Identificación de canciones a través de la transformada rápida de Fourier

Gilberto Juárez Rangel

Fecha: 23 de noviembre de 2020

Índice

Resumen

Para la identificación de canciones se uso de la transformada rápida de Fourier para hacer comparación entre picos frecuencias con una tolerancia de 2 hz, se detectaron los 15 picos más altos en un rango de 500 a 2500 hz, evitando los hz más abajo debido a los problemas que tuvimos con el micrófono y porque gran porcentaje de las canciones tienen mucha coincidencia dentro de ese rango, de mismo modo no se utilizaron los hz después del 2500 ya que eran sonidos muy agudos y no eran muy significativos, de mismo modo esto generaría un proceso más desarrollado por lo que sería más tardado y por ende menos eficiente con respecto al tiempo.

El código hecho en *Matlab* se explicara paso a paso para su total entendimiento, se genera una base de datos con la información de los picos de cada canción en forma de matriz y esta es comparada con un vector de picos generado a través del audio recibido por el micrófono.

1. Introducción

En el mundo existen millones de canciones y cada día se pueden escuchar una o más canciones nuevas que provienen de distintos dispositivos. Es común que dicha canción no haya sido nombrada o sea escuchada después de que se nombró, causando una duda sobre cómo se llamaba la canción para así poder agregarla en nuestra lista. Este problema surge en cada parte del mundo en distintos idiomas, géneros, bandas etc. por lo cuál es importante para muchos usuarios una aplicación o dispositivo que detecte canciones. En la actualidad ya existen aplicaciones como *Shazam* o *SoundHound* que ya cumplen dicha función. Es importante demostrar como funcionan las bases de dichas aplicaciones y poder hacer nuestro propio programa. Esta solución que se presenta es sólo una simplificación de lo que hace *Shazam* y *SoundHound*.

No obstante es importante demostrar que la transformada rápida de Fourier o FFT, es útil en otros aspectos, cómo reconocimientos de voz para seguridad, detector de terremotos, traductores, escritura manos libres entre otras cosas. Esta aplicación que le daremos es una de muchas otras.

2. Marco Teórico

La transformada de Fourier[Morin, 2009] esta definida cómo:

$$\hat{f}(w) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-iwx} dx \tag{1}$$

Siendo w la frecuencia a analizar y la función $f(x)$ es la función original, lamentablemente en la práctica normalmente no es posible utilizar la transformada de Fourier continua dado a que nuestros datos son discretos, no continuos. Para ello se obtiene la transformada discreta de Fourier[Feregrino and Morales, 2015] descrita cómo:

$$\hat{f}_n = \sum_{k=0}^{N-1} f_k e^{-inx_k}, \quad n = 0, 1, 2, ..., N-1 \tag{2}$$

Donde n es la frecuencia a analizar, N el número de datos, f_k es tu vector de datos y k define el lugar del dato es decir:

$$f_k = [f_0, f_1, f_2, ...f_{N-1}]$$

$$k = 0, 1, 2, ..., N-1$$

Dado a que la transformada toma en cuenta un parámetro de $[0, 2\pi]$ podemos definir que cada valor de x_k es el siguiente:

$$x_k = \frac{2\pi}{N} k$$

Sería posible trabajar con esto sin embargo el tiempo y trabajo de la computadora al usar datos relativamente grandes son masivos, al hacer muestras de sonido se obtienen 44100 datos cada segundo por lo tanto el sólo hecho de calcular la transformada discreta de Fourier llevaría acabo 44100^2 de operaciones matemáticas en sólo 1 seg, sin embargo, tenemos la fortuna de que exista la Transformada Rápida de Fourier[ELLIOTT, 1987] que a través de un algoritmo matemático se puede reducir la cantidad de operaciones en gran escala.

