PROYECTO DE SONIDO

Grupo 15

Fundamentos de Sonido e Imagen

2021-2022

1. Objetivo

El objetivo de este proyecto es crear una función o un conjunto de funciones capaces de detectar si un audio (tanto estéreo como mono) contiene componentes tonales o si no las tiene. Como extra también detectamos, en el caso de que sea tonal, el máximo de la amplitud y en el caso de que sea una nota también detectaría que nota es. Además, para ambos casos (tonal y no tonal), mostramos tres gráficas con las que podríamos comprobar si existen realmente componentes tonales, en las que se muestra el audio en función del tiempo, la amplitud en dB en función de la frecuencia y el espectrograma.

2. Funciones

`deteccion_tonos`

FINALIDAD: esta función devuelve el parámetro (salida), que nos indica si la señal tiene componentes tonales o no. Si la salida sale “Verdadero” la señal es tonal, y “Falso” en caso contrario.
Dos parámetros de entrada:

-archivo: señal a analizar, formato .wav.

-modo: 1 para indicar que el programa va a funcionar tambien como afinador(Importante: introducer una nota). 0 para indicar que no quiere usar el modo afinador, es decir, modo de trabajo normal.

SINTAXIS: `salida = deteccion_tonos(archivo,modo)`

Parámetros de entrada:

archivo: Nombre del archivo de audio a analizar.

Modo: Indica si se quiere usar o no la función afinador_simple.

Parámetros de salida:

salida: Devuelve “Verdadero” si la señal contiene componentes tonales o “Falso” si no las contiene.

Primero leemos la señal de audio: *[audio, fs] = audioread(archivo);*
fs: frecuencia de muestreo.

A continuación debemos pasar la señal de audio de estéreo a mono, en el caso de que esta ya sea mono no se verá afectada:
x = sum(audio, 2) / size(audio, 2);

Luego para sacar el vector con todos los tonos, incluidos los irrelevantes:
amplitud = findpeaks(mfss);

Después para poder obviar los tonos irrelevantes, obtenemos el umbral (media cuadratica de los picos) de la señal para en el siguiente paso sacar el vector de tonos que se encuentren solo por encima del umbral:
amplitud = findpeaks(mfss, fs, 'Threshold', umbral);

Y finalmente para poder enseñar por pantalla si hay componentes tonales usamos: *isempty(amplitud)*, esta función nos ayuda a ver si el vector está vacío, en ese caso no habría tono, o por el contrario contiene algo, y habría tonos.

FUNCIONES EXTRAS:

Para obtener la amplitud de tono mayor, hacemos el maximo de la transformada de Fourier(mfss), y con eso conseguimos el máximo de amplitude.
[mx] = max(mfss);

Si la variable modo es 0 entonces no haria nada y si la variable modo fuera 1 entonces se llamaría a la función afinador_simple que diría si se trata de una nota perteneciente al banco de notas, identificando la nota, o si no pertenece a ese banco.

`calcular_espectro`

FINALIDAD: Devuelve el vector de los valores de la DFT de la señal y los valores de frecuencia asociados a la vector.

SINTAXIS: `[mfss,fHz]= calcular_espectro(señal,fs,Nfft)`
Parámetros de entrada:
- señal: señal a analizar.
- fs: frecuencia de muestreo de la señal.
- Nfft: tamaño de la ventana.

Parámetros de salida:
- mfss: vector con los valores de la DFT.

fHz: vector de las frecuencias asociadas a mfss.

FUNCIONAMIENTO:

1º Calculamos el número total de muestras y le asignamos la longitud a la ventana.

2º Reordenamos el vector de datos en una matriz de datos.

3º Enventanamos y calculamos la FFT de cada columna de datos.

4º Nos quedamos solo con las frecuencias entre 0 y Pi.

5º Calculamos la media y el vector de frecuencias asociado a mfss.

INFORMACION ADICIONAL:
El tipo de ventana que utilizamos es una ventana rectangular con una longitud de Nfft muestras. Para ello, hemos reescrito la función *ver_espectro()* de la práctica 1.

Afinador_simple

FINALIDAD: muestra por pantalla la nota musical introducida como señal.

SINATXIS: `[]= afinador_simple(x,fs)`

Parámetros de entrada:
X: señal a analizar(muestreada)
Fs: frecuencia de muestreo

FUNCIONAMIENTO:

Primero volvemos a llamar a calcular_espectro pero ahora con una ventana más grande para obtener mejor calidad de los tonos.

Luego utilizamos la funcion findpeaks para hallar todos los picos.

Despues usamos la función findpeaks con ‘threshold’ usando como umbral la media cuadrática de la amplitud, de esta manera podemos obtener solo los picos que nos interesan.

