Trabajo Fin de Grado
Ingeniería de las Tecnologías de Telecomunicación

Software para identificación de música

Autor: Valentín Montero Jiménez

Tutor: Juan Ramón Cerquides Bueno

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Ingeniería de las Tecnologías de Telecomunicación

Software para identificación de música

Autor:
Valentín Montero Jiménez

Tutor:
Juan Ramón Cerquides Bueno
Profesor Titular

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: Software para identificación de música

Autor: Valentín Montero Jiménez
Tutor: Juan Ramón Cerquides Bueno

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Re sumir en unas pocas líneas todo lo vivido durante estos años de aprendizaje sería imposible ya que son innumerables las experiencias vividas, no me bastarían los dedos de las manos para contar las grandes personas que he conocido en este periodo. Un agradecimiento especial para todos aquellos compañeros de clase que terminaron siendo amigos, no solo entre las paredes de la escuela.
Pero no todo durante estos años fueron luces, por eso quiero dedicar estas líneas de mi trabajo para homenajear a todas las personas que me apoyaron en los momentos más complicados, a mis padres, familiares, amigos y profesores.
Especial mención a Ramón Cerquides por su guía y apoyo como tutor.

Valentín Montero Jiménez
Sevilla, 2020

Resumen

El objetivo de este trabajo es la recreación del algoritmo de Shazam en el lenguaje matemático MATLAB, así como el estudio de su robustez y comparación con otras técnicas. Para ello, profundizaremos en el desarrollo de los frameworks que tienen como base la identificación a partir de audio fingerprints extraídas de las características relevantes de las señales.
El análisis del algoritmo lo divideremos en dos fases: en la primera trabajaremos en la extracción de fingeprints, englobando de esta forma todo tratamiento previo que reciben las señales de audio y su obtención a partir del espectro, para posteriormente, en la segunda fase, poder realizar la identificación del fragmento de audio.
Posteriormente, valoraremos las posibles modificaciones que resultan prácticas en un mundo tecnológico, donde la fiabilidad no es suficiente si no va acompañada de una velocidad y eficiencia acordes a ella. Además, estudiaremos la variación de parámetros hacia la obtención de unos resultados óptimos para que la aplicación sea reproducible en cualquier dispositivo móvil.
Con el objetivo de justificar el uso de las técnicas utilizadas para el desarrollo del algoritmo, dedicamos un capítulo al procesamiento de señales, siendo el foco la Transformada de Fourier y su aplicación en el cálculo del espectrograma, para así poder concluir con una serie de demostraciones experimentales que respalden las conclusiones obtenidas.

Abstract

The aim of this work is to recreate Shazam's algorithm in the mathematical language MATLAB, as well as study its strength and comparison with other techniques. In order to do so, we will deepen the development of the frameworks, which are based on the identification from audio fingerprints taken/extracted from the relevant characteristics of the signals.
The analysis of the algorithm will be separated in two phases: in the first phase we will work on the extraction of fingeprints, thus covering all previous treatment that the audio signals receive and their obtention from the spectrum, and later, in the second phase, we will be able to identify the audio fragment.
Afterwards, we will evaluate the possible modifications that are useful in a technological world, where reliability is not enough if it is not accompanied by a speed and efficiency according to it. In addition, we will study the variation of parameters towards obtaining optimal results so that the application is reproducible in any mobile device.
In order to justify the use of the techniques used for the development of the algorithm, we dedicate a chapter to signal processing, being the focus the Fourier Transform and its application in the calculation of the spectrogram, to be able to conclude with a series of experimental demonstrations that support the conclusions obtained.

Índice

Notación

$\mathbb{R}$	Cuerpo de los números reales
$\mathbb{C}$	Cuerpo de los números complejos
$\|\mathbf{v}\|$	Norma del vector $\mathbf{v}$
$\langle \mathbf{v}, \mathbf{w} \rangle$	Producto escalar de los vectores $\mathbf{v}$ y $\mathbf{w}$
$\|\mathbf{A}\|$	Determinante de la matriz cuadrada $\mathbf{A}$
$\det(\mathbf{A})$	Determinante de la matriz (cuadrada) $\mathbf{A}$
$\mathbf{A}^\top$	Transpuesto de $\mathbf{A}$
$\mathbf{A}^{-1}$	Inversa de la matriz $\mathbf{A}$
$\mathbf{A}^\dagger$	Matriz pseudoinversa de la matriz $\mathbf{A}$
$\mathbf{A}^{\mathrm{H}}$	Transpuesto y conjugado de $\mathbf{A}$
$\mathbf{A}^*$	Conjugado
c.t.p.	En casi todos los puntos
c.q.d.	Como queríamos demostrar
■	Como queríamos demostrar
□	Fin de la solución
e.o.c.	En cualquier otro caso
e	número e
$\mathrm{e}^{\mathrm{j}x}$	Exponencial compleja
$\mathrm{e}^{\mathrm{j}2\pi x}$	Exponencial compleja con 2π
$\mathrm{e}^{-\mathrm{j}x}$	Exponencial compleja negativa
$\mathrm{e}^{-\mathrm{j}2\pi x}$	Exponencial compleja negativa con 2π
$\mathbb{R}\mathrm{e}$	Parte real
$\mathbb{I}\mathrm{m}$	Parte imaginaria
sen	Función seno
tg	Función tangente
arc tg	Función arco tangente
$\sin^y x$	Función seno de x elevado a y
$\cos^y x$	Función coseno de x elevado a y
Sa	Función sampling
sgn	Función signo
rect	Función rectángulo
Sinc	Función sinc
$\frac{\partial y}{\partial x}$	Derivada parcial de y respecto a x
x°	Notación de grado, x grados.
$\Pr(A)$	Probabilidad del suceso A
$E[X]$	Valor esperado de la variable aleatoria X
σ_X^2	Varianza de la variable aleatoria X
$\sim f_X(x)$	Distribuido siguiendo la función densidad de probabilidad $f_X(x)$
$\mathcal{N}(m_X, \sigma_X^2)$	Distribución gaussiana para la variable aleatoria X, de media m_X y varianza σ_X^2

$\mathbf{I}_n$	Matriz identidad de dimensión n
$\mathrm{diag}(\mathbf{x})$	Matriz diagonal a partir del vector $\mathbf{x}$
$\mathrm{diag}(\mathbf{A})$	Vector diagonal de la matriz $\mathbf{A}$
SNR	Signal-to-noise ratio
MSE	Minimum square error
$:$	Tal que
$\stackrel{\mathrm{def}}{=}$	Igual por definición
$\lVert\mathbf{x}\rVert$	Norma-2 del vector $\mathbf{x}$
$\lvert\mathbf{A}\rvert$	Cardinal, número de elementos del conjunto $\mathbf{A}$
$\mathbf{x}_i, i = 1,2,\ldots,n$	Elementos i, de 1 a n, del vector $\mathbf{x}$
$\mathrm{d}x$	Diferencial de x
$\leqslant$	Menor o igual
$\geqslant$	Mayor o igual
$\backslash$	Backslash
$\Leftrightarrow$	Si y sólo si
$x = a + 3 \underset{\substack{\uparrow \\ a=1}}{=} 4$	Igual con explicación
$\frac{a}{b}$	Fracción con estilo pequeño, a/b
Δ	Incremento
$b \cdot 10^a$	Formato científico
$\underset{x}{\rightarrow}$	Tiende, con x
O	Orden
™	Trade Mark
$\mathbb{E}[x]$	Esperanza matemática de x
$\mathbf{C}_\mathbf{x}$	Matriz de covarianza de $\mathbf{x}$
$\mathbf{R}_\mathbf{x}$	Matriz de correlación de $\mathbf{x}$
σ_x^2	Varianza de x

1 Introducción

«Si no conozco una cosa, la investigaré».
Louis Pasteur.

Formamos parte de una sociedad moderna deseosa por tener en la palma de la mano acceso a todo tipo de servicios en tiempo real, estando continuamente expuestos al bombardeo sobre las ventajas y aplicaciones del internet de las cosas (IoT), cuya finalidad no es otra que la interconexión de dispositivos de forma automática para así poder compartir datos entre ellos y permitir un acceso rápido a servicios que nunca habíamos imaginado. Este tipo de aplicaciones son diversas, abarcando herramientas para la automatización de la temperatura del hogar, control de electrodomésticos e incluso la conducción autónoma.
Los asistentes de voz son una de las aplicaciones más populares. Estos implementan un gran número de funcionalidades como programar alarmas, realizar llamadas, encender o apagar las luces, reproducir música, etc. Cabe destacar entre sus utilidades la detección de piezas musicales, ya que esto despierta un gran interés dentro del reconocimiento de voz. La idea surge de la necesidad de conectar a las personas con una gran base de datos musical que permita, en un periodo muy corto de tiempo, identificar la canción que está sonando en la radio, en la tele o incluso en un pub.

El **objetivo principal** de este trabajo es la implementación del algoritmo de reconocimiento de las piezas musicales mas populares del mercado en el lenguaje de programación MATLAB, así como realizar un estudio de la robustez en diferentes situaciones de ruido que emulen un entorno real.
Gracias a la versatilidad del algoritmo, su utilidad evolucionó constantemente según las necesidades de la época, hasta el punto de no limitarse a identificar fragmentos de canción. Su uso se fue extendiendo hacia temas relacionados con la verificación de la reproducción de anuncios en la radio, comprobar si un artista utilizaba playback en sus conciertos, copyright etc.
En sus inicios la aplicación de Shazam solo estaba disponible en algunos países de Europa. El nombre oficial de lanzamiento era “2580”[1], que a su vez era el código necesario para realizar la llamada con un máximo de 30 segundos. Esta pista de audio era procesada y emparejada unos segundos mas tarde el usuario recibía el resultado vía SMS.
Las funcionalidades se fueron expandiendo, ya no solo determinaba a que canción pertenecía y su autor, sino que también te redireccionaba a un lugar web donde poder comprar la canción o un tono de llamada. Estas búsquedas sirvieron para empezar a sugerir otras canciones y grupos, la conocida hoy en día como publicidad personalizada. Este tema trajo controversia allá por el año 2014 [2] cuando aún el grosso de la población era más disconforme a la hora de aceptar que las empresas vendían su información personal en forma de paquetes de datos a otras de forma anónima para obtener publicidad personalizada, se descubrió que Shazam enviaba de manera oculta datos personales a al menos dos portales de publicidad.

Shazam nace como empresa en 1998 de la mano de un grupo de estudiantes de la universidad de Berkeley, California a partir de una firma consultora con sede en Londres, para llevar a cabo la aplicación necesitaron de un experto en el tratamiento de señales. En ese momento reclutan a Avery Wang estudiante de Stanford[1], Avery Wang junto a su equipo desarrolla un algoritmo novedoso y robusto ante el ruido basado en un hash combinacional que supone un avance espectacular respecto a las técnicas anteriores, su algoritmo no solo

tenía un grado muy bajo de error también supuso una reducción del coste computacional que a su vez se vería reflejado en el tiempo de detección. La robustez de un algoritmo se mide en su capacidad de identificar correctamente un fragmento de una canción en situaciones de ruido distintas.
Antes de analizar como funciona el algoritmo así como su implementación dedicamos el capítulo 2 a profundizar sobre los inconvenientes de trabajar con la forma de onda de una señal en su dominio temporal frente al dominio frecuencial. La forma de onda está compuesta por la superposición de varias ondas a distinta frecuencia entre las cuales se encuentra el ruido. El espectrograma permite descomponer la señal en sus distintos componentes de frecuencia, de esta forma podemos atenuar la frecuencia a la que se encuentre el ruido.
La cancelación del ruido en una señal no es el único motivo para trabajar en el dominio frecuencial, ya que comparar dos señales a partir de sus formas de ondas no es eficiente. Una base de datos puede contener millones de canciones con una media 10MB, trabajar directamente con formas de ondas en su dominio temporal supondría un coste computacional tan alto que obligaría a trabajar en súper ordenadores para que los tiempos de búsqueda fuesen aceptables. Necesitamos un algoritmo que pueda funcionar sobre todos los móviles que están en el mercado, algo que esté al alcance de todos.
Para cumplir con todas las expectativas propuestas necesitamos un algoritmo basado en huellas de audio. Una huella de audio o audio fingerprint como lo llamaremos a partir de ahora por la popularidad del término anglosajón, se trata de un identificador de tamaño reducido basado en el contenido de un fragmento de audio o canción. Un audio fingerprint será la unidad básica mínima a partir de la que se pondrán enlazar dos fragmentos de audio.

A partir de las características relevantes obtenidas a partir del espectrograma conseguimos los fingerprints, aplicamos el mismo proceso tanto a las canciones que incluiremos en la base de datos, como a los fragmentos de audio a analizar. En el postratamiento del espectrograma eliminaremos el ruido e irrelevancias que no nos aporten nada a la búsqueda, quedándonos con las características que presenten mayor resistencia al ruido, los picos de energía. La aportación novedosas de Avery Wang consiste en el emparejamiento de máximos en el espectro a partir de la posición de los máximos emparejados y su distancia en tiempo y frecuencia calcularemos un valor hash, identificativo.

Si nuestro objetivo general es lograr una implementación completa del algoritmo en MATLAB, el objetivo específico es conseguir un equilibrio entre el coste computacional y la robustez del algoritmo, para ello dividimos la investigación en etapas.
El desarrollo se ha llevado a cabo a partir de un método de **investigación cuantitativo** dividiendo la investigación en 3 etapas, la **investigación descriptiva** tiene lugar en el capítulo 2 donde se explican los fundamentos teóricas sobre las que se sustentas las aplicaciones y técnicas que veremos posteriormente en el capítulo 3, donde vemos las características generales que presentan los frameworks basados en audio fingerprints.
En la segunda etapa de **investigación analítica** en los capítulos 3 y 4 analizamos el algoritmo de Shazam desarrollado por Avery Wang, así como otras técnicas presentando una comparativa entre ellas. La técnica basada en el emparejamiento de máximos es la más popular por el alcance obtenido por Shazam pero no es la única válida, existen una serie de algoritmos de gran interés de estudio con unos resultados notables.
La última etapa de **investigación experimental** en el capítulo 4.2 realizamos un estudio exhaustivo de la robustez del algoritmo de Shazam sobre nuestra implementación en MATLAB. En el capítulo 4.3 realizaremos una serie de estudios aplicando distintos niveles de ruido sobre las pistas de audio. Para aumentar la profundidad valoraremos ciertos parámetros que pueden ser modificados en el algoritmo buscando un punto óptimo entre la robustez y la velocidad asociada a una disminución del coste computacional.

Hasta el momento se han expuesto muchas posibles aplicaciones para técnicas basadas en audio fingerprints partiendo del trabajo previo realizado por el profesor Dan Ellis de la universidad de Columbia.[3]

2 Fundamentos Teóricos

Una de las virtudes del ingeniero es la eficiencia.

Guang Tse

Los fundamentos teóricos son la base para entender la motivación y el desarrollo del estudio realizado. En este capítulo mostraremos los conocimientos fundamentales necesarios del procesamiento de señales, como son la descomposición de la señal en senos y cosenos, el espectro de frecuencia para entender las características de los fenómenos ondulatorios, los formantes, la Transformada de Fourier y sus aplicaciones en el espectrograma o cómo afecta una ventana temporal para evitar la fuga espectral.

2.1 Espectro de frecuencia

El espectro de frecuencia se caracteriza por la distribución de amplitudes para cada frecuencia de un fenómeno ondulatorio ya sea sonoro, luminoso o electromagnético. Estos fenómenos nacen de la superposición de ondas de varias frecuencias.[4] En el caso de la luz, gracias a un prisma transparente, podemos apreciar como una fuente de luz está compuesta por fotones con distinta frecuencia. El espectro luminoso muestra la intensidad para cada componente de frecuencia, esto nos permite apreciar cuál es la cantidad de cada color que compone la fuente de luz. De la misma forma, las fuentes sonoras pueden descomponerse en una superposición de ondas de frecuencias diferentes, denominado espectro sonoro.

Las aplicaciones que estudiaremos tienen la finalidad de la identificación de canciones pero son la base de futuros estudios relacionados con la detección de voz, por ello necesitamos introducir el concepto de formante[5].

2.2 Formantes

Los formantes son bandas de frecuencia donde se concentra la mayor parte de la energía sonora, y es gracias a ellos por lo que podemos distinguir los diferentes sonidos del habla humana.
Desde el punto de vista fisiológico, el habla se produce por la resonancia del tracto vocal. Son inicialmente las cuerdas vocales las que producen ondas sonoras. Sin embargo, estas presentan un espectro muy distribuido, siendo en el tracto vocal donde unas frecuencias salen potenciadas o atenuadas dando lugar a los sonidos que producimos habitualmente para comunicarnos.[6]

Serán dichas frecuencias son las formantes principales, ya que la mayoría de los sonidos resultan de la superposición de varias ondas a distintas frecuencias. Para identificarlos nos encontramos con varios formantes ordenados de forma creciente, siendo F1 el formante de frecuencia más baja. Normalmente es suficiente con la información obtenida de los dos primeros formantes, pero en lenguajes con un mayor número de vocales son necesarios hasta 4.
[1]

[1] De Davius - Trabajo propio, CC0, https://commons.wikimedia.org/w/index.php?curid=17076270

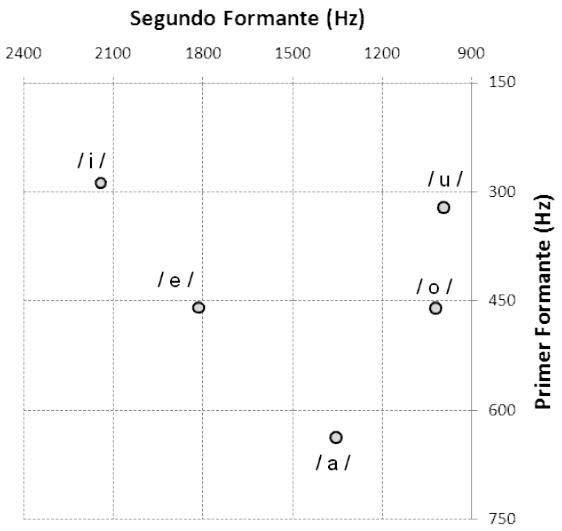


Figura 2.1 Formantes Idioma Español.

