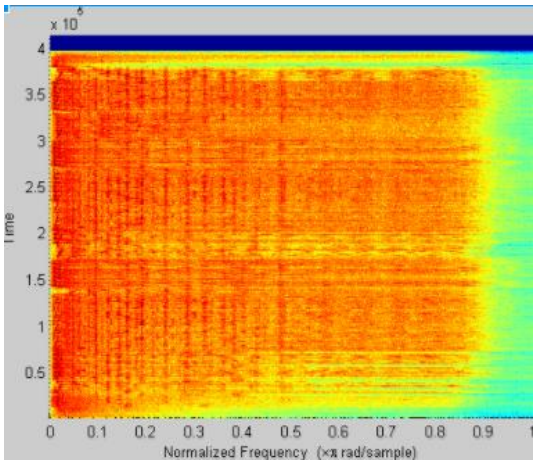


Ilustración 8: Imagen del espectrograma

Para realizar un buen uso del espectrograma tenemos que tener en cuenta diferentes conceptos:

- Señal a la que vamos a realizar el espectrograma en este caso nuestro trozo de audio. Hemos escogido 262144 muestras que corresponde a 2^{18}.
  Tamaño total en muestras de la canción 10406912
  Fragmentos de canción: 40
  Duración de la canción 3 min 55 segundos
  Duración segmento escogido de audio 6 segundos
  Frecuencia de muestreo = 44100 KHz

- Tamaño de la ventana donde divide la señal de entrada en segmentos. El valor escogido ha sido 4098.

- Tipo de ventana: La ventana escogida ha sido de tipo Hann. La ventana de Hann es una combinación lineal de ventanas rectangulares moduladas. De la fórmula de Euler (ecuación 5) debido a las propiedades básicas de la transformada de Fourier su espectro corresponde a la (ecuación 6). Si aplicamos el espectro de la ventana rectangular el factor de modulación desaparece si las ventanas se desplazan en el tiempo alrededor de 0 como se puede observar en la (ecuación 7) [2].

$$w(n) = \frac{1}{2}\,w_r(n) - \frac{1}{4}e^{\mathrm{i}2\pi\frac{n}{N-1}}\,w_r(n) - \frac{1}{4}e^{-\mathrm{i}2\pi\frac{n}{N-1}}\,w_r(n)$$

Ecuación 6: Fórmula de Euler

$$\hat{w}(\omega) = \frac{1}{2}\hat{w}_r(\omega) - \frac{1}{4}\hat{w}_r\left(\omega + \frac{2\pi}{N-1}\right) - \frac{1}{4}\hat{w}_r\left(\omega - \frac{2\pi}{N-1}\right)$$

Ecuación 7: Ecuación del espectro aplicada la Transformada de Fourier

$$\hat{w}_r(\omega) = e^{-i\omega\frac{N-1}{2}} \frac{\sin(N\omega/2)}{\omega/2}$$

Ecuación 8: Ecuación del espectro con la ventana rectangular

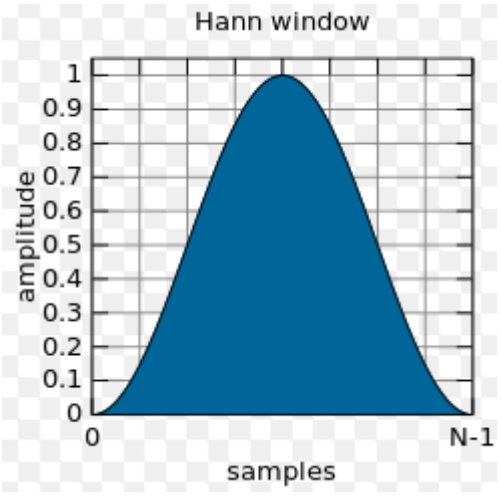


Ilustración 9: Función Hann

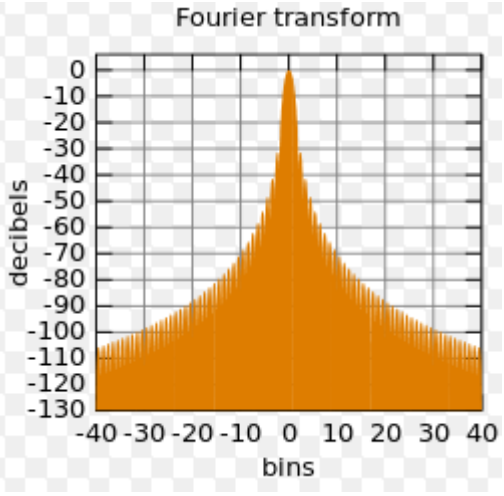


Ilustración 10: Respuesta frecuencial

- Noverlap: Es un valor que depende del overlap. En este proyecto el valor de relación de superposición (overlap) utilizado ha sido 0.5 y por tanto lo que hemos aplicado es la siguiente formula:

$$noverlap = overlap * (tamaño\ de\ la\ ventana)$$

Ecuación 9: Ecuación cálculo noverlap

Como nuestra ventana era de 4098 el noverlap es el valor dividido entre 2 que es 2048.

Una vez realizado el espectrograma obtenemos la FFT. Pero nosotros necesitamos sacar el valor del espectro. Para ello lo primero que hacemos es hacer el módulo y finalmente sacar el espectro con la siguiente operación matemática:

$$espectro(x) = modulo(x)^2$$

Ecuación 10: Cálculo del espectro a partir del espectrograma generado

3.6 Normalización de la imagen

La normalización se hace necesaria, para tener una cierta independencia de las propiedades, como lo son el brillo y el contraste, y así poder comparar huellas por su índice de calidad (QI), que más adelante definiremos.

Para ello necesitamos obtener el valor máximo y mínimo de la matriz generada por la función del espectrograma. Los comandos que se deben utilizar son los siguientes:

Una vez tenemos los valores máximos y mínimos aplicamos la fórmula para normalizar imágenes que es la siguiente:

$$\frac{(nombre\ matriz\ o\ imagen\ a\ normalizar) - Valor\ mínimo}{Valor\ máximo - Valor\ mínimo}$$

Ecuación 11: Fórmula para normalizar la imagen

El resultado es el siguiente

- valor mínimo: -189
- valor máximo: -30

Al realizar la normalización tenemos la matriz siguiente:

0.1739	0.4575	0.6727	0.6373	0.7280	0.6946	0.6667	0.7006	0.7628	0.7091	0.7570	0.7148	0.7361	0.7934	0.7885	0.803
0.1739	0.5407	0.6384	0.6888	0.6985	0.7069	0.7145	0.7393	0.7443	0.7540	0.7529	0.7664	0.7117	0.7961	0.7508	0.783
0.1739	0.5340	0.6412	0.7087	0.7302	0.6617	0.7428	0.7438	0.7576	0.8088	0.7802	0.7752	0.8027	0.8221	0.7439	0.802
0.1740	0.5933	0.6406	0.6995	0.7550	0.7160	0.7533	0.7031	0.7618	0.8176	0.7981	0.7872	0.8040	0.8370	0.8202	0.838
0.1740	0.6129	0.6932	0.7388	0.7317	0.7668	0.7667	0.7567	0.7797	0.7783	0.8461	0.8314	0.8321	0.8618	0.8600	0.840
0.1740	0.6143	0.7132	0.7356	0.7043	0.7753	0.7648	0.7943	0.7856	0.8218	0.8462	0.8313	0.8467	0.8392	0.8384	0.826
0.1740	0.6001	0.7184	0.7324	0.7450	0.7004	0.7696	0.7564	0.7984	0.8178	0.8112	0.7994	0.8347	0.8420	0.7895	0.803
0.1741	0.6185	0.7092	0.7344	0.7557	0.7810	0.7162	0.7859	0.8274	0.7947	0.8029	0.8284	0.8373	0.8292	0.8464	0.842
0.1741	0.6561	0.7021	0.7241	0.7877	0.8102	0.7557	0.8068	0.8544	0.8426	0.8166	0.8446	0.8354	0.8296	0.8501	0.847
0.1741	0.6710	0.6854	0.7702	0.8074	0.8070	0.7881	0.8059	0.8603	0.8796	0.8687	0.8732	0.8563	0.8396	0.8357	0.872
0.1742	0.6677	0.6409	0.7725	0.7870	0.7956	0.7991	0.8007	0.8593	0.8531	0.8830	0.8533	0.7892	0.8286	0.7557	0.858
0.1742	0.6467	0.7110	0.7502	0.7181	0.7919	0.7751	0.8091	0.8366	0.8284	0.8685	0.8200	0.8210	0.8599	0.8500	0.853
0.1743	0.6410	0.7260	0.7857	0.7666	0.7893	0.7712	0.8074	0.8140	0.8558	0.8321	0.8304	0.8424	0.8453	0.8936	0.887
0.1744	0.6680	0.7287	0.7798	0.8035	0.7890	0.8123	0.8270	0.7921	0.8523	0.7826	0.8502	0.8555	0.8142	0.8638	0.872
0.1744	0.6852	0.7497	0.7270	0.7826	0.8061	0.8172	0.8359	0.7690	0.8284	0.8448	0.8490	0.8652	0.8606	0.8610	0.871
0.1745	0.6922	0.7568	0.7585	0.7634	0.6886	0.8065	0.8150	0.8154	0.8709	0.8512	0.8572	0.8895	0.7896	0.8370	0.911
0.1745	0.6877	0.7651	0.7253	0.7723	0.7864	0.7813	0.7619	0.8281	0.8598	0.8601	0.8885	0.8728	0.8610	0.8800	0.907
0.1746	0.6665	0.7683	0.7081	0.8179	0.7887	0.8098	0.8154	0.7732	0.7961	0.8259	0.8829	0.7854	0.8578	0.8081	0.888
0.1747	0.6209	0.7521	0.7789	0.8165	0.7655	0.8057	0.8009	0.7891	0.7633	0.8365	0.8494	0.8616	0.8782	0.8828	0.883
0.1748	0.6461	0.7465	0.7879	0.8008	0.7748	0.7966	0.7570	0.7862	0.7873	0.8270	0.8341	0.8846	0.8908	0.8696	0.886
0.1748	0.6694	0.7253	0.7752	0.7655	0.7599	0.7861	0.7396	0.8073	0.8088	0.8022	0.8371	0.8643	0.8732	0.8166	0.839
0.1749	0.6804	0.7303	0.7500	0.7428	0.7851	0.7830	0.7491	0.8060	0.8305	0.8088	0.8569	0.8470	0.8426	0.8655	0.755

Ilustración 11: Ejemplo trozo matriz normalizada del espectrograma

Como podemos observar en la ilustración al quedar la imagen normalizada todos los valores están entre 0 y 1 negro y blanco respectivamente

3.7 Filtro de máximos

Una vez tenemos la imagen normalizada el siguiente paso es encontrar los máximos locales de esta.

Para ello primero de todo lo que realizamos es la estructura del filtro. El filtro será de generación binaria 1 y 0 (true or false) y siguiendo la forma de cruz tipo al de la ilustración siguiente.

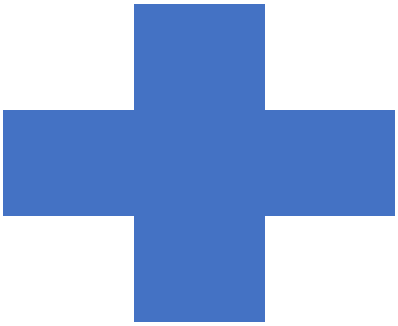


Ilustración 12: Estructura filtro de máximos

Una vez tenemos la estructura del filtro generamos una matriz para poder aplicarla a nuestra imagen. Para generar el filtro lo que hacemos es aplicar esta base sobre ella 20 veces para conseguir un filtro de una dimensión de 41x41. Este filtro como se puede observar en los anexos lo hemos generado en una hoja de Excel y lo hemos leído en el programa a partir de la función xlsread.

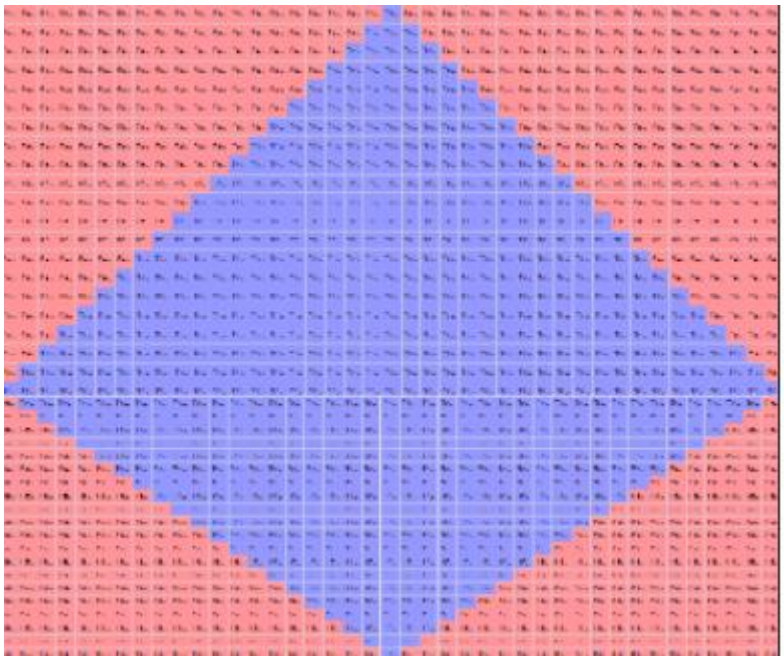


Ilustración 13: Filtro utilizado

Una vez aplicado este filtro no lineal en el espectrograma obtendremos una serie de pixeles que estarán en estado true (1) y otros que estarán en false (0). Cuando hay un true significa que tenemos en ese punto un máximo local.