3. Desarrollo

3.1. Señales de Audio

Antes de empezar a hacer el análisis de audio, es necesario entender que son estas señales que se reciben. Inicialmente tenemos dos factores de nuestros archivos que son las señales, y la cantidad de señales por segundo. Cada señal representa un valor entre -1 y 1, esta representa la amplitud que se percibe, un valor cerca del -1 o 1 indica un ruido fuerte mientras que un valor cerca del 0 indica silencio. Seguido de eso tenemos un valor constante indicando el número de señales por segundo, es necesario que este sea alto para que la calidad del audio no se pierda, en un disco CD esta frecuencia es de 44100 señales por segundo.
Dado a que tenemos una señal cada segundo podemos representar dicha información de la siguiente manera:

$$f(t) = [f_0, f_1, f_2, ...f_{N-1}]$$
$$t = [\frac{0}{44100}, \frac{1}{44100}, \frac{2}{44100}, ..., \frac{N-1}{44100}]$$
$$\text{donde } -1 \leq f_t \leq 1$$

De esta manera se pueden obtener las amplitudes con su respectivo tiempo, una vez que se tienen estos datos es posible hacer uso de la FFT, para ello hay que entender qué representa la FFT en una señal de audio. La FFT define el peso que tiene cada frecuencia en una señal de audio. Al hacer gráficas sobre ello podemos notar picos en las frecuencias, estas se pueden interpretar como el espectro en la señal de audio, dado a que una canción tiene un ritmo o repeticiones en notas no es necesario escuchar la canción completa. Al momento de obtener una muestreo de datos la FFT mostrará relación con las *"huellas"* que contiene la canción base, esto debido a que esas *"huellas"* son aquellas frecuencias que se repiten constantemente dado el ritmo de la canción.

3.2. Elaboración en Matlab

3.2.1. Base de Datos

Dado a que $f(t)$ y t son dos vectores relacionados y del mismo tamaño estos se pueden gráficar para demostrar como se observa la señal de sonido.

```
1 Datos = {'DrumgoDum.wav';'Filter.wav';'ReasonsUnknown.wav';'More.wav';'
   YoNoMeLlamoJavier.wav';'DreamyNight.wav';'NadaPersonal.wav'};
2 [y,fs] = audioread(string(Datos(1)));
3 t = linspace(0,length(y)/fs,length(y));
4 plot(t,y)
5
```

Listing 1: Obtención de señales de una canción en *Matlab*

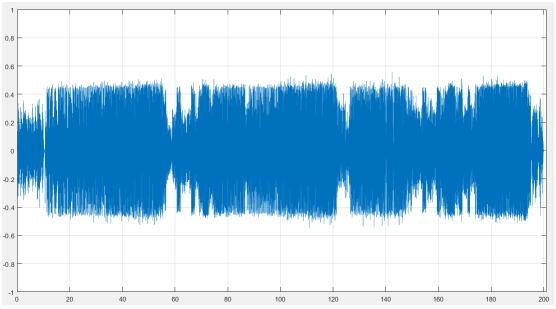

Figura 1: Gráfica de Sonido 200s

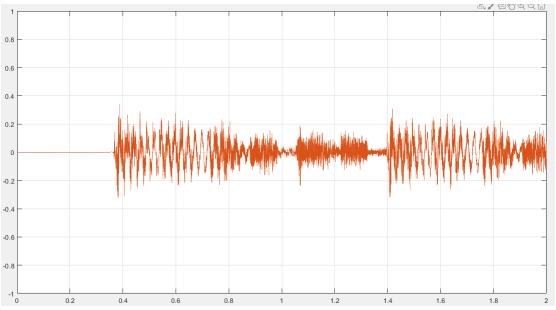

Figura 2: Gráfica de Sonido 2s

Dado a que ya se pueden entender los datos ahora estos ya son analizables, cómo se mencionó en la Metodología, nuestros datos no son ni van a ser continuos, estos serán discretos en todo momento, por lo tanto es necesaria la transformada discreta de Fourier(DFT), o para mejor optimización del programa la transformada rápida de Fourier(FFT). El uso de la FFT es más sencillo a través de *Matlab* dado a que dicha función ya existe, sólo es necesario establecerla en *Matlab* con los parámetros correctos.

```
FFourierT = 0;
for i = 1:floor(length(y)/fs)
  k = abs(fft(y((1+(i-1)*44100):i*44100),fs));
  FFourierT = k+FFourierT;
end

```

Listing 2: Implementación de fft() en *Matlab*

En el código 2 podemos notar distintas variables, inicialmente tenemos *FFourierT* es una variable de almacenamiento, aquí vamos a almacenar el vector de datos que contiene la FFT, nuestro vector k va a generar la FFT de cada segundo de la canción, es innecesario hacer una FFT de la canción completa de manera directa dado a que frecuencias encima de 20 khz no son percibidas por el oído humano por lo que es más sencillo optar por cada segundo de la canción y sumar cada resultado de la FFT en otro vector (FFourierT). Para simplificar el

problema de la FFT es altamente recomendable obtener los valores absolutos. de este.