2.3 La Transformada de Fourier

Muchas de las aplicaciones que usamos día a día funcionan bajo la misma premisa, la descomposición de la señal en su espectro frecuencial. Una aplicación muy común es la radio de nuestro coche. Para el funcionamiento de esta, nos llega una serie de señales moduladas, y al elegir una frecuencia particular, podemos escuchar una cadena u otra.
El tratamiento del ruido en las señales sería impensable sin la descomposición en frecuencia, ya que permite un estudio muy concreto del ruido, de forma que podemos potenciar y, en nuestro caso, atenuar las frecuencias que deseemos. Otra aplicación un tanto menos conocida fuera de nuestro campo es el estudio de las vibraciones sísmicas, permitiendo construir edificios e infraestructuras que resistan o acompañen levemente el movimiento de estas, e incluso, evitar los derrumbes.

Volviendo al enfoque principal del trabajo, en matemáticas el análisis de Fourier proporciona un estudio armónico de una señal, debido a la descomposición de esta en frecuencias[7]. Trabajar sobre el dominio frecuencial nos ha permitido realizar un análisis preciso sobre qué el ancho de banda en el que se concentra la energía de una señal, así como el estudio de la fase de las señales[8].
La fase representa la posición en el ciclo, siendo esta una fracción del periodo. Para su cálculo y representación son necesarios los números complejos, implícitos en la ecuación que veremos posteriormente en el estudio de la transformada de Fourier.

2.4 Transformada rápida de Fourier(FFT)

La FFT permite calcular de forma eficiente la DTF, es la técnica más utilizada en los software matemáticos debido a las ventajas que presenta en términos de eficiencia. En Matlab, la función que utilizamos para calcular el espectrograma, usa este algoritmo más eficiente de la transformada de Fourier, es por ello por lo que haremos especial hincapié en ella. [21]

Para evaluar dicha operación de forma directa, se requiere de

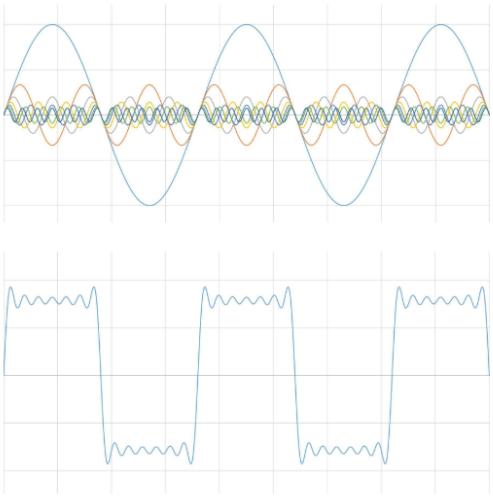
$$N^2 \tag{2.1}$$

operaciones aritméticas, mientras que con el algoritmo FFT, se puede obtener el mismo resultado con solo

$$Nlog(N) \tag{2.2}$$

operaciones.

La idea principal es dividir la transformada en otras más simples para luego ir escalando el algoritmo en niveles superiores. A través de esta descomposición llegamos a transformadas de 2 elementos, donde k solo puede tomar valores binarios. Resueltas las transformadas más simples, hay que agruparlas en otras de nivel



a

Figura 2.2 Descomposición de una señal en sus componentes de frecuencia.

[a] https://www.ni.com/es-es/innovations/white-papers/06/understanding-ffts-and-windowing.html#section-1472285423

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi k \frac{n}{N}}, \qquad k = 0, \ldots, N-1.$$

Figura 2.3 Transformada Fourier.

superior, para después llevar a cabo este proceso de subida de nivel hasta llegar al más alto, en el cual se deben reordenar los resultados.

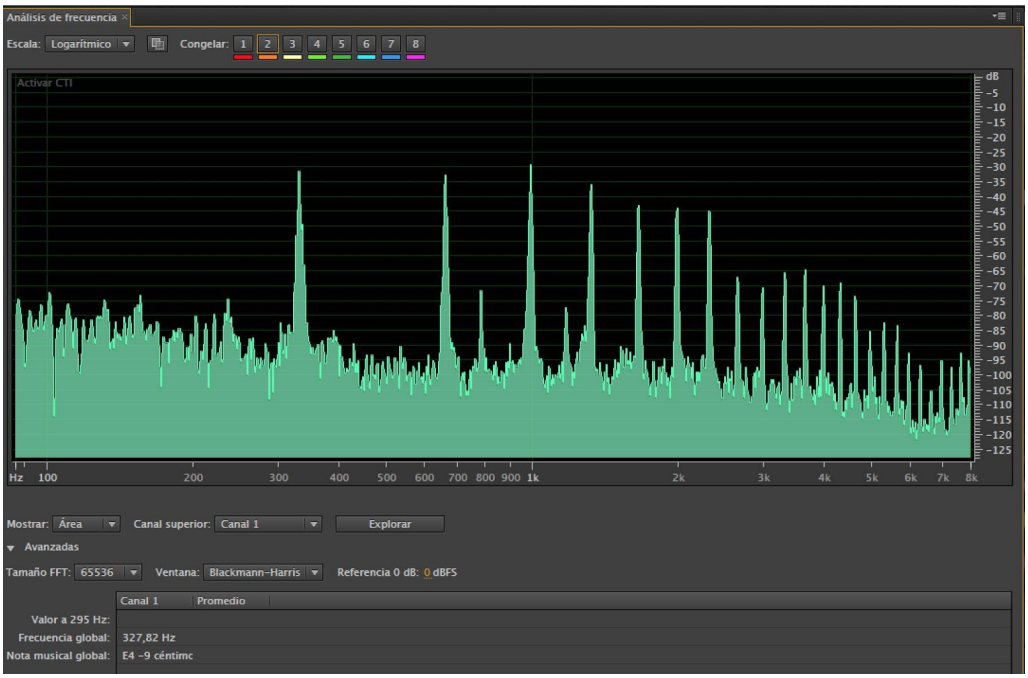
2.5 Espectrograma

El espectrograma es una representación 3-D de la evolución temporal del espectro de una señal, representa la variación del contenido frecuencial a lo largo del tiempo. Nos proporciona la distribución de amplitudes para cada frecuencia de un fenómeno ondulatorio compuesto por la superposición de otras ondas de frecuencia variable. Es decir, obtenemos la energía del contenido frecuencial de la señal.
El espectrograma se calcula según los principios básicos expuestos en la explicación de la Transformada rápida de Fourier, la descomposición de la transformada en elementos de orden inferior.

En el espectrograma, la descomposición se hace a partir de ventanas temporales con un tamaño variable según el análisis que queramos realizar[10]. Si deseamos hacer un estudio armónico de la señal, priorizamos una mejor resolución en el lóbulo principal temporal, para ello elegimos una ventana de banda estrecha[Figura 4]. En cambio, si nuestro deseo es estudiar la estructura resonante de la señal, usamos una ventana de banda ancha, ya que la formación de las palabras necesitamos una representación de la cavidad resonante ya que es el lugar donde se generan. [Figura 5]

2.6 Ventana temporal

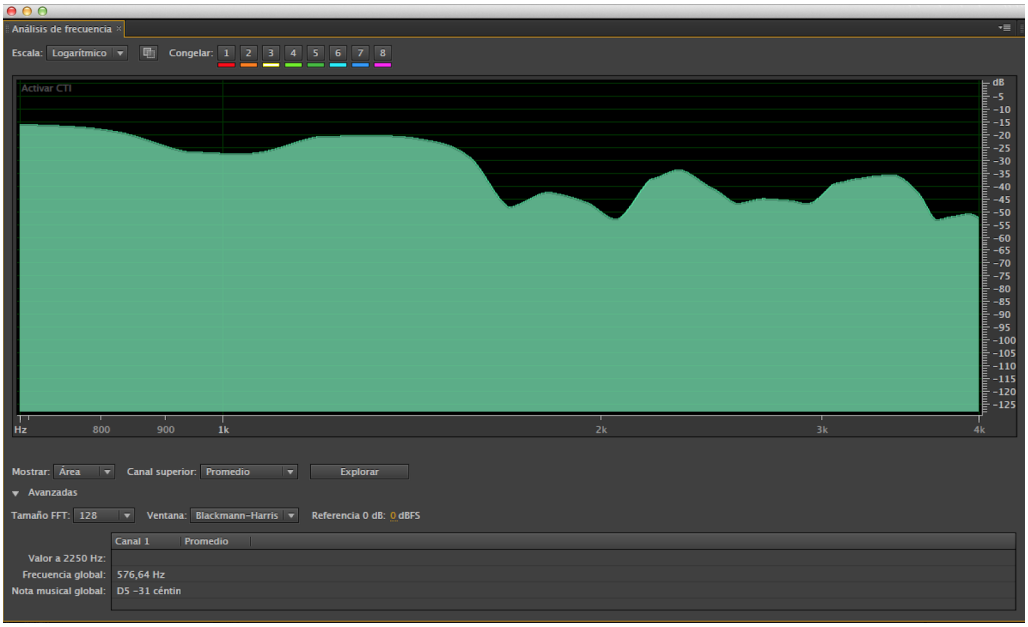
Una ventana temporal desde el punto de vista matemática representa una gráfica real con un espectro continuo compuesto por un lóbulo principal y varios lóbulos laterales. Mientras que el lóbulo principal se centra en cada componente de frecuencia de la señal de dominio temporal, los laterales se aproximan a cero. La altura de estos indica el efecto que tendrán sobre las frecuencias altas alejadas del lóbulo principal.



a

Figura 2.4 Análisis armónico. Banda estrecha.

[a] De Albertcastan - Trabajo propio, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=37028435



a

Figura 2.5 Análisis armónico. Banda ancha.

[a] De Albertcastan - Trabajo propio, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=37028435

Las ventanas más típicas son: Rectangular, Hanning, Hamming, Blackman, Blackman-Harris, Blackman-Nuttall,Flat top, Gauss, Triangular, Bartlett, Bartlett-Hann y Kaiser.

El resultado es una representación tridimensional donde podemos ver el la suma del contenido frecuencial de la señal mientras va variando sucesivamente a lo largo de la señal según avanza la ventana sobre el tiempo.

La representación de la energía suele verse expresada en decibelios como el módulo de la amplitud de la transformada de Fourier.[10]

El desplazamiento de la ventana suele ser solapada con el fin de asegurar que no se pierda la información entre tramas

3 Audio Fingerprint System

No entiendes realmente algo a menos que seas capaz de explicárselo a tu abuela.

ALBERT EINSTEIN

Existe una gran variedad de técnicas de reconocimiento de piezas de audio basadas en su contenido, las cuales se engloban en los sistemas audio fingerprint. En este capítulo estudiaremos la estructura general que siguen todos los sistemas, partiendo de la premisa sobre la eficiencia de utilizar un elemento discriminante basado en el contenido de la canción en lugar de trabajar con formas de onda completas. Como estudiaremos posteriormente, una huella de audio o audio fingerprint, llamado así a partir de ahora por la popularidad del término anglosajón, se trata de un identificador de tamaño reducido basado en el contenido de un fragmento de audio o canción. Un fingerprint será la unidad básica mínima a partir de la que se pondrán enlazar dos fragmentos de audio[11]. Para ello, generamos una base de datos de fingerprints con millones de canciones a las que aplicaremos el mismo proceso de extracción.

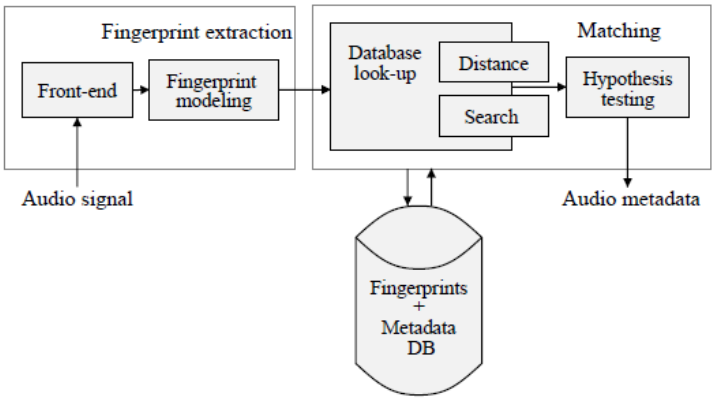


Figura 3.1 Content-Based Audio Identification Framework.

Para entender su éxito, necesitamos comprender el por qué se utilizan y cómo se generan, desarrollando para ello cierto contexto histórico. Pese a que la impresión que obtenemos con la definición vista de un audio fingerprint, esta no será la única, al igual que tampoco será su única utilidad. Ciertamente, en sus inicios, este término fue asociado a los algoritmos watermarking, cuya funcionalidad consiste en añadir a la señal de audio original un mensaje independiente que no generase una diferencia perceptible para el ser humano, de forma que permitía verificar la integridad de la señal. Siguiendo el mismo mecanismo, es posible determinar de qué canción se trata o detectar la transmisión ilegal de canciones con copyright[12]. Sin embargo, esta definición difiere totalmente con los algoritmos audio fingerprint actuales basados en el contenido de la señal(Content BID) y por tanto, más complejos computacionalmente.

3.1 Watermarking

Como hemos visto anteriormente, los fingerprint en los algoritmos watermarking no están basados en el contenido de las señales de audio, sino que se generan de forma independiente y son únicos para cada canción. Sirven para la detección de una canción e incluso de unos segundos con distorsiones. Sin embargo, su uso principal viene relacionado con el copyright. Estos algoritmos están implementados en las aplicaciones con la intención de automatizar y asegurar que no se violan los derechos de autor. El mecanismo es mucho más simple que los basados en el contenido de la señal, ya que se añade un mensaje identificativo en la señal original a una frecuencia imperceptible para el ser humano, pero de forma que esta sea identificable. En este caso, no solo se aplica al mundo de la música, también es muy común encontrarlo en pruebas forenses o incluso a la hora de verificar la integridad de las declaraciones en un juicio.[12]

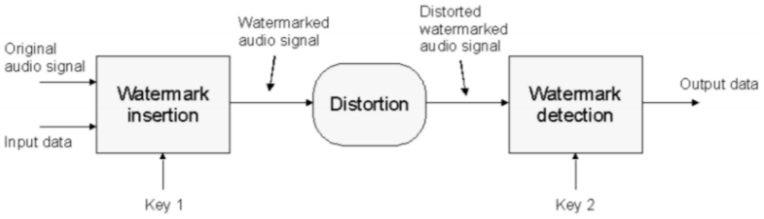


Figura 3.2 Watermarking System.

La clave va encriptada en la señal, incluso en las copias piratas, el audio puede escucharse sin ningún problema a pesar del encriptamiento, este solo afecta al watermarking. Como podemos ver, hacen falta dos clave: una para encriptar y otra para desencriptar, pudiendo ser iguales o no.[13] Si las claves son iguales estaríamos hablando de un sistema simétrico. En contraste, tenemos los sistemas asimétricos, donde la clave de encriptado es privada, sin embargo la de desencriptado es pública.

Existe una variedad de técnicas que difieren en la forma de añadir el mensaje a la señal. Además de añadir el mensaje a una frecuencia inaudible del espectro[14], también encontramos técnicas que enmascaran el mensaje en el "eco" de la señal original [15]. Existen otros mecanismos más complejos donde la marca de agua se inserta directamente en el flujo de bits generados por el codificador de audio[16].

3.2 Audio Fingerprint

Cuando analizamos las técnicas anteriores de identificación de piezas de audio basadas en la forma de onda de la señal, observamos la ineficiencia que supone trabajar con tan ingentes cantidades de información para cada canción. Una aproximación eficiente consiste en obtener una representación binaria de la canción a partir de una función hash como MD5 (Message Digest 5) or CRC (Cyclic Redundancy Checking). Sin embargo, su robustez y escalabilidad es escasa, ya que para dos señales perceptualmente iguales o muy parecidas, sus formas de onda pueden ser muy diferentes simplemente teniendo en cuenta el formato de almacenamiento o la compresión, dando lugar a valores hash que nada tienen que ver entre sí.

Motivado por este inconveniente nacen los sistemas basados en el contenido de la señal(CBID), a partir de características de la señal resistentes al ruido, alimentaremos un sistema de modelado de audio fingerprints[17] cimentado en el contenido de la señal con una gran versatilidad ante distintos formatos y robustez ante el ruido.

La creación de las fingerprints está estrechamente relacionado con la capacidad perceptual de los seres humanos, siendo este el comportamiento a imitar. Es por este motivo por el que se trabaja principalmente en el dominio frecuencial de la señal, dividiendo el espectro de energía de la señal en bandas logarítmicas que se asemejan a los niveles de percepción humanos. Del filtrado de características relevantes del espectro de la señal obtenemos una serie de fingerprints que permiten sacar una representación compacta del contenido de la señal a partir de una función hash, dando lugar a un vector de longitud fija que servirá como índice en la base de datos, en la que cada celda apunta a la canción y el frame a partir del que se calculó.

En la mayoría de aplicaciones [18] la obtención se hace a partir de dividir la canción en frames(n) de aproximadamente 370 ms, calculando para cada frame un sub-fingerprint o valor hash de 32 bits. Tras calcular el espectro, con objetivo de asemejarse a la percepción humana, se divide 33 bandas y se obtiene la energía para cada una de ellas. Análogamente, un fingerprint se conforma de 256 sub-fingerprint consecutivos.

