UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

Desarrollo en Entorno Matlab de un VST Plug-in de Audio Para Aplicaciones de Sonido Envolvente

Titulación: Grado en Ingeniería en Tecnologías de la Telecomunicación
Autor: Sergio Falcón Castellano
Tutor: Eduardo Hernández Pérez
Fecha: Junio 2017

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

Desarrollo en Entorno Matlab de un VST Plug-in de Audio Para Aplicaciones de Sonido Envolvente

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo.:

Vocal

Fdo.:

Secretario/a

Fdo.:

Fecha: Junio 2017

Índice de contenido

Índice de figuras

Índice de tablas

Índice de ecuaciones

PARTE I MEMORIA

Capítulo 1. Introducción

1.1. Antecedentes

La relación entre la imagen en movimiento y el sonido en el cine no se consagró hasta la aparición del Vitáfono, hasta ese momento el cine sonoro estaba limitado principalmente por los problemas técnicos que ocasionaba la sincronización del sonido con la imagen y las necesidades de amplificación requeridas por el tamaño de las salas.

El Vitáfono es una herramienta desarrollada por la compañía del mismo nombre (Vitaphone), cuyo funcionamiento consistía básicamente en reproducir discos de gramófono sincronizados con la imagen. Una de las primeras películas en hacer uso de un acompañamiento sonoro con dicha herramienta se estrenó en 1927, "El Cantor de Jazz", de Alan Crosland [1]. A esta película se la considera la causante de la extinción del cine mudo y fue la primera en incluir acompañamiento musical y diálogos sonoros. A partir de ese momento es casi imposible separar los términos cine y sonoro.

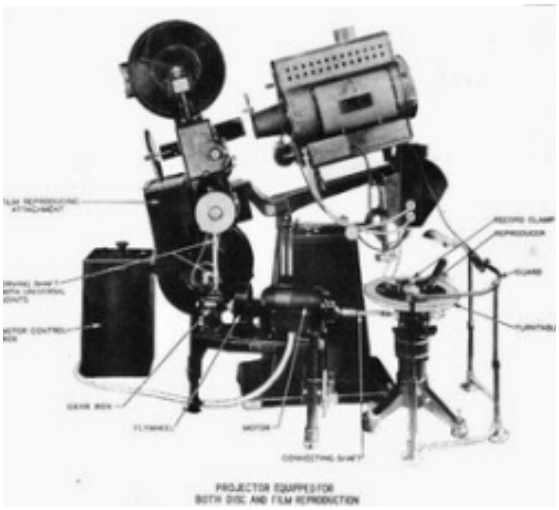


Figura 1: Sistema Vitaphone. [1]

Los sistemas sonoros inmersivos tratan de potenciar las sensaciones sonoras con el uso de un sistema de altavoces, con ellos se recrea la imagen sonora que se representa en pantalla. Su principal objetivo es el de transportar al espectador a un cierto entorno a través de las sensaciones producidas por el sonido.

El sonido envolvente en salas de cine debe su origen a la película de animación “Fantasía” de Disney (1940). Este ambicioso proyecto ideado por Walt Disney y apoyado por Leopold Stokowski, director de la banda sonora de la película, propició que los ingenieros de Disney crearan Fantasound, un sistema de tres canales frontales y dos en las esquinas traseras de la sala (muy similar al sistema 5.1 surround actual), además de un conjunto de técnicas y de herramientas, como la grabación multicanal, necesarias para crear contenido para el mismo. [2]

Figura 2: Cartel de la película Fantasía (1940) [3]

Con el paso de los años, la experiencia sonora en las salas de cine (y en los entornos domésticos) ha evolucionado casi al mismo ritmo que el desarrollo tecnológico, y las experiencias de inmersión sonora forman parte, prácticamente, de cualquier sistema de reproducción multimedia.

1.2. Objetivos

Debido al momento de transición actual, los estudios de grabación todavía combinan el uso de herramientas analógicas y digitales, por lo tanto, existe la necesidad de desarrollar herramientas software que sustituyan con solvencia a los antiguos equipos. Además, el desarrollo de tecnologías como la realidad virtual o los videojuegos de última generación, obliga a los desarrolladores a diseñar sistemas, con funciones muy específicas, para cubrir la gran demanda existente de herramientas destinadas a generar contenido multimedia con sonido envolvente.

El objetivo de este trabajo fin de título es el de desarrollar una herramienta software VST (Virtual Studio Technology), que permita posicionar una muestra monofónica en el espacio virtual bidimensional formado por un conjunto de altavoces.

El uso de esta herramienta permite generar un entorno sonoro envolvente. Para ello, es necesario insertarla en cada uno de los canales que conforman un proyecto de audio multicanal y, a cada uno de ellos, asignarle una localización concreta en el espacio bidimensional. Esta herramienta debe ser, por lo tanto, compatible con aquellos sistemas DAW (Digital Audio Workstation) que soporten procesado de audio multicanal y el propio formato VST.

Se ha decidido desarrollar dicha aplicación en el entorno de programación MATLAB que, en su última versión, incluye “Audio System Toolbox”, un conjunto de herramientas y librerías que permiten la programación y el testeo de aplicaciones de audio. Además, “Audio System Toolbox” es compatible con el codificador C/C++ de Matlab, que permite generar sistemas Plug-in y aplicaciones de audio autónomas.

Los objetivos específicos planteados en el anteproyecto de este trabajo de fin de título son:

- Establecer la metodología genérica a seguir en el desarrollo de un VST-Plug-In en entorno Matlab.
- Especificación y desarrollo de un VST-Plug-In capaz de operar con sistemas de, al menos seis canales de audio (sistemas 5.1).
- El VST-Plug-in desarrollado será compatible con sistemas DAW profesionales.

1.3. Metodología y plan de trabajo

Para cumplir los objetivos planteados en el apartado 1.2 de esta memoria, se ha ejecutado el siguiente plan de trabajo:

- Realizar un análisis de documentación relacionada con los siguientes ámbitos:
    - El método de programación de VST-Plug-in.
    - Tecnologías y técnicas aplicadas a sistemas de sonido envolvente.

    - Documentación del paquete de software “Audio System Toolbox” de Matlab.
- Realizar un estudio del “Audio System Toolbox” de Matlab para poder desarrollar herramientas que hagan uso del mismo. Además, se ha configurado en el laboratorio de sonido el hardware necesario para reproducir contenido con sonido envolvente 7.1.
- Desarrollar un conjunto de prototipos y tests para poner a prueba probar el algoritmo VBAP empleado para programar el software. Una vez desarrollado el Plug-in y probado su funcionamiento y se ha elaborado una demostración en un sistema DAW profesional.

Desde el punto de vista metodológico se ha seguido inicialmente con la aplicación el método analítico. Distinción, estudio y ordenación de los elementos, experimentación y análisis de casos.

1.4. Estructura de la memoria

Éste trabajo de fin de título está dividido en los siguientes apartados: Memoria, Pliego de condiciones, Presupuesto y Anexos.

La memoria se compone de 7 capítulos, el Capítulo 2 presenta las estaciones de trabajo de audio digital, de gran importancia, debido a que serán el entorno en el que se va a ejecutar el programa final. En el capítulo 3 se caracterizan los sistemas de sonido envolvente, en el capítulo 4 se detallan las herramientas necesarias y los procedimientos que se han llevado a cabo para el desarrollo del software objetivo de este proyecto, en el capítulo 5 se muestran las conclusiones que se han extraído durante el desarrollo del trabajo, en el capítulo 6 se presentan las posibles líneas futuras y en el capítulo 7 se incorpora la bibliografía. En la segunda parte del documento se presenta el desglose del presupuesto con los costes que componen el proyecto y en el apartado final se encuentran los anexos que incluyen el código de los diferentes proyectos que se han generado durante el desarrollo de este trabajo de Fin de Grado.

Capítulo 2. Estaciones de trabajo de Audio Digital

2.1. Estaciones de trabajo de audio digital

Las estaciones de trabajo de audio digital, conocidas comúnmente por sus siglas en inglés DAW (Digital Audio Workstation), nacieron con el objetivo de sustituir a los estudios de grabación analógicos tradicionales. Debido al abaratamiento de los ordenadores personales y a su progresivo crecimiento en potencia de cálculo, se convirtieron en la herramienta central de los estudios de grabación de audio. Se considera que las siglas DAW hacen referencia únicamente al software principal de un estudio de grabación, pero éstos programas necesitan un mínimo soporte hardware para poder funcionar correctamente. Es por eso que se considera a las estaciones de trabajo digital cómo un conjunto de elementos hardware y software que permiten la grabación, edición y reproducción de audio.

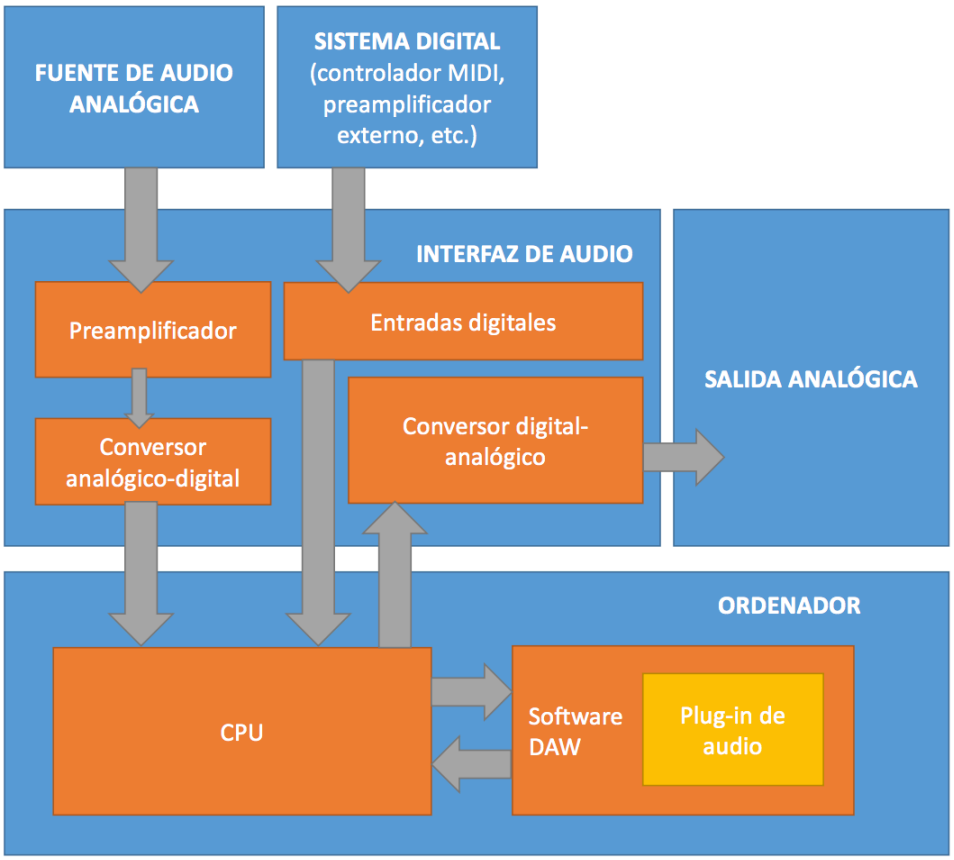


Figura 3: Diagrama DAW

A diferencia de los entornos de trabajo digitales, los equipos que conformaban los estudios de grabación analógicos, eran de grandes dimensiones, eran más complejos de operar y no permitían realizar edición no lineal de audio. La edición lineal es aquella que requiere recorrer toda la pista hasta el punto que se va a editar, equivalente a editar con cinta analógica. En cambio, en los sistemas en los que es posible editar de manera no lineal, se puede "saltar" directamente al fragmento de audio que se va a editar, además, existe la posibilidad de duplicarlo y deshacer los cambios de manera inmediata.

