



# **Tecnológico de Monterrey**

## **Campus Guadalajara**

### **Procesamiento digital de señales**

#### **Proyecto final: HRTF**

Luis Alfredo Aceves Astengo	A01229441
Vanya Michelle Medina Garcia	A01228016
Perla Vanessa Jaime Gaytan	A00344428

Profesor: Fernando Peña

11/06/2020

<b>Motivación.</b>	<b>3</b>
Aplicaciones	3
Video juegos.	3
Audio 3D.	3
Problemas que implica resolver.	4
<b>Objetivo.</b>	<b>5</b>
<b>Metodología.</b>	<b>6</b>
Marco teórico.	6
Implementación.	9
Diagrama general del Sistema con descripción de la interconexión entre tarjeta y PC & Diagrama de hardware.	9
Diagrama de flujo del software en el MCU.	9
Estructura del software.	9
Estructura de tramas de datos entre PC y MCU.	11
<b>Resultados.</b>	<b>13</b>
<b>Conclusiones.</b>	<b>24</b>
Luis Alfredo.	24
Perla Vanessa.	24
Vanya Michelle.	25
<b>Bibliografía.</b>	<b>26</b>

## **Motivación.**

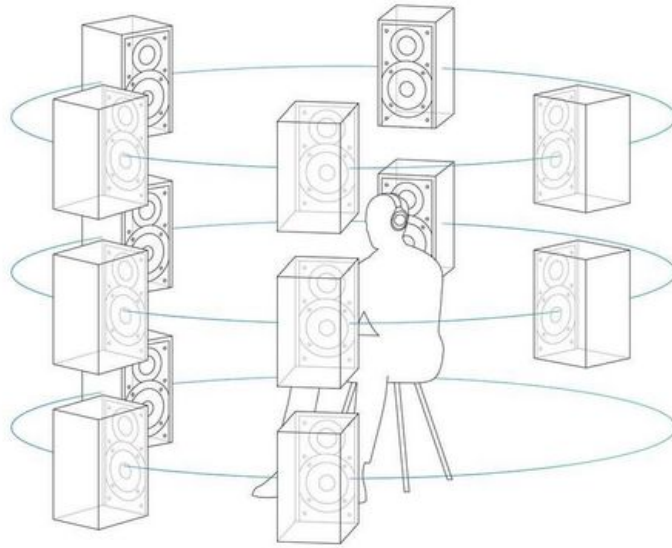
### *Aplicaciones*

#### *Video juegos.*

Una de las novedades más exitosas estos últimos años es el uso de audio 3D para los videojuegos. Uno de estos ejemplos sería el Playstation 5 de Sony. El audio 3D es una evolución del surround, lo audio envolvente, pudiendo dirigir el sonido al jugador desde cualquier dirección. Puedes hacer que el jugador escuche el audio desde atrás, desde arriba a la derecha, desde abajo a la izquierda, en su nuca, etc. Y lo bueno de esta tecnología es que lo puedes hacer con un sistema de sonido envolvente preparado, una barra de sonido compatible o, simplemente, con unos audífonos o auriculares modernos. Uno de los ejemplos que menciona Sony para promocionar el nuevo Play station con esta característica es cuando se juega Resident evil. Antes solo escuchabas el movimiento de las cadenas cuando un zombi se aproximaba y ahora puedes escuchar la dirección de donde viene.

#### *Audio 3D.*

Para la implementación de audio en sistemas 3D o Realidad Virtual se utiliza un audio binaural el cual es desarrollado por medio de filtros HRTF. Para crear este sonido estéreo en 3D es necesario otorgar al receptor la posibilidad de distinguir distintas posiciones de la fuente del sonido dentro de este espacio. Es por ello que se utilizan los filtros HRTF para modificar la señal de la fuente de sonido y de la misma manera “engañar” al cerebro sobre la localización de la fuente. Estos filtros son utilizados en Unity para la creación de videojuegos en Realidad Virtual y Realidad Aumentada.



### *Problemas que implica resolver.*

Las aplicaciones que mencionamos son un ejemplo de la necesidad del humano a sentirse en un ambiente más real. La HRTF de una persona se puede medir poniendo un set de bocinas girando alrededor de una persona utilizando pequeños micrófonos en cada uno de los oídos. Las señales de prueba son grabadas de cada una de las posiciones de las bocinas para medir el patrón de dirección espacial de los oídos. Claramente este procesos necesitan las herramientas especializadas. Es muy interesante como las aplicaciones pueden ser un videojuego, un ambiente virtual, una realidad combinada para un experiencia virtual más realista. Otra de las aplicaciones aun no muy desarrolladas, seria el uso de esta tecnología para personas con cierto tipo de discapacidad. La aplicación seria la misma pero una persona con discapacidad visual no tendría el mismo efecto que una persona sin ninguna discapacidad. Algunos juegos se pueden usar para entrenar la orientación y movimiento de niños y adolescentes con discapacidad visuales.

## **Objetivo.**

La investigación de audición espacial humana incluye tareas de audición direccional, investigaciones de rendimiento de localización, técnicas de medición y grabación, métodos de reproducción en aplicaciones de realidad virtual, etc. Encontrar la ubicación de la fuente de sonido es el problema más crítico. El objetivo de este proyecto es procesar una entrada de audio mono o ruido para procesarla en un aspecto tridimensional(3D). Esto se logra con etapas de pre-filtrado, filtrado con el uso de FIRs y un arreglo de coeficientes proporcionados por el profesor. Para lograr el efecto 3D utilizamos un microcontrolador STM32F4 programado en C y la herramienta de Matlab.

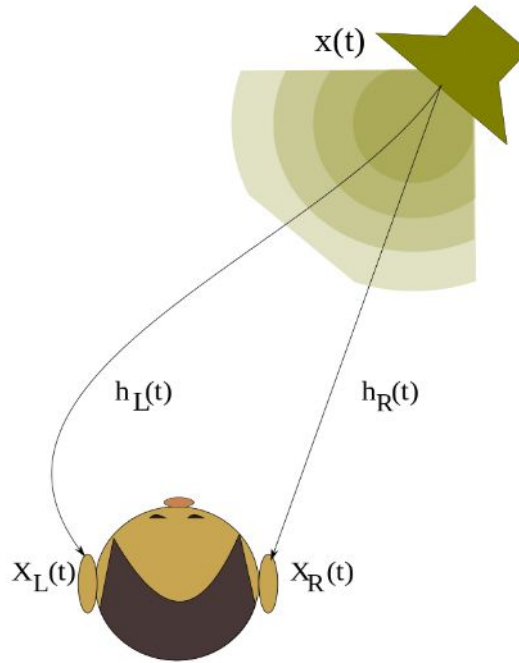
## Metodología.