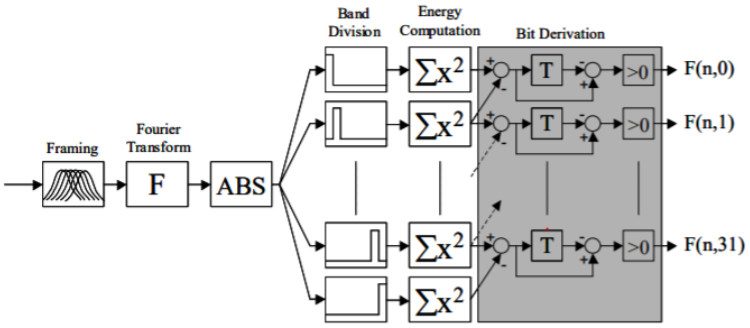


Figura 3.3 Calculo espectrograma y filtro binario posterior.

Una de las formas más comunes de obtener una sub-fingerprint es la aplicación de un filtro para sub-banda de energía que compone el espectro, obteniendo según el umbral, una representación binaria de 32 bits para cada banda(m) de cada frame(n).

$$F(n,m)=\begin{cases}1 \text{ if } E(n,m)-E(n,m+1)-\big(E(n-1,m)-E(n-1,m+1)\big)>0\\ 0 \text{ if } E(n,m)-E(n,m+1)-\big(E(n-1,m)-E(n-1,m+1)\big)\leq 0\end{cases}$$

Figura 3.4 Calculo espectrograma y filtro binario posterior.

No hay una forma única de modelado de fingerprints ni la forma de trabajar con ellas, sin embargo, hay una serie de parámetros que debe cumplir cualquier algoritmo basado en un sistema audio fingerprint[19]. La **robustez** del algoritmo es uno de los parámetros clave, para ello es muy importante la elección de características relevantes de la señal que sean resistentes ante el ruido. Se mide como la capacidad de obtener una identificación correcta a pesar de que las distorsiones a partir de un fragmento de audio sean de corta duración. Esta duración necesaria para realizar la identificación se mide en la **granularidad**. La **fiabilidad** mide el porcentaje de canciones identificadas incorrectamente, conocido en el estudio como un falso positivo. El **tamaño** de los fingerprints es otro de los parámetros más importantes limitantes en vista de la eficiencia del algoritmo, cuando trabajamos con millones de canciones necesitamos que estos sean lo más compacto posibles, no solo por el almacenamiento, sino también por lo que conlleva el coste computacional de trabajar con ellos, viéndose reflejado directamente en la **velocidad de búsqueda**, así como en la **escalabilidad** que mide el comportamiento del sistema ante una base de datos de tamaño considerable. **Versatilidad**, mide la capacidad del algoritmo de trabajar correctamente con distintos formatos de canciones, así como ser capaz de utilizar la base de datos para distintas aplicaciones.

Siguiendo el patrón visto para los algoritmos fingerprint, veremos en el siguiente capítulo una comparativa general de los frameworks, dividiendo por etapas donde veremos en cada una de ellas distintas aplicaciones y referencias a técnicas.

3.3 Estructura General Audio Fingerprint Systems

Esta visión general de algoritmos basados en sistemas audio fingerprint está cimentada en el estudio de Pedro y Eloi Battle de la Universidad Pompeu Fabra en colaboración con Ton Kalker y Jaap Haitsma del centro de investigación de Philips en Eindhoven[20].

Cuando valoramos las características que debe tener un sistema de identificación, pensamos en el grado de discriminación alcanzado a pesar del ruido y la compresión a la que se vea expuesta la señal. Pero eso no es todo en el mundo de la tecnología; la velocidad y el tamaño también son dos puntos muy importantes. La eficiencia es la característica más limitante, ya que encontrar el equilibrio entre la velocidad de búsqueda, la complejidad de la extracción de los fingerprints, su tamaño y ser capaz de obtener un alto porcentaje de acierto, son los objetivos primordiales.

Estos Framework presentan una serie de características y procedimientos generales comunes. Sin embargo, las particularidades de estos framework son notables a la hora de transformar la señal y elegir que características son las más fuertes ante las distorsiones en vista de poder extraer fingerprints que cumplan los requisitos descritos. En este capítulo estudiaremos la estructura general dividida en dos bloques, la extracción de fingerprints y la Identificación(Matching).

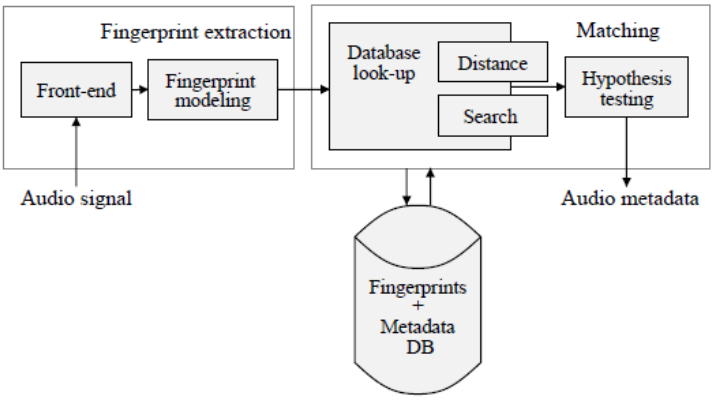


Figura 3.5 Content-Based Audio Identification Framework.

3.3.1 Estructura FRONT-END

En el "FRONT-END" trabajamos con la señal original, a la que aplicamos una serie de técnicas con la finalidad de eliminar la redundancia facilitando su posterior procesado y transformación. En este bloque tratamos los fundamentos del tratamiento de señales, técnicas cuyas bases son la teoría de la información, la estadística y las matemáticas aplicadas. En este apartado evaluaremos desde la conversión de una señal analógica a digital hasta el tratamiento del espectrograma cuya base es la Transformada de Fourier, sabiendo que del espectrograma obtendremos las características que son relevantes ante situaciones de distorsión, estos valores obtenidos para cada frame son importados al bloque encargado del modelado de fingerprints.

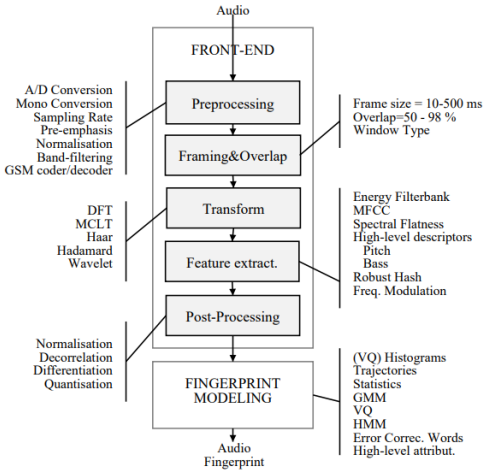


Fig. 2. Fingerprint Extraction Framework: Front-end (top) and Fingerprint modeling (bottom).

Figura 3.6 Fingerprint Extraction Framework.

Pre Procesado

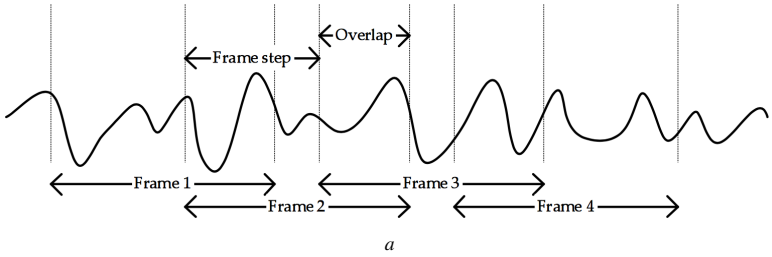
La señal de audio de entrada puede llegarnos en una variedad de condiciones y formatos muy diferentes, por ello debemos pre procesarla con el fin de trabajar con ella de la forma más simple posible sin perder

información importante. El primer paso es la conversión de la señal analógica a digital si el fragmento de audio ha sido capturado a partir del micro de nuestro dispositivo. Normalmente las canciones con las que trabajamos en la creación de la base de datos estarán en formato estéreo, para simplificar el tratamiento y las operaciones convertimos siempre a mono, ya que la pequeña pérdida de información es rentable respecto al coste computacional. La tasa de muestreo cuando procesamos canciones en alta calidad será de 44Khz, no necesitamos en general tanta resolución para nuestro algoritmo, así que muestreamos a 8Khz.

En el capítulo 4.3, vemos múltiples ensayos en los cuales la señal de entrada ha sido procesada de forma que simulamos un canal real de transmisión con el fin de hacer un estudio lo más fiel posible a la realidad. Por ejemplo añadimos ruido blanco +-3dB para simular un decodificador GSM.

Framing and Overlap

En el tratamiento de señales digitales puede demostrarse el carácter estacional de la señal en intervalos pequeños de tiempo, por ello trabajamos con intervalos de tiempos conocidos como frames. Las ventanas son funciones matemáticas usadas en el análisis y procesamiento de señales para evitar las discontinuidades al principio y al final de los bloques analizados, se trata de un tratamiento previo pensando en la posterior transformación, en nuestro caso la Transformada de Fourier Discreta Rápida(FFT).



a

Figura 3.7 Framing and Overlap.

[a] https://i.stack.imgur.com/d7Rdx.png

Para entender por qué aparecen estas discontinuidades entre bloques necesitamos introducir el concepto de fuga espectral. Este fenómeno aparece cuando el número de periodos no es entero, ya que los intervalos de la FFT basado en el conjunto finito de datos entiende que el dominio temporal y frecuencial presentan topologías circulares, es decir que ambos extremos de la onda están conectados, si el número de periodos es entero no tendremos problema. Sin embargo, cuando no es un número entero la topología circular no coincide y la señal presenta transiciones bruscas que se convierten en discontinuidades.[9] La ventana Hamming es una de las más utilizadas debido a su buena corrección de la fuga espectral y aporta una alta resolución frecuencial.

Transform

La elección de las características que nos interesan para el posterior procesado es muy importante cuando buscamos cuales son más resistentes al ruido, por otra parte necesitamos eliminar la redundancia, ya que de las señales obtenemos una gran cantidad de información pero no toda nos es útil. Desde el punto de vista del almacenamiento de información así como de la obtención de características incorreladas existen transformadas muy efectivas como son Karhunen-Loève(KL) o la Descomposición en valores singulares(SVD) sin embargo, estas transformadas suponen un coste computacional muy alto[21]. Una de las más utilizadas es la Transformada de Fourier Rápida(FFT) siendo esta un algoritmo eficiente que permite calcular la Transformada de Fourier Discreta(DFT) .Otras técnicas muy utilizadas incluyen la transformada del coseno (DCT) con una gran capacidad de compresión y concentración de la energía en pocos coeficientes [22], así como la transformada de Haar o la transformada de Walsh-Hadamard.[23]

Feature extract

La extracción de características es la fase donde mayor variedad podemos observar. Pero antes analizaremos los rasgos comunes, la señal de audio es dividida en frames y se extraen las características de cada uno de ellos. De este modo buscamos trabajar con características que se asemejan en su percepción a como lo hace el Sistema Auditivo Humano (HAS), como son la división en banda logarítmicas que aproxima la percepción del odio. Una simplificación que aplicamos cuando trabajamos con el espectro de frecuencia es la eliminación de la componente en fase, ya que el HAS es relativamente insensible a esta. Además de dividir el espectro en

bandas logarítmicas, podemos aplicar el banco de filtros correspondiente a la escala MEL, estos coeficientes aproximan la representación del habla basados en la percepción auditiva humana.[24]. El estudio realizado por Kimura et al [25] analiza el uso de la energía de cada banda para obtener un "hash string" de un espectro dividido en 33 bandas haciendo referencia a la percepción del HAS en la escala logarítmica en el rango entre 20Hz y 20kHz.

Una forma diferente de obtener características relevantes son los valores derivados, estos son obtenidos a partir de la media o varianza siendo representados de una forma compacta utilizando algoritmos de clasificación como los Hidden Markov Models [26]

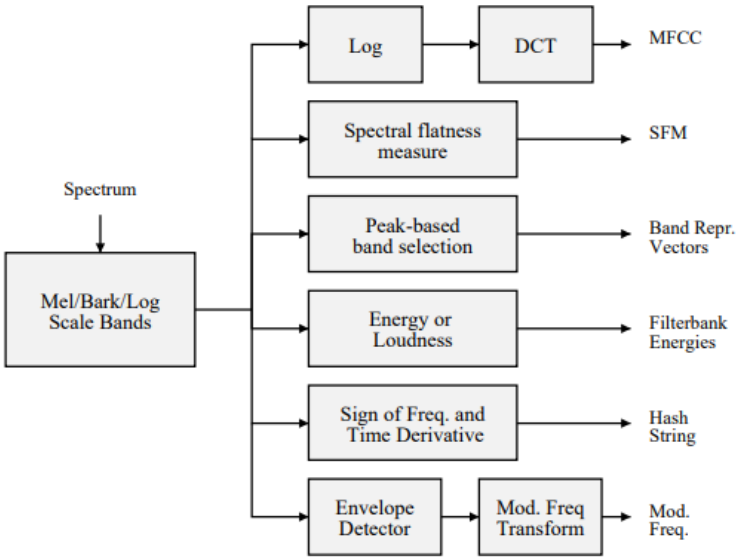


Fig. 3. Feature Extraction Examples

Figura 3.8 Feature Extraction.

Post procesado

Cuando hablábamos anteriormente del cálculo de la transformada de Fourier, Coseno o los coeficientes Cepstrales en las Frecuencias de Mel(MFCC), hablamos de valores absolutos sin embargo para la optimización del algoritmo resulta muy productivo implementar el cálculo de valores derivados. El caso de estudio del algoritmo Shazam[27] es un buen ejemplo, ya que a partir del espectro calculamos un landmark para cada pareja de máximos asociados y las componentes de estas incluyen la diferencia en frecuencia y en tiempo, aparte del tiempo y frecuencia inicial donde se encuentra el máximo de esa zona local del espectro.

4 Shazam Framework

"En algún lugar, algo increíble está esperando ser conocido".

Carl Sagan

A continuación vamos a detallar cómo funciona el algoritmo desarrollado por Shazam[27] con Avery Li-Chun Wang a la cabeza, así como la comparación con el desarrollo de Phillips[29] y otro Framework amateur desarrollado en la universidad de Seúl[22].

4.1 Descripción Algoritmo Shazam

El almacenamiento tendrá lugar en una base de datos, donde los índices serán los fingerprints calculados, pudiendo de esta forma ir asociados a uno o varios valores según las coincidencias que hayan aparecido en el cálculo de cada canción incluida en la base de datos. Una vez asociados, encontramos información sobre la canción a partir de la cual fue calculada y el instante dónde se encuentra el pico de energía en el espectro asociado.

Las canciones incluidas en la base de datos, así como los fragmentos de audio, reciben el mismo tratamiento previo y son pasados por la misma función hash. Para todas las pistas se realiza una conversión a mono, así como un remuestreo a 8Khz independientemente del formato de cada una. Esto se realiza para minimizar la complejidad del tratamiento y las operaciones posteriores.

4.1.1 Obtención landmarks

Los conceptos teóricos estudiados anteriormente sobre la Transformada de Fourier y su aplicación en su forma discreta con la particularidad vista (DFT) para la obtención del espectrograma, han servido como cimientos para el proceso de obtención de los landmarks y, a partir de estos, los fingerprints.

El espectrograma es una representación tridimensional del espectro de frecuencia de una señal, según los niveles de intensidad, podemos ver la amplitud de los picos para cada coordenada tiempo-frecuencia. [Figura 15]

La FFT basada en la descomposición frecuencial de la señal original presenta discontinuidades, ya que el dominio del tiempo y frecuencia muestra topologías circulares, esto quiere decir que ambos extremos de la forma de onda se tratan como si estuviesen conectados entre sí. Cuando la señal es periódica y un número entero de periodos ocupa el intervalo de tiempo de adquisición, la suposición de topología circular es correcta y no se presentan discontinuidades. No obstante, cuando el número de periodos no es entero aparecen las discontinuidades. Con el fin de eliminar o reducir al máximo estas discontinuidades, utilizamos las ventanas en el intervalo de tiempo de adquisición, de manera que obtenemos una forma de onda continua y sin transiciones bruscas.[9]

En nuestro caso, utilizamos una ventana Hamming(Ver capítulo de bases teóricas) con un overlap del 50 % . Esta destaca por su mayor resolución frecuencial en el primer lóbulo y cancelando los lóbulos laterales siendo muy efectiva para nuestro objetivo. Una vez obtenido el espectrograma, le aplicamos un preprocesado para disminuir su tamaño y eliminar información redundante. Se suprimen los picos pequeños y sustraemos del espectrograma su media, por último aplicamos un filtro paso alto con el fin de atenuar las pequeñas frecuencias. El último paso del preprocesado previo al emparejamiento, consiste en la aplicación del operador

morfológico del dilatado, extrayendo como resultados la constelación con los máximos de cada región estudiada determinada por la máscara.[Figura 16]. Según el tamaño de la máscara elegida variarán el número de máximos y los emparejamientos. Las dimensiones de la máscara es uno de los parámetros a estudiar en el capítulo de Conclusiones.