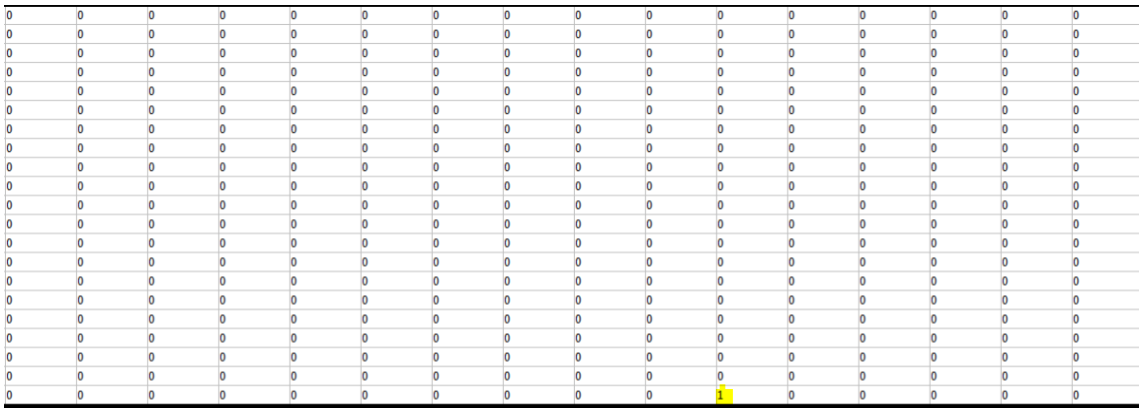


Ilustración 14: Ejemplo trozo matriz del espectrograma una vez aplicado el filtro de máximos

3.8 Filtro de erosión binaria

Una vez hemos aplicado el filtro de máximos en la imagen o matriz el siguiente paso es aplicar el filtro de erosión binaria para eliminar los pixeles que no pertenecen a ser candidatos y que debido al filtro anterior lo son.

Para crear este tipo de filtro primero de todo lo que hacemos es poner en toda la matriz de pixeles de la imagen 0.

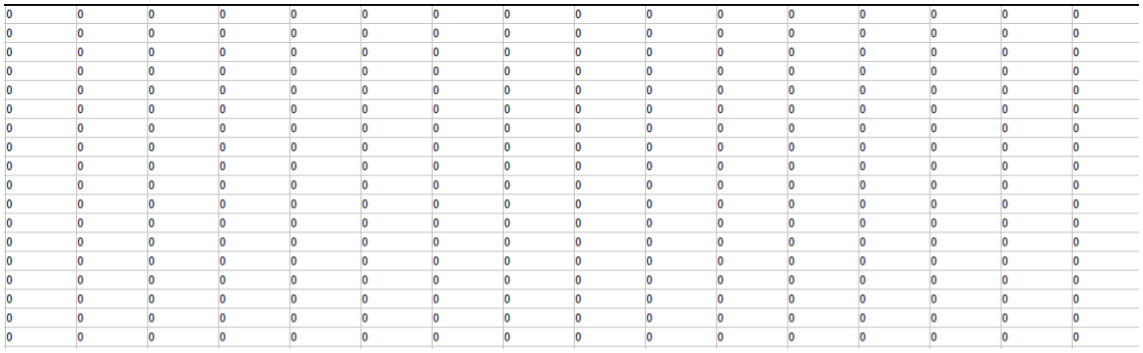


Ilustración 15: Ejemplo trozo matriz del filtro de erosión binaria

Este filtro tiene las siguientes características:

- Entrada: Se utiliza la imagen del espectrograma y se le añade en toda la matriz el booleano false (0)
- Estructura: Utilizamos el mismo filtro que al realizar la dilatación. Los elementos que contengan un valor se consideran true los demás false.

3.9 Detección de puntos candidatos

Una vez hemos diseñado los dos filtros el de máximo y el de erosión binaria ya podemos proceder a encontrar la matriz de detección de puntos candidatos. Para poder obtener esta matriz se realiza la siguiente operación:

$$puntos\ candidatos = filtro\ de\ máximos - filtro\ de\ erosión\ binaria$$

Ecuación 12: Fórmula para detectar los puntos candidatos

La matriz de la imagen obtenida también contiene valores binarios 1, 0. Los que son 1 están en estado true y se consideran como puntos candidatos.

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ilustración 16: Matriz de detección de puntos candidatos

Una vez tenemos la matriz de detección de candidatos tenemos que proceder a ejecutar la extracción de estos puntos. Realizamos dos procedimientos que son los siguientes:

- Procedimiento de encontrar Amplitudes: Encontramos las amplitudes (energía en dB) de los diferentes puntos que se han considerado como candidatos.
- Procedimiento de encontrar el punto (x, y) de la matriz en el cual están estos puntos candidatos. La coordenada x en nuestro caso será el eje de abscisas y la y será el eje de ordenadas. En nuestra imagen, (x) correspondería al tiempo y (y) sería la frecuencia.

Una vez tenemos los puntos candidatos de un fragmento de canción lo que realizamos es un procedimiento para encontrar todos los puntos de toda la canción. Eso se podrá realizar gracias a la opción hold del Matlab donde podremos representar en una misma gráfica el resultado de los X fragmentos de cada canción.

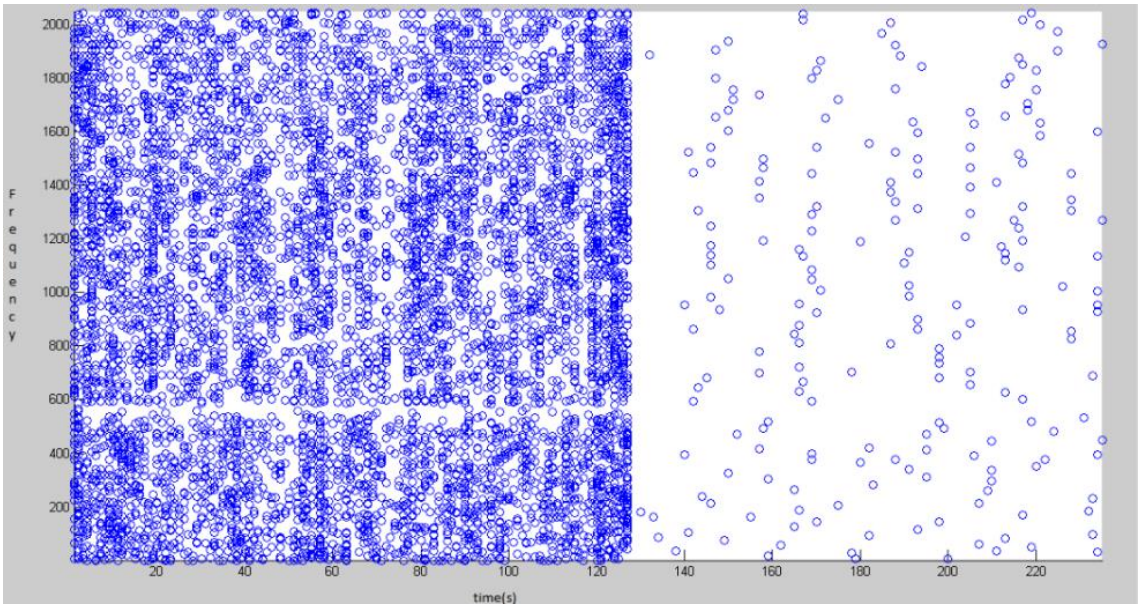


Ilustración 17: Espectrograma final con los puntos candidatos

El siguiente paso antes de exportar el csv es concatenar las diferentes matrices de los X fragmentos de la canción.

Finalmente, cuando tenemos el resultado final exportaremos todos los resultados de los puntos candidatos en un fichero csv que más tarde importaremos en el siguiente paso en el programa Spyder que admite el lenguaje Python donde realizaremos el proceso de Hashing. Esto lo hacemos con la función del Matlab csvwrite.

3.10 Generación del Hash

Una vez tenemos el csv con los diferentes puntos candidatos a partir del lenguaje informático Python generaremos una función para generar los Hashes que serán posteriormente guardados en la base de datos.
Para ello lo primero que tenemos que hacer es leer el fichero csv que hemos generado en Matlab con los diferentes puntos candidatos. Esto lo hacemos con la función pd.read_csv

Una vez esto primeramente sacaremos los valores del csv y generaremos diferentes arrays para poder guardar los diferentes valores necesarios a la hora de hacer el Hashing.