### *Marco teórico.*

La función de transferencia relacionada con la cabeza o HRTF por sus siglas en inglés (*Head-related transfer functions*) captura las transformaciones que tiene una onda de sonido desde su fuente hasta nuestros oídos. El HRTF permite parametrizar el sonido para poder indicar en qué posición deberíamos de escuchar el sonido debido a que puede describir la manera en la que se filtro la entrada de sonido. La alteración que se crea por la HRTF es muy compleja y diferente en cada persona debido a que, además de depender de la frecuencia y lugar físico de la fuente, depende de cada individuo que desarrolló, durante su infancia, una HRTF propia.

El ser humano recibe las señales de audios por medio de la corteza auditiva de cada oído en señales monoaurales, después se envían a la corteza de de asociación donde son comparadas dando como resultado una señal binaural. En esta señal se logra combinar la diferencia entre el tiempo de llegada y la diferencia de intensidad. Las señales monoaurales anteriormente mencionadas provienen de la interacción entre la fuente y la anatomía del ser humano, en la cual el sonido se modifica previamente al canal auditivo donde es procesada. Todas estas modificaciones mencionadas anteriormente codifican la ubicación de la fuente de origen y puede obtenerse mediante una respuesta de impulso relacionado con la cabeza o HRIR (Head-related impulse response) la cual relaciona la ubicación del origen del sonido y la ubicación del oído con respecto a ella.

La HRTF se puede calcular mediante una transformada de Fourier de la HRIR. En la Figura 1 podemos observar la manera en que una onda de sonido se propaga desde su fuente hasta los oídos de la persona mediante notaciones convencionales para funciones de filtros.



**Figura 1.** Filtrado de una señal por medio de dos funciones de transferencia.

En la figura 1,  $x(t)$  es la presión de la fuente de sonido,  $h_L(t)$  es la respuesta al impulso para el oído izquierdo,  $h_R(t)$  es la respuesta al impulso para el oído derecho,  $x_L(t)$  es la presión del oído izquierdo y  $x_R(t)$  es la presión del oído derecho. En el dominio del tiempo, la presión de los oídos puede ser representada mediante una convolución de la señal y la HRIR por cada oído tal y como se observa en la ecuación (1). En el dominio de frecuencias, la convolución es transformada en una multiplicación tal y como se observa en la ecuación (2).

$$x_{L,R}(t) = h_{L,R}(t) * x(t) = \int_{-\infty}^{\infty} h_{L,R}(t - \tau)x(\tau)d\tau \quad (1)$$

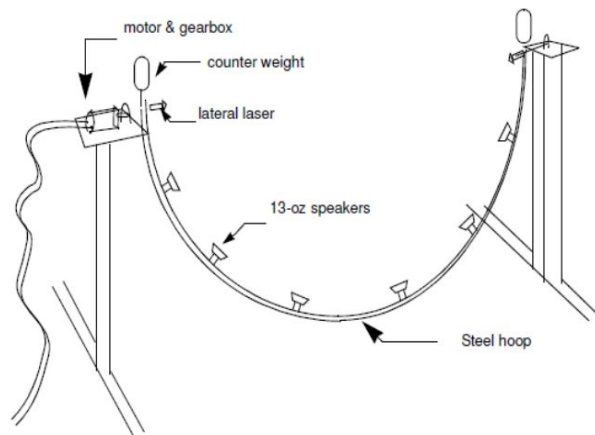
$$X_{L,R}(\omega) = F(h_{L,R}(t) * x(t)) = H_{L,R}(\omega)X(\omega) \quad (2)$$

Las mediciones del HRTF son costosas, debido a que para un funcionamiento adecuado se necesita un audio de alta calidad así como también el equipo adecuado para reproducirlo. Es indispensable utilizar una cámara anecoica para reducir la reflexión de

cualquier sonido. Para lograr la medición de la función de transferencia se produce una señal de entrada de un lugar específico y se mide la salida a la entrada de los oídos. De la ecuación (3) se puede obtener la función deseada.

$$H_{L,R}(\omega) = \frac{X_{L,R}(\omega)}{X(\omega)} \quad (3)$$

Es necesario reproducir señales de varias direcciones, como se puede mostrar en la figura 2, debido a que la HRTF varía dependiendo del ángulo de incidencia de las señales de sonido. Dichas direcciones es de donde se tienen que realizar las mediciones de la HRTF. En la figura 2 se muestran como las bocinas están montadas en un semicírculo lo cual ayuda a para una mejor medición. Al centro de este semicírculo se coloca la persona y el eje interaural se encuentra alineado con el eje de rotación. Un par de láseres se utilizan para apuntar dentro del canal del oído para asegurar que la persona no mueva su cabeza demasiado.



**Figura 2.** Semicírculo giratorio con bocinas.

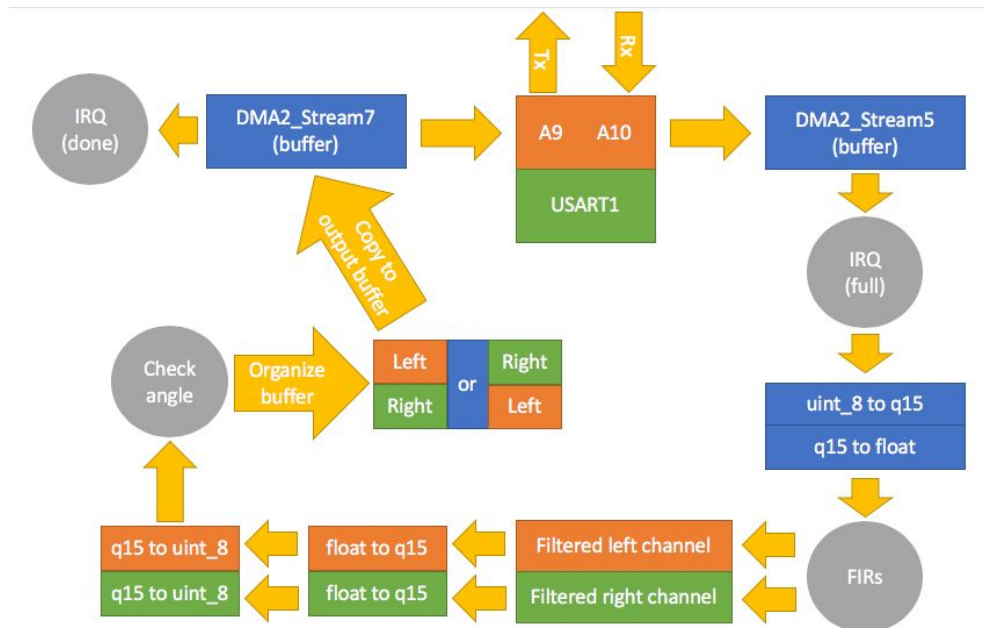


*Implementación.*

***Diagrama general del Sistema con descripción de la interconexión entre tarjeta y PC & Diagrama de hardware.***



***Diagrama de flujo del software en el MCU.***



***Estructura del software.***

Para las estructuras listadas a continuación, BUFFERSIZE = 256;

