# Situación problema:
# ¿Cuál es el nombre de esa canción?
# Código

Yahír Rodríguez, Sofía Cañas, Eugenio Santiesteban, Raúl Ramírez, Luis Gramillo*
(Dated: December 5, 2021)

*En el presente reporte se explica el programa implementado para la identificación de canciones, se discuten además sus limitaciones y ventajas.*

## I. INTRODUCCIÓN

El código generado tiene la función de identificar el nombre de una canción de entre una base de datos de canciones, a partir de un audio grabado por una pequeña cantidad de tiempo, simulando el comportamiento que tiene la aplicación Shazam. Para realizar correctamente una señal e identificar su similitud, necesariamente deben pasar por el mismo tratamiento, o que los tratamientos realizados lleguen al mismo resultado. Por tal motivo, el proceso y códigos para la señal grabada son realizados de una forma similar.

Para este trabajo se partirá de la explicación de los códigos utilizados, partiendo primeramente de la canción captada por grabación para después explicar el funcionamiento del código para la generación del identificador de las canciones en la base de datos (Ordenador personal).

Para fines prácticos durante el desarrollo del presente informe se mostrarán cortes de cada código, para ir ejemplificando, sin embargo serán anexados al final del mismo los códigos completos sin cortes o comentarios adicionales.

## II. CAPTACIÓN DE AUDIO

Partiendo del código **rec_song.m** (ver archivos anexos), es necesario primero la definición de los parámetros para la captación del audio (a través del micrófono del ordenador), se definen la tasa de muestreo, el formato en bits y la cantidad de canales a captar, una vez ingresada esta información se utiliza el comando *recordblocking* para captar el sonido durante 10 segundos. La cantidad de tiempo en la que se graba el sonido está dada por el parámetro *longi*, que indica el tiempo en segundos.

```
clc, clear, close all
longi=10; %tiempo (sec)
tasa=44100;bits=16;chan=1;
%datos de la cancion
Songm=audiorecorder(tasa,bits,chan);
%obtencion de audio
 disp('Start speaking.')
 recordblocking(Songm,longi)
 disp('End of Recording.');
```

Una vez que se tiene el objeto *Songm* como la grabación de audio, este es transformado a un vector que almacena la información relacionada a dicha grabación.

## III. PROCESADO DE SEÑAL

Una vez adquirida la información del audio grabado, esta es enviada a la función **principal.m** que será la encargada de realizar el procesamiento de la información, así como el despliegue de las gráficas para la representación visual de los tratamientos.

Dentro de esta función, es graficada la señal a lo largo del tiempo (para analizar arbitrariamente si fue captada una señal). Después de esto, se decidió particionar el código y llamar a la función **separaFourier.m**.

La función **separaFourier.m**. recibe la señal, así como la tasa de muestreo y el tiemp analizado, y una vez hecho esto, es utilizado el comando FFT para aplicar el algoritmo de la transfomada rápida de Fourier y pasar del dominio del tiempo al dominio de las frecuencias, por cada unidad de tiempo (10 iteraciones para el caso del tiempo escogido).

* Also at Instituto Tecnológico y de Estudios Superiores de Monterrey.; A01236136@itesm.mx