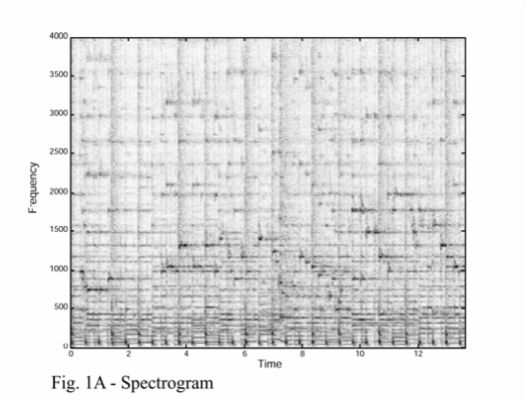


Figura 4.1 Espectrograma.

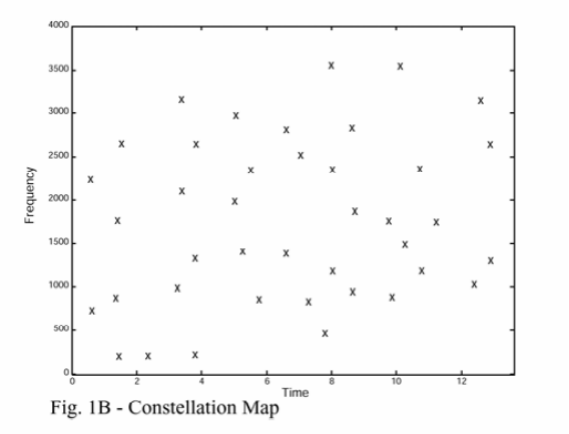


Figura 4.2 Constelacion.

Tras aplicar la máscara del dilatado, conservamos el máximo absoluto de cada región. Estos puntos son de especial interés, ya que en situaciones de distorsión será más probable que estos prevalezcan en el espectro. Con el fin de aumentar la robustez del algoritmo, las landmarks se obtendrán a partir de emparejar los máximos cercanos, de esta forma, ya no solo trabajamos con los valores absolutos del máximo del espectro en cuestión sino que lo hacemos con un valor derivado, tanto para la frecuencia como el tiempo. Individualmente, conocemos al máximo que estamos estudiando en ese instante como anchor point [27], mientras que la target zone delimitará los máximos que pueden asociarse por proximidad. En el estudio de conclusiones vemos cómo varía la velocidad y fiabilidad del algoritmo según el número de parejas que pongamos como límite. [Figura 17].

Cada una de estas parejas de máximos constituye un landmark, que está compuesto a su vez por la coordenada temporal y frecuencial del anchor point y los valores derivados entre los dos puntos de frecuencia y tiempo. Por cada uno de estos landmarks se calcula un fingerprint a partir de la función hash[20]. En el procedimiento de almacenamiento de fingerprints en la base de datos, el fingerprint o valor hash, funciona como índice de una tabla de direccionamiento indirecto. Hay asociado un valor o varios para cada hash, estos hacen referencia a distintas canciones, ya que en una base de datos amplia es normal que se repitan. Cada valor contiene la información necesaria para determinar a qué canción pertenece y el frame de su origen.

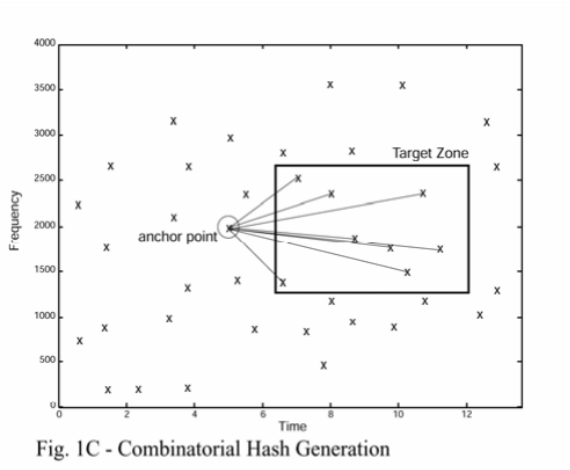


Figura 4.3 Combinatorial Hash Generation .

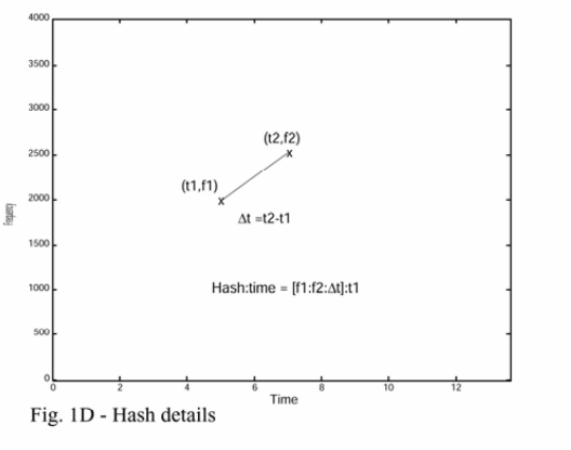


Figura 4.4 Hash details.

4.1.2 Algoritmo de Búsqueda.

El uso de estos hash combinados nos diferencia de otros métodos basados en sub-fingerprints, no solo en la generación de los hash, sino también en la forma de búsqueda.

Mientras que en métodos anteriores se obtenía una sub fingerprint para cada banda en cada frame de una canción, siendo generalmente un bloque fingerprint la secuencia de 256 sub fingerprints consecutivos. Cada sub fingerprint supone una entrada en la tabla de hash donde funcionan como índices. En cambio, al trabajar con valores absolutos y teniendo en cuenta que la función hash es muy sensible a nivel de bit, un fingerprint puede ser totalmente diferente con solo un poco de ruido. Es por ello que el número de fingerprints coincidentes era muy escaso. El algoritmo de Philips parte de la suposición que al menos habrá una coincidencia a pesar de las distorsiones[29]. Los pocos puntos coincidentes se tomarán como puntos candidatos de referencia para comprobar si la secuencia coincide con la canción original.

En el algoritmo de Shazam tenemos un fingerprint o valor hash para cada pareja de máximos obtenidos y está basada igualmente en el direccionamiento indirecto hacia un frame en concreto de una canción.

No obstante, en el algoritmo desarrollado por el equipo de Wang, la búsqueda se centra en la obtención de una nube de puntos donde todos tengan el mismo “offset” o desplazamiento, como hemos visto anteriormente, a cada fingerprint va asociado un valor o más, denominado token, a partir del cual podemos obtener el ID de la canción, así como el instante original de la canción donde estaba el máximo estudiado.

Para nosotros, la identificación se da como correcta cuando podemos asegurar que entre un fragmento corto de audio con ruido añadido que recibe el mismo tratamiento que las canciones insertadas en la base de datos, hay una serie de **hash coincidentes**, los cuáles en su mayoría apuntarán a una canción concreta y comprobamos que para una serie de hash coincidentes el **offset relativo** tiene el mismo valor [fig 20].Es

decir, hemos encontrado coincidencias de una series de parejas de máximos los cuales están distanciados en igual medida respecto a la canción original,ergo son consecutivos y forman una nube de puntos en diagonal que indica la existencia de una secuencia de puntos coincidentes.[fig. 19]. En la [fig.21]., vemos una serie de puntos incorrelados, que resultan como una detección no concluyente.

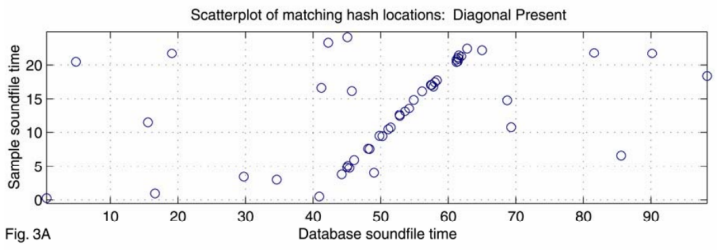


Figura 4.5 Scatterplot of matching hash locations: Diagonal Present.

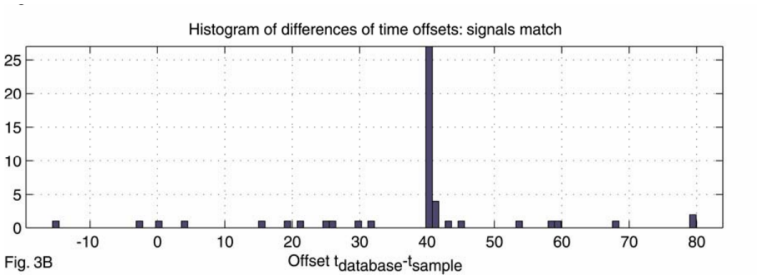


Figura 4.6 Histogram of differences of time offsets: signals match.

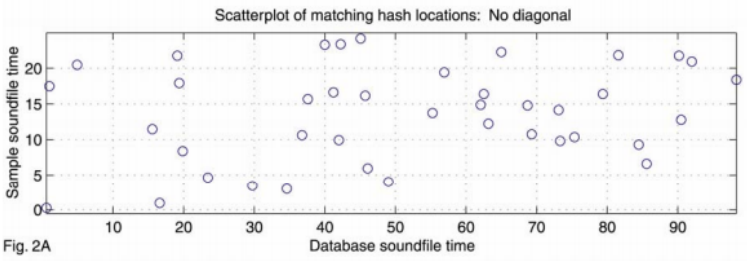


Figura 4.7 Scatterplot of matching hash locations: No diagonal.

Agrupamos los puntos según los offset, un grupo grande de puntos con el mismo offset indica que aparecen de forma consecutiva tanto en el fragmento de audio como en ese fragmento de canción original, dando lugar a un **match**, y teniendo en cuenta el ritmo y el estribillo constante de una canción es posible encontrar coincidencias dentro de una misma canción, valoramos según la densidad de puntos para determinar el instante de la ocurrencia.

4.2 Implementación en código Matlab del algoritmo Shazam

En este capítulo detallaremos la implementación realizada del algoritmo de Shazam en Matlab, con el fin de comprobar su robustez en distintas situaciones de distorsión. Comentaremos en profundidad la parte más importante de los ficheros desarrollados, así como la relación que se establece entre ellos. Debido a la complejidad inicial que tiene familiarizarse con algunos de los conceptos relacionados con los audios fingerprints descritos anteriormente en este trabajo, expondremos un ejemplo simple del funcionamiento unido con algunas referencias a los procesos que hemos analizado a lo largo de este trabajo.

Para facilitar la comprensión del código vamos a dividirlo en dos fases:

4.2.1 Modelado de fingerprints

La primera fase comprende todo el proceso de creación de la base de datos y la obtención de los fingerprints a partir de los landmarks. Los landmarks se obtienen a partir del emparejamiento de los picos de energía cercanos de la señal. Estos se calculan a partir de la información que obtenemos del espectrograma, en principio, este cálculo parece simple, sin embargo, necesitamos realizar un filtrado preciso para eliminar toda la información redundante que obtenemos del espectrograma. Siendo este un punto de especial interés que detallaremos en la función *getLandmarks.m*, ya que repercutirá directamente en la eficacia y eficiencia del algoritmo como veremos en la fase de testeo. Un landmark es una matriz que posee información sobre las coordenadas temporales y frecuenciales de los máximos emparejados. Como hemos estudiado anteriormente, un fingerprint es un elemento identificativo de pequeño tamaño basado en el contenido de la señal, a partir del cual se puede realizar una identificación rápida de una canción. Un fingerprint es calculado a partir de una funciópn hash cuyo elemento de entrada es el landmark, obtenemos una representación compacta y de longitud fija del landmark introducido a partir de sus parámetros. En una base de datos grande, se darán casos donde podemos encontrar landmarks de diferentes canciones con distintos valores que darán lugar a un mismo fingerprint, por simple probabilidad. Por lo tanto, necesitamos que cada fingerprint apunte a todas y cada una de las canciones a partir del que fue calculado, así como el frame exacto. Estos valores asociados a cada fingerprint son los tokens. En la función *getTokens.m* veremos cómo rellenamos la base de datos y detallamos cómo funciona esta tabla de direccionamiento indirecto con la estructura donde los fingerprints o hash values serán los índices y los tokens son los valores asociados.

4.2.2 Proceso identificación.

Para realizar la identificación de un fragmento de audio aplicaremos el mismo proceso que a las canciones de la base de datos. Según la naturaleza de las pruebas que queramos realizar en la función *identificadorpieces.m* podemos realizar una identificación a partir de un archivo que contenga un fragmento de audio que le pasamos como parámetro, una grabación de unos que se realiza en directo desde el mismo dispositivo o un test completo como los detallados en la fase de testeo en el cual generamos un gran número de fragmentos aleatorios a los que podremos añadir ruido a distintos niveles. La función *getmatches.m* nos permite obtener las coincidencias encontradas entre los fingerprints calculados para nuestro fragmento de audio con la base de datos. De la coincidencia de dos fingerprints nace un **match**, a partir del instante temporal asociado a cada fingerprint obtenemos un **offset** o desplazamiento entre ellas. Cada uno de los matches obtenidos tendrá asociado además del fingerprint o hash value, la id de la canción, así como el offset. Para determinar de qué canción se trata, evaluaremos los **matches** con el mismo **offset**, quedándonos con la canción que predominante que forme una diagonal de puntos.

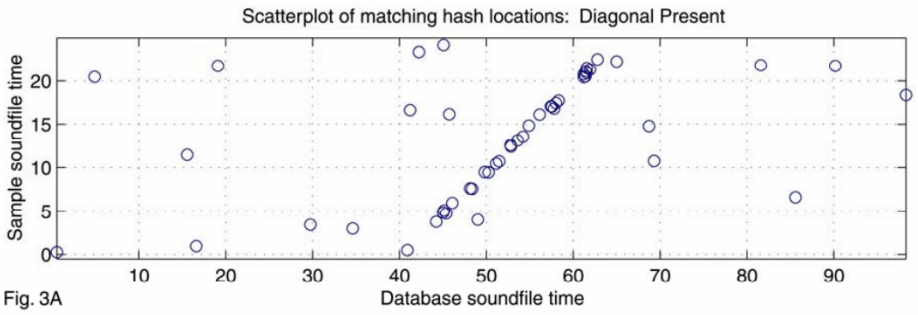


Figura 4.8 Scatterplot of matching hash locations: Diagonal Present.

Tras identificar una canción como coincidente podemos obtener a partir del offset el instante concreto del fragmento en la canción, para así realizar un Bit Error Test entre ambas secuencias. Un ratio menor a 0.35 es suficiente para aceptar una identificación.

Tras realizar una visión general del algoritmo, entramos en detalle para el desarrollo de cada función. Algunas funciones muy básicas se desarrollarán en el fichero donde se realice la llamada.

4.2.3 Código Matlab

Las canciones que deseemos añadir a la base de datos deben estar en el directorio 'canciones' y las indexamos en un contenedor de mapas que funcionará como nuestra base de datos de canciones cuyo formato será [key:value]. Los contenedores de mapas son estructuras de datos que permiten asociar valores a una key, de forma indexada, las keys son únicas. Los contenedores map aportan una gran flexibilidad, ya que permiten que las claves sean números reales o un vector de caracteres.

Creación Base de Datos

Las canciones que deseemos añadir a la base de datos deben estar en el directorio 'canciones' y las indexamos en un contenedor de mapas que funcionará como nuestra base de datos de canciones cuyo formato será key:value. Los contenedores de mapas son estructuras de datos que permiten asociar valores a una key, de forma indexada, las keys son únicas. Los contenedores map aportan una gran flexibilidad, ya que permiten que las claves sean números reales o un vector de caracteres.

$$M = containers.Map('KeyType',kType,'ValueType',vType) \tag{4.1}$$

La inclusión de las figuras se realiza mediante un conjunto de instrucciones que se muestran en el Código 4.1.

Código 4.1 Creación Base de datos y preparación lectura canciones.

```
%Lectura del fichero donde se encuentran las canciones que formarán la base
%de datos con la que trabajaremos.
listing = dir('canciones');
s = cell(length(listing),1);
contador=0;

M = containers.Map('KeyType','uint32','ValueType','any');
for i=1:length(listing)
    if contains(listing(i).name,'.mp3')
        contador=contador+1;
        s{contador,1}=listing(i).name;
    end
end
s=s(~cellfun('isempty',s));
```

Las keys que funcionan como índices serán los hash calculados a partir de los parámetros que componen las landmarks, por lo tanto el keyType será uint32.
En esta primera parte de la función hemos creado un contenedor vacío, y realizado una lectura de los ficheros que se encuentran en el directorio de canciones. Hacemos las comprobaciones pertinentes para asegurarnos la única lectura de ficheros de audio .mp3 , ya que en en Windows es común encontrar ficheros ocultos cuando realizamos una lectura en forma de barrido para un directorio.

Código 4.2 lectura de canciones.

```
for i=1:length(s)
[y,fs]=audioread(strcat('canciones\',s{i,1}));
yRs=audio_converter(y,fs);
landmarks = getLandmarks(yRs);
x=getTokens(landmarks,i,M);
tokens=[tokens;x];
fprintf("Numero de tokens encontrados para la cancion:\%d :\%d\n", i,length(x))
    ;
end
```

Con el objetivo de simplificar y unificar el formato de las canciones con las que trabajaremos, procedemos en *audioconverter.m* a comprobar el formato de las pistas y a convertirlas a mono si están en estéreo. Por último, aplicamos un muestreo a 8Khz.

Código 4.3 audioconverter.

```
function[yRs]=audio_converter(y,fs)
[m,n]=size(y);
if n==2
yMono=0.5\*(y(:,1)+y(:,2));
else
yMono=y;
end
yRs=resample(yMono,8000,fs);
end
```

GetLandmarks.m

Argumentos de entrada:
data_song: pista de audio convertida a mono, resampleada a 8Khz.
Argumentos de salida:
Landmarks: matriz con una fila para cada conjunto de dos picos emparejados.
Landmarks=[tiempo_inicial, frencuencia_inicial, delta_frecuencia,delta_tiempo].