- ❖ Buffer de entrada (arreglo de datos de 8 bits): `uint8_t bufferIn[BUFFERSIZE];`
- ❖ Entrada convertida a q15 (mitad de tamaño): `q15_t q15In[BUFFERSIZE/2];`

- ❖ Entrada convertida a float (después de q15) para filtrar: `float32_t`

```
preFiltered[BUFFERSIZE/2];
```

- ❖ Arreglos para la salida de ambos filtros (derecho e izquierdo):

```
float32_t FilteredLeft[BUFFERSIZE/2];  
float32_t FilteredRight[BUFFERSIZE/2];
```

- ❖ Arreglos para convertir lo filtrado a q15:

```
q15_t q15OutLeft[BUFFERSIZE/2];  
q15_t q15OutRight[BUFFERSIZE/2];
```

- ❖ Pasar de q15 a arreglos de 8 bits (regresan a ser de tamaño BUFFERSIZE)

para poder enviar de regreso por UART:

```
uint8_t bufferOutLeft[BUFFERSIZE];  
uint8_t bufferOutRight[BUFFERSIZE];
```

- ❖ Buffer de salida (tamaño 2\*BUFFERSIZE) pues carga ambos canales:

```
uint8_t bufferOut[2*BUFFERSIZE];
```

- ❖ Inicializaciones de los filtros:

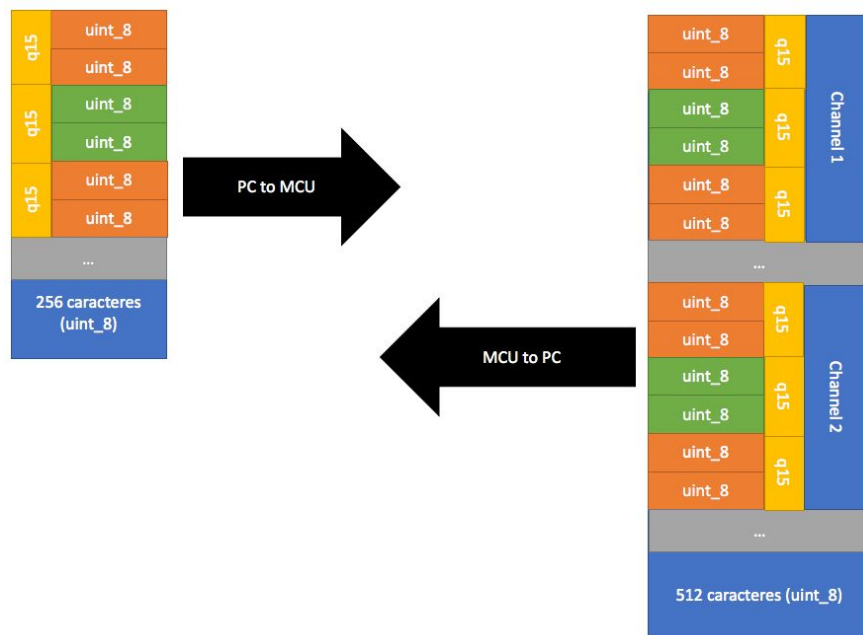
```
float32_t leftStates[BUFFERSIZE+NUMTAPS-1];  
float32_t rightStates[BUFFERSIZE+NUMTAPS-1];  
arm_fir_instance_f32 LeftFIR;  
arm_fir_instance_f32 RightFIR;  
uint32_t buffer_size = BUFFERSIZE;  
uint16_t num_taps = NUMTAPS;
```

Además de estas variables globales, tenemos funciones para:

- ❖ Inicializar FIRs (esto se hace en el main)
- ❖ Inicializar GPIOs utilizados por la UART
- ❖ Inicializar la UART
- ❖ Inicializar relojes para los módulos utilizados
- ❖ Inicializar DMA de recepción de la UART (Rx), apuntando a bufferIn
- ❖ Inicializar DMA de transmisión de la UART (Tx), apuntando a bufferOut

- ❖ Inicializar interrupciones
- ❖ Handler de DMA2\_Stream5 (buffer de recepción lleno)
  - Aquí ejecutamos todo el proceso de filtrado: convertir la entrada a q15, luego a float, llamar a las funciones de filtrar para obtener ambos canales filtrados, regresar datos filtrados a q15 y luego a float, y copiar los datos al buffer de salida para regresarlos por UART. A sugerencia del profesor, esto no debe de hacer en la interrupción; es preferible usar banderas y ejecutar en main. No hubo problema en este caso pues solo usamos 2 interrupciones.
  - Se limpia la bandera de lleno también
- ❖ Handler de DMA2\_Stream7 (transmisión de salida completa)
  - Se limpia la bandera nada más.

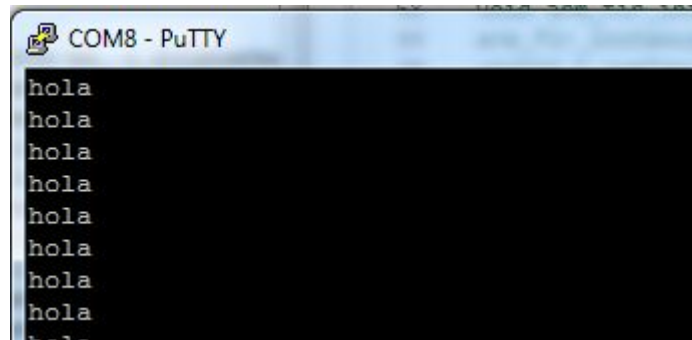
***Estructura de tramas de datos entre PC y MCU.***



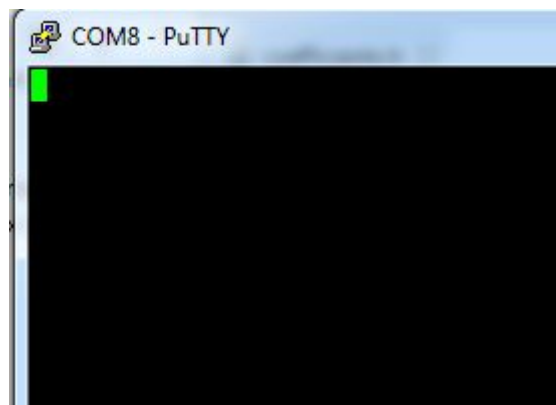
Cada trama, la PC manda 256 datos de 8 bits, que agrupados por parejas, representan un número en formato q15 (utilizamos formato Big Endian). Una vez procesados los datos, el MCU manda 512 datos de 8 bits (igual, por parejas son números en formato q15). Es el doble de tamaño de lo que recibe, pues estamos enviando los datos para ambos canales de audio (derecho e izquierdo). En matlab nos encargamos de separar estas tramas recibidas a la mitad.

