Proyecto Especial: Reconocimiento de audio mediante huellas digitales acústicas

Vázquez, Matías - 91523
mfvazquez@gmail.com

7 de diciembre de 2015

En este trabajo se implementará un software de reconocimiento de canciones mediante huellas digitales acústicas en Octave. Para su desarrollo se hará uso de técnicas y herramientas de análisis de señales; DFT, espectrogramas, diseño de filtros digitales y cambio de la tasa de muestreo.

1. Introducción

1.1. Reconocimiento mediante huellas digitales acústicas

El reconocimiento de audio mediante huellas digitales acústicas es una técnica que busca identificar una pieza de audio, contrastando la información contenida en dicha pieza contra la almacenada en una base de datos de pistas conocidas.

Existen varias formas de abordar el problema del reconocimiento, pero todas persiguen la misma finalidad:

- Simplicidad computacional: el reconocimiento debe realizarse en forma rápida.
- Ser eficiente en el uso de memoria y poseer buena capacidad de discriminación
- Ser robusto ante distorsiones: ruido de fondo, degradación por compresión digital del audio, ecualización por respuesta en frecuencia no plana del lugar y/o parlantes, etc.
- Granularidad: capacidad de reconocimiento utilizando solo un segmento del audio.
- Alta tasa de aciertos, baja tasa de falsos positivos y de rechazos.

El procedimiento básico consiste en analizar el segmento de audio, buscando extraer características particulares en el espacio tiempo-frecuencia (es decir, en su espectrograma) que sirvan para identificarlo luego. Una característica podría ser, por ejemplo, la potencia que posean las diferentes bandas de frecuencias. Al conjunto de estas características se las denomina huella digital acústica , ya que es análogo a como una huella dactilar se identifica por las convergencias, desviaciones, interrupciones y demás particularidades de las crestas papilares.

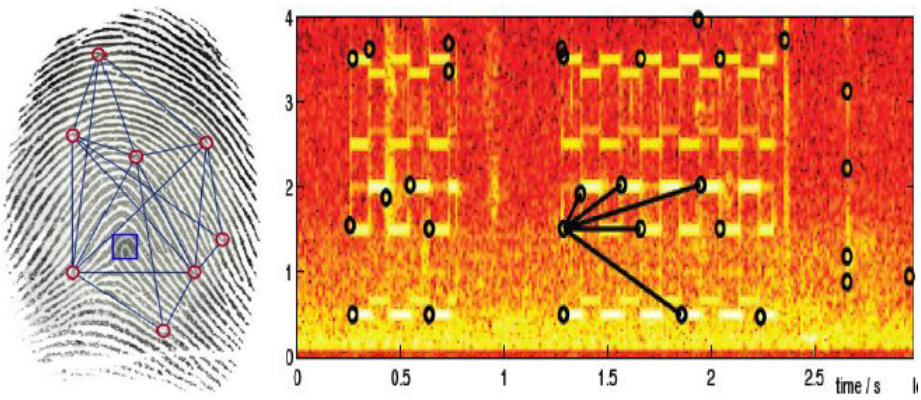

Figura 1: Analogía entre una huella digital tradicional y una huella digital acústica

Este procedimiento de extracción de huellas se utiliza tanto para confeccionar la base de datos de canciones conocidas como para posteriormente reconocer segmentos de audio sin identificar, consultando la base de datos.

La elección de las características es la base del funcionamiento del método, ya que serán las claves que se utilizan luego para realizar la búsqueda en la base de datos. Por lo tanto, deben ser lo suficientemente específicas como para identificar a cada canción unívocamente, pero a la vez ocupar poca memoria para permitir realizar la búsqueda en forma rápida y eficiente. También deberán ser relativamente inmunes ante ruidos y distorsiones del audio, de manera de lograr un sistema de reconocimiento robusto.

En este trabajo se desarrollará una implementación que utiliza como características el signo de la diferencia de energía entre bandas, también conocido como algoritmo Philips.

Cabe destacar que este y los algoritmos antes mencionados sólo reconocen las mismas versiones de las canciones que se almacenan en la base de datos, es decir, no están diseñados para reconocer versiones en vivo o interpretaciones diferentes a la original.

1.2. Descripción del sistema de reconocimiento

El sistema de reconocimiento consta de dos bloques principales:

1. El algoritmo para extraer las huellas digitales acústicas.
2. La base de datos que almacena las huellas acústicas de las canciones conocidas y permite realizar búsquedas a partir de la huella acústica de una canción desconocida.

El algoritmo para extraer las huellas acústicas toma como entrada el canal de audio y, luego de un preprocesamiento de la señal, extrae las características y entrega como resultado la huella acústica. El proceso se esquematiza a continuación:

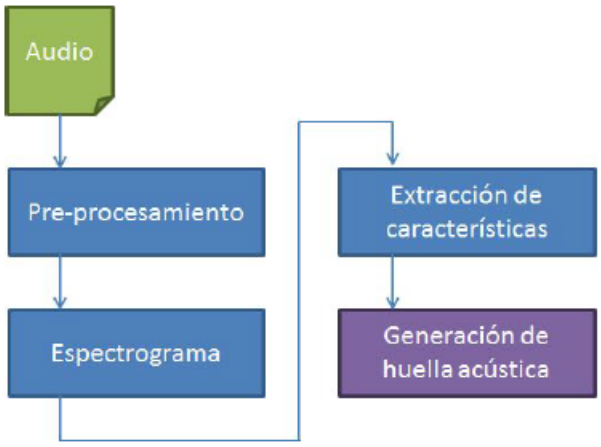

Figura 2: Esquema del algoritmo para extraer la huella digital acústica

La base de datos guardará las huellas acústicas de las canciones conocidas. Tiene además un sistema de búsqueda tal que al realizar un query con una huella acústica dada nos devuelve la canción más probable a la cual corresponde.

El esquema del sistema completo se presenta a continuación:

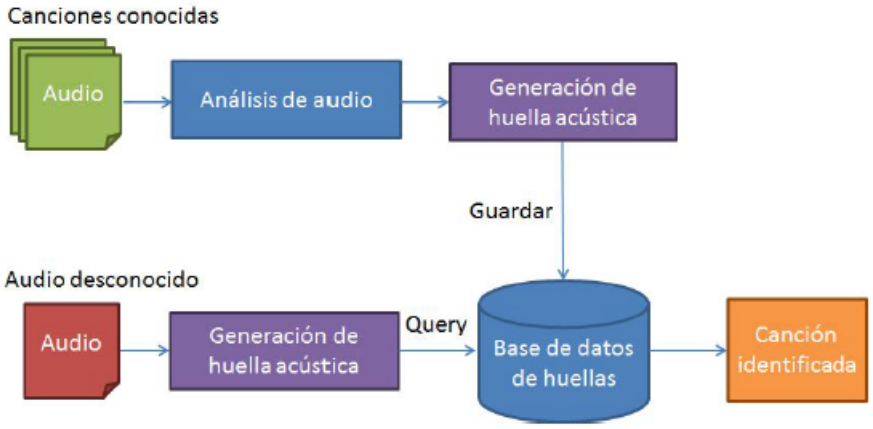

Figura 3: Esquema representando la generación de la base de datos de huellas acústicas y la consulta.

2. Pre-procesamiento del audio

El pre-procesamiento del audio consiste en pasarlo a canal mono (si está en estéreo) y luego sub-muestrearlo, debido a que la información útil para la extracción de características se encuentra en bajas frecuencias. En general, el audio está muestreado a 44100 Hz (calidad de CD). Para este algoritmo basta con tenerlo muestreado a 1/8 de su frecuencia original, es decir 5512,5 Hz. Esto permite trabajar con menos muestras, aliviando la carga computacional.

2.1. Obtención de la pista de audio

Mediante la función `audioread` se cargó la pista de audio. Si el audio cargado es una matriz de 2 columnas, cada columna corresponde a un canal de audio, se promedian los canales utilizando la función `mean` para tener 1 solo canal. En la figura 4 se muestra una porción de la señal resultante.

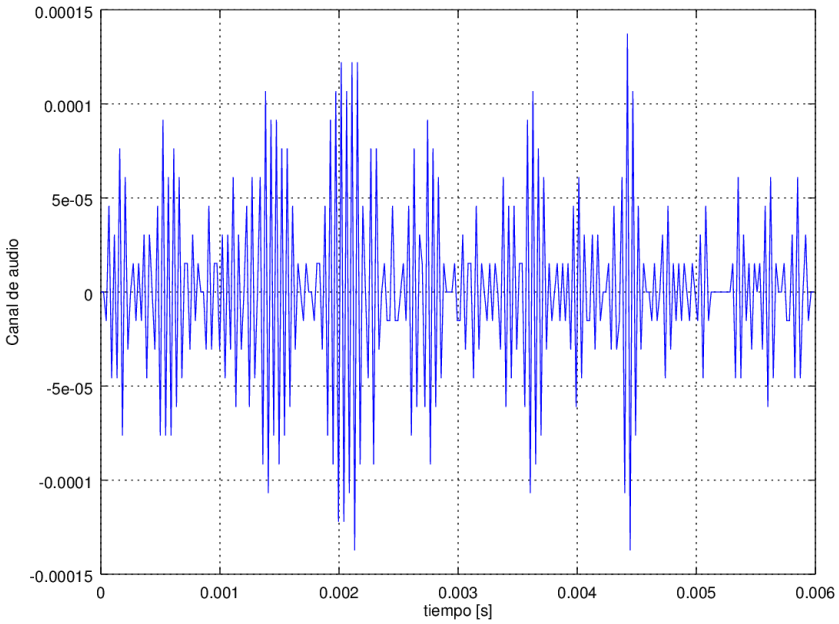

Figura 4: Porción de la pista de audio.

2.2. Análisis espectral de la señal

Se realizó la FFT del canal de audio, para luego obtener la densidad espectral de potencia. La densidad espectral de potencia S_{xx} se define como:

$$S_{xx}(w) = |X(w)|^2 \tag{1}$$

donde $X(w)$ es la transformada de Fourier del canal de audio. Para pasarla a escala logarítmica se utilizó la siguiente ecuación:

$$P_{xx} = 10log_{10}(S_{xx}(w)) \tag{2}$$

La estimación de la densidad espectral mejora si se divide el audio en segmentos y promedia (en potencia) el espectro de cada segmento. Por lo que se utilizó la función `specgram`[1], con una longitud de ventana de 256 y overlap 0, para luego obtener su densidad espectral de potencia mediante la ecuación 2 y finalmente promediarla en tiempo. En la figura 5 se muestra el resultado obtenido, como se puede observar la mayor densidad de potencia se encuentra en las bajas frecuencias. Para frecuencias superiores a los 2,5 KHz la densidad espectral de potencia se encuentra por debajo de los 0 dB.

