ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Mejora de una aplicación para localización de fuentes acústicas en tiempo real

AUTOR: Javier Tacero Díaz

TITULACIÓN: Grado en Ingeniería de Sonido e Imagen

TUTOR: Rubén Fraile Muñoz

DEPARTAMENTO: Ingeniería Telemática y Electrónica

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Eduardo Nogueira Díaz

TUTOR: Rubén Fraile Muñoz

SECRETARIO: Nicolás Sáenz Lechón

Fecha de lectura: 29/10/2020

Calificación:

El Secretario,

Mejora de una aplicación para localización de fuentes acústicas en tiempo real

Resumen

La localización de fuentes acústicas es un tema que cuenta actualmente y desde hace años con cierta relevancia dentro del mundo de la investigación. Son numerosos los artículos científicos que tratan sobre el tema, así como las universidades y centros de investigación que en ello trabajan.

Cuando se habla de localización de fuentes acústicas ésta se refiere a estimar la dirección o posibles direcciones de llegada de una onda sonora proveniente de un emisor mediante el procesado de señales captadas por un array de micrófonos, y la distancia a la que se encuentra la fuente si así fuese requerido. La localización puede ser estimada en un plano o en un espacio tridimensional, dependiendo del número de micrófonos del array y de su disposición. El grado de precisión también varía en función del número de micrófonos utilizado en el proceso.

Existen numerosos métodos de cálculo de localización que evalúan la señal temporal, la señal espectral u otras características. En principio, en este proyecto se hará uso de la TDOA (Time Difference Of Arrival) como método de cálculo para obtener hipérbolas de las posibles posiciones. Para lograr el objetivo de hallar una posición se hará uso del algoritmo de Gauss-Newton para resolver el problema que surge al determinar cuál es la posición de la fuente sonora a partir de las hipérbolas obtenidas.

El resultado final de este proyecto es la obtención de una aplicación informática compatible con Windows capaz de estimar la dirección y posición de una fuente sonora en tiempo real implementando algoritmos que proporcionen precisión y robustez a la aplicación partiendo de una aplicación diseñada en un PFG anterior como base.

Las conclusiones extraídas de este proyecto revelan la inadecuación de Windows para desarrollar aplicaciones de captura de audio, así como las condiciones que deben darse para un adecuado funcionamiento del algoritmo de Gauss-Newton para estimar la posición correcta. Por último, se establecen las bases para futuros proyectos en este campo de investigación.

Improvement of an application for location of acoustic sources in real time

Abstract

The location of acoustic sources is a subject that has been of some relevance in the world of research for many years. There are numerous scientific articles on the subject, as well as universities and research centres working on it.

When talking about acoustic source location, this refers to estimating the direction or possible directions of arrival of a sound wave coming from a transmitter by means of the processing of signals captured by an array of microphones, and the distance to the source if required. The location can be estimated on a plane or in three-dimensional space, depending on the number of microphones in the array and their arrangement. The degree of precision also varies depending on the number of microphones used in the process.

There are numerous location calculation methods that evaluate the temporal signal, the spectral signal or other characteristics. In principle, in this project the TDOA (Time Difference Of Arrival) will be used as a calculation method to obtain hyperbolas of the possible positions. To achieve the objective of finding a position, the Gauss-Newton algorithm will be used to solve the problem that arises when determining the position of the sound source from the obtained hyperboles.

The final result of this project is a Windows-compatible computer application capable of estimating the direction and position of a sound source in real time by implementing algorithms that provide precision and robustness to the application, using a previous PFG as a basis.

The conclusions drawn from this project reveal the inadequacy of Windows for developing audio capture applications, as well as the conditions that must be met for the proper functioning of the Gauss-Newton algorithm to estimate the correct position. Finally, the bases are established for future projects in this field of research.

Índice

Lista de acrónimos

API	Application Programming Interface
WASAPI	Windows Audio Session API / Interfaz de programación de Windows para la gestión de flujos de audio
MMDEVICE	Windows Multimedia Device API / Interfaz de programación de Windows para detectar dispositivos y sus características
TDOA	Time Difference of Arrival / Diferencia de tiempos de llegada
MFC	Microsoft Foundation Classes / Biblioteca de Windows para desarrollar aplicaciones de escritorio
IDE	Integrated Development Environment / Entorno de desarrollo integrado
GCC	Generalized Cross-Correlation / Función de correlación cruzada generalizada
PHAT	The Phase Transform
SCOT	The Smoothed Coherence Transfom
GN	Gauss-Newton / Método iterativo de Gauss-Newton
NLLS	Non-Linear Least Squares / Mínimos cuadrados no lineales
WAV	Waveform Audio Format / Formato de audio de forma de onda

1. Introducción

Este proyecto nace como la continuación de un Proyecto de Fin de Grado presentado en 2018 llamado "Procesamiento de audio en tiempo real para estimar la dirección de fuentes sonoras", de Juan Carlos Pereira Martínez [1]. Aquel trabajo culminó con la creación de una aplicación para Windows que era capaz de estimar la dirección de una fuente sonora a partir de dos señales captadas por micrófonos en tiempo real. Dichos micrófonos van conectados al ordenador, ya sea mediante una interfaz de sonido externa o directamente a las entradas minijack o USB del ordenador. El algoritmo implementado se basaba en la técnica de la TDOA (*Time Difference of Arrival* o diferencia de tiempos de llegada), en la cual se obtiene el retardo mediante la correlación cruzada de las dos señales captadas por los micrófonos. A partir de este retardo se obtenía una hipérbola que era dibujada en un plano para representar la dirección estimada de la fuente acústica. En la interfaz de la aplicación, el usuario podía introducir los parámetros necesarios para el cálculo como el tamaño del buffer, la separación de micrófonos o el dispositivo de entrada del cual se iban a tomar las muestras.

Dicha aplicación fue escrita en lenguaje C++ y compilado en el entorno de desarrollo Microsoft Visual Studio, lo que ha marcado determinantemente el desarrollo de este proyecto. El proyecto fue desde el principio diseñado, según palabras del propio autor, para servir de base para futuras ampliaciones y mejoras en las funcionalidades de la aplicación. Además, el código fuente original está estructurado con el objeto de ser reutilizado por futuros usuarios. Una parte significativa del presente proyecto está constituido por código fuente del proyecto original que ha sido modificado en mayor o menor medida para adaptarlo a las necesidades del proyecto. A lo largo de esta memoria se detalla qué partes de código son reutilizadas y qué cambios se han efectuado en ellas. Además de esto, el proyecto original contaba con un anexo en el que se proponía una solución para capturar audio de distintos dispositivos. De esta parte no ha sido tanto la reutilización de código como sí algunas ideas conceptuales las que se han tomado y han sido de gran ayuda para implementar la captura de varios dispositivos en esta aplicación.

Uno de los principales objetivos del proyecto base era implementar una solución que fuera capaz de procesar audio utilizando únicamente las librerías propias de Windows para audio: WASAPI (*Windows Audio Session API* o API de sesión de audio de Windows) [2] y MMDevice (*Windows Multimedia Device API* o API de dispositivos multimedia de Windows) [3]. Con esto no solo se aseguraba una total compatibilidad con el entorno Windows, sino que se evitaba la dependencia de otras librerías de audio que, aunque ofrecían mayores prestaciones, requerían el pago de una licencia. Dicho objetivo fue cumplido exitosamente en aquella ocasión; el software era plenamente operativo y cumplía su función correctamente. En esta ocasión se mantiene ese mismo objetivo, pero esta vez llevando al límite las capacidades de las librerías de Windows.

Para esta nueva versión de la aplicación se han llevado a cabo sustanciales modificaciones. La principal diferencia es que se pasa de estimar una dirección a estimar una posición exacta en la que se estima que se encuentra la fuente acústica, también a partir de los retardos entre micrófonos. Para la estimación de un punto en el plano se ha hecho uso de herramientas matemáticas de optimización de cierta complejidad ya que la estimación de la posición de la fuente a partir de distintos retardos no es trivial. Además, se implementa un nuevo método de estimación del retardo entre micrófonos, también basado en la técnica TDOA, y que ofrece mayores prestaciones debido a su mayor versatilidad.

Originalmente el programa admitía dos canales de audio de un mismo dispositivo de captura estéreo con una profundidad de bits de 16. En esta ocasión, el código está diseñado para trabajar con n canales de n dispositivos distintos para trabajar con una profundidad de bits de 16, 24 o 32. Además de trabajar con audio capturado in situ se ha añadido un modo de trabajo para procesar ficheros de audio multicanal pregrabados en formato WAV (*Waveform Audio Format* o formato de audio de forma de onda).

Aunque, como se ha mencionado, el código está preparado para soportar un número indeterminado de canales, en esta solución se ha limitado a 8 canales como máximo para la adecuación del algoritmo a una interfaz de usuario que sea manejable.

En esta interfaz de usuario se configura la disposición de los micrófonos utilizados para el proceso de captura, ya sea un proceso de captura in situ o a partir de un fichero pregrabado.

Además, la interfaz incluye un control visual para simular el proceso de localización en una sala. Lo que se representa de esta sala es la planta, de unas dimensiones introducidas por el usuario y en la que se configura la posición en la sala del array de micrófonos utilizado en la captura de audio. Así, durante el proceso de estimación de la posición, las hipérbolas obtenidas a partir de los retardos se dibujan en el plano de la sala, dependiendo de cuál sea la posición del array. Los resultados de la estimación se muestran en pantalla a la vez que se escriben en un fichero de texto.

La memoria de este proyecto se ha estructurado de la siguiente manera. En primer lugar, se establece el marco tecnológico en el cual se encuadra esta solución. En el siguiente apartado se establece el modelo teórico de la estimación de fuentes acústicas, así como la descripción de las herramientas matemáticas utilizadas para tal fin. Después, se establecen los objetivos principales del proyecto y se describen las especificaciones de la aplicación resultante. A continuación, se pasa a detallar la estructura del código fuente de la aplicación, haciendo hincapié en los aspectos más importantes del diseño, en cuál es el flujo de los datos y en la descripción detallada de la función encargada a cada clase. Tras haber expuesto la solución en sí, se detallan las pruebas a las cuales se ha sometido la aplicación, ya sea la aplicación en sí en casos reales o sus componentes de forma aislada mediante simulaciones. Después se incluye un apartado en el que se citan los problemas notificados durante el desarrollo de la aplicación o durante las pruebas y qué soluciones se han implementado o propuesto. Por último, se desarrollan las conclusiones, en las que se reflexiona sobre la adecuación de las librerías de Windows para aplicaciones de este tipo y se proponen líneas de actuación futuras para la estimación de fuentes sonoras.

La memoria contiene además dos anexos. En el primer anexo se encuentra el manual de usuario de la aplicación, mientras que en el segundo se detalla el presupuesto del proyecto.

2. Marco tecnológico y técnico

2.1 Antecedentes

La localización de un emisor a partir de la TDOA no es un concepto de reciente creación. De hecho, esta técnica lleva usándose desde hace décadas en el ámbito de la navegación, desde que a finales de la Segunda Guerra Mundial fueran puestos en marcha los primeros sistemas de navegación hiperbólica, técnica también conocida como multilateración [4]. Dichos sistemas trazaban hipérbolas a partir de las diferencias de tiempos de llegada de una señal de radio emitida por un transmisor y captada por varios receptores. El lugar en el que las hipérbolas intersecaban era la posición estimada del emisor.

Este proyecto sigue el mismo principio, solo que en esta ocasión es aplicado a señales acústicas en vez de electromagnéticas. Desde hace años, son varios los artículos publicados en esta línea de investigación que proponen distintos métodos para hallar la dirección de una fuente sonora o su posición.

En ocasiones la estimación de la posición de la fuente sonora va asociada a la detección de eventos acústicos, donde a partir del procesado del audio se identifica la naturaleza de la fuente acústica, es decir el quien o el qué genera ese sonido. Junto con la estimación de la posición, estos sistemas son potenciales herramientas para el monitorizado de escenarios, en los que solo con la señal captada por un array de micrófonos se puede saber en qué lugar se ha producido un determinado evento acústico y qué es lo que lo genera.

El presente proyecto se enmarca en la estimación de la posición de una única fuente acústica estática en un plano a partir de la TDOA. La estimación de la TDOA también ocupa una parte importante en el proyecto ya que se implementa un método avanzado de correlación cruzada propuesto en 1976 [5] y que desde entonces ha ocupado un importante puesto en soluciones de estimación de posición y dirección. Para la resolución del problema NLLS (*Non-Linear Least Squares* o mínimos cuadrados no lineales) surgido al calcular las intersecciones de la hipérbola se ha optado por el algoritmo de Gauss-Newton, ya utilizado anteriormente en sistemas de radionavegación hiperbólica.

2.2 Marco tecnológico

En cuanto al proceso de adquisición de audio, se aprovechan las funcionalidades que ofrece el sistema operativo Windows a través del *Windows Core Audio APIs* [6], un conjunto de interfaces de programación de bajo nivel para capturar o reproducir audio de manera optimizada. Fue introducido con el lanzamiento del sistema operativo Windows Vista en 2007 con posteriores ampliaciones y mejoras en los lanzamientos de Windows 7 y Windows 10. Estas interfaces suponen una evolución con respecto a *DirectSound* [7], la librería de audio predominante en Windows hasta entonces. Este conjunto de interfaces lo forman cuatro interfaces de programación:

- **MMDevice API**. Permiten al usuario detectar e identificar los dispositivos de audio, ya sean de entrada o de salida, disponibles en el sistema.
- **WASAPI.** Permite al usuario a crear y gestionar flujos de datos de audio entre una aplicación y los dispositivos de audio.
- **DeviceTopology API**. Permite el acceso a parámetros más específicos del hardware, como controles de graves y agudos, ganancia automática, etc., que no pueden ser accedidos desde el resto de las interfaces.
- **EndpointVolume API**. Permite al usuario controlar directamente el volumen de los dispositivos de audio.

Principalmente, en esta aplicación se hace un uso importante de las interfaces **WASAPI** y **MMDevice**, y en menor de medida de **EndpointVolume.**

El lenguaje de programación utilizado para la creación de la aplicación es C++. Este lenguaje junto con C# son los que pueden integrar las interfaces del *Windows Audio Core*. La razón principal de haber escogido C++ radica en que fue el lenguaje utilizado en la primera versión de la aplicación desde la que parte este proyecto. Al ser C++ un lenguaje de más bajo nivel que C#, proporciona algunas herramientas que pueden ser valiosas a la hora de mejorar el rendimiento de los algoritmos.

Para el diseño de las interfaces de usuario y el manejo de los hilos de trabajo de la aplicación, se trabaja con la librería MFC (*Microsoft Foundation Classes*) [8]. Esta librería es la recomendada por Microsoft para la creación de aplicaciones de escritorio en C++. Permite diseñar desde aplicaciones simples hasta otras de complejidad muy alta debido a la ingente cantidad de funciones que posee. Precisamente la alta complejidad de esta biblioteca ha supuesto una considerable dificultad a la hora de diseñar esta aplicación. No obstante, a cambio de esta complejidad lo que se obtiene es un número de posibilidades casi ilimitadas para diseñar una aplicación de escritorio; desde la variedad de los controles visuales hasta la realización de operaciones básicas de entrada/salida, almacenamiento de colecciones de objetos de datos, etc. Una de las funcionalidades que ofrece MFC y que se ha implementado en esta aplicación es la posibilidad del usuario de definir su propio mapa de mensajes para establecer un sistema de notificación de mensajes entre los diferentes módulos (clases) que componen la solución. Por suerte, MFC cuenta con una extensa y detallada documentación que permite, no solo al programador sino también a usuarios que quieran introducirse en el código, inferir con relativa facilidad cual es la función de una sección determinada.

2.3. Modelo de estimación de la posición de un emisor con dos micrófonos

Se consideran dos micrófonos situados sobre el eje x a una distancia ***D*** entre ellos y centrados en el eje y. En la posición (x_s,y_s) se sitúa una fuente acústica puntual. En la figura 1 se representa este modelo.

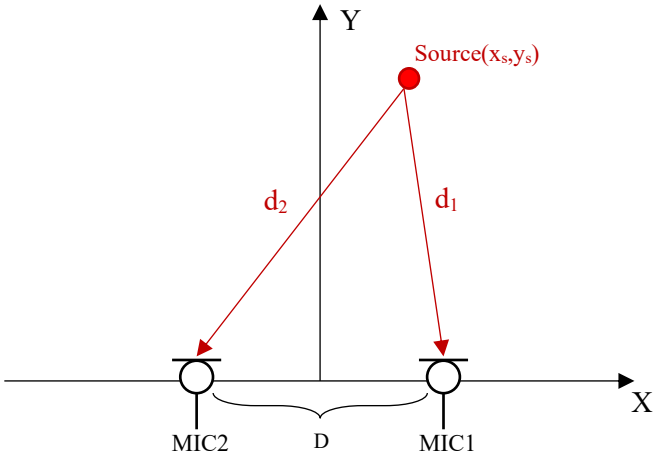

Figura 1. *Esquema de dos micrófonos en un plano para localizar emisor.*

Las distancias entre la fuente y los micrófonos se definen como:

$$d_1 = \sqrt{\left(D/2 - x_s\right)^2 + y_s^2} \qquad (1)$$

$$d_2 = \sqrt{\left({}^{D}/_{2} + x_s\right)^2 + y_s^2} \tag{2}$$

Por lo tanto, la diferencia de distancias queda como la ecuación (3):

$$\Delta d = d_2 - d_1 = \sqrt{\left({}^{D}/_{2} + x_s\right)^2 + y_s^2} - \sqrt{\left({}^{D}/_{2} - x_s\right)^2 + y_s^2} \tag{3}$$

Aquellos puntos del plano cuya diferencia de distancias a **MIC1** y **MIC2** es igual a Δd son infinitos y todos ellos forman una hipérbola cuyo centro coincide con el origen de coordenadas y cuyo eje focal coincide con el eje x. Precisamente esto coincide con la descripción de la hipérbola.

Citando la definición de Lehmann de una hipérbola [9]:

> *"Una hipérbola es el lugar geométrico de un punto que se mueve en un plano de tal manera que el valor absoluto de la diferencia de sus distancias a dos puntos fijos del plano, llamados focos, es siempre igual a una cantidad constante, positiva y menor que la distancia entre los focos."*

Dicho esto, se deben puntualizar ciertas características de nuestro modelo. El signo del término Δd resulta en este caso imprescindible para escoger la rama hiperbólica correcta. Bajo esta condición, el termino Δd debe ser constante, menor que la distancia entre focos y positivo (o negativo).

La ecuación canónica de la hipérbola (4) cuyo centro se halla en el origen de coordenadas (0,0) y cuyo eje focal es paralelo con el eje x tiene la siguiente forma:

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1 \tag{4}$$

En consecuencia, reordenando convenientemente los términos de la ecuación (3) se obtiene la ecuación de la hipérbola (5):

$$\frac{x^2}{{}^{\Delta d^2}/_{4}} - \frac{y^2}{{}^{D^2}/_{4} - {}^{\Delta d^2}/_{4}} = 1 \tag{5}$$

Por lo tanto, para obtener la hipérbola se tienen dos incógnitas: la distancia entre micrófonos D y la diferencia de distancia recorrida por el sonido Δd. La distancia entre micrófonos es un valor ya conocido por lo que no necesita de ningún cálculo mientras que Δd se puede obtener multiplicando el retardo τ entre micrófonos por la velocidad del sonido. En el siguiente apartado se detalla la técnica utilizada para obtener este retardo τ, denominada TDOA.

2.4. Método de estimación del retardo entre micrófonos

Para estimar el retardo temporal entre micrófonos se aplica el método de la GCC (*Generalized Cross-Correlation* o correlación cruzada generalizada) [5]. El método de la GCC es el más utilizado para estimar diferencias de tiempos de llegada entre dos señalas capturadas desde dos micrófonos distintos. La principal diferencia entre este método y la correlación cruzada estándar utilizada en el

proyecto original es que aquí se opera en el dominio de la frecuencia, lo que permite filtrar las señales para así obtener mejores resultados.

La técnica descrita supone entornos donde las características de la señal y el ruido permanecen estables durante un tiempo. Partiendo de un modelo teórico en el que una misma señal llega a dos micrófonos situados en distintas posiciones (figura 2) y considerando que s_1 es una señal incorrelada con el ruido n_1 y n_2, puede calcularse el retardo D aplicando la correlación cruzada:

$$R_{x1x2}(\tau) = E[x_1(t)x_2(t-\tau)] \tag{6}$$

La E se refiere a la esperanza matemática mientras que $\boldsymbol{\tau}$ es la variable que proporciona una estimación del retardo, denominada $\widehat{D}$. Dado que el tiempo de observación es finito, la función de correlación solo puede ser estimada, no calculada.

En x_2, α representa la atenuación de la señal captada en MIC 2 con respecto a la señal captada en MIC 1.