## Resultados.

La complejidad de este proyecto implicaba ir agregándole cosas nuevas poco a poco. Iniciamos con lo más básico: inicializar la UART y asegurarnos que había comunicación entre el MCU y la PC:

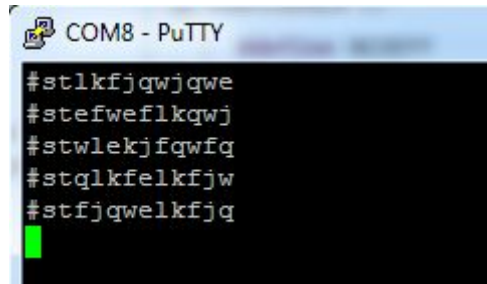


De ahí, nos interesaba subir el baudrate. Intentamos con 1800000 como el ejemplo en clase, pero fue demasiado. La terminal se veía vacía:

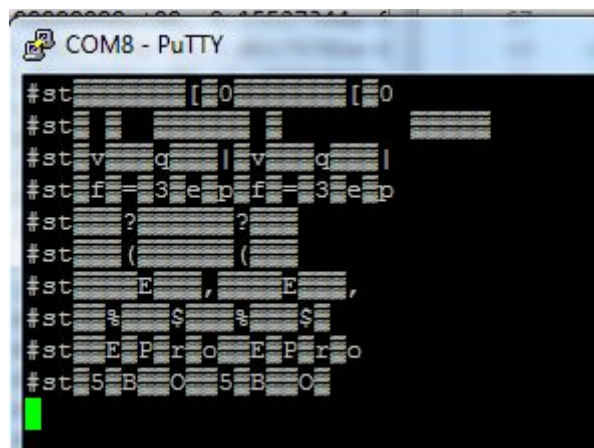


Por lo tanto se dejó en 1600000 que sí funcionaba. Una vez que se logró esto, el siguiente paso era inicializar los DMAs de recepción y transmisión para la UART. Consultando un poco de literatura en internet, se logró lo anterior. Para asegurarnos que funcionara, mandábamos unos cuantos caracteres en la terminal, y hacíamos que el código

copiara el buffer de entrada al de salida y lo enviará. Siguiendo el ejemplo del profesor, agregamos además caracteres de inicio (#st) y final (\n\r) de trama. Y tenemos lo siguiente:



El buffer de prueba era pequeño (10 caracteres), pero ya sabemos que sí funcionan ambos DMA. podemos proceder a las instancias de los filtros. Así mismo, se agregaron todas las instancias de memoria intermedias al proceso que ya se mencionaron anteriormente. Entonces ahora se esperaba de respuesta del MCU el doble de datos de entrada (20 en esta prueba) y probablemente caracteres aleatorios (no lo mismo que la entrada sino algo ya filtrado). Se obtuvo lo siguiente:

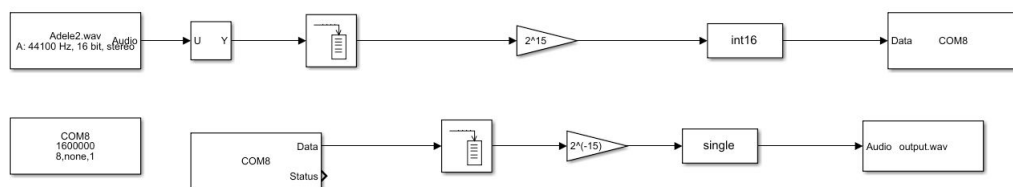


Está bien, si son bloques de 20 los que salen (10 de entrada filtrados en 2 canales). Así mismo, con el debugger podemos ver que si cambien la entrada y la salida:

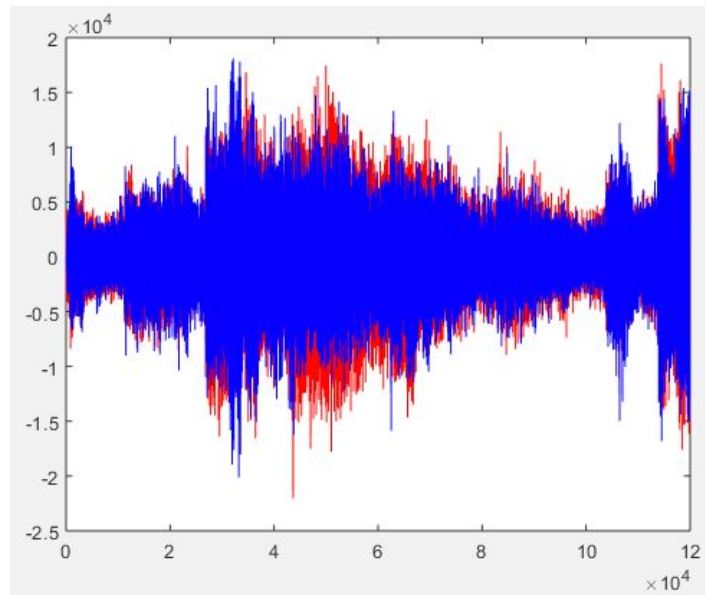
Expression	Type	Value
bufferIn	uint8_t [10]	0x2000090c <bufferIn>
(x)= bufferIn[0]	uint8_t	106 'j'
(x)= bufferIn[1]	uint8_t	97 'a'
(x)= bufferIn[2]	uint8_t	115 's'
(x)= bufferIn[3]	uint8_t	100 'd'
(x)= bufferIn[4]	uint8_t	108 'l'
(x)= bufferIn[5]	uint8_t	102 'f'
(x)= bufferIn[6]	uint8_t	107 'k'
(x)= bufferIn[7]	uint8_t	106 'j'
(x)= bufferIn[8]	uint8_t	115 's'
(x)= bufferIn[9]	uint8_t	97 'a'

bufferOut	uint8_t [20]	0x2000092c <bufferOut>
(x)= bufferOut[0]	uint8_t	239 'i'
(x)= bufferOut[1]	uint8_t	53 '5'
(x)= bufferOut[2]	uint8_t	239 'i'
(x)= bufferOut[3]	uint8_t	66 'B'
(x)= bufferOut[4]	uint8_t	239 'i'
(x)= bufferOut[5]	uint8_t	2 '\002'
(x)= bufferOut[6]	uint8_t	239 'i'
(x)= bufferOut[7]	uint8_t	79 'O'
(x)= bufferOut[8]	uint8_t	240 'ð'
(x)= bufferOut[9]	uint8_t	166 'i'
(x)= bufferOut[10]	uint8_t	239 'i'
(x)= bufferOut[11]	uint8_t	53 '5'
(x)= bufferOut[12]	uint8_t	239 'i'
(x)= bufferOut[13]	uint8_t	66 'B'
(x)= bufferOut[14]	uint8_t	239 'i'
(x)= bufferOut[15]	uint8_t	2 '\002'
(x)= bufferOut[16]	uint8_t	239 'i'
(x)= bufferOut[17]	uint8_t	79 'O'
(x)= bufferOut[18]	uint8_t	240 'ð'
(x)= bufferOut[19]	uint8_t	166 'i'

