Práctica Final

PROCESADO DIGITAL DEL SEÑAL

La Salle Campus Barcelona

2018

Índice

1. Planteamiento del problema.

Los ingenieros de Sistemas Audiovisuales están desbordados de trabajo debido al poco personal que hay para la sonorización de la Xmas Party que organiza el Social Club el próximo 13 de diciembre en la sala de congresos.

Para la música en directo de la Coral se requiere utilizar micrófonos de ambiente para amplificar su voz, pero al tener los altavoces en el techo en línea recta de los micrófonos se genera el efecto de feedback o retroalimentación.

La retroalimentación de un sistema, consiste en la transferencia de la señal de salida a la entrada del mismo sistema o circuito, lo que deriva en un aumento en el nivel de salida y auditivamente podemos identificar como un tono puro de alta frecuencia.

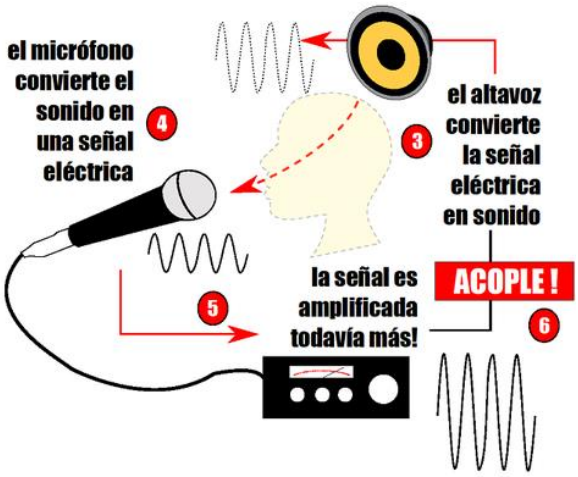

Ilustración 1 Grafico del efecto Feedback

El Objetivo de la práctica es realizar un software de prueba en Matlab que permita identificar la frecuencia a la que se acopla la señal (tono interferente) para poder eliminarlo a nivel frecuencial y que sea inaudible para el público del concierto.

En la primera parte de la practica trabajaremos sobre un audio, previamente grabado de la sala de congresos para poder escuchar de manera audible como podremos eliminar las frecuencias y tener así una primera simulación.

En la segunda parte, el objetivo es poder monitorizar todo el concierto y poder ver en tiempo real si aparece cualquier otra interferencia

2. Ejercicio de cancelación de interferencias sinusoidales.

El archivo de audio `testing1.wav` contiene una señal de voz que se ha contaminado con un tono puro o sinusoidal real, que varía su frecuencia en función del tiempo. Nuestro objetivo es diseñar un filtro que nos permita eliminar estas interferencias evitando distorsionar el resto de la señal.

En la figura siguiente se muestra el diagrama de bloques del sistema de cancelación de interferencias que vais a implementar el cual está formado por:

- Un módulo de procesamiento a bloques (*Blocking + Windowing*), que se encarga de escoger de toda la señal S(n) una trama de longitud constante Sk(n).
- Un módulo que realizará la detección del tono interferente en la trama bajo análisis, obteniendo el valor de su pulsación discreta Wok.
- Un filtro elimina banda que cancelará dicho tono interferente en cada trama, generando una señal de salida Sfilt(n) que será el resultado de la concatenación de todas las tramas filtradas.

Como se puede observar en la figura, la señal filtrada se guardará en un fichero de salida de audio (.WAV), y se visualizará dicha señal mediante un análisis tiempo-frecuencia con espectrogramas, lo que nos permitirá observar si se ha conseguido cancelar la interferencia.