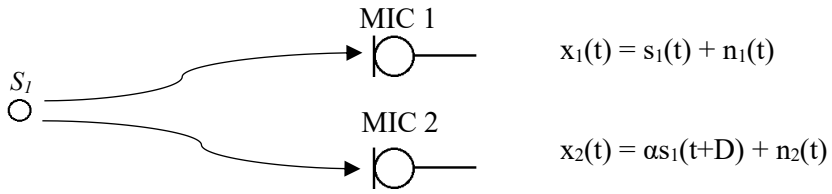

***Figura 2**. Esquema señal emitida desde una fuente en un entorno ruidoso.*

Para un proceso ergódico[1], una estimación de la correlación cruzada viene dada por la siguiente expresión:

$$\hat{R}_{x1x2}(\tau) = \frac{1}{T-\tau}\int_{\tau}^{T} x_1(t)x_2(t-\tau)dt \tag{7}$$

donde T representa el intervalo de observación. Para mejorar la precisión en la estimación de $\widehat{D}$ es conveniente filtrar $x_1(t)$ y $x_2(t)$ antes de integrar.

Como se muestra en la figura 3, x_i puede ser filtrada por H_i dando como resultado y_i para i=1,2. La función de los filtros es acentuar las frecuencias en las cuales la relación S/N es mayor para un mejor funcionamiento del correlador, así como eliminar el ruido.

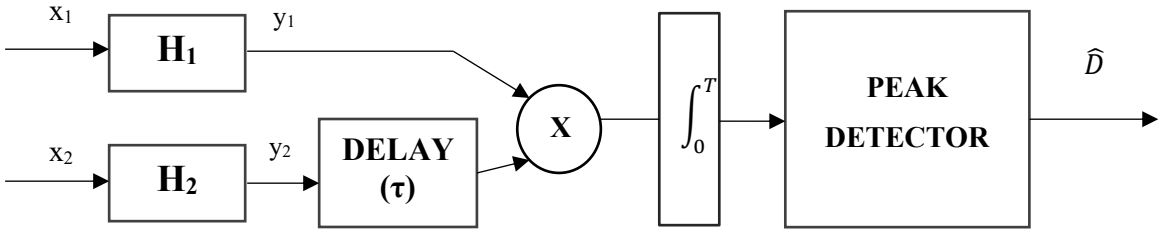

***Figura 3**. Diagrama de flujos de la estimación del retardo a partir de la GCC*

[1] Proceso ergódico: proceso en el cual los parámetros estadísticos calculados en un conjunto de realizaciones (promedios de conjunto) son iguales que los parámetros estadísticos calculados en una única realización (promedios temporales). Concepto asociado a los procesos estacionarios referidos en el documento.

Las señales y_1 e y_2 son multiplicadas e integradas para un rango de retardos temporales τ (variable representada por la caja "DELAY" en el diagrama). El retardo τ para el cual se obtenga el pico en la función de correlación es una estimación del retardo, $\widehat{D}$.

Cuando $H_1(f)=H_2(f)=1\ \forall f$, $\widehat{D}$ es simplemente el valor de la abscisa para el cual la función de correlación cruzada es máxima. Eligiendo apropiadamente los filtros $H_1(f)$ y $H_2(f)$ se facilita la estimación del retardo.

La correlación cruzada está relacionada con la función de densidad espectral de potencia cruzada a través de la transformada de Fourier:

$$R_{x1x2}(\tau) = \int_{-\infty}^{\infty} G_{x1x2}(f)\, e^{j2\pi f\tau} df \tag{8}$$

Cuando $x_1(t)$ y $x_2(t)$ han sido filtrados la potencia del espectro cruzada es:

$$G_{y1y2}(f) = H_1(f)\overline{H_2}(f)\, G_{x1x2}(f) \tag{9}$$

donde H_2 esta conjugada. Así llegamos a la expresión de la función de correlación generalizada (GCC):

$$R_{y1y2}^{(g)}(\tau) = \int_{-\infty}^{\infty} \psi_g(f)\ G_{x1x2}(f)\, e^{j2\pi f\tau}\, df \tag{10}$$

donde $\psi_g(f) = H_1(f)\ \overline{H_2}(f)$ es la ponderación frecuencial.

Como se ha dicho antes, el tiempo de observación es finito por lo que una estimación $\hat{G}_{x1x2}(f)$ de $G_{x1x2}(f)$ puede obtenerse a partir de observaciones finitas de $x_1(t)$ y $x_2(t)$. Teniendo esto en cuenta la ecuación (10) quedaría de tal manera:

$$\hat{R}_{y1y2}^{(g)}(\tau) = \int_{-\infty}^{\infty} \psi_g(f)\ \hat{G}_{x1x2}(f)\, e^{j2\pi f\tau}\, df \tag{11}$$

Existen varios tipos de ponderación frecuencial pero los más importantes y los que se van a implementar en la solución son los definidos a continuación:

· The Roth processor[10]: $\psi_R(f) = \frac{1}{G_{x1x1}(f)}$ (12)

· The Smoothed Coherence Transform (SCOT)[11]: $\psi_S(f) = \frac{1}{\sqrt{G_{x1x1}(f)G_{x2x2}(f)}}$ (13)

· The Phase Transform (PHAT)[12]: $\psi_P(f) = \frac{1}{|G_{x1x2}(f)|}$ (14)

La función de las ponderaciones frecuenciales es facilitar la identificación del pico máximo en la función de correlación agudizando el pico de valor máximo y atenuando el resto de los picos de la función. No obstante, estos picos son más sensibles a errores introducidos por observaciones finitas como es el caso, especialmente en entornos de bajo SNR (*Signal To Noise Ratio* o relación señal/ruido).

2.5. Modelo de estimación de la posición de un emisor con más de dos micrófonos

En el modelo más sencillo posible, donde se utilizan dos micrófonos para localizar un emisor en un plano a partir de un retardo, se obtienen una serie infinita de puntos, todos susceptibles de ser la posición de la fuente acústica, que siguen el patrón de una hipérbola (figura 4).

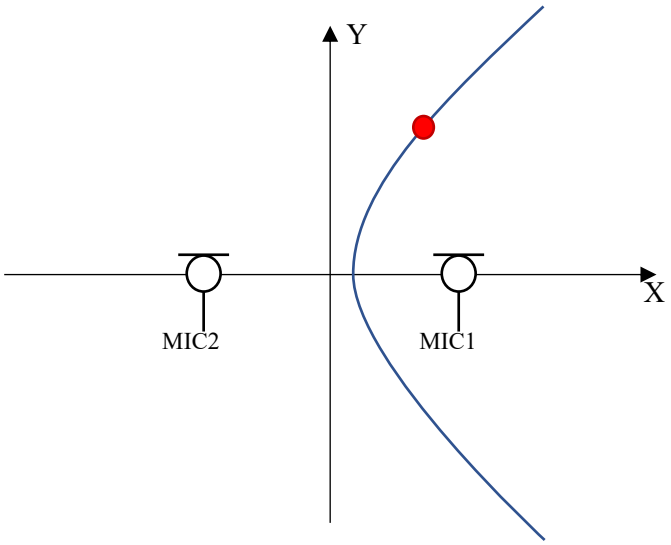

***Figura 4**. Esquema rama hiperbólica calculada a partir del retardo.*

Una línea como resultado dista de ser una posición estimada, pero al fin y al cabo el problema planteado consiste en resolver una ecuación con dos incógnitas (coordenada ***x*** y coordenada ***y***). Si lo que se quiere es obtener un punto como solución debe añadirse al sistema otra ecuación, es decir, otro micrófono al modelo. De esta forma se obtiene otra hipérbola, cuyo hipotético punto de intersección con la otra hipérbola es la posición estimada de la fuente acústica.

Si se considera un caso ideal, en el que el error de estimación de los retardos para cada par de micrófonos es cero, las hipérbolas obtenidas a partir de estos retardos intersecan en un único punto, proporcionando así una solución única y exacta al problema de localización.

Pero cuando en el modelo hay más de dos micrófonos, el sistema de ecuaciones pasa a estar sobredeterminado, y como generalmente ocurre, estos sistemas de ecuaciones son no compatibles, es decir, carecen de solución analítica. Aplicado al modelo, las hipérbolas obtenidas a partir de los retardos entre cada par de micrófonos intersecan unas con otras en distintos puntos del plano (figura 5), en vez de en un solo punto como sería deseable.

Llegados a este punto debe hacerse uso de alguna técnica de optimización matemática para obtener la mejor aproximación a una solución y por ende encontrar una solución numérica que sea considerada como la posición estimada de la fuente acústica.

El problema planteado en el modelo de más de dos micrófonos es el de un sistema sobredeterminado de ecuaciones no lineales. Este tipo de problema se corresponde con un problema de mínimos cuadrados no lineales. Para la resolución de este problema se opta por el algoritmo de Gauss-Newton.

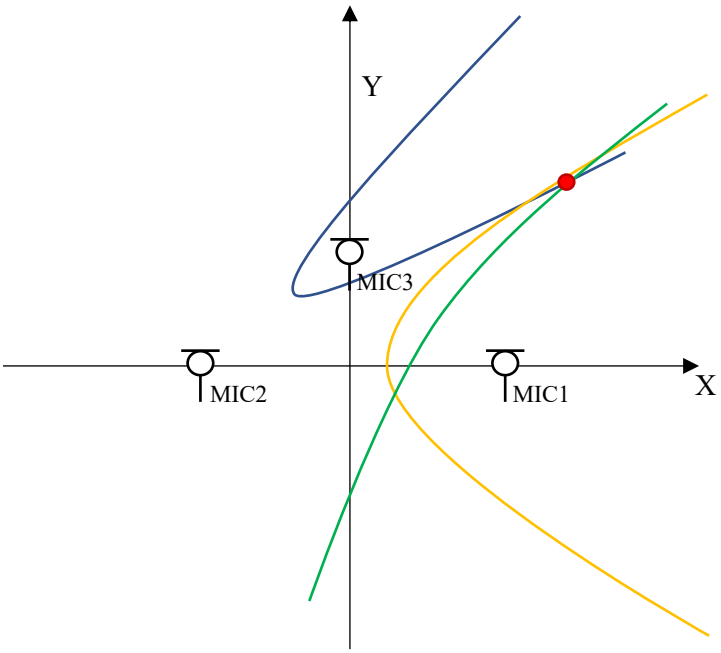

***Figura 5**. Representación de tres hipérbolas obtenidas a partir de los retardos entre micrófonos. Azul: retardo entre MIC2 y MIC3; Naranja: retardo entre MIC1 y MIC2; Verde: retardo entre MIC1 y MIC3.*

2.6. Método de obtención de la posición de una fuente acústica

El problema a resolver una vez se han obtenido las hipérbolas es un sistema de K ecuaciones no lineales sobredeterminado. Para ello se elige el algoritmo de Gauss-Newton como método de optimización para obtener la mejor aproximación.

El método iterativo de Gauss-Newton para mínimos cuadrados no lineales es el algoritmo estándar para resolver sistemas de ecuaciones no lineales como el que se pretende resolver.

Partiendo de una estimación inicial, representado por el vector $x^{(0)}$, en cada iteración k se calcula un factor de ajuste δ^k que se suma a la estimación de la anterior iteración, de la manera indicada por la ecuación (15):

$$x^{k+1} = x^k + \delta^k \tag{15}$$

Normalmente, cuando el algoritmo está convergiendo, el factor de ajuste se hace más pequeño según se acerca al resultado óptimo para así realizar el ajuste con precisión. El bucle seguirá en funcionamiento hasta que se cumpla un criterio de convergencia $\in$, de tal manera que cuando $\delta^k < \in$, se considera que se ha llegado a la convergencia del algoritmo.

El factor de ajuste δ^k se obtiene despejando la ecuación (16), cuyo despeje queda como la ecuación (17), que es la ecuación que debe resolverse en cada iteración:

$$J_f(x^k)^T J_f(x^k)\delta^k = -J_f(x^k)^T f(x^k) \tag{16}$$

$$\delta^k = -\left(J_f(x^k)^T J_f(x^k)\right)^{-1} \cdot J_f(x^k)^T f(x^k) \tag{17}$$

Donde f es la función de coste que se pretende minimizar y J es la matriz jacobiana de la función de coste f. La función de coste para el problema de localización (18) se obtiene restando el vector diferencia de distancias $\widehat{\Delta d}$ calculadas a partir de los retardos estimados con la GCC menos el vector diferencia de distancias $\Delta d(x^k)$:

$$f(x^k) = \widehat{\Delta d} - \Delta d(x^k) \tag{18}$$

Replanteando el problema de forma general, se tiene un emisor, representado por el punto S = (x_s,y_s). Se tienen n micrófonos, situados en las posiciones $M_n=(x_n,y_n)$. Por lo tanto, la distancia del micrófono n al punto P es:

$$d_n = \sqrt{(x_n - x_s)^2 + (y_n - y_s)^2} \tag{19}$$

La diferencia de distancias para cada combinación de par de micrófonos se obtiene:

$$\Delta d_{ij} = d_i - d_j = \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2} - \sqrt{(x_j - x_s)^2 + (y_j - y_s)^2} \tag{20}$$

para $i,j \leq n$, $i \neq j$.

Llevando la ecuación (20) al contexto del algoritmo GN, encontramos que en cada iteración el vector diferencia de distancia $\Delta d(x^k)$ se calcula según indica la ecuación (21):

$$\Delta d_{ij}(x^k) = \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2} - \sqrt{(x_j - x_k)^2 + (y_j - y_k)^2} \tag{21}$$

Una vez definida la función de coste, para obtener la matriz jacobiana se deben calcular las derivadas parciales de la función de coste con respecto a x e y. Dichas derivadas parciales tienen la siguiente forma:

$$\frac{\partial f_{ij}}{\partial x} = \frac{x - x_j}{d_j} - \frac{x - x_i}{d_i} \tag{22}$$

$$\frac{\partial f_{ij}}{\partial y} = \frac{y - y_j}{d_j} - \frac{y - y_i}{d_i} \tag{23}$$

Así pues, la matriz jacobiana queda de la siguiente forma:

$$J_f(x) = \begin{pmatrix} \frac{\partial f_{12}}{\partial x} & \frac{\partial f_{12}}{\partial y} \\ \frac{\partial f_{13}}{\partial x} & \frac{\partial f_{13}}{\partial y} \\ \vdots & \vdots \\ \frac{\partial f_{(i-1)j}}{\partial x} & \frac{\partial f_{(i-1)j}}{\partial y} \end{pmatrix} = \begin{pmatrix} \frac{x-x_2}{d_2} - \frac{x-x_1}{d_1} & \frac{y-y_2}{d_2} - \frac{y-y_1}{d_1} \\ \frac{x-x_3}{d_3} - \frac{x-x_1}{d_1} & \frac{y-y_3}{d_3} - \frac{y-y_1}{d_1} \\ \vdots & \vdots \\ \frac{x-x_j}{d_j} - \frac{x-x_{(i-1)}}{d_{(i-1)}} & \frac{y-y_j}{d_j} - \frac{y-y_{(i-1)}}{d_{(i-1)}} \end{pmatrix} \quad (24)$$

El algoritmo de GN supone el último paso en el proceso de estimación de la posición de una fuente acústica. Una vez que el algoritmo converge, el vector x contiene las coordenadas (x,y) de la posición estimada de la fuente que es, en primera instancia, el objetivo de este proyecto.

Un problema que presenta el algoritmo GN es que requiere de una estimación inicial sobre la cual ir ajustando en cada iteración hasta llegar a la convergencia en el algoritmo y obtener así una solución. La importancia de esta estimación inicial es significativa, ya que una buena estimación inicial proporciona una rápida convergencia y una estimación precisa. En cambio, una mala estimación inicial puede provocar la divergencia del algoritmo impidiendo así obtener una solución [13].

Otro aspecto, a priori desventajoso, está relacionado con la naturaleza de no linealidad del problema. Esto conlleva a mayor complejidad computacional que otros métodos más sencillos aunque menos precisos [14]. Realmente esta objeción resulta trivial en vista de los potentes procesadores que pueden obtenerse hoy en día en el mercado, por lo que esto no debería suponer una preocupación.

3. Especificaciones

Lo que se pretende en este proyecto es modificar y añadir prestaciones a una aplicación ya existente de estimación de dirección de fuentes sonoras, de manera que la nueva aplicación sea más versátil y permita aumentar el grado de precisión de los resultados.

La aplicación de la que se parte, y que en adelante será referida como aplicación base, fue diseñada para un Proyecto Fin de Grado en 2018 y fue implementada con el objetivo de servir para futuras ampliaciones y mejoras. El proyecto fue desarrollado en lenguaje C++ en el entorno de desarrollo Visual Studio y el código está listo para ser reutilizado.

La aplicación base estima la dirección de fuentes sonoras en tiempo real a partir del procesado de dos señales provenientes de dos micrófonos colocados en el mismo plano que la fuente acústica. El resultado no es un punto en el plano, sino una infinidad de puntos que, unidos, forman una hipérbola en el plano. Los puntos que forman la hipérbola son las posibles direcciones de la fuente sonora.

Para la consecución de este proyecto se definen tres objetivos principales:

• Procesado de ficheros de audio: la aplicación base solo puede procesar audio multicanal proveniente de dispositivos de entrada, como interfaces de audio externas. Se pretende que una de las funcionalidades añadidas sea la posibilidad de seleccionar desde la aplicación ficheros de audio multicanal de un evento acústico ya grabado para su procesado y estimación de la posición.

• Implementación del algoritmo GCC [5]: la aplicación base estima la dirección a partir de la diferencia de tiempos de llegada (TDOA) obtenida a partir de una correlación cruzada. El algoritmo GCC puede proporcionar estimaciones más precisas de la TDOA que ayudan a una mejor localización de las fuentes sonoras.

• Aumento del grado de precisión de la estimación: a partir de la adición de un tercer micrófono se dispondría de la suficiente información como para estimar una posición concreta en el plano.

Para el desarrollo de este proyecto se imponen las siguientes condiciones que el software debe cumplir:

- La aplicación debe ser capaz de estimar dirección y posición en tiempo real, ya provenga el audio de un dispositivo de entrada o de un fichero.
- La aplicación debe ser compatible con Windows y no depender de librerías de audio externas.
- La interfaz gráfica de la aplicación permitirá al usuario fijar las características del proceso (fuente del audio a procesar, número de canales, disposición de los micrófonos, parámetros de cálculo, etc.).
- La aplicación debe adaptarse a cualquiera de las posibles configuraciones impuestas por el usuario sin ver afectada su funcionalidad ni rendimiento.
- Capacidad de guardar los audios capturados, así como los resultados del cálculo de la estimación.
- La aplicación debe mostrar gráficamente la dirección estimada en un plano, así como indicar el resultado numérico de la estimación de la posición.
- El código de la aplicación debe estar estructurado de tal forma que permita fácilmente futuras mejoras y ampliaciones, tal como lo hacía la aplicación base.

4. Desarrollo del software

El código fuente escrito (en lenguaje de programación C++) para esta aplicación ha sido desarrollado en el IDE (*Integrated Development Environment* o entorno de desarrollo integrado) Visual Studio 2019. Para la implementación de cada uno de los aspectos de la aplicación han sido necesarios el uso de las librerías de Windows específicas para audio WASAPI y MMDevice.

Como ya se ha mencionado, las interfaces de WASAPI se encargan de la gestión del flujo de audio. Existen dos modalidades de flujo: modo exclusivo y modo compartido. Estos dos modos están disponibles tanto para los procesos de captura como para los de reproducción. Si una aplicación establece un flujo en modo exclusivo, la aplicación tendrá el uso exclusivo de ese dispositivo, impidiendo así a otras aplicaciones acceder a él. En cambio, si una aplicación establece un flujo en modo compartido, otras aplicaciones podrán establecer flujos con el mismo dispositivo [15]. En la figura 6 se ilustra el diagrama de los *Windows Audio Core APIs* y sus relaciones con el resto de los elementos de Windows.

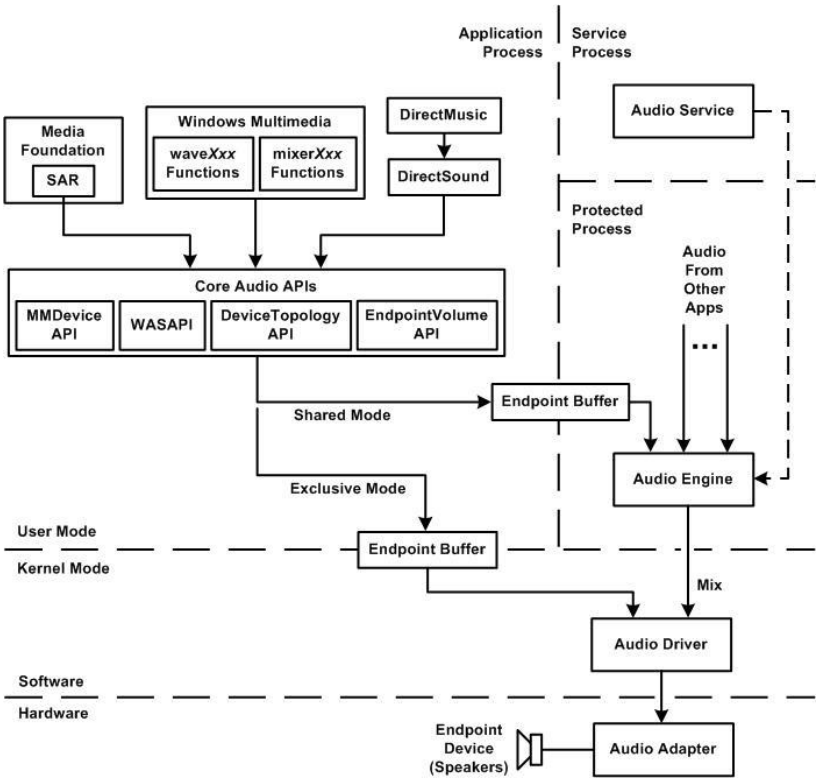

***Figura 6**. Diagrama de Windows Core Audio [16]*

Una ventaja que presenta el modo exclusivo frente al modo compartido es que las muestras no sufren ningún tipo de procesamiento como si ocurre en el modo compartido. Precisamente es en el módulo *Audio Engine* donde se produce la conversión del audio al formato 32 bits para que todas las aplicaciones que funcionen en modo compartido puedan acceder a esos datos, es decir, las muestras sufren modificaciones desde que el audio es capturado hasta que llegan a la aplicación. En cambio, en modo exclusivo las muestras de audio pasan directamente del buffer del dispositivo de captura a la aplicación. Por esta razón, en este software se implementa la captura en modo exclusivo, ya que interesa trabajar con las muestras obtenidas del dispositivo sin ningún tipo de modificación.

