UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DETECCIÓN AUTOMÁTICA DEL RETARDO EN LA FUNCIÓN DE TRANSFERENCIA RELACIONADA CON LA CABEZA

GRADO EN INGENIERÍA DE
SISTEMAS ELECTRONICOS

DAVID RODRIGUEZ LOBACO
MÁLAGA, 2022

E.T.S. DE INGENIERÍA DE TELECOMUNICACIÓN, UNIVERSIDAD DE MÁLAGA

DETECCIÓN AUTOMÁTICA DEL RETARDO EN LA FUNCIÓN DE TRANSFERENCIA RELACIONADA CON LA CABEZA

Autor: David Rodriguez Lobaco

Tutor: Luis Molina Tanco

Cotutor: María Cuevas Rodríguez

Departamento: Tecnología Electrónica (TE)

Titulación: Grado en Ingeniería de Sistemas Electrónicos

Palabras clave: 3D, Audio, BRIR, C++, HRIR, HRTF, ILD, ITD, MATLAB, OpenFrameworks, *Outlier*, Retardo, Toolkit, Umbral.

Resumen

En la última década se han llevado a cabo gran cantidad de estudios, proyectos y avances en el área del audio espacial. Uno de estos proyectos fue el proyecto europeo 3D Tune-In, donde nació el *software* 3D Tune-In Toolkit, un *software* de código abierto con la intención de ayudar a otros con su funcionalidad de simulación binaural.

Para su funcionamiento se necesita de un archivo que contenga la HRTF (*Head-Related Transfer Function*) de una persona y que estén definidos los retardos de las HRIR (*Head-Related Impulse Response*) a cada canal auditivo. En este Trabajo Fin de Grado se ha estudiado la utilidad de distintos métodos para comprobar si permiten calcular los retardos de las HRIR de forma automática. Además, se ha añadido una nueva funcionalidad al 3D Tune-In Toolkit que permite calcular estos retardos, y se ha desarrollado una aplicación interactiva de simulación binaural que permite probarla, entender su alcance y sus parámetros.

Los desarrollos realizados pueden servir para seguir ampliando las funcionalidades de la librería 3D Tune-In Toolkit, en la que participó DIANA (Diseño de Interfaces AvaNzAdos), grupo de investigación de la ETSIT (Escuela Técnica Superior de Ingeniería de Telecomunicación), en colaboración con Imperial College London.

E.T.S. DE INGENIERÍA DE TELECOMUNICACIÓN, UNIVERSIDAD DE MÁLAGA

AUTOMATIC DETECTION OF DELAY IN HEAD-RELATED TRANSFER FUNCTION

Author: David Rodriguez Lobaco

Supervisor: Luis Molina Tanco

Co-supervisor: María Cuevas Rodríguez

Department: Electronic Technology (ET)

Degree: Degree in Electronic Systems Engineering

Keywords: 3D, Audio, BRIR, C++, Delay, HRIR, HRTF, ILD, ITD, MATLAB, Open-Frameworks, Outlier, Threshold, Toolkit.

Abstract

In the last decade, a large quantity of studies, projects and advances have been carried out in the area of spatial audio. One of these projects was the European 3D Tune-In project, where the 3D Tune-In Toolkit software was born, an open-source software with the intention of helping others with its binaural simulation functionality.

For its operation, the file that contains the HRTF of a person requires the HRIR delays to be present for each auditory channel. In this Final Degree Project, we have studied the suitability of different methods for the automatic calculation of HRIR delays. In addition, a new functionality has been added to the 3D Tune-In Toolkit that allows these delays to be calculated, together with an interactive binaural simulation testing application that allows understanding the scope and parameters of this functionality.

The developments made can be used to continue expanding the functionalities of the 3D Tune-In Toolkit library, a open software initiative suppoerted by DIANA, a research group from ETSIT Málaga, in collaboration with Imperial College London.

Dedicado a todas aquellas
personas que amplían mi mundo.

David

Agradecimientos

Por una parte agradecer a las personas que me han ayudado de manera directa con el TFG:

A mi tutor, Luis Molina, a María Cuevas, Arcadio Reyes, y Daniel Toledo. Por enseñarme tantas cosas, su paciencia y su ayuda para hacerme entender el funcionamiento interno del Toolkit. Y a Pablo Guerrero por haber intentado ayudarme con algunos de los problemas.

En cuanto al apoyo moral y efectivo:

A mi familia, por apoyarme en todo momento.

A mis compañeros de carrera: Salvador, Manuel, Daniel, Adil y Ramón, por todos los buenos momentos durante la carrera y su ayuda a lo largo de ella.

Acrónimos

AES	*Audio Engineering Society*
ARI	*Acoustics Research Institute*
BRIR	*Binaural Room Impulse Response*
DIANA	Diseño de Interfaces AvaNzAdos
ETSIT	Escuela Técnica Superior de Ingeniería de Telecomunicación
HRIR	*Head-Related Impulse Response*
HRTF	*Head-Related Transfer Function*
ILD	*Interaural Level Difference*
ITD	*Interaural Time Difference*
SADIE	*Spatial Audio for Domestic Interactive Entertainment*
SOFA	*Spatially Oriented Format for Acoustics*
SRIR	*Spatial Room Impulse Response*
TFG	Trabajo Fin de Grado
UMA	Universidad de Málaga

Índice

Índice de figuras

Índice de Tablas

Parte I

Introducción

Capítulo 1

Introducción y visión general

Contenido

Sinopsis

Este capítulo introduce los objetivos de este **Trabajo Fin de Grado** (TFG) y su contexto, así como la estructura del resto de este documento.

1.1. Conceptos básicos

La diferencia de tiempo interaural **ITD** (*Interaural Time Difference*) es la diferencia de tiempo de llegada de un sonido entre el sistema auditivo derecho e izquierdo de una misma persona. Esta diferencia de tiempo nos da información sobre la localización de una fuente sonora. Cuando se produce un sonido en el plano horizontal, el azimut (o acimut) es el ángulo entre la fuente y el oyente. Una fuente sonora enfrente del oyente se encuentra a 0° de azimut, a la izquierda se encuentra a 90°, a la derecha 270° y en la espalda a 180°. Esta es la relación de

ángulos que se usa en el proyecto, pero puede haber otros en los que el ángulo del oído derecho e izquierdo se intercambien. Por otro lado, la posición en el eje vertical se denomina elevación. Si el sonido se encuentra justo enfrente, la elevación es de 0°, a medida que la fuente sonora se eleva por encima de la cabeza se vuelve positiva, hasta 90°, por el contrario, al descender se vuelve negativa hasta -90°.

La diferencia de nivel interaural **ILD** (*Interaural Level Difference*) es la diferencia de intensidad que hay en un mismo sonido al llegar a ambos oídos. A medida que el sonido viaja por un medio, en este caso, generalmente, el aire, su fuerza se disipa. Si nos colocamos de perfil a una fuente sonora, el sonido se escucha más elevado en el lado incidente que en el lado opuesto. Esto es debido a que la cabeza proyecta una sombra acústica en el lado opuesto del que proviene el sonido, cambiando el volumen y la distribución de frecuencias. Esta sombra es más prominente para los sonidos de alta frecuencia que para los de baja frecuencia. [3]

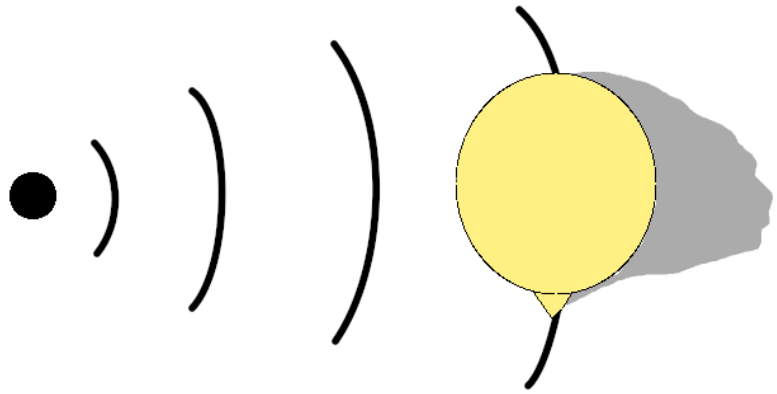


Figura 1.1: Sombra acústica proyectada por el sonido y la cabeza

La función de transferencia relacionada con la cabeza **HRTF** (*Head-Related Transfer Function*) es una respuesta que caracteriza cómo cada canal auditivo percibe un sonido desde distintos puntos del espacio. A medida que el sonido llega al oyente, se ve afectado por las características de la cabeza del sujeto. Tales cómo el tamaño, la forma de la cabeza, las orejas, el canal auditivo... afectan al sonido, aumentando algunas frecuencias y atenuando otras. Es decir, la HRTF está compuesta por indicios binaurales (ITD e ILD) e indicios monoaurales, que son los mencionados antes.[16]

Las HRTF constituyen el elemento básico de la simulación binaural. Es el conjunto de **HRIR** (*Head-Related Impulse Responses*), que son las respuestas

al impulso en el dominio del tiempo. Se almacenan en formato ***sofa*** (Spatially Oriented Format for Acoustics); un formato de archivo usado para almacenar datos acústicos orientados espacialmente, como el mencionado antes, las **BRIR** (*Binaural Room Impulse Response*) y las **SRIR** (*Spatial Room Impulse Response*); y que ha sido estandarizado por la **AES** (*Audio Engineering Society*) como AES69-2015. Elegir una HRTF que no sea idéntica o muy similar al del oyente, cuando se simula un entorno virtual, suele provocar inexactitud a la hora de intentar percibir la procedencia de un sonido. [15]

La forma de recoger los datos para la creación de una HRTF es introducir dos micrófonos al inicio del canal auditivo de una persona o maniquí; situar al sujeto bajo prueba en una sala anecoica y usar un altavoz para emitir sonidos que serán recogidos por los micrófonos. Para simular los distintos puntos del espacio, se recurre a mover el altavoz verticalmente para la elevación y girar al sujeto para el azimut. También cabe la posibilidad de que sea el altavoz el que se mueve por toda la sala y sea el sujeto el que está quieto.[16]

Esto tiene diferentes tipos de costes, como son el económico, para comprar el equipo necesario, de espacio, tanto físico para la instalación del equipo como digital para guardar los datos, y de tiempo para obtener las muestras necesarias. Lo cual explica el reducido número de base de datos y de las HRTF de las que se componen las mismas. Debido a esta limitación, en cuanto al número de funciones de transferencias disponibles, es que se recurre a la interpolación para simular la espacialización de una fuente sonora situada en un punto en el que no se ha podido medir la HRTF del oyente [15]. Sin embargo, la interpolación de las HRTF no es un proceso trivial, como se verá en este trabajo.

1.2. Objetivo

En el año 2018, el grupo de investigación **DIANA** (Diseño de Interfaces Avanzados) de la Universidad de Málaga, en colaboración con otros socios europeos (principalmente Imperial College London), publicó el software **3D Tune-In Toolkit**[1], uno de los resultados principales del proyecto europeo 3D Tune-In, que nació con la intención de ayudar a empresas y usuarios de audífonos y educar sobre la pérdida auditiva mediante el uso de sonidos, imágenes y juegos. Es una herramienta de código abierto, multiplataforma y desarrollada en *C++*. La librería ofrece múltiples algoritmos para la simulación de audio espacializado binaural con una gran configurabilidad. Esto le permite ser utilizado en múltiples experimentos de psicoacústica virtual.

[1] <https://github.com/3DTune-In/3dti_AudioToolkit>

El objetivo de este trabajo es automatizar el cálculo de los retardos de las señales de las HRTF, que pueden estar ausentes en las bases de datos, mediante la adición de funcionalidades a uno de los módulos, o la creación de un nuevo modulo, dentro del Toolkit. La necesidad de esta función se debe a que el Toolkit espera que la HRTF cargada tenga cada función de transferencia con el retardo inicial quitado, es decir, que esté almacenado por separado la función de transferencia y el ITD. De este modo puede realizar la interpolación evitando el efecto de filtrado de peine (*comb filtering*), del cual se hablará más adelante.

El objetivo de este TFG es contribuir al 3D Tune-In Toolkit, que es una librería de *C++*. No obstante, el proyecto se acometerá inicialmente en MATLAB, donde ya hay implementadas algunas funciones para trabajar con archivos *sofa*, como pueden ser las funciones para obtener el ITD. De esta forma, el estudio previo del uso de filtros con distintas características, distintas base de datos y otras propiedades se puede complementar con recursos visuales, permitiendo una mayor comprensión y visualización de los problemas y sus posibles soluciones antes de su programación en *C++* y su integración en el 3D Tune-In Toolkit.

1.3. Estado del arte

En la actualidad, aunque pueda parecer que el campo de investigación y desarrollo relacionado con el sonido no tiene mucha importancia, no es así. Hay diversos estudios y proyectos en desarrollo, como el articulo *Interaural Time Difference Prediction Using Anthropometric Interaural Distance* [10] publicado en octubre de este año. En él se estudia la viabilidad de mejorar los modelos de cabeza esférica, como el de Woodworth (se habla de él más adelante), al predecir el ITD conociendo la distancia interaural antropométrica personal. Esto permite obtener un ITD personalizado sin la necesidad de medir los HRIR y su uso en aplicaciones de audio envolvente.

1.3.1. La interpolación de las HRIR y el Toolkit 3D-Tune In

Como se ha contado anteriormente, en el proyecto 3D Tune-In se crearon múltiples aplicaciones, a las que dio soporte el kit de herramientas **3D Tune-In Toolkit**. En esta biblioteca, la espacialización binaural se lleva a cabo a través de convolución con funciones de transferencia relacionadas con la cabeza (HRTF) y respuestas de impulso de sala binaural (BRIR). También se añadió funcionalidades como simulación de fuente de campo cercano o lejano, personalización de diferencias de tiempo interaural, reverberación, perdida auditiva simulada a

través de filtros de tonos gamma y expansores/compresores multibanda, que incluyen características no lineales avanzadas, como la difusión de frecuencia y distorsión temporal; entre otros[5].

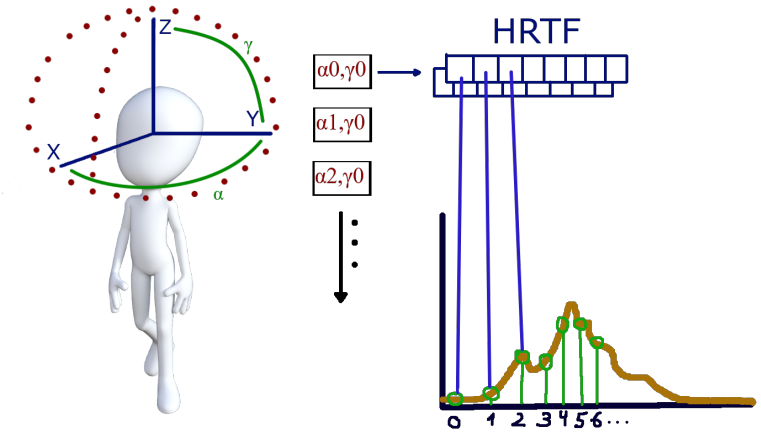


Figura 1.2: Representación gráfica de la información que recibe ToolKit

El funcionamiento básico del Toolkit se puede ver representado en la figura 1.2 y es el siguiente:

Se elige una HRTF de la base de datos. Uno de los campos es una matriz con los azimuts y elevaciones de las que se compone la HRTF (figura 2.8b, página 21), cuyas posiciones están relacionadas con el campo en el que se almacenan las HRIR. El campo de la HRIR está formado por dos columnas, una columna contiene las respuestas al impulso del oído derecho y la otra la del oído izquierdo. Cada posición contiene un *array* con los datos discretizados de la señal. Posteriormente, en el programa, se crea un oyente con una posición determinada; lo mismo con los sonidos que queramos generar. Ya que las HRTF no están discretizadas para todos los puntos del espacio, si un sonido no se encuentra en uno de estos puntos se interpola usando los 3 puntos más cercanos. De este modo se consigue simular bastante bien como oiríamos un objeto real en ese punto.