Las DAW están formadas, como mínimo, por los siguientes elementos: equipos que captan las señales analógicas (acústicas o eléctricas), conversores analógico-digitales, unidades de procesamiento de datos, software DAW, conversores digital-analógico y sistemas de escucha (monitores o auriculares). En la Figura 3 se presenta un diagrama con las diferentes partes que componen una DAW.

2.2. Interfaz de audio

La interfaz de audio es el elemento hardware que gestiona las entradas y salidas de la estación de trabajo de audio digital. Las primeras interfaces de audio se conectaban al ordenador a través de Firewire debido a las bajas velocidades que proporcionaba el USB, en la actualidad es común encontrar interfaces que trabajan con USB 2.0, USB 3.0 e incluso con conexiones Thunderbolt.

Las interfaces de audio contienen los siguientes elementos:

- **Entradas de audio:**

  Por lo general, las interfaces de audio poseen preamplificadores de audio para entradas de micrófono XLR, entradas de línea y entradas de instrumento. A su vez, las interfaces de audio que poseen preamplificadores de audio llevan integrados conversores analógico-digitales que convierten la señal eléctrica que llega a su entrada en un conjunto de bits que puede ser interpretado y modificado por el ordenador.

  Las características técnicas de los conversores analógico-digitales que son de interés son las siguientes:

- ***Frecuencia de muestreo***: representa el número de muestras por segundo que el conversor analógico-digital extrae de la señal original para convertirlas en una serie numérica. Tal como indica el teorema de Nyquist, la frecuencia de muestreo debe ser dos veces superior al ancho de banda de la señal a muestrear para evitar que se produzca aliasing, cómo el ancho de banda de la audición humana se extiende aproximadamente desde los 20 Hz hasta los 20000 Hz, la mínima frecuencia de muestreo debe ser de 40000 Hz. La frecuencia de muestro a partir de la cual empiezan a trabajar los conversores analógico-digitales dedicados a audio es de 44100 Hz debido a que es la frecuencia de muestreo del CD, el principal formato de audio digital. Las frecuencias de muestreo típicas en la producción de audio son: 44100 Hz, 48000Hz, 88200Hz, 96000 Hz, 176400 Hz y 192000 Hz.
- ***Profundidad de bits****:* hace referencia al número de bits de cuantificación, es el número de bits necesarios para codificar las diferentes muestras de audio, por lo tanto, a mayor número de bits, mayor número de niveles de cuantificación de la señal analógica. Los valores típicos de profundidad de bits son: 16 y 24 bits.

- **Salidas de audio:** Salidas para monitores y auriculares con sus respectivos controles de ganancia (algunos pueden contener interruptores para atenuar la señal), puede tratarse de salidas XLR o salidas de línea balanceadas.
- **Entradas y salidas digitales**: Permiten la comunicación entre el software DAW y los equipos digitales que componen el estudio de grabación. Las conexiones digitales que podemos encontrar en una interfaz de audio son: conexiones MIDI, S/PDIF, salidas de señal de reloj (Word Clock Out) y conexiones ópticas.

Figura 4: Interfaz de audio Focusrite 18i20

2.3. Software DAW

El software DAW es el elemento que controla el procesado de todos los elementos que conforman la estación de trabajo de audio digital, en él se configuran los parámetros técnicos que controlan la grabación y la reproducción de audio del proyecto (frecuencia de muestreo, profundidad de bit, etc.), se gestionan las entradas y salidas (tanto analógicas cómo digitales) y permite realizar la grabación, la edición y la mezcla de audio.

Hay gran variedad de sistemas software DAW disponibles y cada uno posee cualidades que lo diferencian del resto, pero el flujo de trabajo es prácticamente idéntico en todas ellos. Por lo general, dichos programas se componen de las siguientes ventanas de trabajo:

- **Ventana de edición**: contiene las diferentes pistas que componen el proyecto, las pistas de audio se representan con su respectiva forma de onda y es posible modificarla con las diferentes herramientas disponibles: cortar, copiar, desplazar, fundidos de entrada y salida, etc.

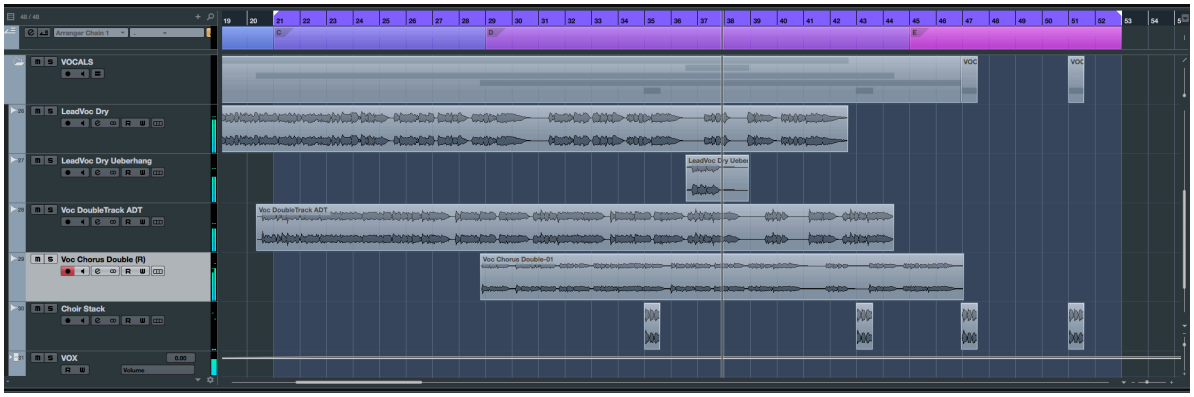


Figura 5: Ventana de edición de Cubase 9

- **Ventana de mezcla**: Trata de imitar a las clásicas mesas de mezcla analógicas, en esta ventana es posible crear buses, controlar los faders, panoramizar la señal, insertar plug-ins, etc.

La ventana de mezcla, a su vez, se compone de los siguientes elementos:

- ***Entradas y salidas***: Las entradas pueden estar enlazadas con pistas del propio proyecto o con entradas de la interfaz de audio. Las salidas de audio permiten enviar la señal al sistema de monitorización o pueden servir de salida para realizar envíos a equipos físicos externos.
- ***Elementos fijos de control***: Elementos fijos que forman parte del canal de la mesa, son: Ecualizador (generalmente paramétrico), control de panorama (puede ser estereofónico o multicanal, dependiendo de la configuración del canal) y el "fader" de control de ganancia de salida.
- ***Insertos:*** permite desviar la ruta de la señal mediante envíos de audio para hacerla pasar a través de módulos de efecto. Pueden ser "pre-fader" o "post-fader", dependiendo de si están controlados por este elemento o no.
- ***Interruptores de control***: los interruptores de "mute", "solo" y "record" también forman parte de las pistas de mezcla.

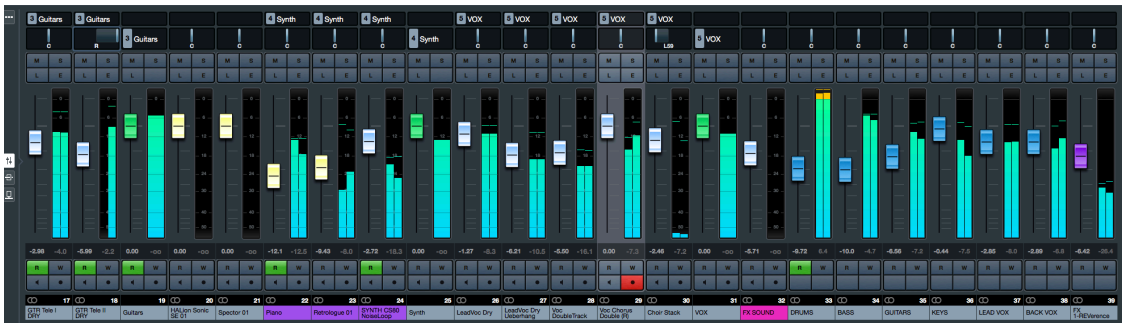


Figura 6: Ventana de mezcla de Cubase 9

- **Ventana de edición MIDI**: en forma de "piano roll" o de editor de partituras, permite recibir señales desde controladores MIDI para actuar sobre instrumentos virtuales, puede enviar señales MIDI a instrumentos externos y también permite "pintar" las notas, moverlas, recortarlas, alargarlas y, en general, modificar la interpretación musical de un instrumento controlado por el protocolo MIDI.

2.4. Plug-in de audio

2.4.1. Steinberg VST

Los plug-ins VST (Virtual Studio Technology), cuya tecnología fue lanzada en 1996 por la empresa Steinberg, forman una interfaz que permite ampliar las funcionalidades del estudio virtual. Su función es la de acoplar unidades de efectos o instrumentos virtuales, realizando cálculos y compartiendo información con el programa principal (host). [4]

Existen Plug-ins VST que modelan el funcionamiento de instrumentos y efectos hardware, que permiten recrear su funcionamiento desde el propio ordenador. Además, gracias a la tecnología VST, se han desarrollado instrumentos y efectos digitales de nueva creación.

Este tipo de tecnología tiene grandes ventajas, permite centralizar todo el procesado de audio en un mismo equipo, dando lugar a un estudio más compacto y accesible. Además, todos los parámetros de control pueden ser automatizados, de esta forma es posible manipular los controles de varios efectos a la vez de forma autónoma.

La tecnología VST se ha convertido en el estándar en Plug-ins de audio y es por eso que son compatibles en la gran mayoría de sistemas DAW profesionales. Steinberg proporciona a los desarrolladores su kit de desarrollo SDK, un conjunto de clases codificadas en C++, para fomentar el desarrollo de este tipo de herramientas.

Hay tres tipos de VST:

- Efectos VST: Son efectos que se añaden a las pistas de audio para modificar algún parámetro de las mismas (amplitud, frecuencia, etc.), los compresores o ecualizadores son algunos ejemplos de efectos VST.
- Instrumentos VSTi: tales cómo los sintetizadores, generan señales y permiten modificar sus parámetros para crear un sonido concreto.
- Efectos MIDI VST: como los arpegiadores, permiten modificar el procesado de los parámetros MIDI.