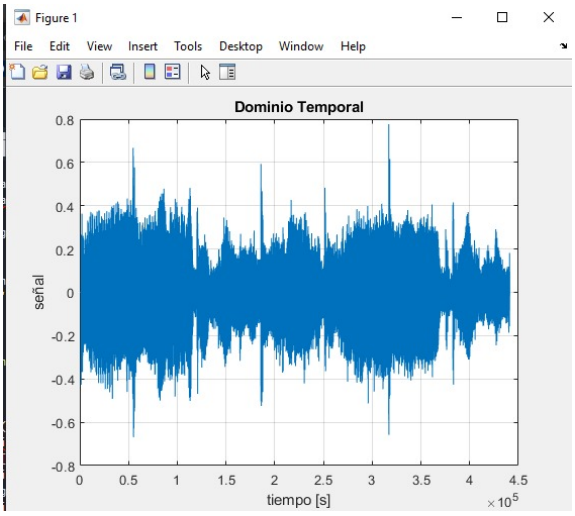


FIG. 1. Dominio Temporal

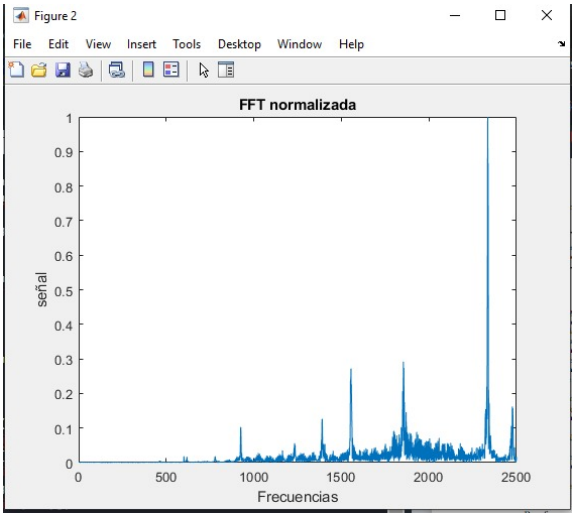


FIG. 2. FFT normalizada

```
function [canF,muestras]= ...
separaFourier(canT,tasa,longi)

%Transformada rapida de Fourier.
canF=abs(fft(canT));


%Normalizacion
canF=canF(1:2500);
canF=canF/max(canF);
muestras=zeros(2500,longi);

for i=1:longi
    muestraT=canT(tasa*(i-1)+ ...
    1:tasa*i);
    muesFn=abs(fft(muestraT));
    muesFn=muesFn(1:2500);
    muestras(1:2500,i)=muesFn;
end
end
```

Una vez que se trata la señal, esta se reduce únicamente a 2500 muestras, después es desplegado un gráfico con la información de la transformada de Fourier centrada (reduciendo el dominio de las frecuencias a los positivos).

También, puesto que la señal es grabada, la amplitud de sus frecuencias puede ser diferente a la obtenida por las canciones originales, es por esto que se normalizan las señales, dividiendo los valores por el valor máximo; de esta forma es posible encontrar una mayor coincidencia al momento de buscar.

El parámetro que devuelve la función **separaFourier** es una matriz de nombre *muestras* que, como su nombre lo indica, contiene las muestras obtenidas por FFT de dimensiones $2500 \times longi$, es decir, contiene un vector de 2500 datos por cada segundo grabado.

## IV. IDENTIFICACIÓN DE VALORES REPRESENTATIVOS

Como es sabido, el identificador de una melodía debe de ser representativo de esta, considerando la menor cantidad de información para reducir el tiempo de procesamiento del ordenador. Para esto, son considerados los máximos locales de la señal analizada, considerando que las frecuencias con mayor amplitud, son aquellas que representan de la mejor manera la señal analizada, de esta manera, se utiliza la función *maximos.m*.

La función anteriormente mencionada utiliza el comando *findpeaks* para identificar una serie de valores máximos dentro de cada intervalo de tiempo, puesto que, pudiesen existir periodos de tiempo para los cuales la amplitud máxima sea un valor considerablemente pequeño. Es utilizado el parámetro *alt* para establecer a partir de qué numero, puede ser considerado como máximo, igualmente, para limitar la cantidad de máximos locales identificados es utilizado el parámetro *anch*, que representaría la fracción de segundo para el cual se identificará un máximo.

```
function signa = maximos(longi, ...
muestras)

anch=200; alt=100;
cols_sub=3;
```

```
for i=1:longi
    muesF=muestras(1:2500,i);
    [maxi,frec]=findpeaks(muesF, ...
    'MinPeakDistance',anch, ...
    'MinPeakProminence',alt);
    signa(1:length(maxi),2*(i-1)+ ...
    1:2*i)=[frec,maxi];

end
end
```

Después de esto, la información identificada es almacenada dentro de la variable *signa* (De "signal"), y posteriormente es devuelta esta variable a la función **principal.m** para su posterior uso.