Primero de todo realizaremos un recorrido para poder coger todos los valores de los puntos candidatos y lo que haremos es crear una zona de influencia para poder ejecutar el hashing de cada punto candidato.

Este valor se le conoce como (poner nombre) Cuanto más alto es este valor más fingerprints encontrará ya que la zona de influencia será más grande, pero tendrá un coste computacional más elevado.

El valor escogido después de realizar un breve estudio a nivel de coste computacional con la base de datos de Oracle es 15.

Una vez tenemos la zona de influencia definida empezamos a buscar la información necesaria para el hashing.

Para ello cogemos la frecuencia y el tiempo del primer punto y del punto con el que vamos a emparejarlo. Una vez tenemos los valores de frecuencia y tiempo de los puntos en los que se ha realizado el emparejamiento lo que realizamos es la siguiente operación matemática para encontrar el diferencial en tiempo de estos puntos.

$$\Delta t = t2 - t1$$

Ecuación 13: Fórmula diferencial de tiempo

Finalmente, con la información del diferencial de tiempo más la información de las dos frecuencias de los putos del emparejamiento procedemos a ejecutar el algoritmo SHA1 explicado en la parte de teoría y con el que realizamos el hashing.

Cada uno de los hashes tendrá dos informaciones:

- La primera será el valor del hash en hexadecimal
- La segunda será el tiempo del punto al que se ha realizado el hash. De esta forma tendremos almacenados diferentes hashes para un tiempo determinado.

Una vez tenemos los hashes en el array lo que tenemos que hacer es generar un fichero csv separado por comas ya que lo necesitamos más tarde para hacer la importación en la base de datos.

Para guardarlo en la base de datos lo que haremos es almacenar el Hash en binario y guardaremos el tiempo en segundos al que pertenece ese hash. Para hacer la conversión lo haremos mediante una función en lenguaje PL/SQL explicado en el siguiente punto.

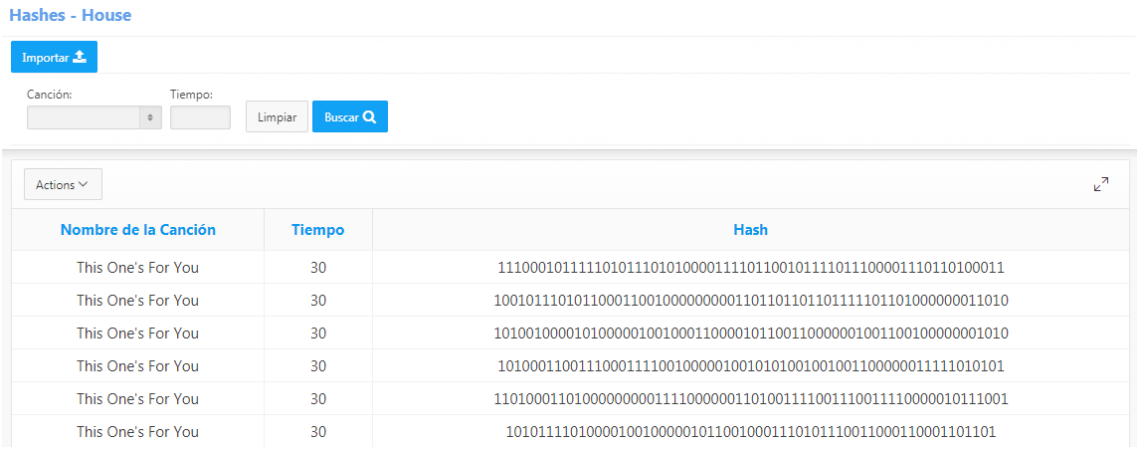


Ilustración 18: Ejemplo de Hashes importados en la web

3.11 Gestión de Base de Datos

Para poder llevar a cabo este tipo de tecnología como hemos comentado necesitamos una base de datos que pueda almacenar mucha información ya que una canción puede contener muchísimos hashes y todos se han de almacenar para poder identificar el audio en cualquier momento de su reproducción.

Para ello hoy en día, por ejemplo, la empresa Oracle ha creado un gestor de base de datos muy útil llamado APEX. APEX(Oracle Application Express) te permite diseñar, desarrollar e implantar aplicaciones responsivas sobre la base de datos usando sólo tu navegador web.

De esta forma mediante el navegador web podemos controlar toda la información que queremos, así como clasificar por tipología. De esta forma tenemos toda la información más ordenada y podemos trabajar mejor con ella.

Antes de explicar muy brevemente como funciona APEX hace falta conocer algo sobre el lenguaje de programación que vamos a utilizar.

El lenguaje de programación utilizado el PL/SQL es un lenguaje de programación incrustado en Oracle. PL/SQL soportará todas las consultas, ya que la manipulación de datos que se usa es la misma que en SQL. Nosotros con este lenguaje crearemos lo que se conoce como un package donde contiene diferentes funciones para poder realizar la importación de un csv y guardar los datos en la base de datos. Este código del package lo podréis ver en el apartado de Anexos.

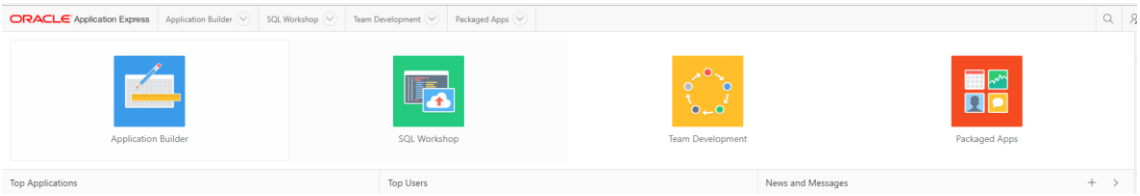
Con Apex nosotros podemos gestionar un gran contenido de información de forma rápida y eficaz.

Primero de todo necesitaremos crear un Workspace (Entorno de trabajo) y tener acceso a una instancia de base de datos de Oracle.

Una vez témenos esto ya podemos acceder a crear la aplicación:

- Primero de todo hacemos un clic en el boton application builder

- Más tarde accedemos a crear la aplicación. Se podrá crear el gestor de base de datos vía interfaz web o vía móvil.



Cuando tenemos creada la aplicación nos aparecerá el aplicativo en la herramienta.

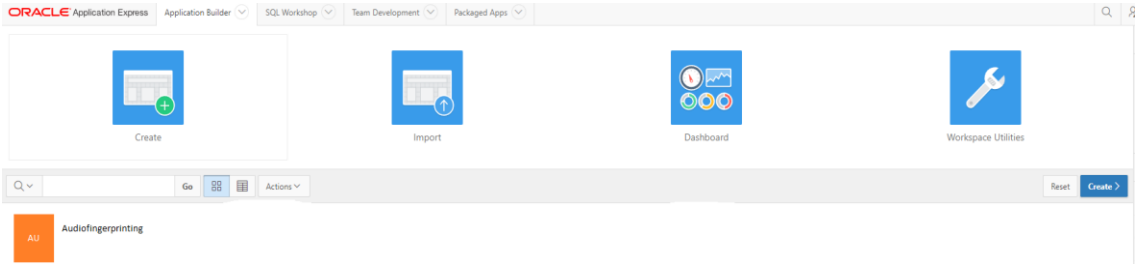


Ilustración 19: Herramienta APEX

Si hacemos clic en ella solo tendremos la página de Login hecha, por tanto, necesitaremos crear las páginas convenientes como interactive report para que pueda aparecer la tabla con toda la información de los hashes de las canciones.