Capítulo 3. Sistemas de sonido inmersivo

3.1. Introducción a los sistemas sonoros inmersivos

Los sistemas sonoros inmersivos tienen como objetivo dotar de realismo a una escena sonora artificial, para ello deben generar sensaciones en el espectador que potencien el dramatismo de un cierto material audiovisual. Se basan, generalmente, en sistemas de varios altavoces que radian señales de audio desde distintas direcciones.

El sistema de sonido envolvente más extendido es el sistema 5.1 discreto, basado en la norma ITU-R BS.775-1 [5], cuyas aplicaciones cubren ámbitos como la difusión de programas de televisión, cine, videojuegos y artes escénicas, pero existen sistemas que buscan potenciar aún más dicho efecto, como los sistemas “7.1 surround” o el reciente “Dolby Atmos”.

3.2. Jerarquía de sistemas de sonido multicanal compatibles para la radiodifusión y la grabación

A la hora de abordar el diseño de sistemas sonoros inmersivos, hay que tener en cuenta que las necesidades de las salas de proyecciones cinematográficas son diferentes a las de un entorno doméstico, debido principalmente al tamaño de la sala, su tratamiento acústico, tamaño de la pantalla y la distribución de los oyentes.

Además, hay que considerar las limitaciones de los canales de radiodifusión de televisión digital y la posibilidad de que el receptor de dichas señales no disponga de un sistema de altavoces capaz de reproducir correctamente el contenido multimedia.

Por lo tanto, para asegurar la compatibilidad con los sistemas de audio que posean un número de canales inferior, la ITU-R BS.775-3 [5] recomienda seguir la estructura descendente de la Figura 7.

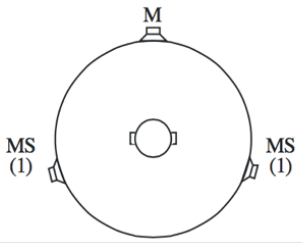
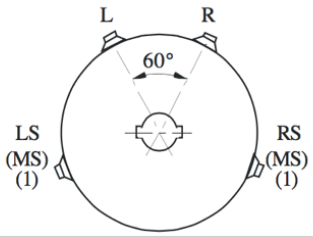
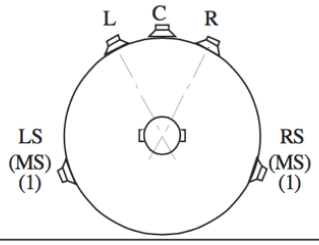
Sistema	Canales	Código	Disposición de los altavoces
Sistema monocanal	M	1/0	
Mono más monopanorámico	M/MS	1/1	
Esteorofónico de dos canales	L/R	2/0	
Esteorofónico de dos canales más 1 canal panorámico	L/R/MS	2/1	
Esteorofónico de dos canales más 2 canales panorámicos	L/R/LS/RS	2/2	
Esteorofónico de tres canales	L/C/R	3/0	
Esteorofónico de tres canales más 1 canal panorámico	L/C/R/MS	3/1	
Esteorofónico de tres canales más 2 canales panorámicos	L/C/R/LS/RS	3/2	

Figura 7: Jerarquía de sistemas de sonido multicanal compatibles para la radiodifusión y la grabación [5]

3.3. Sistema de sonido “5.1 Surround”

La configuración de sonido “5.1 Surround”, también conocido como configuración 3-2, es un sistema estandarizado y popularizado en gran variedad de aplicaciones entre las que se incluyen el cine, la televisión y los sistemas de sonido domésticos.

El estándar que describe la configuración 5.1 indica que éste consta de tres altavoces frontales que forman un estéreo de tres canales y el resto (subwoofer y altavoces laterales) tienen como objetivo añadir efectos, generar cierto tipo de ambiente o están destinados a crear un efecto de reverberación que simule la acústica de un recinto cerrado con unas características determinadas.

El término ".1" hace referencia al canal de efectos de baja frecuencia, conocido como LFE (Low Frecuency Effects) o canal de sub-graves. Este canal es opcional, está limitado en banda y su uso está limitado a la reproducción de registros sonoros de baja frecuencia.

La disposición de los altavoces y la configuración de los canales está especificada en el estándar ITU-R BS.775 [5] e incluye, entre otras, las siguientes recomendaciones:

- La disposición de los altavoces será la siguiente:

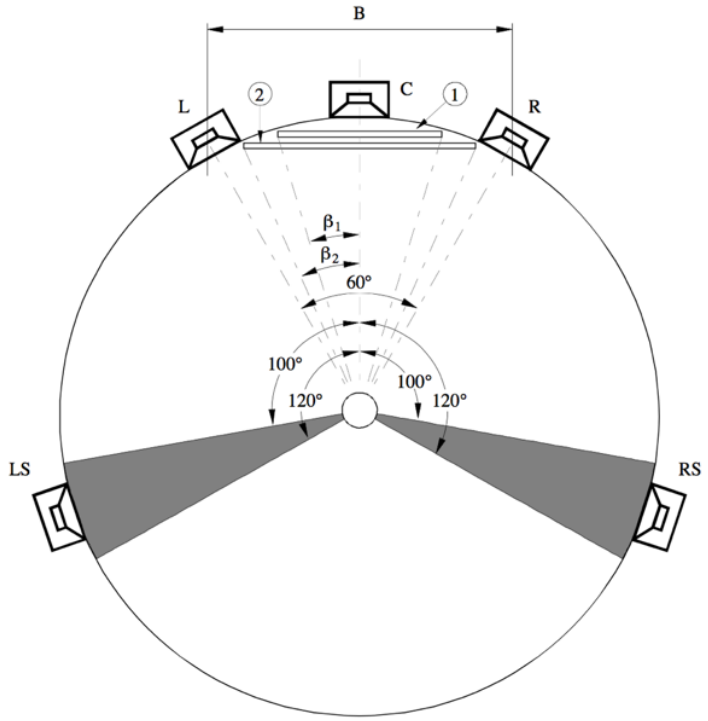


Figura 8: Disposición de los altavoces de un sistema "Surround 5.1" [6]

Donde "B" representa la distancia entre los altavoces.

- Los arcos representados en la Figura 8 se trazan sobre un circulo cuyo centro es el punto de escucha de referencia.
- Cuando, por razones de espacio, sea necesario situar los altavoces frontales en línea recta, es posible que se presente la necesidad de incluir un retardo en la señal que alimenta al altavoz central.

- La posición de los altavoces laterales no tiene que ser precisa, pero si deben situarse en el interior del arco formado por los ángulos 100º y 120º con respecto al eje central. Será necesario que se instalen a la misma altura que los frontales, pero en el caso de que no sea posible, será necesario añadir retardos a los mismos.

- Se utilizan cinco canales para la grabación o la transmisión de los canales de audio. Concretamente: canal izquierdo (L), derecho (R), central (C), panorámico izquierdo (LS), panorámico derecho (RS) y, de forma adicional, el canal de efectos de baja frecuencia (LFE).

3.4. Sistema de sonido "7.1 Surround"

Es el sistema de sonido introducido en Dolby Digital Plus y DTS HD, puede encontrarse en formatos Blu-ray y HD DVD. Consiste en añadir dos canales panorámicos al sistema "5.1 Surround". A pesar de que no existe una recomendación específica para este sistema de sonido multicanal, en la figura 11 de la recomendación ITU-R BS.2159 [7], se muestran un conjunto de posibles configuraciones de altavoces. En la Figura 8, se presentan las distribuciones que presentan todas las fuentes en el mismo plano, hecho que resulta de especial interés en el desarrollo del Plug-in de este trabajo de fin de grado:

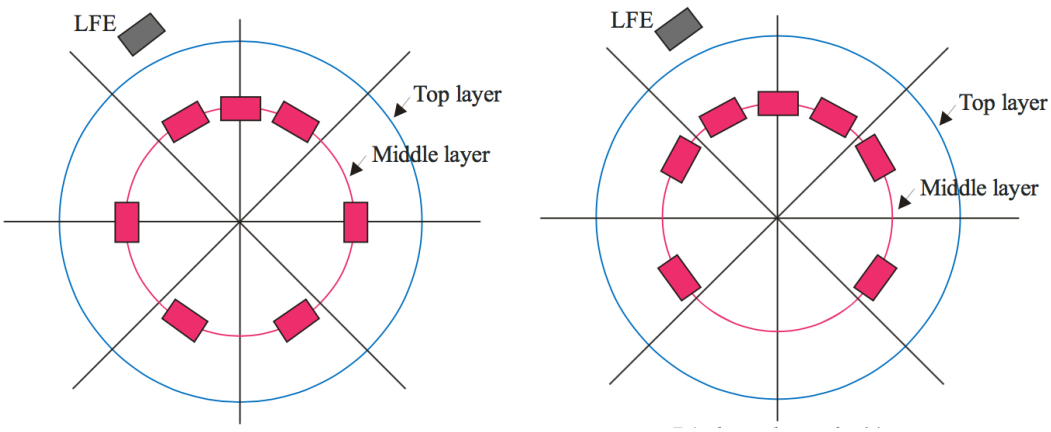


Figura 9: configuraciones del sistema Surround 7.1 recomendadas en la ITU-R BS.2159 [7]

Por su parte, Dolby recomienda la distribución de altavoces de la Figura 11 para los sistemas de sonido envolvente domésticos:

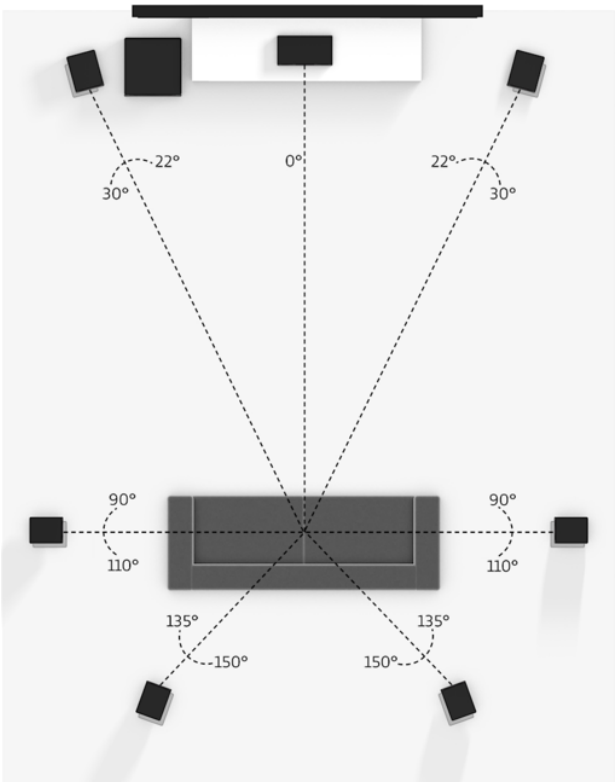


Figura 10: Distribuciones recomendadas por Dolby, Dolby 7.1 Surround (izq.) [8]

3.5. Sistema SDDS

El "SDDS" (Sony Dynamic Digital Sound) es un sistema de sonido envolvente para salas de cine que permite el uso de 8 canales: cinco frontales, dos canales surround y uno de efectos de baja frecuencia. A diferencia del sistema "7.1 Surround", esta distribución de altavoces está pensada para salas con grandes pantallas de cine debido a sus cinco canales frontales (detrás de la pantalla). Esta distribución produce un frente de ondas más uniforme en la dirección frontal debido al número de fuentes sonoras delanteras.