Después de realizarse casos de prueba grabando silencio, se identificó que la función **maximos.m** devuelve un vector vacío, puesto que el procesamiento posterior requiere forzosamente que la variable *signa* contenga máximos y mínimos. Entonces se optó por insertar una condición, que establece que mientras esta variable contenga información, se realizará el procesamiento posterior, en caso contrario, se descarta procesamiento (es ruido, no una melodía).

```
if length(signa)==0
    referencePair=[0,0];
else
points=size(signa);

for i=1:longi
    plot(i,signa(1:points(1),2*(i ...
    -1)+1),'xk','MarkerSize',7);
end
```

## V. IDENTIFICADOR

Una vez que son identificados los puntos significativos, se realiza una combinación lineal de estos, es decir, por cada frecuencia identificada, se comparará la distancia temporal entre esta y cada una de las frecuencias identificadas. Sin embargo, puesto que considerar todas las posibles combinaciones requiere un alto costo computacional así como para el almacenamiento del mismo, únicamente se consideran las 3 frecuencias más cercanas, cambiando de una cantidad de datos $n^3$ donde $n$ es la cantidad de frecuencias representativas identificadas, a $3 \times n$, reduciendo considerablemente el costo computacional sin afectar realmente al procesamiento de la información.

```

counter=1;
for k=1:points(2)-1
    for i=2:points(1)-1
        front=[signa(i,k), ...
        signa(i,k+1)];
        up=[signa(i,k), ...
        signa(i+1,k+1)];
        ab=[signa(i,k), ...
        signa(i-1,k+1)];
ab=[signa(i,k),signa(i-1,k+1)];
        if front(1)*front(2)~=0
            referencePair(counter, ...
            1:2)=front;
            counter=counter+1;
        end
        if up(1)*up(2)~=0
            referencePair(counter, ...
            1:2)=up;
            counter=counter+1;
        end
        if ab(1)*ab(2)~=0
            referencePair(counter, ...
            1:2)=ab;
            counter=counter+1;
        end
    end
end
end
end
```

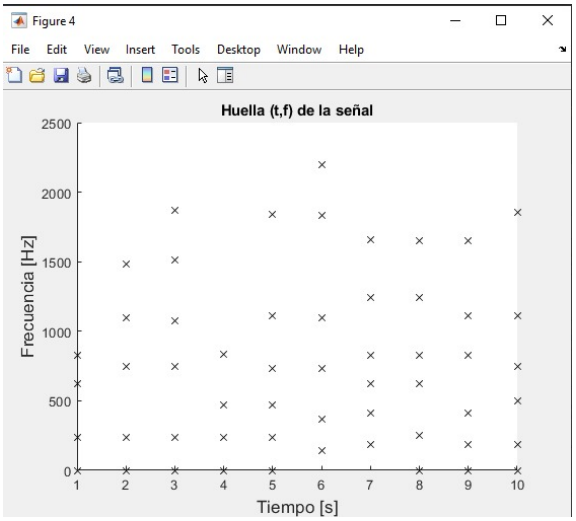


FIG. 3. Puntos significativos

La sección de lo anterior, se encarga de identificar la distancia temporal entre cada una de las frecuencias representativas, con 3 frecuencias contiguas. El valor obtenido de nombre *referencePair* ya representa el identificador de la señal analizada, siendo esta una matriz con dos columnas donde la columna 1 representa el valor frecuencial (repitiéndose 3 veces) y la columna 2, su correspondiente distancia temporal hacia otra frecuencia contigua.