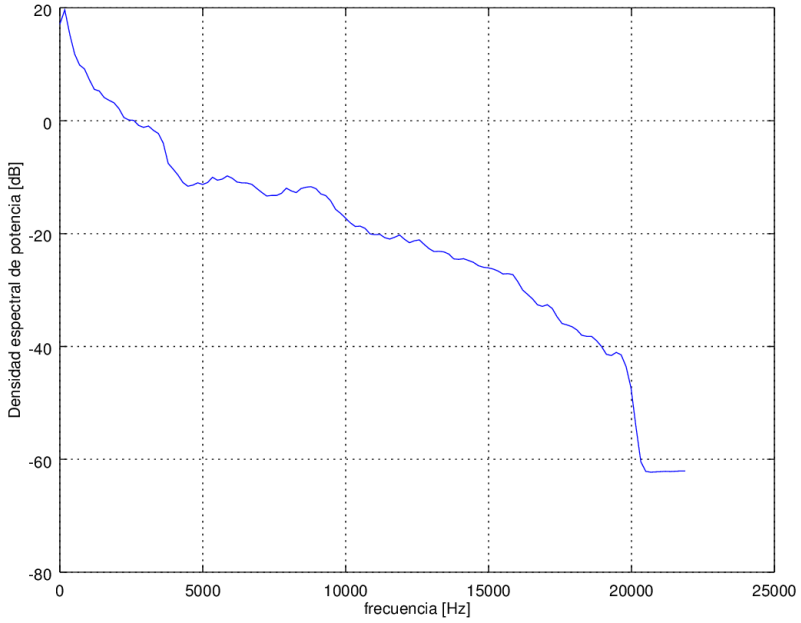

Figura 5: Densidad espectral de potencia del canal de audio.

2.3. Reducción de la frecuencia de muestreo

Se implementó un sistema para reducir la frecuencia de muestreo del audio de 44100 Hz a 5512,5 Hz. En la figura 6 se muestra el diagrama en bloques del sistema. Primero se filtra la señal con un filtro pasabajos, para evitar el aliasing al submuestrear la señal. Ya que si se reduce la frecuencia de muestreo a 5512,5 Hz debemos filtrar las frecuencias mayores o iguales $\frac{5512,5}{2}$ Hz = 2756,25 Hz.

Figura 6: Diagramas en bloques del sistema.

[1] Se utilizó esta función en lugar de `fft` ya que realiza de forma más eficiente la obtención de la FFT del canal, además de que utiliza menos memoria ya que solo retorna las frecuencias entre 0 y π a diferencia de la `fft` que devuelve las frecuencias entre 0 y 2π.

Para el diseño del filtro pasa-bajos se impusieron las siguientes condiciones:

- Fase lineal
- Retardo menor a 1 ms

Como la Fase debe ser lineal se optó por utilizar filtros tipo FIR, ya que los filtros IIR no tienen fase lineal a diferencia de los FIR.

Para garantizar un retardo menor a 1 ms el orden del filtro debe ser menor o igual a 88, ya que para ordenes superiores el retardo será mayor a 1 ms.

Se utilizó una ventana de Hamming de orden 88 para el filtro, ya que presenta una mejor atenuación en las frecuencias superiores a la frecuencia de corte, a diferencia de la ventana rectangular. Con una frecuencia de corte de 1750 Hz. Se eligió una frecuencia de corte menor a los 2756,25 Hz debido a que el filtro no es ideal y a que no puede tener un orden superior a 88.

Mediante la función `zplane` se obtuvo el diagrama de polos y ceros mostrado en la figura 7

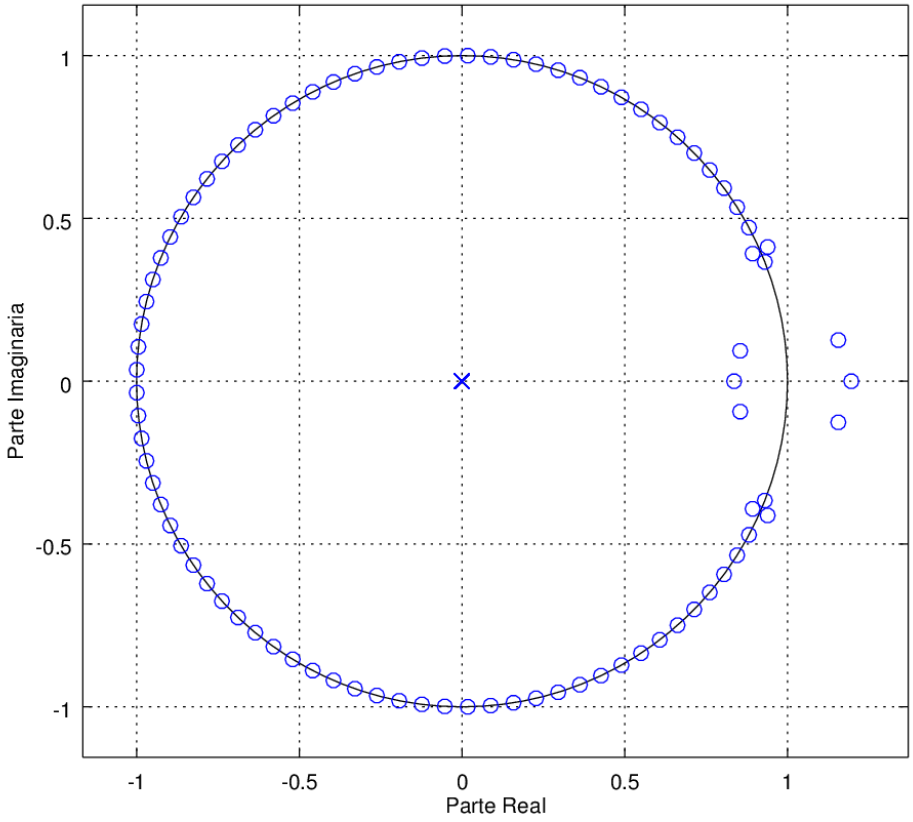

Figura 7: Diagrama de polos y ceros

Mediante la funcion `freqz` se obtuvo la respuesta en frecuencia mostrada en la figura 8.

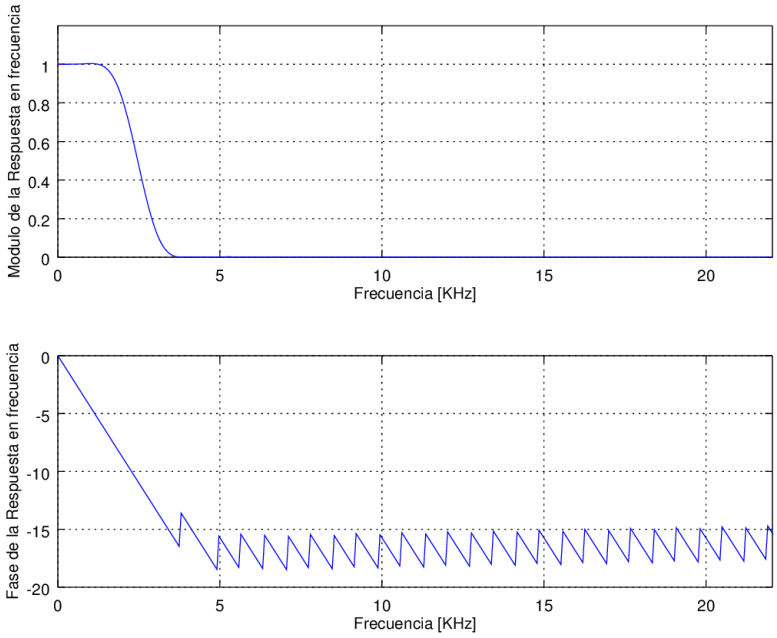

Figura 8: Respuesta en frecuencia

Mediante la funcion `impz` se obtuvo la respuesta al impulso mostrada en la figura 9.

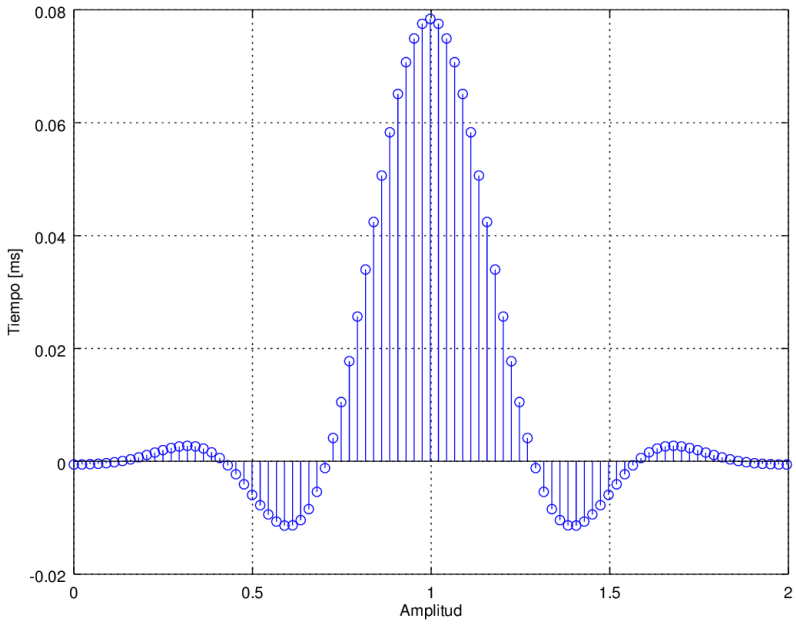

Figura 9: Respuesta al impulso

Mediante la funcion `grpdelay` se obtuvo la respuesta en frecuencia mostrada en la figura 10. Siendo el retardo de grupo de 997,732 μs

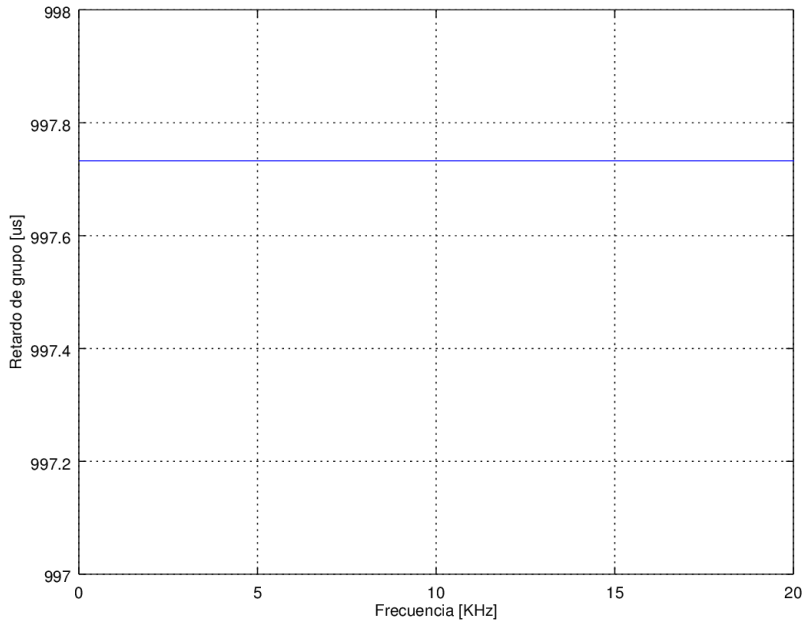

Figura 10: Retardo de grupo

2.4. Espectrogramas de la señal original y la filtrada

Mediante la función `specgram`, con una ventana de Hamming de longitud 256 y overlap del 50 %, se obtuvieron los espectrogramas del canal de audio(figura 11) y del canal de audio luego ser filtrado (figura 12). Como puede observarse en el espectrograma de la señal filtrada, para frecuencias superiores a los 2,5 KHz la potencia se encuentra alrededor de los -50 dB.