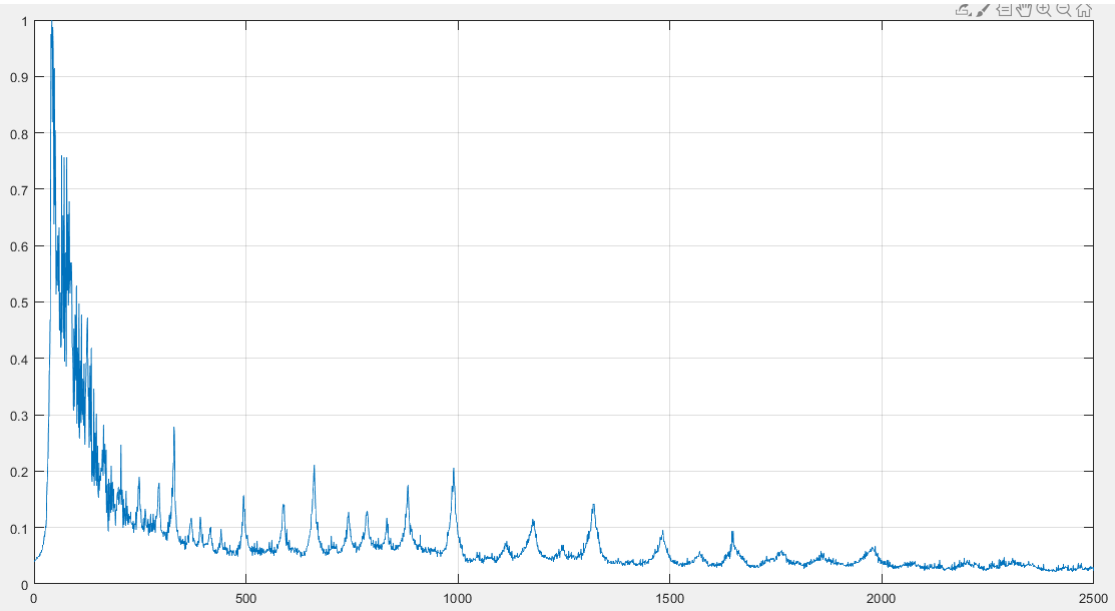

Figura 3: FFT de 1 a 2500 hz (Normalizada)

Dado a que gran parte de las canciones tiene una concentración muy alta en el rango de frecuencias desde 0 hasta 500 hz esta se va a ignorar en nuestro análisis ya que nos causaría mucha confusión al analizar los datos. Esto se debe a que dentro de esas frecuencias se encuentran gran parte de las canciones, esta zona nos podría generar mucho ruido al momento de hacer el análisis. Además de eso yo específicamente tuve problemas para que mi micrófono identificara dichas frecuencias por lo que estas se ignoran. Podemos notar que hay picos muy específicos en ciertas frecuencias, esto nos podría funcionar como guía para identificar entre una canción y otra, para hacer esto más notorio se va a gráficar la FFT encima de la gráfica anterior con sólo 10 segundos de la misma canción recibida por el micrófono. Dicho proceso se explicará más adelante.

Como podemos notar después de la frecuencia 500 existe una alta relación entre la gráfica original contra la de los 10 seg grabados por mi micrófono. El motivo de esto se debe a que en esas frecuencias están las voces, violines, piano, solos de guitarra etc.[Reyes Alonzo, 2013].Esto quiere decir que podemos enfocarnos precisamente en dichos picos para obtener una relación entre. Esta será la guía para identificar las canciones, la pregunta a resolver es la siguiente,