Finalmente, la función **principal.m** devuelve esta variable, enviándola al programa **rec_song.m** para finalmente, compararla con la base de datos.

## VI. CREACIÓN DE BASE DE DATOS

Una vez conocido el proceso realizado para la señal grabada, es realizado el mismo procesamiento para las canciones que se desean almacenar. Sin embargo, mientras que la señal grabada utiliza la función **principal.m**, el código **crea_hash** utiliza dentro de su código una reducción de **prinpipal.m** (Eliminando únicamente el despliegue de gráficos, puesto que las melodías contaban con un tiempo considerablemente mayor).

El programa **crea_hash.m** cuenta adicionalmente con lineas de código que se comentarán a continuación.

## VII. CREACIÓN DE HASH

Antes de comenzar a generar el identificador para cada melodía, es necesario considerar primeramente la cantidad de canciones con las que se cuenta, así como la posibilidad de que estas ya cuenten con un identificador hecho (corridas de código anteriores).

```
%'C:\Users\luisg\Google Drive\ ...
%MATLAB\FISMAT\PROYECTO\canciones'
%direc=input('Deposite la ...
%direccion donde se ubica las ...
%canciones:\n','s');

direc='C:\Users\luisg\Google ...
Drive\MATLAB\FISMAT\PROYECTO\ ...
canciones';

sonams=dir([direc,'\*.mp3']);
nombres=struct2table(sonams);
canciones=[];
for i=1:height(nombres)
canciones=[canciones, string ...
(table2array(nombres(i,1)))];
end
```

Primero, son escritas dentro del código las carpeta en la que las canciones se identificarán (la carpeta debe de estar dentro del *path* de MATLAB, estas en formato *.mp3*, mediante el comando *dir* son identificadas las canciones, sin embargo el formato *struct* que maneja dificulta su tratamiento por lo que se hace un procesamiento para transformarlo en un vector de caracteres que contenga el nombre de cada archivo, almacenado en *canciones*.

```
%Crea referencepair de cada cancion
for u=1:length(canciones)
    %nombre=sprintf(['REFERENCE_', ...
    char(canciones(u))]);
    temporal=char(canciones(u));
    temporal=['REFERENCE_', ...
    temporal(1:end-4),'.mat'];
    if exist(temporal)~=2
    [song,tasa]=audioread([direc, ...
    '\',char(canciones(u))]);
    song=song(:,1)';
    songinf=audioinfo([direc, ...
    '\',char(canciones(2))]);
    longis=fix(songinf.Duration);
```

Una vez que se conocen el nombre de las canciones, es necesario saber si fue realizado su identificador por lo que se usa el comando *exist()* para identificar si el nombre de la canción más una extensión *.mat* fue anteriormente generada, en caso de que no procede con el mismo procesamiento que la señal grabada, con la diferencia de que es seccionada en partes de 10 segundos cada canción (para poder realizar exactamente el mismo tratamiento que la melodía grabada), aunque finalmente es construido el identificador uniendo cada parte generada.

Finalmente, la matriz de dos columnas (de la misma dimensión que *referencePair* en la señal grabada) es exportada como un archivo tipo *.mat* llevando el nombre de la canción procesada. De esta forma es creado el identificador para cada canción dentro del directorio descrito para finalmente compararlo con la señal grabada.

## VIII. DETECCIÓN DE COINCIDENCIAS

Una vez que se obtiene la base de datos, el código *rec_song* identifica las canciones del directorio anteriormente dicho, y envía, el nombre de las canciones o archivos en el directorio, junto con el identificador a la función **compara.m**.