Para poder comparar valores, tenemos una serie de datos de las frecuencias correspondientes a cada nota en una actava.

Además añadimos un valor c como margen de error porque a pesar de que la nota do tenga frecuencia x puede haber variaciones y que en espectro el pico esté en x+4. Para esto añadimos el valor c con valor 9 para los problemas de la precision.

Escogemos el valor 9 porque es un valor razonable, ya que la diferencia minima entre frecuancias de las notas musicales de la escala que trabajo es 20, por lo que menos de la mitad para evitar coincidencias.

Por ejemplo el mi(329Hz) y fa(349Hz) entre ellos tienen una diferencia de 20 hz, si yo escogo 10 o mas, puede que al analizar un fa obtenga frecuencia 339, y como tengo 10 de factor de sensibilidad el programa me daria que es un fa y un mi, por eso escogo el 9.

Por último, utilizamos una serie de ifs para comprobar y mostrar si la nota se encuentra entre nuestro banco de notas y muestra la nota seleccionada en caso de que este.

INFORMACION ADICIONAL:

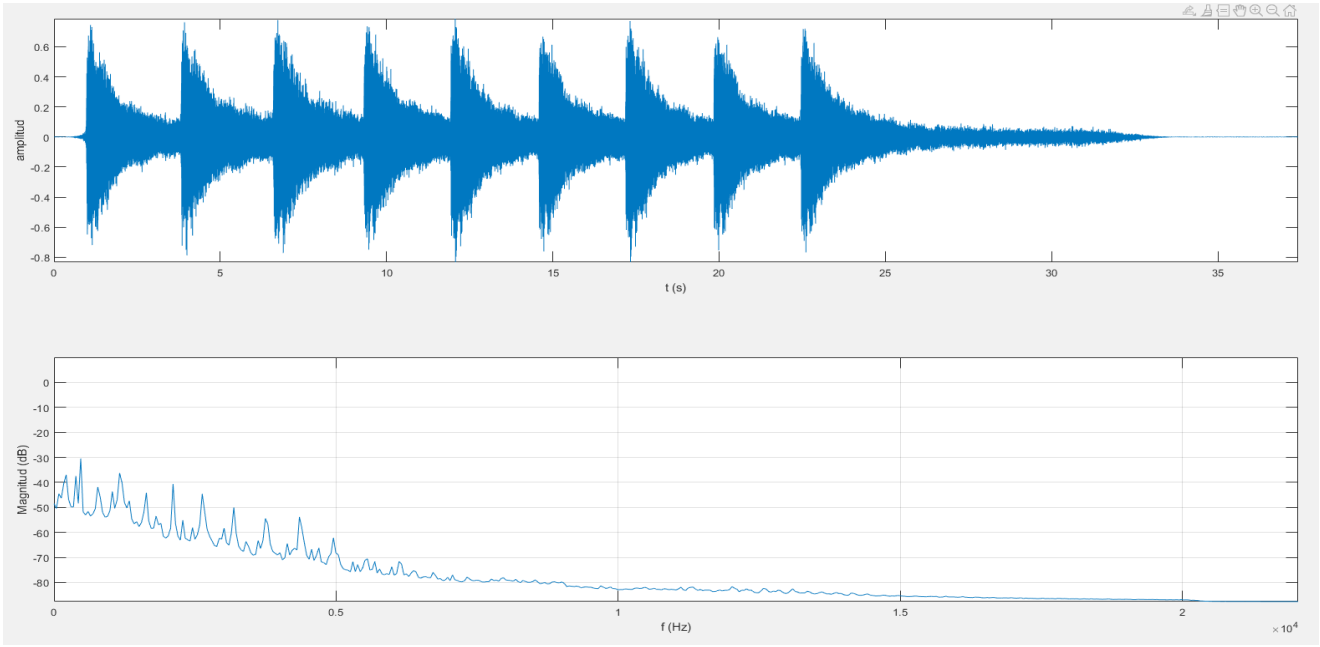
Solo trabaja con las notas sin sostenidos y en una escala determinada, porque para la implementacion de los sostenidos es añadir datos a la base de datos de las notas. Y para añadir más octavas podríamos, analizar la frecuencia dada, encagarla en una escala, es decir, una escala tiene un maximo y un minimo de frecuencia porque lo si la frecuencia analizada esta dentro de ese intervalo pertenece a esa escala.