La librería MFC juega un importante papel en esta aplicación, participando de la implementación de los controles de las interfaces gráficas y demás aspectos funcionales dentro de la aplicación como la gestión de hilos de trabajo paralelos a la interfaz principal y la creación y manejo de distintos objetos como arrays, eventos y elementos de dibujo muy importantes para la clase **CRadarView**, encargada de la representación de resultados en la interfaz de usuario.

A parte de las librerías integradas en Windows, se ha utilizado la librería externa FFTW (*Fastest Fourier Transform in the West* o La Transformada de Fourier más rápida en Occidente) [17] para ejecutar las transformadas de Fourier del método de la GCC.

Los detalles del funcionamiento de la aplicación desarrollada pueden ser consultados en el manual de usuario del Anexo I. Para posibilitar una mejor comprensión de este apartado se puntualizan los aspectos más elementales del software.

La aplicación tiene capacidad para procesar muestras provenientes de un fichero de audio en formato WAV o de un dispositivo de captura conectado al ordenador, como un micrófono. A estos dos modos de trabajo se les denomina Modo Fichero y Modo Dispositivo, respectivamente. Ambos modos están implementados en hilos de trabajo independientes de la aplicación principal, de manera que durante la ejecución del proceso de localización hay funcionando siempre dos hilos simultáneamente. Cada modo de trabajo tiene unas clases dedicadas específicamente, y hacen también uso de clases y funciones comunes para procesar las muestras y representar los resultados en la interfaz de usuario. Las clases de este proyecto se dividen en 5 tipos:

- Clases dedicadas a las interfaces de usuario
- Clases dedicadas al Modo Fichero
- Clases dedicadas al Modo Dispositivo
- Clase dedicada al control visual
- Clases y funciones auxiliares

4.1. Estructura del código fuente

4.1.1. Clases dedicadas a las interfaces de usuario

Estas clases son las encargadas de proporcionar las interfaces visuales de la aplicación. Todos los cuadros de diálogo que forman esta aplicación han sido diseñados en el editor de cuadros de diálogo del entorno de desarrollo Visual Studio 2019. Integrando los cuadros de diálogo también se incluyen otros recursos visuales como iconos y menús, también desarrollados en el entorno de desarrollo de recursos. En estas clases es donde se produce el uso más intensivo de las clases MFC ya que todos los controles gráficos están implementados con clases MFC. A continuación, se describen cuáles son estas clases y cuáles son sus funciones.

a) Clase CSoundSourcev2App

CSoundSourcev2App es una clase derivada de la clase CWinApp MFC.

Esta clase implementa la aplicación propiamente dicha. La clase CWinApp está creada para servir de clase base sobre la que se derive un objeto de aplicación [18]. Su función es inicializar los procesos para poner en marcha la aplicación e iniciar la interfaz de usuario principal, implementada en la clase **CSoundSourcev2Dlg.**

b) Clase CSoundSourcev2Dlg

CSoundSourcev2Dlg es una clase derivada de la clase CDialog de MFC.

Su función principal es implementar la interfaz de usuario principal. El aspecto visual de la interfaz ha sido diseñado en el editor de diálogos de Visual Studio bajo el nombre de IDD_MAIN_DIALOG.

A parte de las funciones de control de la interfaz de usuario, la clase **CSoundSourcev2Dlg** ejerce otras funciones que la convierten en la clase más importante de toda la aplicación. Algunas de estas funciones son gestionar el envío de la información recogida en la interfaz a las clases de la aplicación y gestionar la creación y destrucción de los hilos de trabajo independientes que se encargan del procesamiento del audio. Otra de las características de esta clase es que actúa como nodo de todos los mensajes que las clases envían mediante el mapa de mensajes diseñado en el proyecto.

En la figura 7 se muestra el aspecto visual de esta interfaz y se identifican los controles de usuario con un número colocado encima de cada control:

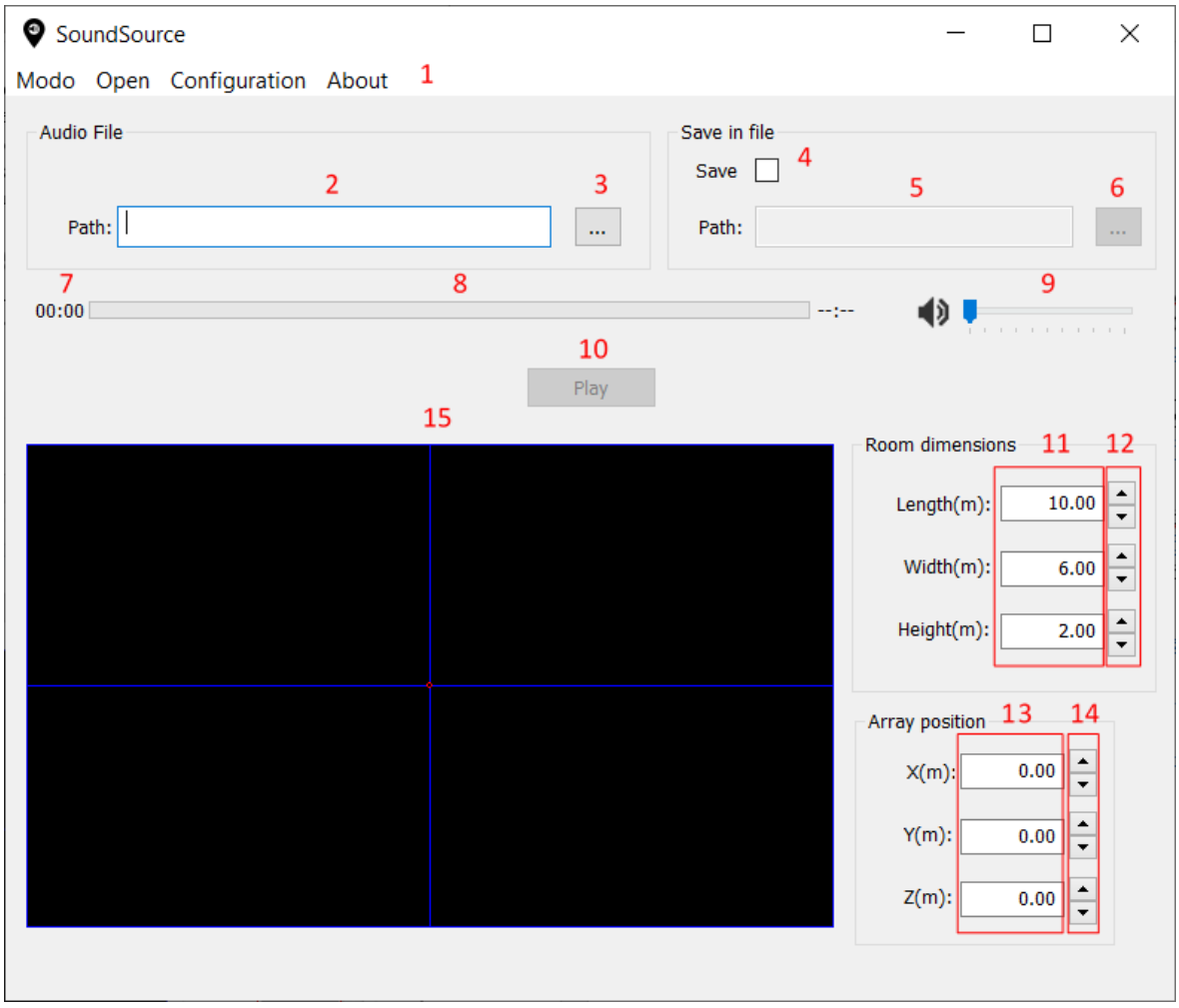

***Figura** 7. Interfaz de usuario principal de SoundSource 2.0*

1. Menú de la aplicación. Objeto de tipo CMenu de MFC. Dispone de cuatro elementos:

- Modo: se abre un submenú para elegir el modo de trabajo (Modo Fichero o Modo Dispositivo).

- Open: se abre un cuadro de diálogo para seleccionar un fichero para ser procesado. Solo habilitado en Modo Fichero.

- Configuration: se abre un submenú de dos elementos (Inputs y Delay Estimation) que abren los respectivos cuadros de diálogo de configuración.

- About: información sobre el software.

2. Cuadro de edición "Path" (Audio File). Objeto de tipo CEdit de MFC. Permite escribir la ruta de acceso del fichero de audio que va a ser analizado. Solo habilitado en Modo Fichero.

3. **Botón "Path" (Audio File)**. Objeto de tipo CButton de MFC. Al pulsarlo se abre un cuadro de diálogo para seleccionar la ruta de acceso del fichero a ser procesado. Tiene la misma función que el elemento Open del menú.

4. Casilla de verificación "Save". Objeto de tipo CButton de MFC. Indica si el procesamiento va a generar un fichero de resultados en el caso del Modo Fichero, y además un fichero del audio capturado en el caso del Modo Dispositivo. Al marcarlo se habilitan los controles 5 y 6.

5. Cuadro de edición "Path" (Save in file). Objeto de tipo CEdit de MFC. Permite escribir el directorio donde se quieren guardar los ficheros resultantes. Si este espacio queda en blanco se guardarán en el directorio del ejecutable.

6. **Botón "Path" (Save in file)**. Objeto de tipo CButton de MFC. Al pulsarlo se abre un cuadro de diálogo para seleccionar el directorio donde se quieren guardar los ficheros resultantes.

7. Indicador numérico de tiempo. Objeto de tipo CStatic de MFC. Indica el tiempo que lleva el fichero de audio reproduciéndose. Solo habilitado en el Modo Fichero.

8. **Barra de progreso**. Objeto de tipo CProgressCtrl de MFC. Indica visualmente el progreso del análisis del fichero. Solo habilitado en el Modo Fichero.

9. **Control deslizante de volumen**. Objeto de tipo CSliderCtrl de MFC. Controla el volumen de salida del fichero de audio que se está analizando. Solo habilitado en el Modo Fichero.

10. **Botón "Play/Pause"**. Objeto de tipo CButton de MFC. Permite poner en marcha el proceso de localización y pausarlo/reanudarlo tantas veces se desee.

11. **Cuadros de edición "Room dimensions"**. Objeto de tipo CEdit de MFC. Permite introducir las dimensiones en metros de la sala. Por defecto estas dimensiones son de 10x6x2.

12. **Control spin "Room dimensions"**. Objeto de tipo CSpinButtonCtrl de MFC. Permite la modificación del primer decimal de la dimensión a través del ratón.

13. **Cuadros de edición "Array position"**. Objeto de tipo CEdit de MFC. Permite introducir las coordenadas de la posición del array, considerando el origen de coordenadas el centro de la sala. Los valores están limitados por el tamaño de sala y por la colocación de los micrófonos en el array de manera que no se puedan salir del plano de la sala.

14. **Control spin "Array Position"**. Objeto de tipo CSpinButtonCtrl de MFC. Permite la modificación del primero decimal de la coordenada a través del ratón.

15. Control visual de la sala. Objeto de tipo **CRadarView**. Permite visualizar el plano de la sala, la colocación de los micrófonos en esta y las hipérbolas. Además, se incluye una cadena de texto en la parte inferior que indica el resultado de la estimación de posición. El tamaño de este control se ve modificado por los controles 11 y 12, siempre conservando la relación de aspecto.

c) Clase CChannelsConfigurationDlg

CChannelsConfigurationDlg es una clase derivada de la clase CDialog de MFC.

La función de esta clase es implementar la interfaz de la ventana de configuración de las entradas y enviar la información introducida por el usuario a la interfaz principal (clase **CSoundSourcev2Dlg**) mediante el mapa de mensajes de MFC. Ha sido diseñado en el editor de diálogos del entorno de desarrollo de recursos bajo el nombre de IDD_CONF_DIALOG. Esta ventana se abre a través del menú de la aplicación en Configuration → Inputs.

En la figura 8 se muestra el aspecto visual de este cuadro de diálogo cuando se abre por primera vez en la ejecución de la aplicación. Además, se identifican numéricamente sus controles:

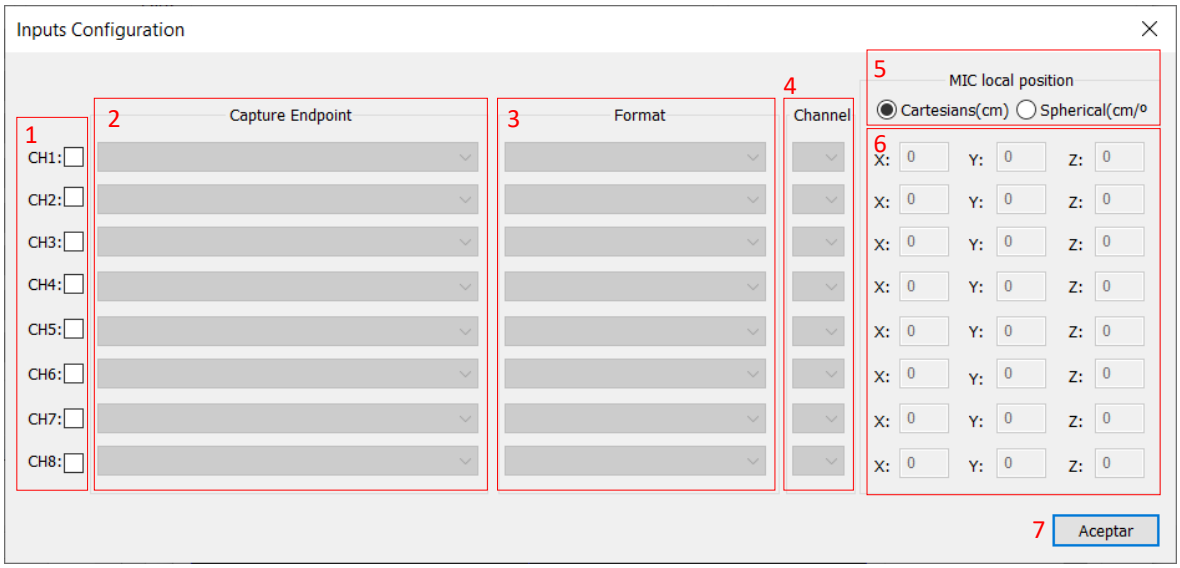

***Figura 8**. Interfaz de usuario de la ventana de configuración de entradas.*

1. Casillas de verificación "CH". Objeto de tipo CButton de MFC. Estas casillas indican cuantos canales van a ser utilizados en el proceso de localización. En Modo Fichero deben marcarse tantas casillas como canales tenga el archivo de audio multicanal. Una vez activado un canal se habilitan los cuadros de edición de las coordenadas (6) para introducir los datos de cada canal. En Modo Dispositivo deben marcarse tantas casillas como micrófonos vayan a utilizarse en el proceso de captura. Una vez marcado se habilitan los controles Capture Endpoint (2), Format (3), Channel (4) y MIC local position (6). A estas entradas CH- se las denomina canales de entrada virtuales.

2. Listas desplegables "Capture Endpoint". Objeto de tipo CComboBox de MFC. En cada una de estas listas desplegables se selecciona el dispositivo de captura del cual se van a tomar las muestras de audio para ese canal. Si, por ejemplo, en un proceso de captura de 4 canales

estos 4 canales provienen del mismo dispositivo, habría que seleccionar el mismo dispositivo para CH1-CH4. Este control solo se habilita en Modo Dispositivo.

3. **Listas desplegables "Format"**. Objeto de tipo CComboBox de MFC. En cada una de estas listas desplegables aparecen los formatos compatibles del dispositivo de captura seleccionado en la lista desplegable "Capture Endpoint" de ese canal. Este control solo se habilita en Modo Dispositivo.

4. **Listas desplegables "Channel"**. Objeto de tipo CComboBox de MFC. En estas listas desplegables se selecciona de qué entrada física del dispositivo se va a tomar la señal. Por ejemplo, en el caso del dispositivo de captura de cuatro canales, la configuración de este campo sería asignar cada uno de los 4 canales de entrada físicos del dispositivo a las 4 entradas virtuales CH1-CH4. El número de canales disponibles en esta lista depende del formato de captura seleccionado en la lista desplegable "Format". Este control solo se habilita en Modo Dispositivo.

5. **Botón de opción sistema de coordenadas**. Variable de tipo BOOL. En este botón se elige el sistema de coordenadas en el cual se van a introducir las posiciones de los micrófonos. Los datos pueden ser introducidos en coordenadas cartesianas (cm) o esféricas (grados y cm).

6. **Cuadros de edición "MIC local position"**. Objeto de tipo CEdit de MFC. En estos cuadros de edición se deben introducir las coordenadas de las posiciones de los micrófonos a la hora de realizar el proceso de captura. Las coordenadas se deben introducir con respecto a un centro del hipotético array, no con respecto a la sala. Este array luego puede moverse mediante los controles de "Array Position" de la interfaz principal. Estos cuadros de edición están implementados de tal forma que no puedan introducirse posiciones que sobrepasen los límites de la sala, ya sea debido a que las posiciones son demasiadas grandes para las dimensiones de la sala o debido a la colocación del array en la sala. En el caso de las coordenadas esféricas también se limita el rango de ángulos para que sean válidos. Para la elevación este rango es de 0º a 180º y para el azimut de 0º a 360º, siendo el frente la posición (90º, 0º).

7. **Botón "Aceptar"**. Objeto de tipo CButton de MFC. Al pulsar este botón se envían los datos de la configuración a la interfaz principal tras realizar la comprobación de que todos los parámetros introducidos son válidos. Las comprobaciones que se realizan son las siguientes:

   - Se comprueba que haya un mínimo de dos canales seleccionados para poder realizar el procesamiento.

   - Se comprueba que todos los campos de las entradas virtuales habilitadas no estén sin seleccionar.

   - Se comprueba que los formatos de captura seleccionados no difieren entre sí. La elección de distintos formatos en los canales de entrada virtuales tendría como principal efecto la invalidación del cálculo de la correlación cruzada ya que el tener dos señales con distinta frecuencia de muestreo implica que para una misma duración el número de muestras es distinto, lo que hace ineficaz una medida de similitud entra ambas señales sin antes haber remuestreado una de las señales, lo que inevitablemente conlleva una modificación de la señal original.

d) Clase CDelayEstimationConfDlg

CDelayEstimationConfDlg es una clase derivada de la clase CDialog de MFC.

Esta clase implementa la interfaz de usuario de la ventana de configuración de la estimación del retardo entre micrófonos. Al finalizar su ejecución envía los datos introducidos por el usuario a la interfaz principal mediante el mapa de mensajes de MFC. Su aspecto visual ha sido diseñado en el

editor de diálogos del entorno de desarrollo de recursos de Visual Studio bajo el nombre de IDD_TIMEDELAY_DIALOG. Esta ventana se abre a través del menú de la aplicación en Configuration → Delay Estimation.

En la figura 9 se muestra el aspecto de esta ventana y se identifica cada control de usuario numéricamente:

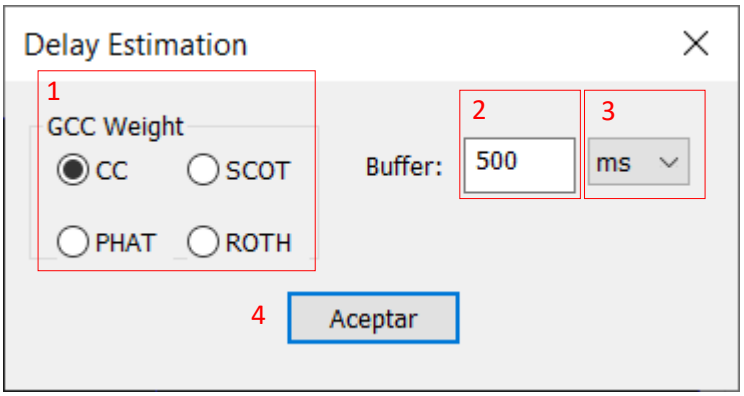

***Figura 9**. Interfaz de usuario de la ventana de configuración de estimación del retardo.*

1. **Botón de opción "GCC Weight"**. Variable de tipo int. En este grupo de botones se selecciona la ponderación frecuencial de la GCC (véase apartado 4.2).
2. **Cuadro de edición "Buffer"**. Objeto de tipo CEdit de MFC. En este espacio se introduce la longitud del buffer que se va a utilizar, que a la postre será el fragmento de audio que se analice en cada iteración del proceso de localización. Por defecto su valor es 500 ms.
3. **Lista desplegable "Buffer"**. Objeto de tipo CComboBox de MFC. Se selecciona las unidades del buffer (milisegundos o segundos). Por defecto está en ms.
4. **Botón "Aceptar"**. Objeto de tipo CButton de MFC. Cierra el cuadro de diálogo y envía la información recogida en la ventana a la interfaz principal.