Existe una aplicación descargable[2] denominada BiTA (Binaural Test Application) que permite probar la mayoría de la funcionalidad del Toolkit (salvo la que

[2] <https://github.com/3DTune-In/3dti_AudioToolkit/releases>

se refiere a la espacialización basada en altavoces). Su aspecto puede verse en la Figura 1.3.

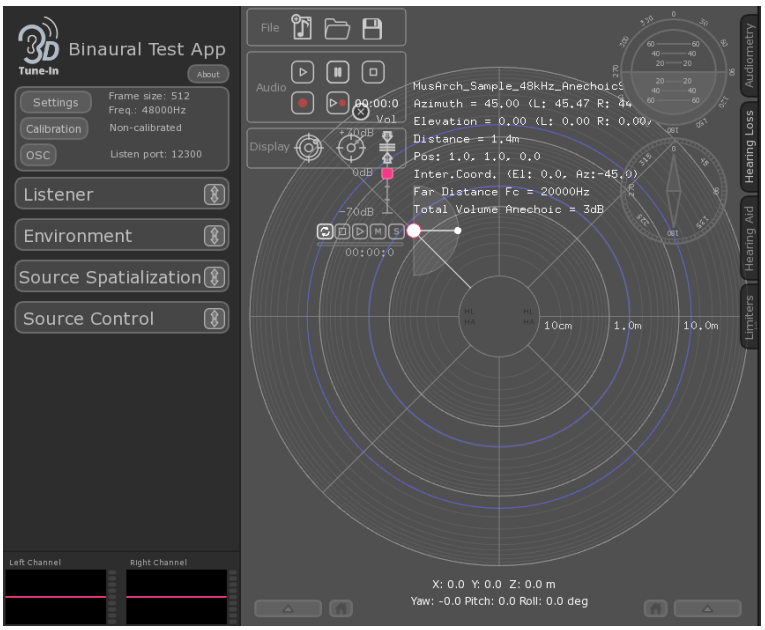


Figura 1.3: Aspecto del interfaz de usuario de la aplicación BiTA, que permite probar el Toolkit 3D Tune-In [5]

Para llevar a cabo la interpolación es necesario separar la parte que contiene el retardo y la que contiene la señal en las HRIR. La finalidad es evitar el *comb filtering*, que ocurre cuando un sonido se acopla a sí mismo con una pequeña diferencia de tiempo, normalmente causado por una reflexión del sonido en una superficie, o por dos o más micrófonos recogiendo el mismo sonido desde distintas posiciones. En este caso se debe al agregar dos o más señales de audio con magnitud similar pero fase diferente, lo que hace que la señal resultante se amplifique o se atenúe a ciertas frecuencias. De esta manera, la respuesta de frecuencia presenta una serie de valles espaciados regularmente, dando la apariencia de un peine. Esto provocaría un efecto de sonido más robótico en algunos casos, reduciendo la calidad del renderizado. [13] [4]

En la figura 1.4 se puede observar las pruebas realizadas sobre las HRIR del canal auditivo izquierdo y derecho a 45° de azimut y 0° elevación. La prueba consiste en eliminar una HRIR de la base de datos y recuperarlo mediante la interpolación de las HRIR a los que se le ha extraído el retardo inicial (HRIR alineadas) y HRIR a los que no se les ha extraído el retardo inicial (HRIR no alineadas). La linea de color azul representa la HRIR original, La roja segmentada muestra el resultado de la interpolación utilizando HRIR no alineadas, y la verde

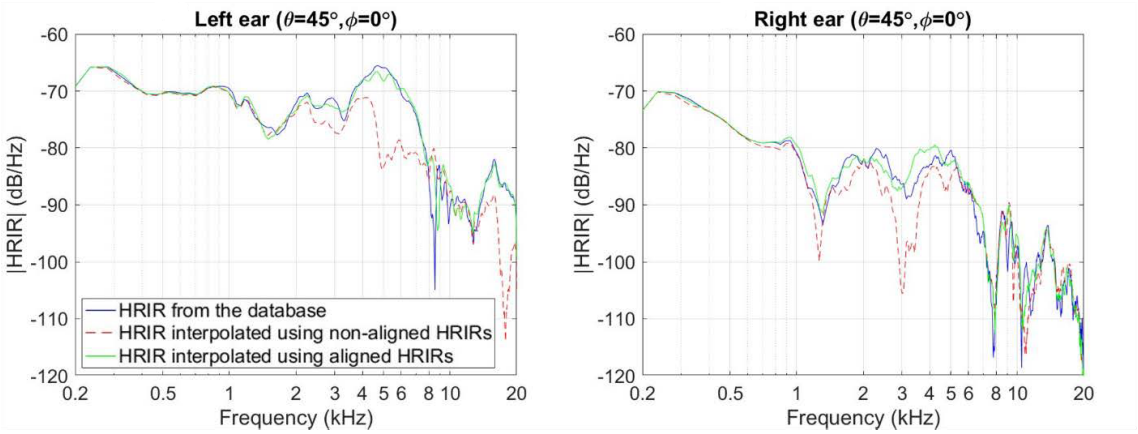


Figura 1.4: Comparación de la interpolación entre una HRIR alineada y una no alineada a 45º de azimut[4]

el resultado de la interpolación utilizando HRIR alineadas. La conclusión es que la interpolación de HRIR alineadas, donde se interpola por separado el retardo y la señal para juntarlos luego y reconstruir la HRIR original, da como resultado una HRIR interpolada mucho más parecida a la que se obtuvo al medir. Estos datos son sacados de la tesis de María Cuevas Rodríguez, donde se describe en profundidad toolkit y el efecto *comb filtering* mencionado [4] .

1.3.2. El cálculo del ITD y el modelo de Woodworth

El retardo o diferencia de tiempo interaural (ITD) es un indicio binaural importante para la localización de la fuente de sonido. Los cálculos de los valores de ITD se obtienen a partir de las respuestas de impulso relacionadas con la cabeza (HRIR) medidas en el dominio del tiempo o de sus funciones de transferencia relacionadas con la cabeza (HRTF) de transformación de frecuencia [11].

Aunque en este trabajo se menciona mucho el ITD, el objetivo real es poder determinar qué parte de una HRIR se considera retardo (*delay*) y cuál señal. Esto, además, nos permite calcular el ITD, puesto que la diferencia entre los retardos a cada canal desde una dirección es precisamente el ITD. Como se verá, nos apoyamos en el cálculo del ITD para evaluar los resultados de las pruebas que se realizan, porque existe un modelo geométrico, el modelo de Woodworth, que nos permite aproximar directamente el ITD a partir del tamaño de una cabeza esférica cuyo diámetro fuera la distancia entre los canales auditivos.

El modelo y fórmula de Woodworth, para la diferencia de tiempo interaural, suele ser usado como estándar en estudios, tanto en humanos como animales.

Es un modelo de una cabeza esférica rígida independiente de la frecuencia. Aunque este modelo conduce a diferentes fórmulas definidas por el ángulo del oído y la distancia a la fuente, la función más conocida de Woodworth proviene de ignorar el más largo de los dos caminos alrededor de la cabeza. [1].

Hay un TFG anterior [14] que estudia diferentes métodos de obtención del ITD propuestos en [11], y se usará como base, explicándolo más en detalle en el siguiente capítulo

1.4. Estructura del documento

El resto de este documento, memoria del Trabajo Fin de Grado titulado "Detección automática del retardo en la función de transferencia relacionada por la cabeza", está organizado en cuatro capítulos:

- Capítulo 2 - Especificaciones: Detalla los antecedentes sobre los que se basa este trabajo, estudiando los distintos métodos y decantándose por uno, cuya viabilidad se evalúa, dando como resultado los requisitos del trabajo y las pruebas que los verifican.
- Capítulo 3 - Diseño y Desarrollo: Documenta la modificación del 3D Tune-In Toolkit y el diseño y desarrollo de la aplicación de prueba.
- Capítulo 4 - Estudio de Resultados: Se detalla el estudio de viabilidad realizado en el capítulo 2, separados por bases de datos y mostrando como afectan los parámetros del método elegido.
- Capitulo 5 - Describe las conclusiones de los resultados y del trabajo realizado.

Parte II

Desarrollo del proyecto

Capítulo 2

Especificaciones

Contenido

Sinopsis

2.1. Estudio de los antecedentes

Como se indicó en el primer capítulo, se parte del proyecto de otro alumno[1] en el que se implementaron distintas funciones, tanto en MATLAB como *C++*, para calcular el ITD de una HRTF en un punto del espacio y así poder comparar

[1] MÉTODOS DE EXTRACCIÓN DEL ITD, Jesús Rodríguez Galán, 2020

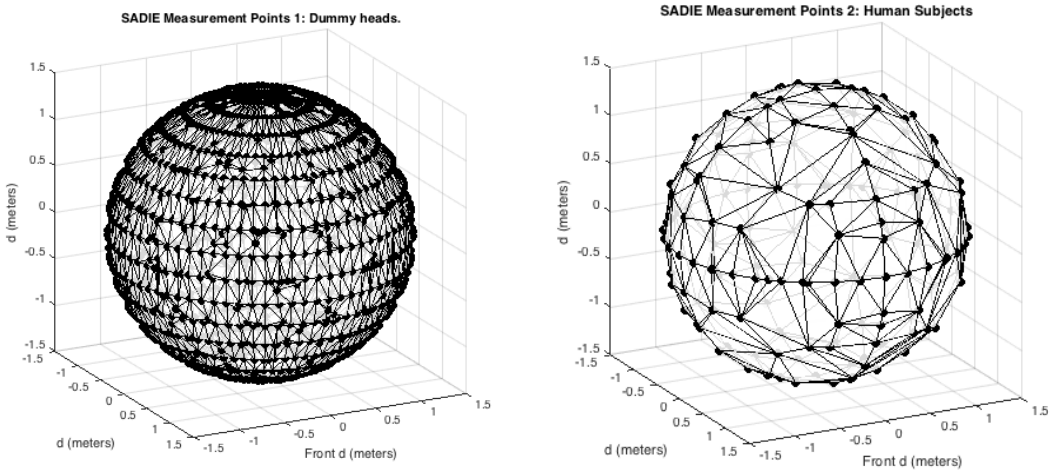
la diferencias entre métodos. El objetivo de este proyecto es distinto, ya que lo que se busca es estimar el retardo inicial de cada respuesta al impulso, mientras que el proyecto del que se parte busca calcular la diferencia de retardos entre las respuestas al impulso entre el oído derecho y el izquierdo.

La meta de esta parte es la familiarización con las funciones, el almacenamiento de los datos y comprobar si se puede obtener los retardos individuales de cada oído adaptando las funciones ya creadas.

2.1.1. Bases de datos de HRTF

Las bases de datos *sofa* a nuestra disposición y que serán usadas para las pruebas son:

- **SADIE**: Resultado de un proyecto de investigación financiado por EPSRC (*Engineering and Physical Sciences Research Council*) realizado en el Departamento de Electrónica de la Universidad de York. La base de datos se compone de 20 sujetos, de los cuales 2 son maniquíes (D1-D2) y el resto personas (H3-H20). El número de muestras varía entre maniquíes y humanos, siendo 8802 puntos medidos para los maniquíes y entre 2818 y 2114 para los humanos, en todo el espacio. Las Figuras 2.1a y 2.1b ilustra la distribución espacial de las medidas de esta base de datos.[6]



(a) Distribución de los puntos de medida de las cabezas de muñecos [6]

(b) Distribución de los puntos de medida de las cabezas humanas [6]

Figura 2.1: Representación espacial de los datos tomados por *SADIE*

- **ARI**: El *Acoustics Research Institute* es una institución de investigación interdisciplinaria de la Academia de Ciencias de Austria, dedicada a la investigación fundamental en acústica de aplicación abierta, en cooperación con socios científicos de renombre internacional. Con una base de datos de más de 200 sujetos, cada *sofa* se compone de 1550 posiciones medidas. Estas medidas se han realizado en todo el espacio azimutal y en un rango de -30° a 80° de elevación, con una separación de 2.5°. [9] La Figura 2.2 ilustra la distribución espacial de las medidas de esta base de datos.

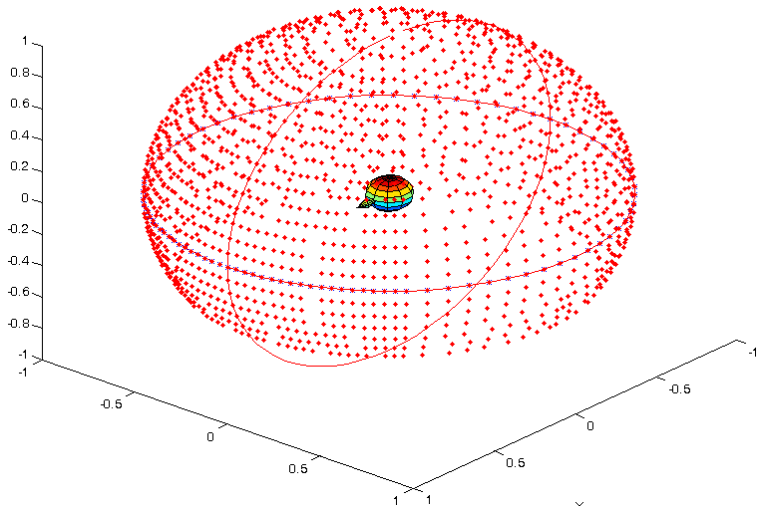


Figura 2.2: Representación espacial de los datos tomados por *ARI* [9]

- **CIPIC**: Base de datos de dominio público de mediciones HRTF de alta resolución espacial para 45 sujetos diferentes, incluido el maniquí KEMAR con pabellones auriculares pequeños y grandes. La base de datos incluye 2.500 mediciones de respuestas de impulso relacionadas con la cabeza para cada sujeto. [2]

2.1.2. Métodos de cálculo del ITD

A continuación revisamos brevemente los distintos métodos a nuestra disposición para calcular el ITD.

- **Método del Umbral** Consiste en obtener el ITD mediante la diferencia de tiempo que tarda una misma señal en llegar a cada oído, dado un azimut (θ) y una elevación (ϕ). Este método es el que más interesante para este

trabajo, pues es el que permite obtener el retardo de una HRIR individualmente y de forma absoluta, mientras que el resto de métodos calculan la diferencia de retardos entre las HRIR de ambos oídos de forma directa [14].

Dado que las HRTF no son ideales y contienen ruido, hay que determinar que parte de la misma se considera retardo y señal. El método del umbral se basa en seleccionar un nivel menor del punto máximo de la señal como umbral, que al ser superado, se tome a partir del índice de dicho valor como la señal sin retardo. A priori se suele filtrar la señal para eliminar ruidos indeseados, como se puede observar en las figuras 2.3 y 2.4. Se aprecia en la figura con claridad como se han atenuado picos en la señal.

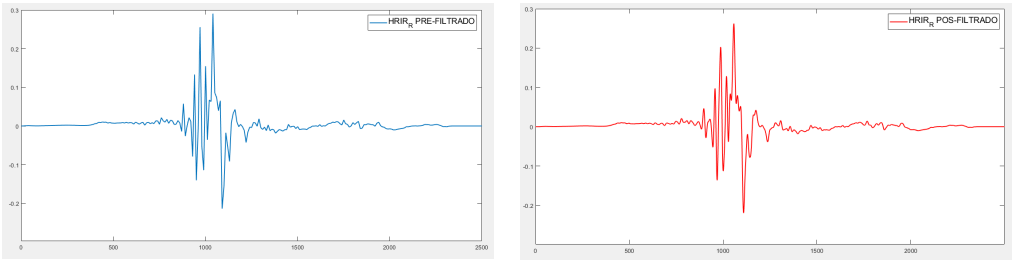


Figura 2.3: HRIR del oído derecho a 0º azimut y 0º de elevación antes del filtrado (azul) y después del filtrado (rojo)

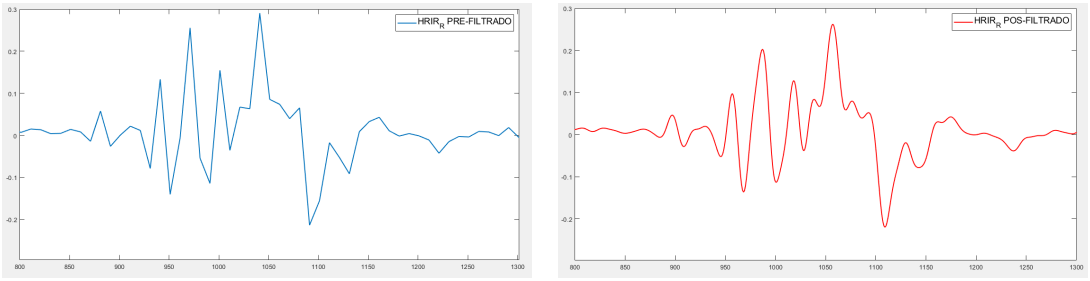


Figura 2.4: Ampliación de la parte central de la figura 2.3

Determinar un umbral ideal no es fácil, ya que dependiendo de la HRTF y de la base de datos puede variar mucho. Se pone como ejemplo la figura 2.5.