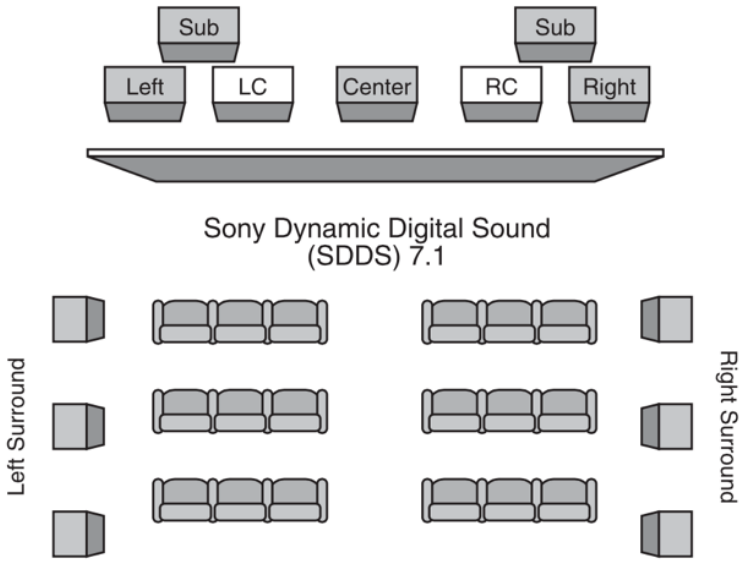


Figura 11: Distribución de los altavoces de un sistema SDDS [9]

3.6. Dolby Atmos

La tecnología “Atmos”, del fabricante Dolby, es el último gran avance en sonido inmersivo. Se trata de un conjunto de tecnologías y principios de diseño destinados a salas de cine y equipos domésticos.

Dolby “Atmos” es un sistema en el que se codifican objetos sonoros en vez de canales. Cada objeto sonoro está asignado a una pista y lleva asociada un cierta metainformación con las coordenadas que lo localizan en el espacio.

“Atmos” soporta hasta 128 objetos sonoros simultáneos y el número de altavoces de la sala depende del tamaño y la distribución de la misma (es capaz de controlar hasta 64 altavoces independientes).

El sistema se integra en un ordenador que, con la metainformación contenida en los objetos sonoros, es capaz de posicionarlos en el espacio con técnicas de panoramización en tres dimensiones. Además, las instalaciones de dolby “Atmos” también son compatibles con el sistema Surround 7.1.

Las principales diferencias que caracterizan a “Atmos” del resto son:

- Se aumenta el número de altavoces instalados en la sala, destaca la Instalación de altavoces en la parte alta de la sala que permiten reproducir sonidos sobre los oyentes. De esta forma, se consigue una mezcla más realista al añadir sonidos que deberían sonar encima de los espectadores de forma natural.
- En los sistemas 5.1 y 7.1 los canales surround alimentaban a un conjunto de altavoces conectados en paralelo. En cambio, en los sistemas “Atmos” se aumenta el número de canales que controlan a los altavoces de manera independiente, esto permite una localización más precisa de los sonidos y produce un aumento del realismo de la ambientación. [10]

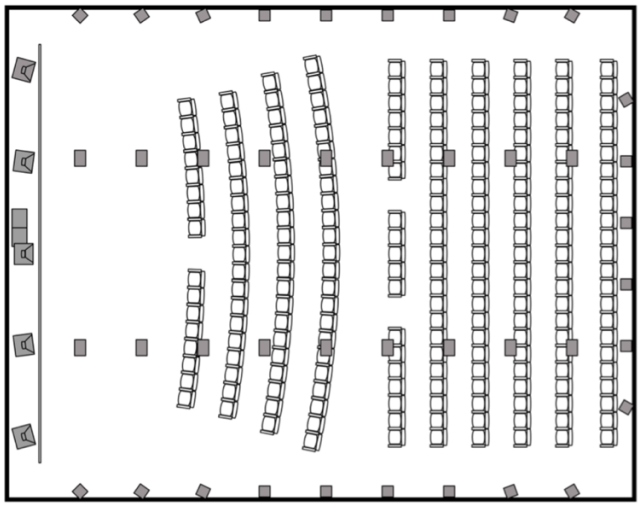


Figura 12: Distribución de altavoces de un sistema Dolby Atmos [10]

3.7. Canal LFE

El canal de efectos de baja frecuencia está destinado a transmitir señales de baja frecuencia y alto nivel. Tal cómo está descrito en la ITU-R BS.775, el canal LFE debe ser capaz de tratar señales en la gama de 20 Hz a 120 Hz. [5] El canal LFE se graba con un desplazamiento de nivel de -10 dB. Ese desplazamiento se compensa en el sistema de reproducción, en el que el altavoz LFE tiene una salida acústica dentro de su banda de paso de baja frecuencia de +10 dB con respecto a los otros canales. [11]

Está orientado principalmente a su uso en cine para reproducir efectos tales como explosiones y vibraciones. Cómo el uso de este canal es opcional, cuando el material audiovisual está destinado a sistemas de audio domésticos se debe reducir la cantidad de señal que alimenta al canal LFE para posibilitar que se pueda reproducir correctamente en ausencia del mismo.

3.7.1. Filtros digitales

Para implementar el canal LFE en un sistema software es necesario filtrar la señal de audio a la entrada, es por eso que se hace necesario realizar esta breve introducción a los filtros digitales.

Un filtro es un sistema que discrimina, enfatiza o atenúa parte de la información presente a su entrada. En concreto, los filtros digitales tienen cómo entrada y salida secuencias discretas, las diferencias que se pueden producir a la salida de dicho filtro pueden ser en amplitud, frecuencia o fase y variarán dependiendo de las características de dicho filtro.

Los filtros digitales son sistemas lineales porque cumplen el principio de superposición. Es decir, si una entrada consiste en la suma ponderada de varias señales, entonces la salida es la superposición de las respuestas del sistema a cada una de esas entradas. Son causales debido a que su salida en cualquier instante de tiempo únicamente depende de los valores de entrada actuales y pasados. Como consecuencia, el hecho de que un filtro sea causal implica que la salida será cero si no se aplica una señal de entrada al sistema. También son invariantes en el tiempo porque su comportamiento y características están fijos en el tiempo. Es decir, el resultado de procesar una señal con un filtro dado, en un instante de tiempo concreto, debe ser el mismo que el filtrado de esa misma señal, en el mismo filtro, pasado un cierto periodo de tiempo. Al ser sistemas lineales e invariantes en el tiempo, se puede realizar una caracterización completa del filtro a partir de su respuesta al impulso. [12]

Por lo tanto, si un filtro es lineal, causal e invariante en el tiempo, puede ser descrito con la ecuación (1).

Ecuación 1: Ecuación de un filtro digital

$$y(n) = \sum_{i=0}^{M} a_i x[n-i] - \sum_{i=1}^{N} b_i y[n-i] \quad (1)$$

Donde a_i y b_i son coeficientes constantes que definen las características del filtro

Si realizamos la transformada z a la ecuación (1) obtenemos:

Ecuación 2: Transformada Z de la ecuación del filtro

$$Y(z) = \sum_{i=0}^{M} a_i z^{-i} X(z) - \sum_{i=1}^{N} b_i z^{-i} Y(z) \rightarrow \quad (2)$$

Ecuación 3: Transformada Z de la ecuación del filtro (2)

$$\rightarrow Y(z)(1 + b_1 z^{-1} + \cdots + b_N z^{-N}) = X(z)(a_0 + a_1 z^{-1} + \cdots + a_M z^{-M}) \rightarrow \quad (3)$$

Y sabiendo que:

Ecuación 4: Función de transferencia general

$$H(z) = \frac{Y(z)}{X(z)} \quad (4)$$

A partir de (3) y (4) obtenemos:

Ecuación 5: función de transferencia de un filtro digital

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_0 + a_1 z^{-1} + \cdots + a_M z^{-M}}{1 + b_1 z^{-1} + \cdots + b_N z^{-N}} = \frac{N(z)}{D(z)} \quad (5)$$

N(z) representa el polinomio numerador y D(z) el polinomio denominador. Cada valor de 'z' que de cómo resultado cero en el polinomio N(z), producirá que H(z) sea igual a cero y, por tanto, equivale a un cero de la función de transferencia. Cada valor de z que de cómo resultado cero en el polinomio D(z), producirá que H(z) sea igual a infinito y equivale a un polo de la función de transferencia.

Los parámetros que especifican la respuesta en frecuencia de un filtro, tales como la ganancia de paso, rizado de la banda de paso, frecuencias de corte, atenuación de la banda de corte, etc. dependen del valor de los coeficientes a_i y b_i. [13]

3.7.2. Filtros FIR

Los filtros FIR (Finite Impulse Response) se caracterizan por estar formados únicamente por ceros y, por lo tanto, únicamente dependerán de valores pasados de la señal de entrada. Como su nombre indica, su respuesta al impulso es finita y está compuesta por 'M' coeficientes que dan lugar a un filtro de orden 'M'.

La respuesta al impulso de un filtro FIR de orden 'M' se describe en la ecuación (6)

Ecuación 6: Respuesta al impulso de un filtro FIR

$$h[n] = \sum_{i=0}^{N} a_i \cdot \delta[n-i] \tag{6}$$

Una de las propiedades que caracterizan a los filtros FIR y que los hacen realmente interesantes es que permiten obtener una respuesta en fase exactamente lineal, esto significa que dichos filtros permiten alterar la amplitud de las diferentes componentes frecuenciales de la señal de entrada sin incorporar distorsión de fase a la señal de entrada, es decir, sin alterar la alineación en el tiempo de las componentes frecuenciales.

3.8. Técnicas de panoramización multicanal

3.8.1. Panoramización de amplitud entre pares de altavoces

Ésta técnica se usa en sistemas estereofónicos, pero su implementación en sistemas multicanal resulta sencilla. Básicamente consiste en enviar una señal monofónica a un par de altavoces, con diferentes amplitudes, para crear una fuente virtual situada entre los mismos. Por tanto, en sistemas multicanal, una muestra monofónica panoramizada mediante esta técnica, únicamente se reproducirá en dos altavoces simultáneamente.

El estudio de esta técnica resulta de especial interés debido a que es la usada en el algoritmo VBAP (Vector Base Amplitude Panning) que se ha implementado en el software desarrollado en este trabajo fin de título. Se describe con detalle en el apartado 4.2. de esta memoria.

3.8.2. Panoramizadores basados en el sistema Ambisonics

El Sistema Ambisonics es capaz de reproducir campos sonoros bidimensionales y tridimensionales, incluyendo la componente de altura y se puede adaptar a un gran número de configuraciones de altavoces.