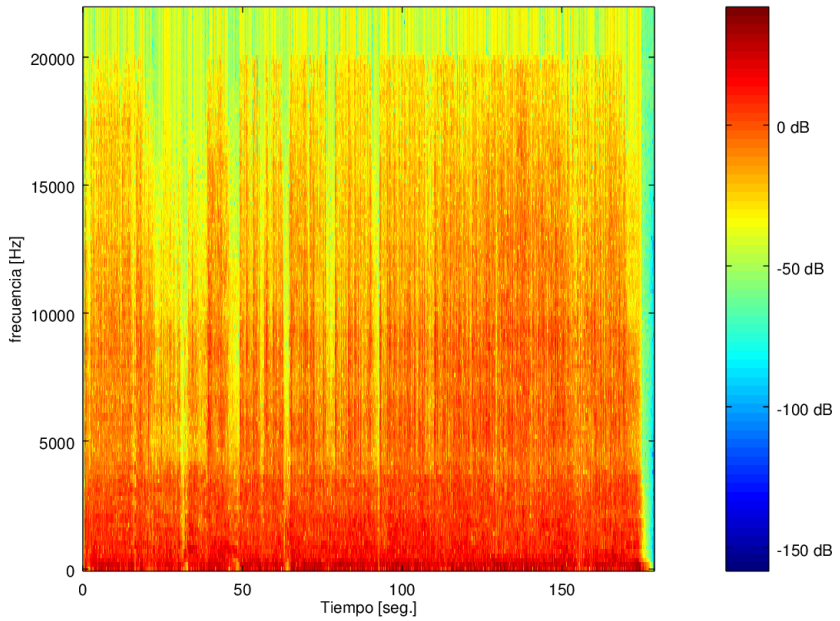

Figura 11: Espectrograma del canal de audio

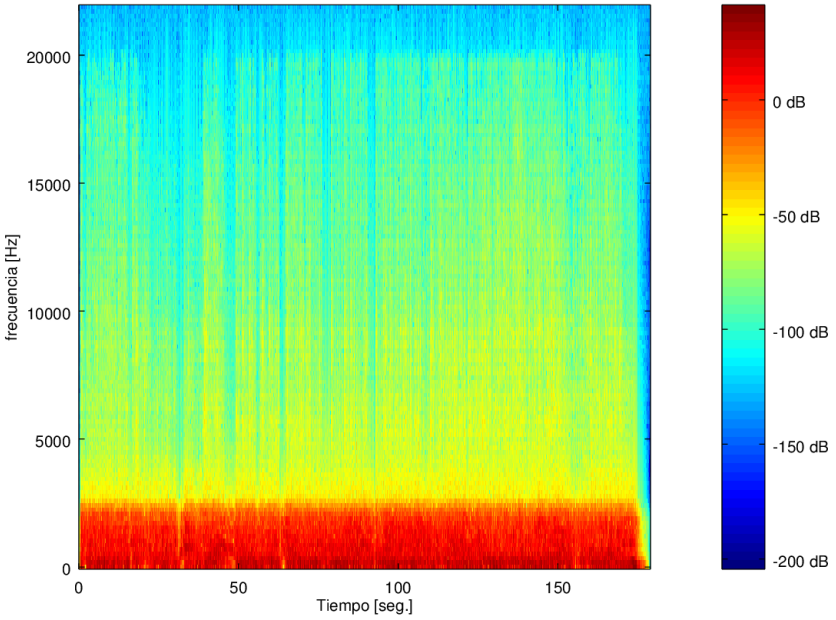

Figura 12: Espectrograma del canal de audio filtrado

2.5. Comparación entre ventana rectangular y Hamming

Mediante la función `fft` se obtuvo el espectro de una ventana rectangular y una de Hamming, ambas de duración 26. En la figura 13 se graficó la potencia de ambas ventanas en escala logarítmica en un mismo gráfico para compararlos, con una separación entre los puntos del gráfico de 70 Hz por lo que la resolución es $\frac{44100\,\text{Hz}}{70\,\text{Hz}} = 630$.

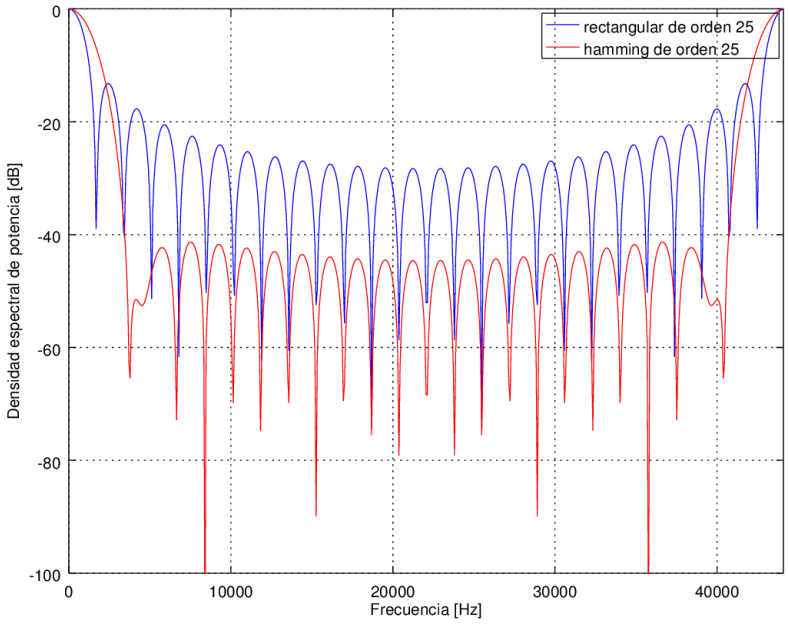

Figura 13: Potencia en escala logaritmica del espectro de una ventana rectangular y Hamming

Como puede observarse en la figura la ventana de Hamming cuenta con una mayor atenuación en sus lóbulos secundarios a comparación de la ventana rectangular, esto es importante ya que al utilizarla

para realizar espectrogramas se va a obtener menos solapamiento que utilizando la ventana rectangular. También puede observarse que el lóbulo principal de Hamming tiene un ancho mayor al de la rectangular.

2.6. Comparación de espectrogramas con diferentes ventanas

Se realizaron espectrogramas de la señal submuestreada con ventana rectangular y Hamming, con 3 longitudes de ventana diferentes para cada ventana.

A continuación se muestran, para los dos tipos de ventana utilizadas, los espectrogramas con longitudes de 64, 4096 y 8192[2] y overlap del 50 %. Se hizo un zoom entre los 60 s y 80 s del audio.

Espectrogramas usando ventana rectangular:

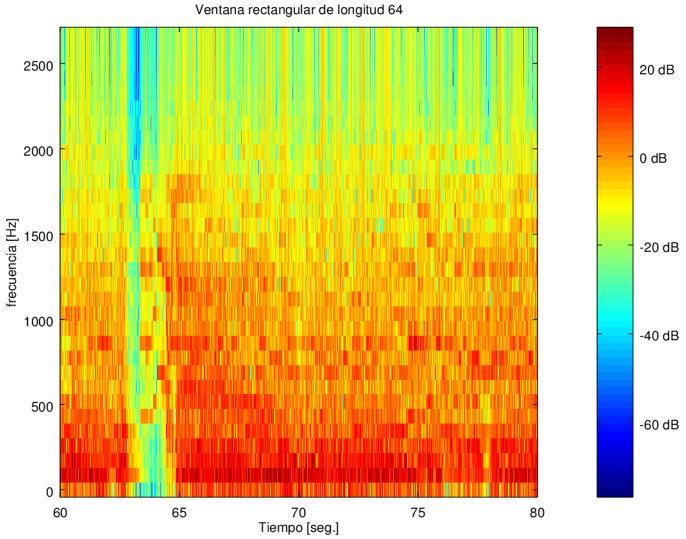

Figura 14: Espectrograma de la señal submuestreada con ventana rectangular de longitud 64

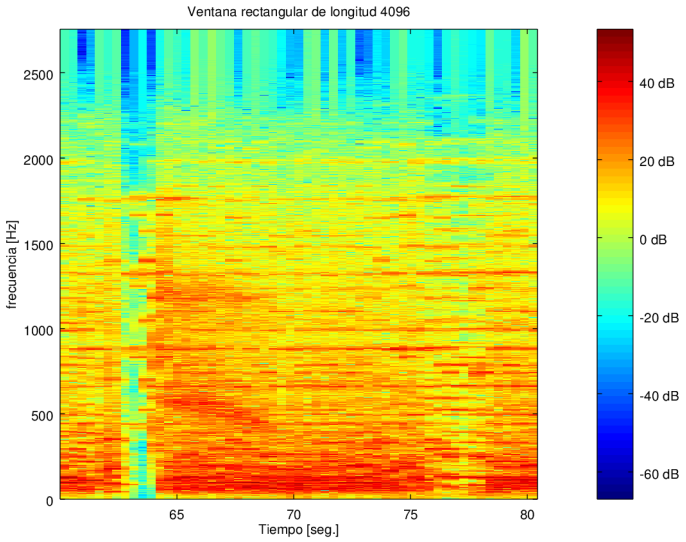

Figura 15: Espectrograma de la señal submuestreada con ventana rectangular de longitud 4096

[2]se eligieron estas longitudes de ventana para aprovechar la eficacia de la FFT ante longitudes que son potencias de 2