4.1.2. Clases dedicadas al Modo Fichero

En estas clases se implementan todas las funciones necesarias para procesar un fichero de audio multicanal.

a) Clase CRenderFile

CRenderFile es una clase derivada de la clase CObject de MFC.

Su cometido es cargar, almacenar, caracterizar y separar por canales el fichero de audio multicanal seleccionado por el usuario.

Al inicializar un objeto de esta clase, se carga el fichero de audio en modo binario desde la ruta de acceso que se le pasa como parámetro. Seguidamente en la misma función de inicialización, denominada ***Initialize()***, se extraen los datos de la cabecera. Los datos binarios que contienen las muestras de audio codificadas (sin cabecera) se almacenan en un array de bytes. Por último, se procede a separar el audio por canales y almacenarlos en un array bidimensional de bytes, en el que

cada fila se corresponde con un canal distinto. En resumen, inicializando el objeto con un solo método se cargan todos los datos y la información necesaria para llevar a cabo el procesamiento pertinente.

El fichero de audio WAV está estructurado en dos partes. Por un lado, en los primeros bytes del archivo se encuentra la cabecera, de longitud variable, la cual contiene todos los atributos de la forma de onda de audio contenida en el fichero (freq. de muestreo, bits/muestra, tamaño en bytes, etc.). Por otro lado, inmediatamente después se encuentran las propias muestras de audio codificadas.

Para extraer la información de la cabecera es necesario conocer cuál es la estructura de una cabecera WAV. En la figura 10 se muestra la forma canónica de una cabecera WAV. Esta estructura puede variar tanto en contenido como en longitud. Una descripción detallada de las posibilidades de una cabecera WAV puede consultarse en [19].

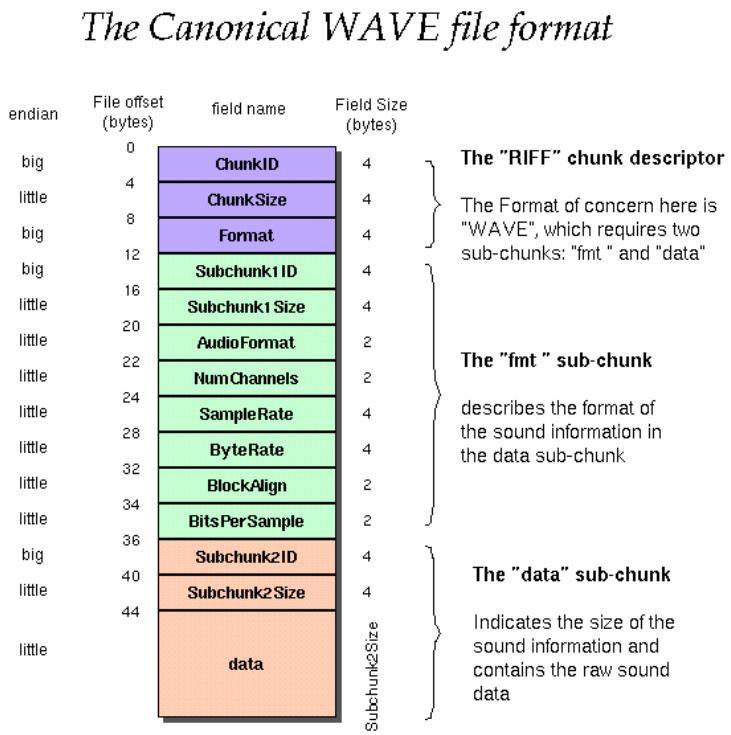

***Figura 10**. Cabecera WAV [20].*

Con los datos extraídos de la cabecera se crean dos estructuras de tipo WAVEFORMATEXTENSIBLE [21], una con la información del fichero de audio al completo y otro con la información de un solo canal de dicho fichero. Este tipo de estructura de la API de *Windows Multimedia* está destinada a especificar los detalles del formato de una forma de onda de audio. En la figura 11 se indica el contenido de esta estructura, así como de la estructura WAVEFORMATEX contenida en su interior. Todos los datos contenidos en la estructura pueden ser extraídos de la cabecera WAV.

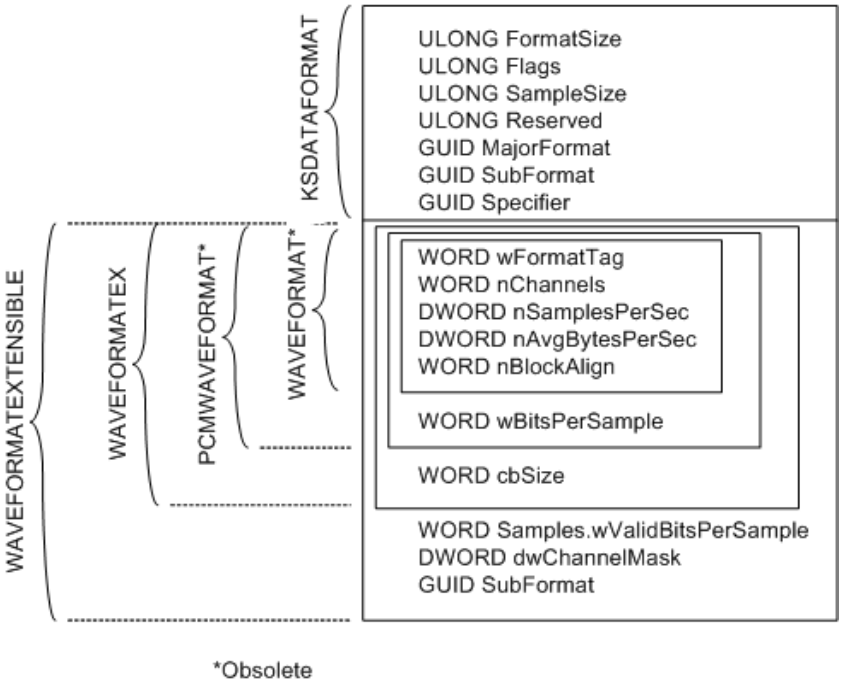

***Figura 11**. Estructura WAVEFORMATEXTENSIBLE [22].*

Para extraer los datos binarios que contienen las muestras de audio codificadas es necesario conocer en qué posición termina la cabecera y comienzan las muestras de audio. En la figura 10 se representa una cabecera WAV para muestras de audio en PCM. En este caso la longitud de la cabecera es de 44 bytes, por lo tanto, las muestras de audio se localizan a partir del byte 44.

La longitud de la cabecera varía en función del formato de audio. Generalmente se pueden dar tres casos:

- **Formato PCM**. La longitud de la cabecera es 44 bytes y el valor del parámetro ***Subchunk1 Size*** es de 16.
- **Formato no PCM**. El formato de audio en estos casos es 32 bit *floating* La longitud de la cabecera es de 46 bytes y el valor del parámetro ***SubChunk1 Size*** es de 18.
- **Formato extensible**. Este formato está destinado a formatos de audio de más de dos canales y/o profundidad de bits superior a 16[2]. La longitud de la cabecera es de 68 bytes y el valor del parámetro ***SubChunk1 Size*** es de 40.

Precisamente es el atributo ***SubChunk1 Size*** el que permite identificar el comienzo de los datos de audio. Este campo representa el tamaño de la sub-sección de formato (fmt), señalado en verde en la figura 10. Es el tamaño de esta sub-sección lo que varía de un formato a otro. Los otros 28 bytes que forman la cabecera canónica siempre están presentes en los tres casos señalados. Por lo tanto, la longitud de la cabecera se obtiene sumando 28 más el valor de **SubChunk1 Size.**

Una vez que se almacenan las muestras en un array de bytes (variable llamada *m_pbAudioData*), se procede a separar el audio por canales y almacenarlos en un array de bytes bidimensional denominado *m_ppbSeparatedChannels.*

En un fichero WAV las muestras de audio de cada canal se intercalan dando lugar a una ristra de bytes que contiene todas las muestras de todos los canales en orden. En la figura 12 se representa

[2] Aunque el formato extensible está destinado para los supuestos citados, es común encontrar ficheros WAV multicanal o con profundidad de bits de 24 con la denominación de formato PCM, lo cual por otra parte no es incorrecto.

el proceso de desentrelazamiento de esta ristra para obtener la separación de canales. Este proceso se implementa en la función ***SplitChannels()***.

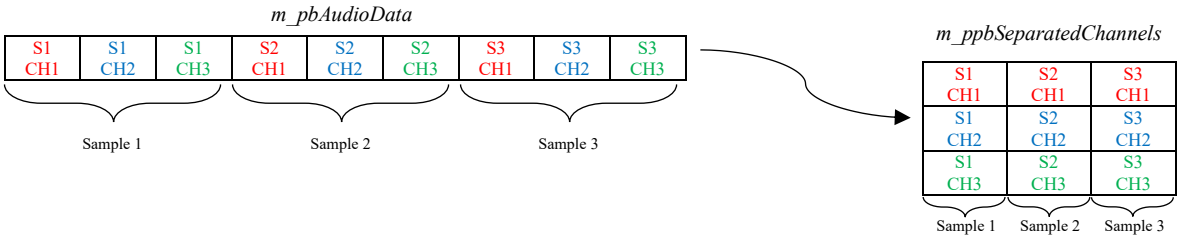

Figura 12. *Esquema proceso de SplitChannels().*

Para este proceso el parámetro más importante a tener en cuenta es la profundidad de bits, ya que ello determina de cuantos bytes está formada cada muestra y resulta crucial para hacer la separación por canales correctamente. El algoritmo está implementado para resoluciones de 8, 16, 24 y 32 bits. Para los casos de 8, 16 y 32 bits esta implementación es sencilla debido a que existen tipos de datos de 1, 2 y 4 bytes (*signed char*, *signed short* y *signed int* respectivamente) pero para el caso de muestras de 24 bits no existe un tipo definido de 3 bytes. Para suplir esta falta se ha implementado el tipo INT24, el cual se describe en el apartado de **Clases Auxiliares**. También se proporciona una solución en el código a modo de comentario sin usar la clase INT24.

b) Clase CRenderFileInterface

CRenderFileInterface es una clase derivada de la clase CWinThread de MFC.

En esta clase se implementa el hilo de trabajo independiente que procesa el audio para obtener los retardos en el modo de trabajo **Modo Fichero.**

Esencialmente lo que hace esta clase es acceder al dispositivo de salida predeterminado a través de las funciones de la librería MMDevice y gestionar el proceso de reproducción de un audio mediante las funciones de WASAPI a través de dicho dispositivo. Asociado al proceso de reproducción se realiza de manera paralela y síncrona el análisis de audio para obtener los retardos mediante la GCC. Visto desde otro punto de vista, se aprovechan las herramientas proporcionadas por WASAPI a través de sus funciones, tales como el acceso al buffer de salida y el control sobre este, para realizar el procesamiento.

A continuación, se detalla la secuencia de procesos que se producen cuando este hilo de trabajo se pone en marcha.

En primer lugar, se inicializa un objeto de tipo **CRenderFile** a partir de la ruta de acceso seleccionada por el usuario. Después, se obtiene una referencia del dispositivo predeterminado de salida a través de las interfaces MMDevice y se fija la longitud del buffer de salida en función de la duración de buffer elegida por el usuario. A través de este dispositivo de salida se va a efectuar la reproducción de audio. El tamaño del buffer determina la longitud temporal del fragmento que va a ser analizado en cada iteración.

El audio que se va a reproducir no es el fichero de audio en su totalidad sino el primer canal únicamente. Por este motivo en **CRenderFile** se ha creado una estructura WAVEFORMATEXTENSIBLE de los datos de un solo canal, el cual sirve de parámetro para configurar el flujo de datos. Esto es debido a que los dispositivos de salida comunes en los ordenadores son estéreo y por lo tanto no soportan audio multicanal. En cambio, un audio de un solo

canal (mono) siempre es compatible con cualquier dispositivo de salida por lo que esta simplificación evita dicho problema de compatibilidad.

A la vez que se reproduce el archivo de audio mono, se procesan todos los canales de audio del fichero para obtener los retardos. La sincronización de este proceso es posible gracias a que la unidad de datos que se maneja en las funciones de gestión del buffer es el *frame*. En este contexto, un *frame* de audio (figura 13) lo forman el conjunto de muestras (una muestra por cada canal) que se reproducen al mismo tiempo y su duración es la del periodo de muestreo. En consecuencia, para una duración de audio determinada, el número de *frames* en un canal es el mismo número de *frames* que en todos los canales.

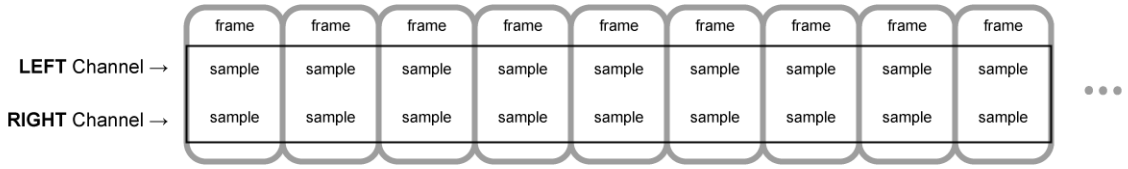

***Figura 13**. Frame de audio.*

Una vez que ya está todo preparado para la reproducción de audio y se ha configurado todo lo necesario para su análisis, incluida la longitud del buffer y las características de la GCC, comienza a ejecutarse el bucle de reproducción y análisis, cuyo diagrama de flujos se detalla en la figura 14.

En cada iteración de este bucle se escribe en el buffer de salida del dispositivo predeterminado un fragmento de audio del canal 1, que en la práctica se localiza en la primera fila del array bidimensional del objeto **CRenderFile** donde se almacenan las muestras separadas por canales. Seguidamente, los datos de audio de ese mismo fragmento temporal son tomado de todos los canales y procesados por la función ***CalculateDelays()***, la cual devuelve los retardos estimados para cada par de combinación de canales (comparación de i,j canales donde $i \neq j$, para $i,j = 1, 2, \ldots, n$ canales). Estos resultados se envían a la interfaz principal, encarnada por la clase **CSoundSourcev2Dlg**, que a su vez se encarga de enviarlos a la clase **CRadarView**, la cual se ocupa de dibujar las correspondientes hipérbolas en el plano y de ejecutar el algoritmo de Gauss-Newton para obtener una posición estimada. El resultado de esta posición estimada es enviado de vuelta a la interfaz principal, e inmediatamente enviada a **CRenderFileInterface**. Todo este intercambio de datos se realiza mediante el mapa de mensajes establecido. El último paso del bucle es escribir en un documento de texto el resultado de la operación, junto con los datos de la captura.

La implementación de este proceso se ha hecho mediante programación dirigida por eventos. WASAPI permite esta opción al proporcionar un evento que es señalizado cada vez que el buffer está listo para ser escrito. El bucle queda en espera hasta que este evento, denominado **Evento Buffer** en el diagrama, es señalizado. Además, se han introducido otros tres eventos:

- **Evento Pausa** y **Evento Reanudar**: desde la interfaz principal puede pausarse y reanudarse el proceso tantas veces considere el usuario. En caso de pausar el proceso se señaliza el Evento Pausa, que detiene la ejecución del bucle hasta que el **Evento Reanudar** sea señalizado. El **Evento Reanudar** también se señaliza desde la interfaz principal.
- **Evento Fin**: este evento conduce a la finalización del hilo de trabajo. Puede ser señalizado por diferentes razones como interrumpir el análisis de un fichero de audio antes de su finalización para escoger otro fichero, en cuyo caso se destruye el hilo antiguo y se crea uno nuevo, o cuando se produce un error.

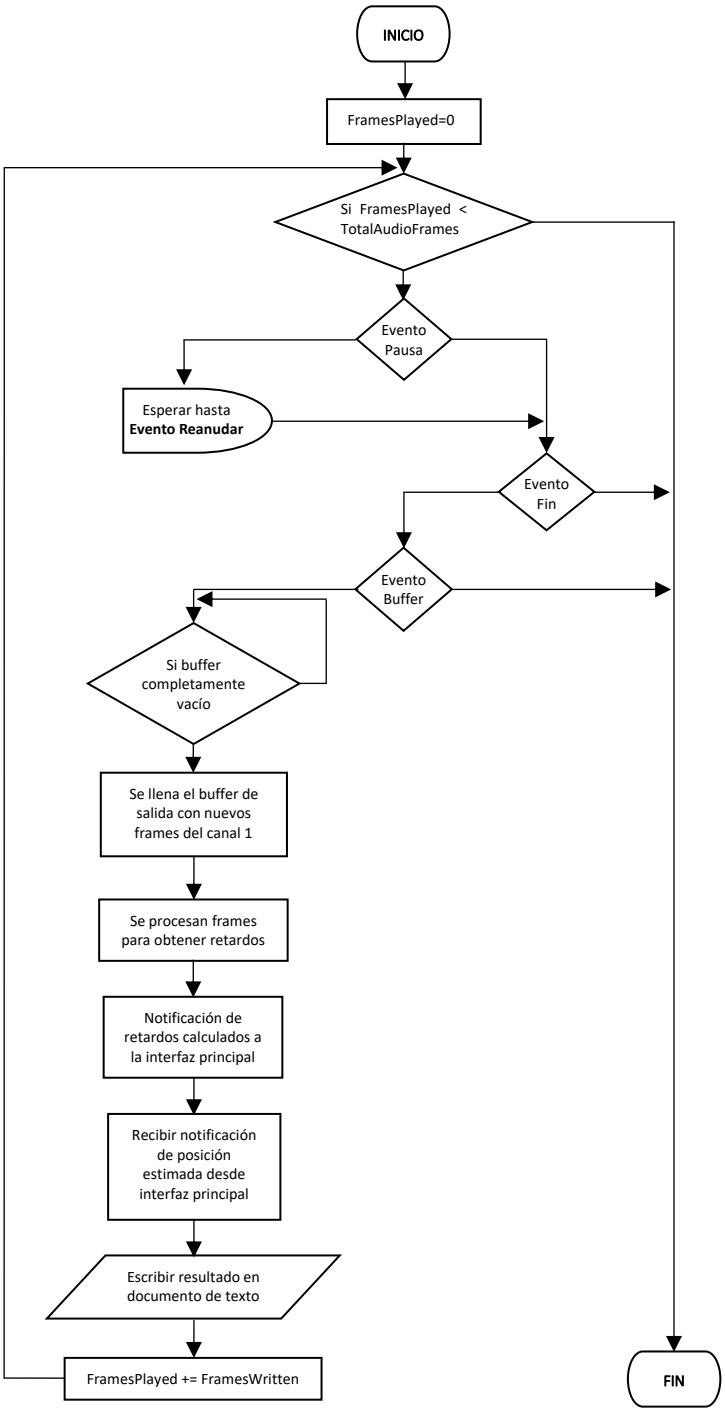

Figura 14. *Diagrama de flujos del bucle de ejecución en Modo Fichero.*

En el diagrama de la figura 14, la variable *FramesPlayed* indica el número de *frames* de audio que han sido analizados/reproducidos; la variable *TotalAudioFrames* indica el número total de *frames* de audio que contiene el fichero y la variable *FramesWritten* representa el número de *frames* que se escribe en el buffer en cada iteración, que en este caso coincide con el tamaño en *frames* del buffer de salida.

4.1.3. Clases dedicadas al Modo Dispositivo

La mayor parte de estas clases son reutilizadas del proyecto original. El Modo Dispositivo es, en realidad, una versión de la aplicación original escalada a más canales y a más dispositivos, por lo que se han realizado algunas modificaciones para adaptar el código a las necesidades del proyecto. En el apartado de cada clase se detalla cuáles son estas modificaciones o añadiduras.

a) Clase CAudioCaptureEndpoint

CAudioCaptureEndpoint es una clase derivada de la clase CObject de MFC.

Su función es describir un determinado dispositivo de captura al más bajo nivel a través de tres tipos de propiedades.

El primer tipo consiste en cadenas de texto que contienen parámetros tales como el ID del dispositivo, su nombre, su descripción y el nombre del adaptador.

El segundo tipo de propiedad son los formatos compatibles con el dispositivo. Estos formatos son guardados en forma de estructuras WAVEFORMATEX o WAVEFORMATEXTENSIBLE en un array llamado *m_arrWaveFormat.* A su vez los formatos se guardan también como una cadena de texto en un array de tipo *string*.

El tercer tipo de propiedad, que ha sido implementado para este proyecto, son dos arrays de enteros que contienen, en el caso de *m_arrDeviceInputChannels,* los canales de entrada físicos del dispositivo de los cuales se va a capturar audio y en el caso del array *m_arrChannelPosition,* la posición que ocupa cada canal de entrada físico en la configuración de las entradas virtuales. Ambos arrays se completan a partir de los datos introducidos por el usuario en la ventana **Inputs Cofiguration**. A continuación, se muestra un ejemplo que ilustra como se rellenan estos arrays. En la figura 15 se representa un ejemplo de configuración de entradas.