El sistema ambisonics codifica los sonidos en términos de sus componentes de presión y velocidad. Además, incluye una matriz de amplitudes que determina el nivel de señal que debe enviarse a cada una de las salidas del sistema de altavoces que se ha configurado.

Los panoramizadores basados en Ambisonics tratan de posicionar señales monofónicas en el espacio tridimensional haciendo uso de leyes de panoramización optimizadas psicoacústicamente. Las características más importantes de estas leyes de panoramización son:

- La señal de salida puede excitar a más de dos altavoces del sistema (a diferencia de otros sistemas de panoramización).
- Para posiciones de panoramización concretas, es posible que se radien señales con ganancia negativa (fuera de fase), en algunos canales.
- La separación de los canales es bastante limitada. [14]

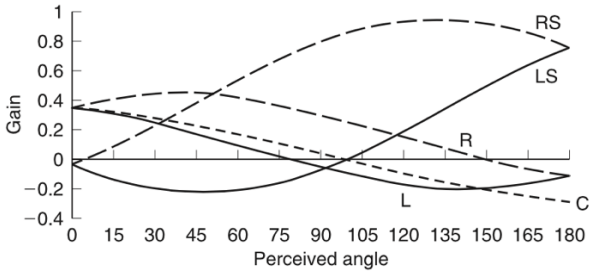


Figura 13: Ley de panoramización de 5 canales basada en principios psicoacústicos de Gerzon [14]

Capítulo 4. Desarrollo de un Plug-in para sonido envolvente

4.1. Implementación del algoritmo VBAP en Matlab

El objetivo de este trabajo fin de título es el de implementar el algoritmo VBAP (Vector Base Amplitude Panning) en un Plug-in de audio para ser utilizado en sistemas DAW profesionales. Para desarrollar esta herramienta se han realizado, de manera consecutiva, los siguientes programas:

- Implementación del algoritmo VBAP estéreo en formato Matlab
- Implementación del algoritmo VBAP estéreo en formato Plug-in
- Implementación del algoritmo VBAP de 5 canales en formato Matlab
- Implementación del algoritmo VBAP de 5 canales en formato Plug-in
- Implementación del algoritmo VBAP de 7 canales en formato Plug-in

En los siguientes apartados se presentará en profundad el algoritmo VBAP, se explicará también su implementación en lenguaje MATLAB y su posterior conversión al formato Plug-in. Para describir el proceso que se ha llevado a cabo en la creación de dicha herramienta, se hará referencia a parte del código de los diferentes proyectos que se han enumerado y que se incluyen en el anexo de este documento.

4.2. Posicionamiento de fuentes virtuales usando VBAP (Vector Base Amplitude Panning)

4.2.1. Panoramización de amplitud en dos dimensiones

Este tipo de panoramización permite crear y posicionar una fuente virtual (fuente imaginaria cuya localización depende de la interpretación que el cerebro realiza de la información sonora que recibe) radiando la misma señal por dos fuentes situadas en posiciones simétricas, respecto a un eje central, con diferentes amplitudes. Éste es el tipo de

panoramización más usado y es típico encontrarlo en mesas de mezclas y sistemas estéreo en general.

El cerebro es capaz de localizar la dirección de una fuente sonora debido a la diferencia de intensidad que llega a sus oídos. Este fenómeno se debe a que la forma de la cabeza se opone al paso de la onda y genera una diferencia interaural de intensidad (IID) [15]. El algoritmo de panoramización VBAP se basa en crear esta sensación emitiendo una muestra monofónica por dos fuentes a la vez, pero con una diferencia de amplitud. De esta forma el cerebro interpretará que la fuente virtual está más próxima a la fuente que presente mayor ganancia.

Puede darse el caso de que la fuente virtual esté en movimiento, en ese caso, es necesario que la sonoridad sea constante, para ello es necesario normalizar los valores de ganancia que se aplican a las fuentes.

Ecuación 7: Constante 'C'

$$g_1^2+g_2^2 = C \tag{7}$$

El valor 'C' controlará la ganancia total del sistema, por lo tanto, con la variación de este parámetro, se puede modificar la percepción de la distancia entre la fuente virtual y el oyente, a mayor nivel, mayor proximidad de la fuente.

Si fijamos el valor de 'C', se puede posicionar la fuente virtual en cualquier punto del arco que forman las dos fuentes físicas, el valor de radio es equivalente a la distancia entre el oyente y las fuentes. Éste arco se denomina arco activo.

4.2.2. Formulación trigonométrica del algoritmo VBAP

La formulación trigonométrica del algoritmo VBAP trata de establecer una relación entre el ángulo de la fuente virtual, percibido por el oyente, y la ganancia aplicada a cada uno de los canales. El resultado de esta relación se conoce como la ley de la tangente y está descrita con la ecuación 8.

Ecuación 8: Ley de la tangente

$$\frac{\tan\varphi}{\tan\varphi_0} = \frac{g_1 - g_2}{g_1 + g_2} \tag{8}$$

Donde: $0º < \varphi_0 < 90º$, $-\varphi_0 < \varphi < \varphi_0$, $g_1, g_2 \in [0,1]$

El proceso de cálculo de ganancias que se presenta a continuación debe cumplir con las ecuaciones (7) y (8) [6].

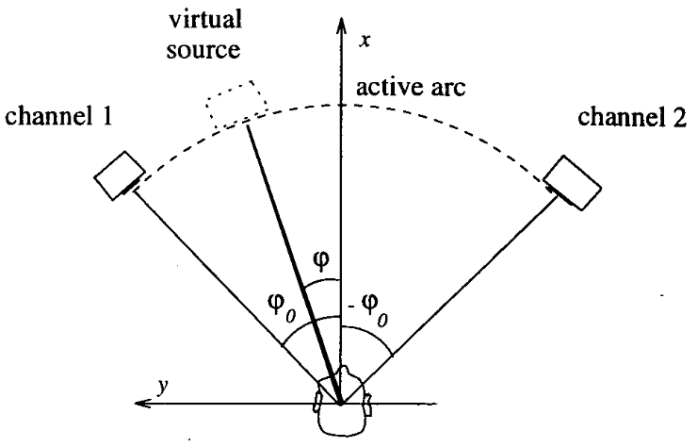


Figura 14: Posicionamiento de una fuente virtual en el arco activo [6]

4.2.3. Formulación con vectores

En esta formulación, la configuración estereofónica VBAP se describe con vectores, es por tanto ideal para implementar las diferentes operaciones necesarias para el desarrollo del algoritmo.

La base se define por los siguientes vectores unitarios:

Ecuación 9: Vector base 1

$$l_1 = [l_{11}\ l_{12}]^T \tag{9}$$

Ecuación 10: Vector base 2

$$l_2 = [l_{21}\ l_{22}]^T \tag{10}$$

Dichos vectores apuntan a los canales 1 y 2 (izquierdo y derecho) respectivamente.

Se define el vector 'p' unitario que apunta a la fuente virtual de la siguiente forma:

Ecuación 11: Vector 'p'

$$p = [p_1\ p_2]^T \tag{11}$$

El vector p se puede considerar como una combinación lineal de los vectores que apuntan a los altavoces, de la siguiente forma:

Ecuación 12: Vector 'p' como combinación lineal de vectores

$$p = g_1 l_1 + g_2 l_2 \tag{12}$$

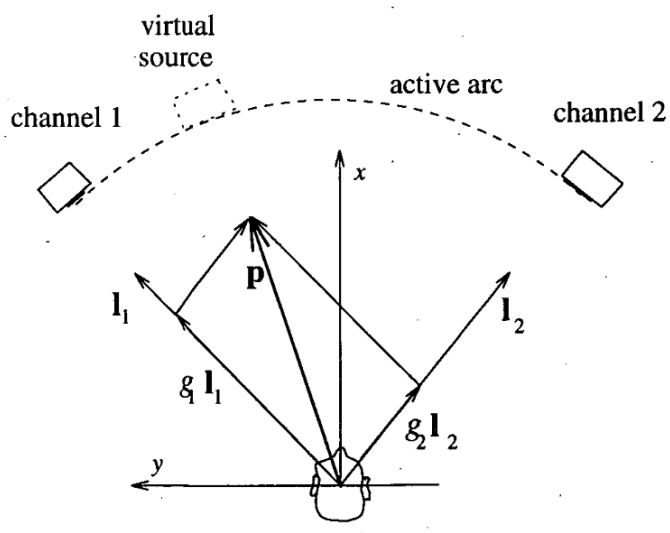


Figura 15: Posicionamiento de la fuente virtual con vectores [6]

Se construye la matriz L_{12}, que está compuesta por las proyecciones en los ejes de los vectores l_1 y l_2 respectivamente.

Ecuación 13: Matriz L12

$$p^T = gL_{12} \tag{13}$$

Donde: $g = [g_1\ g_2]$ *y* $L_{12} = [l_1\ l_2]^T$

Y despejando 'g' obtenemos:

Ecuación 14: Vector de ganancia

$$g = p^T L_{12}^{-1} = [p_1\ p_2] \begin{bmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \end{bmatrix}^{-1} \quad (14)$$

Los factores de ganancia calculados en la ecuación (14) satisfacen la ley de la tangente [6].

Cuando la fuente virtual está en movimiento, hay que normalizar los valores de ganancia, para ello se usa la siguiente ecuación:

Ecuación 15: Vector de ganancia normalizado

$$g^{norm} = \frac{\sqrt{C} \times g}{\sqrt{g_1^2 + g_2^2}} \quad (15)$$

4.3. VBAP en dos dimensiones para más de dos altavoces en el plano horizontal

El algoritmo de panoramización VBAP también es compatible con configuraciones de más de dos altavoces, en este caso la panoramización se realiza en el plano virtual formado por los mismos, pero también se puede diseñar para sistemas tridimensionales.

VBAP es una panoramización por pares, es decir, la muestra monofónica que se desea posicionar en el plano virtual se sitúa en el arco activo formado por uno de los pares de altavoces contiguos que forman el sistema y únicamente se radiará por dichos altavoces. Debido a que existe la posibilidad de saltar de un par de altavoces a otro, es necesario que los valores de ganancia estén normalizados para evitar cambios bruscos de nivel. [6]

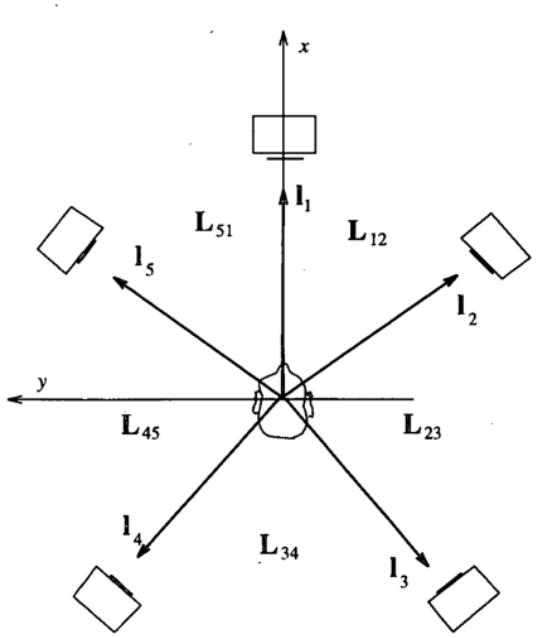


Figura 16: Sectores que componen la panoramización VBAP de 5 canales [6]

4.4. Audio System Toolbox

"Audio System Toolbox" es un conjunto de herramientas y algoritmos incluidos en Matlab R2016a que permiten el diseño, simulación y desarrollo de prototipos de sistemas de procesado de audio.