Realizamos el cálculo de las landmarks de forma individual para todas las canciones que componen la base de datos, así como del fragmento target a identificar. Como hemos estudiado anteriormente, los Landmarks son obtenidos a partir del emparejamiento de máximos en el espectro cercanos tanto en frecuencia como en tiempo.

Código 4.4 Parámetros espectrograma.

```
fs = 8000;
tWindow = 64e-3;
NWindow = fs*tWindow;
window = hamming(NWindow);

nfft=512;
noverlap = NWindow/2;

[S, F, T,P] = spectrogram(data_song,window,noverlap,nfft,fs);
%S.Transformada Fourier a corto plazo.
%F.Coordenada frecuencial.
%T.Coordenada temporal.
%P.Respuesta en magnitud.
```

El primer paso es el cálculo del espectro de energía de la señal, utilizando la función spectrogram implementada en Matlab. El espectrograma se calcula a partir de la Transformada Rápida De Fourier Discreta, en el estudio será indispensable calcular un espectrograma óptimo. Para el cálculo del espectro trabajamos en frames, debido a la fuga espectral es común que aparezcan discontinuidades en los extremos. Esta discontinuidad es producida por la propiedad cíclica de la transformada implementada, la transformada entiende que el principio y el final del frame están conectados por lo tanto cuando aparecen un número impar de periodos suceden discontinuidades. A la hora de eliminar las discontinuidades, aplicamos una ventana temporal a cada frame, en nuestro caso elegimos una ventana Hamming debido a su versatilidad y buena respuesta tanto en frecuencia como en tiempo.

El siguiente paso para el cálculo de los Landmarks es emparejar los máximos encontrados en el espectrograma. Sin embargo, el cálculo del espectro aporta una cantidad muy grande información. Siendo nuestro

objetivo extraer características robustas del espectro, podemos eliminar información sobre los máximos relativos que quedarán fácilmente enmascarados en caso de ruido. Por ello realizamos un preprocesado del espectrograma en el cual eliminamos los pequeños picos para aliviar así la carga computacional del algoritmo. En este caso procedemos a suprimir los valores inferiores a un factor de 10 $\hat{6}$, este es un tratamiento muy común a la hora de trabajar con el espectrograma, ya que los puntos que nos aportan información relevante son los máximos y queremos reducir a toda costa el coste computacional. Por último, le restamos la media y aplicamos un filtro paso alto.

Código 4.5 Preprocesador del espectograma.

```
%% Preprocesador del espectograma, eliminando los picos pequeños y restando
%la media.
S=abs(S);
Smax=max(S(:));
S=10*log10(max(Smax/1e6,S));
S = S - mean(S(:));
% Filtro paso alto.
S = (filter([1 -1],[1 -0.98],S')');
```

Tras el preprocesado procedemos a la búsqueda de máximos. Cuando determinamos qué punto es el máximo de una región pequeña, utilizamos el operador morfológico dilatado. Utilizando la regla de la vecindad, asocia al pixel de salida, el valor más alto de toda la vecindad. La vecindad es un parámetro variable, en nuestro caso hemos trabajado con una máscara que equivale a un cuadrado de dimensiones 20,20.

Código 4.6 Operador dilatado.

```
%% Busqueda de maximos en el espectograma.
%A partir del operador morfologico dilatado(max) encontraremos los puntos
%que tras aplicar la máscara siguen manteniendo su valor, estos serán los
%maximos a almacenar.
mask= strel('rectangle',[20,20]);
SDilated = imdilate(S,mask);
[c_frecuencia, c_tiempo] = find(SDilated == S);
```

Nos quedamos con las coordenadas de los puntos que se mantengan antes y después del dilatado. Reduciendo considerablemente el número de máximos a estudiar.

Código 4.7 Emparejamiento de máximos.

```
%% Emparejamiento de maximos.
%Comprobaremos para cada uno de los máximos obtenidos, si hay algún otro
%máximo que cumpla con los requisitos de cercania en tiempo y frecuencia.
landmarks=zeros(length(maxes)*3,4);
c=0;
for i=1:length(c_frecuencia)-1
    fini=c_frecuencia(i);
    tini=c_tiempo(i);
    [Y1,X1]=find((abs(c_frecuencia - fini) < 32) & ((c_tiempo - tini) > 0) & ((
        c_tiempo - tini) < 64), 10);
    %Obtenemos las coordenadas de los puntos emparejados y los almacenamos
    for j = 1:length(Y1)
         Landmarks = [tini, fini, c_frecuencia(Y1(j)) - fini, c_tiempo(Y1(j)) -
              tini];
         c = c+1;
         L(c,:) = Landmarks;
    end
end
```

Para cada uno de estos máximos haremos una búsqueda dentro de su zona target. El número de emparejamientos es un parámetro muy importante en el estudio del algoritmo. Ya que incrementando el número de emparejamientos mejoramos la robustez del algoritmo pero a su vez, también aumenta el coste computacional. Podemos ver que con la configuración actual almacenamos 3 Landmarks por cada máximo, que es el resultado óptimo analizado en fase de testeo.

GetTokens.m

La función recibe como parámetros:
[L]: landmarks calculados previamente.
[id_song]: número de referencia que indica el orden de inserción de la canción en la base de datos. Si en este parámetro recibimos un 0 no los almacenamos en la base de datos, ya que son calculados a partir del fragmento que deseamos identificar.
[M]: map container utilizado como base de datos. El container.map[M] será utilizado como una LUT(Look up Table) cuya estructura es [key:value]

En esta función vamos a calcular los fingerprints que servirán como índices de la tabla, así como los valores asociados que aportan la información necesaria para determinar de qué canción y frame fue obtenido ese fingerprint. Al conjunto de fingerprint y su valor o valores asociados les denominamos tokens. Para evitar problemas a la hora de trabajar con datos de longitud variable, utilizamos una función hash que a partir de unos parámetros variables nos permite obtener un valor de longitud fija. Según el tamaño de la base de datos, necesitaremos configurar una función con más o menos bits, en nuestro caso es suficiente con 20 bits. Comentamos dos posibles funciones igualmente válidas.

La función utilizada es la siguiente:

$$hash = fini * 2^{12} + fdelta * 2^{6} + tdelta \tag{4.2}$$

Otra función común es:

$$hash = fini * 2^{7} + fdelta * 2^{7} + tdelta * 2^{6} \tag{4.3}$$

Código 4.8 Función hash.

```
for i = 1:length(L)
    tini = L(i,1);
    fini = L(i,2);
    fdelta = L(i,3);
    tdelta = L(i,4);
    hash = (fini)*2^12 + fdelta*2^6 + tdelta;
    tokens(i, :) = [id_song, tini, hash];

end
```

Aquí podemos como para los valores de cada Landmark obtendremos un fingerprint de forma sencilla. Los tokens almacenarán la información necesaria para determinar a partir de qué canción fue obtenida, así como el tiempo.

En este fragmento de código, observamos cómo se realiza la inserción en la base de datos siempre que el id_song sea distinto de 0, ya que no deseamos almacenar los tokens de los fragmentos que vamos a identificar.

Código 4.9 Obtención de fingeprints.

```
if id_song ~= 0
      %los hash serán los indices.
      index=tokens(:,3);
      ids_song = tokens(:,1);
      t_ini = tokens(:,2);

      for j=1:length(index)
         data=(ids_song(j))*TIMESIZE+t_ini(j);
          if M.isKey(index(j))
              %un mismo hash puede encontrarse a partir de dos canciones
              %diferentes, almacenaremos todos los encontrados
              %concatenandolos.
               M(index(j)) = [M(index(j)) uint32(data)];
          else
               M(index(j)) = uint32(data);

          end
      end
    end
```

Los tokens para ser almacenados necesitan pasar por una función hash para obtener una longitud fija y ser fácilmente recuperables, se calculan a partir del id_song y el tiempo inicial. Debido a la alta densidad de landmarks calculados, encontraremos casos donde para distintas canciones con distintos parámetros aparezcan dos fingerprints iguales. En ese caso tendremos un fingerprint que apunta a más de una canción, por lo tanto, concatenamos los valores calculados. Con lo visto anteriormente la base de datos estaría completa y podríamos dar por finalizada la primera etapa del algoritmo, procedemos a la segunda fase conocida como proceso de identificación. Buscaremos las coincidencias más probables en la base de datos realizando el mismo proceso de obtención de fingerprints para fragmentos de audio generados aleatoriamente a partir de las canciones contenidas en la base de datos o bien de grabaciones.

GetMatches.m

En esta función desarrollamos el código necesario para identificar los matches existentes entre un fragmento de audio y la base de datos previamente creada.

function[matches] = getMatches(tokens_song,M)
Parámetros de entrada:
tokens_song: tokens calculados previamente para un fragmento o grabación que deseamos identificar. M: base de datos que contiene los tokens de las canciones leídas previamente.
Parámetros de salida:

matches: Nos proporciona el número de coincidencias entre los tokens calculados para el fragmento y los encontrados en la base de datos.

La función recibe como parámetros los tokens del fragmento de audio que queremos analizar y la base de datos. Realizamos una búsqueda de los fingerprints calculados para un fragmento.

Realizamos el proceso inverso al realizado para obtener un valor de longitud fija que contenga el id de la canción y el tiempo inicial. Para cada uno de los matches devolvemos el id de la canción, así como el offset o desplazamiento temporal entre el fragmento y la canción original. Posteriormente, analizaremos el número de matches con el mismo offset que apunten a la misma canción, ya que estos muy probablemente darán lugar a un emparejamiento correcto.

Código 4.10 GetMatches.m.

```
hashes = tokens_song(:,3);
index=0;
TIMESIZE=2^17;
for i = 1:length(hashes)
    %EN la SIguiente condicion comprobamos que hashes de la cancion a
    %estudiar: hashes = tokens_song(:,3);
    %Existen en la base de datos.
    if(M.isKey(hashes(i)))
    %para obtener el valor asociado a la key(indice, en nuestro caso los
    %hashes: value=M(hashes(i))

    value=M(hashes(i));
    Lvalue=length(value);

        for j=1:Lvalue
        %sobre un mismo hash, puede tener mas de un valor asociado,
        %lo calculamos previamente y evaluamos.
        index=index+1;
        %Almacenaremos

        n_song=floor(value(j)/(TIMESIZE));
        %extraemos del value los datos como son el tini->deltaT.
        t=round(value(j)-n_song*TIMESIZE);
        dtime=t-tokens_song(i,2);
         matches(index, 1) = n_song;
         matches(index, 2) = abs(dtime);
         matches(index, 3) = hashes(i);

        end
    end
end
```

Identificador_Pieces.m

Este archivo funciona de forma independiente o modo de script, según los parámetros. Para realizar la identificación le podemos pasar una canción completa, un fragmento, o bien pedirle que realicemos una grabación. Analizaremos el caso más complejo en el que realizamos una batería de pruebas para evaluar la robustez del algoritmo, especificaremos el número de fragmentos que queremos realizar para cada canción así como su duración.

Para la obtención de fragmentos aleatorios trabajamos con la información obtenida por la función audioread:

Este archivo funciona de forma independiente o modo de script, según los parámetros. Para realizar la identificación le podemos pasar una canción completa, un fragmento, o bien pedirle que realicemos una grabación. Analizaremos el caso más complejo en el que realizamos una batería de pruebas para evaluar la robustez del algoritmo, especificaremos el número de fragmentos que queremos realizar para cada canción así como su duración.

Para la obtención de fragmentos aleatorios trabajamos con la información obtenida por la función audioread:

```
         Filename: 'C:\Users\SoupierDrake\Documents\TFG\code\demo3_2\canciones\Bohemian_Rhapsody.mp3'
CompressionMethod: 'MP3'
      NumChannels: 2
       SampleRate: 48000
     TotalSamples: 17254080
         Duration: 359.4600
            Title: 'Queen - Bohemian Rhapsody (Official Video Remastered)'
          Comment: []
           Artist: []
          BitRate: 320
```

Figura 4.9 Audioread info.

Tras pasar las canciones por la función audio_converter el Sample Rate es de 8 Khz, para generar los fragmentos para la fase de testeo nos aseguramos de que no empiecen en los últimos 20 segundos de la canción para evitar problemas.

Código 4.11 fragmentos aleatorios.

```
[y,fs]=audioread(strcat('canciones\',s{i,1}));
    yRs=audio_converter(y,fs);
    inicio_muestra= double(uint32(1 + (length(yRs)-20*(8000))*rand(
        numero_fragmentos,1)));
```

El inicio de la muestra se calcula a partir de una distribución normal aleatoria cuya probabilidad es la misma para todos los valores en el rango.

Código 4.12 Obtención fingerprints fragmentos aleatorios.

```
muestra=yRs([inicio_muestra(j):inicio_muestra(j)+(duracion_fragmento*8000)]);
          muestra=awgn(muestra,noise,'measured');
          landmarks = getLandmarks(muestra);
          tokens_song=getTokens(landmarks,0,M);
          name=strcat('fragmentos\autogen_',int2str(j),'_',s{i,1});
          matches=getMatches(tokens_song,M);
```

Generamos para cada uno de los fragmentos una muestra de longitud determinada por el parámetro duracion_fragmento, el ruido blanco aditivo para cada muestra se obtiene a partir de la función awgn, el nivel es determinado por el parámetro noise. Aplicamos el mismo tratamiento que en las canciones originales, primero calculamos las landmarks, en segundo lugar obtenemos los fingerprints y sus valores asociados en getTokens. Podemos apreciar que esta vez llamamos a la función con el parámetro id_song con valor 0 para no almacenar los fingerprints obtenidos.

Código 4.13 Evaluación coincidencias encontradas.

```
%Apartir del numero de coincidencias determinaremos que canción se
          %trata e imprimiremos el porcentaje de acierto.
          total_matches=length(matches);
          [posible_songs,position]=unique(matches(:,1),'first');
          [utime, xt] = unique(matches(:,2), 'first');
          position=[position; total_matches];
          c1=diff(position);
          [pos_descend,ids_songs]=sort(c1,'descend');
          song_match=[pos_descend,ids_songs];
          ntime = size(matches,1);
          utimecounts = diff([xt', ntime+1]);
          [sortedUTiC, UTiCindex] = sort(utimecounts,'descend');
          utime(UTiCindex(1));
```

Para la identificación ordenamos según el número de offsets con el mismo valor, el siguiente ejemplo pertenece a la identificación de un fragmento aleatorio de la canción One de Metallica

	1	2	3
1	2	3755	35
2	4	6300	2
3	3	595	1
4	1	2145	1

Figura 4.10 Ejemplo matches found.

En este caso hemos obtenido para el id_song número dos, un total de 35 matches en el time offset 3755. Este valor multiplicado por la resolución temporal de la ventana que es 0.032 segundos, nos daría como resultado el segundo 120 de la canción original.

Código 4.14 Resultado.

```
if length(pos_descend) > 2
          if pos_descend(1) > 1.25*pos_descend(2)
              fprintf("Resultado:Fragmento analizado: \%s .Canción encontrada:
                   \%s. \n", name,erase(s{ids_songs(1)},".mp3"));
              %%%% Contador de verdaderos y falsos positivos:
              if contains(s{i,1},s{ids_songs(1)})
                  positivos=positivos+1;
              else
                  negativos=negativos+1;
              end

          else
              fprintf("Resultado:Fragmento analizado: \%s .No hay resultado
                  concluyente.\n", name);
               negativos=negativos+1;
          end
         else
             fprintf("Resultado:Fragmento analizado: \%s .No hay resultado
                 concluyente.\n", name);
             negativos=negativos+1;
         end
```

Por último contabilizamos si el fragmento analizado pertenece a la canción correcta y obtenemos resultado de la robustez del algoritmo según los parámetros variables.

4.2.4 Fase testeo

Durante el desarrollo del trabajo hemos ido mencionando en numerosas ocasiones la necesidad de encontrar en el algoritmo un punto de equilibrio entre el coste computacional asociado al tiempo de identificación y la robustez. Es por ello que hemos valorado la forma en la que afecta la modificación de los parámetros, para así poder afianzar las conclusiones alcanzadas, basándonos en los datos obtenidos de los múltiples experimentos realizados en distintas condiciones de distorsión.
En el proceso de testeo han sido utilizados los siguientes parámetros:

1. El **tamaño de la máscara**, el cual determina los picos de energía del espectro que se conservan alrededor del máximo absoluto de una región, determinada por el tamaño de la máscara.
2. El **Número de landmarks**, determina, el número máximo de emparejamientos para cada *anchor point* dentro de su target zone. Su aumento generalmente supondrá una mayor robustez, pero a su vez hace empeorar la eficiencia del algoritmo.

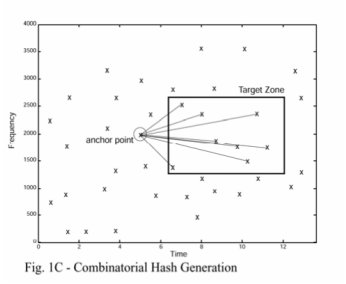


Figura 4.11 Target Zone.

3. La **duración fragmento** evalúa la robustez del algoritmo realizando pruebas con fragmentos de 5, 10 y 15 segundos. En general, con una grabación de 5 segundos, será suficiente para obtener una identificación correcta.
4. La **distorsión de la señal**, Además de las pérdidas asociadas a la compresión, añadimos ruidos blanco a distintos niveles, desde -15 dB hasta 15 dB en intervalos de 3dB.

Realizaremos una serie de tests cubriendo un amplio espectro de valores con el fin de llegar a una configuración óptima. En cada test vamos a poder ver el valor de cada parámetro, mientras que en las gráficas tendremos tres líneas de puntos unidos que representan los resultados de la probabilidad de acierto, para fragmentos de 5,10 y 15 segundos respectivamente.