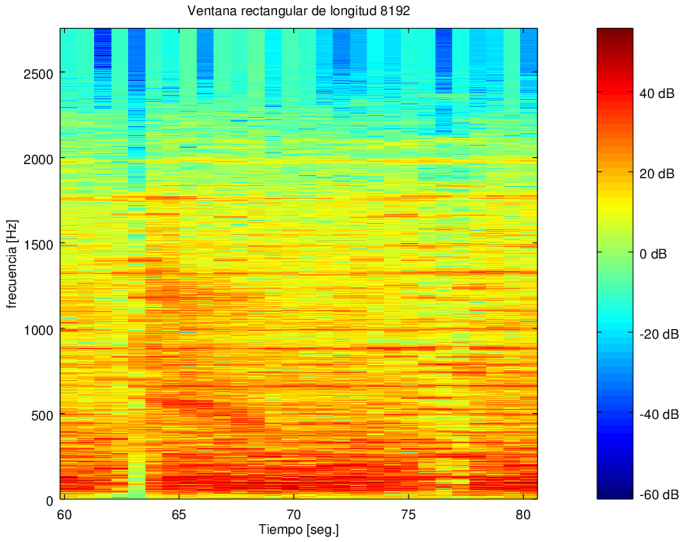

Figura 16: Espectrograma de la señal submuestreada con ventana rectangular de longitud 8192

Espectrogramas usando ventana Hamming:

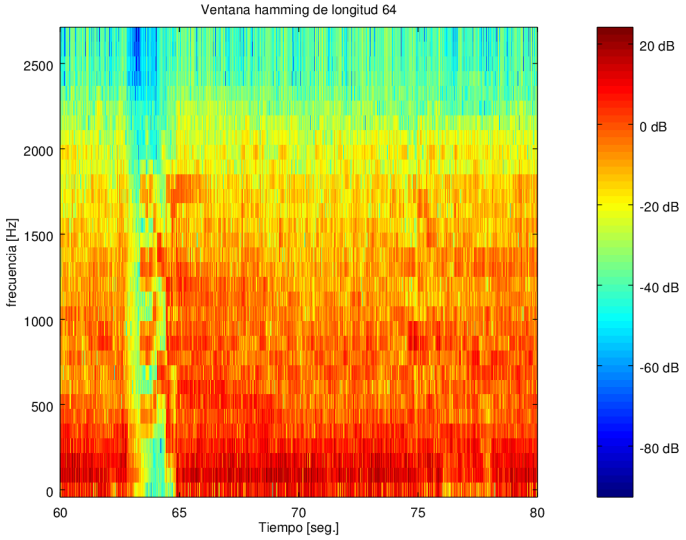

Figura 17: Espectrograma de la señal submuestreada con ventana Hamming de longitud 64

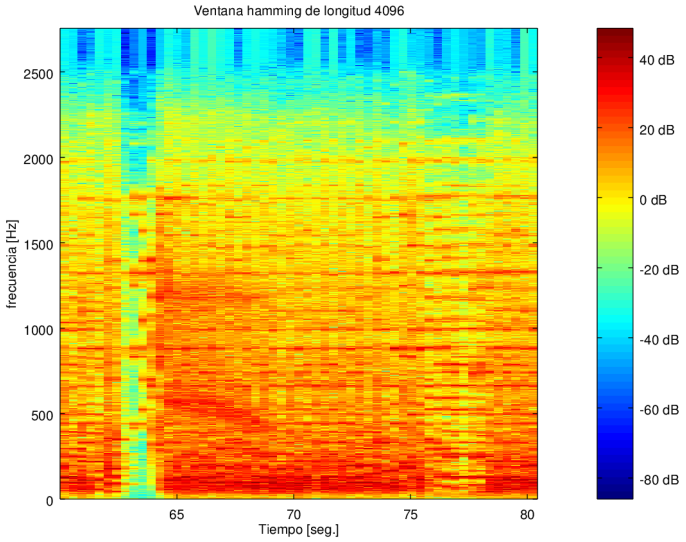

Figura 18: Espectrograma de la señal submuestreada con ventana Hamming de longitud 4096

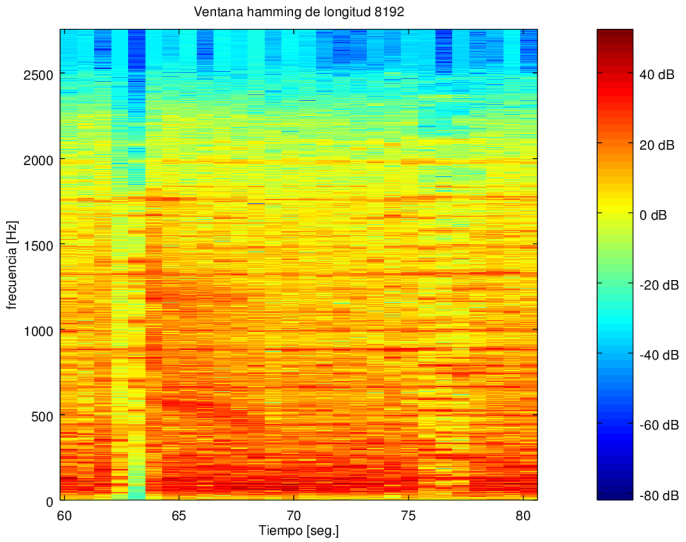

Figura 19: Espectrograma de la señal submuestreada con ventana Hamming de longitud 8192

Como puede observarse en los espectrogramas, a medida que aumenta la longitud de la ventana, disminuye la resolución en tiempo pero aumenta la resolución en frecuencia. De forma inversa a medida que disminuye la longitud de la ventana, aumenta la resolución en tiempo pero disminuye la resolución en frecuencia.

Comparando los espectrogramas entre diferentes ventanas, podemos observar que con ventana Hamming se obtuvieron valores mas precisos. Esto se puede notar en las frecuencias altas, en la ventana rectangular se nota como la potencia de un segmento es influida por la potencia en los segmentos adyacentes. Esto se debe a que los lóbulos secundarios de la ventana rectangular tienen mayor amplitud que los de la ventana de Hamming, por lo que el solapamiento termina influyendo en el resultado.

Esto puede observarse mejor en las siguientes figuras con un zoom ahora en frecuencias entre 1500 Hz y 2756,25 Hz.

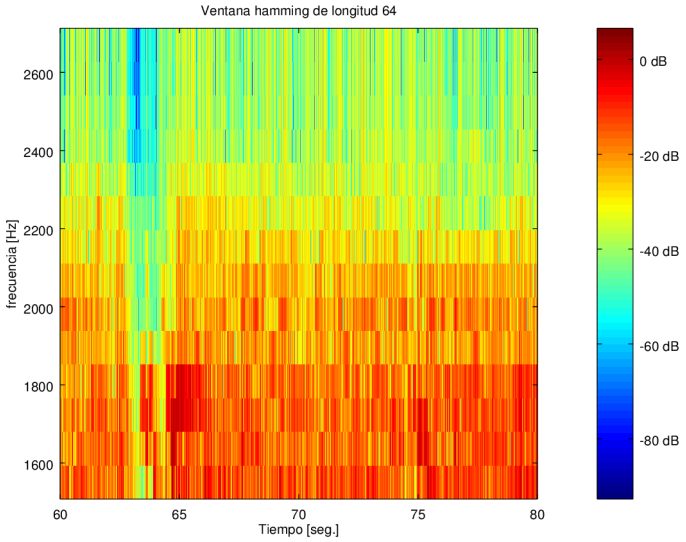

Figura 20: Espectrograma de la señal submuestreada con ventana Hamming de longitud 64

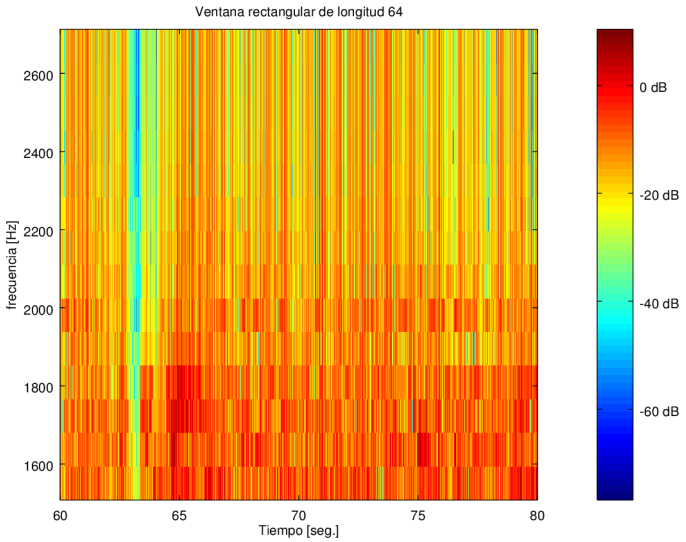

Figura 21: Espectrograma de la señal submuestreada con ventana rectangular de longitud 64

Como se puede observar en el espectrograma con ventana rectangular(figura 21) en las zonas con mayor densidad de potencia alcanza valores cercanos a los 0 dB. En cambio para el espectrograma con ventana Hamming (figura 20) en las mismas zonas alcanza valores de aproximadamente -30 dB.

Para el algoritmo se utilizará una ventana Hamming de longitud 2048.

3. Extracción de características del espectrograma

Para el algoritmo, las características se obtuvieron mediante una función signo $F(m,n)$ que opera sobre la energía de las bandas de frecuencia $E(m,n)$ según:

$$F(m,n) = \begin{cases} 1 & si \quad E(m,n) - E(m+1,n) > E(m,n-1) - E(m+1,n-1) \\ 0 & en\ otro\ caso \end{cases} \tag{3}$$

n es el tiempo y m la banda de frecuencia. Se comprueba que F (m, n) es 1 si la diferencia de energía entre las bandas m y $m+1$ para el frame actual n es mayor a la del frame anterior $n-1$.

Las bandas de frecuencia se definieron con espaciamiento logarítmico (esto es similar a la escala psicoacústica Bark) debido a que el oído diferencia el espaciamiento entre tonos en forma logarítmica.

3.1. Implementación del algoritmo

En el apendice A.1 se encuentra el codigo del algoritmo de extracción de características.

Para dividir el espectrograma en 21 bandas de frecuencia espaciadas logaritmicamente se utilizó la funcion `logspace` para generar un vector de 22 elementos separados logaritmicamente iniciando en 300 Hz y finalizando en 2000 Hz. Luego se recorrió el vector de frecuencias y se fueron tomando las primeras frecuencias que se encontraban entre los segmentos de frecuencias del vector generado. Descartando las demas frecuencias no seleccionadas. En el apéndice A.3 se encuentra el cogido de la función que genera las huellas.

En la figura 22 se ve una huella de un segmento de audio.