"Audio System Toolbox" incluye librerías con algoritmos de procesado de audio (filtros, control de rango dinámico, reverberación), de fuentes de audio (osciladores, síntesis mediante tabla de ondas) y para la realización de medidas (curva de ponderación A, curva de ponderación C). También proporciona interfaces para controladores MIDI externos o controladores de audio de baja latencia como ASIO, ALSA o CoreAudio. Además, permite la generación de código C y C++ y también posibilita la generación de herramientas Plug-in VST directamente desde código escrito en MATLAB. [16]

4.5. Diseño de un Plug-in de audio con Audio System Toolbox

A continuación, se presentan los conceptos necesarios para programar un Plug-in de audio en

Matlab con la herramienta “Audio System Toolbox”

Definiciones básicas:

1) “plug-in”: Cualquier Plug-in de audio que deriva de la clase “audioPlugin” o de la clase “audioPluginSource” de Matlab.
2) “basic plugin”: Plug-in de audio que deriva de la clase “audioPlugin” (para procesar señales de audio).
3) “basic source plug-in”: plug-in de audio que deriva de la clase “audioPluginSource” (para crear y procesar señales de audio).

Además, los Plug-in de audio pueden derivar de la clase “matlab.System” que amplían su funcionalidad (integración con Simulink, etc.).

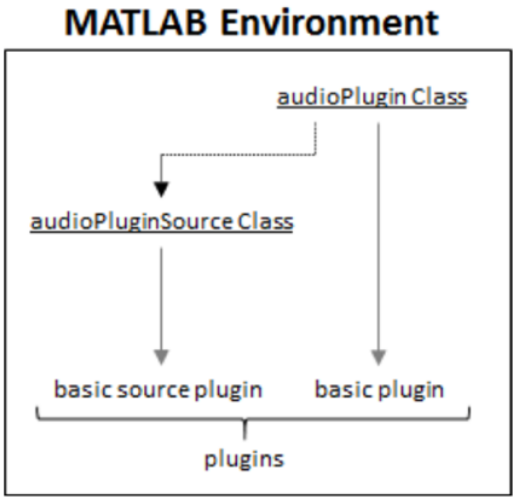


Figura 17: Tipos de audioPlugin Class [16]

4.5.1. Requisitos mínimos de un audio Plug-in

- Derivar de la clase “audioPlugin”.
- Tener un método llamado “process” que contenga el algoritmo de procesado. Por defecto, la entrada y la salida del método “process” está formada por dos canales (columnas). El número de filas que se pasan al método “process” (tamaño del enventanado) está determinado por el entorno en el que se está

ejecutando. La salida debe tener el mismo número de filas que la entrada.

4.5.2. Conceptos para diseñar un "Basic Plugin"

- **Propiedades**: Las propiedades permiten almacenar información en un objeto.
- **Propiedades privadas**: Aquellas que no necesitan ser vistas por el usuario. Es interesante inicializar las propiedades con su tipo y tamaño e inicializar la salida con el mismo tamaño de la entrada en la primera línea de la función process, de esta forma se evitan errores en la creación del Plug-in.
- **Propiedades públicas:** Para permitir que el usuario modifique las propiedades del plug-in es necesario crear las propiedades que se van a ajustar como públicas y asociarlas con un parámetro.
- **Parámetros**: los parámetros del plug-in son la interfaz entre las propiedades del Plug-in y el usuario.
- La interfaz de usuario está controlada por "audioPluginInterface" que contiene objetos del tipo "audioPluginParameter". Para asociar una propiedad a un parámetro hay que introducir un String en el primer argumento de "audioPluginParameter" con el nombre de la propiedad que se quiere asociar.
- Reset: el método reset contiene las instrucciones necesarias para devolver al estado inicial al Plug-in cada vez que se usa o cuando la frecuencia de muestreo cambia.
- "El primer argumento del método process está reservado para el objeto de audio "plugin", si se especifica una variable como primer argumento de process todas las propiedades de myPlugin son accesibles desde el método process".

4.6. Implementación del algoritmo VBAP estéreo en código Matlab

A continuación, se presenta un script de Matlab que implementa el algoritmo VBAP que se ha descrito en el apartado 4.2. En este caso, se importa al entorno de trabajo una muestra de audio monofónica, se introduce por parámetros el ángulo de la fuente virtual, se realiza el proceso de panoramización y el resultado se guarda en un nuevo archivo de audio estéreo.

4.6.1. Descripción de las variables

- La variable “y” representa la señal de entrada que va a ser procesada.
- “Fs” almacena la frecuencia de muestreo del archivo de audio.
- “ang1” y “ang2” son los ángulos de las fuentes físicas y “angV” es el ángulo de la fuente virtual.
- Las variables “L12”, “p”, “g”, “c” siguen la misma nomenclatura que en el apartado 4.2.
- La variable “a” es la señal de salida.

```
[y,Fs] = audioread('SEÑALDEAUDIO.WAV');

ang1 = ((-60)*pi/180);                % -60º en radianes
ang2 = ((60)*pi/180);                 % 60º en radianes

L12 = [cos(ang1), sin(ang1); cos(ang2), sin(ang2)];

angV = input('Teclee el ángulo de la fuente virtual (entre -60 y
60):');
angV = ((angV)*pi/180); % En radianes

p = [cos(angV), sin(angV)];
g = p/L12;

c = 1;
gs = (sqrt(c)/sqrt(g(1)^2+g(2)^2))*g;
a = y*gs;

audiowrite('panD.wav',a, Fs);
```

NOTA: La decisión de darle a la constante “C” el valor 1 está razonada a partir de la ley de panoramización o pan-law, que dice: al duplicar una pista mono para convertirla en señal estéreo es necesario atenuarla 3dB [14]*, o lo que es lo mismo, dividirla por* $\sqrt{2}$*. Teniendo en cuenta que los valores de ganancia* g_1 *y* g_2 *están limitados entre 0 y 1. Al realizar el escalado de esas ganancias en 0º, el punto en el que ambas toman su mayor valor, con la constante c =1, obtenemos lo siguiente:*

En 0º $\rightarrow g_1 = 1$ *y* $g_2 = 1$*:*

$$g_1^{norm} = \frac{\sqrt{c} \times g_1}{\sqrt{g_1^2 + g_2^2}} = \frac{\sqrt{1} \times 1}{\sqrt{1^2 + 1^2}} = \frac{1}{\sqrt{2}}$$

$$g_2^{norm} = \frac{\sqrt{c} \times g_2}{\sqrt{g_1^2 + g_2^2}} = \frac{\sqrt{1} \times g}{\sqrt{1^2 + 1^2}} = \frac{1}{\sqrt{2}}$$

Por lo tanto, al realizar el escalado de la ganancia, automáticamente se limita entre 0 y $\frac{1}{\sqrt{2}}$*, valores de ganancia que respetan la ley de panoramización.*

4.7. Convertir código escrito en Matlab en un Plug-in de audio

Para ejemplificar la conversión de un Script de MATLAB a un Plug-in de audio se utilizará la primera versión del software, el plugin VBAP estéreo descrito en el apartado anterior.

4.7.1. MATLAB Script

Para simplificar su conversión al formato Plug-in el script de Matlab debe tener las siguientes secciones:

- Inicialización de las variables:

En esta sección se inicializan los variables con valores conocidos.

```
%% Sección A: Inicialización de las variables

% Ángulos de las fuentes físicas
ang1 = ((-60)*pi/180); % -60º en radianes
ang2 = ((60)*pi/180);  % 60º en radianes
angV = 0;
angV = ((angV)*pi/180); % En radianes

L12 = [cos(ang1), sin(ang1); cos(ang2), sin(ang2)];

fileInfo = audioinfo('ARCHIVO_DE_AUDIO.WAV');
sampleRate = fileInfo.SampleRate;

frameSize = 256;
```

- Construcción de los objetos:

- audioOscillator System objects: Para crear señales variantes en el tiempo.
- dsp.AudioFileReader System object: Para leer un archivo de audio
- audioDeviceWriter System object: Para escribir una señal de audio en el dispositivo de audio por defecto.

```
%% Sección B: Construcción de los objetos

fileReader = dsp.AudioFileReader('Filename',...
fileInfo.Filename,'SamplesPerFrame',frameSize);
deviceWriter = audioDeviceWriter('SampleRate',...
fileReader.SampleRate);
```

- Audio stream loop:

"En este punto se programa el procesado que se va a realizar a la señal de entrada".

```
%% Sección C: Audio Stream Loop, en éste apartado se especifica
% todo el procesado que se realiza a la señal de audio

while ~isDone(fileReader)

% Se debe leer trama a trama la señal de audio (en este caso    % desde
el archivo)

    in = step(fileReader);

    p = [cos(angV), sin(angV)];
    g = p/L12;
    c = 1;
    gs = (sqrt(c)/sqrt(g(1)^2+g(2)^2))*g;


% Una vez se ha procesado la señal "in", se crea la matriz de
% salida de audio "out".

    out = in*gs;

% Para finalizar se envía dicha matriz a la interfaz de audio    % para
su reproducción

    play(deviceWriter,out);
end
```

4.7.2. Convertir un Script de MATLAB en una "Plugin Class"

Se recomienda hacer la conversión entre un Script de Matlab en una "Plugin Class" siguiendo

los siguientes 6 pasos:

1) Crear el “esqueleto” de una Audio Plugin Class

Resulta útil empezar planteando un esqueleto básico con las principales secciones que debe incluir una Audio Plugin Class.

```
classdef pluginClass < audioPlugin

    properties
    end

    properties (Access = private)
    end

    properties (Constant)
    PluginInterface = audioPluginInterface(...);
    end

    methods
        function myPlugin = pluginClass
        end

        function out = process(myPlugin, in)
        out = [L,R];
        end

        function reset(myPlugin)
        end
    end

end
```

2) Plantear la inicialización de las variables del Script y las propiedades del Plug-in.

Las propiedades permiten almacenar información en las diferentes secciones del Plug-in. Si el acceso a la propiedad es privado, el usuario no podrá acceder al mismo. La inicialización de las variables que se hacía en el script corresponde con las propiedades del Plug-in.

```
properties
        angV = 0;
    end

    properties (Access = private)
        ang1;
        ang2;
    end

    properties (Constant)
        PluginInterface = audioPluginInterface(audioPluginParameter(...
            'angV', 'DisplayName', 'Panoramización', 'Mapping',
{'lin',-60,60}), 'InputChannels',1, 'OutputChannels', 2);
    end
```

Los objetos usados por un plugin deben ser declarados como propiedades para ser usados en diferentes secciones del Plug-in. El método constructor del plugin realiza la construcción de dicho objeto.