En la tabla inferior podemos ver desglosados los resultados obtenidos, cada uno de los test se compone de 4620[1] pruebas de identificación individuales.

La duración de la identificación es uno de los factores más relevantes para el algoritmo, en cada experimento comparamos los tiempos obtenidos para cada configuración: la creación de la base de datos, el tiempo total de cada test y el tiempo individual de una identificación. A la hora de analizar el tamaño de la base de datos, hemos hecho hincapié en las dimensiones de la máscara y el número limitador de landmarks. Asimismo, vamos a utilizarlos junto los ratios de acierto para poder así determinar la configuración óptima. Finalmente, se procederá a un estudio análogo de los resultados no óptimos para distintas configuraciones vistas a partir de los valores que hemos obtenidos, teniendo en cuenta los siguientes valores:

- **t_BBDD**. muestra el tiempo necesario para la creación de la base de datos.
- **t_identif_T**. tiempo de identificación total para cada test completo.
- **t_identif_I**. tiempo de identificación individual para cada fragmento individual.
- **BBDD_size**. Tamaño de la base de datos en MB.

[1] 4620 pruebas = $14 canciones \cdot 10 fragmentos \cdot 11 nivelesderuido[-15:3:15] \cdot 3$ [5,10,15 seg por fragmento]

Test 1

- Numero Fragmentos aleatorios por canción = 10.
- Máximos emparejados = 10.
- Target Zone Dimensiones (64×32).
- mask =20x20.

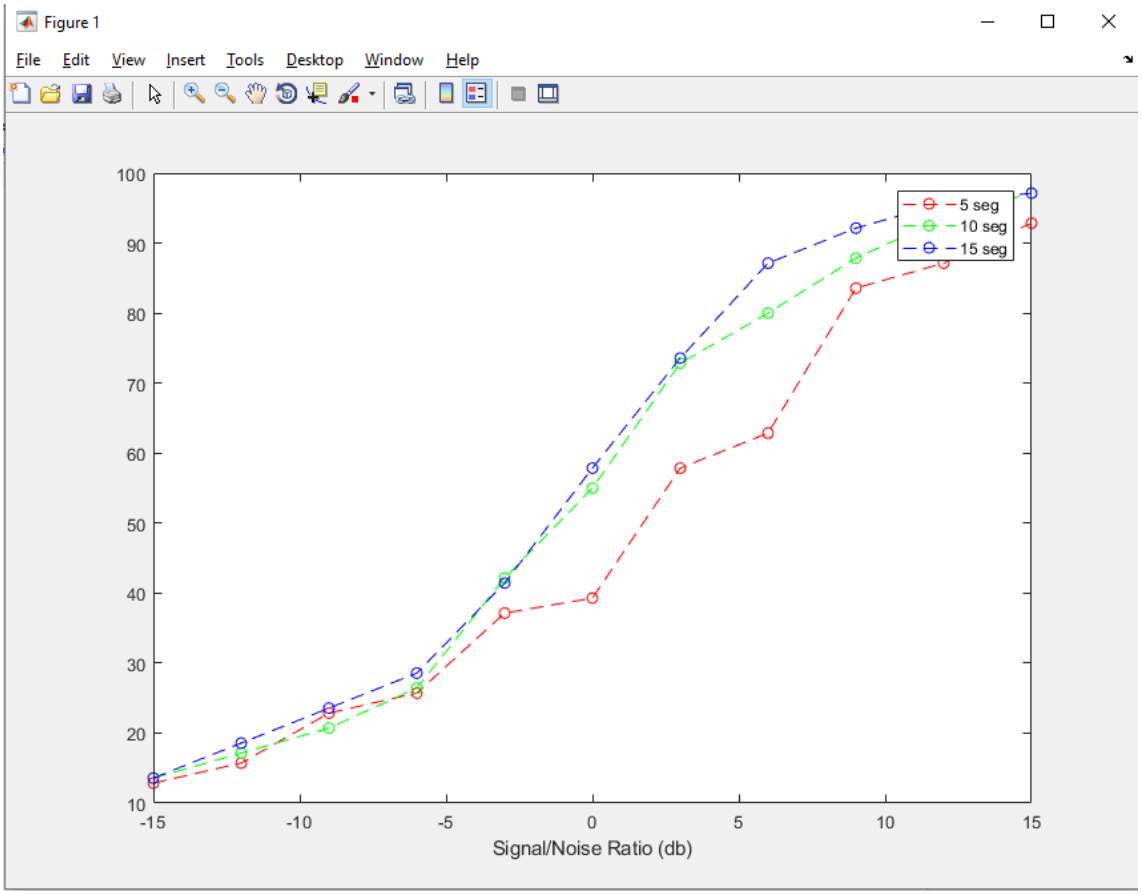


Figura 4.12 Test1.

Tabla 4.1 Test 1.

ratio acierto.(%)	-15 db	-12db	-9 db	-6 db	-3 db	0 dB	3 dB	6 dB	9 db	12 dB	15 dB
5 seg	12.85	15.71	22.85	25.71	37.14	39.28	57.85	62.85	83.57	87.14	92.85
10 seg	13.57	17.14	20.71	26.42	42.1	55	72.8	80	87.85	93.57	97.14
15 seg	13.57	18.54	23.57	28.57	41.42	57.85	73.57	81.14	92.14	95.71	97.14

- t_BBDD = 50.93 seg.
- t_identif_T = 1012.43 seg.
- t_identif_I = 0.22 segundos
- BBDD_size = $355778 \cdot 32$ bits = 11384896 bits =1.42 MB

Test 2

- Numero Fragmentos aleatorios por canción = 10.
- Máximos emparejados = 5.
- Target Zone Dimensiones (64×32).
- mask =20x20.

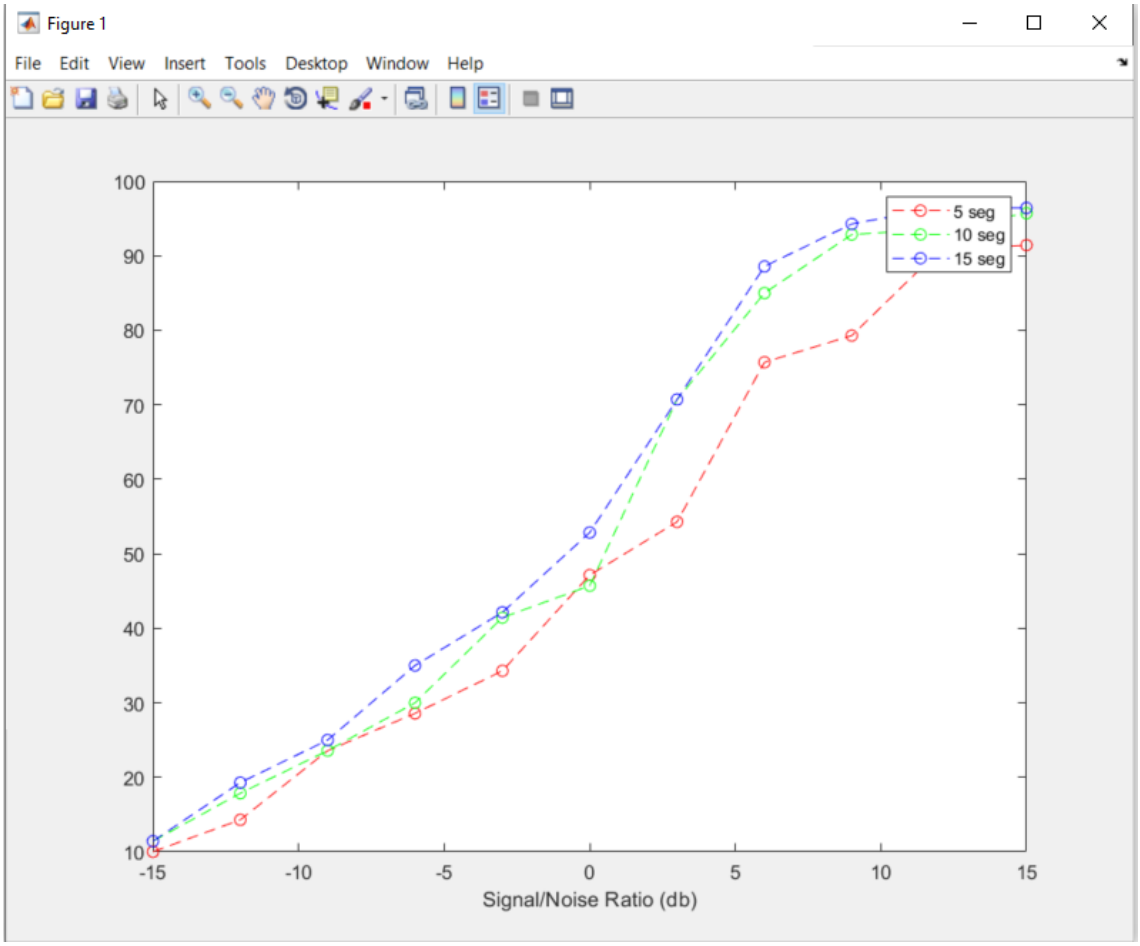


Figura 4.13 Test2.

Tabla 4.2 Test 2.

ratio acierto.(%)	-15 db	-12db	-9 db	-6 db	-3 db	0 dB	3 dB	6 dB	9 db	12 dB	15 dB
5 seg	10	14,28	23,57	28,57	34,28	47,14	54,28	75,71	79,28	90,71	91,42
10 seg	11,42	17,85	23,57	30	41,42	45,71	70,71	85	92,85	93,57	95,71
15 seg	11,42	19,28	25	35	42,14	52,85	70,71	88,57	94,28	96,42	96,42

- t_BBDD = 41.90 seg.
- t_identif_T = 888.68 seg.
- t_identif_I = 0.19 seg
- BBDD_size = $238212 \cdot 32$ bits = 7622784 bits = 0.95 MB

Test 3

- Numero Fragmentos aleatorios por canción = 10.
- Máximos emparejados = 3.
- Target Zone Dimensiones (64×32).
- mask =20x20.

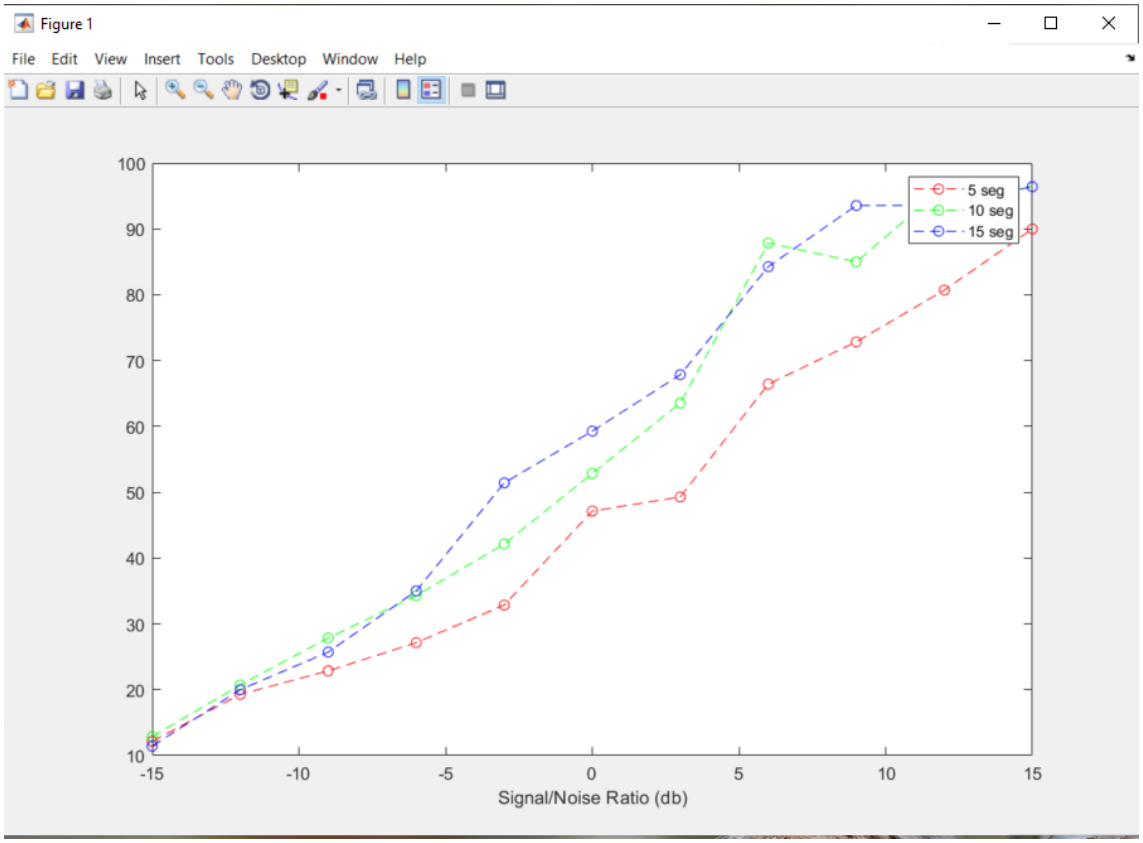


Figura 4.14 Test3.

Tabla 4.3 Test 3.

ratio acierto.(%)	-15 db	-12db	-9 db	-6 db	-3 db	0 dB	3 dB	6 dB	9 db	12 dB	15 dB
5 seg	12,14	19,28	22,85	27,14	32,85	47,14	49,28	66,42	72,85	80,71	90
10 seg	12,85	20,71	27,85	34,28	42,14	52,85	63,57	87,85	85	96,42	96,42
15 seg	11,42	20	25,71	35	51,42	59,28	67,85	84,28	93,57	93,57	96,42

- t_BBDD = 34.37 seg.
- t_identif_T = 798.39 seg.
- t_identif_I = 0.17 seg.
- BBDD_size = $148838 \cdot 32$ =4762816bits=0.59MB

Test 4

- Numero Fragmentos aleatorios por canción = 10.
- Máximos emparejados = 3.
- Target Zone Dimensiones (64×32).
- mask =10x10.

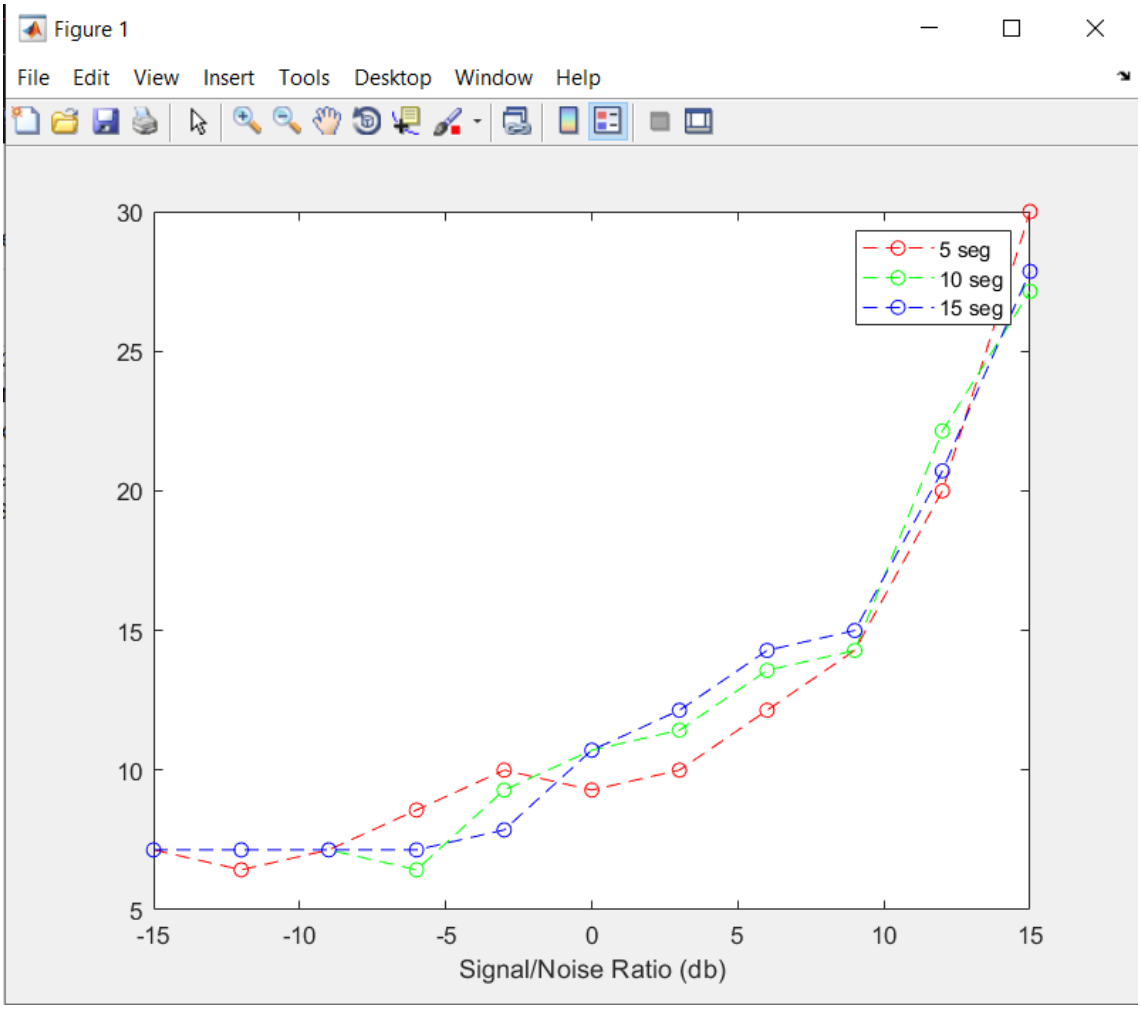


Figura 4.15 Test4.