Para simplificar, se dirá que el valor máximo de la señal es $0{,}3$. Si se usara un valor de umbral de $0{,}1$ ($10\,\%$ del valor máximo), que es el de color rojo, el primer dato escogido por el método sería el índice 800; mientras que un umbral de $0{,}13$ ($13\,\%$ del valor máximo), que es el de color verde, el primer índice escogido seria el 925. Estos puntos marcan donde se separa el retardo de la señal del inicio de la misma. Por lo tanto, un pequeño cambio

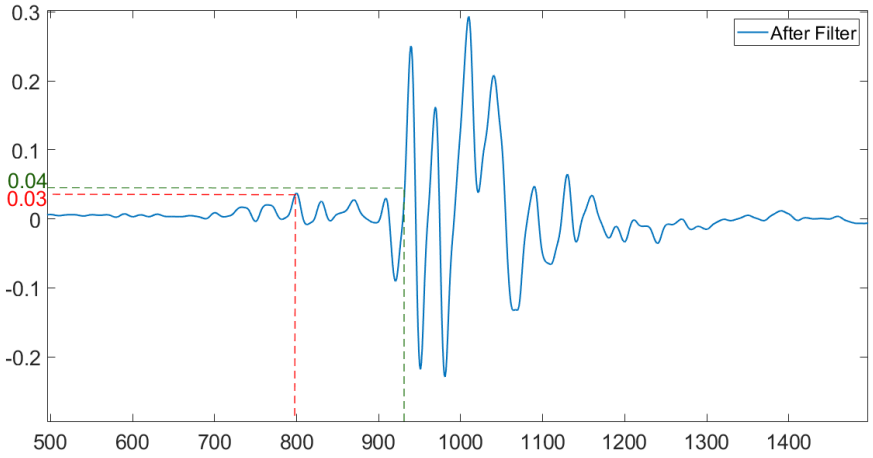
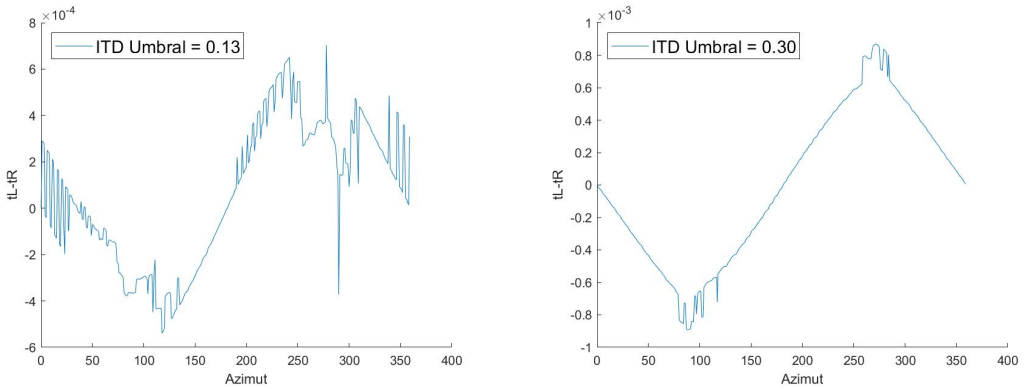


Figura 2.5: Representación de como funciona el umbral en el HRIR derecho a 0º azimut y elevación

en el umbral puede influir mucho en el resultado final, pues, como ya se ha dicho, estas señales no son ideales y se pueden ver afectadas por el ruido, provocando que el dato escogido no sea el que debería ser.



(a) ITD con 13 % valor de umbral (b) ITD con 30 % valor de umbral

Figura 2.6: Resultados de las pruebas para el calculo del ITD en el eje horizontal. Se representa siempre la diferencia de tiempo del oído izquierdo menos el oído derecho, de ahí que entre 0 y 180° el valor representado del ITD sea negativo.

Comparando las figuras 2.6a y 2.6b se entiende mejor que no cualquier umbral es válido. Mientras que la primera da una figura bastante distorsionada, la segunda es más limpia, salvo en los picos, lo cual puede ser posible dado que esos azimut corresponden a las posiciones de las orejas y pueden

provocar esas perturbaciones, incluso causado por los hombros, o debido a la diferencia de amplitud entre las señales en esas zonas, como se observa en la figura 2.7, y que el método no calcula bien.

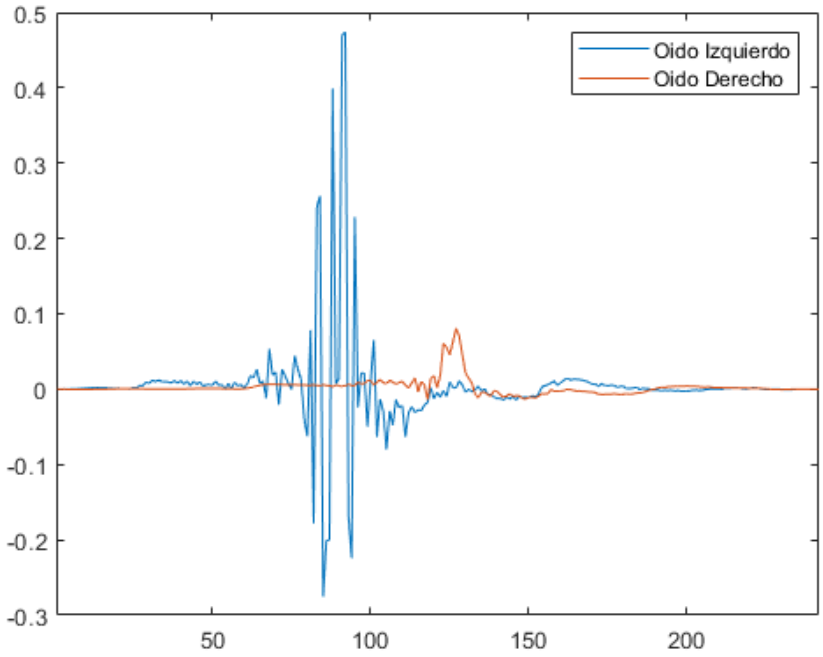


Figura 2.7: Respuesta al impulso a 90º de ambos oídos, siendo el izquierdo (azul) el lado incidente y el derecho (rojo) el lado opuesto.

- **Método de Correlación Cruzada** El método de correlación cruzada entre las HRIR de ambos oídos es otra forma de obtener el ITD, tal como se muestra en la formula 2.1.

$$\Phi_{LR}(\tau) = \frac{\int_{-\infty}^{\infty} h_L(t+\tau)h_R(t)dt}{\{[\int_{-\infty}^{\infty} h_L^2(t)dt][\int_{-\infty}^{\infty} h_R^2(t)dt]\}^{\frac{1}{2}}} \tag{2.1}$$

El uso de la correlación cruzada es frecuente en el procesamiento de señales, ya que permite medir la semejanza entre dos señales mediante su comparación. Existirá mayor similitud entre la respuesta del canal izquierdo, h_L, y la respuesta del canal derecho, h_R, cuanto mayor sea el resultado. El valor de ITD se refleja en la ecuación 2.2, definiendo el ITD como el instante τ_{max} por el cual $\Phi_{LR}(\tau)$ alcanza su valor máximo en el rango $|\tau| \leq 1ms$ [14].

$$ITD(\theta, \phi) = \tau_{max} \tag{2.2}$$

- **Método de Diferencia de Fase** Según este método el ITD resultante se define como:

$$ITD(\theta, \phi, f) = \frac{\triangle\psi}{2\pi f} = -\frac{\psi_L - \psi_R}{2\pi f} \tag{2.3}$$

siendo:

$$\begin{aligned} \psi_L &= arg[H_L(f)] = \arctan\left[\frac{Im\{H_L(f)\}}{Re\{H_L(f)\}}\right] \\ \psi_R &= arg[H_R(f)] = \arctan\left[\frac{Im\{H_R(f)\}}{Re\{H_R(f)\}}\right] \end{aligned} \tag{2.4}$$

Estas ecuaciones muestran que el ITD se puede calcular a través de la diferencia de las fases de las HRTF del oído izquierdo, H_L, y el oído derecho, H_R.

Este método presenta varios problemas, como un resultado dependiente de la frecuencia y ser un método poco fiable para frecuencias superiores a $1{,}5KHz$ [14].

- **Método de Retardo de Grupo** La definición de este método queda reflejada en la siguiente ecuación:

$$ITD(\theta, \phi, f) = -\frac{1}{2\pi}\left(\frac{d\psi_L}{df} - \frac{d\psi_R}{df}\right) \tag{2.5}$$

El ITD resulta de la diferencia entre los retardos de las envolventes de las respuestas de ambos oídos, o tal y como indica la **RAI** (Real Academia de Ingeniería), en el régimen de la frecuencia el ITD es la diferencia interaural entre la primera derivada respecto a la frecuencia de las fases de las HRTF dividido entre (-2π) [14].

- **Método de Recta de Regresión** Este método es una variante del método de Retardo de Grupo, con la diferencia de que en lugar de calcular el ITD usando la derivada de las fases con respecto a la frecuencia, se calcula a través de un ajuste de recta de regresión de la fase en un intervalo concreto de la frecuencia.

El recorrido extra de la onda que va a la contra lateral de la cabeza es lo que provoca el ITD. Al medir las diferencias de fase, el resultado obtenido es no lineal, cuando en su lugar se debería obtener un retardo puro (lineal).

Esto es debido a que la señal muestra un efecto en la fase añadido que no forma parte del ITD, lo que significa que se añade una distorsión de fase que provoca que el método de retardo de grupo falle. Para cada pareja de HRTF se realiza una recta de regresión en ambas fases y se aplica para un rango de frecuencias en concreto. Una vez se calculen, se pasará a obtener el retardo de tiempo para cada oído multiplicando la pendiente de las rectas por $-\frac{1}{2\pi}$. Por último, se calcula el ITD mediante la diferencia entre ambos retardos [14].

- **Método del Tiempo Promedio** Se calcula mediante el tiempo promedio (central) interaural, definido como:

$$\bar{t}_L = \frac{\int_{-\infty}^{\infty} t\, h_L^2(t)dt}{\int_{-\infty}^{\infty} h_L^2(t)dt},\ \bar{t}_R = \frac{\int_{-\infty}^{\infty} t\, h_R^2(t)dt}{\int_{-\infty}^{\infty} h_R^2(t)dt} \tag{2.6}$$

resultando el ITD de la diferencia entre ambos promedios:

$$ITD(\theta, \phi) = \bar{t}_L - \bar{t}_R \tag{2.7}$$

Dicho ITD sirve para cuantificar la diferencia temporal de propagación de la energía entre los oídos. Es independiente a la frecuencia aunque varíe en función de la fuente sonora [14].

Como puede comprobarse, todos los métodos anteriores, menos el método del umbral, calculan directamente el ITD a base de comparar las HRTF del oído izquierdo y del derecho. No permiten, por tanto, hallar el retardo absoluto de un conjunto de HRTF que se quieren interpolar evitando el efecto peine que se explicó en el capítulo anterior.

2.1.3. Formato de la información y manejo práctico de las bases de datos de HRTF

Una vez explicados los métodos y bases de datos de que se disponen para la realización de pruebas y estudios, se procede a una breve introducción de las características que componen las bases de datos y como usarlas en MATLAB.

La primera función a usar es *sofastart*, la cual añadirá todas las direcciones de los archivos necesarios y comprobará si necesitamos la versión de MATLAB u Octave para la API. Posteriormente se carga una HRTF. En los ejemplos mostrados sobre la utilización se usará el fichero *SOFA* usado a continuación:

```
hrtf1 = sofaload('H3_44K_16bit_256tap_FIR_sofa.sofa')
```

Una vez que se ha cargado el archivo se puede comprobar la estructura de **hrtf1** en la Figura 2.8a.

API	*1x1 struct*
ListenerPosition	[0,0,0]
ListenerPosition_Type	'cartesian'
ListenerPosition_Units	'metre'
ReceiverPosition	[0,0.0900,0;0,-0.0900,0]
ReceiverPosition_Type	'cartesian'
ReceiverPosition_Units	'metre'
SourcePosition	*2818x3 double*
SourcePosition_Type	'spherical'
SourcePosition_Units	'degree, degree, metre'
EmitterPosition	[0,0,0]
EmitterPosition_Type	'cartesian'
EmitterPosition_Units	'metre'
ListenerUp	[0,0,1]
ListenerView	[1,0,0]
ListenerView_Type	'cartesian'
ListenerView_Units	'metre'
Data	*1x1 struct*

(a) Estructura de la HRTF y la información que contiene

hrtf.SourcePosition

hrtf.SourcePosition

	1	2	3
1	0	-90	1.2000
2	0	-81	1.2000
3	0	-75	1.2000
4	0	-64.8000	1.2000
5	0	-60	1.2000
6	0	-54	1.2000
7	0	-45	1.2000

(b) Posición de las señales por azimut y elevación de hrtf1

Figura 2.8: Información más relevante sobre la HRTF

Por un lado, se tiene la matriz *SourcePosition* que indica el número de puntos del espacio del que se compone su HRTF, 8802 puntos; sus 3 columnas indican el azimut, la elevación y la distancia a la que se tomó la medida, en ese orden, como se puede observar en la Figura 2.8b. Estas posiciones hacen referencia a la fila en la que se encuentra su HRTF dentro de la estructura *Data* (abajo de la Figura 2.8a. En el interior de *Data* se encuentra la matriz *IR*, que contiene las respuestas al impulso de ambos canales auditivos por separado en 2 columnas, una para cada oído. Hay que tener cuidado, ya que dependiendo de la base de datos, la respuesta al impulso del canal auditivo derecho puede estar en la primera columna o la segunda; lo mismo para el izquierdo. Por lo tanto, al usar la función *find()* de MATLAB, para buscar un azimut y elevación determinados, nos devuelve la fila correspondiente si se encuentra entre ellos.

Listado 2.1: Código Metodo Umbral

```
function ITD = MetodoUmbral(Obj, azmth, elev, fc, P)
%   Estimates ITD using Threshold method for a given angle
%   Obj -> sofa file
%   azmt -> azimut angle
%   elev -> elevation angle
%   fc -> frecuencia de corte
%   P -> Umbral

%% Searching position
fs = Obj.Data.SamplingRate;
pos = find(Obj.SourcePosition(:,2) == elev & Obj.SourcePosition(:,1) == ...
    azmth);
A = reshape(Obj.Data.IR(pos,1,:), [1, length(Obj.Data.IR(pos,1,:))]);
B = reshape(Obj.Data.IR(pos,2,:), [1, length(Obj.Data.IR(pos,2,:))]);

%% Oversampling
OS = 10; % Oversampling factor
ta = 0:1/fs:(length(A)-1)/fs;
ty = 0:1/(OS*fs):(length(A)-1)/(fs);
L = interp1(ta,A,ty); R = interp1(ta,B,ty); %% Oversampling x10

%Filtro Butterworth de orden 10
%la frecuencia de corte hay que conveertirla a radianes
[b,a] = butter(10,fc/(fs/2));

%Se aplica el filtro
FL = filter(b, a, L);
FR = filter(b, a, R);

%% Apply Threshold Method.
%Find the first value that surpass the maximun
tL = find(FL > max(FL)*P,1);
tL = tL/(OS*fs);
tR = find(FR > max(FR)*P,1);
tR = tR/(OS*fs);

ITD = abs(tL - tR);
end
```

Para facilitar el manejo de datos convertimos las matrices tridimensionales que vamos a usar en arrays. Para ello se utiliza la función de MATLAB *reshape*, que modifica las dimensiones del elemento dado.