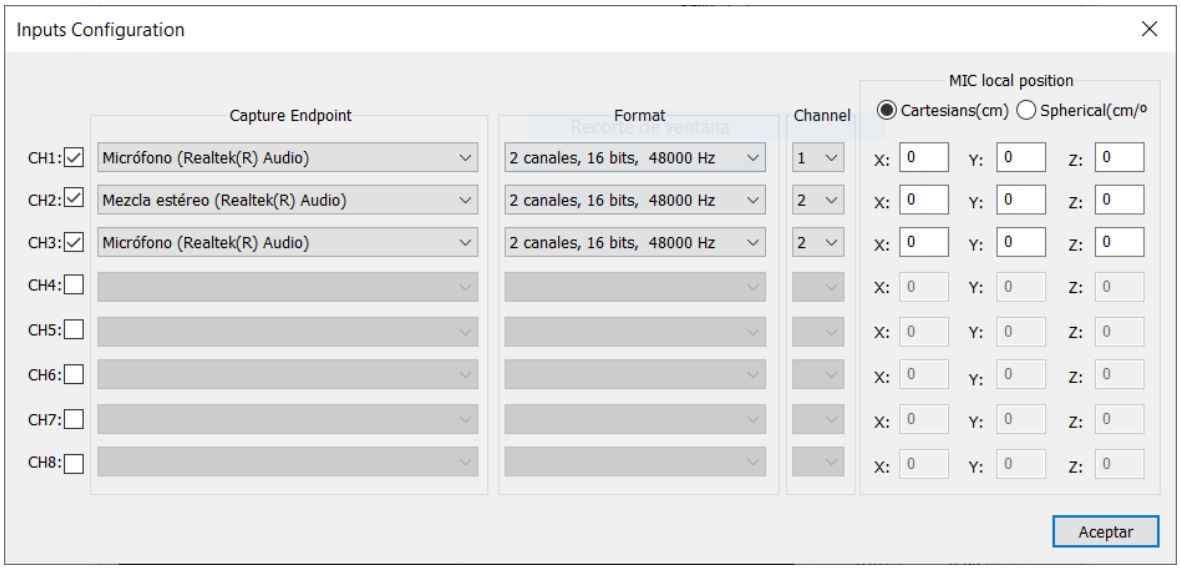

***Figura 15**. Ejemplo de configuración de entradas.*

Se tienen tres canales habilitados, es decir, tres entradas virtuales provenientes de dos dispositivos distintos. Por lo tanto, para este caso la aplicación trabajará con dos objetos de tipo **CAudioCaptureEndpoint**, uno para cada dispositivo. En el caso del dispositivo "Micrófono", el contenido de *m_arrDeviceInputChannels* será {1,2} ya que se están tomando datos de los canales 1 y 2. El contenido de *m_arrChannelPosition* será {1,3} ya que es la posición que ocupan los canales 1 y 2, respectivamente. En el caso del dispositivo "Mezcla estéreo", el contenido de *m_arrDeviceInputChannels* será {2} y el de *m_arrChannelPosition* {3}. Este sistema de índices puede parecer enrevesado, pero es de vital importancia en el desempeño de la clase **CWasapiThread**.

El principal propósito de esta clase dentro de la solución es servir de soporte para la clase **CAudioCaptureEndpointArray**.

b) Clase CAudioCaptureEndpointArray

CAudioCaptureEndpointArray es una clase derivada de la clase CObArray de MFC.

En esta clase se implementan todas las funciones necesarias para detectar los dispositivos de captura disponibles en el ordenador a través de las funciones de las interfaces MMDevice. Esta clase también es responsable de extraer las características de los dispositivos para crear objetos tipo **CAudioCaptureEndpoint** y guardar sus referencias en el array. Los métodos más relevantes de esta clase son:

- ***FillIdentifiers():*** crea la lista de dispositivos de captura activos tipo **CAudioCaptureEndpoint** y rellena la parte correspondiente a identificadores de texto (primer tipo de propiedad).
- ***FillCaptureCapabilities():*** detecta, a partir de la lista de dispositivos creada en ***FillIdentifiers()***, los formatos de captura compatibles con cada dispositivo. Actualmente no existe ninguna función en las interfaces del *Windows Audio Core* que detecte automáticamente los formatos soportados por cada dispositivo. El método utilizado consiste, por medio de bucles *for* anidados, en probar todas las combinaciones posibles de formatos de audio y confirmar su compatibilidad por medio de una función de una interfaz de WASAPI llamada ***IsFormatSupported()***, la cual indica por medio de un booleano si un formato es soportado por un dispositivo.

Para este proyecto se ha añadido otro bucle *for* anidado para variar la profundidad de bits, ya que en el proyecto original solo se contemplaba 16 bits, y para este proyecto se han añadido las profundidades de 24 y 32 bits.

c) Clase CAudioCaptureEndpointInterface

CAudioCaptureEndpointInterface es una clase derivada de la clase **CAudioCaptureEndpoint**.

Esta clase implementa los métodos necesarios para realizar la captura de audio a través de las interfaces de WASAPI y MMDevice. Algunas de sus funciones más relevantes son:

- ***PrepareCapture()***: esta función es llamada desde otra clase. En este método se obtiene el dispositivo de captura a través de las interfaces de MMDevice y se establece el flujo de datos entre el dispositivo de captura y la aplicación en modo exclusivo. Uno de los problemas contemplados al inicializar el flujo de audio es la posible discrepancia entre la duración del buffer elegido por el usuario y la duración real que se necesita. Esta

desavenencia es conocida como problema de alineamiento del buffer. Esto ocurre cuando la duración solicitada no se corresponde con un número entero de *frames* de audio. Mientras que en modo compartido esta duración se ajusta automáticamente redondeando la duración al siguiente tamaño de buffer en *frames*, en modo exclusivo y dirigido por eventos, como es el caso, debe ser el usuario el que gestione este ajuste del buffer. Microsoft proporciona en su documentación el procedimiento para realizar este ajuste [23], el cual se ha seguido en la implementación del código.

- ***StartCapture():*** inicia el proceso de captura.
- ***StopCapture():*** detiene el proceso de captura.
- ***GetBuffer():*** accede a las muestras de audio del buffer del dispositivo de captura
- ***GetSubBuffer():*** accede a las muestras de un determinado canal del buffer. Para separar el audio por canales se sigue un procedimiento parecido al de la función **CRenderFile::*SplitChannels()*** explicada anteriormente, solo que en esta ocasión se colocan todas las muestras en un mismo vector de bytes (figura 16). Con respecto al código original, este se ha adaptado para trabajar con muestras de 24 y 32 bits.

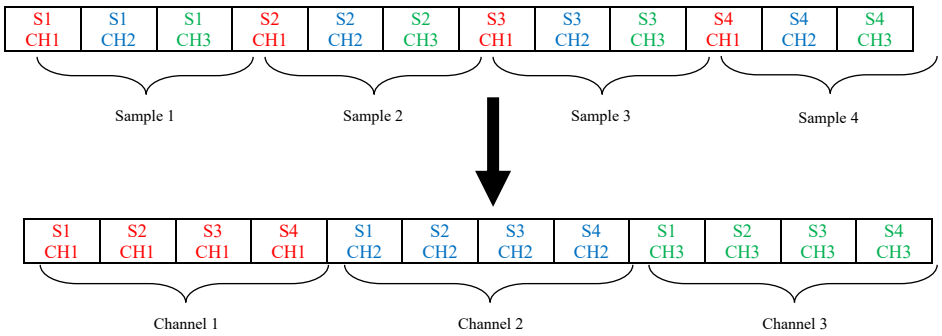

Figura 16*. Esquema proceso GetSubBuffer().*

d) Clase CWasapiThread

CWasapiThread es una clase derivada de la clase CWinThread de MFC.

En esta clase se implementa el hilo de trabajo independiente que procesa el audio para obtener los retardos en el modo de trabajo **Modo Dispositivo.**

Al crear el hilo lo primero que sucede es la inicialización de los flujos de audio con los dispositivos de captura que se van a utilizar en el proceso. Cada dispositivo está representado por un objeto de tipo **CAudioCaptureEndpointInterface.**

Una vez preparada la captura, el hilo empieza a correr y a ejecutar el bucle de procesamiento, representado en la figura 17.

El proceso de captura está dirigido por eventos, de tal manera que hasta que no se señalizan los eventos de buffer de todos los dispositivos involucrados que indican que el buffer está listo para ser escrito no se procede a ejecutar el procesamiento. No obstante, el tiempo de espera no es ilimitado; si se supera un límite de tiempo la aplicación arroja un error y el proceso de captura finaliza. Dicho limite es proporcional a la duración temporal del buffer. Esta técnica de sincronización entre dispositivos está extraída del anexo del proyecto original referido a la multicaptura.

Una vez que los buffers se han llenado con las muestras de audio capturadas, se procede a copiarlas a un array de bytes bidimensional en el que cada fila corresponde a un canal. El orden de los canales está determinado por la configuración de los canales en la ventana **Inputs Configuration**. Es aquí donde entra en juego el sistema de indexación formado por los arrays *m_arrDeviceInputChannels* y *m_arrChannelPosition* de la clase **CAudioCaptureEndpoint.** De esta manera se asegura que a partir de cualquier configuración de canales de entrada virtuales seleccionado en la ventana **Inputs Configuration,** incluso seleccionando canales no consecutivos, se puedan procesar los canales en el orden correcto.

Después se escribe el audio capturado en un fichero de datos en bruto (.raw). El orden de los canales será el seleccionado en la configuración de entradas virtuales. El proceso para escribir las muestras es el opuesto al descrito en la función **CRenderFile::*SplitChannels()***, es decir, aquí a partir del array de bytes bidimensional se crea una ristra de bytes de las muestras intercaladas para así permitir la manipulación de estos ficheros en software de edición de audio.

Seguidamente se pasa a ejecutar la función ***CalculateDelays()***, la cual devuelve un vector con los retardos estimados para cada par de micrófonos tomando como entrada el array bidimensional de las muestras.

Una referencia a este vector es enviada a la interfaz principal, que se encarga de enviarlo a la clase **CRadarView** para que dibuje las hipérbolas obtenidas a partir de los retardos y ejecute el algoritmo GN para estimar una posición en el plano. El valor de esta posición es enviado de vuelta a la interfaz principal que inmediatamente lo devuelve al hilo de **CWasapiThread**. Por último, se escribe el resultado de la estimación de la posición para ese fragmento de audio en un documento de texto junto con los datos de la captura. A continuación, se pasa a procesar el siguiente fragmento de audio capturado en la siguiente iteración.

Como se aprecia en el diagrama, también se ha implementado un evento para finalizar la ejecución del hilo (**Evento Fin**). Dicho evento es señalizado desde la interfaz de usuario al pausar el proceso. Si el proceso se vuelve a reanudar se destruye este hilo y se crea uno nuevo que inicia un nuevo proceso de captura.

Esta clase ha sufrido importantes modificaciones con respecto al código del proyecto original, siendo la mayor parte código nuevo. En la parte de creación del hilo, de la función ***CreateThread()***, se ha modificado el código para permitir la inicialización de varios dispositivos, no solo de uno como estaba originalmente diseñado. En cuanto a la ejecución del propio hilo de trabajo, que incluye el bucle de procesamiento y se ejecuta en la función ***InitInstance()***, el código ha sido totalmente reescrito para implementar el manejo de varios dispositivos y los nuevos algoritmos de cálculo de retardos.

Las funciones que se encargan de escribir los ficheros de audio y texto también han sido modificadas, cambiando el formato de presentación de los resultados en el caso del documento de texto, para adaptarse a las nuevas características del software.

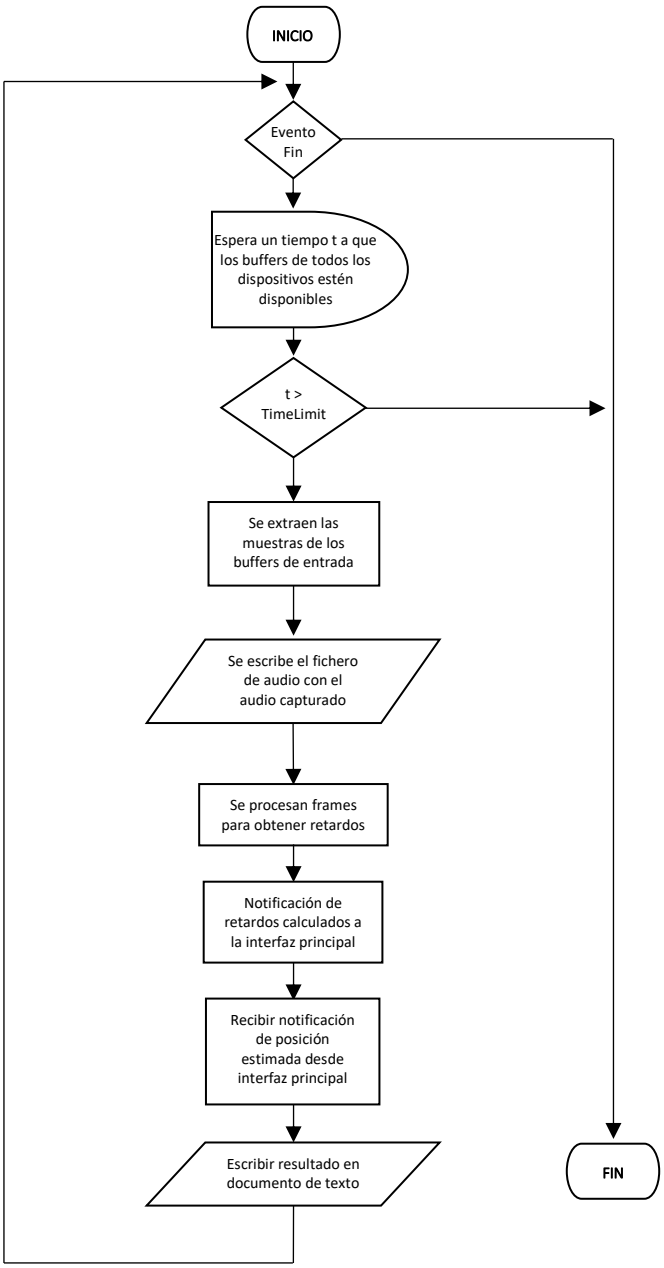

***Figura 17**. Diagrama de flujos del bucle de ejecución del Modo Dispositivo.*

4.1.4. Clase dedicada al control visual

CRadarView es una clase derivada de la Clase CStatic de MFC. A pesar de mantener el mismo nombre que la clase del proyecto original, el código en su totalidad es de aportación propia.

Tiene como principal cometido dibujar los elementos gráficos que componen la simulación del proceso de localización en la sala. Sus funciones son:

- Dibujar la planta de la sala a partir de las dimensiones introducidas por el usuario en la interfaz principal.

- Dibujar los micrófonos con su número identificativo en la posición correspondiente en la sala, dependiendo de la posición introducida por el usuario en la ventana de configuración de entradas y de la posición del array en la sala.
- Dibujar las hipérbolas a partir de los retardos que la clase de la interfaz principal, **CSoundSourcev2Dlg**, envía en cada iteración del bucle de procesamiento.
- Ejecutar el algoritmo de Gauss-Newton para estimar una posición en el plano e imprimir por pantalla en forma de cadena de texto el resultado. La razón de ejecutar el algoritmo GN en esta clase y no en las clases que implementan el bucle de ejecución (**CRenderFileInterface** para el Modo Fichero y **CWasapiThread** para el Modo Dispositivo) es que la función implementada del algoritmo GN requiere de dos parámetros de entrada fundamentales: los retardos estimados por la GCC entre cada par de micrófonos y la posición en la sala de cada micrófono. Los hilos de trabajo independientes solo cuentan con el primer parámetro mientras que la clase **CRadarView** cuenta con los dos, ya que las posiciones de los micrófonos los conoce para poder dibujarlos en el plano y el vector que contiene los retardos es enviado desde el hilo de trabajo independiente a través de la interfaz principal.

Así pues, la secuencia de acciones de este control visual es:

- Recibe el vector retardos e inmediatamente calcula las hipérbolas correspondientes a la vez que las dibuja.
- Una vez que ya están dibujadas todas las hipérbolas se ejecuta el algoritmo GN y se obtiene un punto del plano como resultado. Este resultado es enviado a la interfaz principal que a su vez lo envía al hilo de trabajo correspondiente para poder imprimirlo en un documento de texto.

El proceso de trazado de las parábolas en el plano no es en absoluto trivial y de hecho ha supuesto uno de los mayores retos de este proyecto. Dado que el manejo de múltiples ecuaciones de hipérbolas no centradas en el origen y con ejes focales no paralelos al eje de abscisas es casi inviable, para el dibujo de cada hipérbola se ha procedido de la siguiente forma:

1. En primer lugar, se resuelve la ecuación (5) a partir de los retardos recibidos y las distancias entre micrófonos.
2. Después se calculan una serie de puntos que forman la hipérbola como si estuviera centrada en el origen con el eje focal paralelo al eje x, con mayor resolución de puntos en la curva de la hipérbola.
3. Esta serie de puntos que forman la hipérbola se rotan con respecto al origen de coordenadas y se trasladan en el plano dependiendo de la posición de los micrófonos, de forma que quedan en las posiciones correctas.
4. Por último, se unen todos los puntos mediante líneas rectas para dibujar la hipérbola.

Dado que esta explicación resulta algo árida, se emplaza a revisar el código de la función ***DrawLine()*** para comprobar detalladamente el proceso llevado a cabo.

En la figura 18 se muestra un ejemplo del aspecto de este control visual tras procesar un vector de retardos y pintar las hipérbolas.

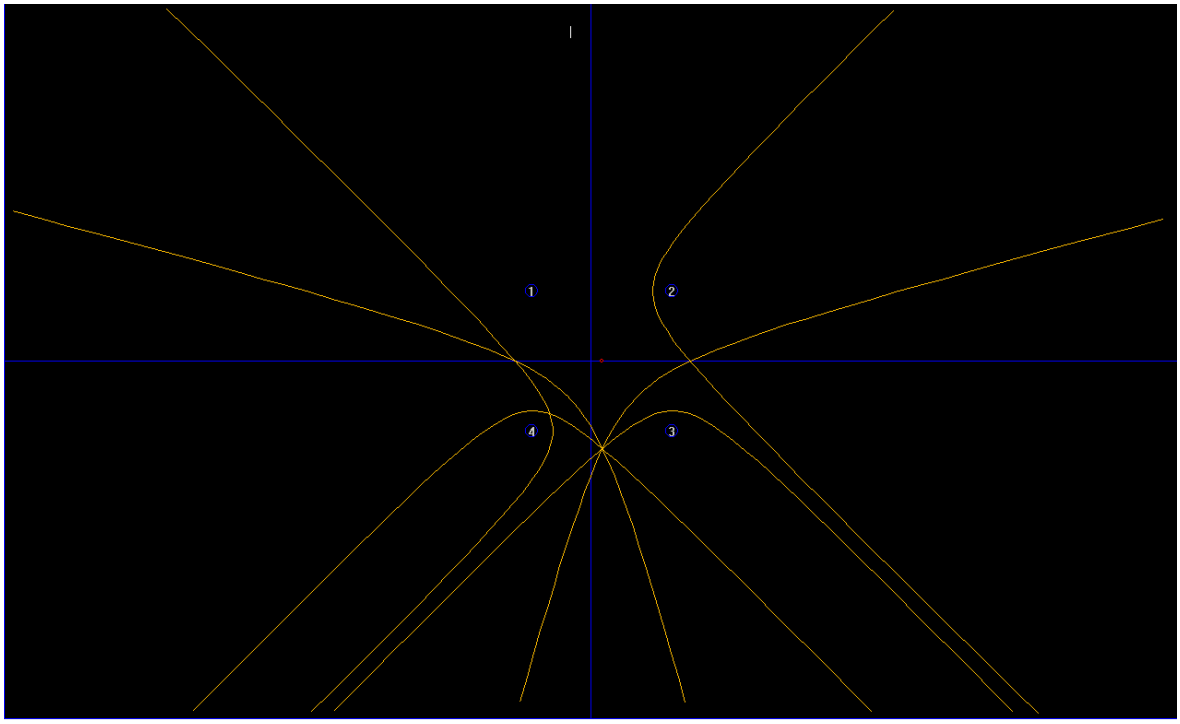

***Figura 18**. Ejemplo de dibujo realizado por la clase **CRadarView**.*

4.1.5. Clases y funciones auxiliares

Estas funciones han sido implementadas para servir de soporte para las operaciones realizadas en el resto de las clases. A continuación, se presentan brevemente cuáles son estas clases y funciones auxiliares.

a) Clase Matriz

Implementa todos los métodos para realizar cálculos con matrices como si de una aplicación de alto nivel se tratase. Esta clase fue implementada con el objetivo de servir de soporte para el algoritmo Gauss-Newton, que tiene una importante carga de cálculo matricial. Las operaciones disponibles con este tipo de objeto son:

- Suma
- Resta
- Multiplicación entre matrices
- Multiplicación por un escalar
- División entre un escalar
- Máximo
- Mínimo
- Matriz traspuesta
- Matriz inversa
- Matriz adjunta
- Determinante

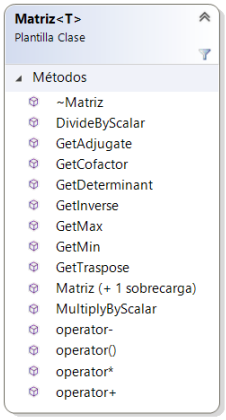

***Figura 19**. Clase Matriz.*

b) INT24

Clase implementada para simular un tipo de dato de 3 bytes, para poder manejar muestras de 24 bits. Esta implementación permite realizar las operaciones básicas de una variable de tipo numérico.