3) Plantear la construcción de los objetos del Script y plantear el constructor del método Plugin.

Añadir un constructor para el método Plugin. El constructor de los métodos de un plugin tiene la siguiente forma:

```
methods
        % Constructor
        function myPlugin = pluginStereoVBAP2
            myPlugin.ang1 = ((-60)*pi/180);
            myPlugin.ang2 = ((60)*pi/180);
        end
```

Si el plug-in usa objetos, es necesario construirlos a la hora de construir el propio plug-in. También es necesario establecer las propiedades invariables de los objetos usados por el plugin durante la construcción.

4) Añadir el método reset

El método reset se llama cada vez que se inicia una nueva sesión o cuando cambia la frecuencia de muestreo del entorno.

```
function reset(myPlugin)
end
```

5) Plantear el Stream Loop del Script en el método process del plugin.

Debemos volcar el contenido del Stream Loop del script en el método process pero teniendo en cuenta las siguientes diferencias:

1. Un Plug-in de audio valido debe aceptar tamaños de enventanado variables, por lo tanto, el tamaño de ventana debe calcularse cada vez que se llame al método "process". Como el tamaño de ventana es variable, cualquier proceso que dependa del tamaño de ventana, debe ser actualizado cuando el tamaño de la ventana de la señal de entrada cambie.
2. El entorno debe controlar la entrada y la salida del método "process".

```
function out = process(myPlugin, in)

            angulo = myPlugin.angV;
            angulo = ((angulo)*pi/180);

            L12 = [cos(myPlugin.ang1), sin(myPlugin.ang1);...
cos(myPlugin.ang2), sin(myPlugin.ang2)];
            p = [cos(angulo), sin(angulo)];
            g = p/L12;
            c = 1;
            gs = (sqrt(c)/sqrt(g(1)^2+g(2)^2))*g;
            Lg = gs(1);
            Rg = gs(2);
            L = in*Lg;
            R = in*Rg;
             out = [L,R];
        end
```

6) Añadir una interfaz al Plug-in

La interfaz del Plug-in permite a los usuarios cambiar las propiedades del Plug-in. Es necesario especificar "PluginInterface" como un objeto "audioPluginInterface" que contenga un objeto "audioPluginParameter".

El código debe escribirse de tal forma que durante cada llamada al método process, los objetos se actualicen con el valor que toma el parámetro en cuestión.

```
    properties (Constant)
        PluginInterface = audioPluginInterface(audioPluginParameter(...
            'angV', 'DisplayName', 'Panoramización', 'Mapping',
{'lin',-60,60}), 'InputChannels',1, 'OutputChannels', 2);
    end
```

Al ejecutar el Plug-in en el "Audio Test Bench", la interfaz gráfica se construirá a partir de los "audioPluginParameter" que hemos creado en el "audioPluginInterface". En este caso, el resultado es el siguiente:

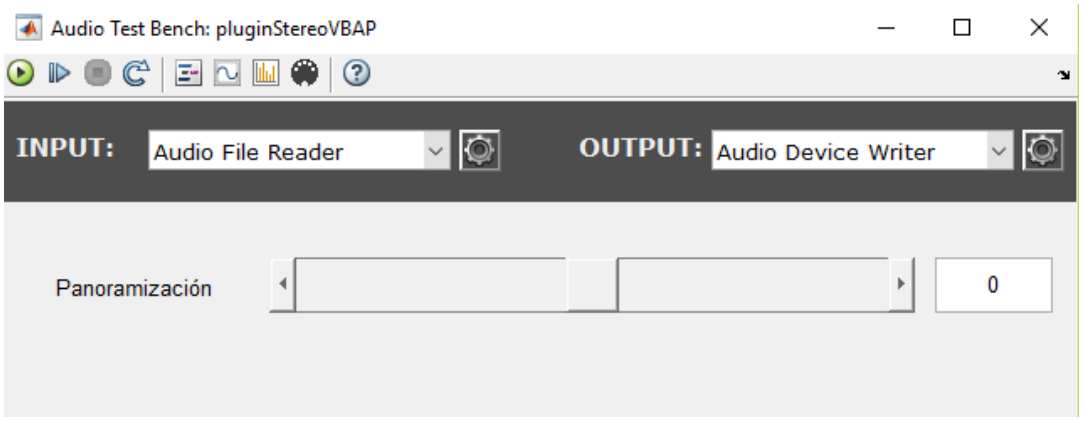


Figura 18: Audio Test Bench del plugin Stereo VBAP

7) Plugin development workflow

Una vez que se ha definido la clase plugin será necesario realizar el proceso conocido como "Plugin development workflow", que consta de los siguientes pasos:

- Guardar el archivo "plugin class definition".
- Validar el Plug-in usando el método "validateAudioPlugin".
- Realizar las pruebas necesarias del prototipo usando el "Audio Test Bench".
- Generar el Plug-in usando el método "generateAudioPlugin". [16]

4.8. Implementación del algoritmo VBAP de 7 canales en formato Plug-in

En este apartado se presenta la versión final del software que se ha desarrollado para este trabajo de fin de grado. Se trata de un programa de Matlab en formato Plug-in que permite realizar una panoramización VBAP entre 7 canales. Además, se ha añadido un módulo que filtra la señal para enviarla al canal LFE. El Plug-in tiene almacenadas las distribuciones de altavoces "7.1 Surround", "Dolby" y "SDDS" para que el usuario pueda seleccionar la configuración de trabajo.

Las asignaciones de los canales de salida que se han programado para cada una de las distribuciones de altavoces se presentan en la Tabla 1 y los sectores posibles entre fuentes se representan en la Figura 20.

Tabla 1: Asignaciones de los canales de salida para pluginVBAP7Canales

Distribución	Canal 1	Canal 2	Canal 3	Canal 4	Canal 5	Canal 6	Canal 7	Canal 8
7.1 (3/4.1)	L	R	C	LS	RS	LFE	LM	RM
Dolby	L	R	C	LS	RS	LFE	LM	RM
SDDS	LC	RC	C	LS	RS	LFE	L	R

Los ángulos físicos que se han programado para las distintas distribuciones de altavoces se presentan en la tabla 2:

Tabla 2: Ángulos físicos para las distribuciones de altavoces de pluginVBAP7Canales

Distribución	Canal 1	Canal 2	Canal 3	Canal 4	Canal 5	Canal 6	Canal 7	Canal 8
7.1 (3/4.1)	-30º	30º	0º	-110º	110º	-	-70º	70º
Dolby	-26º	26º	0º	-135º	135º	-	-100º	100º
SDDS	-22,5º	22,5º	0º	-135º	135º	-	-45º	45º

4.8.1. Funcionamiento

Se trata de un Plug-in con una entrada y ocho salidas, básicamente, su funcionamiento se basa en enviar la señal panoramizada a dos canales del conjunto de salidas que componen el sistema. Además, permite enviar la señal filtrada al canal LFE si el usuario lo considera.

Se debe colocar como inserto en un canal monofónico de audio, y a su salida se obtendrán 8 canales que se deben enviar al canal máster Surround de la DAW. Los filtros programados en éste Plug-in únicamente son compatibles con frecuencias de muestreo 44100 y 48000 Hz, en otro caso el filtro devuelve la señal sin filtrar.

4.8.2. Descripción de las variables

- **AngV:** Angulo de la fuente virtual, accesible por el usuario, está conectado con el parámetro "Panoramización", que permite seleccionar valores entre 0º y 359º con una barra de desplazamiento (valor por defecto: 0).
- **Ang1, Ang2, Ang3, Ang4, Ang5, Ang6, Ang7:** Ángulos de las fuentes físicas, están conectados con el parámetro "Formato", es accesible por el usuario, pero sólo a través del menú desplegable que permite elegir una distribución de altavoces predefinida.
- **Formato:** almacena el valor seleccionado por el usuario en el menú desplegable (valor por defecto: Standard), en concreto:
  - *Standard: hace referencia al formato 7.1, la distribución de altavoces es igual que el sistema 5.1 pero añadiendo dos altavoces más en 70º y en -70º.*
  - *Dolby: hace referencia al formato 7.1 surround recomendado por Dolby.*
  - *SDDS: Sistema con 5 canales frontales y dos surround.*
- **LFE:** variable booleana, si toma el valor "true", permite el paso de la señal por el filtro LFE, si toma el valor "false" se manda la señal "ceros" al canal del subwoofer (valor por defecto: "false").
- **Ganancia_LFE:** valor de ganancia seleccionado por el usuario para el canal LFE (valor por defecto -40 dB).

4.8.3. Descripción de las funciones

- **pluginVBAP7Canales**: Es el constructor de la clase, crea el objeto “myPlugin” y se inicializan las variables con sus valores por defecto.
- **out = process(myPlugin,in**): En éste método se realiza el procesado de la señal de audio, como parámetros de entrada tiene el objeto “myPlugin” y la señal “in” que se trata de un fragmento de la señal de audio que la DAW envía a través del Plug-in (su tamaño dependerá de la configuración del proyecto en la DAW). Como resultado obtenemos una matriz con tantas columnas cómo canales de salida se manden a la interfaz de audio, en este caso ocho. Se seguirá ejecutando el método process en bucle mientras sigan llegándole tramas de audio ‘in’.
- **reset(myPlugin):** En el caso de que sea necesario, devuelve el estado del Plug-in a sus valores iniciales entre cada muestra de audio que procesa (al no haber cambios de tamaño de ventana ni de frecuencia de muestreo no es necesario programar este método en el proyecto).
- **[a] = norvec(myPlugin,g,in):** Ésta función se llama desde el método process(), normaliza el vector de ganancia para que no haya saltos perceptibles al pasar de un sector a otro cuando la fuente virtual está en movimiento. Sus parámetros de entrada son: el objeto “myPlugin”, en el que se almacena la constante “c”, el vector de ganancia “g” y la señal de entrada “in”. Como variable de salida obtenemos el vector “a”, de dos columnas y tantas filas como elementos forman el vector “in”. Cada columna representa un canal del arco activo “L” y “R” respectivamente y están formadas por la señal de entrada tras ser multiplicada por la ganancia normalizada que corresponda.
- **[LFE_signal] = filtrar(myPlugin, in):** filtra la señal de entrada, dependiendo de los valores de frecuencia de muestreo y de frecuencia de corte.

  Los parámetros de entrada son: el objeto “myPlugin”, del que obtenemos el valor de la variable “desplegableFrecuenciaCorte”, la frecuencia de muestreo del sistema y la señal de entrada “in” que va a ser filtrada. Como variable de salida obtenemos “LFE_signal” que es la señal de entrada filtrada.