Tabla 4.4 Test 4.

ratio acierto.(%)	-15 db	-12db	-9 db	-6 db	-3 db	0 dB	3 dB	6 dB	9 db	12 dB	15 dB
5 seg	7,14	6,42	7,14	8,57	10	47,14	9,28	10	12,14	14,28	30
10 seg	7,14	7,14	7,14	6,42	9,28	10,71	11,42	13,57	14,28	22,14	27,14
15 seg	7,14	7,14	7,14	7,14	7,85	10,71	12,14	14,28	15	20,71	27,85

- t_BBDD = 65.046seg.
- t_identif_T =1277,32 seg.
- t_identif_I = 0,27 seg.
- BBDD_size =182975 · 32 bits= 5855200 bits = 0.73 MB.

Test 5

- Numero Fragmentos aleatorios por canción = 10.
- Máximos emparejados = 3.
- Target Zone Dimensiones (64×32).
- mask =30x30.

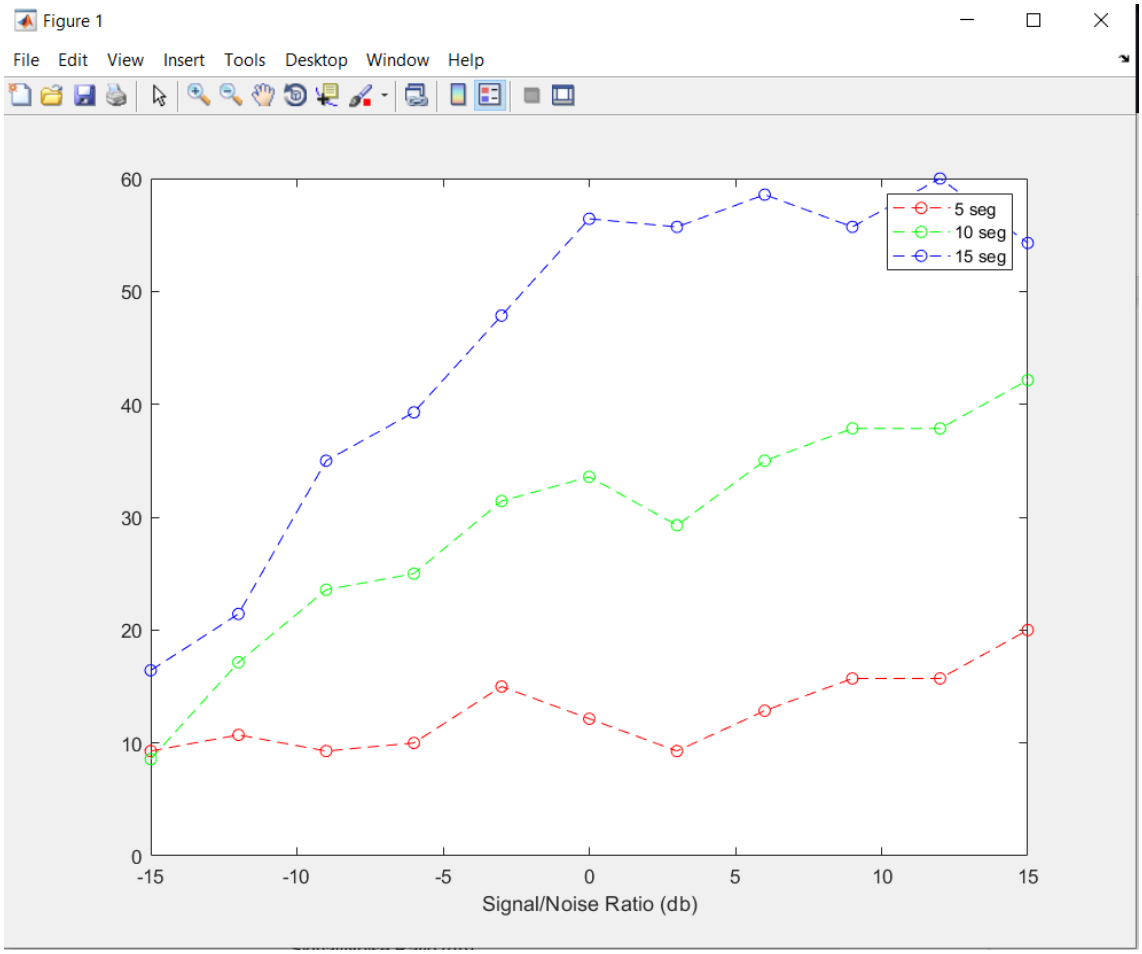


Figura 4.16 Test5.

Tabla 4.5 Test 5.

ratio acierto.(%)	-15 db	-12db	-9 db	-6 db	-3 db	0 dB	3 dB	6 dB	9 db	12 dB	15 dB
5 seg	9,28	10,71	9,28	10	15	12,14	9,28	12,85	15,71	15,71	20
10 seg	8,57	17,14	23,57	25	31,42	33,57	29,28	35	37,85	37,85	42,14
15 seg	16,42	21,42	35	39,28	47,85	56,42	55,71	58,57	55,71	60	54,28

- t_BBDD = 25.93 seg.
- t_identif_T =640.94 seg.
- t_identif_I = 0.13 seg.
- BBDD_size =$71515 \cdot 32$ bits = 2288480 bits = 0.28 MB

Test 6

- Numero Fragmentos aleatorios por canción = 10.
- Máximos emparejados = 3.
- Target Zone Dimensiones (64×32).
- mask =25x25.

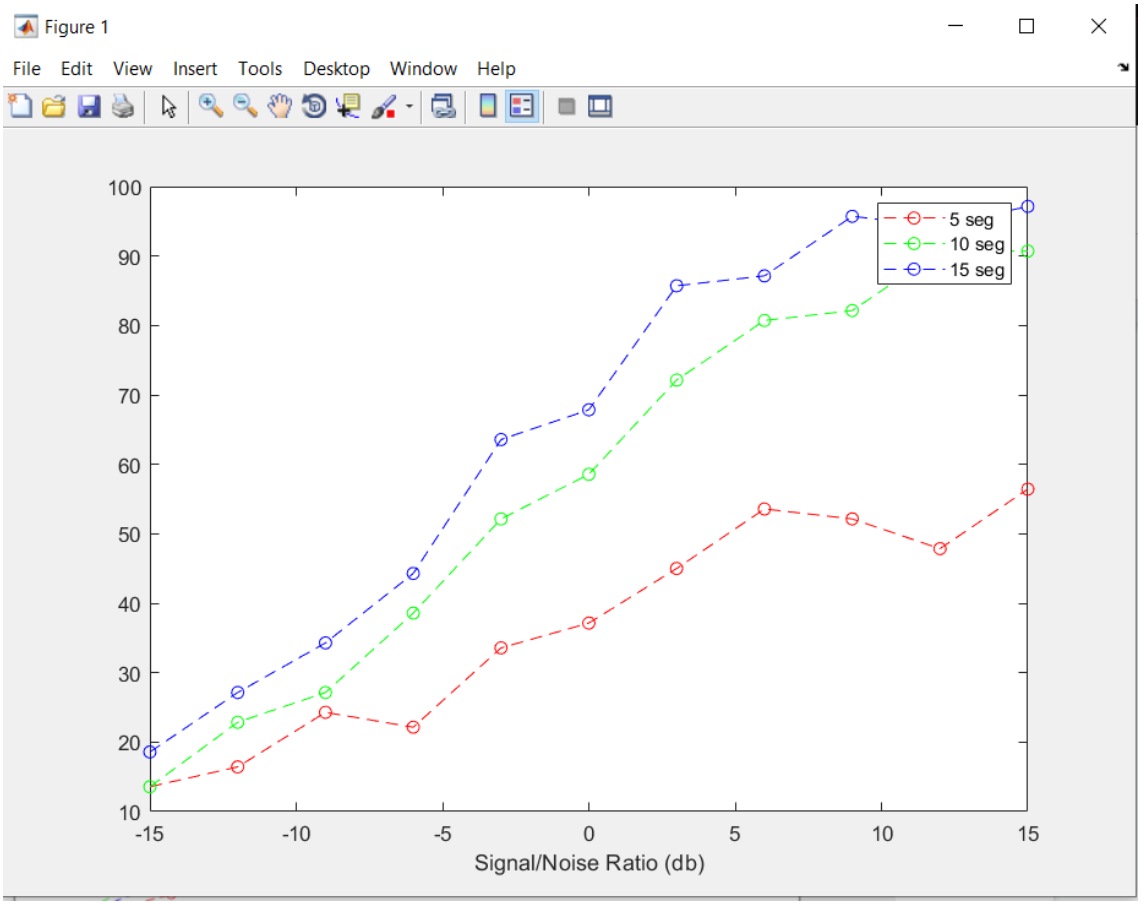


Figura 4.17 Test6.

Tabla 4.6 Test 6.

ratio acierto.(%)	-15 db	-12db	-9 db	-6 db	-3 db	0 dB	3 dB	6 dB	9 db	12 dB	15 dB
5 seg	13,57	16,42	24,28	22,14	33,57	37,14	45	53,57	52,14	47,85	56,42
10 seg	13,57	22,85	27,14	38,57	52,14	58,57	72,14	80,71	82,14	90,71	90,71
15 seg	18,57	27,14	34,28	44,28	63,57	67,85	85,71	87,14	95,71	94,28	97,14

- t_BBDD = 28.57 seg.
- t_identif_T =664.06 seg.
- t_identif_I = 0.14 seg.
- BBDD_size = $109194 \cdot 32$ bits = 3494208 bits = 0.43 MB.

4.2.5 Valoración de los resultados

A la hora de determinar la configuración óptima del algoritmo, analizamos y comparamos los resultados obtenidos de cada uno de los test. Para ello realizaremos una primera criba en función del ratio de acierto, para posteriormente, acercarnos aún más al óptimo del algoritmo a partir de la comparación de tiempos y del tamaño de la base de datos generada.

Viendo las figuras que muestran los ratios de acierto, realizamos una criba inicial, quedándonos con los primeros tres test. Como podemos comprobar, para la máscara de 20x20, los resultados son aceptables. Por lo tanto, pasamos a valorar los tiempos de búsqueda según el número de emparejamientos para cada máximo, es decir, el número máximo de landmarks permitidos, así como el tamaño de la base de datos generada.

Para el Test 1, cuyo número de landmarks por pico de energía es 10, obtenemos los mejores ratios de acierto en todos los niveles, ya que trabajamos con una base de datos más densa, en concreto de 1.42 MB, y tenemos un tiempo de identificación por fragmento aceptable, de 0.22 segundos. No obstante, podemos observar como para 3 emparejamientos, en el Test 3, tenemos una base de datos 3 veces más pequeña(0.59MB), así como una mejora de 0.5 segundos en la identificación, sin que apenas se resienta la robustez. Además, las curvas generadas son parecidas, alcanzando valores cercanos al 60 % incluso en situaciones de distorsión altas para la señal y del 95 % en las mejores situaciones. De la misma forma, nos encontramos en el Test 2 con un máximo de 5 emparejamientos, el cual nos ofrece unos ratios de acierto y tiempo de identificación cercanos a los vistos en los otros tests con la misma máscara (0.95 MB).

Según los resultados obtenidos, observamos que la mejora de ratio de acierto conforme aumentamos el número de landmarks, no supone una mejora tan sustancial como para priorizar una base de datos casi el doble de grande entre los Test 2 y 3. Dado que nuestra base de datos es pequeña, en una situación real con millones de canciones, esta diferencia si será significativa, ya que también aumentará linealmente el tiempo de búsqueda del algoritmo. Con todos los datos presentados, optamos por un modelo cuyos parámetros coinciden con los del tercer test.

Aquí podemos ver una comparativa entre los resultados obtenidos por el algoritmo de Shazam así como nuestra implementación en MATLAB.

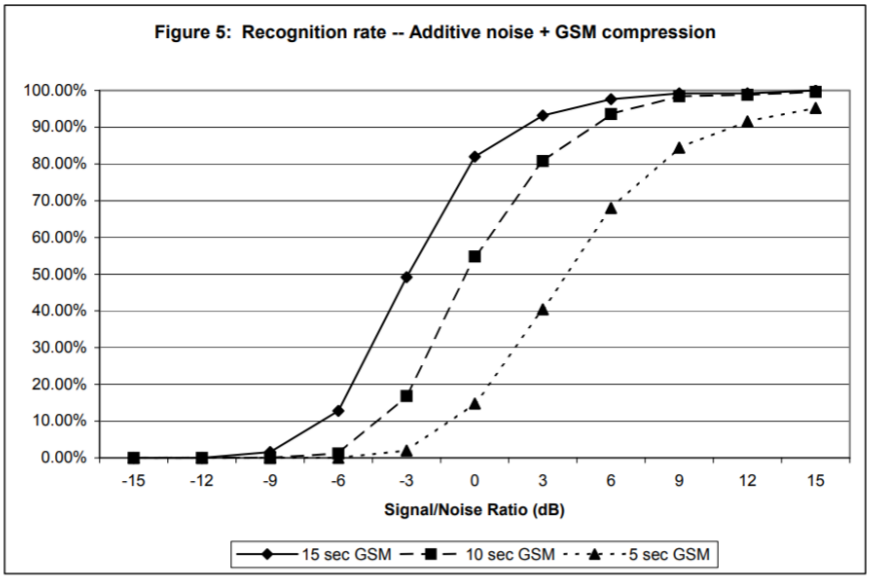


Figura 4.18 Ratio acierto Shazam.

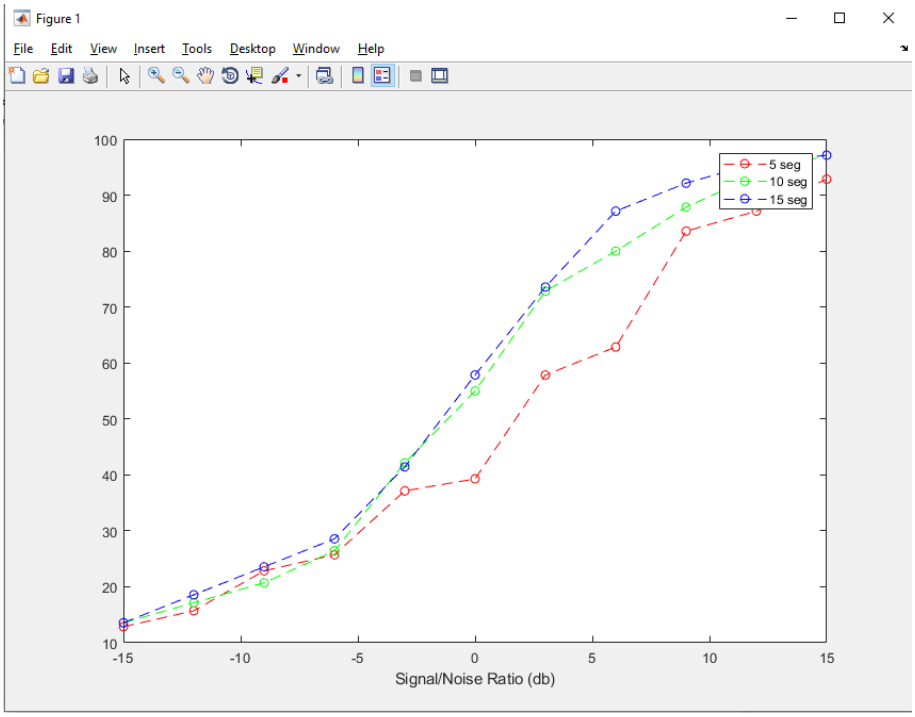


Figura 4.19 Test Optimo.

Tras determinar la comparación entre las tres configuraciones que serían aceptables en términos de fiabilidad y tiempo, pasamos a analizar las grandes diferencias de robustez que encontramos con los test que utilizan distintas variaciones del tamaño de la máscara.

A continuación, analizaremos a qué se debe el empeoramiento de los resultados según nos alejamos de las dimensiones de la máscara óptima:

En el Test 4 tenemos una máscara de 10x10, siendo los resultados de esta los peores en comparación con el resto de las pruebas. Esta configuración presenta varios problemas, ya que un tamaño tan reducido

provoca que tengamos un gran número de emparejamientos de mala calidad, no siendo suficiente el filtrado de máximos, de forma que los puntos emparejados no tienen la suficiente energía y a menudo son enmascarados por el ruido. Tenemos una base de datos de un tamaño considerable, 0.73MB pero que, como podemos comprobar en su figura respectiva, apenas alcanza un 30 % de acierto en el mejor de los casos, siendo esta configuración inaceptable.

Mientras, en el otro extremo analizamos las características de una máscara demasiado grande. El pequeño tamaño de la base de datos (0.28MB) visto en el Test 5 nos indica que al aplicar la máscara hemos eliminado una gran cantidad de máximos aptos para ser emparejados, muchos de ellos lo suficientemente fuertes para resistir las distorsiones. Nos queda una base de datos muy escasa que no nos permite obtener resultados concluyentes, dando lugar a unos ratios de acierto que apenas llegan al 60 % en situaciones muy dispersas. El motivo de unos resultados tan dispares es debido a la aleatoriedad de las pruebas en una base de datos tan pobre, obteniendo unos resultados muy lejanos a lo deseado.

5 Otros Algoritmos

“No documentes el problema; arréglalo”

Atli Björgvin Oddsson

Una vez estudiada la estructura front-end sobre la que funcionan la mayoría de algoritmos fingerprint, vamos a tomar como referencia uno de los algoritmos pioneros y base de los posteriores, el Philips Robust Hash (PRH)[29].