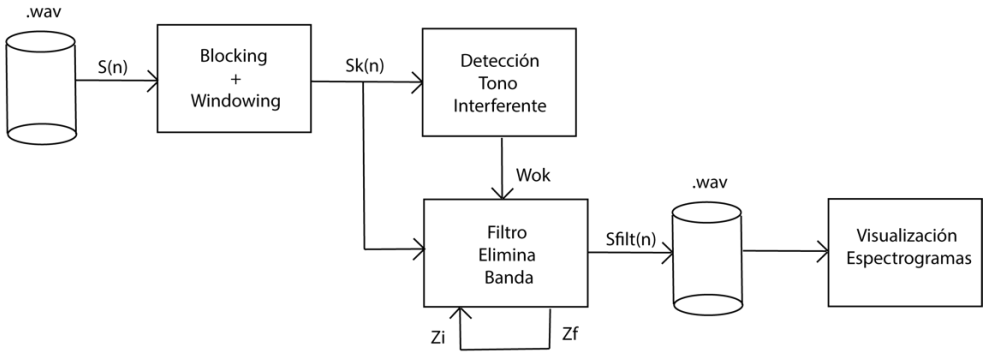

Ilustración 2 Diagrama del sistema a implementar

Al tener unas interferencias dinámicas, no podemos tratar todo el audio por igual y habrá que trabajar en tramas. Utilizaremos el método OLA (Overlapp and Add). En la figura siguiente se muestra un ejemplo de filtrado con este método, donde la señal se divide en tramas de longitud P. El filtrado de cada trama produce una señal de duración mayor, concretamente P+N-1, siendo N la longitud de la respuesta impulsional del filtro. Dado que el filtro es un sistema lineal, la salida global se puede calcular como la superposición en el tiempo de las salidas de cada trama, superponiendo las N-1 muestras finales de una trama con las N-1 muestras iniciales de la trama siguiente.

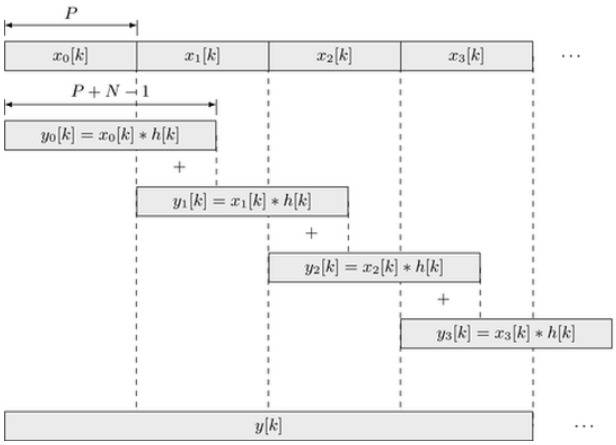

Ilustración 3 Overlapp and Add

En nuestro caso, en vez de implementar el proceso Overlap and Add, utilizaremos la llamada a la función filter.m, la cual nos permite acceder a las condiciones iniciales y finales del filtro, para mantener así la continuidad de la señal global evitando la aparición de transitorios al inicio de cada bloque. En la Ilustración 2, se muestra esto con el traspaso de las variables Zi y Zf.

En resumen, para realizar el proceso de filtrado dinámico, es decir, que se adapte a la frecuencia de la interferencia, se propone que implementéis un procesado por bloques. Para ello, dividid la señal en tramas de longitud constante, utilizando una ventana rectangular de 10 mseg. Utilizad un bucle con la sentencia `for`, iterando para cada una de las tramas a procesar y aplicad el siguiente proceso:

1. Calculad la DFT de la trama (llamada a la función `fft.m`).
2. Buscad el valor máximo de amplitud en dB[1] con la función `max`.
3. Convertid la posición del máximo detectada a un valor de pulsación discreta ω entre 0 y π.
4. Calculad el valor de los dos ceros de un filtro FIR que cancele la frecuencia detectada y convertid dichos ceros a un vector de coeficientes utilizando la función `poly.m`.
5. Filtrad la trama de señal con el vector de coeficientes calculado utilizando la función `filter.m`. Dado que es un filtro FIR fijad el vector de coeficentes del denominador del filtro A = 1.
6. Añadid a la llamada anterior el vector de condiciones iniciales (`Zi`) y retornad también el vector de condiciones finales (`Zf`). Para garantizar la continuidad en el proceso de filtrado, sin que haya transitorios, es necesario que las condiciones iniciales del filtro sean igual a las condiciones finales de la trama anterior. Para el filtro diseñado, que contiene dos ceros, este vector será de dos coeficientes. Dado que la primera vez que llamemos al filtro no tendremos acceso a las condiciones finales de la trama anterior, inicializaremos esta variable a ceros (es decir, a un vector con dos ceros). Para más información consultad la ayuda de la función `filter.m`.