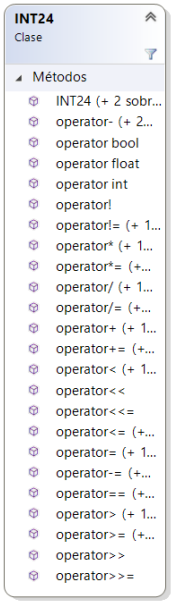

***Figura 20**. Clase INT24.*

c) Tools (funciones auxiliares)

En los ficheros Tools.h y Tools.cpp se aglutinan los algoritmos matemáticos utilizados para la localización de fuentes acústicas. Estas funciones son la siguientes:

- ***RealToComplex()***: convierte vectores de números reales en vectores de tipo *complex<double>* para poder ser manipulados en las funciones de fft de FFTW.
- ***GeneralisedCrossCorrelation()***: ejecuta la función de correlación cruzada generalizada de dos señales de entrada y se devuelve como resultado el retardo en segundos entre dichas señales (figura 3).
- ***CalculateDelays()***: ejecuta la función de la GCC para cada par de combinación de canales, dando como resultado un vector de retardos que será más tarde utilizado por la función **CRadarView** para dibujar las hipérbolas y para ejecutar el algoritmo de GN. Este método supone la parte más costosa computacionalmente de la aplicación.
- ***GaussNewtonAlgorithm()***: implementa el algoritmo iterativo de Gauss-Newton. A partir del vector de retardos devuelto por ***CalculateDelays()*** y de las posiciones de los micrófonos en el plano estima una coordenada donde convergen las hipérbolas y lo devuelve como parámetro de salida.

5. Pruebas y resultados

Se han realizado pruebas para comprobar el funcionamiento de la aplicación es sus dos modos de trabajo: Modo Fichero y Modo Dispositivo.

Para la prueba en Modo Dispositivo surge una problemática, detallada en el apartado 7.2, que impide poner a prueba el funcionamiento de los algoritmos, por lo que se decide poner a prueba la aplicación con un fichero pregrabado.

5.1. Prueba general de la aplicación

Para poner a prueba el funcionamiento de la aplicación se usan dos ficheros de audio multicanal correspondientes a una grabación realizada en cámara anecoica. El origen de coordenadas se considera el centro de la sala. Las dimensiones de la sala en esta prueba con de 5x3 m.

Las posiciones de los micrófonos y de la fuente son los siguientes:

Tabla 1. Posiciones de los micrófonos en la prueba.

MIC	X(m)	Y(m)
1	-0,21	0,05
2	0,04	0,20
3	-0,06	-0,20
4	0,29	-0,05

Tabla 2. Posiciones de las fuentes sonoras.

SOURCE	X(m)	Y(m)
Speaker 1	3,79	-0,95
Speaker 2	1,79	-0,95

Lo primero de todo debe aclararse que, en el montaje de micrófonos de estos audios, el eje z varía, es decir, los micrófonos están ligeramente a diferente altura, lo que afecta levemente al resultado. Como se puede observar en la figura 21, la mayor concentración de hipérbolas se produce en las inmediaciones del array de micrófonos, por lo que el punto que estima el algoritmo de GN es ese, dando de manera errónea una posición que no se asemeja a la realidad. Un poco más a la derecha de esta concentración se encuentra una intersección de tres asíntotas, que podría considerarse una aproximación a la posición real de la fuente. Debe considerarse que las diferentes alturas de los micrófonos pueden generar estas variaciones.

Para el caso de la fuente 1 el dibujo de las hipérbolas queda de la siguiente forma:

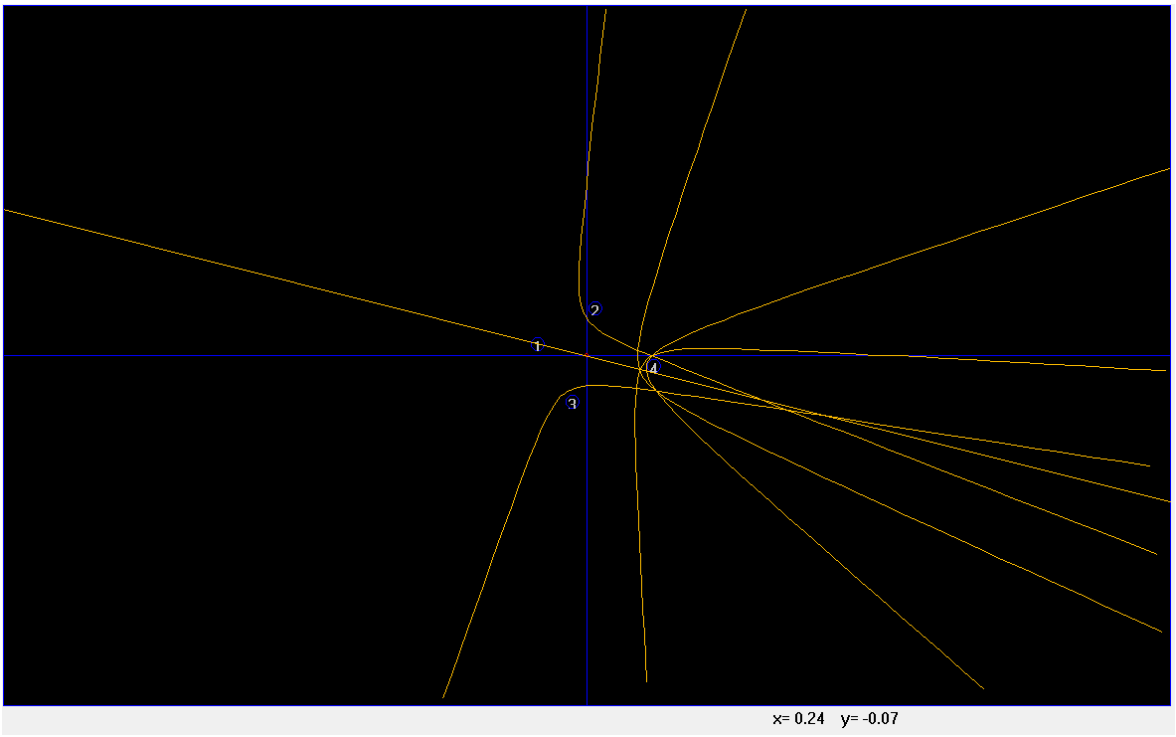

***Figura 21**. Control visual prueba general Speaker 1.*

En el caso del audio con la fuente 2 pasa algo parecido como puede observarse:

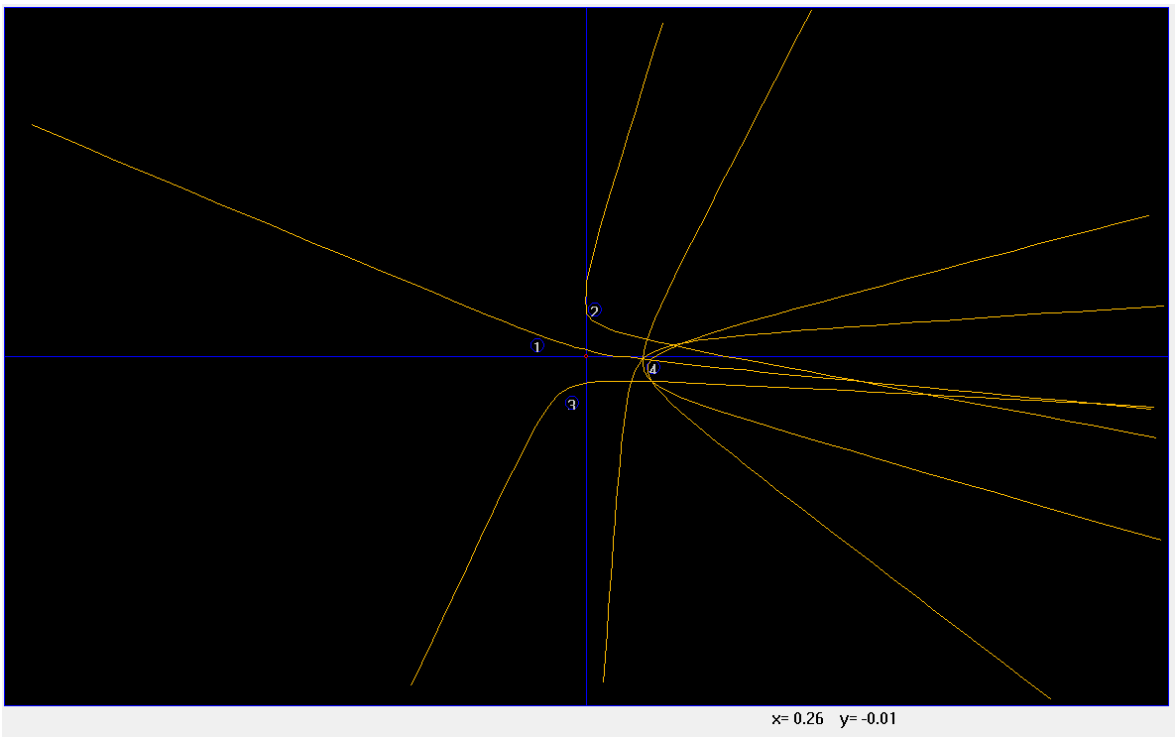

***Figura 22**. Control visual prueba general Speaker 2.*

Esta vez la fuente está en una posición más lejana. Precisamente, la intersección de la derecha es la que se muestra más desplazada con respecto al caso de la fuente 1. No obstante la mayor concentración de intersecciones se vuelve a producir en las inmediaciones del array, por lo que la estimación del punto (0.26, -0.01) vuelve a ser erróneo.

Una posible solución para este problema sería aumentar la separación entre micrófonos para así evitar que se produzca esas concentraciones de hipérbolas, y así permitir que el algoritmo de GN converja hacia la intersección de la derecha, que es considerada la intersección producida por la fuente.

6. Problemas e incidencias

Durante el desarrollo de este software se han notificado problemas eventuales surgidos de causas específicas. A continuación, se detallan dichos problemas y se describen los procedimientos llevados a cabo y las condiciones necesarias para impedir la aparición de estos eventos.

6.1. Demoras en tiempo de ejecución

Se ha experimentado que en ordenadores con características de procesamiento modestas (procesadores con cierta antigüedad o de gama media/baja), en ocasiones al habilitar más de dos canales como entrada se produce un problema de rendimiento que puede comprometer la ejecución de los procesos en tiempo real.

Como ya se ha mencionado anteriormente en el apartado 4.1.5, el cálculo de los retardos entre cada combinación de par de micrófonos mediante la GCC es, con diferencia, la sección de código más costosa computacionalmente, lo que a su vez se traduce en un mayor coste temporal. Precisamente cuando este coste temporal del cálculo supera la duración del tramo de audio que se quiere analizar es cuando la aplicación deja de trabajar en tiempo real.

Realmente si la aplicación está funcionando en **Modo Fichero** no tiene mayores consecuencias. Únicamente tarda en procesar el archivo mayor tiempo que la propia duración del audio, pero los resultados del proceso siguen siendo igual de válidos que si la aplicación funcionara en tiempo real.

En cambio, cuando la aplicación está trabajando en **Modo Dispositivo** se producen discontinuidades en el proceso de captura ya que el ritmo al que se analiza el audio es insuficiente para capturar todo el audio sin interrupciones. Esto da como resultado un archivo de audio sin continuidad temporal, con sucesivos cortes que invalidan por completo los resultados obtenidos.

La presencia de esta incidencia depende de la capacidad de procesamiento del ordenador y en ocasiones surge al habilitar más de 3 canales de entrada, más de 4, etc. Siempre dependiendo de la capacidad de procesamiento del ordenador.

La razón de que esto ocurra es el aumento exponencial de los tiempos de ejecución en función del número de canales a procesar. Por ejemplo: en el caso más sencillo posible, con dos canales de entrada, solo hay que calcular un retardo, para el cual se necesita un tiempo t para obtenerlo. En un caso más usual como es por ejemplo la utilización de 4 canales, hay 6 combinaciones de pares de micrófonos posibles, por lo tanto, se necesita un tiempo $6t$ para obtener los 6 retardos correspondientes[3]. Siguiendo el desarrollo de la progresión de combinaciones posibles de pares de micrófonos sin repetición (definida por $a_n=a_{n-1}+n$, siendo $a_1=1$) se infiere que los tiempos de ejecución para 5, 6, 7 y 8 canales son respectivamente $10t$, $15t$, $21t$ y $28t$.

La diferencia de la carga computacional de la sección de la GCC entre una configuración dada y otra con un mayor número de canales de entrada puede suponer una brecha que distingue entre un tiempo de ejecución dentro de los límites para poder ser considerado tiempo real de otro tiempo de ejecución que exceda estos límites. Es por ello por lo que se recomienda la ejecución de este software en el equipo con mayor capacidad de procesamiento disponible.

[3] Se considera que todos los demás parámetros de la configuración tales como formato de captura, longitud del buffer, etc. son similares tanto en la configuración de dos canales como en la de cuatro.

6.2. Compatibilidad de Windows con dispositivos multicaptura

Existe la posibilidad de que se produzcan determinadas incompatibilidades entre el sistema de captura y el sistema operativo Windows instalado en el ordenador. En el proyecto de la primera versión del software se señalan incompatibilidades surgidas entre una interfaz de audio Behringer modelo U-PHORIA UMC404HD y el sistema operativo Windows 8. En este proyecto se ha encontrado esta misma incompatibilidad entre una interfaz de audio Behringer modelo U-PHORIA UMC1820 y el sistema operativo Windows 10.

Esta incompatibilidad consiste en un reconocimiento erróneo por parte de Windows del dispositivo de entrada. La interfaz utilizada para este proyecto (Behringer modelo U-PHORIA UMC1820) tiene 8 canales de entrada, pero en **Propiedades de dispositivos de entrada** del **Panel de control** figura como 4 dispositivos de entrada estéreo distintos. Esto cambia radicalmente el enfoque del proyecto ya que desde el punto de vista de Windows se está tratando con distintos dispositivos, lo que se traduce en gestionar varios buffers a la vez.

Esta circunstancia ha sido determinante a la hora de diseñar e implementar el proceso de captura. Es por esto por lo que el código implementado en la clase **CWasapiThread** está preparado para capturar audio de **n** dispositivos, aunque para esta solución se ha limitado a 8. La **Ventana de configuración de entradas** también ha sido diseñada para poder ofrecer total versatilidad a la hora de elegir de donde proviene cada flujo de audio capturado, permitiendo escoger cualquier canal de cualquier dispositivo de grabación como entrada.

La captura de varios dispositivos está dirigida únicamente al caso de un dispositivo físico que es reconocido por Windows como distintos dispositivos.

En consecuencia, a pesar de que la aplicación permite la captura de distintos dispositivos se aconseja elegir siempre flujos de datos de entrada provenientes del mismo dispositivo físico. Con esto se evitan disconformidades entre los tiempos de latencia de distintos dispositivos. En este contexto, el tiempo de latencia corresponde al tiempo que transcurre desde que se envía a un dispositivo la orden de capturar hasta que comienza la captura. En función del modelo y del fabricante este factor puede variar en un orden de varios milisegundos. Esto puede no parecer una diferencia notable en un primer momento, pero este tiempo de latencia traducido en espacio recorrido por el sonido sí puede suponer una diferencia notable. Por ejemplo, 3 ms de latencia supone un metro de diferencia, lo que supone una imprecisión inadmisible para una aplicación de localización.

Aun en el caso de conseguir varios dispositivos con el mismo tiempo de latencia, surge un problema relacionado con el formato de captura. Aunque se seleccione el mismo formato de captura para todos los dispositivos, en la práctica la frecuencia de muestreo real no es matemáticamente igual a la teórica. Existe cierta tolerancia que puede inducir errores de sincronización entre las señales cada cierto tiempo, lo que una vez más invalidaría el proceso de localización.

También se han experimentado algunas dificultades en este aspecto, no ya relacionadas con el reconocimiento de Windows de los dispositivos sino de las limitaciones insalvables que impone WASAPI.

Concretamente, en el caso de las pruebas con la interfaz de ocho canales se ha observado que no es posible capturar audio de más de dos canales, perteneciendo ambos al mismo "dispositivo estéreo" erróneamente reconocido. Paradójicamente, a la hora de capturar se establece enlace con el resto de "dispositivos estéreo" sin ningún problema, pero durante el proceso de captura la señal recibida de esos canales es nula. Los dispositivos estéreo actúan excluyéndose mutuamente, e incluso excluyendo a otros dispositivos de captura que también precisan de la instalación de drivers.

Claramente se advierte que la compatibilidad de estos dispositivos con WASAPI no es del todo completa, ya que a partir de los hechos citados se desliza la incapacidad de WASAPI para gestionar distintos dispositivos, cada uno con su controlador. En cambio, en entradas que no precisan de la instalación de drivers (micrófono interno del portátil, entrada por minijack, etc.) no se ha observado

ninguna incidencia que afecte a la captura de audio, a parte del problema de latencia anteriormente citado.

6.3. Fallas en la reproducción en Modo Fichero

Este problema no afecta al rendimiento de la aplicación ni a los resultados obtenidos, pero aun así es pertinente exponerlo.

Como ya se ha mencionado, el proceso de reproducción del audio en **Modo Fichero** está dirigido por eventos. Resumiendo brevemente, se escriben los datos de audio en el buffer del dispositivo de salida predeterminado y hasta que este buffer no está disponible para volver a ser escrito no se señaliza el evento que permite seguir con la escritura de datos.

Ahora bien, lo deseable sería que el evento se señalizara cuando el buffer del dispositivo quedara totalmente vacío para que así la cantidad de datos que se escriban coincidan con el tamaño del buffer que ha configurado el usuario, es decir, que se rellene el buffer al completo con nuevos datos ya que el fragmento de audio que se analiza es el correspondiente a los datos escritos en el buffer.

Como es habitual, para asegurar la continuidad del flujo de datos el sistema no espera a que el buffer se haya vaciado completamente para señalizar el evento que permite la escritura. La señalización se produce en cuanto hay un espacio libre (de tamaño aleatorio) en el buffer, lo que provoca que la cantidad de datos de audio escritos cada vez sea inferior al tamaño total del buffer. Esto resulta en una divergencia entre la longitud del fragmento de audio a analizar configurado por el usuario y el fragmento que en realidad se analiza, que es (mucho) menor del seleccionado.

La interfaz de WASAPI que gestiona el proceso de reproducción (IAudioClient) no proporciona ningún método para controlar el espacio libre mínimo para señalizar el evento, pero sí que proporciona un método para conocer la cantidad de datos que quedan en el buffer a la espera de ser leídos.

La solución a este problema consiste en pausar el bucle de reproducción hasta que la cantidad de datos en el buffer a la espera de ser leídos sea cero, es decir, que se vacíe por completo el buffer. Así se asegura que la longitud del fragmento de audio analizado se corresponda con la longitud seleccionada por el usuario. El inconveniente de esta solución es que cada vez que se vacía por completo el buffer se producen pequeños cortes en la reproducción (*glitches*)que son audibles en el monitoreo de la señal. Por lo demás, no se afecta a ningún aspecto que involucre al proceso de cálculo ni a los resultados.

Dado que el cometido principal de esta aplicación es el análisis de audio, se ha considerado que merece la pena optimizar el procesado de audio en detrimento de un elemento menos prioritario como es el monitoreo de la señal.

6.4. Tiempo de latencia desmesurado

En el modo de trabajo **Modo Dispositivo,** se ha advertido de un problema ocasional que surge al ejecutar por primera vez el proceso de captura. En este primer proceso de captura puede observarse que el desfase entre señales de distintos dispositivos es mucho mayor de lo que correspondería a una diferencia de latencias. Acceder a un dispositivo por primera vez conlleva más tiempo debido a que no se tiene ningún dato en memoria que facilite su acceso como sí ocurre en las siguientes veces.

Por ello se recomienda que, si se va a capturar desde distintos dispositivos, se haga un rápido proceso de captura de prueba antes de proceder a la captura válida.

7. Conclusiones

En vista de los resultados obtenidos en las pruebas planteadas en el apartado 5 y a los problemas notificados en el apartado 6, se concluye que, pese a haber cumplido los objetivos principales de este proyecto, expuestos en el apartado de especificaciones, la línea de investigación de localización de fuentes debe tomar otros caminos fuera del entorno Windows.

Se ha demostrado que las librerías específicas de audio de Windows resultan altamente limitantes en tanto se quieran implementar aplicaciones que requieren de cierta complejidad como es el caso del software aquí desarrollado. La imposibilidad de trabajar adecuadamente con dispositivos de captura multicanal lastra determinantemente el uso de esta aplicación para medidas in situ.