Todo parece que el filtro si está filtrando, pero no podemos comprobar esto más que haciendo los cálculos a mano, o metiendo un audio y viendo que pasa. Entonces, se preparó la siguiente estructura en Simulink:

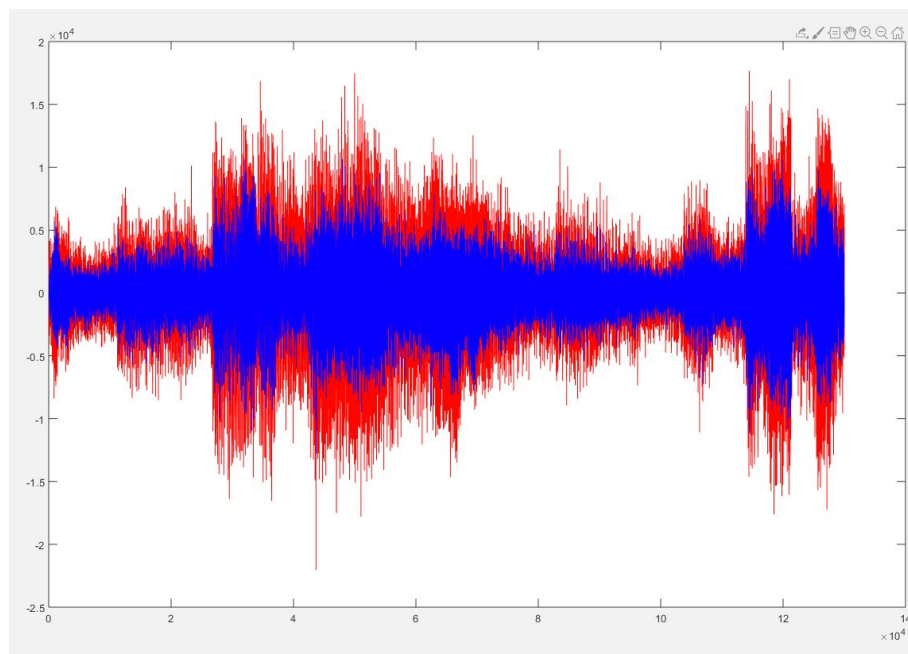


Sin embargo, la estructura no funcionaba y el profesor sugirió que lo hiciéramos en Matlab. Iniciamos probando mandar el audio y filtrando un solo canal. Se obtuvo lo siguiente:



Vemos que ambos conjuntos de datos son muy similares. Esto es esperado pues para esta prueba pusimos el ángulo 0 del filtro, lo cual no debería afectar mucho la entrada.

Además, escuchando el audio, la canción se escuchaba bien de regreso. Repetimos la prueba con un ángulo de 90. Como estamos viendo el filtro izquierdo solamente, y el 90 grados son hacia la derecha, esperamos menos potente la señal filtrada que la original:

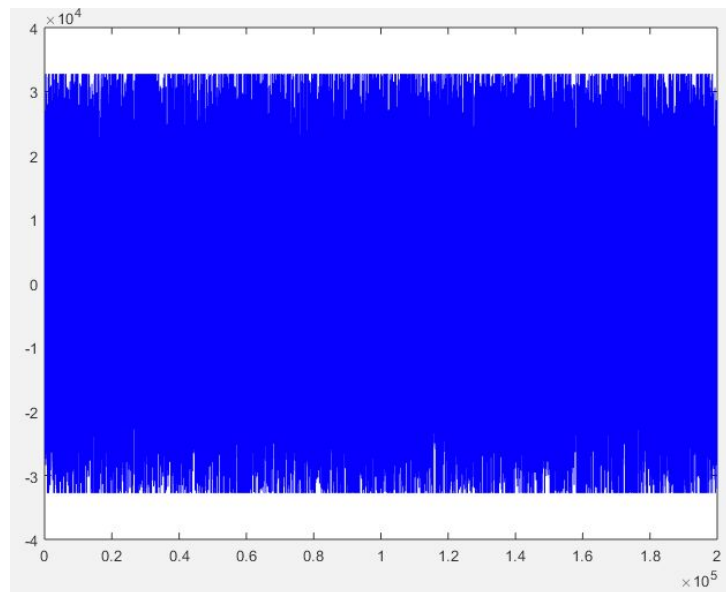




Y efectivamente, el filtrado (azul) tiene menos potencia promedio que el original, rojo. Entonces nuestro filtro si está funcionando correctamente. Es hora de incorporar ambos canales. Después de algunas pruebas y errores con Matlab a la hora de leer ambos canales, descubrimos, con ayuda del profesor, que había que reducir el tamaño del buffer de entrada hacia el micro (quedó de tamaño 256), para que lo que recibiera la computadora no fuera un buffer tan grande (recordemos que recibe el doble de lo que manda por ser 2 canales). Cabe mencionar que entre los intentos que se hicieron cuando esto no funcionamos, intentamos mandar canal derecho e izquierdo por separado, es decir, llenar el buffer de salida con uno, mandarlo, llenar con el otro y mandarlo. Sin embargo al final se optó por enviar un bloque tras el otro en un buffer de doble tamaño y hacer la separación con código de matlab. Entonces a partir de aquí solo se empezaron a probar varios ángulos (ahora utilizando ruido blanco para que fuera más perceptible el cambio al oído.

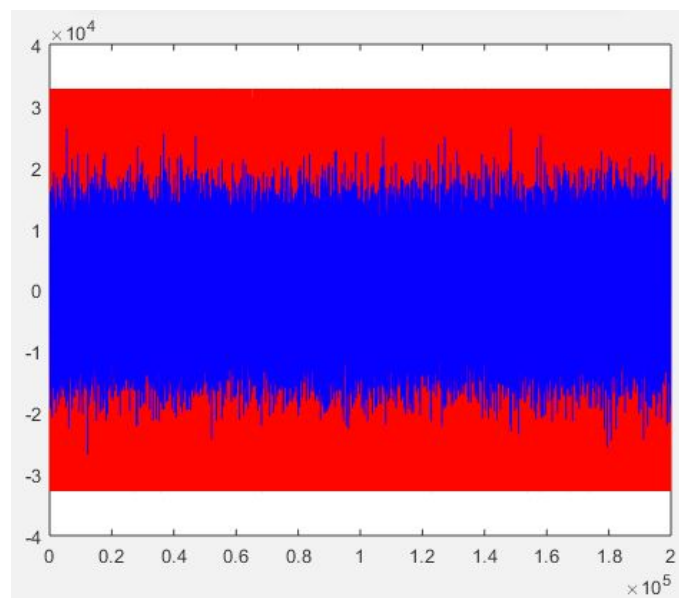
A partir de ahora graficamos el canal derecho (en rojo) contra el izquierdo (azul), para observar el efecto del filtro. Conforme aumentamos el ángulo de 0 a 180, el filtro debe hacer que se escuche más hacia la derecha, entonces debería dominar el color rojo. Así mismo, se calculará el error cuadrático medio entre ambos canales (utilizando el comando:  $\text{sum}([\text{readtemp2}-\text{readtemp}].^2)/\text{numel}(\text{readtemp2})$ , donde `readtemp` y `readtemp2` son los arreglos con las muestras del canal izquierdo y derecho, respectivamente). Aquí lo que se espera es que el error sea 0 en ángulos 0 y 180, sea el máximo a 90, crezca de 0 a 90 grados y disminuya de regreso de 90 a 180 grados. Analizamos los siguientes casos:

0 grados:



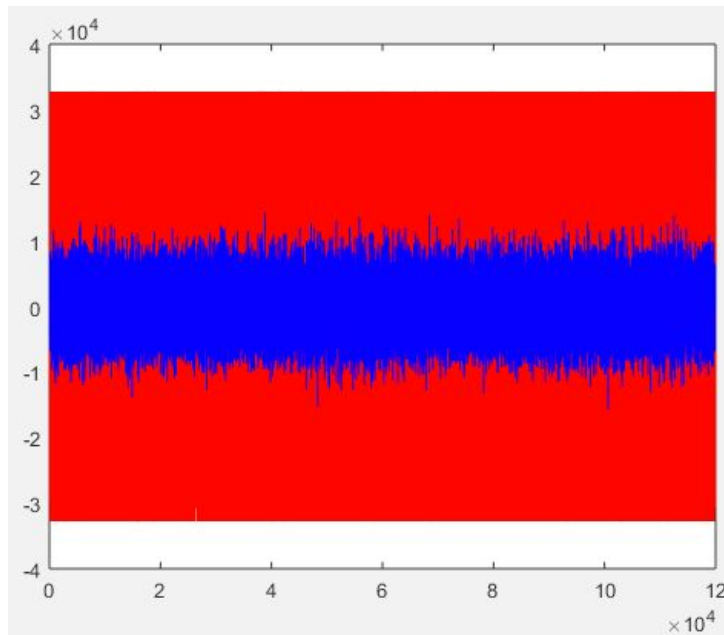
ECM=0

30 grados:



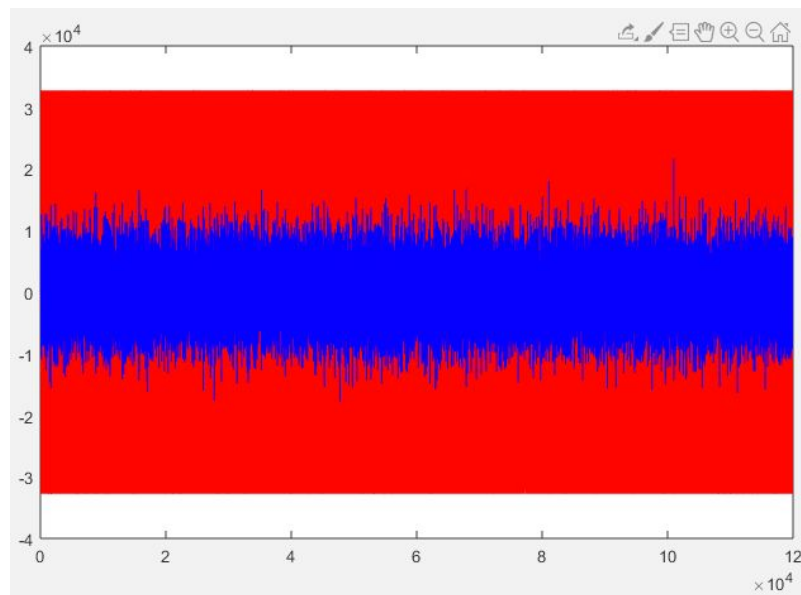
ECM=3.2204e+08

60 grados:



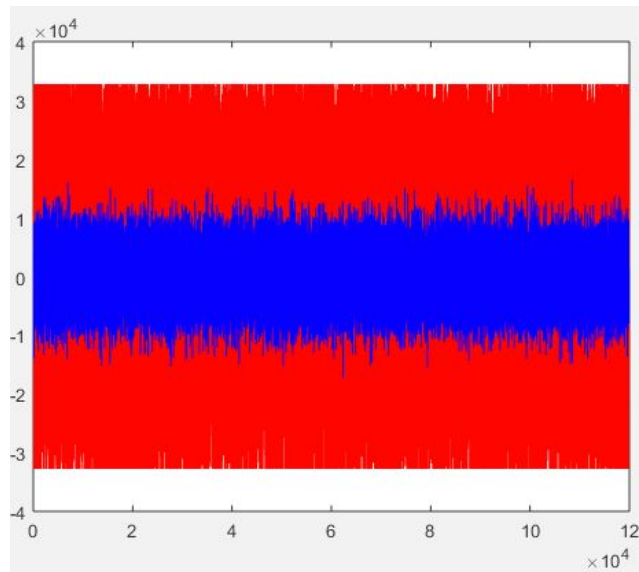
ECM=2.9647e+08

90 grados:



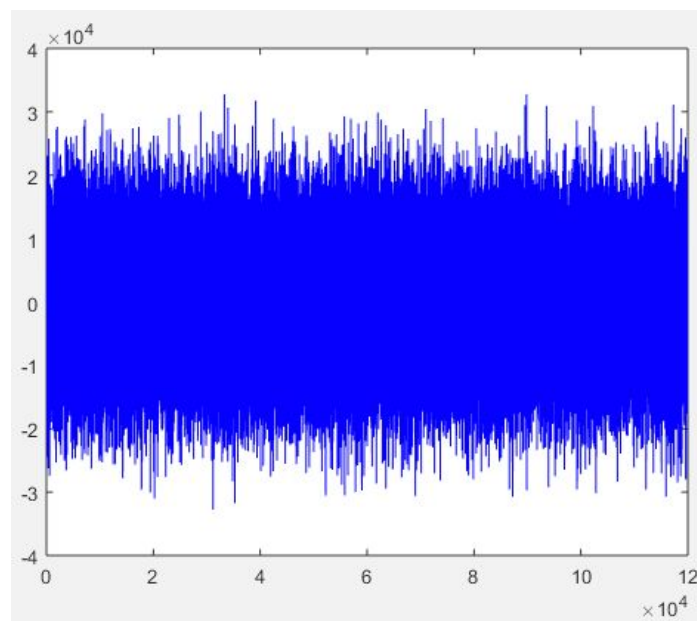
ECM=3.7277e+08

135 grados:



ECM=1.9390e+08

180 grados:



ECM=0

Nos llevamos un par de sorpresas al ver estos resultados. Al llegar al 90 la diferencia en la gráfica no es tan contrastante con los 60 o 135 grados, pero en el error si vemos que es máximo a 90 grados. También sorprende que en 30 grados encontramos más error que en 60. Parece que es un error de cálculo, pues para el 30 tomamos más muestras que para los demás; ya después de ahí fueron uniforme las mediciones. Y en 0 y 180 encontramos error 0 como se esperaba. A pesar de todo, al reproducir continuamente un audio tras otro, si se notaba el efecto de movimiento alrededor de la cabeza.