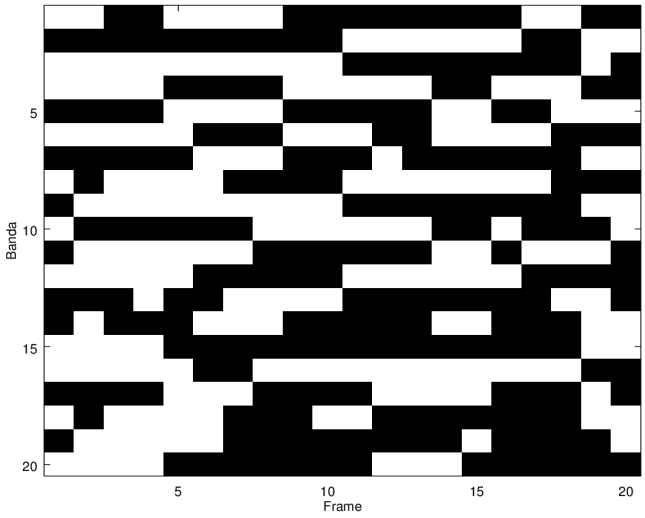

Figura 22: Huella de un segmento de audio

4. Confección de la base de datos

La base de datos es una tabla de 2^{20} filas y N columnas. Para cada frame de la huella acústica, se genera un valor a guardar en la tabla. La fila en la que se guarda cada valor se obtiene pasando a decimal el número binario conformado por las características del frame en cuestión, y la columna en la que se guarda se elige buscando la última columna vacía dentro de esa fila. Note que la cantidad de filas de la tabla es debido a que cada frame de la huella acústica posee 20 elementos binarios. Cada valor a guardar es un entero de 32 bits, de los cuales los 12 bits menos significativos son el ID numérico de la canción que se está analizando, y los 20 más significativos el número del frame. Observe que el máximo número de canciones distintas que se podrán almacenar en esta tabla es 2^{12}.

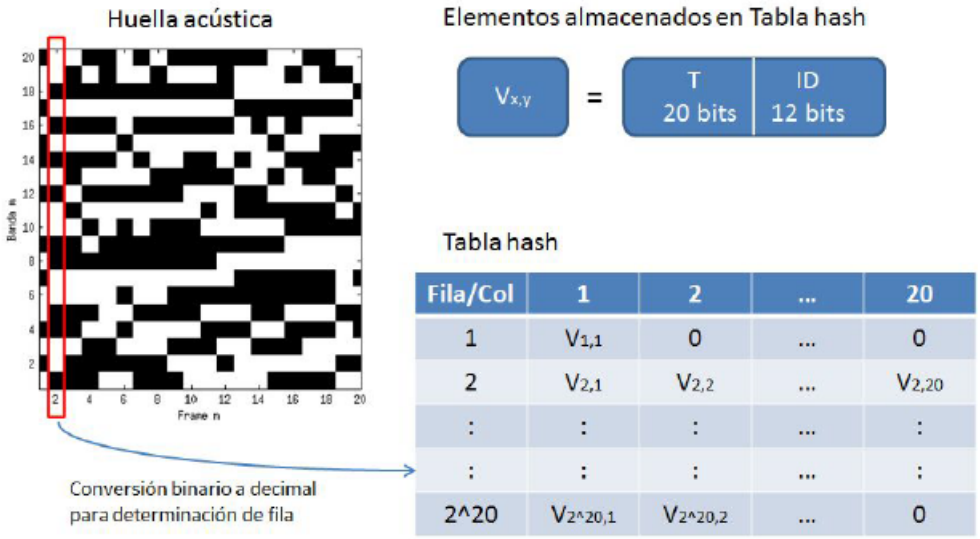

Figura 23: Indexación de tabla hash mediante huella acústica y representación de elementos almacenados.

Para el frame número 2 (recuadrado en rojo) la característica en binario es 11010101011001100110. Pasando este número a decimal obtenemos la fila de la tabla en la que deberá guardarse el valor. Supongamos que la huella corresponde a una canción con ID=5, entonces el elemento a almacenar será, en binario: V=[00000000000000000010 000000000101].

4.1. Script para generar la base de datos

Para generar la base de datos se utilizó el script del apéndice A.4 el cual lee todos los archivos con formato `wav` del directorio `../Musica/`. Utiliza la función que genera huellas acústicas directamente de un archivo, ver apendice A.2.

La base de datos junto a un vector con los nombres de los archivos que la componen son guardadas en un archivo con formato `mat`. El vector con los nombres de los archivos es utilizado para poder leer su información cuando se desee identificar un fragmento de audio.

4.2. Solapamiento entre ventanas del espectrograma

Se calculó un solapamiento de ventanas del espectrograma para obtener una densidad de aproximadamente 25 elementos por segundo.

Como se mencionó antes, el largo de las ventanas es de 2048. El audio cuenta con 5512.5 elementos por segundo. Por lo que se calculó la distancia entre ventanas mediante:

$$distancia = \frac{5512{,}5}{25} = 220{,}5$$

Luego el solapamiento entre ventanas se calcula restándole a la longitud de la ventana la distancia entre ventanas:

$$N_{overlap} = 2048 - 220{,}5 = 1827{,}5$$

Como $N_{overlap}$ debe ser entero, se redondeó el valor utilizando la función `ceil` quedando el siguiente resultado final:

$$N_{overlap} = 1828$$

5. Test del algoritmo

En esta etapa, se puso a prueba la eficacia del algoritmo completo de reconocimiento. El procedimiento consistirá en obtener el porcentaje de aciertos del método al reconocer segmentos de audio, los cuales fueron sometidos a distintas distorsiones.

En los apéndices A.6 y A.7 puede verse el código utilizado para realizar las pruebas de esta sección.

Para todas las pruebas se analizaron dos tipos de aciertos. Como el algoritmo devuelve 5 posibles canciones con orden de prioridad descendente, se analizó la tasa de aciertos cuando la canción figura en el primer puesto y para cuando la canción se encuentra entre los primeros 5 puestos.

5.1. Tasa de aciertos para segmentos de distintas duraciones

Se evaluó la tasa de aciertos del algoritmo identificando segmentos de duración T con tiempo inicial elegido al azar de canciones elegidas al azar. Las canciones son las mismas que se utilizaron para generar la base de datos. Se realizó la evaluación para 50 segmentos con duración T entre 5, 10 y 20 segundos cada vez(150 evaluaciones en total) obteniendo la tasa de aciertos en cada caso.

T[s]	Aciertos en primer puesto	Acierto entre primeros 5
5	50	50
10	50	50
20	50	50

Tabla 1: Aciertos para distintos intervalos de tiempo

Como se ve en la tabla 1 se obtuvo una tasa de aciertos del 100 % para todas las evaluaciones.

5.2. Prueba identificando audios grabados

Se realizó una prueba intentando identificar una canción reproducida de unos parlantes de una computadora, y grabados con un micrófono conectado a la computadora. Para esta prueba la duración del audio fue de 15 segundos y el audio grabado tenia un nivel de volumen mas bajo que el original. El algoritmo logró identificar la canción en el primer puesto de los resultados.

Luego se realizó otra prueba reproduciendo una canción desde el parlante de un celular y grabándolo con el micrófono que viene integrado a una notebook. La duración del audio grabado era de 30 segundos, el volumen del audio era mas bajo que el original. El algoritmo no logró identificar la canción en el primer puesto de los resultados, la canción se encontraba en el segundo puesto. Esto se debe a que los parlantes de los celulares en general son de baja calidad y suelen comportarse como un filtro pasa alto. Si tiene una frecuencia de corte superior a los 300 Hz el algoritmo no devolverá valores acertados ya que esas frecuencias son las que influyen más en la identificación del audio, debido a la escala logaritmica en frecuencia.

5.3. Tasa de aciertos agregando ruido

Se repitió las pruebas de la sección 5.1 pero agregando distintas intensidades de ruido a los segmentos de audio; SNR 0 dB, 10 dB y 20 dB.

Se obtuvo la relación entre la potencia del ruido y la del audio para cada SNR mediante la siguiente ecuación:

$$SNR = 10\ log_{10}\left(\frac{P_x}{P_n}\right) \tag{4}$$

Siendo P_x la potencia del audio y P_n la potencia del ruido.

P_x se puede calcular con la varianza del audio, mediante la función `var` de `Octave`. Las señales de audio son señales probabilisticas, no deterministicas, por lo que se puede aproximar que estas señales tienen media 0 entonces por teoría de probabilidad, la potencia se puede calcular como la varianza del audio.

Se despeja P_n obteniendo:

$$P_n = \frac{P_x}{10^{\frac{10}{SNR}}} \tag{5}$$

Luego se obtiene el vector del ruido que será sumado al audio mediante:

$$ruido = \sqrt{P_n}\ randn(L, 1) \tag{6}$$

Siendo `randn(L,1)` una función que devuelve un vector de longitud L con números al azar entre 0 y 1. L es la longitud del vector del audio.

Finalmente se obtiene el audio con el ruido mediante:

$$audio\ con\ ruido = audio + ruido \tag{7}$$

A continuación se muestra la tasa de aciertos, considerando como acierto que la canción salga en el primer puesto de los resultados:

T \ SNR	0 [dB]	10 [dB]	20 [dB]
5 s	4	45	50
10 s	12	47	50
20 s	12	49	50

Tabla 2: Aciertos en el primer puesto de audios con ruido

Como se puede ver en la tabla 3 a mayor intensidad de ruido menor aciertos tiene el algoritmo. Para reducir este error será necesario tomar intervalos de mayor duración del audio a identificar.

A continuación se muestra la tasa de aciertos, considerando como acierto que la canción se encuentre entre los primeros 5 resultados:

T \ SNR	0 [dB]	10 [dB]	20 [dB]
5 s	50	50	50
10 s	50	50	50
20 s	50	50	50

Tabla 3: Aciertos en el primer puesto de audios con ruido

Según los resultados obtenidos, la canción en todas las evaluaciones se encontraba entre los primeros 5 resultados. Se podría asumir que el algoritmo tiene un margen de error chico, pero para poder afirmar esto es recomendable hacer la prueba con una base de datos mas grande ya que la utilizada tiene tan solo 42 canciones.

5.4. Tasa de aciertos agregando distorciones

Se repitió las pruebas de la sección 5.1 pero solo para $T = 10\,\text{s}$ y afectando los segmentos con al menos 2 tipos de distorsiones elegidas.

5.4.1. Distorsión por Saturación y Ecualización

Para la distorsión por saturación se utilizó un umbral como un porcentaje de la potencia de la señal. Se calculó el valor al cual saturar la señal mediante $sat = \sqrt{var(audio)}\,1{,}5$ Para saturar el fragmento de audio se recorrió el vector del audio y a los elementos del vector que superen a sat se los igualó a dicho valor, y para los valores menores a $-sat$ se los reemplazó por $-sat$.

En la figura 24 se puede ver un espectrograma del fragmento de audio antes de saturarlo y en la figura 25 un espectrograma del fragmento de audio saturado. Se encuentran diferencia en que al audio saturado se le agregaron frecuencias altas en las regiones donde el audio original llegaba a un limite de frecuencia.

La saturación no puede considerarse un sistema LTI ya que no cumple la homogeneidad de la linealidad:

$$T\{\alpha x(t)\} = \alpha T\{x(t)\} = \alpha y(t)$$

Al multiplicar por α a la señal de entrada, podría llegar a saturarse entonces no se cumpliría dicha propiedad.