Todo esto parece indicar que la línea de trabajo debe pasar inevitablemente por el uso de una librería que no adolezca de los numerosos problemas de WASAPI. El principal candidato para esto es el protocolo de audio ASIO (*Audio Stream Input/Output*) desarrollado por Steinberg, utilizado por la mayoría de las aplicaciones de audio profesional en Windows (Cubase, ProTools, Nuendo, WaveLab, etc.). Este protocolo proporciona un enlace de baja latencia sin pasar por las interfaces de Windows entre el software y los dispositivos de audio, de manera que accede de manera directa a los controladores de las tarjetas de sonido, ya sean internas o externas. De hecho, todas las interfaces de sonido externas de altas prestaciones, como la utilizada durante las pruebas (Behringer modelo U-PHORIA UMC404HD), son compatibles con ASIO, debido a que es la librería prioritaria en aplicaciones de audio profesionales, mientras que apenas se menciona la compatibilidad con WASAPI o si lo hacen es sin dar ningún tipo de detalle adicional. También podría ser interesante explorar las posibilidades que ofrece el proyecto de librería ASIO4ALL [24], un software gratuito que pretende simular las prestaciones de la librería ASIO que, en cambio, requiere del pago de una licencia.

No obstante, a pesar de las limitaciones tecnológicas impuestas por Windows con las que se ha topado el proyecto, la aplicación desarrollada contiene numerosas ideas y soluciones que pueden ser exportadas a otros proyectos, independientemente del lenguaje de programación utilizado. El código fuente contiene gran cantidad de comentarios para ayudar en esta labor, ya que de hecho ese era unos de los objetivos del proyecto: diseñar código reutilizable.

Además, se ha demostrado el buen funcionamiento de la aplicación en el procesado de ficheros pregrabados, lo que supone un punto importante a favor de la versatilidad de la aplicación ya que en este modo de trabajo no se depende de formatos soportados por los dispositivos ni demás limitaciones técnicas.

Referido ya al propio proceso de localización de fuentes, se hace patente que la solución propuesta resulta exitosa en la estimación de la dirección de la fuente sonora, pero no tanto en la estimación de una posición. La estimación errónea de la posición de la fuente sonora no es consecuencia de una mala implementación de los métodos matemáticos utilizados sino de la propia naturaleza del método. Este es el caso del algoritmo de Gauss-Newton para resolver el problema de mínimos cuadrados. El objetivo de este método es hallar aquel punto aproximado donde las hipérbolas intersecan. El problema es que los arrays utilizados en estas medidas cuentan con micrófonos muy próximos a otros, lo que produce una concentración de intersecciones mayor en el espacio entre micrófonos que en el propio punto donde se estima que está la fuente, como puede apreciarse en las figuras 21 y 22 del apartado 5. Este es uno de los grandes problemas de las soluciones de estimación de posiciones basados en TDOA. La solución para evitar, en la medida de los posible, que se produzcan estas concentraciones de hipérbolas sería utilizar una mayor distancia entre micrófonos.

Como se ha podido observar lo largo de esta memoria, a pesar de que el software desarrollado estima posiciones en un punto del plano, la coordenada z aparece en algunos controles de la aplicación, como en los controles de posición de los micrófonos o de posición del array. En un

principio se planteó la posibilidad de escalar la aplicación a tres dimensiones, pero finalmente al resultar más complicado de lo previsto, el proyecto se centró en la estimación en un plano. Por esta razón, tanto en la interfaz de usuario como dentro del código, pueden encontrarse multitud de referencias al eje z, así como implementaciones incluyendo esta dimensión que finalmente han quedado inutilizadas pero que están contenidas en el proyecto para facilitar futuras ampliaciones.

Precisamente la estimación de una posición en el espacio es una de las líneas de investigación futuras que pueden seguirse a partir de esta aplicación. Para ello se debe trabajar con la figura tridimensional de la hipérbola, el hiperboloide (figura 23). La solución debería estimar la intersección en el espacio de los múltiples hiperboloides obtenidos a partir del TDOA, relacionando de alguna manera la ecuación del hiperboloide de dos hojas (25) con la ecuación de la diferencia de distancias.

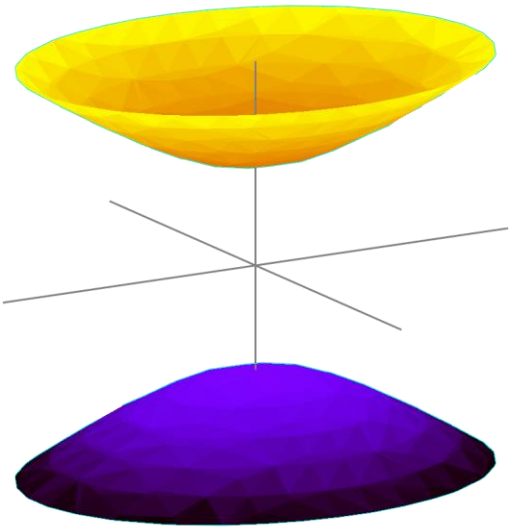

***Figura 23**. Hiperboloide de dos hojas.*

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{z^2}{c^2}=-1 \tag{25}$$

Otro aspecto mejorable de la aplicación podría ser añadir filtrado en frecuencia a las señales de entrada. Como se ha mencionado en el marco teórico de la GCC, las ponderaciones frecuenciales se ven más afectadas por el ruido que la versión de la GCC sin ponderación. Para potenciar estas ponderaciones sería interesante filtrar las bandas de frecuencia que no sean de interés para la localización. Este filtrado se adaptaría en función de la naturaleza espectral de la fuente acústica, para así procesar la información relevante para la localización.

8. Referencias

[1] PEREIRA MARTINEZ, J.C. (2018). *Procesamiento de audio en tiempo real para estimar la dirección de fuentes sonoras*. Proyecto final de carrera. Madrid: ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN. [En línea]. Disponible en: http://oa.upm.es/53213/. [Accedido: 5-oct-2020].

[2] Microsoft. "About WASAPI", 5 de diciembre de 2018. [En línea]. Disponible en: https://docs.microsoft.com/en-us/windows/win32/coreaudio/wasapi. [Accedido: 5-oct-2020].

[3] Microsoft. "About MMDevice API", 5 de diciembre de 2018. [En línea]. Disponible en:https://docs.microsoft.com/en-us/windows/win32/coreaudio/mmdevice-api. [Accedido: 5-oct-2020].

[4] BLANCHARD, Walter. The genesis of the Decca navigator system. *The Journal of Navigation*, 2015, vol. 68, no 2, p. 219-237.

[5] KNAPP, Charles; CARTER, Glifford. The generalized correlation method for estimation of time delay. *IEEE transactions on acoustics, speech, and signal processing*, 1976, vol. 24, no 4, p.320-327.

[6] Microsoft, "About the Windows Core Audio APIs", 5 de diciembre de 2018. [En línea]. Disponible en:https://docs.microsoft.com/en-us/windows/win32/coreaudio/about-the-windows-core-audio-apis. [Accedido: 6-oct-2020].

[7] Microsoft, "DirectSound", 9 de octubre de 2011. [En línea]. Disponible en: https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ee416960(v=vs.85). [Accedido: 6-oct-2020].

[8] Microsoft, "Aplicaciones de escritorio de MFC", 28 de julio de 2019. [En línea]. Disponible en: https://docs.microsoft.com/es-es/cpp/mfc/mfc-desktop-applications?view=vs-2019. [Accedido: 6-oct-2020].

[9] LEHMANN, Charles H. "La hipérbola". En: *Geometría analítica.* México D.F.: EDITORIAL LIMUSA, 1989. pp. 191-211.

[10] ROTH, Peter R. Effective measurements using digital signal analysis. *IEEE spectrum*, 1971, vol. 8, no 4, p. 62-70.

[11] CARTER, G. Clifford; NUTTALL, Albert H.; CABLE, Peter G. The smoothed coherence transform. *Proceedings of the IEEE*, 1973, vol. 61, no 10, p. 1497-1498.

[12] CARTER, G. C.; NUTTALL, A. H.; CABLE, P. G. The smoothed coherence transform (SCOT). *Naval Underwater Systems Center, New London Lab., New London, CT, Tech. Memo TC-159-72*, 1972.

[13] MENSING, Christian; PLASS, Simon. Positioning algorithms for cellular networks using TDOA. En *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*. IEEE, 2006. p. IV-IV.

[14] SIROLA, Niilo. Closed-form algorithms in mobile positioning: Myths and misconceptions. En *2010 7th Workshop on Positioning, Navigation and Communication*. IEEE, 2010. p. 38-44.

[15] Microsoft. "Exclusive-Mode Streams", 5 de diciembre de 2018. [En línea]. Disponible en: https://docs.microsoft.com/en-us/windows/win32/coreaudio/exclusive-mode-streams. [Accedido: 9-oct-2020].

[16] Microsoft. "User-Mode Audio Components", 5 de diciembre de 2018. [En línea]. Disponible en: https://docs.microsoft.com/en-us/windows/win32/coreaudio/user-mode-audio-components. [Accedido: 9-oct-2020].

[17] FFTW, "The Fastest Fourier Transform in the West". [En línea]. Disponible en: http://www.fftw.org/index.html. [Accedido: 14-sep-2020].

[18] Microsoft. "CWinApp (clase)", 15 de julio de 2019. [En línea]. Disponible en: https://docs.microsoft.com/es-es/cpp/mfc/reference/cwinapp-class?view=vs-2019. [Accedido: 7-oct-2020]

[19] KABAL, Peter. Audio File Format Specifications, 2 de mayo de 2017. [En línea]. Disponible en: http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html. [Accedido: 22-sep-2020].

[20] SoundFile. "WAVE PCM soundfile format". [En línea]. Disponible en: http://soundfile.sapp.org/doc/WaveFormat/. [Accedido: 21-sep-2020].

[21] Microsoft. "WAVEFORMATEXTENSIBLE structure (mmreg.h)", 12 de mayo de 2018. [En línea]. Disponible en: https://docs.microsoft.com/en-us/windows/win32/api/mmreg/ns-mmreg-waveformatextensible. [Accedido: 24-sep-2020].

[22] Microsoft. "Extensible Wave-Format Descriptors", 30 de junio de 2020. [En línea]. Disponible en: https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/extensible-wave-format-descriptors. [Accedido: 24-sep-2020].

[23] Microsoft. “IAudioClient::Initialize method (audioclient.h)”, 12 de mayo de 2018. [En línea]. Disponible en: https://docs.microsoft.com/en-us/windows/win32/api/audioclient/nf-audioclient-iaudioclient-initialize. [Accedido: 7-oct-2020].

[24] ASIO4ALL. “ASIO4ALL - Universal ASIO Driver For WDM Audio”. [En línea]. Disponible en: http://www.asio4all.org. [Accedido: 9-oct-2020].

ANEXO I. Manual de usuario

Índice

1. Requisitos técnicos

El software SoundSource v2.0 ha sido desarrollado para trabajar en el entorno Microsoft Windows, siendo compatible con los sistemas operativos Windows 7 y posteriores.

Un requisito imprescindible es tener instalado el IDE Visual Studio 2019 o en su defecto tener instalados los paquetes Microsoft Visual C++ Redistributable 2019[1]. Estos paquetes son necesarios para poder ejecutar aplicaciones de C++ compiladas con Visual Studio 2019.

El programa hace uso de librerías externas precompiladas, por lo que la biblioteca de vínculos dinámicos (DLL) incluido en el archivo comprimido proporcionado siempre debe compartir el mismo directorio que el archivo ejecutable.

El ejecutable proporcionado está compilado en 64 bits. En el apartado 2 se detallan los pasos a seguir para generar un archivo ejecutable, ya sea de 32 o 64 bits, a partir del código fuente.

2. Generación del archivo ejecutable

2.1. Instalación de Microsoft Visual Studio

El primer paso es descargar e instalar el entorno de desarrollo Microsoft Visual Studio, cuya versión de 2019 puede descargarse en [2].

Existen tres versiones del IDE Microsoft Visual Studio: Community, Professional y Enterprise. La versión Community es gratuita para desarrolladores particulares y usos académicos, y es donde se ha desarrollado el software de este proyecto por lo que se recomienda descargar esta versión.

Una vez ejecutado el instalador y aceptados los términos y condiciones aparecerá una ventana (figura 1) donde se deben seleccionar que cargas de trabajo se quieren instalar. Para este proyecto se requieren los siguientes:

- Desarrollo para el escritorio con C++
- Desarrollo de la plataforma universal de Windows

En la misma ventana aparece una pestaña denominada “Componentes individuales” en la cual se debe comprobar que los siguientes componentes están marcados, y marcarlos en caso de ser necesario:

- Paquete de compatibilidad de .NET Framework.
- Compatibilidad con C++/CLI para las herramientas de compilación de v142.
- Compatibilidad con la Plataforma universal de Windows de C++ para las herramientas de compilación de v142.
- Herramientas de compilación de MSVC v142 - VS 2019 C++ para ARM.
- Herramientas de compilación de MSVC v142 - VS 2019 C++ para ARM64.
- Herramientas de compilación de MSVC v142 - VS 2019 C++ para x64/x86.
- MSBuild.
- Herramientas de compilación de ATL de C++ v14.21 para v142.
- Últimas herramientas de compilación de ATL de C++ para v142 (x86 y x64).
- Últimas herramientas de compilación MFC de C++ para v142 (x86 y x64).
- Windows 10 SDK.

Una descripción más detallada de la instalación de Visual Studio puede encontrarse en [3]. Después de haberse instalado todos los componentes, Visual Studio ya está listo para su uso.

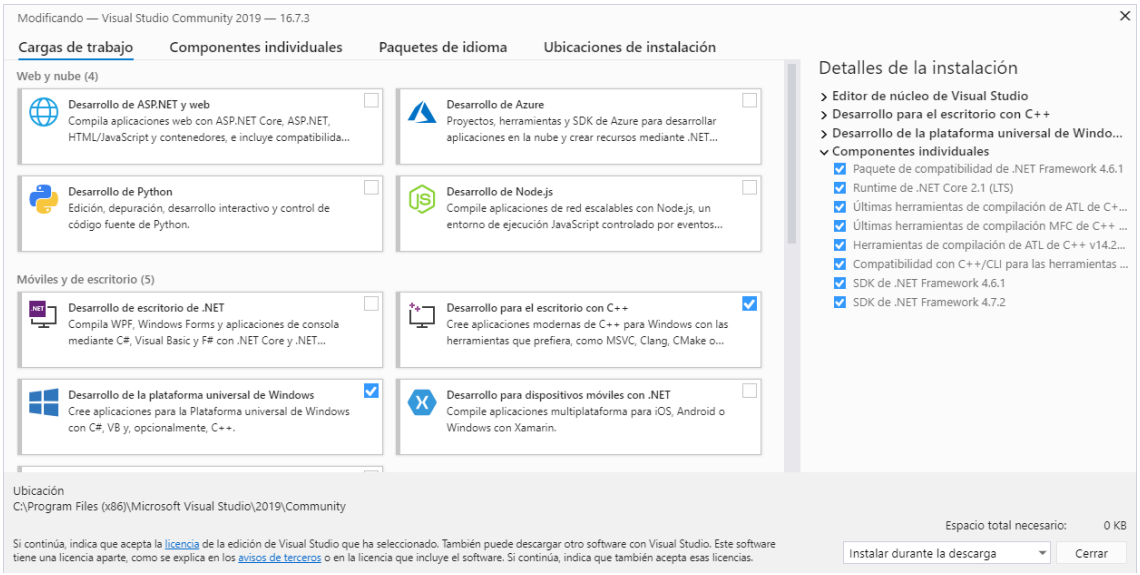

Figura 1. *Ventana de configuración de instalación.*

2.2. Obtención de la librería precompilada[4]

El software SoundSource v2.0 hace uso de la librería de software libre FFTW [4], el cual aglutina las subrutinas necesarias para ejecutar transformadas de Fourier con una alta eficiencia. En su página web se pueden descargar los archivos precompilados para Windows, en 32 y 64 bits [5]. Para obtener la librería de importacion (.lib) que más tarde será vinculada al proyecto deben seguirse los siguientes pasos:

1. Descargar el archivo .zip disponible en la página oficial de FFTW, ya sea la versión de 32 o de 64 bits, y descomprimirlo.
2. Abrir el Visual Studio Developer Command prompt: pulsar **Inicio→ Todos los programas→ Visual Studio 2019→ Visual Studio Tools→ Developer Command Prompt for VS**.

[4] Para la compilación del código proporcionado no es necesario descargar ningún fichero adicional; ya están incluidos en el proyecto y solo es necesaria una breve configuración (descrita en el apartado 2.3) para poder compilar el proyecto. El procedimiento explicado en este apartado va dirigido a aquellos usuarios que pretendan compilar en 32 bits o que simplemente quieran saber cómo se obtiene la librería precompilada (.lib).

3. Navegar hasta el directorio donde se encuentren los archivos descomprimidos e introducir la siguiente línea: `lib /machine:x64 /def:libfftw3-3.def`, x86 si el sistema es de 32 bits. Esto creará la biblioteca de importación **libfftw3-3.lib**. El formato de la librería del ejemplo de la figura 2 tiene una precisión *double*, que es el utilizado en el software. También pueden elegirse librerías con formato en coma flotante de simple precisión (libfftw3-3f.def) y formato *long-double*(libfftw3-3l.def).

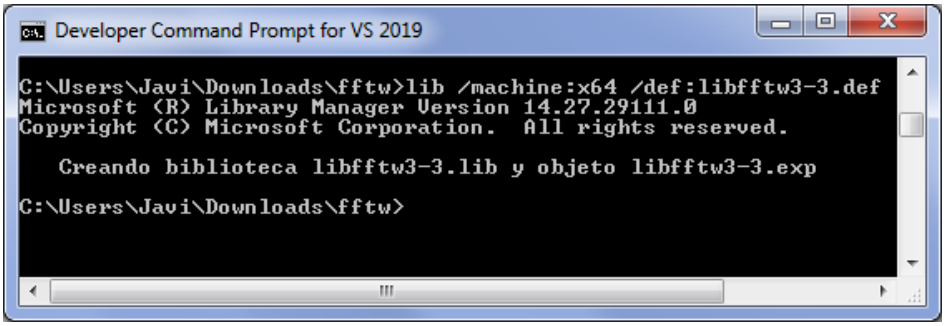

Figura 2. *VS Developer Command Prompt.*

2.3. Compilación del código fuente

Para generar el archivo ejecutable SoundSourcev2.exe deben seguirse los siguientes pasos:

1. Abrir Visual Studio y, en Tareas iniciales, marcar "Abrir un proyecto o una solución". Se abrirá un cuadro de diálogo. Navegar hasta la carpeta del código fuente y seleccionar el fichero de solución SoundSourcev2.sln.

2. En el Explorador de soluciones, situado en la derecha de la ventana, hacer click derecho en el proyecto y seleccionar Propiedades. Aparecerá una ventana como la de la figura 3. En **Configuración** en la esquina superior izquierda debe estar seleccionado **Todas las configuraciones**. Navegar hasta **Propiedades de configuración→ Vinculador→ General**, y en **Directorios de bibliotecas adicionales** seleccionar el directorio donde se encuentre la librería de importacion.(.lib). Después, en **Vinculador→ Entrada→ Dependencias adicionales,** escribir el nombre de la librería de importación, extensión incluida.

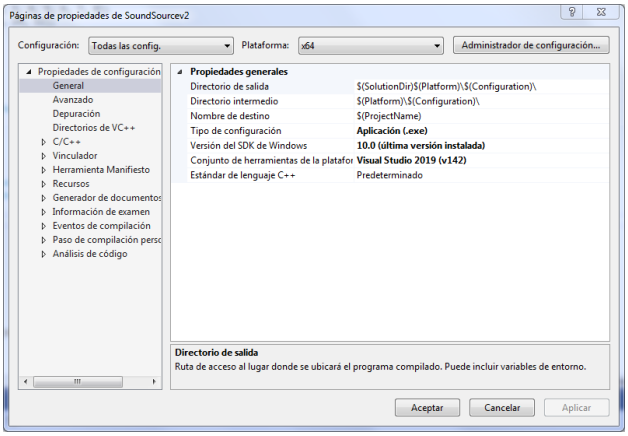

Figura 3. *Propiedades del proyecto.*

3. En la barra de herramientas de Visual Studio (figura 4) se selecciona qué versión se quiere compilar, la versión a distribuir (Release) o la versión con toda la información de depuración (Debug), además de la compatibilidad del software con sistemas de 32 o 64 bits (x86 y x64, respectivamente).

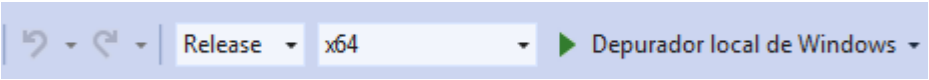

***Figura 4**. Selección del modo de compilación y compatibilidad del sistema en VS*

4. En el menú principal de VS seleccionar **Compilar→ Compilar Solución**. En el subdirectorio x64(o x86) /Debug o en x64(o x86) /Release aparecerá el ejecutable SoundSourcev2.exe. Para que el compilador no arroje errores de vinculación se recuerda que la biblioteca de enlace dinámico (.dll) debe estar en los directorios mencionados anteriormente.

3. Manejo básico de la aplicación

Una vez ejecutado el archivo **Soundourcev2.exe** aparecerá la ventana principal de la aplicación, mostrada en la figura 5. En la ventana se aprecian los distintos controles visuales que el usuario puede manejar.