Esta última función, recorre cada una de las canciones dentro del directorio y obtiene una comparación de cada canción en relación al *referencePair* de la señal grabada.

```
function [Mea,song_fit] = ...
compara(n,canciones,referencia)
```

```
3
4  comparations=zeros(n,1);
5
6  for i=1:n
7      nombre=sprintf(['REFERENCE_', ...
8      char(canciones(i))]);
9      nombre=[nombre(1:end-4),'.mat'];
10     load(nombre)
11     comparations(i) =mean(mean( ...
12     ismembertol(Datos,referencia, ...
13     10^-5)));
14 end
15
16 [Mea,song_fit]=max(comparations);
17 end
```

Por medio del comando *ismembertol* es realizada una comparación, identificando si el *referencePair* se encuentra en la canción que analiza, indica qué tan parecidos son estos vectores tanto en frecuencias como en distancias con una tolerancia (puesto que no serán exactamente el mismo valor, causado por ruido, interferencias, etc.), indicando un valor booleano 1 si hay coincidencias y 0 en caso contrario. Después obtiene el parecido promedio entre estos y después es almacenado este dato dentro de la variable *comparation* que almacenará el parecido promedio de cada canción. Finalmente identifica el valor máximo (aquella canción con mayor parecido) y envía esta información de regreso.

Para concluir, el programa *rec_song* identifica el número de canción con coincidencia máxima y despliega el nombre de la canción así como su artista.

```
1  %Comparacion
2  direc='C:\Users\luisg\Google ...
3  Drive\MATLAB\FISMAT\PROYECTO\ ...
4  canciones';
5  sonams=dir([direc,'\*.mp3' ]);
6  nombres=struct2table(sonams);
7  canciones=[];
8  for i=1:height(nombres)
9  canciones=[canciones, string ...
10 (table2array(nombres(i,1)))];
11 end
12  [x,y]=compara(length(canciones), ...
13  canciones,referencia);
14  fprintf('Canci n: ')
15  fprintf([audioinfo(canciones(y)) ...
16  .Title,'\n'])
17  fprintf('Artista: ')
18  fprintf([audioinfo(canciones(y)) ...
19  .Artist,'\n'])
```

## IX. CASOS DE PRUEBA

En los videos se muestran casos de prueba que demuestran el funcionamiento del programa que se expone en el presente reporte: `https://drive.google.com/drive/folders/1HVzXC9HUA13j8kxLsHFLJtrvWzXWRwOH?usp=sharing`

## X. LIMITANTES Y CONCLUSIONES

Estos códigos, a pesar de ser puestos a prueba y considerando casos hipotéticos, no son perfectos por lo que cuentan con limitantes.Puesto que la creación de un identificador para una canción requiere ser procesada, es segmentada en porciones de 10 segundos, sin embargo el programa no es capaz de analizar el residuo (de una canción de 63 segundos sería capaz de generar un identificador para 60 segundos), por lo que buscar una canción por una grabación de esta al final de la melodía no permitiría identificar con certeza la canción.

Otro detalle a considerar es la base de datos, este programa tiene la limitación de que devuelve la canción con mayor cercanía a la grabación, es decir, dentro de la base de datos identificará aquella con mayor coincidencias, independientemente de que sea o no la misma canción. También, la calidad del programa depende de la cantidad de canciones almacenadas en la base de datos.

En conclusión, hay una gran cantidad de situaciones sin considerar que debilitan la confiabilidad del programa y su correcto funcionamiento en ciertos casos pero sirve de punto de partida para futuras modificaciones y mejorar su funcionamiento, ya que ha logrado identificar la mayoría de las canciones con las que hemos probado. No sólo eso, el programa es inmune en su mayoría al ruido en la señal de entrada ya que el criterio de coincidencia es que exista una cantidad significativa o la mayor de pares de *hashes*.

Este código o sus posibles variantes perfeccionadas se realizan con un objetivo de beneficio para la sociedad y al mismo tiempo nos permite apreciar la belleza de lo que nos rodea y su representación matemática. El algoritmo descrito en el presente trabajo de investigación consigue el aseguramiento de los derechos de autor, teniendo una gran importancia hoy en día y reconociendo las ventajas que usarlo trae

consigo así como las posibles implicaciones en memoria y procesamiento.