5.1 Philips Robust Hash

A continuación, analizaremos el conjunto de transformaciones y procesos por los que pasa la señal. En la primera parte, vemos desglosado el cálculo del espectrograma donde la longitud de cada frame es de 370 ms. El bloque ‘bit Derivation’ equivale a un filtro binario cuya entrada es la energía de las 33 sub-bandas logarítmicas no superpuestas, cuyo rango cubre el desde 300 Hz hasta 2000 Hz. Realizamos esta división del rango para asemejar la detección con la capacidad del ser humano de percibir un estímulo sonoro según la ley de Weber-Fechner[30].

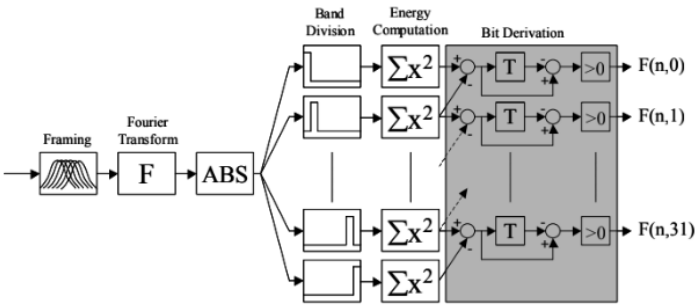


Figura 5.1 Calculo Espectrograma + Filtro.

El filtro binario aplicado es el siguiente:

$$F(n,m)=\begin{cases}1 & \text{if } E(n,m)-E(n,m+1)-\big(E(n-1,m)-E(n-1,m+1)\big)>0\\ 0 & \text{if } E(n,m)-E(n,m+1)-\big(E(n-1,m)-E(n-1,m+1)\big)\le 0\end{cases} \quad (1)$$

Figura 5.2 Filtro binario.

Donde E(n,m) referencia al n frame y a la sub banda m. La salida representa la diferencia de energía entre frames consecutivos y vecinos en bandas de frecuencia.

$$F(n) = [F(n,0), \cdots, F(n,31)],$$

Figura 5.3 Representación binaria 32 bits.

La representación binaria F(n,m) de 32 bits por frame recibe el nombre de sub-fingerprint. Mientras que un fingerprint está compuesta por 256 subfingerprints consecutivas, equivalentes a unos 3 segundos de audio. Una subfingerprint no tiene información suficiente para determinar un emparejamiento, necesitamos como mínimo un fingerprint, por lo tanto, las muestras con las que trabajamos serán como mínimo de 3 segundos.

Cada una de las sub-fingerprint obtenidas funcionará como índice de la Hashtable, su funcionamiento es el mismo que en una tabla de direccionamiento indirecto. Para cada uno de estos índices habrá asociado un valor o más de uno, dichos valores nos redireccionan a la posición original de la canción de la cual se obtuvo este sub-fingerprint. Puede darse el caso de obtener un mismo valor para dos canciones distintas en dos instantes distintos. Ambas serán resultados probables que estudiaremos posteriormente.

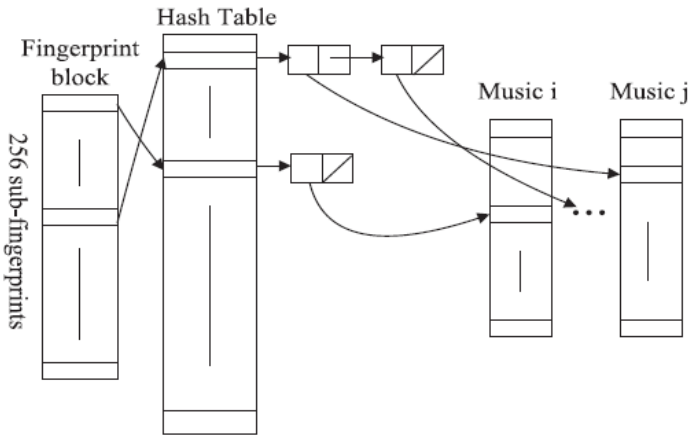


Figura 5.4 Fingerprint block.

El algoritmo de Philips trata en el principio básico de al menos un fingerprint estará en la Tabla a pesar del ruido de la muestra. Para realizar el procedimiento de búsqueda sometemos al fragmento de audio al mismo tratamiento que a las canciones incluidas en la base de datos. Obtendremos una serie de sub-fingerprints las cuales utilizaremos de forma individual para contrastar su existencia en la HashTable. Normalmente tendremos una serie de candidatos,es decir subfingerprints que hemos encontrado también en la tabla. A partir de la posición candidata que hemos encontrado, calculamos a partir de la canción original un bloque del mismo tamaño que el fragmento. En general a partir de un fragmento de 3 segundos, tenemos 1 fingerprint y 256 sub fingerprints de 32 bits cada una. Tendremos una comparación de 256×32=8192 bits entre el fragmento de señal y la posición del sub fingerprint candidato. Si la tasa de error de bit (BER) entre los dos bloques está por debajo de 0.35 nos proporcionará un resultado positivo.

5.2 DCT BASED MULTIPLE HASHING TECHNIQUE

En el estudio de técnicas que mejoran la robustez del algoritmo de Philips encontramos un framework amateur desarrollado en la universidad de Seúl[22] que presenta la siguiente novedad: el uso de múltiples tablas Hash. Este framework propone la extracción de varios sub-fingerprints por frame, utilizando la transformada del coseno en cada una de las bandas de energía.

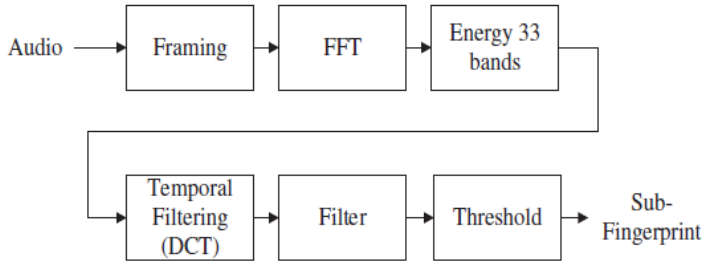


Figura 5.5 Esquema DCT Filter .

Las tres primeras fases del proceso son equivalentes al algoritmo de Philips, pero en lugar de aplicar el filtro sobre las bandas de energía, se aplica sobre cada una de estas la transformada del coseno. La transformada del coseno es una modificación de la transformada discreta de Fourier que trabaja únicamente con números reales, ya que la descomposición de señales se hace en cosenos mientras que la DFT lo hace con exponenciales complejas. Una de sus ventajas es la capacidad de compresión logrando compactar la mayor parte de la información en un número reducido de coeficientes, otro de sus factores determinantes es la reducción de ruido en las discontinuidades[31].

En funciones, el algoritmo trabaja con una DCT de L-puntos, para L bandas de energía consecutivas. Como habíamos visto con anterioridad, la componente n se refiere al frame y m a la banda de energía.

Pese al cálculo de los L coeficientes solo nos quedamos con los primeros K, esta particularidad hace referencia a la capacidad descrita anteriormente de la DCT, que logra compactar la mayor parte de la información en un número reducido de coeficientes. Siendo el filtro aplicado el siguiente:

$$\begin{aligned} ED_k(n,m) = C_k(n,m) - C_k(n,m+1) \\ - (C_k(n-L,m) - C_k(n-L,m+1)). \end{aligned}$$

Figura 5.6 DCT Ecuacion Filtro .

Obtenemos el k sub-fingerprint Fk(n) para el frame n, a partir del umbral visto.

$$F_k(n,m) = \begin{cases} 1, & ED_k(n,m) > 0 \\ 0, & ED_k(n,m) \leq 0. \end{cases}$$

Figura 5.7 DCT Filtro-k .

$$F_k(n) = [F_k(n,0), \cdots, F_k(n,31)],$$

Figura 5.8 DCT Filtro-k Binario .

Como resultado obtenemos k sub-fingerprint para cada frame, que serán utilizados como índices para cada unas de las K Hash Tables.

Pese a que el pensamiento racional sería descartar la opción de utilizar múltiples tablas debido a que el espacio de almacenamiento así como los tiempos de identificación deberían ser a priori más elevados que con otras técnicas lo cierto es que gracias a las propiedades de la DCT esto no es así. Existe una gran compactación de energía en un rango de frecuencias bajos, esto quiere decir que la mayor parte de la información se encuentra concentrada.

La correlación de los coeficientes de la DCT[32] es muy interesante para la comprensión del espectro de energía, ya que posteriormente nos ha permitido su tratamiento de forma independiente sin pérdida de

eficiencia de compresión. Este comportamiento se asemeja en gran medida a al resultado obtenido de la transformada de Karhunen-Loève , con las ventajas de tener un algoritmo de transformada rápida y un coste computacional mucho menor.

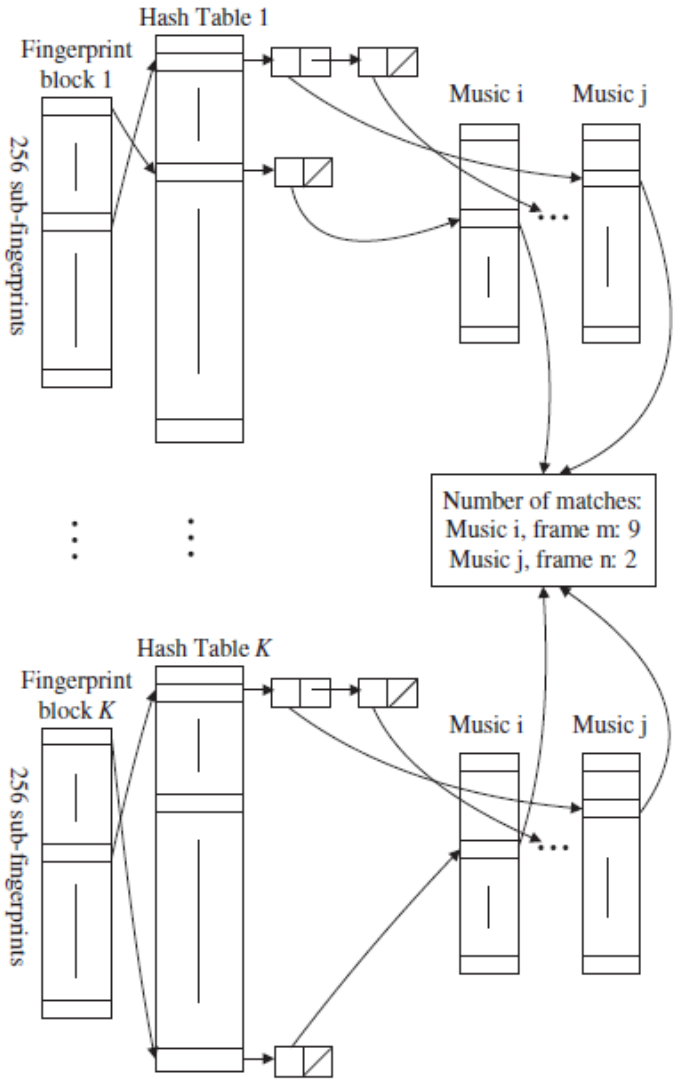


Figura 5.9 DCT MultiHash Fingerprint Block .

El proceso de búsqueda es equivalente al algoritmo de Philips, viéndose ahora incrementado el número de candidatos, estos son ordenados según el número de matches y son comparados frente al bloque obtenido a partir del cálculo de la tasa de error de bit (BER) del fragmento con cada uno de los candidatos. Permitiendo determinar a qué canción pertenece y su instante de origen.

6 Conclusiones

El principal objetivo del presente estudio era lograr una implementación del algoritmo de Shazam en el lenguaje matemático Matlab y realizar un estudio sobre la robustez del mismo. Para conseguirlo, he necesitado realizar un estudio en profundidad de técnicas para el tratamiento digital de señales como el espectrograma. El análisis de otras aplicaciones de identificación basadas en el contenido de la señal, ha resultado enriquecedor para el desarrollo de la técnica, así como de incentivo para desarrollos futuros.
En líneas generales, me considero satisfecho con los objetivos alcanzados, tanto en el desarrollo del algoritmo, como con el estudio de la robustez ya que el algoritmo alcanza un nivel de eficiencia y eficacia óptimos a pequeña escala. A su vez, la robustez está fundamentada por los resultados obtenidos durante la fase de testeo donde se obtienen unos ratios de acierto notables pese a las distorsiones que recrean situaciones reales.

Considero el tema de estudio de gran interés debido a las implicaciones que pueden alcanzar ante situaciones muy diversas los algoritmos de identificación de audio, ya sea relacionado con identificación de música o voz. Siendo aplicables en nuestro día a día, donde las aplicaciones cada vez están más unidas a nosotros y entre ellas mismas, dando lugar a un sinfín de nuevas opciones y funcionalidades.

Índice de Figuras

Índice de Tablas

Bibliografía

[1] https://web.archive.org/web/20120922033806/http://www.shazam.com/music/web/about.html

[2] https://web.archive.org/web/20140714201729/http://ar.todocelular.com/seguridad/noticias/n27085/algunas-apps-android-no-protegen-datos.html

[3] https://www.ee.columbia.edu/ dpwe/resources/matlab/fingerprint/

[4] S. BRAUN. Discover Signal Processing: An Interactive Guide for Engineers. Ed. John Wiley & Sons, 2008.

[5] Ladefoged, Peter (2006) A Course in Phonetics (5ª edición), Boston, MA: Thomson Wadsworth, p. 188. ISBN 1-4130-2079-8.

[6] https://web.archive.org/web/20160303220439/http://claws.eng.ua.edu/attachments/210_-Chapter%206.pdf

[7] Proakis, John; Manolakis, Dimitris (2014). «7. The Discrete Fourier Transform: Its Properties and Applications». Digital Signal Processing (Pearson New International Edition) (en inglés) (4 edición). Pearson Education Limited. p. 468. ISBN 978-1-292-02573-5.

[8] https://www.phon.ucl.ac.uk/courses/spsci/acoustics/week1-10.pdf

[9] https://www.ni.com/es-es/innovations/white-papers/06/understanding-ffts-and-windowing.html#section-1472285423

[10] http://cursodeacusticamusical.blogspot.com/2016/02/capitulo-10-analisis-espectral-de-los.html

[11] P. Cano, E. Batle, T. Kalker and J. Haitsma, "A review of algorithms for audio fingerprinting," 2002 IEEE Workshop on Multimedia Signal Processing., St.Thomas, VI, USA, 2002, pp. 169-173, doi: 10.1109/MMSP.2002.1203274.

[12] Craver, S. A, Wu. M., & Liu, B. (2001). What Can We Reasonably Expect From Watermarks?. Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics.

[13] Furon, T., Moreau , N., and Duhamel, P. (2000). Audio public key watermarking technique. Proceedings of the ICASSP.

[14] Garcia, R. A. (1999). Digital watermarking of audio signals using a psychoacoustic auditory model and spread-spectrum theory. 107th AES Convention.

[15] Bender, W., Gruhl, D., Morimoto, N., & Lu, A., (1996). Techniques for data hiding. IBM System Journal vol. 35, pp. 313-336.

[16] Lacy, J., Quackenbush, S., Reibman, A., Shur D., & Snyder, J. (1998). On combining watermarking with perceptual coding. Proceedings of the International Conference on Acoustic Speech and Signal Processing, Seattle.

[17] Haitsma, J.A., Kalker T., Oostveen, J. (2001). Robust Audio Hashing for Content Identification. Second International Workshop on Content Based Multimedia and Indexing.

[18] Oppenheim, A. V. & Schafer, R. W. (1989). Discrete-Time Signal Processing. Prentice Hall.

[19] P. Cano et al.: Audio Fingerprinting: Concepts And Applications, Studies in Computational Intelligence (SCI) 2, 233–245 (2005)

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

[21] https://www.ni.com/es-es/innovations/white-papers/06/understanding-ffts-and-windowing.html#section-1472285423

[22] S. Theodoris and K. Koutroumbas, Pattern Recognition. Academic Press, 1999

[23] Yu Liu, Kiho Cho, Hwan Sik Yun, Jong Won Shin and Nam Soo Kim, "DCT based multiple hashing technique for robust audio fingerprinting," 2009 IEEE International Conference on Acoustics, Speech and Signal Processing, Taipei, 2009, pp. 61-64, doi: 10.1109/ICASSP.2009.4959520.

[24] G. Richly, L. Varga, F. Kovacs, ´ and G. Hosszu,´ "Short-term sound stream characterisation for reliable, real-time occurrence monitoring of given sound-prints," in Proc. 10th Mediterranean Electrotechnical Conference, MEleCon, 2000.

[25] Logan B., "Mel Frequency Cepstral Coefficients for Music Modeling", Proceeding of the International Symposium on Music Information Retrieval (ISMIR) 2000, Plymouth, USA, October 2000.

[26] A. Kimura, K. Kashino, T. Kurozumi, and H. Murase, "Very quick audio searching: introducing global pruning to the time-series active search," in Proc. of Int. Conf. on Computational Intelligence and Multimedia Applications, Salt Lake City, Utah, May 2001.

[27] Neuschmied H., Mayer H. and Battle E., "Identification of Audio Titles on the Internet", Proceedings of International Conference on Web Delivering of Music 2001, Florence, Italy, November 2001.

[28] Avery Li-Chun Wang and Julius O. Smith, III., WIPO publication WO 02/11123A2, 7 February 2002, (Priority 31 July 2000)

[29] (2002) Etantrum. Online. Available: http://www.freshmeat.net/projects/songprint

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

[31] W. Matlin, Margaret; Foley, Hugh J. (1996) Sensación y percepción, Prentice Hall Hispanoamericana 1996. ISBN 9688806773

[32] https://maixx.files.wordpress.com/2012/10/cap12_dct_v01_01_03.pdf

[33] https://signalsprocessingup.files.wordpress.com/2011/12/dct.pdf