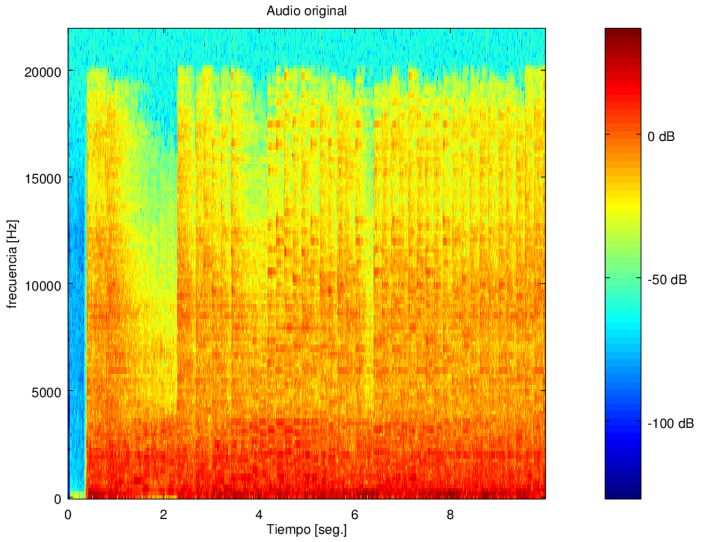

Figura 24: Espectrograma del fragmento de audio antes de saturar.

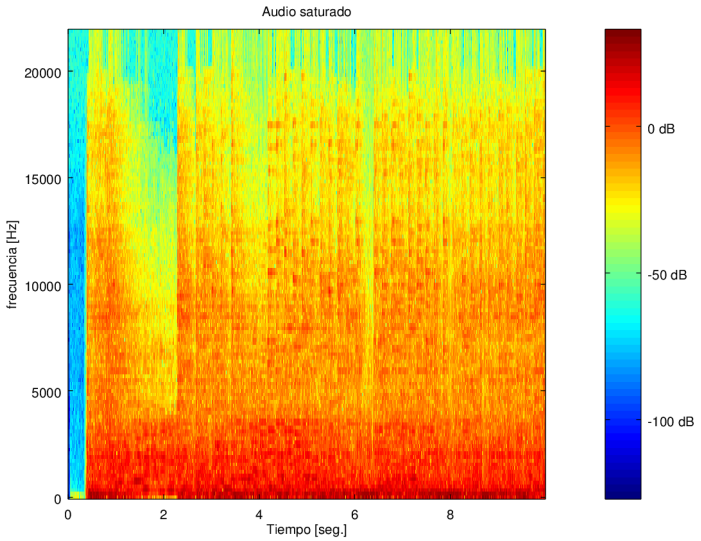

Figura 25: Espectrograma del fragmento de audio despues de saturar.

Para la distorsión por ecualización, se diseñó un filtro pasa altos con frecuencia de corte $F_c = 1000\,\text{Hz}$ y un orden de 200 para filtrar al fragmento de audio.

Esta distorsión es un sistema LTI. Ya que por linealidad, si $h(t)$ es el filtro en tiempo y $x(t)$ el fragmento de audio que vale 0 para $t < 0$ entonces la salida esta dada por:

$$y(t) = x(t)u(t) * h(t) = \int_{-\infty}^{\infty} x(t-\tau)u(t-\tau)h(\tau)d\tau = \int_{-\infty}^{t} x(t-\tau)h(\tau)d\tau \tag{8}$$

Como la salida esta representada por una relación lineal, una integral, se cumplirá la aditivdad y homogeneidad.

Para probar la invarianza en el tiempo. Se desplaza la señal de entrada en t_0:

$$y(t) = x(t-t_0)u(t-t_0) * h(t) = \int_{-\infty}^{\infty} x(t-\tau-t_0)u(t-\tau-t_0)h(\tau)d\tau = \int_{-\infty}^{t-t_0} x(t-\tau-t_0)h(\tau)d\tau \tag{9}$$

Reemplazando t por $t - t_0$ en la ecuación 8 se obtiene:

$$y(t-t_0) = \int_{-\infty}^{t-t_0} x(t-\tau-t_0)h(\tau)d\tau \tag{10}$$

Como las ecuaciones 9 y 10 son iguales queda demostrada la invarianza en el tiempo de esta distorsión.

Aplicando estas dos distorsiones se obtuvieron 13 aciertos de 50 evaluaciones considerando como acierto que la canción esté en el primer puesto. Considerando que la canción esté entre los primeros 5 se obtuvieron 50 aciertos de las 50 evaluaciones.

6. Conclusiones

La mayor densidad de potencia se encuentra en las bajas frecuencias de las pistas de audio, por lo que es posible realizar un submuestreo aplicando un filtro pasa bajos a la señal para reducir la cantidad de datos a analizar.

La elección de la ventana tiene una gran influencia en los filtros cuando se está limitado el orden de la ventana, para la ventana rectangular se encontraron lóbulos secundarios con amplitudes mayores a los lóbulos secundarios a una ventana Hamming.

Estas ventanas también son importantes para los espectrogramas, con ventanas rectangulares se detectó para un segmento de tiempo una dependencia con sus segmentos adyacentes, a comparación de con una ventana Hamming.

Se encontró una relación de dependencia entre la resolución de la frecuencia y la resolución en el tiempo en los espectrogramas, a medida que se usa ventanas mas grandes la resolución en el tiempo disminuye pero aumenta en frecuencia. Caso contrario, para ventanas mas chicas la resolución en el tiempo aumenta pero disminuye la resolución en frecuencia.

El algoritmo se mostró robusto ante distintos intervalos de audios a identificar si es que el audio a identificar es el mismo que el de la base de datos. Para audios a identificar con ruido, mientras mayor es la duración del audio mejor serán los resultados obtenidos.

No demostró ser muy eficiente ante distorsiones, aunque la canción a evaluar siempre se encontró entre los primeros 5 resultados, esto no permite concluir que el algoritmo tiene poco margen de error ya que habría que realizar la misma evaluación con una base de datos mas grande.

Se encontraron problemas para identificar canciones que fueron reproducidas en parlantes de baja calidad, como los de los celulares, ya que se comportan como un filtro pasa-altos y las frecuencias que mas influyen en la generación de la huella son las frecuencias bajas.

Si la base de datos esta compuesta por canciones del mismo genero y artista la tasa de aciertos disminuye drásticamente.

A. Codigo del proyecto

A.1. caracteristicas.m

```
%funcion que devuelve las caracteristicas del espectrograma
%Pre: E debe ser una matriz
%Post: Retorna una matriz E que contiene 1 o 0

function F = caracteristicas(E)

N = length(E(1,:));
M = length(E(:,1));

for m = 1:M-1
  for n = 2:N
    if( E(m,n) - E(m+1,n) > E(m,n-1) - E(m+1,n-1) )
      F(m,n-1) = 1;
    else
      F(m,n-1) = 0;
    endif
   endfor
endfor

endfunction
```

A.2. generar_huella_arch.m

```
function H = generar_huella_arch(archivo)
pkg load signal

[audio, Fs] = obtener_audio(archivo);
H = generar_huella(audio, Fs);

endfunction
```

A.3. generar_huella.m

```
function H = generar_huella(audio, Fs)

% Utilice un filtro FIR ya que tienen fase lineal como pide el enunciado.
% El maximo orden posible sin que el retardo supere los 1ms es 88.
Divisor = 8;
orden = 88; % retardo de 997.7 us
Fc = 1750;
flag  = 'scale';          % Sampling Flag

win = hamming(orden+1);
b  = fir1(orden, Fc/(Fs/2), 'low', win, flag);
y = filter(b, 1, audio);
audio_muestreado = y(1:Divisor:end);
Fs = Fs/Divisor;

N = 2048;
Noverlap = ceil(N-(Fs/25));
[S, F, T] = specgram(audio_muestreado, N, Fs, N, Noverlap);
S = abs(S).^2;

frec_log = logspace(log10(300),log10(2000),22);
S_log(1:21,1:length(T)) = 0;
F_log(1:21) = 0;
for i = 1:21
  x = find(F > frec_log(i) & F < frec_log(i+1), 1, 'first');
  F_log(i) = F(x);
  S_log(i,:) = S(x,:);
endfor


H = caracteristicas(S_log);

endfunction
```

A.4. generar_DB.m

```
% generar_DB.m
%
% Este script recorre todos los archivos de audio ubicados en la carpeta
% path_in, extrae las huellas digitales acusticas de cada uno y las va
% guardando en la base de datos DB. Adapte este script segun crea necesario
% para utilizar su implementacion de la funcion generar_huella().
%
% 66.74 Senales y Sistemas, 2do cuat 2015 - FIUBA


% Carpeta donde estan mis archivos de audio
path_in = '../Musica/';
files = dir([path_in,'/*.wav']);   % Estructura con info de cada archivo


%% Analisis de las canciones

% Incializo tabla hash
% Tamanio en memoria para hash de 20 bits: 4MB x N_columnas
DB.hash_nbits = 20;       % Cantidad de bits de los hashes
DB.n_entries = 20;        % Cantidad de columnas de la tabla
DB.ID_nbits = 12;         % Cantidad de bits para ID numerico de la cancion
DB.tabla = zeros(2^DB.hash_nbits,DB.n_entries,'uint32');

for k=1:length(files)
    file = files(k).name;             % Nombre del archivo de audio
    path_file = [path_in,file];       % Path completo al archivo

    % Muestro por consola el archivo que analizamos
    fprintf('%d de %d: %s\n',k,length(files),file);
    % Generamos la huella acustica H
    H = generar_huella_arch(path_file);

    % Guardamos la huella en la DB. A esta cancion se asigna el
    % identificador numerico k, el cual se guarda en la base de datos
    % La matriz H debera ser una matriz con elementos binarios de
    % hash_nbits filas
    DB = guardar_huella(uint32(k),H,DB);
end

% Guardamos la base de datos generada
save('DB_v01.mat','DB','files');
```