2.1.4. Estudio de viabilidad del Método del Umbral mediante la implementación de un prototipo en MATLAB

En primer lugar, en prevención de un posible *aliasing* durante el filtrado, se recurre a un proceso de sobremuestreo (*oversampling*) para estrechar el espectro de la señal. A continuación se usa un filtro paso bajo de 3 KHz (filtro Butterworth

de orden 10) [12]. Esta es la frecuencia de corte que se suele utilizar cuando el retardo se utiliza para calcular el ITD. A la velocidad normal del sonido en el aire, esta frecuencia corresponde con una longitud de onda próxima a 10 cm. Se considera que, a partir de estas longitudes de onda, las características espectrales de la HRTF dependen más de las características del pabellón auricular que de la forma y tamaño de la cabeza. Por lo tanto se filtran para evitar que introduzcan ruido en la aplicación del método del umbral. El proceso de filtrado se ilustra en la figura 2.4.

En MATLAB, para crear el filtro, se utilizan las funciones:

```
[b,a] = butter(n,Wn);
Y = filter(b, a, X);
```

Donde 'n' indica el orden, 'Wn' frecuencia de corte normalizada, '*butter()*' devuelve los coeficientes, 'a' y 'b', de la función de transferencia para poder aplicar el filtro a las señales, '*filter()*' ejecuta el filtro con los valores especificados, 'X' la señal sin filtrar e 'Y' la señal tras el filtro.

La figura 2.9 muestra el prototipo completo realizado, una vez analizado y simplificada una parte del código de partida del Trabajo Fin de Grado del que partíamos[2].

Este prototipo nos permite realizar pruebas introduciendo varias HRTF y un número de distintos umbrales, comprendidos entre 0 y 1, que especifican el valor relativo al punto máximo. Posteriormente se realizan otras pruebas en las que para ciertos umbrales se modifica la frecuencia del filtro para estudiar los cambios que produce esta modificación.

Una vez aplicado el método a ambas HRIR, izquierda y derecha del mismo punto, tendremos el punto de inicio de cada señal en el tiempo; y una vez se resten ambas salidas como se muestra a la derecha de la figura 2.9, que corresponde con los instantes de tiempo, se obtiene el **ITD**, y es donde aparecen los problemas con los *outliers*. T_R y T_L corresponden con el índice que marca la duración del retardo.

Por lo tanto, hay como mínimo dos parámetros en el método. Por un lado tenemos el tanto por ciento del máximo para el umbral, y por otro la frecuencia de corte del filtro. Los pasos que no varían son el *oversampling* y el tipo de filtro.

En este punto, la forma de evaluar los valores del umbral será de forma indirecta; utilizando las diferencias entre los retardos de cada oído para el mismo punto del espacio y un modelo analítico aproximado, el modelo de Woodworth.

[2]MÉTODOS DE EXTRACCIÓN DEL ITD, Jesús Rodríguez Galán, 2020

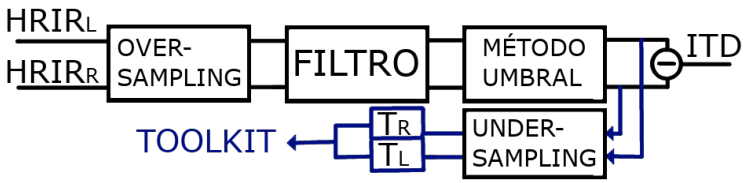


Figura 2.9: Esquema del prototipo que aplica el método del umbral a la obtención de retardos

2.2. Aplicación del Modelo de Woodworth para la evaluación del Método del Umbral

Para estudiar cuál es el mejor umbral para cada HRTF hacemos uso de un modelo ideal, el modelo de Woodworth, en el que se supone una cabeza esférica, con las orejas a los lados, y el ITD viene dado por la fórmula 2.8 y la figura 2.10.

$$ITDw = \frac{a}{c}(\theta + \sin\theta) \tag{2.8}$$

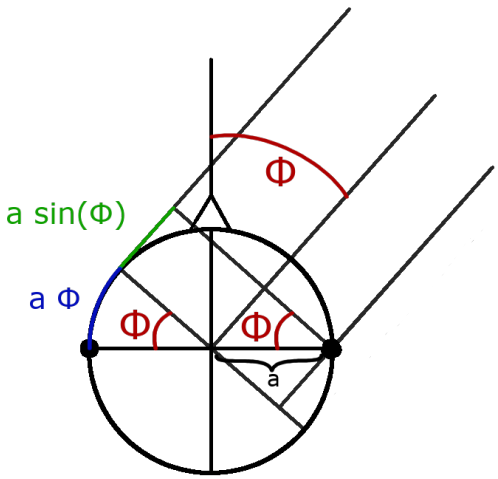


Figura 2.10: Representación del teorema de Woodworth

Donde **a** es el radio de la cabeza en metros, **c** es la velocidad del sonido y θ el ángulo de incidencia. Este modelo es solo una orientación visual que ayuda a la comparación de los resultados.

Para conocer el tamaño de la cabeza se recurre a la fórmula de la correlación cruzada. Se toma los datos más próximos al azimut 90° y 270° de la horizontal

pura para obtener el ITD de esos puntos. Con estos datos despejamos el radio de la cabeza de la formula del ITD de Woodworth, quedando la ecuación 2.9

$$a = \frac{ITDw \cdot c}{\theta + \sin\theta} \tag{2.9}$$

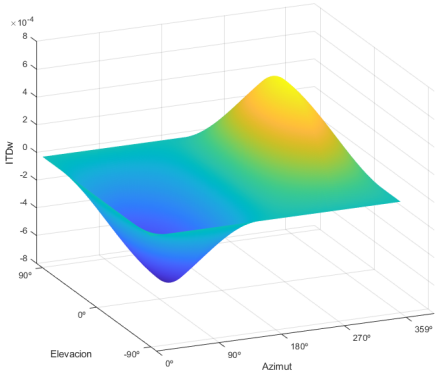


Figura 2.11: ITD de Woodworth en todo el espacio

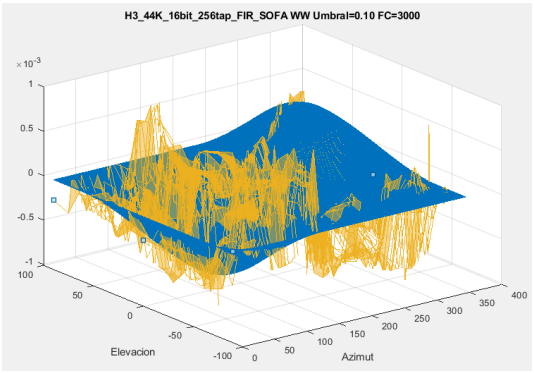
Una vez que tenemos el ITD de Woodworth para cada umbral, figura 2.11, se sacan las diferencias de tiempo totales entre este ITD y el obtenido mediante el método del umbral. Los datos con mayor diferencia de tiempo se descartan, pues los resultados de las pruebas demuestran que estos poseen una gran cantidad de outliers. Un ejemplo de un mal nivel de umbral seria el mostrado en la figura 2.12a. Podemos observar que, con un umbral de 0.1, los datos obtenidos no se asemejan a la forma del ITD en e modelo Woodworth

Esto es debido a que el método del umbral es muy susceptible a ruidos y, dado que el filtro no limpia completamente la señal, empeora los resultados. Así mismo, el ILD y la sombra proyectada por la cabeza afecta a este método, como se puede mostró antes en la figura 2.7. Se puede apreciar cómo un pequeño ruido puede afectar al método.

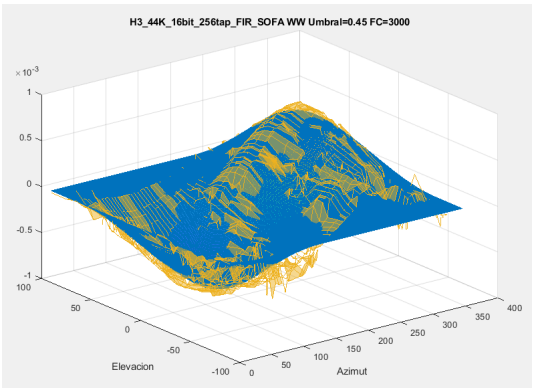
En la Figura 2.12b se representa los resultados de un umbral que proporciona mayor similitud entre el método y el ideal. No obstante se siguen apreciando ciertos outliers que hay que solventar.

2.3. Definición de Requisitos

Una vez que se han estudiado los antecedentes y la viabilidad del sistema, en la tabla 2.1 se recogen los requisitos que se pretenden conseguir con este



(a) ITD que no se ajusta a la forma teórica



(b) ITD que se ajusta a la forma teórica pero con bastantes *outliers*

Figura 2.12: Representaciones del ITD teórico de Woodworth (azul) con uno real (amarillo) con distintos umbrales

proyecto

Tabla 2.1: Agrupación de requisitos propuestos para el TFG.

Nº	**Nombre**	**Descripción**
R1	Umbral	El sistema permitirá elegir el umbral que se desee para el método estudiado en el apartado 2.1.4.
R2	Frecuencia de corte	El sistema permitirá elegir la frecuencia de corte del filtro paso bajo.
R3	Retardos	El sistema calculará automáticamente los retardos de las HRIR al introducir un archivo que no contenga los retardos.
R4	Curva ITD	Todos las HRTF deben tener un umbral y frecuencia de corte que dé, como resultado, una representación del ITD similar a la ideal y con un número de puntos que no se adapten a esa representación muy bajo.
R5	Outliers	En caso de encontrar puntos que no se adapten a la curva del ITD, se intentará corregirlos.
R6	Cargar datos	Al cargar una HRTF comprobará si tiene el campo de los retardos completo. Si es el caso utilizará esos datos, si no los calculará.

R7	Actualizar datos	El sistema permitirá actualizar los retardos con el umbral y frecuencia de corte elegidos, de forma interactiva.
R8	Visualización del ITD	Al cargar los retardos predefinidos, o al ser calculados por el método, se dispondrá de una representación del ITD horizontal.
R9	Sonido	Se debe poder cargar una fuente sonora para convolucionarla con la HRTF cuyos retardos han sido calculados por el sistema.
R10	Manipulación del sonido	Se debe disponer de un control que permita mover la fuente sonora por el espacio.
R11	Muñeco	En la interfaz se podrá observar una cabeza vista desde arriba y un punto con una orbita alrededor de la cabeza. Al usar el control simulará el movimiento de la fuente sonora.

El objetivo es, por tanto, construir un sistema que cumpla dichos requisitos, si bien algunos, tras el estudio de viabilidad realizado, sabemos que serán difíciles de alcanzar (R4 y R5). Puesto que no podremos eliminar completamente los *outliers*, introducimos en los requisitos la máxima interactividad posible, de manera que al usuario le sea cómodo corregir el umbral o la frecuencia de corte (R1 y R2) y observar los efectos que ello produce en el cálculo del ITD (R4, R7, R8, R9, R10 y R11).

Parte de los requisitos serán satisfechos por una versión modificada del 3D Tune-In toolkit. Otra parte, como se verá, serán satisfechos por una aplicación que utilizará esta versión modificada del toolkit.

2.4. Definición de Pruebas

En la tabla 2.2 se recopilan las pruebas propuestas para la verificación de los requisitos anteriormente propuestos. Esto concluirá en el capítulo de conclusiones, donde se expondrá los requisitos que no se han podido cumplir, por qué motivo y como solventarlo, si procede.

Tabla 2.2: Agrupación de las pruebas realizadas para la comprobación del cumplimiento de los requisitos.

Nº	**Nombre**	**Descripción**	**Requisito**
P1	Mejor umbral	Para cada HRTF se aplicará un intervalo de umbrales comprendidos entre 0.1 y 0.6, con un paso de 0.01. Se realizará la diferencia entre los ITD calculados y el modelo de Woodworth. Aquel o aquellos ITD con menores diferencias se considerarán con pocos datos desviados, y se estudiará visualmente.	R4, R5
P2	Diferentes frecuencias de corte	Para un mismo umbral, que se considere apto, se le aplicará el método con un intervalo de frecuencias de corte para el filtro y se estudiará si las diferencias son significativas.	R4, R5
P3	Cargar *sofa*	Se probará a cargar diferentes bases de datos, con y sin retardos definidos. El programa deberá cargar los retardos, o calcularlos si no estaban.	R3, R6, R8.
P4	Actualizar umbral y frecuencia de corte	Una vez cargado un HRTF, se permitirá seleccionar un nuevo umbral y frecuencia de corte. El sistema permitirá, de forma interactiva, actualizar los retardos con el umbral y frecuencia de corte seleccionados, y se mostrará el nuevo ITD	R1, R2, R3, R8
P5	Fuente sonora	Una vez ejecutado el programa se mostrará una interfaz. En ella aparecerá el icono de una cabeza, una fuente sonora alrededor suya y una gráfica que muestra el ITD actual. Se dispondrá de una barra deslizante que permitirá mover la fuente sonora, dando la sensación de que se mueve alrededor nuestra si tenemos puesto los cascos.	R9, R10, R11

Estas pruebas estarán centralizadas en un solo sistema: una aplicación de prueba que utilizará el 3D Tune-In Toolkit modificado. En el siguiente capítulo se explican, por un lado, la modificación del Toolkit, y, por otro, la aplicación de prueba.

Capítulo 3

Diseño y desarrollo

Contenido

Sinopsis

En este capítulo se hará una breve introducción a la estructura del 3D Tune-In Toolkit. A continuación, se muestra la intervención realizada en el mismo. Por último, se documenta el diseño y el desarrollo de la Aplicación de Prueba, que implementa los requisitos de interactividad y centraliza todas las pruebas definidas en el capítulo anterior.

3.1. Clases del Toolkit

Una vez realizadas las distintas pruebas en MATLAB, que se mostrarán en el siguiente capítulo, se tiene la información suficiente para poder trabajar dentro del 3D Tune-In Toolkit.

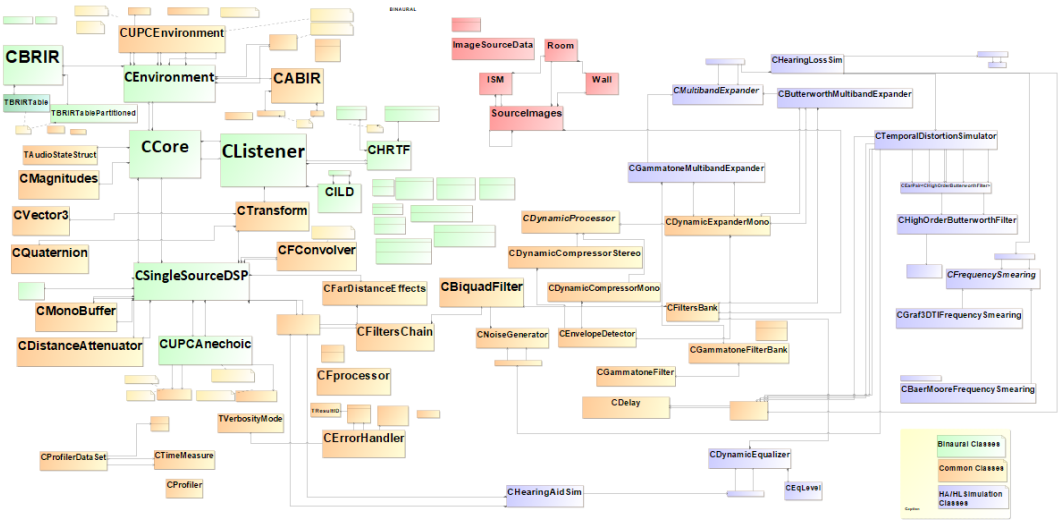


Figura 3.1: Relaciones del Toolkit

El 3D Tune-In Toolkit [1] está desarrollado en *C++*, utilizando el paradigma de orientación a objetos. Las relaciones entre clases dentro del Toolkit son extensas, como se trata de ilustrar en la figura 3.1, Sin embargo, la intervención en este trabajo se limita a añadir nuevos métodos en una de las clases, con el fin de incluir el método del umbral dentro de la parte del Toolkit encargada de cargar los ficheros *sofa* (el *Resource Manager*, que se explica más adelante), y que puedan calcularse los retardos por el método descrito en el capítulo anterior.