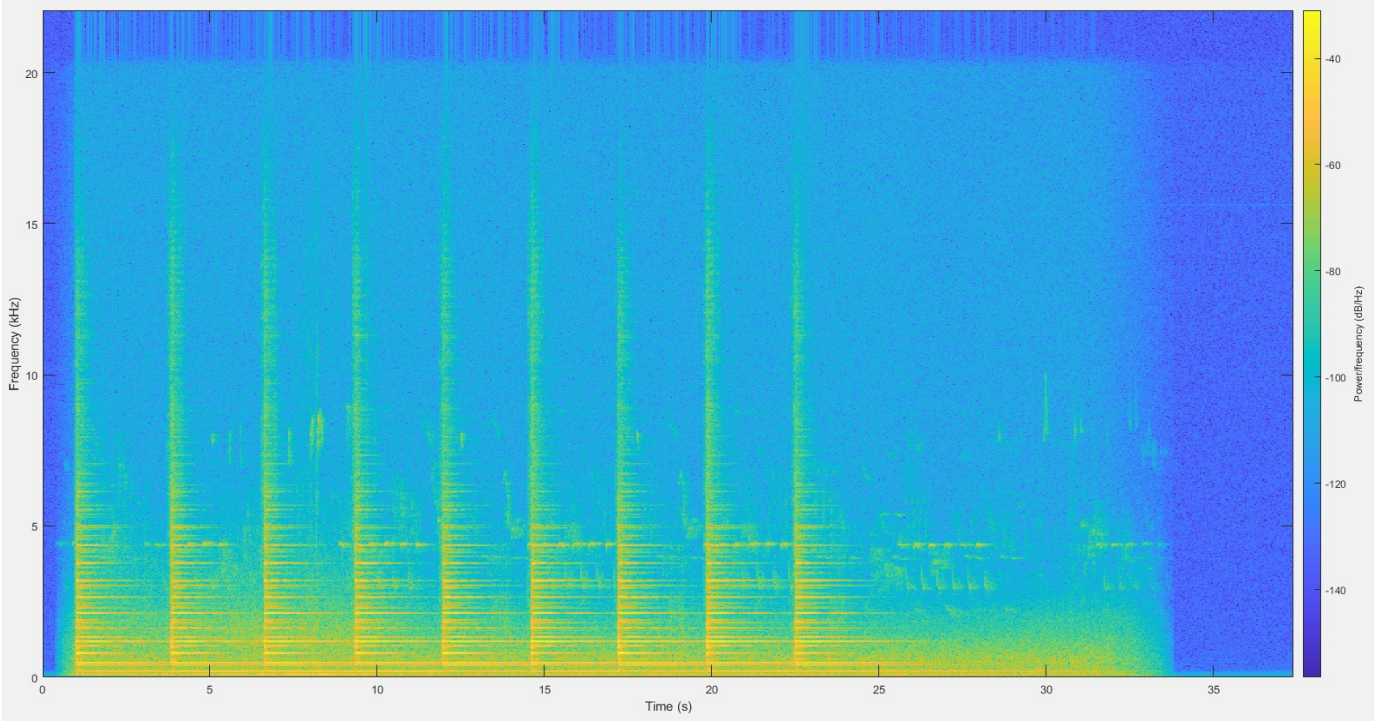
Esta separación de escalas creemos que sería más eficiente porque en cada escala la diferencia entre frecuencias es diferente y habría que usar otro factor de escala.

3. Resultados de pruebas

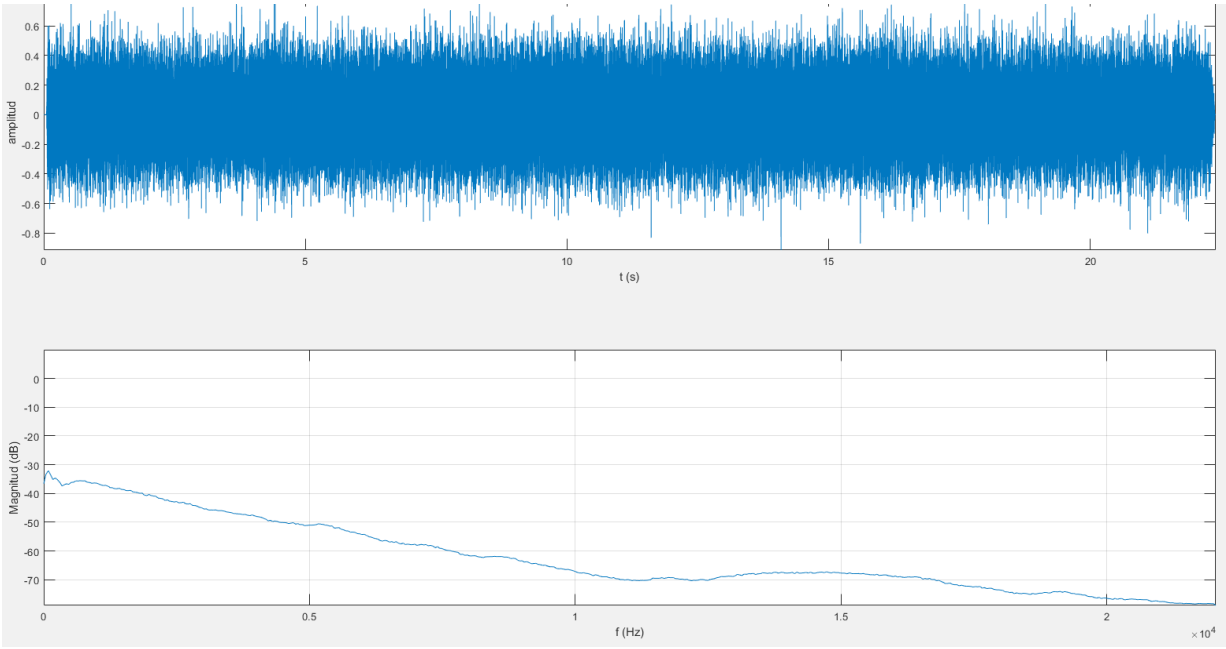
Audio con componentes tonales: (*campana.wav*)