Finalmente, se cambió el código para que el efecto funcionara en 360 grados. Se logró con un par de modificaciones sencillas:

```
#define angle 0

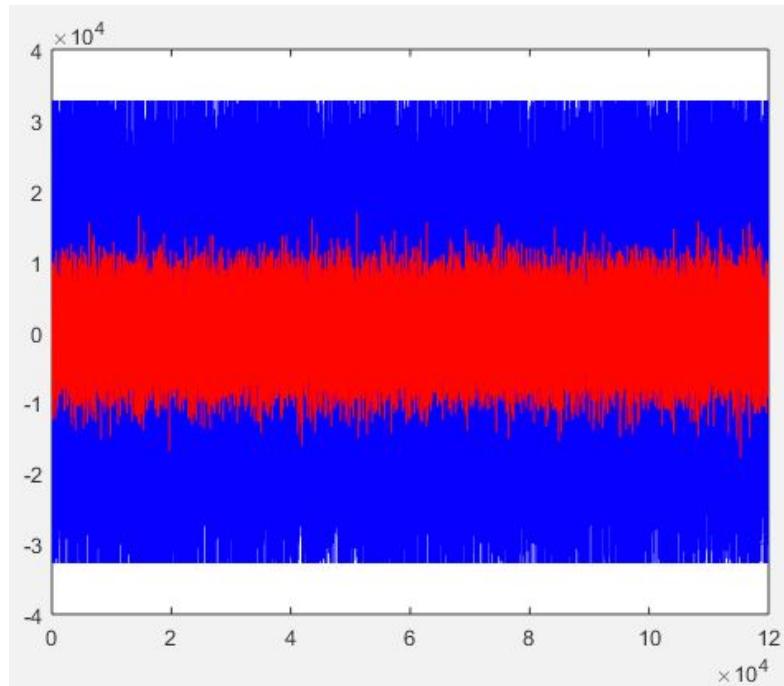
#if angle>180
    #define coefficient 128*(360-angle)/5
#else
    #define coefficient 128*angle/5
#endif

if(angle>180){
    for(int x=0; x<BUFFERSIZE/2; x++){
        bufferOut[2*x]=(q15OutRight[x]>>8);
        bufferOut[2*x+1]=(q15OutRight[x]&0xFF);
    }
    for(int y=0; y<BUFFERSIZE/2; y++){
        bufferOut[BUFFERSIZE+2*y]=(q15OutLeft[y]>>8);
        bufferOut[BUFFERSIZE+2*y+1]=(q15OutLeft[y]&0xFF);
    }
}
else{
    for(int x=0; x<BUFFERSIZE/2; x++){
        bufferOut[2*x]=(q15OutLeft[x]>>8);
        bufferOut[2*x+1]=(q15OutLeft[x]&0xFF);
    }
    for(int y=0; y<BUFFERSIZE/2; y++){
        bufferOut[BUFFERSIZE+2*y]=(q15OutRight[y]>>8);
        bufferOut[BUFFERSIZE+2*y+1]=(q15OutRight[y]&0xFF);
    }
}
```

(si el ángulo es mayor a 180, voltemos los filtros).

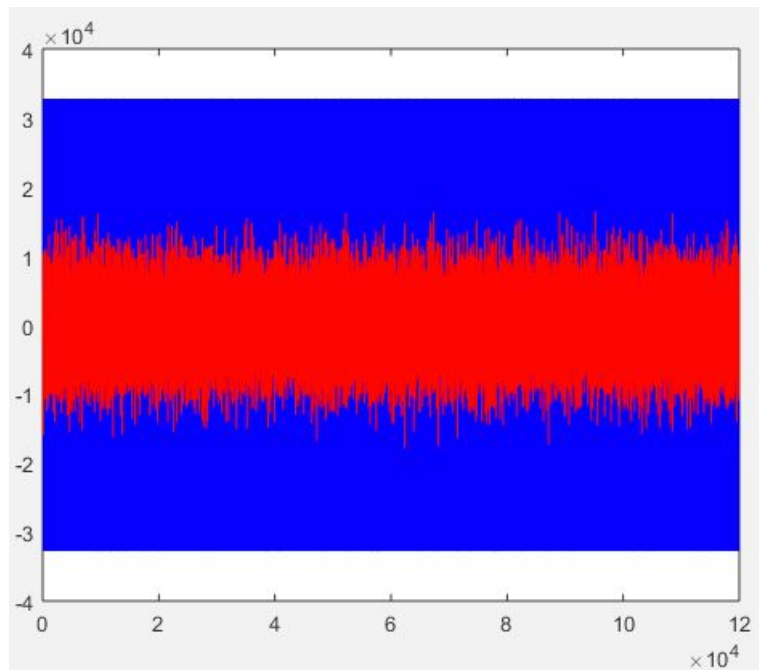
Los resultados para mediciones de los ángulos arriba de 180 son los siguientes:

225 grados:



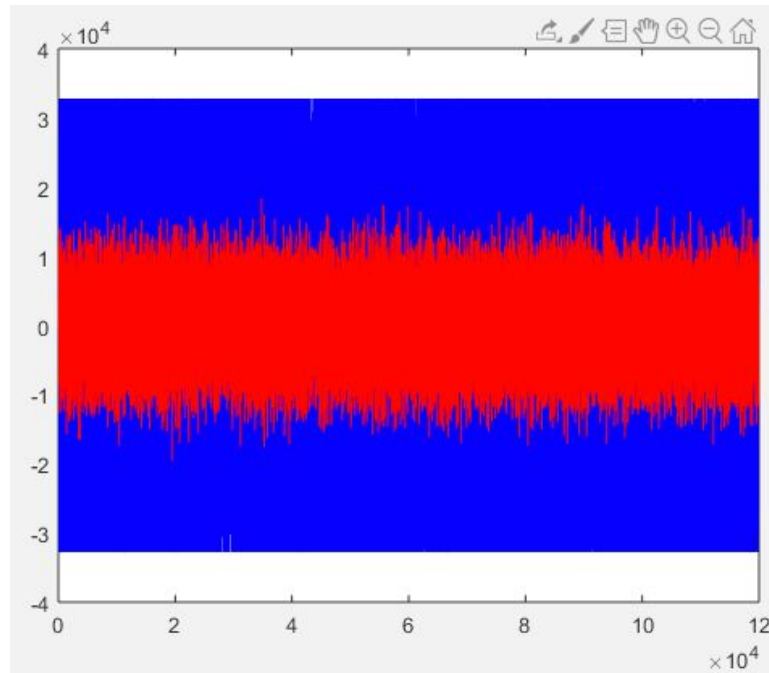
ECM= 1.9234e+08

270 grados:



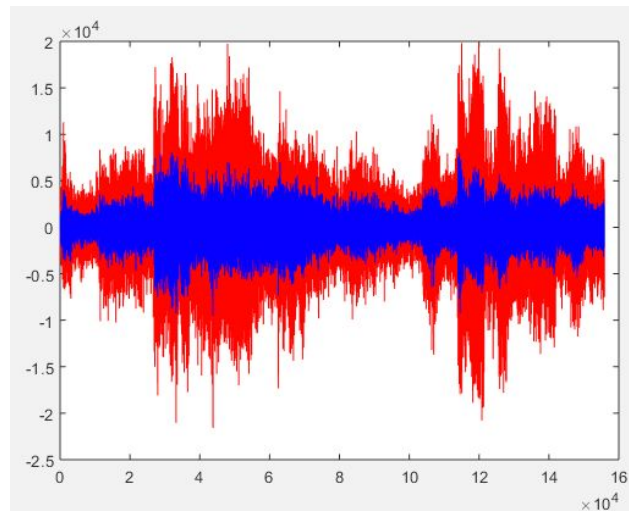
ECM=3.7527e+08

315 grados:



ECM= 2.5864e+08

Notamos que ahora predomina el canal izquierdo (azul) y en el 270 hay máximo error entre los canales (porque efectivamente es el lugar más alejado del derecho). Nuevamente se juntaron todos los fragmentos a diferentes ángulos en orden y pudimos escuchar como el ruido blanco daba la vuelta respecto a nuestra cabeza. Finalmente, hicimos una prueba extra con una canción, para ver que no fuera suerte que todo estuviera saliendo bien con el ruido blanco. La pusimos a 135 y si lo escuchamos atrás hacia nuestra derecha. La gráfica, nuevamente, lo respalda (derecha rojo, izquierda azul):



## Conclusiones.

*Luis Alfredo.*

Este proyecto me pareció muy interesante. Muchas veces en la música se nota el efecto de que algunas cosas suenan en un canal y otras en el otro, pero rara vez he escuchado el efecto que quisimos replicar aquí: que el sonido venga de muchas direcciones. La verdad impresiona mucho ya que lo escuchas y vas cambiando la dirección. Definitivamente fue un reto implementarlo en un microcontrolador. Hubo que ir superando pequeñas metas de funcionalidad poco a poco hasta ir logrando lo que queremos con el código. Cabe destacar que gran parte de nuestros problemas fue conectar con Simulink/Matlab, porque una vez que lo lográbamos nos dábamos cuenta que el código servía bien. Sin duda es un proyecto retador, pero está padre plasmar los contenidos del curso en algo con aplicación real.

*Perla Vanessa.*

Con este proyecto pude entender mucho mejor los filtros y su funcionalidad. Fue muy interesante ir descubriendo los pequeños detalles del código y su implementación sobre la marcha. Este proyecto fue un poco desafiante ya que fue algo que nos tomó varios días



realizarlo y no sabíamos cómo resolver unos pequeños detalles de su funcionalidad. Creo que al final pude aprender la finalidad de estos tipos de filtros y cómo se pueden utilizar con distintos objetivos. Antes de esta clase no tenía conocimiento sobre cómo manipular las distintas señales y fue muy enriquecedor irlo descubriendo conforme fue avanzando el curso.

*Vanya Michelle.*

Con este proyecto logre entender el funcionamiento de un HRTF. Las nuevas tecnologías empezaron a utilizar esta evolución de audio. Al principio fue un poco difícil para mi entender la configuración del DMA. Pero una vez teniendo las variables inicializadas para cada una de las etapas de prefiltrado y filtrado utilizamos un método fácil para convertir de q15 a float y viceversa para cada uno de los buffers de cada lado. Estuvo retador conectarlo a simulink por el hecho del tiempo real y al final se utilizó matlab. En la salida de los audios se puede apreciar los ángulos del sonido. Se me hizo muy interesante crear y poner en práctica un proyecto de este estilo, el cual se puede percibir en alguna canción de electrónica. Y claramente fue un resumen de filtros y procesamiento de señales vistos en clase.

## **Bibliografía.**

¿Qué es una cámara anecoica? (2018). INFAIMON. Recuperado de:

<https://blog.infaimon.com/la-camara-anecoica/>

Alonso, R. (2020). *HRTF, la tecnología que permite escuchar audio posicional en juegos.*

HardZone. Recuperado de:

<https://hardzone.es/reportajes/que-es/hrtf-tecnologia-audio-posicional/>

Ayora, M. (2014). *Implementación de sonido binaural para aplicaciones de realidad aumentada.* Universidad Zaragoza. Recuperado de:

<https://zaguan.unizar.es/record/15298/files/TAZ-TFG-2014-1038.pdf>

Crespo, M. (2017). *¿Cómo ubicamos los sonidos en el espacio? HRTF y otros indicios.*

hispasonic. Recuperado de:

<https://www.hispasonic.com/tutoriales/sonidos-espacio-hrtf-otros-indicios/43298>

Potisk, T. (2015). *Head-Related Transfer Function.* University of Ljubljana: Faculty of Mathematics and Physics. Recuperado de:

<https://pdfs.semanticscholar.org/2397/14fced9554364fb1dfdd2cd071a89f72bcd5.pdf>

UART with DMA mode (2012). *ST Community.* Recuperado de

<https://community.st.com/s/question/0D50X00009XkZeLSAV/uart-with-dma-mode>

¿Qué es el audio 3D de PlayStation 5 y cómo podría afectar a los videojuegos?. (2020).

Retrieved 11 June 2020, from

[https://neox.atresmedia.com/games/noticias/actualidad/que-es-el-audio-3d-de-playstation-5-y-como-podria-afectar-a-los-videojuegos-video\\_201904185cb843800cf2bb473d5d2b85.html](https://neox.atresmedia.com/games/noticias/actualidad/que-es-el-audio-3d-de-playstation-5-y-como-podria-afectar-a-los-videojuegos-video_201904185cb843800cf2bb473d5d2b85.html)

Spatial Audio - Microsoft Research. (2020). Retrieved 11 June 2020, from

<https://www.microsoft.com/en-us/research/project/spatial-audio/>