Las partes principales en las que se divide el Toolkit son:

Por una parte se encuentra ***Binaural Spatialiser***, que contiene los archivos encargados de la declaración y definición usados para la espacialización binaural. La biblioteca dispone de un renderizador de audio binaural 3D en tiempo real que ofrece una espacialización 3D completa. Entre las diferentes características se encuentra:

- Interpolación baricéntrica de HRIR entre los tres disponibles más cercanos.
- Se tiene en cuenta el efecto de paralaje acústico.
- El ITD se gestiona por separado de la HRIR.
- Simulación ILD, agregando una sombra acústica en el oído contralateral.
- La reverberación espacial se simula en tiempo real, utilizando una convolución dividida con BRIR que emplean un enfoque ambisónico virtual.

[1] <https://github.com/3DTune-In>

- Permite el desplazamiento de las fuentes sonoras y del oyente, gestionando los cálculos geométricos.

Entre otros. Se puede observar el diagrama de una de las clases de la *Binaural Spatialiser* en la figura 3.2

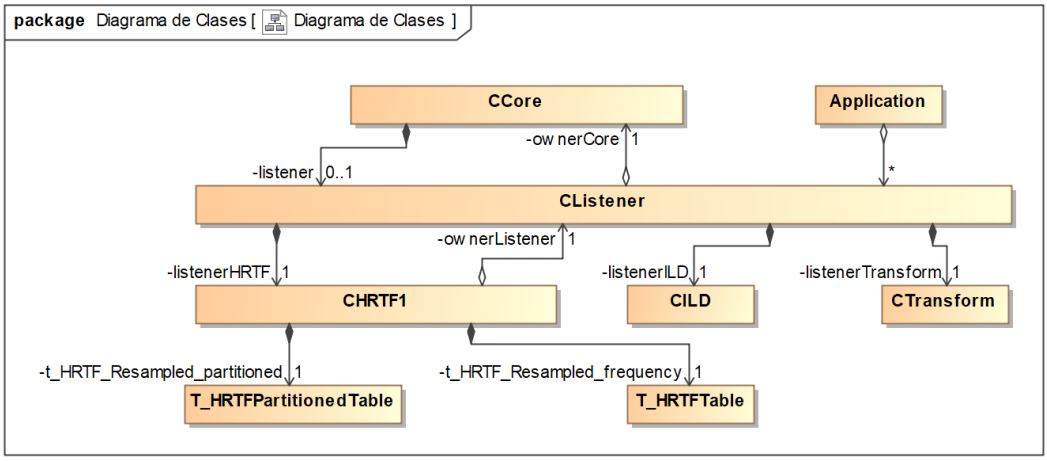


Figura 3.2: Diagrama de clases detallado de las clases que rodean al *Listener*, que se encuentra dentro de *Binaural Spatialiser*. El resultado la intervención realizada en este proyecto no modifica esta estructura, sino el contenido de los datos de la clase *CHRTF*, donde se guardan las respuestas al impulso y los retardos de cada una.

Dentro de *Binaural Spatialiser* se encuentra la carpeta ***Common***, que contiene los archivos que son compartidos por los demás componentes del Toolkit. Estos archivos se puede agrupar en cuatro categorías: transformación geométrica, usada para el desplazamiento de fuentes sonoras como se ha mencionado antes; procesamiento de señales, soporte para desarrolladores y general, que se encarga de magnitudes, *buffers*,...

Por otra parte se encuentra ***Resource Manager***, que contiene las herramientas implementadas para la conversión de formato y lectura de los diferentes archivos de recursos necesarios para configurar el modelo de escucha. Estas herramientas incluyen:

- Lector de archivos de formato *sofa* para HRTF y lector de formato binario 3DTI-HRTF, formato binario portátil multiplataforma para datos HRTF.
- Lector de archivos en formato *sofa* para BRIR y lector de formato binario 3DTI-BRIR

- Lector de formato binario 3DTI-ILD.

La intervención en el 3D Tune-In toolkit se centrará en modificar el *Resource Manager* y se explicará en el siguiente apartado.

3.2. Modificaciones realizadas en el 3D Tune-In Toolkit

La intervención en el proyecto se lleva a cabo dentro del apartado *Resource Manager*, en el archivo *HRTFFactory.cpp* y *HRTFFactory.h*.

Se ha añadido al código original la condición de que si los campos *Delay* del archivo cargado se encuentran vacíos se llame al método del umbral para rellenar los campos. Al método se le pasa la variable que contiene los valores de la HRIR, una copia de esta que será la que se modificará y el filtro.

Listado 3.1: Condicion de llamada al método

```
1  namespace HRTF
2  {
3  // [...]
4  bool LoadHRTFTableFromsofa(const std::string & sofafile, ...
       shared_ptr<Binaural::CListener> listener, bool & specifiedDelays)
5  {
6  // [...]
7  for (std::size_t i = 0; i < nMeasurements; i++)
8  {
9      //Creacion de variables que contendran las HRIR con y sin retardo
10     THRIRtruct hrir_value;
11     THRIRtruct hrir_value_wo_delay; //linea anyadida
12
13     //Ajusta la dimension de hrir.value a la longitud de las muestras.
14     hrir_value.leftHRIR.resize(nSamples);
15     hrir_value.rightHRIR.resize(nSamples);
16
17     /*
18     Parte de codigo que se encarga de cargar los retardos que contiene el archivo
19     */
20
21     //if() anyadido. Cuando el archivo no contiene los retardos o contiene un ...
           numero irregular de ellos se ejecuta el metodo.
22
23     if (!specifiedDelays) {  /* ejecuta el metodo si no hay delays */
24         hrir_value_wo_delay = hrir_value;
25         void ApplyThresholdMethod(hrir_value, hrir_value_wo_delay, lowPassFilter);
26
27         listener->GetHRTF()->AddHRIR(azimuth, elevation, ...
               std::move(hrir_value_wo_delay));
28      }
29      else  //if there are delays, those of the file are used
```

```
30        {
31            listener->GetHRTF()->AddHRIR(azimuth, elevation, std::move(hrir_value));
32        }
33  } //for()
34      // [...]
35  } //bool LoadHRTFTableFromsofa()
36      // [...]
37  } //namespace HRTF
```

El Toolkit tiene una herramienta para crear filtros haciendo una cadena de secciones de segundo orden. Los coeficientes de dichas cadenas los obtenemos de MATLAB con las funciones:

```
% fc -> frecuencia de corte
% fs -> frecuencia de muestreo
[U, I, O, P] = butter(10,fc/(fs/2),"low");
sos = ss2sos(U,I,O,P);
```

Donde la primera función *butter()*, toma el orden del filtro digital y la frecuencia de corte normalizada; y la segunda, *ss2sos()*, convierte el resultado de la primera en secciones de segundo orden. Posteriormente se rellena el filtro siguiendo el orden de la ecuación 3.1 [2]:

$$sos = \begin{bmatrix} b_{01} & b_{11} & b_{21} & 1 & a_{11} & a_{21} \\ b_{02} & b_{12} & b_{22} & 1 & a_{12} & a_{22} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{0L} & b_{1L} & b_{2L} & 1 & a_{1L} & a_{2L} \end{bmatrix} \tag{3.1}$$

Se quiere un filtro de orden 10, por lo que se encadenan 5 filtros de orden 2. De manera que la construcción del filtro queda de la siguiente forma:

Listado 3.2: Creación del filtro

```
1  Common::CFiltersChain lowPassFilter;
2  if (!specifiedDelays) {
3      int nFilters = 5;
4      for (int i = 0; i < nFilters; i++)
5      {
6          shared_ptr <Common::CBiquadFilter> oneFilter = lowPassFilter.AddFilter();
7          oneFilter->SetSamplingFreq(SAMPLERATE);
8          oneFilter->Setup(SAMPLERATE, 3000, LPF_Q, Common::LOWPASS);
9
```

[2] La referencia de esta función de MATLAB puede encontrarse en <https://es.mathworks.com/help/signal/ref/ss2sos.html>

```
10      // Si la frecuencia de muestreo es de 44.1 KHz
11          if (SAMPLERATE == 44100) {
12              if (i == 0) { oneFilter->SetCoefficients(5.83237805842881e-08, ...
                    1.23790232693845e-07, 6.57097356419428e-08, -1.29135417056970, ...
                    0.419015766235189); }
13              else if (i == 1) { oneFilter->SetCoefficients(1, 2.07251690861975, ...
                    1.07659543608299, -1.32914904519877, 0.460546993069658); }
14              else if (i == 2) { oneFilter->SetCoefficients(1, 1.99623914806128, ...
                    1.00017482226091, -1.40750534391516, 0.546649493682106); }
15              else if (i == 3) { oneFilter->SetCoefficients(1, 1.92515305704335, ...
                    0.928954048314882, -1.53179651063724, 0.683227923675955); }
16              else if (i == 4) { oneFilter->SetCoefficients(1, 1.88362503351858, ...
                    0.887346547066281, -1.70923173544711, 0.878204164299093); }
17          }
18        // Si la frecuencia de muestreo es de 48 KHz
19          else if (SAMPLERATE == 48000) {
20              if (i == 0) { oneFilter->SetCoefficients(2.73443806791476e-08,  ...
                    5.79676316967698e-08,   3.07325874953726e-08, - ...
                    1.34092645780358,   0.451408339092306); }
21              else if (i == 1) { oneFilter->SetCoefficients(1, 2.07105979891257, ...
                    1.07497131664538, -1.37792369101677, 0.491453855754655); }
22              else if (i == 2) { oneFilter->SetCoefficients(1, 1.99636858974677, ...
                    1.00014534704996, -1.45424358625159, 0.574061915083955); }
23              else if (i == 3) { oneFilter->SetCoefficients(1, 1.92669263562003, ...
                    0.930341629227868, -1.57425622265844, 0.703962656667260); }
24              else if (i == 4) { oneFilter->SetCoefficients(1, 1.88596876905156, ...
                    0.889542117010788, -1.74339116555251, 0.887032999652699); }
25          }
26      }
27  }
```

Cuando se llama al método se ejecutan las siguientes funciones, que hacen lo mismo que se vio en el listado 2.1 en la página 22, es decir implementan el Método del Umbral, pero esta vez utilizando C++. Las funciones se han separado para ayudar a la visualización y se compone desde el listado 3.3 al listado 3.7

Listado 3.3: Función principal, llamada en el listado 3.1 y que aplica el método del umbral al HRTF cargado

```
1  void ApplyThresholdMethod(const THRIRtruct& hrir_value, THRIRtruct& ...
       hrir_value_wo_delay, Common::CFiltersChain & lowPassFilter)
2  {
3      //Se hace una copia porque al filtrar se sobreescriben los datos
4      CMonoBuffer<float> hrir_filtered_left = hrir_value.leftHRIR;
5      CMonoBuffer<float> hrir_filtered_right = hrir_value.rightHRIR;
6
7      //variables usadas para guardar el indice de donde acaba el delay
8      int delayLIndex = 0; //indice delay izquierdo
9      int delayRIndex = 0; //indice delay derecho
10
11     //// OVERSAMPLING  ////
12     //Llama a la funcion auxiliar oversampling(). Sobreescribe los datos
13     oversampling(hrir_filtered_left); //izquierdo
14     oversampling(hrir_filtered_right); //derecho
15
16     //// FILTER APPLICATION ////
```

```
17      //Funcion interna del TOOLKIT que aplica el filtro creado.
18      //Se aplica en las HRIR de ambos oidos.
19      lowPassFilter.Process(hrir_filtered_left);
20      lowPassFilter.Process(hrir_filtered_right);
21
22
23      //// THRESHOLD METHOD ////
24      //Llama a la funcion auxiliar para la aplicacion del metodo, que devuelve los ...
            delays
25      Threshold(hrir_filtered_left, hrir_filtered_right, delayLIndex, delayRIndex);
26
27      //undersampling para obtener el indice sobre la senyal original
28      int pL = delayLIndex / 10; //indice izquierdo
29      int pR = delayRIndex / 10; //indice derecho
30
31      //Se guarda el indice del delay en el campo correspondiente
32      hrir_value_wo_delay.leftDelay = pL; //izquierdo
33      hrir_value_wo_delay.rightDelay = pR; //derecho
34  }
```

Listado 3.4: Función auxiliar *oversampling()* usado para estrechar el espectro de la señal y evitar el efecto de *aliasing*

```
1   void oversampling(CMonoBuffer<float> & HRIR1)
2   {
3       //Aplicacion del oversampling
4       //Se amplia el numero de muestras. Las nuevas muestras se interpolan.
5       size_t ns = HRIR1.size();
6       CMonoBuffer<float>  aux(-(OVERS - 1) + ns * OVERS), t(ns), taux(-(OVERS - 1) ...
            + ns * OVERS);
7
8       for (int i = 0; i < ns; i++) {  // Time vector
9           t[i] = i / float(SAMPLERATE);
10      }
11      for (int i = 0; i < (OVERS * ns) - 9; i++)  // New time vector
12          taux[i] = i / (OVERS * float(SAMPLERATE));
13
14      int j = 0;
15      for (int i = 0; i < taux.size(); i++) { // Linear interpolation
16          aux[i] = HRIR1[j] + (taux[i] - t[j])*(HRIR1[j + 1] - HRIR1[j]) / (t[j + ...
                1] - t[j]);
17          if (t[j + 1] == taux[i])
18              j++;
19      }
20
21      HRIR1 = aux; //Overwrite current hrir
22  }
```

Listado 3.5: Función auxiliar *Threshold()* que aplica el método del umbral después del filtrado y el *oversampling()*

```
1   float Threshold(CMonoBuffer<float> & HRIRL, CMonoBuffer<float> & HRIRR, int ...
        &delayLIndex, int &delayRIndex) // Returns ITD using Threshold method
2   {
3       //Left time index and Right time index when is oversampled
4       int SL, SR;
```

```
5
6       //left and right maximun value
7       int maxL, maxR.
8
9       //Call the HMAX() auxiliary function to return max element index
10      maxL = HMAX(HRIRL); //left
11      maxR = HMAX(HRIRR); //right
12
13      //Auxiliary function HTHESH. Given a signal, a maximum value and a threshold ...
            returns the first index that exceeds the threshold
14      //P is an external variable that is passed from the main code
15      SL = HTHRESH(HRIRL, maxL, P); // left
16      SR = HTHRESH(HRIRR, maxR, P); // right
17
18      //Overwriting delay index variables
19      delayLIndex = SL;
20      delayRIndex = SR;
21  }
```

Listado 3.6: Función auxiliar *HMAX()* que devuelve el índice con el mayor valor de una HRIR

```
1   int HMAX(CMonoBuffer<float> &HRIR1) // Returns hrir index with max value
2   {
3       int i = 0, max = 0;
4       while (i < HRIR1.size()) {
5           if (HRIR1[i] > HRIR1[max])
6               max = i;
7
8           i++;
9       }
10      return max;
11  }
```

Listado 3.7: Función auxiliar *HTHRESH()* que devuelve el primer valor que sobrepasa el umbral dado *P*, que es un valor externo pasado desde el código principal

```
1   int HTHRESH(CMonoBuffer<float> &HRIR1, int maxL, float P) // returns first ...
        element which overpass Threshold
2   {
3       int i = 0;
4       while ((i + 1) < HRIR1.size() && HRIR1[i] < (HRIR1[maxL] * P))
5           i++;
6
7       if (i ≥ HRIR1.size())
8           i = -1;
9
10      return i;
11  }
```

3.3. Aplicación de prueba (diseño)

El programa principal que se ha desarrollado llama a la función principal del apartado anterior, y tras realizar los cálculos, permite la manipulación de una fuente sonora. Toda la aplicación se ha realizado mediante la librería de C++ OpenFrameworks[3], y su interfaz creada con un complemento de OpenFrameworks llamado GUI[4]. La aplicación permite, además, recalcular el ITD con diferentes umbrales y frecuencias de corte sin tener que volver a ejecutar el programa.