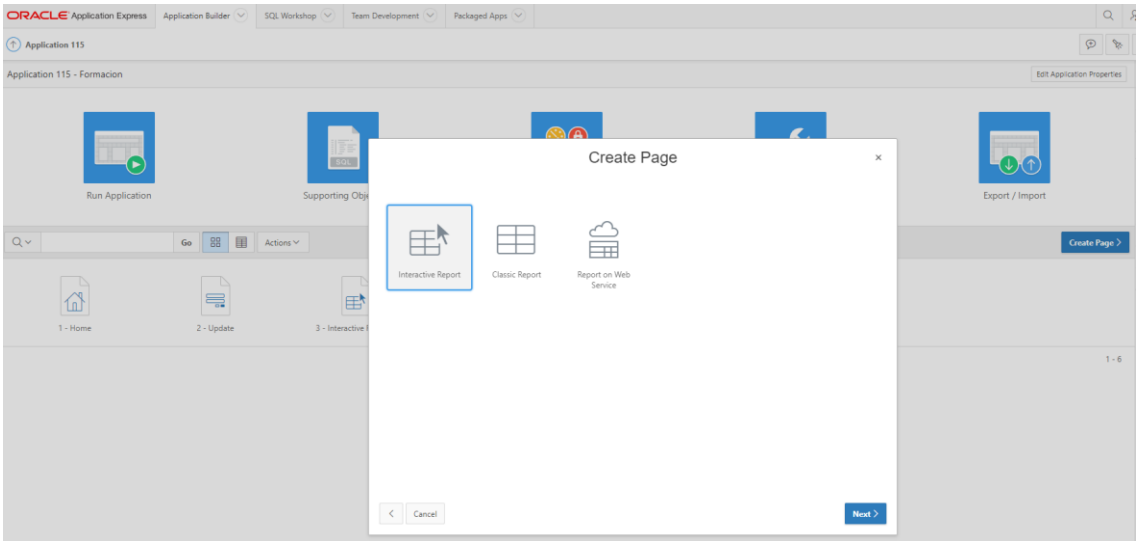


Ilustración 20: Herramienta APEX (Creación de páginas)

También una parte clave de esta herramienta es el menú de navegación ya que en ella es donde haremos la clasificación sobre los temas. Por ejemplo si el audiofingerprinting es de canciones en el navigation menu estará la clasificación (Rock, Pop, Pop-rock, Punk, Heavy-Metal, House, jazz…) o si es de anuncios estarán (Deportes, Productos de Limpieza, Apuestas, Productos de Higiene Personal, Programas Televisivos…) sobretodo de cara a los anuncios es muy útil ya que la cadena de televisión tendría una ventaja muy importante ya que por ejemplo si la emisión televisiva es un partido de baloncesto, la mejor publicidad a emitir es la de deportes o apuestas.

En el aplicativo existirán dos partes:

- Datos Maestros: Creación y edición de nuevas canciones
- para cada estilo musical Hashes: Página donde se podrán ver para cada estilo musical los hashes de las diferentes canciones

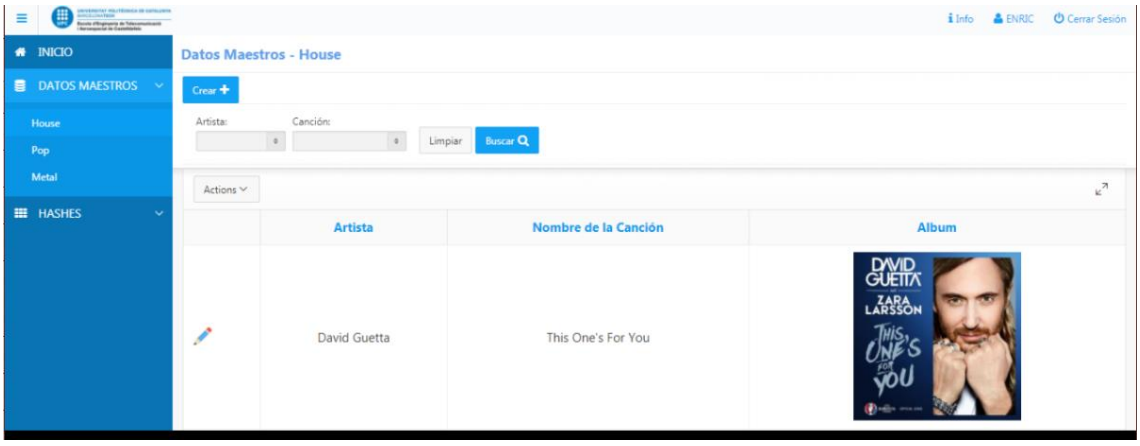


Ilustración 21: Aplicación Audiofingerprint APEX

En el apartado de Datos maestros tenemos una región de busqueda donde podremos buscar las diferentes canciones del género musical creado en el menu de navegación o bien por artista o bien por el nombre de la canción.

También podremos crear canciones nuevas que despues serán necesarias para poder generar los hashes. En ella se pondrá el nombre de la canción, el artista y la foto de album. Esta se guardaría en la base de datos en formato BLOB.

También podremos editar la información seleccionando el lapiz del reporte donde esta toda la información

Ilustración 22: Aplicación Audiofingerprint APEX (Edición/Creación de una canción)

En el apartado de Hashes tenemos una región de busqueda donde podremos buscar los diferentes hashes o por cación o por tiempo.

A partir de esta pagina será donde realizaremos la importación de los ficheros csv generados por Python:

- Primero de todo caragremos el fichero csv generado

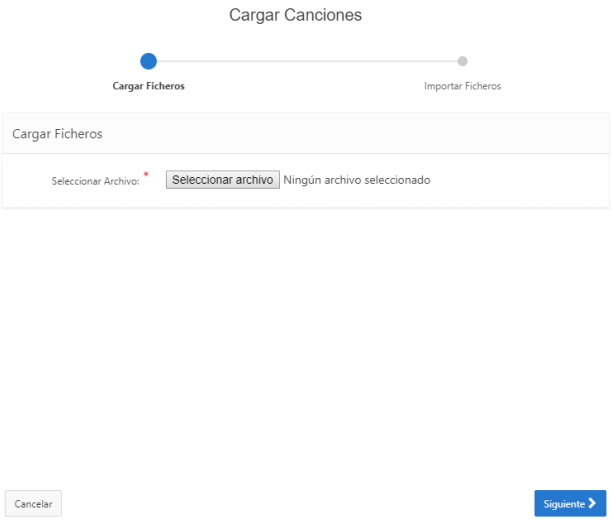


Ilustración 23: Aplicación Audiofingerprint APEX (Importar Hashes a una canción)

- Seguidamente como hemos comentado anteriormente a partir del package de importación escrito en lenguaje PL/SQL guardamos los datos en una tabla temporal del servidor de APEX.
- Finalmente en el ultimo paso meteremos toda la información en nuestra base de datos y podremos ver el informe con el nombre de la canción el hash y el tiempo al que pertenece ese hash.

  En cada uno de los reportes podremos mostrar la información que queramos de las tablas donde están almacenados todos los hashes.