[1] Para calcular la amplitud en dB de X lo podéis realizar con la llamada `20*log10(abs(X))`

Una vez finalizado el proceso de filtrado por bloques visualizad el espectrograma[2] de la señal contaminada `testing1.wav` y la señal una vez filtrada `testing.wav`. A continuación tenéis un ejemplo de visualización del espectrograma.

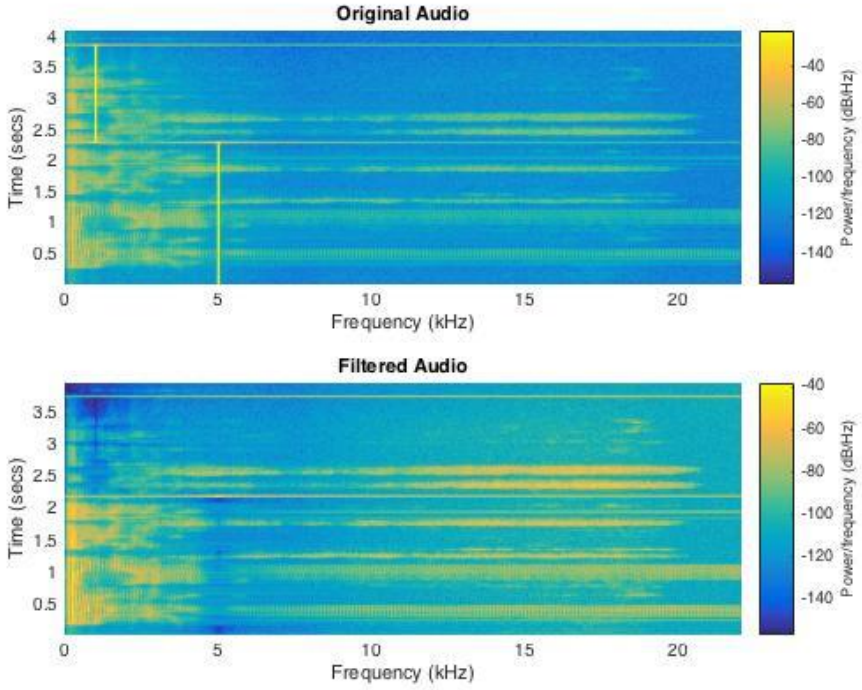

Ilustración 4 Espectrogramas de la señal original y filtrada

Justificad matemáticamente en la memoria de la práctica el diseño del filtro FIR que se adapte para eliminar la frecuencia interferente detectada en cada trama (que varía en el tiempo), mediante dos ceros, de forma que éstos estén situados justo encima de las componentes frecuenciales a cancelar (para cada tono interferente, en dos fases concretas del plano Z, y encima del círculo unidad).

Finalmente, exportad la señal filtrada en formato WAV usando la llamada a la función `audiowrite.m` o `wavwrite.m` dependiendo de la versión de Matlab, y llámala `testing_out.wav`.

Generad un script de Matlab[3] (con nombre `ejercicio1.m`) que incorpore todo el proceso de filtrado por tramas de la señal de audio, así como las ecuaciones de diseño de los filtros hasta llegar a los vectores de coeficientes. Dentro del script se generarán todas las gráficas necesarias para el análisis de los resultados (indicadas en los parágrafos anteriores). Cada gráfica tendrá en su título el nombre del aquello que representa.

[2] La función `spectrogram.m` os permite calcular un análisis tiempo-frecuencia de la señal, obteniendo como resultado una representación que nos permite ver cómo cambia el espectro de la señal a lo largo del tiempo. Escoged como parámetros de entrada los siguientes: `WINDOW = round(0.04*fs),NOVERLAP = round(0.03*fs),NFFT = round(0.04*fs)`, siendo `fs` la frecuencia de muestreo del audio original.

[3] Un script es una función que no tiene la cabecera de función, y permite así poder inspeccionar todas las variables generadas una vez éste se ha ejecutado.

3. Cancelación de Interferencias en Real Time.

En el primer ejercicio hemos podido escuchar la cancelación de frecuencias de manera dinámica, aunque siempre sobre un archivo de audio previamente grabado.