OpenFrameworks ofrece una estructura para el *main* de la aplicación que divide en 4 funciones: *setup()*, que sirve para la inicialización de lo que se vaya a usar y solo se ejecuta la primera vez; *update()* y *draw()* que actualizan las variables y lo que se muestra por la interfaz ejecutándose continuamente; y los manejadores de interrupción que se activan cuando ocurre una acción esperada como pulsar o soltar un botón.

Lo primero que se hace en el *setup()* es la inicialización de la interfaz, un oyente, una fuente sonora y cargar un archivo *sofa*, con o sin delays predeterminados. Como la función *setup()* es bastante extensa solo se mostrarán los elementos más importantes en el listado 3.8.

Listado 3.8: Inicialización de la interfaz del programa

```
1  void ofApp::setup(){
2
3      ofSetVerticalSync(true);
4      //Se obtiene la frecuencia del muestreo del archivo que se va a usar
5      SAMPLERATE = HRTF::GetSampleRateFromsofa("H3_44K_16bit_256tap_FIR_sofa.sofa");
6
7      //Inicializacion de las variables de la interfaz
8      refreshButton.addListener(this, &ofApp::refreshButtonPressed);
9      gui.setup(); // most of the time you don't need a name
10     /*
11         Initialization of different elements
12     */
13
14     // Core setup
15     audioState.bufferSize = BUFFERSIZE; // Setting buffer size
16     audioState.sampleRate = SAMPLERATE; // Setting frame rate
17     myCore.SetAudioState(audioState);   // Applying configuration to core
18     myCore.SetHRTFResamplingStep(15);   // Setting 15-degree resampling step for HRTF
19
20
21     // Inicializacion del oyente
22     listener = myCore.CreateListener(); // First step is creating listener
23     Common::CTransform listenerPosition = Common::CTransform();  // Setting ...
```

[3] <https://openframeworks.cc>

[4] <https://openframeworks.cc/documentation/ofxGui/>

```
        listener in (0,0,0)
24      listenerPosition.SetPosition(Common::CVector3(ofGetWidth()*.5, ...
            ofGetHeight()*.28, 0));
25      listener->SetListenerTransform(listenerPosition);
26      listener->DisableCustomizedITD();    // Disabling custom head radius
27
28      // HRTF can be loaded in sofa (more info in https://sofacoustics.org/) Some ...
            examples of HRTF files can be found in 3dti_AudioToolkit/resources/HRTF
29      bool specifiedDelays;
30
31      ///Calling the function that loads the delays ///
32      bool sofaLoadResult = ...
            HRTF::CreateFromsofa("H3_44K_16bit_256tap_FIR_sofa.sofa", listener, ...
            specifiedDelays);
33
34
35      if (!sofaLoadResult) {
36          cout << "ERROR: Error trying to load the sofa file" << endl<<endl;
37      }
38
39      // Iicializacion de la fuente sonora
40      source1DSP = myCore.CreateSingleSourceDSP();    // Creating audio source
41      LoadWavFile(source1Wav, "speech_female.wav");   // Loading .wav file
42      Common::CTransform source1Position = Common::CTransform();
43      /*
44          Configuration of different parameters of the sound source
45          [...]
46      */
47
48      T_HRTFTable hrtf_table = listener->GetHRTF()->GetRawHRTFTable();
49
50      myCore.SetAudioState(audioState);
51      LoadITDGraphic(hrtf_table);
52      //AudioDevice Setup
53      // Before getting the devices list for the second time, the strean must be ...
            closed. Otherwise,
54      // the app crashes when systemSoundStream.start(); or stop() are called.
55      systemSoundStream.close();
56      SetDeviceAndAudio(audioState);
57  }
```

En la función de *update()* se actualizan las coordenadas de la fuente sonora, al usar la barra deslizante *azimut* para simular el sonido en la posición indicada.

En la función *draw()* se actualiza lo mostrado por pantalla, como puede ser el oyente, la fuente sonora respecto al oyente y una gráfica que muestra el ITD en el plano horizontal con un umbral y frecuencia de corte dados. La gráfica contiene una barra vertical deslizante que también se desplazará al modificar el valor de *azimut*. Esta parte del código no contiene nada relevante que se deba mencionar, pero si se quiere ver el código entero, en el siguiente apartado se facilitará el enlace al repositorio público, alojado en Github del proyecto. La interfaz del programa se muestra en la figura 3.3 y también ofrece la posibilidad de modificar algunos de los colores de la misma.

Al pulsar el botón *refresh* se activa una interrupción (una *callback*), mostrada en el listado 3.9, que detiene la ejecución principal del programa para volver a calcular los delays con los valores de umbral y frecuencia de corte presentes en las barras deslizantes. Esto dará como resultado una nueva gráfica representativa del ITD, mostrada en la parte de abajo.

Listado 3.9: Función de interrupción que es llamada al presionar el botón *refresh* y que llama de nuevo al método para calcular los nuevos retardos

```
1  void ofApp::refreshButtonPressed()
2  {
3      systemSoundStream.stop();
4      P = Umbral; //Se le da a la variable externa P el valor del umbral seleccionado
5      FC = FrecCorte*1000; //Se le da a la variable externa FC el valor de la ...
           frecuencia de corte seleccionada
6      bool specifiedDelays;
7      bool sofaLoadResult = ...
           HRTF::CreateFromsofa("H3_44K_16bit_256tap_FIR_sofa.sofa", listener, ...
           specifiedDelays);
8      if (!sofaLoadResult) {
9          cout << "ERROR: Error trying to load the sofa file" << endl << endl;
10     }
11
12     T_HRTFTable hrtf_table = listener->GetHRTF()->GetRawHRTFTable();
13
14     source1DSP->ResetSourceBuffers();
15     myCore.SetAudioState(audioState);
16
17     LoadITDGraphic(hrtf_table); //Func. auxiliar que dibuja la grafica del ITD
18     systemSoundStream.start();
19 }
```

3.3.1. Partes modificables del código y precauciones

Si se desea ejecutar el programa de este proyecto con archivos distintos a los usados se debe añadir los archivos a la carpeta del proyecto y modificar las siguientes lineas de código:

- En el listado 3.8, al inicio del *setup()*, donde se obtiene la frecuencia de muestreo del archivo, cambiarlo por el nombre del archivo deseado

  ```
  SAMPLERATE = HRTF::GetSampleRateFromsofa("ArchivoHRTF.sofa");
  ```

- Lo mismo en la función *CreateFromsofa()* que se encuentra un poco más abajo.

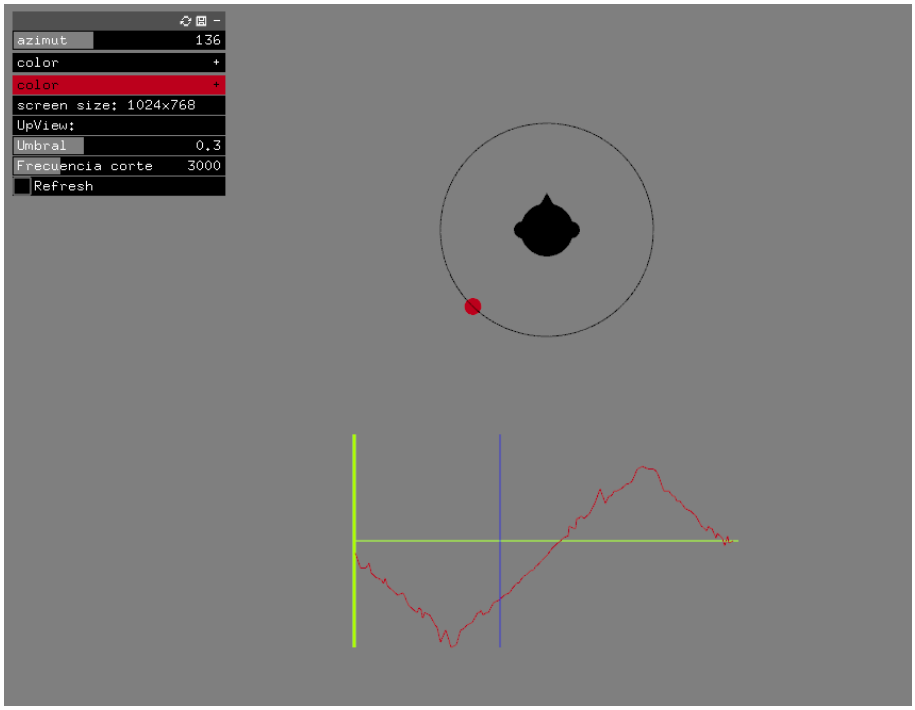


Figura 3.3: Interfaz del programa que permite la manipulación de una fuente sonora en el espacio y cargar HRTF con distintos umbrales y frecuencias de corte

```
HRTF::CreateFromsofa("ArchivoHRTF.sofa", ...
listener, specifiedDelays);
```

- También se puede modificar el archivo de audio que se reproduce en la fuente sonora cambiando el archivo *wav*

  ```
  LoadWavFile(source1Wav, "archivoAudio.wav");
  ```

- Por último hay que modificar el nombre del archivo *sofa* del listado 3.9, pero este archivo no debe tener retardos predefinidos, ya que en vez de calcular los retardos lo que hará será cargar de nuevo los del archivo.

3.4. Repositorio del código

Se buscaba partir de un ejemplo ya creado anteriormente, para evitar posibles problemas con la configuración de las dependencias del proyecto, y así crear la

aplicación de prueba dentro de Github[5]. Para llevar esto a cabo se realizaron los siguientes pasos:

1. *Fork* en Github, que es la creación de un nuevo proyecto en base a uno ya creado, del repositorio del Toolkit.
2. *Fork* en Github del repositorio de los ejemplos del Toolkit.
3. *Clonación* de los ejemplos del paso anterior en una carpeta local, dentro de la carpeta de OpenFrameworks.
4. Redirección del *Fork* del ejemplo apunte al Fork del Toolkit (que referencia como submódulo) cambiando la URL original por la del Fork del Toolkit.

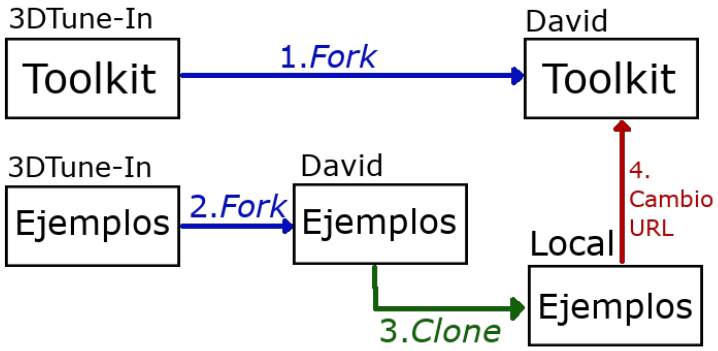


Figura 3.4: Representación del proceso para la creación del repositorio en Github del apartado 3.4.

[5] <https://github.com/David-Rodriguez-Lobaco/3dti_AudioToolkit_Examples/tree/master/example_4_interactive_basic_spatialisation_OF>

Capítulo 4

Estudio de los resultados

Contenido

Sinopsis

En este capítulo se mostrarán los distintos resultados obtenidos durante el capítulo 2.1 página 13, separándolos en bases de datos y como afectan a estos los diferentes umbrales y frecuencias de corte.

Las pruebas en MATLAB han seguido el siguiente orden:

- Diferentes umbrales con la misma frecuencia de corte para el filtro, la cual se recomendaba en un documento citado anteriormente [12].

- Elegir el umbral que se considera el mejor resultado de cada HRTF y otro umbral que contenga defectos (outliers) para ser sometidos a la misma prueba con distintas frecuencias de corte y comprobar como le afecta a cada uno, si empeora los resultados o si por el contrario corrige algunos errores.

4.1. Diferentes valores del umbral

4.1.1. SADIE

Los resultados con SADIE se separan en dos partes. La primera corresponde a las cabezas de muñeco D1 y D2; y la segunda a las cabezas humanas.

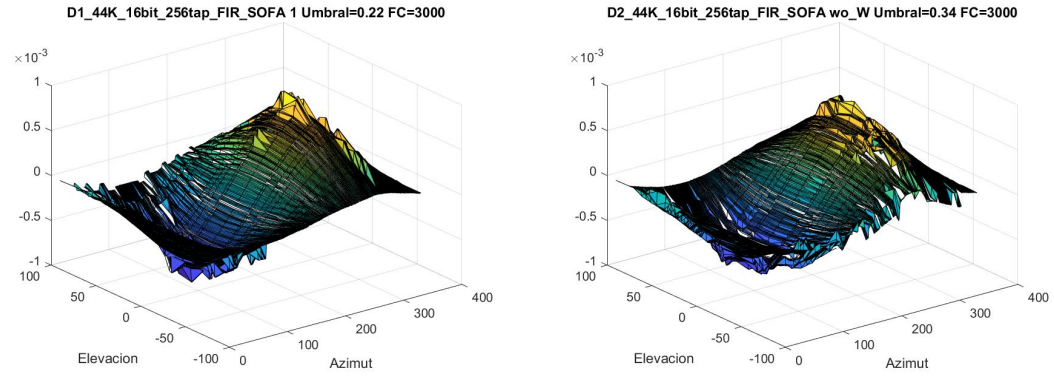
(a) Cabeza correspondientes a las HRTF de D1. [6][8]

(b) Cabeza correspondientes a las HRTF de D2. [6][17]

Figura 4.1: Formas de las cabezas de los muñecos usados para la base de datos de SADIE.[6]

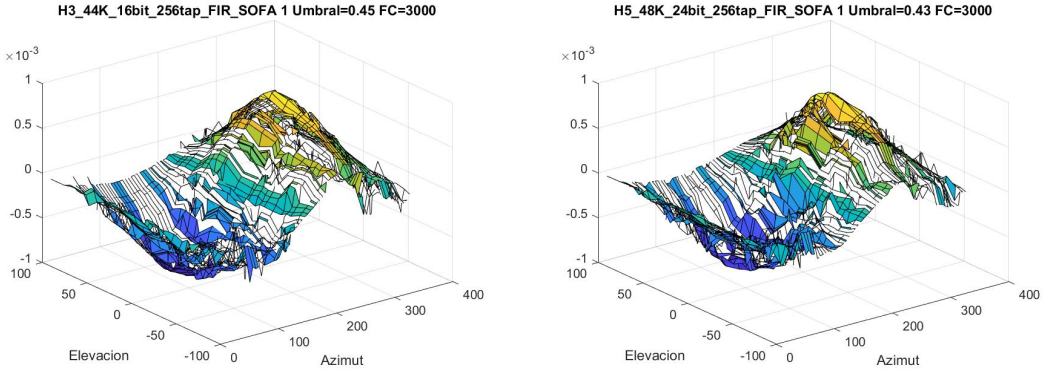
Las cabezas de los muñecos de las pruebas son distintas. Mientras que D1, figura 4.1a, tiene una cabeza más cuadrada, D2, figura 4.1b, se asemeja más a la cabeza humana. Esto da como resultado que los mejores umbrales de ambas cabezas difieran, como se muestra a continuación en la figura 4.2.

Para el caso de las cabezas humanas, los umbrales que mejor funcionan están entre el 42 % - 47 % del valor máximo del umbral.



(a) ITD de D1 con 22 % valor de umbral (b) ITD de D2 con 34 % valor de umbral

Figura 4.2: Mejores umbrales encontrados para los sujetos no humanos de SADIE



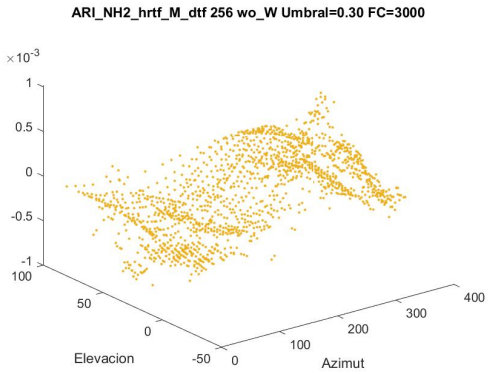
(a) ITD de H3 con 45 % valor de umbral (b) ITD de H5 con 43 % valor de umbral

Figura 4.3: Mejores umbrales encontrados para los sujetos humanos de SADIE

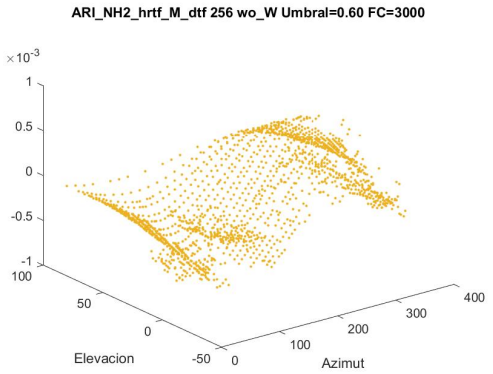
4.1.2. ARI

Para el caso de la base de datos ARI es más difícil la representación del ITD, pues los datos están ordenados de forma distinta a los de SADIE, aun así se pueden apreciar que umbrales se adaptan mejor a esta base de datos. Como comprobación adicional se ha representado el ITD en la horizontal pura para ver si se puede relacionar los resultados de los umbrales de ambas representaciones.