  La información se tendrá que sacar con una select utilizando el lenguaje de programación SQL en la casilla donde está marcado en rojo en la siguiente ilustración.

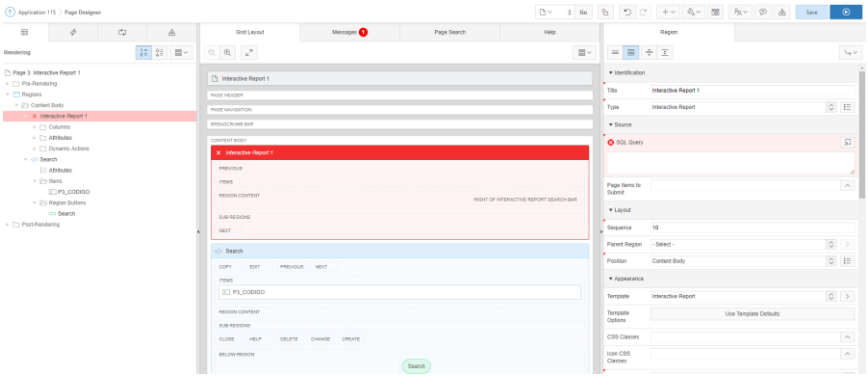


Ilustración 24: Aplicación Audiofingerprint APEX (Creación del reporte)

CONCLUSIONES

Este proyecto se basa en el estudio del audiofingerprinting para la identificación automática de contenidos audiovisuales. En él, hemos hecho el estudio de uno de los métodos que se utilizan para la detección de canciones a partir de la generación de hashes.

Para lograr este objetivo primero hemos expuesto los fundamentos teóricos en los que se sustenta el proyecto. En este apartado tenemos la descripción de la metodología y describimos los diferentes conceptos uno por uno y con detalle para poder entender el funcionamiento del audio fingerprint desde los aspectos más globales de la tecnología hasta los conceptos más detallados como la función Hash.

A continuación, se ha implementado un ejemplo para poder simular de forma práctica la identificación automática de canciones y se ha generado una aplicación web donde hemos podido ver la información que contiene cada una de las huellas que se generan en la base de datos. Durante el desarrollo hemos podido observar la rapidez del sistema de hashing ya que se procesa un gran volumen de información a una velocidad muy alta.

Para implementar todo el desarrollo se ha utilizado tres programas informáticos con sus tres lenguajes propios, los cuáles hemos desarrollado nuestros propios programas o funciones para la identificación de huellas en audio.

También haciendo un estudio del tiempo que tarda en procesar los hashes de una canción creo que podría ser muy útil el método de cara a anuncios, pero considero que de cara a películas o series se queda un poco limitado.

Por tanto una de las mejoras que se tendrían que hacer de cara a utilizarlo en series o películas es la ampliación de las capacidades de la base de datos ya que la información que se guardaría seria aproximadamente 25 veces la de una canción y por tanto el sistema actualmente tendría un gran coste computacional.

REFERENCIAS

[1] Avery Li-Chun WangWang - 2003 - An Industrial Strength Audio Search Algorithm.pdf

[2] Definición y caracteristicas de la ventana Hann. Disponible en https://es.wikipedia.org/wiki/Ventana_(funci%C3%B3n)#Hann

[3] Pedro Cano, Eloi Batlle, Emilia Gómez, Leandro de C.T.Gomes yMadeleine Bonnet.Chapter 17. Audio Fingerprinting: Concepts And Applications Audio Fingerprinting: Concepts and Applications, 2005, pp. 233- 245

[4] Ayuda del Matlab

[5] Wes Hatch.A Quick Review of Audio Fingerprinting. Marzo 2003.

[6] Página web de Stackoverflow (http://www.stackoverflow.com)

[7] Definición y caracteristicas del algoritmo de Hashing SHA1. Disponible en https://en.wikipedia.org/wiki/SHA-1

[8] Definición y caracteristicas de la función Hash. Disponible en https://en.wikipedia.org/wiki/Hash_function

[9] Ayuda gestor base de datos. Disponible en https://docs.oracle.com/cd/E59726_01/index.html

[10] Ayuda creación packages y funciones para importar ficheros csv. https://docs.oracle.com/cd/B10501_01/appdev.920/a96624/09_packs.htm

ANEXOS

A.1 Código en Matlab para encontrar puntos candidatos

```
%%Leemos la canción escogida This One's for You
[rate, data] = wavread('Tfg.wav');
%%Escogemos un fragmento de Canción
channel1 = rate(100000:362144,1);
%%Declaramos las variables necesarias (frecuencia de muestreo,
noverlap, ventana, tamaño de la ventana)
N=4096
fs=44100;
noverlap=1/2*N;
ventana=hanning(N);
%%Generamos el espectrograma
arr2D = (1/N)*spectrogram(channel1,ventana,noverlap,N,fs);
%%Generamos el espectro a partir del espectrograma recibido
modulo=abs(arr2D);
spectro= modulo.^2;
p= 10 * log10(spectro);
%%Realizamos la normalización
valor_minimo=min(min(p));
valor_maximo=max(max(p));
img_bw=(p-valor_minimo)/(valor_maximo-valor_minimo);
img_test=img_bw(1:8,1:8);
%%Leemos el filtro que vamos a utilizar
filt=xlsread('filtro');
%Encontramos los máximos locales a partir del filtro de dilatación
local_max=imdilate(p,filt)==p;
%Generamos un filtro de erosión
background = (p==0);
eroded_background = imerode(background,filt);
%Encontramos los puntos candidatos y los ploteamos
detected_peaks = local_max-eroded_background;

amps =p(find(detected_peaks==1));
[I J]=find(detected_peaks==1);
axis([1 145 1 2048])
hold on

plot(J,2048-I,'o')

hold off
```

A.2 Código en Python para generar Hashes

```
import numpy as np
import matplotlib.mlab as mlab
import scipy.io.wavfile
import pandas as pd

import matplotlib.pyplot as plt

from scipy.ndimage.filters import maximum_filter
from scipy.ndimage.morphology import (generate_binary_structure,
                                      iterate_structure,
binary_erosion)
import hashlib
from operator import itemgetter


######################################################################
# Degree to which a fingerprint can be paired with its neighbors --
# higher will cause more fingerprints, but potentially better
accuracy.
DEFAULT_FAN_VALUE = 15


######################################################################
# Number of cells around an amplitude peak in the spectrogram in order
# for Dejavu to consider it a spectral peak. Higher values mean less
# fingerprints and faster matching, but can potentially affect
accuracy.
PEAK_NEIGHBORHOOD_SIZE = 20

######################################################################
# Thresholds on how close or far fingerprints can be in time in order
# to be paired as a fingerprint. If your max is too low, higher values
of
# DEFAULT_FAN_VALUE may not perform as expected.
MIN_HASH_TIME_DELTA = 0
MAX_HASH_TIME_DELTA = 10000

######################################################################
# If True, will sort peaks temporally for fingerprinting;
# not sorting will cut down number of fingerprints, but potentially
# affect performance.
PEAK_SORT = True

######################################################################
# Number of bits to throw away from the front of the SHA1 hash in the
# fingerprint calculation. The more you throw away, the less storage,
but
# potentially higher collisions and misclassifications when
identifying songs.
FINGERPRINT_REDUCTION = 16
# leemos el fichero csv y sacamos los valores x,y
l_total_res=pd.read_csv("csvoutputres.csv", sep=",",header=None)
l_total_res.values
test= len(l_total_res);
x=l_total_res[0]
z=l_total_res[1]
#if PEAK_SORT:
```

```
# Creamos los diferentes arrays necesarios para guardar todos los
hashes
l_res = []
l_time = []
l_binary = []
hex_str = []
hex_int = []
new_int = []
l_x = range(1,test)
l_y = range(1,DEFAULT_FAN_VALUE)
peaks.sort(key=itemgetter(1))
for x in (l_x):
      for y in (l_y):
        if (x + y) < len(peaks):


                freq1 = peaks[x][1]
                freq2 = peaks[x + y][1]
                t1 = peaks[x][0]
                t2 = peaks[x + y][0]
                t_delta = t2 - t1
                if t_delta >= MIN_HASH_TIME_DELTA and t_delta <=
MAX_HASH_TIME_DELTA:
                    h = hashlib.sha1(
                        "%s|%s|%s" % (str(freq1), str(freq2),
str(t_delta)))
                    l_res.append
(h.hexdigest()[0:FINGERPRINT_REDUCTION])
                    l_time.append(t1)
                    hashes=zip(l_res,l_time)
                    datos = np.asarray(hashes)
lengthres = range(1,len(l_res))

   # Generamos el fichero csv.
np.savetxt("outputdef6.csv",   # Archivo de salida
          datos,         # Trasponemos los datos
          fmt="%s",        # Usamos números enteros
           delimiter=",")
```