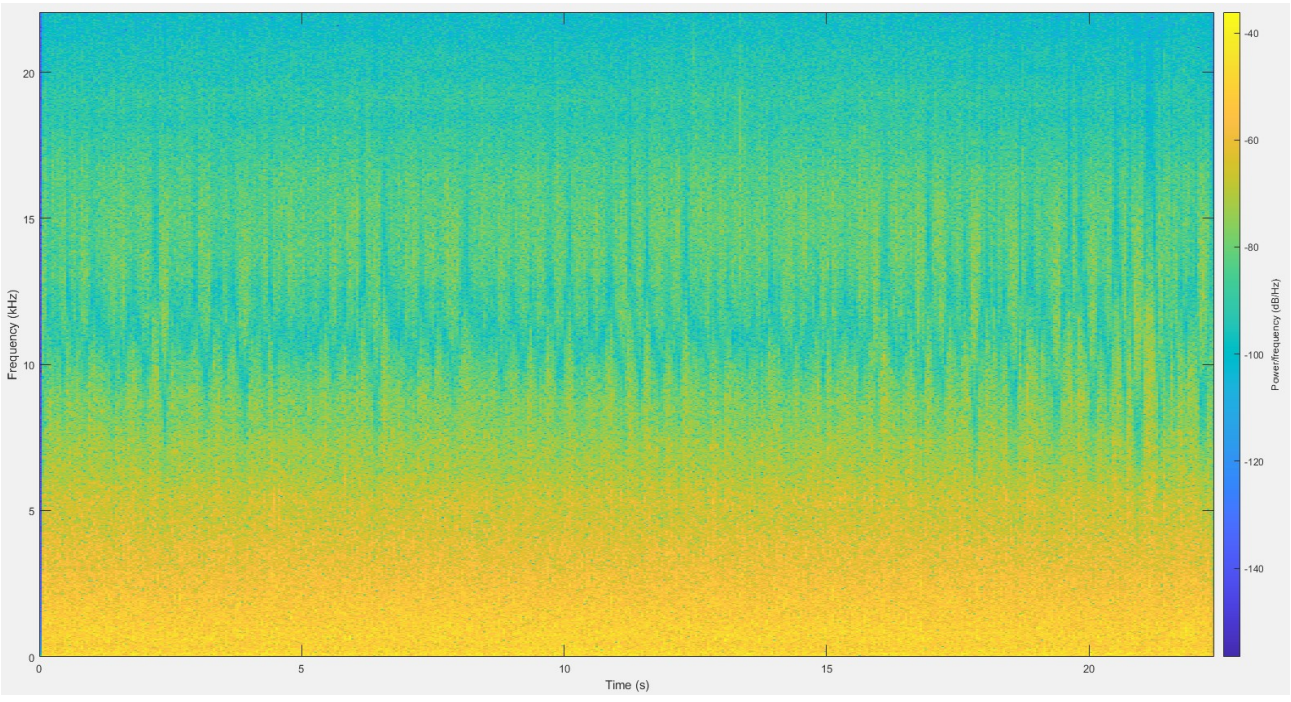
- Amplitud: -30 dB





Audio sin componentes tonales: (*cascada.wav*)





Nota musical: (*do.wav*)

Saca por pantalla ” nota introducida: DO”
Ademas de todos los datos de la propia function.