A primera vista se puede observar como un umbral del 30 % las figuras 4.4a y 4.5a se acercan a la forma que debe tener el ITD pero con bastantes puntos

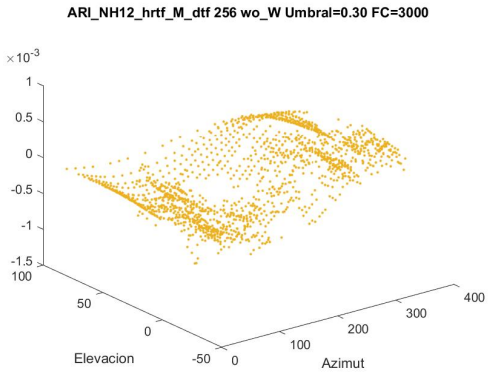


(a) ITD de NH2 con 30 % valor de umbral

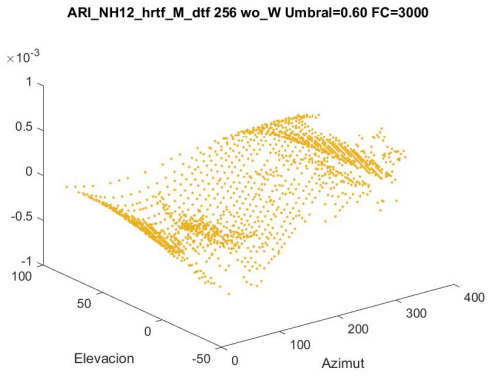


(b) ITD de NH2 con 60 % valor de umbral

Figura 4.4: Comparativa de umbrales usados en el sujeto NH8 de ARI



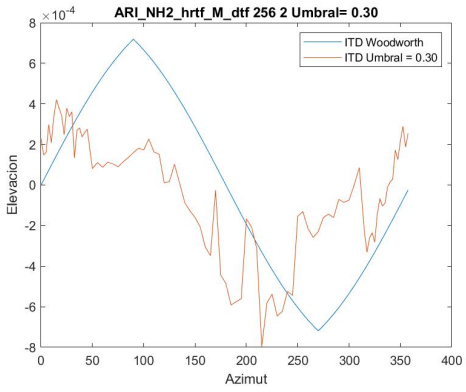
(a) ITD de NH12 con 30 % valor de umbral



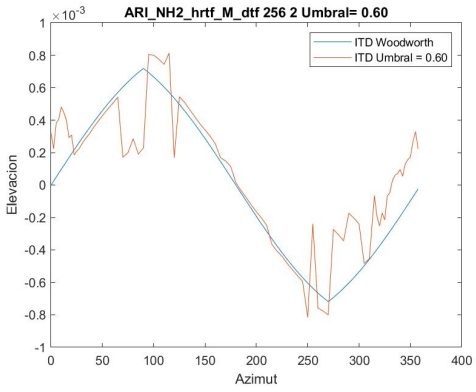
(b) ITD de NH12 con 60 % valor de umbral

Figura 4.5: Comparativa de umbrales usados en el sujeto NH12 de ARI

dispersos o outliers. Por el contrario, las figuras 4.4b y 4.5b se acercan más a una figura uniforme pero siguen apareciendo ciertos outliers, aunque en menor medida. En la figura 4.6 y 4.7 se muestra los ITD de las figuras 4.4 y 4.5 en la horizontal pura, linea roja, junto al ITD teórico de Woodworth, linea azul. Se comprueba como el ITD calculado con un 60 $\%$ de umbral es mejor que el de 30 $\%$.

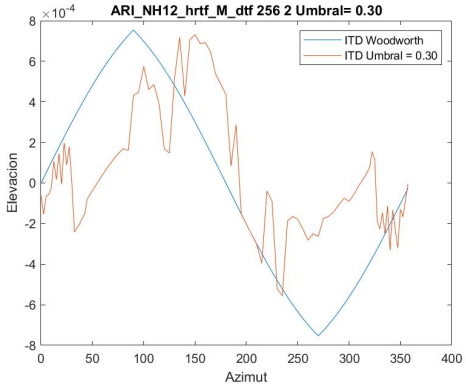


(a) ITD en la horizontal pura de NH2 con 30 % valor de umbral

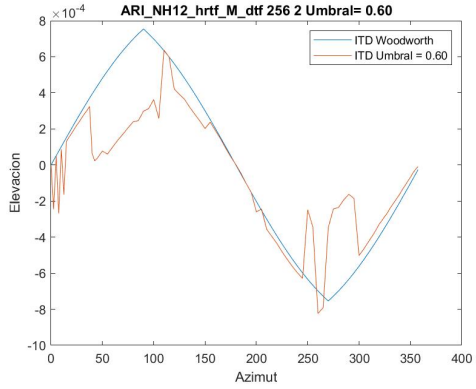


(b) ITD en la horizontal pura de NH2 con 60 % valor de umbral

Figura 4.6: Distintos umbrales usados en el sujeto NH8 de ARI



(a) ITD en la horizontal pura de NH12 con 30 % valor de umbral



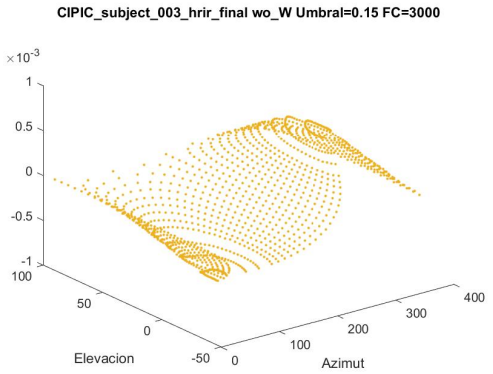
(b) ITD en la horizontal pura de NH12 con 60 % valor de umbral

Figura 4.7: Distintos umbrales usados en el sujeto NH12 de ARI

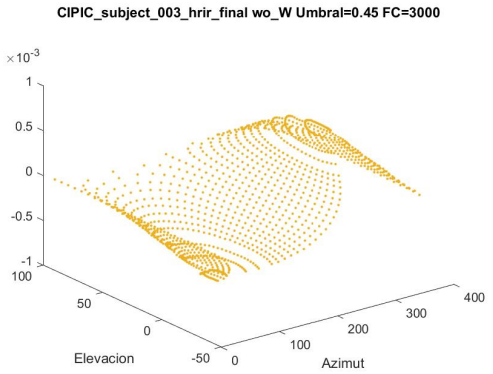
4.1.3. CIPIC

Los resultados obtenidos para la base de datos CIPIC se podrían considerar uniformes en comparación a las bases de datos anteriores. Aunque, como ocurre con ARI, el orden y los lugares de medida hacen difícil poder crear una gráfica que sea representativa; con los resultados obtenidos se puede apreciar bien las gráficas en las figuras 4.8 y 4.9.

Las pocas diferencias a destacar entre estos resultados es algún outlier que pudiese aparecer y diferencias en la amplitud del ITD al variar de umbral.

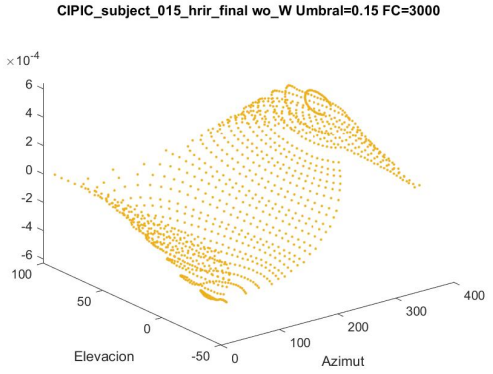


(a) ITD del sujeto 003 con 15 % valor de umbral

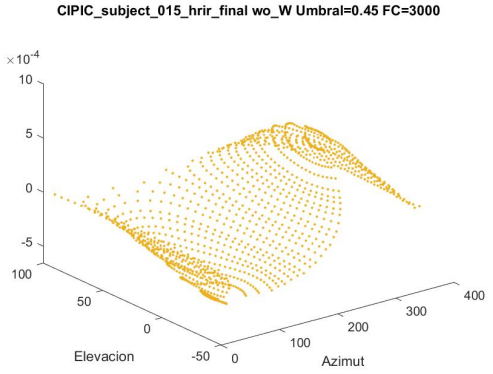


(b) ITD del sujeto 003 con 45 % valor de umbral

Figura 4.8: Comparativa de umbrales usados en el sujeto 003 de CIPIC



(a) ITD del sujeto 015 con 15 % valor de umbral



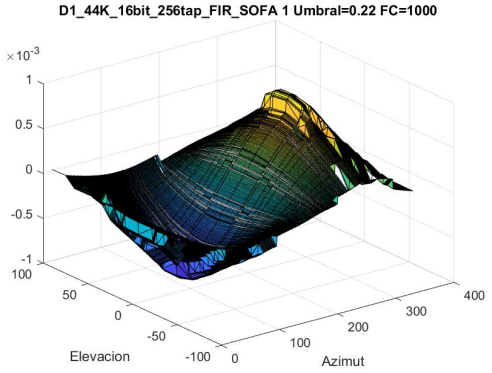
(b) ITD del sujeto 015 con 45 % valor de umbral

Figura 4.9: Comparativa de umbrales usados en el sujeto 015 de CIPIC

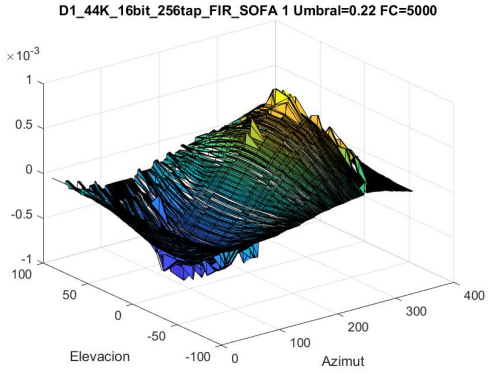
4.2. Diferentes valores de frecuencia de corte para el filtro

Mencionar que para los mismos umbrales del apartado anterior se han cambiado las frecuencias de corte. Estos cambios en la frecuencia en algunos casos ha mejorado los resultados

4.2.1. SADIE

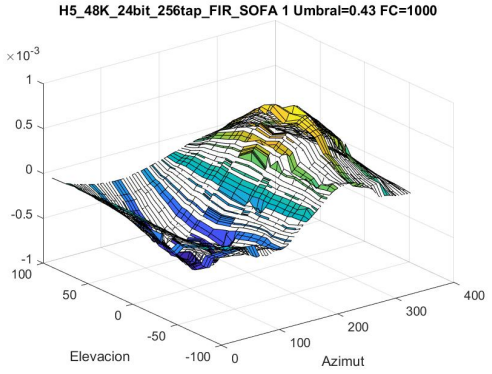


(a) ITD de D1 con una frecuencia de corte de 1KHz

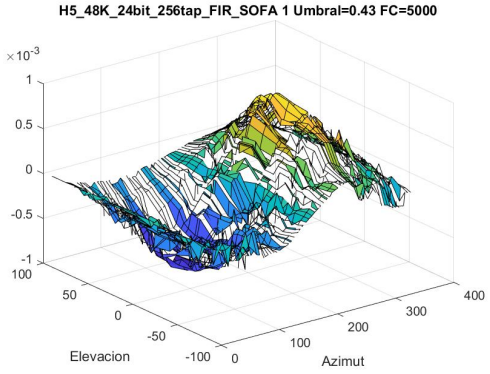


(b) ITD de D1 con una frecuencia de corte de 5KHz

Figura 4.10: Pruebas en el sujeto D1 con el umbral de la figura 4.2a pero con frecuencias de corte del filtro distintas



(a) ITD de H5 con una frecuencia de corte de 1KHz



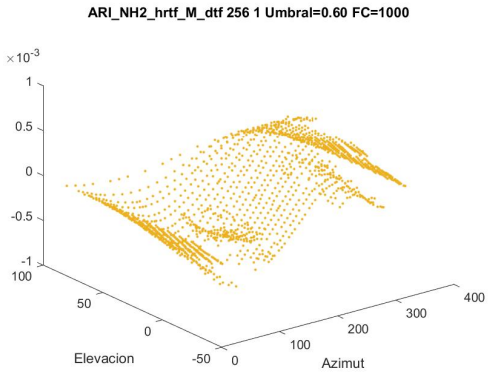
(b) ITD de H5 con una frecuencia de corte de 5KHz

Figura 4.11: Pruebas en el sujeto H5 con el umbral de la figura 4.3b pero con frecuencias de corte del filtro distintas

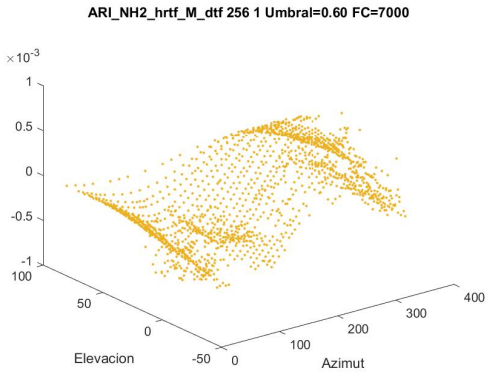
Las pruebas realizadas con distintas frecuencias de corte para el filtro de Butterworth para la base de datos SADIE, dan como conclusión que una frecuencia de corte menor a la usada a priori es mejor en este caso. Como se puede observar en las figuras 4.10 y 4.11, al usar una frecuencia de corte de 1KHz el ITD representado es más liso que en sus pruebas anteriores (figuras 4.2a y 4.3b

respectivamente). Por el contrario, al aumentar la frecuencia de corte aparecen más picos en la gráfica.