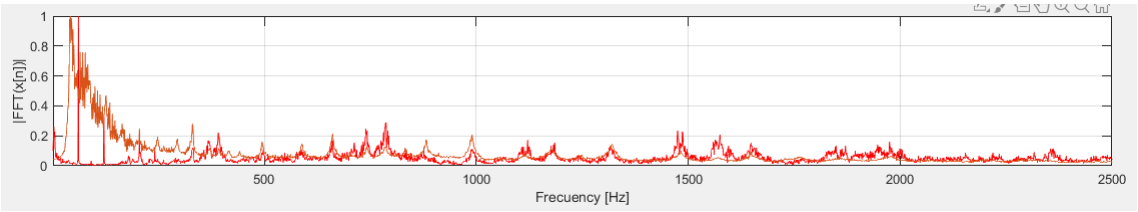

Figura 4: Comparación con 10 seg grabados

¿Cómo identifico los picos en un vector de puntos? Esta pregunta se puede resolver de la siguiente manera.

Primeramente hay que ignorar los datos debajo de la frecuencia 400, e incluir hasta la frecuencia 2500. Para esto se utiliza la siguiente línea de código.

```
1    Final(500:2500)
```

Listing 3: Abstracción de Datos

Una vez obtenido esto puede uno organizar los datos de mayor a menor para así poder registrar los picos, *Matlab* tiene una función *"sort"*que te permite ordenarlos de mayor a menor sin perder la ubicación en donde que se encuentra es decir si se tiene un vector de puntos:

$$Final = [0, 9, 12, 7, 8, 2, 31, 25, 6, 10]$$

Regresa dos vectores nombrados al inicio:

$$[val \;\; ind] = sort(Final, \text{'descend'})$$

$$val = [31, 25, 12, 10, 9, 8, 7, 6, 2, 0]$$

$$ind = [7, 8, 3, 10, 2, 5, 4, 9, 6, 1]$$

A nosotros lo que nos interesa son las posiciones (ind) ya que estas contienen las frecuencias en la que se encuentran, dado a que la frecuencia inicia desde 500 y termina hasta 2500 el indice 1 corresponde al valor de frecuencia 500, el 2 al 501 y así respectivamente. No es necesario realizar dicha suma ya que este proceso se repetirá al analizar el audio extraído del micrófono.
Una vez hecho esto se obtienen los índices de mayor a menor, sin embargo se obtiene el siguiente problema. Al tener este arreglo de datos si existe un pico lo suficientemente alto o con forma más ancha se pueden obtener datos del mismo pico de manera cercana, como ejemplo se mostraran los resultados del audio de la canción "Drum go Dum".

$$ind = [262, 591, 261, 590, 263, 483, 592, 593, 587, 260, 588, 482, 259, 589, 594]$$

Como podemos notar tenemos muchos valores similares, esto nos genera un problema dado que no es necesaria información relativamente igual. Para esto hacemos un doble loop de *for* que nos identifique entre cada elemento, y si este elemento tiene una diferencia menor o igual a 5 con los siguientes 100 datos hará que estos sean 0. Este proceso hará que los picos registrados tengan un radio de 5. Como nuevo resultado obtenemos:

$$ind = [262, 591, 483, 95, 920, 189, 585, 386, 343, 256, 477, 433, 597, 777, 270]$$

La forma final en el que se obtendría este llamado índice según nuestro código sería el siguiente:

```
1  [val ind1] = sort(Final(400:Freq),'descend');
2    for i = 1:100
3      for k = i+1:i+100
4        if abs(ind1(i)-ind1(k)) <= 5
5          ind1(k) = [0];
6        end
7      end
8    end
9  ind1(ind1==0)=[];
10 B(D,:)= ind1(1:15);
11
12
```

Listing 4: Abstracción

Las línea 9 en el código elimina los 0's de la matriz y la última línea registra los primeros 15 datos por filas en una matriz B de tamaño 7×15

Una vez obtenida la matriz B es necesario guardar esta matriz como un archivo diferente, para ello se utiliza el siguiente comando:

```
1    save('Canciones','B')
2
```

Listing 5: Guardar matriz como archivo

El primer elemento es el nombre del archivo en el que se va a guardar y el siguiente sería la matriz a guardar.

3.2.2. Identificador de canciones

Una vez creada nuestra base de datos, podemos empezar a buscar relación con estás canciones, la repetición de pasos en el código no se demostrará dado a que fueron explicados

previamente.