El objetivo de este segundo ejercicio es realizar el análisis a tiempo real del audio capturado directamente desde el ordenador. Debido a las limitaciones de las librerías de la versión de prueba de Matlab. No podremos escuchar en tiempo real la cancelación, pero podremos hacernos una idea de manera visual.

Con las funciones `audiorecorder.m`[4], `recordblocking.m`[5] y `getaudiodata.m`[6] podremos capturar por bloques el audio del micrófono.

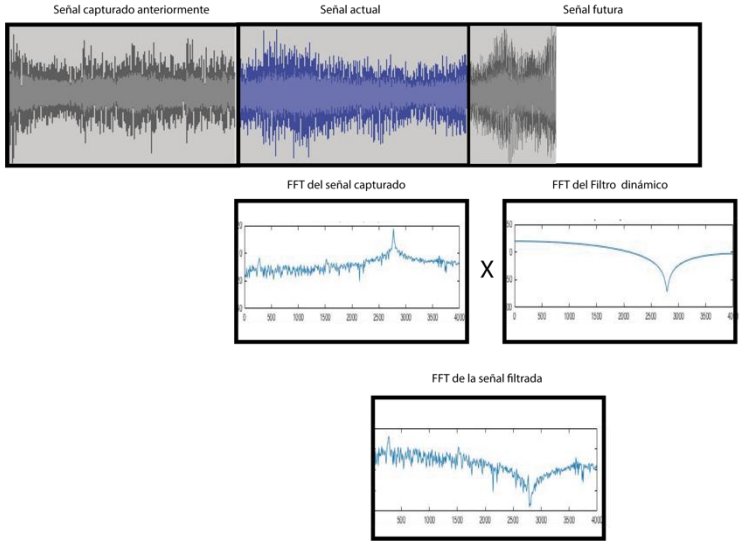

Ilustración 5 Proceso Interno RT

En este caso, realizaremos el procesado de la señal capturada por el micrófono también por bloques, pero mostraremos el resultado de cada bloque justo después de haberlo obtenido. Por ello, usaremos un tamaño de bloque mayor (que será en este caso de 100 ms), y para el filtrado utilizaremos la técnica basada en la DFT[7], utilizando el siguiente diagrama de bloques:

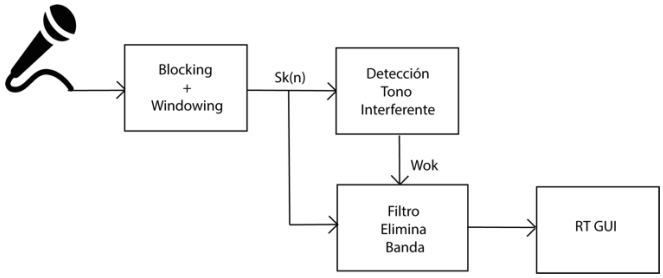

Ilustración 6 Diagrama de bloques RT

[4] Audiorecorder.m genera un objeto de audio que se conecta con la tarjeta de audio.
[5] Recordblocking.m permite capturar audio en bloques de manera continuada.
[6] Getadudiodata.m convierte el archivo de audio del bloque en un array de valores.
[7] Propiedad de convolución circular de la DFT. NOTA: Recordad que para que la convolución circular coincida con la convolución lineal es necesario que el número de puntos de las DFTs será mayor o igual a la suma de longitudes de ambas señales (señal de entrada y respuesta impulsional) menos 1.

Igual que en el ejercicio anterior, procesaremos cada bloque de audio de manera individual dentro de un bucle, buscando la posición de la frecuencia de máxima amplitud. En este caso, diseñaremos un nuevo filtro, a partir del diseño anterior, añadiendo para cada tono interferente dos polos con las mismas fases de los dos ceros del filtro que lo cancelan, pero con módulos igual a r, donde $0 < r < 1$ (filtro notch). Esta estructura permite obtener un filtro con mayor selectividad, con un ancho de banda de rechazo regulable en función del valor de r. En este caso, deberéis replantear la ecuación de diseño del filtro, teniendo en cuenta la nueva estructura (ahora tenemos un filtro IIR, mientras que en el apartado anterior teníamos un filtro FIR). Calcula los vectores de coeficientes de la ecuación en diferencias (o también de la función de transferencia en Z) del filtro, que en este caso tendrá un numerador y un denominador. Obtendremos el espectro filtrado a partir de la multiplicación de espectros.