A.5. Ejercicios.m

```
pkg load signal

fig = 1;
%------------------------------ Ejercicio 1 ------------------------------

info = audioinfo('../Musica/haggard.wav');
audio_doble = audioread('../Musica/haggard.wav');

audio = mean(audio_doble,2);
N = 265;
intervalo = info.Duration/info.TotalSamples;
figure(fig++);
plot(0:intervalo:(N-1)*intervalo ,audio(1:N));
grid
xlabel ('tiempo [s]');
ylabel ('Canal de audio');
print('../Imagenes/Ej1_canal.png','-dpng');

% player = audioplayer(audio, info.SampleRate);
% play(player);

%------------------------------ Ejercicio 2 ------------------------------

N = 256;
[S, f, t] = specgram(audio, N, info.SampleRate, N, 0);
S = 10*log10(abs(S).^2);
S = mean(S',length(f(:,1)));
figure(fig++);
plot(f,S);
grid
xlabel('frecuencia [Hz]');
ylabel('Densidad espectral de potencia [dB]');
print('../Imagenes/Ej2_densidad_frecuencias.png','-dpng');


%------------------------------ Ejercicio 3 ------------------------------
% Utilice un filtro FIR ya que tienen fase
% lineal como pide el enunciado.
% El maximo orden posible sin que el retardo supere los 1ms es 88.
Divisor = 8;
Fs = info.SampleRate;
orden = 88; % retardo de 997.7 us
Fc = 1750;
flag  = 'scale';          % Sampling Flag

win = hamming(orden+1);
b  = fir1(orden, Fc/(Fs/2), 'low', win, flag);
y = filter(b, 1, audio);
audio_muestreado = y(1:Divisor:end);

% Diagrama de Polos y Ceros
figure(fig++);
zplane(b);
xlabel('Parte Real');
ylabel('Parte Imaginaria');
print('../Imagenes/Ej3_polos_y_ceros.png','-dpng');

% Respuesta en frecuencia
```

```
[H, W] = freqz(b);
W = W.*10;

H_abs = abs(H);
H_arg = unwrap(arg(H));

figure(fig++);
subplot(2,1,1);
plot(W,H_abs);
grid
xlabel('Frecuencia [KHz]');
ylabel('Modulo de la Respuesta en frecuencia');
axis([0 Fs/(2*1000) 0 1.2]);
subplot(2,1,2);
plot(W, H_arg);
grid
xlabel('Frecuencia [KHz]');
ylabel('Fase de la Respuesta en frecuencia');
axis([0 Fs/(2*1000) -20 0]);
print('../Imagenes/Ej3_respuesta_en_frecuencia.png','-dpng');

% Respuesta al impulso

[x, t] = impz(b);
t = (t ./ Fs)*1000; % t es el sample time, lo divido por Fs
                    % para obtener el tiempo y lo paso a mseg
figure(fig++);
stem(t,x);
grid
xlabel('Amplitud');
ylabel('Tiempo [ms]');
print('../Imagenes/Ej3_respuesta_al_impulso.png','-dpng');

% retardo de grupo

retardo_grupo = grpdelay(b) ./ Fs; % divido por Fs para pasarlo a tiempo
retardo_grupo = retardo_grupo .* 1000000; % lo paso a micro segundos

fprintf('retardo de grupo del filtro = %d\n',retardo_grupo(1));

figure(fig++);
plot(W, retardo_grupo);
grid
xlabel('Frecuencia [KHz]');
ylabel('Retardo de grupo [us]');
axis([0 20 997 998]);
print('../Imagenes/Ej3_retardo_de_grupo.png','-dpng');

% player = audioplayer(audio_muestreado, info.SampleRate/Divisor);
% play(player);

%------------------------------ Ejercicio 4 ------------------------------
% uso la ventana por defecto que es una hanning(N) y overlap de 0.5

% senial original
N = 256;
win = hamming(N);
[S1, F1, T1] = specgram(audio, N, Fs, win);
S1 = abs(S1).^2;
figure(fig++);
```

```
imagesc(T1, F1, 10*log10(S1)), axis xy
h = colorbar('EastOutside');
ytick = get(h, 'ytick');
set(h, 'yticklabel', sprintf('%g dB|', ytick));
ylabel('frecuencia [Hz]');
xlabel('Tiempo [seg.]');
print('../Imagenes/Ej4_espectrograma_original.png','-dpng');

% Senial filtrada
[S2, F2, T2] = specgram(y, N, Fs);
S2 = abs(S2).^2;
figure(fig++);
imagesc(T2, F2, 10*log10(S2)), axis xy
h = colorbar('EastOutside');
ytick = get(h, 'ytick');
set(h, 'yticklabel', sprintf('%g dB|', ytick));
ylabel('frecuencia [Hz]');
xlabel('Tiempo [seg.]');
print('../Imagenes/Ej4_espectrograma_filtrado.png','-dpng');

%------------------------------ Ejercicio 5 ------------------------------
% hago las 2 ventanas con el mismo orden que el filtro diseniado
% orden = 88
Fs = 44100;
Fc = 1750;
N = 25;
M = floor((Fs*N)/Fc);
f = 0:Fs/M:Fs;

% ventana rectangular
win_rec = rectwin(N+1);
fft_rec = fft(win_rec, M+1);
fft_rec = abs(fft_rec).^2;
fft_rec = fft_rec ./ fft_rec(1);
fft_rec = 10*log10(fft_rec);

% ventana hamming
win_ham = hamming(N+1);
fft_ham = fft(win_ham, M+1);
fft_ham = abs(fft_ham).^2;
fft_ham = fft_ham ./ fft_ham(1);
fft_ham = 10*log10(fft_ham);

figure(fig++);
hold on
plot(f, fft_rec, 'b-');
plot(f, fft_ham, 'r-');
grid
xlabel('Frecuencia [Hz]');
ylabel('Densidad espectral de potencia [dB]');
legend('rectangular de orden 25','hamming de orden 25')
axis([0 Fs -100 0]);
print('../Imagenes/Ej5_comparacion_ventanas.png','-dpng');

%------------------------------ Ejercicio 6 ------------------------------
Divisor = 8;
Fs = info.SampleRate/Divisor;
N1 = 64;
N2 = 4096;
N3 = 8192;
```

```
% Ventana rectangular
%N1
win = rectwin(N1);
[S, F, T] = specgram(audio_muestreado, N1, Fs, win);
S = abs(S).^2;
Ti = find(T>=60,1,'first');
Tf = find(T>=80,1,'first');
figure(fig++);
imagesc(T(Ti:Tf), F, 10*log10(S(:,Ti:Tf))), axis xy
h = colorbar('EastOutside');
ytick = get(h, 'ytick');
set(h, 'yticklabel', sprintf('%g dB|', ytick));
ylabel('frecuencia [Hz]');
xlabel('Tiempo [seg.]');
title('Ventana rectangular de longitud 64');
print('../Imagenes/Ej6_espetrograma_rectangular_N=64.png','-dpng');


% con mas zoom
Fi = find(F >= 1500, 1, 'first');
figure(fig++);
imagesc(T(Ti:Tf), F(Fi:end), 10*log10(S(Fi:end,Ti:Tf))), axis xy
h = colorbar('EastOutside');
ytick = get(h, 'ytick');
set(h, 'yticklabel', sprintf('%g dB|', ytick));
ylabel('frecuencia [Hz]');
xlabel('Tiempo [seg.]');
title('Ventana rectangular de longitud 64');
print('../Imagenes/Ej6_espetrograma_rectangular_zoom.png','-dpng');



%N2
win = rectwin(N2);
[S, F, T] = specgram(audio_muestreado, N2, Fs, win);
S = abs(S).^2;
Ti = find(T>=60,1,'first');
Tf = find(T>=80,1,'first');
figure(fig++);
imagesc(T(Ti:Tf), F, 10*log10(S(:,Ti:Tf))), axis xy
h = colorbar('EastOutside');
ytick = get(h, 'ytick');
set(h, 'yticklabel', sprintf('%g dB|', ytick));
ylabel('frecuencia [Hz]');
xlabel('Tiempo [seg.]');
title('Ventana rectangular de longitud 4096');
print('../Imagenes/Ej6_espetrograma_rectangular_N=4096.png','-dpng');


%N3
win = rectwin(N3);
[S, F, T] = specgram(audio_muestreado, N3, Fs, win);
S = abs(S).^2;
Ti = find(T>=60,1,'first');
Tf = find(T>=80,1,'first');
figure(fig++);
imagesc(T(Ti:Tf), F, 10*log10(S(:,Ti:Tf))), axis xy
h = colorbar('EastOutside');
ytick = get(h, 'ytick');
```

```
set(h, 'yticklabel', sprintf('%g dB|', ytick));
ylabel('frecuencia [Hz]');
xlabel('Tiempo [seg.]');
title('Ventana rectangular de longitud 8192');
print('../Imagenes/Ej6_espetrograma_rectangular_N=8192.png','-dpng');

% Ventana hamming

%N1
win = hamming(N1);
[S, F, T] = specgram(audio_muestreado, N1, Fs, win);
S = abs(S).^2;
Ti = find(T>=60,1,'first');
Tf = find(T>=80,1,'first');
figure(fig++);
imagesc(T(Ti:Tf), F, 10*log10(S(:,Ti:Tf))), axis xy
h = colorbar('EastOutside');
ytick = get(h, 'ytick');
set(h, 'yticklabel', sprintf('%g dB|', ytick));
ylabel('frecuencia [Hz]');
xlabel('Tiempo [seg.]');
title('Ventana hamming de longitud 64');
print('../Imagenes/Ej6_espetrograma_hamming_N=64.png','-dpng');

% con mas zoom

Fi = find(F>=1500, 1, 'first');
figure(fig++);
imagesc(T(Ti:Tf), F(Fi:end), 10*log10(S(Fi:end,Ti:Tf))), axis xy
h = colorbar('EastOutside');
ytick = get(h, 'ytick');
set(h, 'yticklabel', sprintf('%g dB|', ytick));
ylabel('frecuencia [Hz]');
xlabel('Tiempo [seg.]');
title('Ventana hamming de longitud 64');
print('../Imagenes/Ej6_espetrograma_hamming_zoom.png','-dpng');


%N2
win = hamming(N2);
[S, F, T] = specgram(audio_muestreado, N2, Fs, win);
S = abs(S).^2;
Ti = find(T>=60,1,'first');
Tf = find(T>=80,1,'first');
figure(fig++);
imagesc(T(Ti:Tf), F, 10*log10(S(:,Ti:Tf))), axis xy
h = colorbar('EastOutside');
ytick = get(h, 'ytick');
set(h, 'yticklabel', sprintf('%g dB|', ytick));
ylabel('frecuencia [Hz]');
xlabel('Tiempo [seg.]');
title('Ventana hamming de longitud 4096');
print('../Imagenes/Ej6_espetrograma_hamming_N=4096.png','-dpng');


%N3
win = hamming(N3);
[S, F, T] = specgram(audio_muestreado, N3, Fs, win);
S = abs(S).^2;
Ti = find(T>=60,1,'first');
```

```
Tf = find(T>=80,1,'first');
figure(fig++);
imagesc(T(Ti:Tf), F, 10*log10(S(:,Ti:Tf))), axis xy
h = colorbar('EastOutside');
ytick = get(h, 'ytick');
set(h, 'yticklabel', sprintf('%g dB|', ytick));
ylabel('frecuencia [Hz]');
xlabel('Tiempo [seg.]');
title('Ventana hamming de longitud 8192');
print('../Imagenes/Ej6_espetrograma_hamming_N=8192.png','-dpng');

%------------------------------ Ejercicio 7 -------------------------------

H = generar_huella(audio,info.SampleRate);
figure(fig++);
imagesc(H(:,1:20));
colormap(gray());
ylabel('Banda');
xlabel('Frame');
print('../Imagenes/Ej7_huella.png','-dpng');
```