Para empezar nuestra identificación de canciones tenemos que subir nuestra base de datos ya que esta será la base a comparar para identificar entre una canción y otra. Esta operación se declara de la siguiente forma:

```
1    load('Songs.mat','B')
2    load('SongsName.mat','Datos')
3
```

Listing 6: Subir base de datos

El primer valor B indica la matriz de frecuencias a identificar de cada canción, mientras que el segundo valor $Datos$ es nuestro vector de canciones.

Una vez declarada esta función podemos empezar el proceso de identificar canciones. Para ello se declaran los parámetros para grabar el audio, es importante que estos concuerden con los que se obtuvieron de las canciones en la base de datos, es decir que haya 44100 puntos por segundo, se guíe en un canal, y que maneje 16 bits. Estos valores se utilizaran en la función *audiorecorder*. Así como se explico en la sección 3.2.1 es innecesario hacer la FFT de una grabación completa, es más eficiente hacerlo en fragmentos de 1 seg. Por ende vamos a establecer estos valores desde un inició.

```
1    %parametros
2    F1=44100;
3    bits=16;
4    channels = 1;
5    audio = audiorecorder(F1,bits,channels);
6    audiolength = 1;%seconds
7    fragments = 5;
8
```

Listing 7: Parametros de audio

Una vez obtenidos los parámetros empezamos la grabación dentro de un for loop. La grabación se declara a través de la línea 3, y se recuperan los datos en la variable *aud* de las amplitudes en un vector. La FFT se obtiene de la misma manera que se realizo en la sección 3.2.1:

```
1  for k= 1:fragments
2    % obtencion del audio
3    recordblocking(audio,audiolength);
4    aud = getaudiodata(audio);
5    % transformada rapida de Fourier
6    Segg = abs(fft(aud));
7    auf = auf + Segg;
```

```
8   end
9
```

Listing 8: Grabación y recopilación de datos

Con la obtención de este vector se obtienen los 15 picos más altos con la misma metodología utilizada en la sección 3.2.1:

```
1     [val ,ind] = sort(auf(500:Freq),'descend');
2     for i = 1:100
3       for k = i+1:i+100
4         if abs(ind(i)-ind(k)) <= 5
5           ind(k) = [0];
6         end
7       end
8     end
9     ind(ind == 0) = [];
10
```

Listing 9: Obtención de picos de mayor a menor

Una vez que tenemos este vector *ind* podemos empezar a hacer nuestra comparación entre canciones. Teniendo la matriz B como base de datos de cada canción y el vector *ind* con la información de nuestra grabación.

Al momento de realizar gráficas podemos notar picos similares pero estos varían en dos cosas, en altura y ubicación, el proceso llevado a cabo toma en cuenta los siguientes factores, dado que las alturas pueden cambiar el orden de los indices pueden no ser los mismos, sin embargo se espera que haya alguna relación entre estos. Para ello se realizaran restas de cada valor con toda la fila para ver si existe alguna similitud. Dado a que difícilmente se encontrara un pico exactamente en la misma frecuencia se tendrá un margen de error de 2 unidades. Este margen se escoge para evitar que un sólo valor tenga doble relación al hacer la resta con cada elemento. Dicha demostración se hace de la siguiente manera:

Supongamos un valor x, se tomara en cuenta una relación si existe una resta menor o igual a 2 es decir:

$$(x-2) \leq x \leq (x+2)$$

por ende $(x-2)$ y $(x+2)$ son los máximos valores que encuentran relación, sin embargo la diferencia entre estos 2 valores es 4 y debido a lo establecido en la sección 3.2.1 estos dos valores no existirían en la base de datos dado que su diferencia es menor a 5, por ende no se encontraría doble relación.