En el apartado de este ejercicio de la memoria de la práctica visualiza el espectro de la señal capturada la respuesta en frecuencia del nuevo filtro notch diseñado, para los dos casos siguientes: $r = 0.9$ i $r = 0.99$ y el espectro de la señal filtrada. Realizad esto para una trama concreta, y silbando para generar la interferencia variante con el tiempo. A continuación, se muestra un ejemplo de dicha representación.

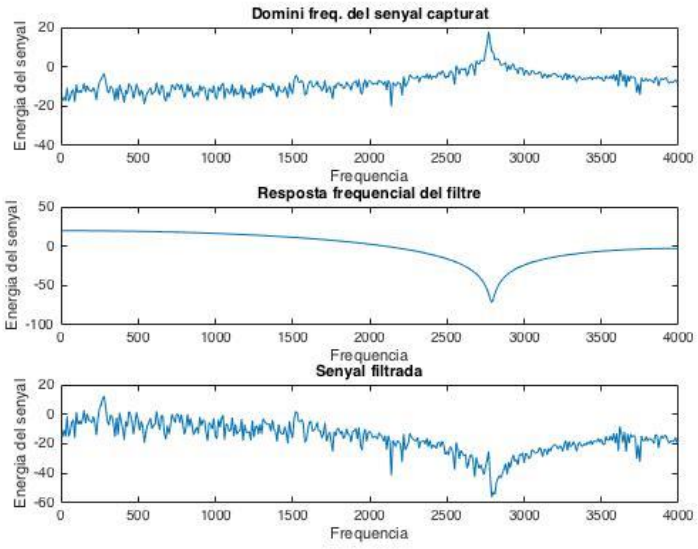

Ilustración 7 Captura instantánea del RT

Genera un script de Matlab que incorpore todas las ecuaciones de diseño de los filtros hasta llegar a los vectores de coeficientes, y realice el proceso de filtrado en tiempo real dentro un tiempo de procesado máximo de 10 segundos. Dentro del script se generarán todas las gráficas necesarias para el análisis de los resultados (indicadas en los parágrafos anteriores). Este script servirá para testear el funcionamiento del código antes de incluirlo en la interfaz gráfica.

4. Interfaz gráfica RT

La Interfaz gráfica en la que integraréis vuestro código del apartado anterior contiene dos botones y las tres graficas (ver figura siguiente). EL objetivo de la GUI es poder activar y desactivar el analizador en el momento que se considere. A continuación, tenéis una breve explicación del diseño de interfaces con la herramienta guide de Matlab, más una explicación de cómo realizar esta integración.

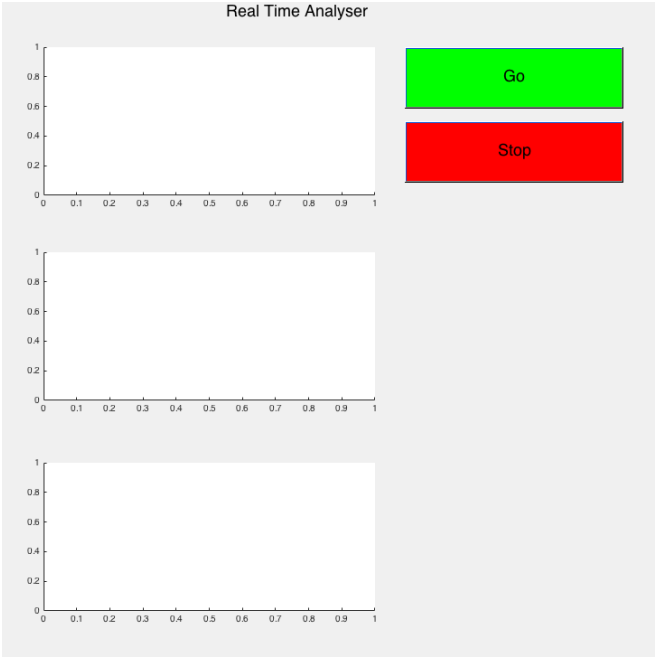

Ilustración 8 Interfaz gráfica del analizador

1.1 Breve explicación de GUI

Una GUI (Graphical User Interface) es una herramienta que permite facilitar la interacción del usuario para realizar funciones demostrativas o que impliquen dar feedback visual gráfico.