A.3 Package para importar ficheros csv

```
create or replace package body apexc_data_loader
--función que carga el fichero csv y lo mete en una tabla temporal
FUNCTION load_csv(
            p_data_load_name VARCHAR2,
            p_csv_content  IN BLOB,
            p_separator IN VARCHAR2 := g_default_separator,
            p_skip_rows IN NUMBER   := 0,
            p_register_batch BOOLEAN DEFAULT FALSE,
            p_file_name VARCHAR2 DEFAULT NULL)
        RETURN NUMBER
    IS
        l_e_data_load_exception EXCEPTION;
        l_e_error_batch_register  EXCEPTION;

        l_batch_id NUMBER;
```

```
l_data_load_name VARCHAR2(50);
l_current_rows NUMBER;

CURSOR l_cur_csv
IS
    SELECT
        LINE_NUMBER,
        C001,
        C002,
        C003,
        C004,
        C005,
        C006,
        C007,
        C008,
        C009,
        C010,
        C011,
        C012,
        C013,
        C014,
        C015,
        C016,
        C017,
        C018,
        C019,
        C020,
        C021,
        C022,
        C023,
        C024,
        C025,
        C026,
        C027,
        C028,
        C029,
        C030,
        C031,
        C032,
        C033,
        C034,
        C035,
        C036,
        C037,
        C038,
        C039,
        C040,
        C041,
        C042,
        C043,
        C044,
        C045,
        C046,
        C047,
        C048,
        C049,
        C050,
        C051,
        C052,
        C053,
        C054,
```

```
            C055,
            C056,
            C057,
            C058,
            C059,
            C060,
            C061,
            C062,
            C063,
            C064,
            C065,
            C066,
            C067,
            C068,
            C069,
            C070,
            C071,
            C072,
            C073,
            C074,
            C075,
            C076,
            C077,
            C078,
            C079,
            C080,
            C081,
            C082,
            C083,
            C084,
            C085,
            C086,
            C087,
            C088,
            C089,
            C090,
            C091,
            C092,
            C093,
            C094,
            C095,
            C096,
            C097,
            C098,
            C099,
            C100
        FROM TABLE(apexc_data_loader.blob_to_csv(
            p_csv_blob  => p_csv_content,
            p_separator => p_separator,
            p_skip_rows => p_skip_rows)) ;
BEGIN
    l_data_load_name := TRIM(UPPER(p_data_load_name));
    l_batch_id := apexc_data_loads_seq.nextval;
    l_current_rows := 0;

    FOR l_csv_line IN l_cur_csv LOOP
        INSERT
        INTO APEXC_DATA_LOADS
            (
                DATA_LOAD_NAME,
                BATCH_ID,
```

```
LINE_NUMBER,
C001,
C002,
C003,
C004,
C005,
C006,
C007,
C008,
C009,
C010,
C011,
C012,
C013,
C014,
C015,
C016,
C017,
C018,
C019,
C020,
C021,
C022,
C023,
C024,
C025,
C026,
C027,
C028,
C029,
C030,
C031,
C032,
C033,
C034,
C035,
C036,
C037,
C038,
C039,
C040,
C041,
C042,
C043,
C044,
C045,
C046,
C047,
C048,
C049,
C050,
C051,
C052,
C053,
C054,
C055,
C056,
C057,
C058,
C059,
C060,
```

```
        C061,
        C062,
        C063,
        C064,
        C065,
        C066,
        C067,
        C068,
        C069,
        C070,
        C071,
        C072,
        C073,
        C074,
        C075,
        C076,
        C077,
        C078,
        C079,
        C080,
        C081,
        C082,
        C083,
        C084,
        C085,
        C086,
        C087,
        C088,
        C089,
        C090,
        C091,
        C092,
        C093,
        C094,
        C095,
        C096,
        C097,
        C098,
        C099,
        C100,
        CREATION_DATE,
        CREATED_BY,
        LAST_UPDATE_DATE,
        LAST_UPDATED_BY
    )
    VALUES
    (
        l_data_load_name,
        l_batch_id,
        l_csv_line.line_number,
        l_csv_line.C001,
        l_csv_line.C002,
        l_csv_line.C003,
        l_csv_line.C004,
        l_csv_line.C005,
        l_csv_line.C006,
        l_csv_line.C007,
        l_csv_line.C008,
        l_csv_line.C009,
        l_csv_line.C010,
        l_csv_line.C011,
```

```
l_csv_line.C012,
l_csv_line.C013,
l_csv_line.C014,
l_csv_line.C015,
l_csv_line.C016,
l_csv_line.C017,
l_csv_line.C018,
l_csv_line.C019,
l_csv_line.C020,
l_csv_line.C021,
l_csv_line.C022,
l_csv_line.C023,
l_csv_line.C024,
l_csv_line.C025,
l_csv_line.C026,
l_csv_line.C027,
l_csv_line.C028,
l_csv_line.C029,
l_csv_line.C030,
l_csv_line.C031,
l_csv_line.C032,
l_csv_line.C033,
l_csv_line.C034,
l_csv_line.C035,
l_csv_line.C036,
l_csv_line.C037,
l_csv_line.C038,
l_csv_line.C039,
l_csv_line.C040,
l_csv_line.C041,
l_csv_line.C042,
l_csv_line.C043,
l_csv_line.C044,
l_csv_line.C045,
l_csv_line.C046,
l_csv_line.C047,
l_csv_line.C048,
l_csv_line.C049,
l_csv_line.C050,
l_csv_line.C051,
l_csv_line.C052,
l_csv_line.C053,
l_csv_line.C054,
l_csv_line.C055,
l_csv_line.C056,
l_csv_line.C057,
l_csv_line.C058,
l_csv_line.C059,
l_csv_line.C060,
l_csv_line.C061,
l_csv_line.C062,
l_csv_line.C063,
l_csv_line.C064,
l_csv_line.C065,
l_csv_line.C066,
l_csv_line.C067,
l_csv_line.C068,
l_csv_line.C069,
l_csv_line.C070,
l_csv_line.C071,
l_csv_line.C072,
```

```
                    l_csv_line.C073,
                    l_csv_line.C074,
                    l_csv_line.C075,
                    l_csv_line.C076,
                    l_csv_line.C077,
                    l_csv_line.C078,
                    l_csv_line.C079,
                    l_csv_line.C080,
                    l_csv_line.C081,
                    l_csv_line.C082,
                    l_csv_line.C083,
                    l_csv_line.C084,
                    l_csv_line.C085,
                    l_csv_line.C086,
                    l_csv_line.C087,
                    l_csv_line.C088,
                    l_csv_line.C089,
                    l_csv_line.C090,
                    l_csv_line.C091,