4.2.2. ARI

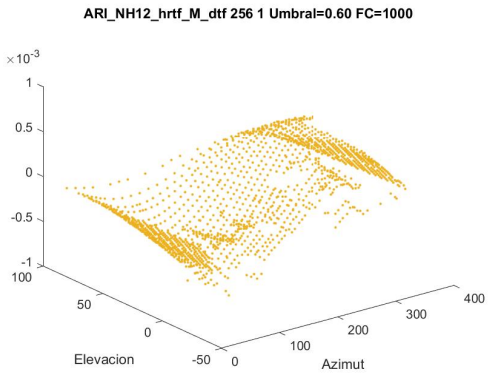


(a) ITD de NH2 con una frecuencia de corte de 1KHz y umbral del 60 %

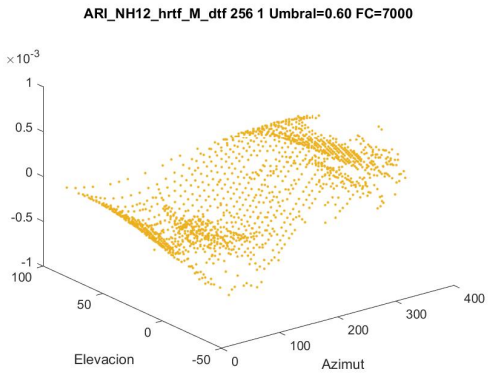


(b) ITD de NH2 con una frecuencia de corte de 7KHz y umbral del 60 %

Figura 4.12: Pruebas en el sujeto NH2 con el umbral de la figura 4.4b pero con frecuencias de corte del filtro distintas



(a) ITD de NH12 con una frecuencia de corte de 1KHz y umbral del 60 %



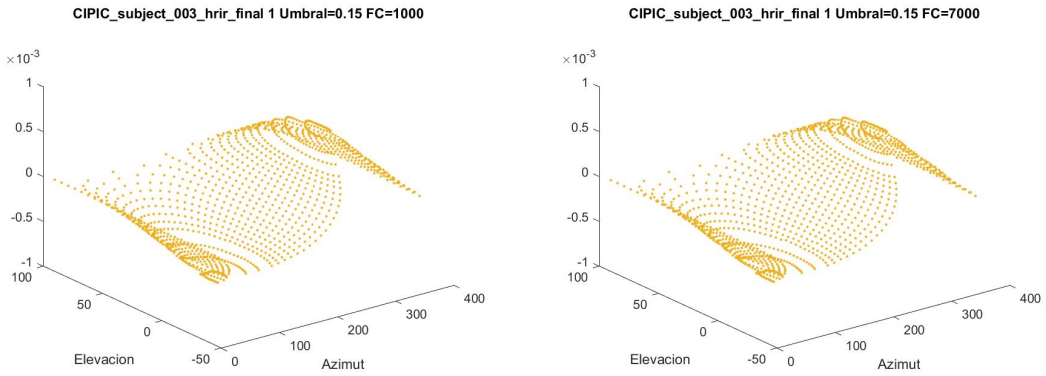
(b) ITD de NH12 con una frecuencia de corte de 7KHz y umbral del 60 %

Figura 4.13: Pruebas en el sujeto NH12 con el umbral de la figura 4.5b pero con frecuencias de corte del filtro distintas

Al igual que con la base de datos de SADIE, a la base de datos ARI le afecta mejor una frecuencia de corte para el filtro más baja, figuras 4.12a y 4.13a, frente

a una alta, figuras 4.12b y 4.13b. Aun así, los resultados se consideran lejos de lo que se busca, pero podría tener la explicación de que estos resultados se ven afectados por los hombros de las personas. Los outliers aparecen cuando la elevación es próxima a 0^o o por debajo, y próximos al azimut 90^o y 270^o, donde se encuentran los hombros. La explicación sería los rebotes que produce el sonido en esta parte del cuerpo.

4.2.3. CIPIC



(a) ITD del sujeto 003 con una frecuencia de corte de 1KHz y umbral del 15 %

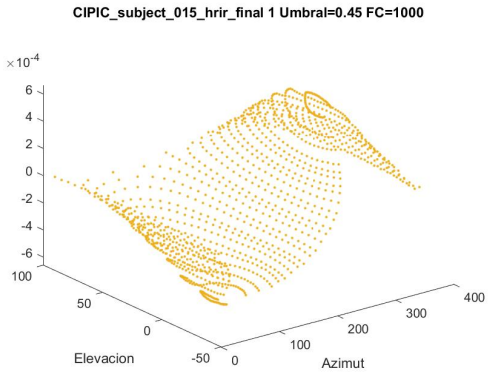
(b) ITD del sujeto 003 con una frecuencia de corte de 7KHz y umbral del 15 %

Figura 4.14: Pruebas en el sujeto 003 con el umbral de la figura 4.8a pero con frecuencias de corte del filtro distintas

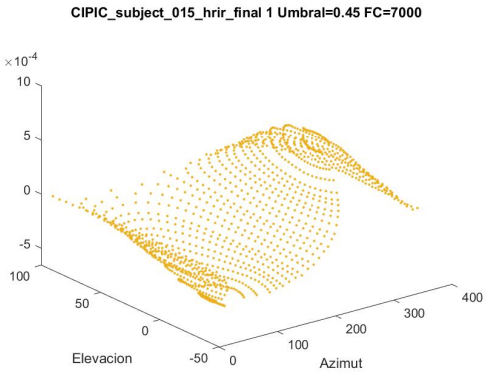
Los resultados de esta base de datos apenas varían entre ellos al modificar el umbral o la frecuencia de corte, así se puede observar al comparar la figura 4.8a con la figura 4.14; y la figura 4.9b con la figura 4.15, que a simple vista se podría decir que son prácticamente iguales.

4.3. Comparación de los resultados en el prototipo y en la Aplicación de Prueba

Para corroborar el correcto funcionamiento de la aplicación creada se realiza una comparativa entre las gráficas obtenidas en el prototipo realizado en MATLAB (Capítulo 2) y la aplicación (Capítulo 3). Dado que la aplicación puede proporcionar una gráfica en 2D de la horizontal pura se mostrarán gráficas de la horizontal pura en MATLAB.

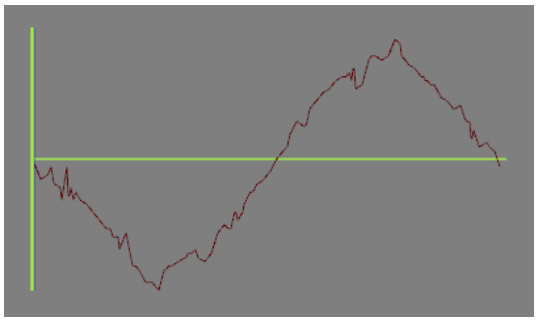


(a) ITD de del sujeto 015 con una frecuencia de corte de 1KHz y umbral del 45 %

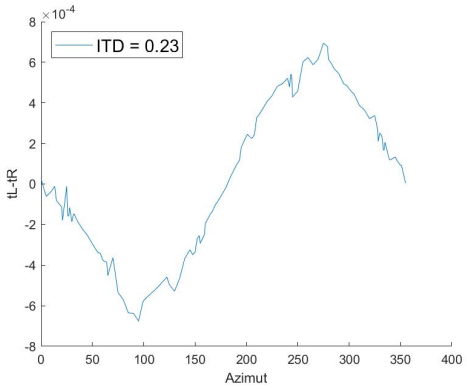


(b) ITD del sujeto 015 con una frecuencia de corte de 7KHz y umbral del 45 %

Figura 4.15: Pruebas en el sujeto 015 con el umbral de la figura 4.9b pero con frecuencias de corte del filtro distintas



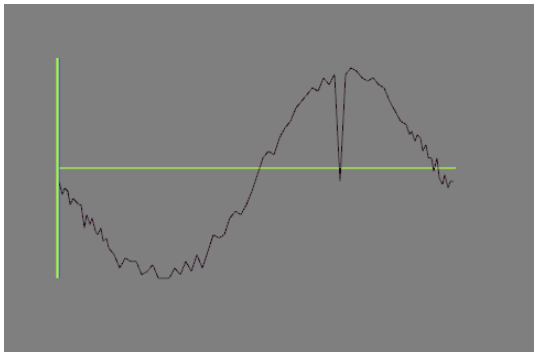
(a) ITD horizontal del sujeto H5 con un umbral del 23 % en la Aplicación de Prueba



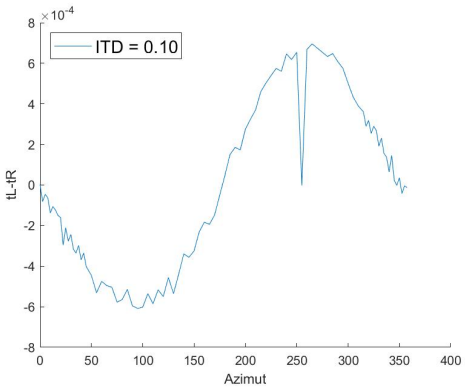
(b) ITD horizontal del sujeto H5 con un umbral del 23 % en el prototipo de MATLAB

Figura 4.16: Gráficas en la horizontal pura del sujeto H5 de SADIE para la comprobación de resultados en el prototipo y en la Aplicación de Prueba

Se han tomado como ejemplos los sujetos expuestos en las figuras 4.16 y 4.17. Se comprueba como ambos casos las gráficas resultantes son idénticas en cuanto a forma. Las figuras 4.16a y 4.17a presentan un pequeño error en el escalado que provoca que se represente un poco más abajo de lo que debería. Aún así, los datos obtenidos son los buscados y se puede asumir la correcta implementación del código.



(a) ITD horizontal del sujeto NH2 con un umbral del 10 % en la Aplicación de Prueba



(b) ITD horizontal del sujeto NH2 con un umbral del 10 % en el prototipo de MATLAB

Figura 4.17: Gráficas en la horizontal pura del sujeto NH2 de ARI para la comprobación de resultados en el prototipo y en la Aplicación de Prueba

A la vista de los resultados mencionados en este capítulo, se pueden sacar varias conclusiones expuestas en el siguiente capitulo.

Parte III

Parte tercera.

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones de los requisitos y las pruebas

Aquí se exponen los requisitos (tabla 2.1) y pruebas (tabla 2.2) del apartado 2.3 y 2.4 respectivamente, señalando cuales han sido superados y cuales no.

- Con las pruebas P1 (Mejor umbral) y P2 (Diferentes frecuencias de corte) se pretendía cumplir los requisitos R4 (Curva ITD) y R5 (*Outliers*). En el caso de R4 si se ha encontrado en todas las bases de datos umbrales y frecuencias de corte para los que la representación del ITD se acerca a la teórica. Por otro lado, no se ha cumplido el requisito R5 al no encontrar un modelo matemático u otra forma de corregir los *outliers*. Referente a las pruebas P1 y P2; P1 antes mostraba el ITD teórico junto al calculado en la interfaz, pero se decidió eliminar debido a que el usuario podría confundir el que no sean iguales exactamente con un ITD malo. Para la prueba P2, se tuvo que añadir nuevos coeficientes al filtro por cada frecuencia de corte añadida.

- Las pruebas P3 (Cargar *sofa*) y P4 (Actualizar umbral y frecuencia de corte) se encargaban de probar la inicialización del programa y actualización de los datos. En estas pruebas se comprobaban los requisitos R1 (Umbral), R2 (Frecuencia de corte), R3 (Retardos), R6(Cargar datos) y R8 (Visualización de ITD), llegándose a cumplir en su totalidad, como se puede ver en la figura 3.3

- Por último, la prueba P5 (Fuente sonora) pretendía satisfacer la interactividad del usuario final completando los requisitos R9 (Sonido), R10 (Manipulación del sonido) y R11 (Muñeco). Estos requisitos son superados. En esta

prueba se probó que ocurriría si se cargase una HRTF con los retardos entre los oídos intercambiados. Los resultados fueron que, debido a que ITD e ILD se calculan por separado, cuando el sonido, por ejemplo, se acerca al oído izquierdo se escucha más alto que por el derecho, pero en el derecho se escucha antes que en el izquierdo.

En resumen, se logra superar la mayoría de requisitos y las pruebas resultan exitosas, menos el R5, debido al problema descrito anteriormente.

Las conclusiones a las que se llegan en este proyecto a través de las pruebas y resultados obtenidos durante su duración son las siguientes:

- Cada base de datos tiene un conjunto de umbrales diferentes a los otros para los que se puede decir que el método se aproxima bastante al ITD deseado. Las frecuencias de corte del filtro también influyen en los resultados, por lo que no se puede definir un rango de ambas variables que asegure buenos resultados con todas las bases de datos.
- Diferentes filtros producen distintos resultados. En el TFG del que se procede[1] se usaba un filtro distinto al usado en este proyecto, y como resultado los valores que se consideran óptimos son diferentes en ambos.
- Aunque en cierto momento se llegó a probar distintos tipos de suavizado y eliminación de ruido que se encuentran en MATLAB[2], como pueden ser *filloutliers()*, que detecta y reemplaza *outliers* en los datos, o un filtrado de Savitzky-Golay (*sgolayfilt()*); no se halló uno que mejorase significativamente los resultados.

5.2. Lineas futuras

Este Trabajo Fin de Grado debe continuarse mediante la búsqueda de una forma de automatizar la selección del umbral y la frecuencia de corte en el método del umbral, que aquí se quedan como parámetros que se pueden, al menos, seleccionar de forma interactiva. Mientras se escribe esta memoria se está colaborando con investigadores especializados en eliminación de ruido en medidas de datos mediante métodos basados en teoría de la información [7].

Además, existen distintas funciones que se pueden mejorar o incorporar al programa, como son:

[1] MÉTODOS DE EXTRACCIÓN DEL ITD, Jesús Rodríguez Galán, 2020

[2] <https://es.mathworks.com/help/signal/smoothing-and-denoising.html?s_tid=CRUX_lftnav>

- Permitir cambiar de HRTF entre las disponibles sin tener que salir del programa con una ventana desplegable.
- Añadir nuevos filtros.
- Añadir la opción de cambiar la elevación de la fuente sonora y que la gráfica muestre el ITD de la elevación actual. De esta forma se obtendría más información sobre el filtro, umbral y frecuencia de corte elegidos.
- Mejorar el método para suavizar los *outliers*.
- Incorporar la opción de activar o desactivar distintas funciones que ofrece el Toolkit, como es la reverberación.

Bibliografía

[1] Neil L Aaronson and William M Hartmann. Testing, correcting, and extending the woodworth model for interaural time difference. *The Journal of the Acoustical Society of America*, 135(2):817–823, 2014.

[2] V Ralph Algazi, Richard O Duda, Dennis M Thompson, and Carlos Avendano. The cipic hrtf database. In *Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No. 01TH8575)*, pages 99–102. IEEE, 2001. [PDF; accedido 27-11-2022].

[3] Stanley T Birchfield and Rajitha Gangishetty. Acoustic localization by interaural level difference. In *Proceedings.(ICASSP'05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, volume 4. IEEE, 2005.

[4] Maria Cuevas Rodriguez et al. 3d binaural spatialisation for virtual reality and psychoacoustics. 2022.

[5] Maria Cuevas-Rodríguez, Daniel González-Toledo, Ernesto De La Rubia-Cuestas, Carlos Garre, Luis Molina-Tanco, Arcadio Reyes-Lecuona, David Poirier-Quinot, and Lorenzo Picinali. The 3d tune-in toolkit–3d audio spatialiser, hearing loss and hearing aid simulations. In *2018 IEEE 4th VR workshop on sonic interactions for virtual environments (SIVE)*, pages 1–3. IEEE, 2018. [Web; accedido 15-11-2022].

[6] University of York Department of Electronics. Sadie ii database. <https://www.york.ac.uk/sadie-project/database.html>, 09 2019. [Web; accedido 27-11-2022].

[7] Domenico Giordano and Felice Iavernaro. Maximal-entropy driven determination of weights in least-square approximation. *Mathematical Methods in the Applied Sciences*, 44(8):6448–6461, 2021.

[8] Georg Neumann Gmbh. Dummy head ku 100. <https://es-es.neumann.com/ku-100>, 2018-2022. [Web; accedido 15-12-2022].

[9] Acoustics Research Institute. Ari hrtf-database. <https://www.oeaw.ac.at/en/isf/das-institut/software/hrtf-database>, 2014. [Web; accedido 27-11-2022].

[10] Jaan Johansson, Aki Mäkivirta, Matti Malinen, and Ville Saari. Interaural time difference prediction using anthropometric interaural distance. *Journal of the Audio Engineering Society*, 70(10), 2022.

[11] Brian FG Katz and Markus Noisternig. A comparative study of interaural time delay estimation methods. *The Journal of the Acoustical Society of America*, 135(6):3530–3540, 2014.

[12] Brian FG Katz and Markus Noisternig. A comparative study of interaural time delay estimation methods. *The Journal of the Acoustical Society of America*, 135(6):3530–3540, 2014.

[13] DPA Microphones. The basics about comb filtering (and how to avoid it). <https://www.dpamicrophones.com/mic-university/the-basics-about-comb-filtering-and-how-to-avoid-it>. [Web; accedido 09-08-2022].

[14] Jesus Rodríguez Galán. Métodos de extracción del itd. 2020.

[15] AES (Audio Engineering Society). Aes69-2015, spatial acoustic data file format. *AES STANDARDS NEWS BLOG*, 05/03/2015. [Web; accedido 07-08-2022].

[16] Enrique Valido Moure. Medida de la hrtf. B.S. thesis, Universidad de las Palmas de Gran Canaria, 2014.

[17] GRAS Sound Vibration. Gras 45bb-1 kemar head torso for hearing aid test, 1-ch lemo. <https://www.grasacoustics.com/products/head-torso-simulators-kemar/kemar-for-hearing-aid-test-1-ch/product/499-45bb-1>. [Web; accedido 15-12-2022].