La GUI contiene dos tipos de ficheros, ambos con el mismo nombre:

<Nombre_función> .fig Es la parte visual de la interfaz y contiene los objetos que facilitan la interacción con el usuario (controles, botones, ejes gráficos, etc.)
<Nombre_función> .m Es la parte de código que controla la interfaz visual y ejecuta los pedidos que se activan desde esta.

Para empezar a diseñar o modificar una interfaz existente simplemente teclee desde línea de comandos:
>> guide;

Cada objeto que hayamos incorporado a la interfaz tendrá una parte de código asociada (funciones definidas dentro del mismo .m). La parte llamada "Callback" será aquella que

se ejecutará cada vez que el usuario interactúa con ese objeto (por ejemplo al pulsar un botón, o al modificar la posición de una barra de desplazamiento)

La parte de código indicada como "OpeningFcn" es la que se ejecutará justo antes de abrir la interfaz, y es donde pondremos todas las inicializaciones.

La variable handles es una estructura que contiene todo el resto de objetos y datos de la interfaz. Si lo modificamos, antes de retornar de la función tendremos que hacer la llamada guiados (hObject, handles);

1.2 Integración del proceso de filtrado en tiempo real con una GUI

Integraréis el código diseñado en el ejercicio anterior dentro de la función `ejercicio2.m` que se os pasará. El fichero `ejercicio2.fig` contiene la parte visual de la interfaz ya programada y que no deberéis modificar.

Las funciones get y set nos permiten leer y escribir, respectivamente, las propiedades de cada uno de los objetos de la interfaz. Por ejemplo:

```
set(handles.btnStop,'userdata',flag)
```

Este es el flag que utilizaremos para activar (`flag = 0`) y desactivar (`flag = 1`) el analizador RT.

Para indicar en qué grafico se quiere pintar utilizaremos la función:

```
axes(handles.axes1);
```

Al tener tres gráficos, modificaremos el nombre axes1, axes2, axes3 respectivamente.

Todo el código del proceso de filtrado en tiempo real del apartado anterior, lo incluiremos en:

```
function btnGo_Callback(hObject, eventdata, handles)
```

La interfaz de usuario nos permite que al apretar el botón Go asociado a la variable handles.btnStop lancemos el proceso de filtrado en tiempo real de forma indefinida. Para ello, substituiréis el bucle de un tiempo de procesado máximo de 10 segundos a un bucle infinito[8]. Una manera fácil de salir del bucle infinito es la siguiente:

```
if get(handles.btnStop, 'userdata') % stop condition
break;
end
```

La línea de código para desactivar el analizador, la incluiremos en:

```
function btnStop_Callback(hObject, eventdata, handles)
```

[8] Utilizad la llamada `while true` en vez de `for`

5. Entrega

Depositad en el pozo correspondiente un único fichero ZIP por grupo con nombre:

practicapds_X.zip

siendo X el número de grupo, y que contenga el código programado (`ejercicio1.m`, `ejercicio2.m`, `ejercicio2.fig`), la señal filtrada del primer ejercicio (`testing.wav`) y el fichero PDF con vuestros diseños y reflexiones (`memoria.pdf`).

Documentación asociada al fichero PDF:

- Incluid dos apartados, uno para cada ejercicio (Ejercicio 1 y Ejercicio 2).
- Incluid en cada apartado una explicación de los desarrollos y resultados tanto teóricos como con Matlab de los diseños realizados, poniendo tanto el código Matlab usado, los gráficos que se os piden, como las explicaciones y comentarios pertinentes.

La **fecha límite de entrega de la práctica es el viernes 11 de Enero a las 23:55h**.

"En algún lugar algo increíble está esperando a ser descubierto"
-Carl Sagan-

#300LaSalle