                    l_csv_line.C092,
                    l_csv_line.C093,
                    l_csv_line.C094,
                    l_csv_line.C095,
                    l_csv_line.C096,
                    l_csv_line.C097,
                    l_csv_line.C098,
                    l_csv_line.C099,
                    l_csv_line.C100,
                    SYSDATE,
                    apexc_user_manager.get_user_id,
                    SYSDATE,
                    apexc_user_manager.get_user_id
                );
                l_current_rows := l_current_rows + 1;
                IF(l_current_rows > 50000) THEN --Partial commits
every 50000 records for big data loads
                    l_current_rows := 0;
                    COMMIT;
                END IF;
        END LOOP;

        COMMIT;
--Si se produce un error hace un borrar de la tabla temporal
        DELETE FROM apexc_data_loads
        WHERE
            c001 IS NULL
            AND c002 IS NULL
            AND c003 IS NULL
            AND c004 IS NULL
            AND c005 IS NULL
            AND c006 IS NULL
            AND c007 IS NULL
            AND c008 IS NULL
            AND c009 IS NULL
            AND c010 IS NULL
            AND c011 IS NULL
            AND c012 IS NULL
            AND c013 IS NULL
            AND c014 IS NULL
            AND c015 IS NULL
            AND c016 IS NULL
```

```
AND c017 IS NULL
AND c018 IS NULL
AND c019 IS NULL
AND c020 IS NULL
AND c021 IS NULL
AND c022 IS NULL
AND c023 IS NULL
AND c024 IS NULL
AND c025 IS NULL
AND c026 IS NULL
AND c027 IS NULL
AND c028 IS NULL
AND c029 IS NULL
AND c030 IS NULL
AND c031 IS NULL
AND c032 IS NULL
AND c033 IS NULL
AND c034 IS NULL
AND c035 IS NULL
AND c036 IS NULL
AND c037 IS NULL
AND c038 IS NULL
AND c039 IS NULL
AND c040 IS NULL
AND c041 IS NULL
AND c042 IS NULL
AND c043 IS NULL
AND c044 IS NULL
AND c045 IS NULL
AND c046 IS NULL
AND c047 IS NULL
AND c048 IS NULL
AND c049 IS NULL
AND c050 IS NULL
AND c051 IS NULL
AND c052 IS NULL
AND c053 IS NULL
AND c054 IS NULL
AND c055 IS NULL
AND c056 IS NULL
AND c057 IS NULL
AND c058 IS NULL
AND c059 IS NULL
AND c060 IS NULL
AND c061 IS NULL
AND c062 IS NULL
AND c063 IS NULL
AND c064 IS NULL
AND c065 IS NULL
AND c066 IS NULL
AND c067 IS NULL
AND c068 IS NULL
AND c069 IS NULL
AND c070 IS NULL
AND c071 IS NULL
AND c072 IS NULL
AND c073 IS NULL
AND c074 IS NULL
AND c075 IS NULL
AND c076 IS NULL
AND c077 IS NULL
```

```
            AND c078 IS NULL
            AND c079 IS NULL
            AND c080 IS NULL
            AND c081 IS NULL
            AND c082 IS NULL
            AND c083 IS NULL
            AND c084 IS NULL
            AND c085 IS NULL
            AND c086 IS NULL
            AND c087 IS NULL
            AND c088 IS NULL
            AND c089 IS NULL
            AND c090 IS NULL
            AND c091 IS NULL
            AND c092 IS NULL
            AND c093 IS NULL
            AND c094 IS NULL
            AND c095 IS NULL
            AND c096 IS NULL
            AND c097 IS NULL
            AND c098 IS NULL
            AND c099 IS NULL
            AND c100 IS NULL;

        COMMIT;

        IF(p_register_batch) THEN
            IF(p_file_name IS NOT NULL) THEN
                register_batch(p_batch_id => l_batch_id, p_file_name
=> p_file_name);
            ELSE
                dbms_output.put_line('Error while registering batch:
Register batch is true but null file name specified');
                RAISE l_e_error_batch_register;
            END IF;
        END IF;

        RETURN l_batch_id;
        EXCEPTION
            WHEN OTHERS THEN
                delete_data_load(p_data_load_name => p_data_load_name,
p_batch_id => l_batch_id);
                RAISE l_e_data_load_exception;
    END;

    --Procedimiento que borra la tabla temporal segun el batch_id que
se genera al insertar el fichero
     PROCEDURE delete_data_load(p_data_load_name VARCHAR2,
        p_batch_id NUMBER DEFAULT -1,
        p_user_id NUMBER DEFAULT apexc_user_manager.get_user_id)
    IS
    BEGIN
        IF(p_batch_id > 0) THEN
            DELETE FROM apexc_data_loads adl
            WHERE UPPER(adl.data_load_name) = UPPER(p_data_load_name)
            AND (adl.batch_id = p_batch_id)
            AND (adl.created_by = p_user_id OR p_user_id IS NULL);
        ELSE
            DELETE FROM apexc_data_loads adl
            WHERE UPPER(adl.data_load_name) = UPPER(p_data_load_name)
            AND (adl.created_by = p_user_id OR p_user_id IS NULL);
```

```
        END IF;
        COMMIT;
    END;
    --Procedimiento que registra el batch al importar un nuevo fichero
      PROCEDURE register_batch(p_batch_id NUMBER, p_file_name
VARCHAR2)
    IS
        l_e_already_exists_batch EXCEPTION;
    BEGIN
        INSERT INTO apexc_data_load_batches(
            batch_id,
            file_name,
            creation_date,
            created_by,
            last_update_date,
            last_updated_by)
        VALUES (
            p_batch_id,
            p_file_name,
            sysdate,
            apexc_user_manager.get_user_id,
            sysdate,
            apexc_user_manager.get_user_id);

        COMMIT;
        EXCEPTION
            WHEN OTHERS THEN
                dbms_output.put_line('An error ocurred in batch
register, batch already exists');
                RAISE l_e_already_exists_batch;
    END;
```

A.4 Función para convertir Hexadecimal a Binario

```
declare
    l_number number;
    l_result varchar2(150);
  cursor c_hash is
      select c001,c002
        from apexc_data_loads;
begin
  for i in c_hash loop
  --convertimos el hexadecimal a decimal
  select to_number( i.c001, 'xxxxxxxx' )
    into l_number
    from dual;
    --convertimos el decimal a binario
  select dec2bin(l_number)
    into l_result
    from dual;
    --insertamos en la tabla final los registros de la tabla temporal
insert into songs (hash, tiempo, canción, song_type)
           values (l_result, i.c002, :P38_CANCION, :P38_TYPE);
  end loop;
end;
```