En el menú de la aplicación hay 4 pestañas:

- **Modo**. Se elige en qué modo de trabajo va a funcionar la aplicación.
- **Open**. Se abre un fichero de audio .wav para su procesamiento (solo para MODO FICHERO).
- **Configuration**. Se abre un submenú con dos elementos:
  - **Inputs**: se activan los canales de entrada, se configura la posición de los micrófonos en el array, así como los dispositivos de entrada a utilizar, formato, canales, etc. en caso de trabajar en MODO DISPOSITIVO.
  - **Delay estimation**: se configuran los parámetros del cálculo de los retardos.
- **About**. Información sobre el software.

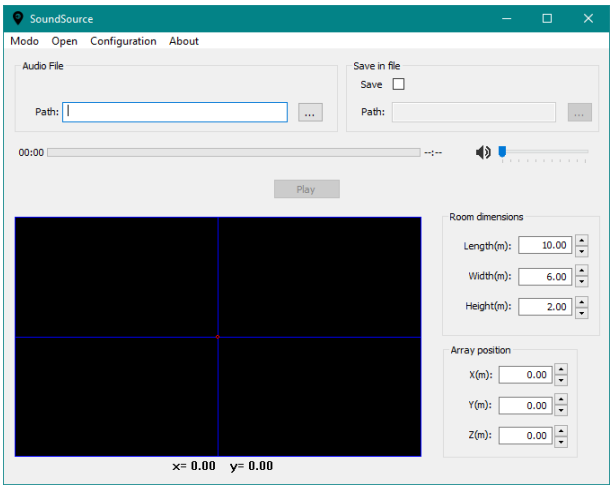

***Figura 5**. Ventana principal de SoundSource v2.0.*

El primer paso es seleccionar en **Menú→Modo** (figura 6) en qué modo de trabajo va a funcionar la aplicación:

- MODO FICHERO: el procesamiento se realiza sobre el audio de un fichero .wav multicanal.
- MODO DISPOSITIVO: el procesamiento se realiza sobre el audio capturado por los dispositivos de entrada que el usuario configure.

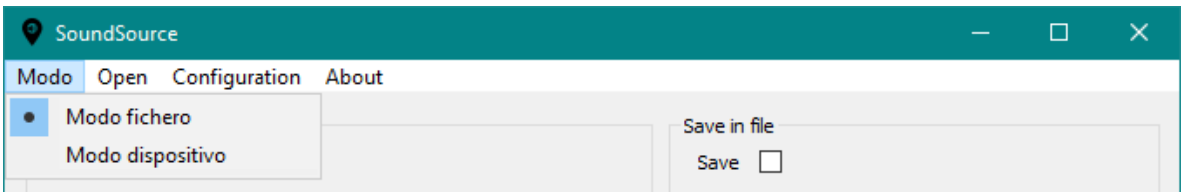

***Figura 6**. Menú de modo de trabajo.*

3.1. Funcionamiento en MODO FICHERO

En **Modo Fichero** el audio a procesar proviene de un fichero .wav de varios canales que ha sido grabado previamente. Este fichero de audio puede abrirse mediante la pestaña **Open** del menú principal o a través del grupo de controles denominado **Audio File,** como se muestra en la figura 7. En ambos casos se abre un cuadro de diálogo para seleccionar el fichero elegido por el usuario.

El siguiente paso es la configuración de las entradas. Pulsar **Menú→Configuration→Inputs** para

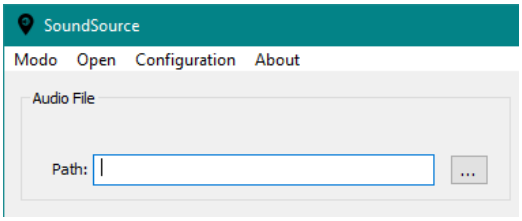

***Figura 7**. Selección del fichero de audio.*

abrir el cuadro de diálogo mostrado en la figura 8.

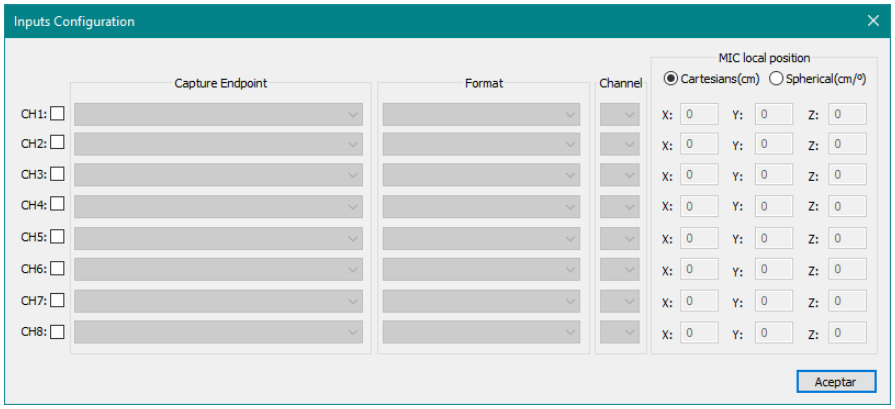

***Figura 8**. Ventana de configuración de entradas.*

En esta ventana de configuración los grupos de controles **Capture Endpoint**, **Format** y **Channel** no están habilitados para este modo de trabajo.

Marcar los canales de entrada que se vayan a utilizar para el procesamiento. En este caso los canales marcados deben coincidir con el número de canales del fichero multicanal seleccionado previamente.

En el grupo de controles **MIC local position** se deben introducir las posiciones que ocupan los micrófonos dentro del array utilizado para grabar el audio multicanal. Estas coordenadas locales pueden introducirse como coordenadas cartesianas (en cm) o como coordenadas esféricas (distancia en cm y ángulos en º).

En la figura 9 se muestra un ejemplo de cómo completar esta ventana para el caso de un fichero de audio de cuatro canales que fueron grabados con un array de 4 micrófonos dispuestos en forma de cuadrado de 1m de lado. El centro del array se considera el origen local de coordenadas.

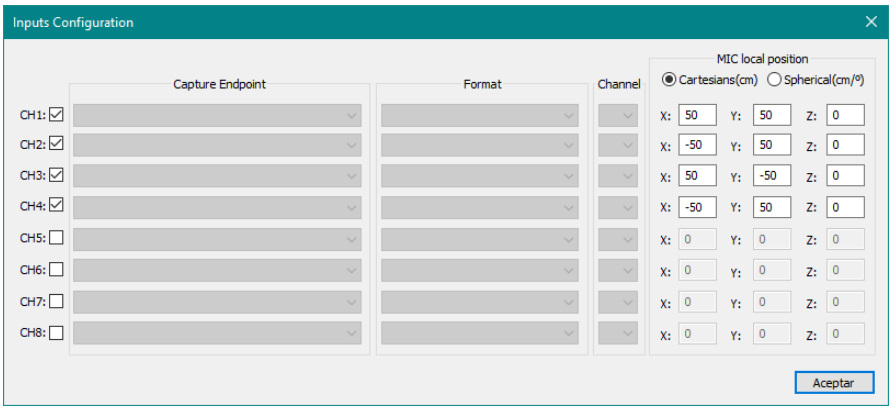

***Figura 9**. Ventana de configuración de entradas rellenada.*

Una vez seleccionado el fichero de audio y configuradas las entradas, el botón **Play** se habilita automáticamente. La reproducción del fichero del audio y su procesamiento pueden ser pausados tantas veces como sea necesarias. La aplicación ya está lista para ponerse en marcha a falta de configurar la sala y la posición del array.

Para elegir los parámetros de estimación del retardo ir a **Menú→Configuration→Delay Estimation**. Aparece la ventana mostrada en la figura 10. En ella se configura la ponderación frecuencial de la GCC y el tamaño del buffer, en ms o s. El tamaño del buffer determina la sección temporal que se analiza en cada cálculo de la posición. La configuración por defecto es sin ponderación (CC) y buffer de 500 ms.

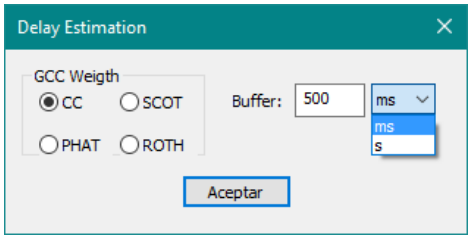

***Figura 10**. Ventana de configuración de estimación retardo.*

Hay otros dos controles visuales que están exclusivamente relacionados con el Modo Fichero:

- **La barra de reproducción**: a parte del indicador visual, se indica mediante un marcador numérico el tiempo transcurrido desde el inicio, así como la duración total del fichero de audio.

- **El control de volumen**: para realizar labores de monitoreo se añade este control para permitir escuchar el audio al tiempo que se analiza. Por defecto el volumen está a cero.

***Figura 11**. Barra de reproducción y control de volumen.*

Para conocer sobre los controles visuales relacionados con la sala y con el guardado de archivos, ir al apartado 3.3 y 3.4 respectivamente.

3.2. Funcionamiento en MODO DISPOSITIVO

En **Modo Dispositivo** el audio a procesar proviene de dispositivos de captura de audio conectados al ordenador. Al seleccionar este modo de trabajo, el grupo de controles Audio File, asociado al Modo Fichero, queda deshabilitado.

Para configurar las entradas de audio, ir a **Menú→Configuration→Inputs.** En el cuadro de diálogo que se abre (figura 8), marcar el número de canales de entrada a utilizar en el proceso. Configurar los canales de entrada mediante la selección de las siguientes propiedades:

- **Capture Endpoint**. Elegir el dispositivo del cual se va a tomar la señal de entre los disponibles en la lista desplegable que aparece.
- **Format**. Elegir el formato en el cual se va a realizar la captura de entre los formatos disponibles que tiene el dispositivo seleccionado anteriormente. La profundidad de bits y la frecuencia de muestreo deben coincidir en todos canales.
- **Channel**. Seleccionar el canal de entrada físico del dispositivo del cual se va a tomar la señal.

En el grupo de controles **MIC local position,** introducir las posiciones que ocupan los micrófonos en el array. Estas coordenadas locales pueden introducirse como coordenadas cartesianas (en cm) o como coordenadas esféricas (distancia en cm y ángulos en º).

En la figura 12 se muestra un ejemplo de configuración. En este ejemplo se habilitan dos canales de entrada, ambos provenientes del mismo dispositivo de captura. El formato elegido es 16 bits, 44100 Hz. El dispositivo Microphone es un dispositivo de captura de dos canales. Se asigna a cada canal virtual un canal físico del dispositivo. La distancia entre los micrófonos asociados a cada canal aquí es de 1,4 m, ya que se considera el centro entre estos dos micrófonos como el centro del array.

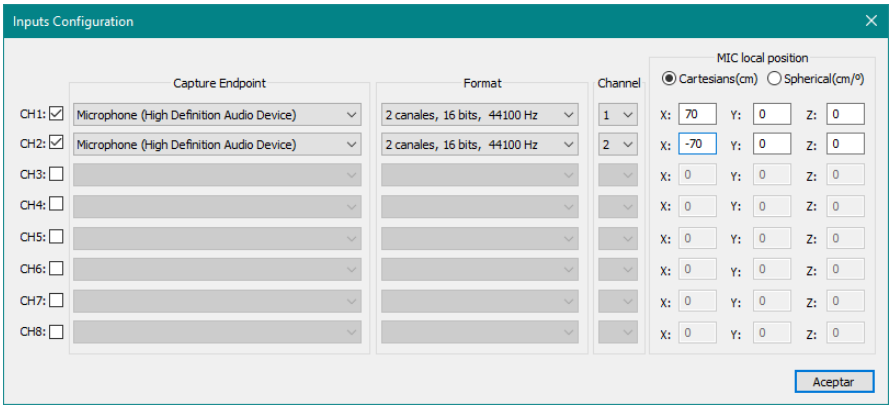

***Figura 12**. Ventana de configuración de entradas modo dispositivo.*

Una vez configuradas las entradas, se habilita automáticamente el botón **Play**. Ya se puede iniciar el proceso de localización, aunque en este punto, con la configuración por defecto de sala, array y estimación de retardos.

Para seleccionar los parámetros de estimación del retardo ir a **Menú→Configuration→Delay Estimation**. En la ventana mostrada en la figura 10 se configura la ponderación frecuencial de la GCC y el tamaño del buffer, en ms o s. El tamaño del buffer determina la sección temporal que se analiza en cada cálculo de la posición. La configuración por defecto es sin ponderación (CC) y buffer de 500 ms.

Para conocer sobre los controles visuales relacionados con la sala y con el guardado de archivos, ir al apartado 3.3 y 3.4 respectivamente.

3.3. Configuración de la sala y posición del array

En la aplicación SoundSource v2.0 se incluye un control visual para simular la colocación de un array en una sala. En la figura 13 se representa la planta de una sala de unas dimensiones introducidas por el usuario en el grupo de controles **Room dimensions**. En este plano es donde se dibujan las hipérbolas durante el proceso de localización. Por defecto, al abrir la aplicación el tamaño de la sala es de 10x6 m.

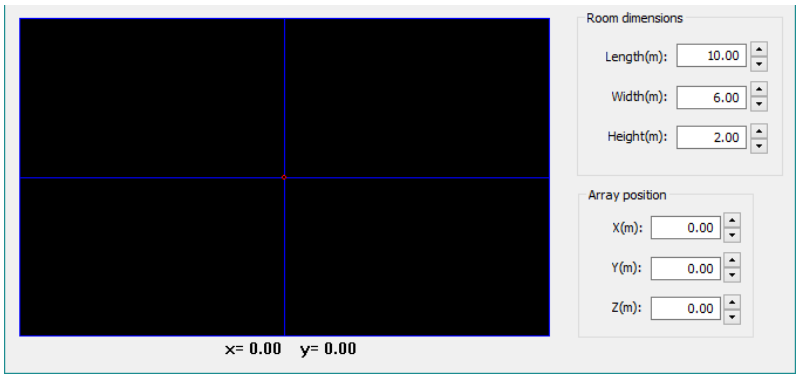

***Figura 13**. Controles visuales de la sala y posición del array.*

En el grupo de controles **Array Position** se selecciona la posición que ocupa en la sala el array de micrófonos configurado en la **Ventana de configuración de entradas**. El centro de la sala se considera el origen de coordenadas, siendo el eje x el paralelo a la longitud de la sala, y el eje y el correspondiente al ancho, tal como se muestra en la figura 14.

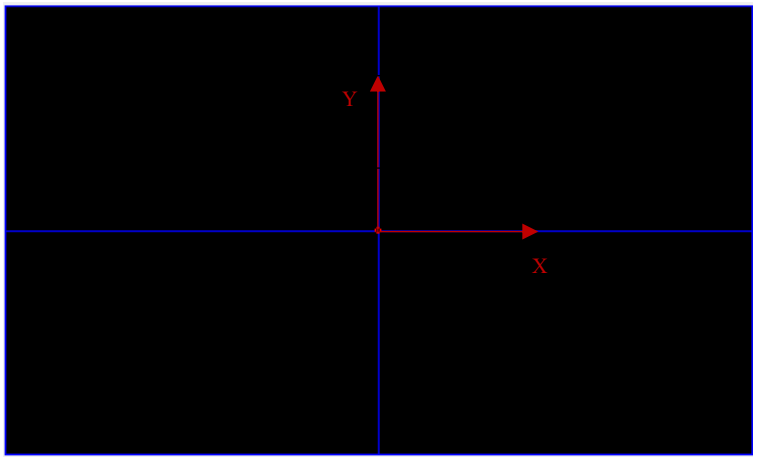

***Figura 14**. Sistema de coordenadas de la sala.*

Debajo del dibujo de la sala se encuentra una cadena de texto en la que aparece el resultado de cada cálculo de la posición, es decir, las coordenadas (x,y) estimadas de la fuente acústica. Cuando el proceso de estimación arroje resultados erróneos, en la cadena de texto figurará como IND (indeterminado).

En la figura 15 se muestra el aspecto de la sala para el ejemplo de la figura 9, en el que un array de cuatro micrófonos situados en los vértices de un cuadrado está situado en el centro de la sala. El centro del array está representado por una pequeña circunferencia roja, mientras que los micrófonos están representados por su número según haya sido configurado.

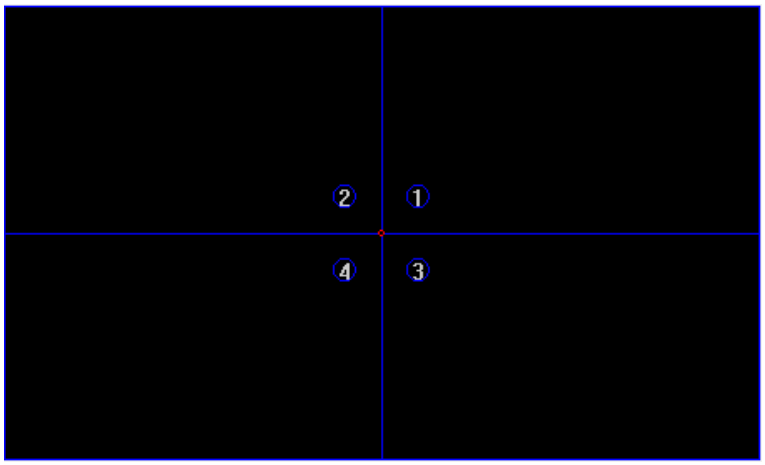

***Figura 15**. Ejemplo de aspecto visual sala con array.*

3.4. Ficheros de salida

Durante la ejecución de SoundSource v2.0 pueden generarse ficheros de salida marcando la casilla **Save** en el grupo de controles **Save in File** (figura 16)**.** Al hacerlo se habilita la selección del directorio de salida. Si no se selecciona ningún directorio, los ficheros se guardarán en el directorio del ejecutable SoundSourcev2.exe.

***Figura 16**. Grupo de controles Save in File.*

Se generan dos tipos de ficheros:

- Un fichero de texto (.txt) con información de la captura de audio (fecha de la realización, frecuencia de muestreo, canales, profundidad de bits, tamaño del buffer y ponderación frecuencial de la GCC) y los resultados de la localización. En la figura 17 se muestra un ejemplo del formato del documento.

- Un fichero de datos en bruto (.raw) que contiene el audio multicanal capturado en el formato seleccionado en la configuración. El orden de los canales es el configurado en la **Ventana de configuración de entradas.** El ordenamiento de bytes es Little-Endian. Este fichero se genera solo en Modo Dispositivo.

```
#Audio Capture - 15/09/2020_18:39:44
#Sampling frequency = 44100 Hz
#Channels = 3
#Bits per sample = 16
#Buffer size = 500.0 ms
#GCC weight = CC

#time offset->localizated point
00:00:0.00->
00:00:0.50->x= 0.00     y= 0.00
00:00:1.00->x= 0.00     y= 0.00
00:00:1.50->x= 0.00     y= 0.00
00:00:2.00->x= 0.00     y= 0.00
00:00:2.50->x= 0.00     y= 0.00
00:00:3.00->x= 0.00     y= 0.00
00:00:3.50->x= 0.00     y= 0.00
00:00:4.00->x= 0.00     y= 0.00
00:00:4.50->x= 0.00     y= 0.00
00:00:5.00->x= 0.00     y= 0.00
00:00:5.50->x= 0.00     y= 0.00
00:00:6.00->x= 0.00     y= 0.00
00:00:6.50->x= 0.00     y= 0.00
00:00:7.00->x= 0.00     y= 0.00
00:00:7.50->x= 0.00     y= 0.00
00:00:8.00->x= 0.00     y= 0.00
00:00:8.50->x= 0.00     y= 0.00
00:00:9.00->x= 0.00     y= 0.00
00:00:9.50->x= 0.00     y= 0.00
```

***Figura 17**. Formato del documento de texto generado.*

En el caso del Modo Dispositivo, cada vez que se pausa el proceso y se vuelve a reanudar se genera un nuevo fichero de texto, ya que se considera que cada proceso de captura es independiente e individual.

El nombre de estos ficheros sigue el patrón **AAAA-MM-DD_HH;mm;ss_bps**bits_**freq**Hz_**n**Ch**,** correspondiendo **AAAA** al año, **MM** al mes, **DD** al día, **HH** a la hora, **mm** a los minutos, **ss** a los segundos, **bps** a la profundidad de bits, **freq** a la frecuencia de muestreo y **n** al número de canales.

4. Referencias

[1] Microsoft, “Descargas más recientes compatibles de Visual C++”, 28 de agosto de 2019. [En línea]. Disponible en: https://support.microsoft.com/es-es/help/2977003/the-latest-supported-visual-c-downloads. [Accedido: 14-sep-2020].

[2] Microsoft, “Visual Studio 2019”. [En línea]. Disponible en: https://visualstudio.microsoft.com/es/vs/. [Accedido: 14-sep-2020].

[3] Microsoft, “Install Visual Studio”, 12 de junio de 2019. [En línea]. Disponible en: https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2019. [Accedido: 14-sep-2020].

[4] FFTW, "The Fastest Fourier Transform in the West". [En línea]. Disponible en: http://www.fftw.org/index.html. [Accedido: 14-sep-2020].

[5] FFTW, “Windows Installation Notes”, *Precompiled FFTW 3.3.5 Windows DLLs*. [En línea]. Disponible en: http://www.fftw.org/install/windows.html. [Accedido: 15-sep-2020].

ANEXO II. Presupuesto del proyecto

Tabla 1. Presupuesto del proyecto.

Concepto	**Precio por unidad**	**Unidades**	**Total**
Esfuerzo humano (horas ingeniero)	18€	600	10800€
Ordenador Acer Aspire 3 A315	600€	1	600€
Acumulado			11400€