El proceso simplificado y hecho en *Matlab* se realiza de la siguiente manera:

```
cont = 0; %Contador de smilitud
C = []; %Vector de relaciones
for L = 1:length(Datos)
  for i = 1:15
    for k = 1:15
      if abs(ind(i)-B(L,k)) < 3
        cont = cont + 1;
      end
    end

  end
  C(L) = cont;
  cont = 0;
end
```

Listing 10: Relación con cada canción

Como podemos notar tenemos un contador que registra el número de relaciones encontradas en cada fila, este valor se guarda en un vector C el cuál indica su posición dado el número de cada canción a la que se hizo la relación, dejándonos un vector:

$$C = [C_1, C_2, C_3, ..., C_N]$$

Una vez obtenido este vector lo siguiente es sencillo, se extrae la posición en donde se encuentra el valor máximo del vector C utilizando el comando $find$ e imprimiendo el nombre del archivo de la canción a la que corresponda de la siguiente manera:

```
T = find(C == max(C));
disp(Datos(T))
```

Listing 11: Relación con cada canción

Se podría decir que en este punto se termina el código, sin embargo existe un error entre las similitudes de canciones, la manera más eficiente y sencilla en la que se puede corregir esto es al hacer dicho proceso múltiples veces, y sumar cada resultado del vector C en cada matriz, de esta manera las probabilidades de que se obtenga una canción con valor mayor o igual a la canción correcta se reducen considerablemente, para esto se utilizo una función for en la que se grabaron 3 fragmentos de 1 segundo 4 veces.

Como elemento adicional podemos reproducir la canción seleccionada como la correcta con el comando *sound*.

4. Conclusiones

4.1. Fourier

Las transformadas de Fourier son herramientas computacionales muy importantes, estas están presentes en distintas partes de nuestras vidas con distintos propósitos. Las podemos encontrar en aplicaciones cómo *Shazam*, pero también son vistas en reconocimientos de voz, afinadores, detector de terremotos etc. El caso utilizado anteriormente es una pequeña prueba de lo que aplicaciones como *Shazam* o *SoundHound* hacen para identificar canciones.

La transformada de Fourier nos sirvió para identificar relaciones entre una canción y una grabación de audio. Estas relaciones se encuentran en los picos de frecuencias que se generan en una canción, principalmente en el rango medio de frecuencias ya que aquí se encontraran la mayor cantidad de relaciones.

4.2. Matlab

El código generado en Matlab se basa en utilizar la FFT para poder identificar los picos notorios dentro de una canción grabada. Estos picos los podemos interpretar como huellas de canciones dado que los picos después de 500 hz son aquellos que se encuentran las notas de guitarra, piano y principalmente las voces. Estas nos sirvieron para identificar relaciones entre ellas.

Dada la experimentación, se utilizaron los valores utilizados para realizar las pruebas de manera exitosa, notamos la importancia entre que los picos tengan un margen de 5 unidades para evitar repeticiones, y que la detección de una señal varía entre 0 y 2 unidades de radio. Estos valores nos sirvieron para poder afinar el detector de canciones, sin embargo, es posible cambiarlos de ser necesario. Aparte de dicha resta se obtenían errores donde una canción errónea salía con mayor correlación que la correcta. Esto se debe a distintas razones, algunos inicios de canciones son similares, o rondan por las mismas frecuencias, también existe el error del micrófono dado que no es un micrófono profesional tiende a tener ligeros errores al momento de recopilar datos. Para resolver dicho problema como se comento anteriormente se hizo una repetición de 4 grabaciones, de esta manera reducimos ese pequeño error que se puede causar por ligeras similitudes. Después de dicha corrección se obtiene un error ligero, 1 de cada 20 o 25 grabaciones se obtiene un resultado erróneo.

Ambos códigos completos se encuentran en los archivos *Database.m* y *Final2.m* correspondiendo el primero como la base de datos y el segundo para la identificación de canciones. Es necesario que se ejecute la base de datos antes para precisamente tener canciones con que

comparar, de caso contrario se generaran errores en el segundo archivo.

Referencias

[ELLIOTT, 1987] ELLIOTT, D. F. (1987). Chapter 7 - fast fourier transforms. In Elliott, D. F., editor, *Handbook of Digital Signal Processing*, pages 527 – 631. Academic Press, San Diego.

[Feregrino and Morales, 2015] Feregrino, C. and Morales, A. (2015). The Discrete Fourier Transform. Technical report.

[Morin, 2009] Morin, D. (2009). Fourier analysis. Technical report.

[Reyes Alonzo, 2013] Reyes Alonzo, J. L. (2013). Niveles de intensidad de instrumentos musicales. *Revista de la Escuela de Física*, 1(1):38–41.