A.6. rendimiento.m

```
fprintf('Tests de segmentos de audio:\n');
tests([5 10 15], false,0,false,false,false);
fprintf('Tests con ruido de 0dB\n');
tests([5 10 15], true,0,false,false,false);
fprintf('Tests con ruido de 10dB\n');
tests([5 10 15], true,10,false,false,false);
fprintf('Tests con ruido de 20dB\n');
tests([5 10 15], true,20,false,false,false);
fprintf('Tests con distorsion saturacion y ecualizacion\n');
tests(10, false,0,true,true,false);
saturacion;
```

A.7. tests.m

```
function tests(T, agregar_ruido, SNR, saturar, ecualizar ,ralentizar)

pkg load signal
if (exist('DB_v01.mat') != 2)
  generar_DB;
endif;
load DB_v01

%Duracion de los segmentos de cada prueba
evaluaciones = 50;

aciertos1 = zeros(length(T),1);
aciertos5 = zeros(length(T),1);

for t=1:length(T)
  for i = 1: evaluaciones

    n = floor(rand*length(files)+1);
    archivo = strcat('../Musica/',files(n).name);
    [audio, Fs] = obtener_audio(archivo);
    segmento = Fs*T(t);
    inicio = floor(rand*(length(audio)-segmento+1));
    fin = inicio+segmento;
    audio = audio(inicio:fin);

    if (agregar_ruido)
      Px = var(audio);
      divisor = 10^(SNR/10);
      Pn = sqrt(Px/divisor)*randn(length(audio),1);
      audio = audio+Pn;
    endif

    if (saturar)
      %saturacion
      sat = sqrt(var(audio))*1.5;
      for x = 1:length(audio)
        if (audio(x) > sat)
          audio(x) = sat;
        elseif(audio(x) < -sat)
          audio(x) = -sat;
        endif
      endfor
    endif
```

```
    if (ecualizar)
      % pasa altos
      orden = 200; % retardo de 997.7 us
      Fc = 1000;
      flag  = 'scale';          % Sampling Flag

      win = hamming(orden+1);
      b  = fir1(orden, Fc/(Fs/2), 'high', win, flag);
      audio = filter(b, 1, audio);
    endif

    if (ralentizar)
     % desacelera
      [audio, h] = resample(audio, Fs*0.99,Fs);
    endif

    H = generar_huella(audio,Fs);
    resultado = query_DB(DB,H);

    if (resultado(1) == n)
      aciertos1(t) = 1 + aciertos1(t);
    endif

    if (length(find(resultado,n)) != 0)
      aciertos5(t) = 1 + aciertos5(t);
    endif

  endfor;
endfor;

for i = 1:length(T)
  fprintf('Para T=%d\n',T(i));
  fprintf('Aciertos de primer puesto: %d\n', aciertos1(i));
  fprintf('Aciertos de primeros 5 puesto : %d\n', aciertos5(i));
endfor

endfunction
```

A.8. saturacion.m

```
[audio, Fs] = obtener_audio('../Musica/haggard.wav');

audio = audio(1:10*Fs);

N = 256;
[S, F, T] = specgram(audio, N, Fs);
S = abs(S).^2;
figure;
imagesc(T, F, 10*log10(S)), axis xy
h = colorbar('EastOutside');
ytick = get(h, 'ytick');
set(h, 'yticklabel', sprintf('%g dB|', ytick));
ylabel('frecuencia [Hz]');
xlabel('Tiempo [seg.]');
title('Audio original');
print('../Imagenes/Ej14_espectrograma_audio_original.png','-dpng');

sat = sqrt(var(audio))*1.5;
for x = 1:length(audio)
  if (audio(x) > sat)
    audio(x) = sat;
  elseif(audio(x) < -sat)
    audio(x) = -sat;
  endif
endfor

N = 256;
[S, F, T] = specgram(audio, N, Fs);
S = abs(S).^2;
figure;
imagesc(T, F, 10*log10(S)), axis xy
h = colorbar('EastOutside');
ytick = get(h, 'ytick');
set(h, 'yticklabel', sprintf('%g dB|', ytick));
ylabel('frecuencia [Hz]');
xlabel('Tiempo [seg.]');
title('Audio saturado');
print('../Imagenes/Ej14_espectrograma_audio_saturado.png','-dpng');
```

A.9. identificar_cancion.m

```
function identificar_cancion(archivo)

pkg load signal
if (exist('DB_v01.mat') != 2)
  generar_DB;
endif;
load DB_v01;

H = generar_huella_arch(archivo);
resultados = query_DB(DB,H);

fprintf('Resultados:\n');

for x = 1:length(resultados)
  cancion = files(resultados(x)).name;
  info = audioinfo(strcat('../Musica/',cancion));
  fprintf('%d -\n',x);
  fprintf('Cancion: %s\n', info.Title);
  fprintf('Artista: %s\n', info.Artist);
  fprintf('Duracion: %d\n\n', info.Duration);

endfor



endfunction;
```

A.10. query_DB.m

```
function [ID,MATCHES] = query_DB(DB, huella)
pkg load communications
%[ID,MATCHES] = query_DB(DB,HUELLA)
%
%Hace un query a la base de datos para obtener el ID de las canciones
%coincidentes con la HUELLA. Devuelve los 5 primeros resultados que mejor
%coinciden, ordenados en orden de prioridad descendente.
%
%Inputs:
%  DB: estructura con los siguientes campos
%      - hash_nbits: numero de bits para el hash
%      - n_entries: numero de columnas de la tabla
%      - ID_nbits: numero de bits para guardar ID numerico de la cancion
%      - tabla: tabla hash de 2^hash_nbits filas x n_entries columnas
%  HUELLA: huella acustica en forma de matriz binaria (1 y 0)
%
%Outputs:
%  ID: identificador numerico de los primeros 5 resultados que matchean
%  MATCHES: numero de matches que tuvo cada ID al realizar el query
%
%Nota:
%  Si no encuentra coincidencias, devuelve: ID=0, MATCHES=0
%
%66.74 Senales y Sistemas, 2do cuat 2015 - FIUBA


%Obtenemos las filas a guardar en la tabla, pasando las caracteristicas de
%binario a decimal. Sumo 1 porque Matlab indexa desde 1
hash = bi2de(huella.') + 1;

%Extraemos los elementos de la tabla que corresponden a los hashes dados
vals = DB.tabla(hash,:);
vals = vals(vals~=0);
if isempty(vals)
    %Si los elementos fueron todos nulos, devuelve cero
    ID = 0; MATCHES = 0; return;
end

%Extraemos de cada elemento el ID numerico y su frame
ID1 = mod(vals,2^DB.ID_nbits);
frames = floor(vals/2^DB.ID_nbits);


%Filtramos por tiempo: para cada ID, se cuenta el numero de coincidencias
%dentro del intervalo de duracion temporal de la huella (i.e. frame_span)
frame_span = size(huella,2);
ID = unique(ID1);
MATCHES = zeros(size(ID));
for k=1:length(ID)
    frame_aux = frames(ID1==ID(k));
    matches = 0;
    for k2=1:length(frame_aux)
        %match_aux: numero de matches en intervalo frame_span
        match_aux = nnz((frame_aux>=frame_aux(k2)) & (frame_aux<=frame_aux(k2)+frame_spa
        if match_aux>matches
            matches = match_aux;
        end
    end
```

```
    MATCHES(k) = matches;
end

%Ordeno ID y MATCHES por orden descendente
[MATCHES, idx] = sort(MATCHES, 'descend');
ID = ID(idx);

%Me quedo con los primeros Nmax elementos
Nmax = 5;
N = min([Nmax, size(ID,1)]);
MATCHES = MATCHES(1:N);
ID = ID(1:N);

end
```

A.11. obtener_audio.m

```
function [audio, Fs] = obtener_audio(archivo)

info = audioinfo(archivo);
Fs = info.SampleRate;

audio = audioread(archivo);
if (length(audio(1,:) > 1))
  audio = mean(audio,2);
endif


endfunction
```

A.12. guardar_huella.m

```
function DB = guardar_huella(ID,huella,DB)
%DB = guardar_huella(ID,HUELLA,DB)
%
%Guarda la huella digital acustica HUELLA en la tabla hash de la estructura
%DB, identificando la cancion con el identificador numerico ID.
%
%Inputs:
%  ID: identificador numerico del tema que corresponde a la huella
%  HUELLA: huella acustica en forma de matriz binaria (unos y ceros)
%  DB: estructura con los siguientes campos
%      - hash_nbits: numero de bits para el hash
%      - n_entries: numero de columnas de la tabla
%      - ID_nbits: numero de bits para guardar ID numerico de la cancion
%      - tabla: tabla hash de 2^hash_nbits filas x n_entries columnas
%
%Outputs:
%  DB: estructura con tabla hash actualizada
%
%66.74 Senales y Sistemas, 2do cuat 2015 - FIUBA

pkg load communications

%Verificamos que HUELLA sea una matriz binaria
if any(huella(:)~=0 & huella(:)~=1)
    error('La matriz HUELLA contiene elementos no binarios')
end

%Verificamos que HUELLA tenga hash_nbits filas
if size(huella,1) ~= DB.hash_nbits
    error('La matriz HUELLA tiene %d filas en lugar de %d',size(huella,1),DB.hash_nbits)
end

%Verificamos que ID sea un entero mayor o igual a 1
if ID<1 || ~isinteger(ID)
    error('El identificador ID debe ser un entero mayor o igual a 1')
end

%Generamos el elemento VAL a guardar, concatenando los bits del ID con los
%bits del tiempo de cada frame
frames_nbits = 32-DB.ID_nbits;
frames = mod(1:size(huella,2),2^frames_nbits);
val = uint32(ID + 2^DB.ID_nbits*frames);

%Obtenemos las filas a guardar en la tabla, pasando las caracteristicas de
%binario a decimal. Sumo 1 porque Matlab indexa desde 1
hash = bi2de(huella.') + 1;

%Primero grabamos en las filas en que queda espacio
fila_ok = DB.tabla(hash,end)==0;   %Filas que tienen espacio al final
hash_ok = hash(fila_ok);
DB.tabla(hash_ok+size(DB.tabla,1)*(sum(DB.tabla(hash_ok,:)~=0,2)+1)) = val(fila_ok);

%Finalmente grabamos en las filas que estan llenas, pisando algun
%valor anterior al azar (colision)
hash_col = hash(~fila_ok);
idx_rand = ceil(rand*DB.n_entries);
DB.tabla(hash_col,idx_rand) = val(~fila_ok);
```

end