4.8.4. Diagrama de flujo del método process:

NOTA: Los procesos que se realizan en métodos externos se identifican con los cuadros de color naranja

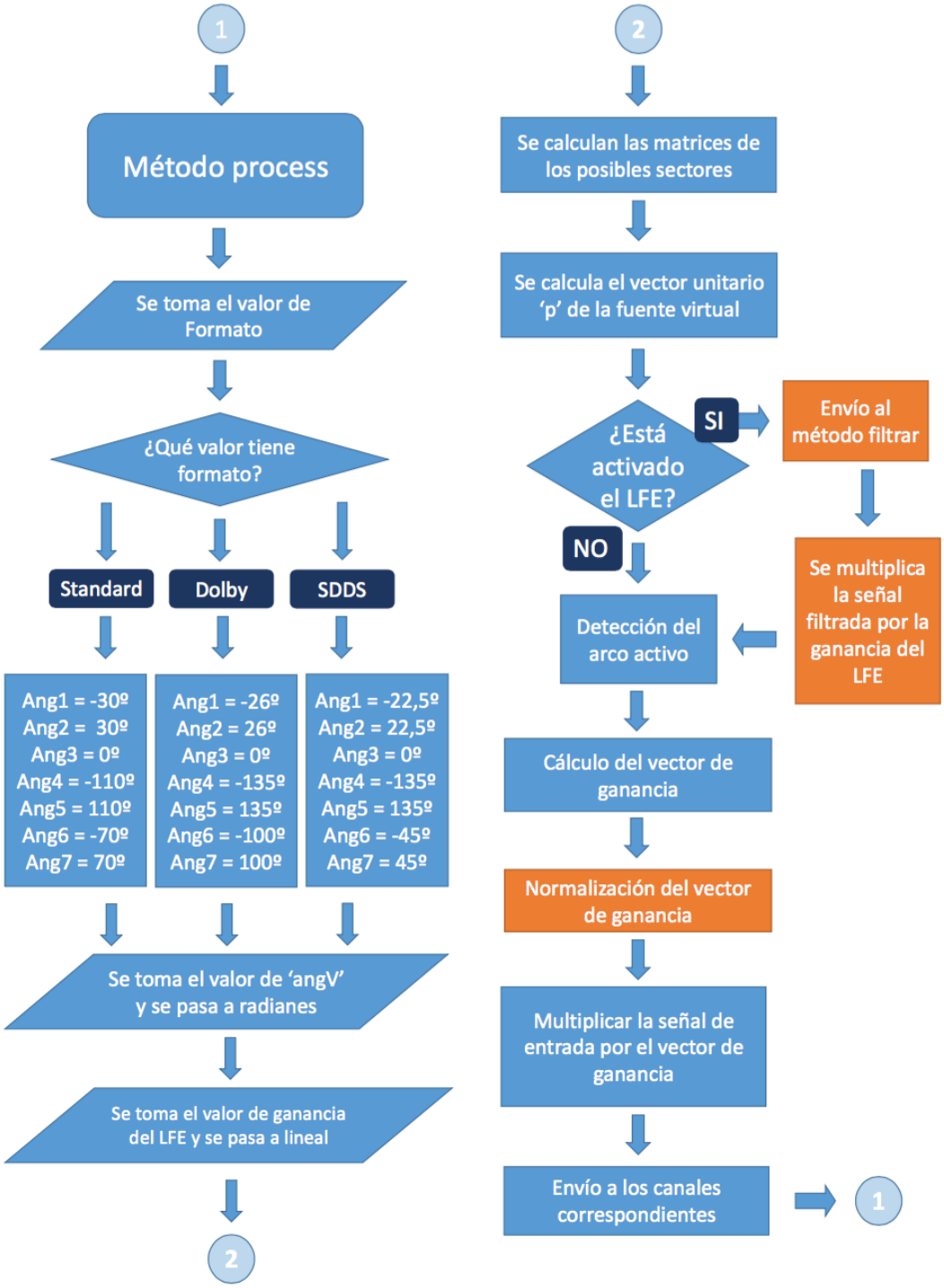


Figura 19: Diagrama de flujo de pluginVBAP7Canales

4.8.5. Descripción del código del proyecto pluginVBAP7Canales:

Se define la clase del Plug-in, pluginVBAP7Canales, que deriva de la clase audioPlugin de MATLAB.

```
classdef pluginVBAP7Canales < audioPlugin
```

Se declaran las propiedades y se inicializan con sus valores por defecto.

```
     properties
        angV = 0;
        ang1 = ((330)*pi/180);
        ang2 = ((30)*pi/180);
        ang3 = ((0)*pi/180);
        ang4 = ((250)*pi/180);
        ang5 = ((110)*pi/180);
        ang6 = ((290)*pi/180);
        ang7 = ((70)*pi/180);
        Formato = desplegableFormato.Standard;
        Fc = desplegableFrecuenciaCorte.Cientoveinte;
        LFE = false;
        Ganancia_LFE = -40;

    end

    properties (Access = private)
        c = 1;
    end
```

Se crea la interfaz gráfica añadiendo elementos “audioPluginParameter”, asociando cada uno de ellos con su correspondiente propiedad:

```
PluginInterface = audioPluginInterface(audioPluginParameter(...
            'Formato','DisplayName','DisposiciÛn','Mapping',{'enum','7.1
(3/4.1) Surround Format','7.1 SDDS Surround Format'}),...
            audioPluginParameter('angV', 'DisplayName', 'Panoramización',
'Mapping', {'lin',0,359}),...
            audioPluginParameter('LFE', 'Mapping', {'enum','Block
signal','Pass through'}),...
            audioPluginParameter('Fc','DisplayName','Frecuencia de
corte','Mapping',{'enum','120 Hz','100 Hz','80 Hz'}),...
            audioPluginParameter('Ganancia_LFE','Label',
'dB','Mapping',{'pow', 1/3, -40, 3}),...
            'InputChannels',1, 'OutputChannels', 8);
    end
```

Se construye el objeto myPlugin y se ejecuta el método process.

```
   function myPlugin = pluginVBAP7Canales
        end

   function out = process(myPlugin, in)
```

MÉTODO PROCESS

Se asignan los valores a las fuentes físicas dependiendo del valor de "desplegableFormato", esta función es básicamente un listado de elementos que formarán parte de un menú desplegable.

```
classdef desplegableFormato < int8
    enumeration
        Standard  (0)
        Dolby     (1)
        SDDS      (2)
    end
end
```

Se toma el valor del ángulo virtual (ángulo en el que el usuario desea posicionar la fuente virtual), se pasa a radianes, se toma el valor de ganancia y se pasa a lineal.

```
angulo = myPlugin.angV;
angulo = ((angulo)*pi/180);
gananciaLFE = 10^(myPlugin.Ganancia_LFE/20);
```

Se crean las matrices de los diferentes sectores posibles. Representados en la Figura 20.

```
L32 = [cos(myPlugin.ang3), sin(myPlugin.ang3); cos(myPlugin.ang2),
sin(myPlugin.ang2)];
L27 = [cos(myPlugin.ang2), sin(myPlugin.ang2); cos(myPlugin.ang7),
sin(myPlugin.ang7)];
L75 = [cos(myPlugin.ang7), sin(myPlugin.ang7); cos(myPlugin.ang5),
sin(myPlugin.ang5)];
L54 = [cos(myPlugin.ang5), sin(myPlugin.ang5); cos(myPlugin.ang4),
sin(myPlugin.ang4)];
L46 = [cos(myPlugin.ang4), sin(myPlugin.ang4); cos(myPlugin.ang6),
sin(myPlugin.ang6)];
L61 = [cos(myPlugin.ang6), sin(myPlugin.ang6); cos(myPlugin.ang1),
sin(myPlugin.ang1)];
L13 = [cos(myPlugin.ang1), sin(myPlugin.ang1); cos(myPlugin.ang3),
sin(myPlugin.ang3)];
```

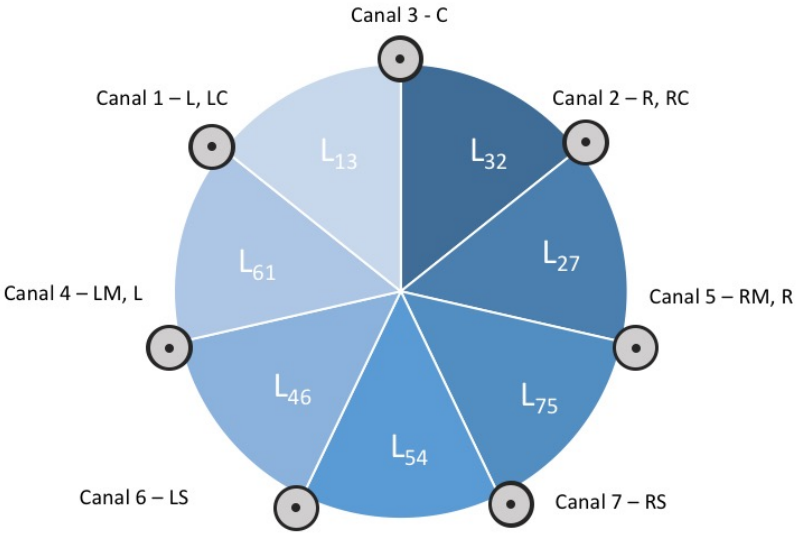


Figura 20: Sectores del arco sonoro

Cada una de estas matrices representan un de los arcos activos correspondientes a los sectores en los que es posible posicionar a la fuente virtual.

Se crea el vector “p” que es el vector unitario en la dirección y sentido de la fuente virtual.

```
p = [cos(angulo), sin(angulo)];
```

Se crea el vector “ceros”, un vector con el mismo número de elementos que la señal de entrada, pero formado por ceros. Éste se enviará a los canales que no formen parte del arco activo.

```
ceros = in*0;
```

Comprobamos si el LFE está activado, en caso afirmativo filtramos la señal con el método filtrar:

```
if myPlugin.LFE == true
               LFE_signal = filtrar(myPlugin, in);
               canal_LFE = LFE_signal*gananciaLFE;
            else
               canal_LFE = ceros;
            end
```

El método filtrar, a su vez, llama a una de las funciones en las que se han programado los filtros. La elección de dicho filtro depende del valor de frecuencia de corte que el usuario ha seleccionado en la interfaz gráfica y de la frecuencia de muestreo de trabajo (este parámetro depende de la configuración del DAW y lo obtenemos con la función "getSampleRate").

Los filtros almacenados en el proyecto están programados de manera similar, únicamente difieren en los parámetros que definen sus características. A continuación, se presenta un ejemplo de filtro FIR programado para filtrar la señal del canal LFE:

```
function y = LFE_120_44100(x)

persistent FILTER_LFE_120_44100;

if isempty(FILTER_LFE_120_44100)
    n = 300;
    fc = 120;
    fs = 44100;

    FILTER_LFE_120_44100 = fir1(n,fc/(fs/2),'low');

end

y = filter(FILTER_LFE_120_44100,1,x);
```

"FILTER_LFE_120_44100" es un vector en el que se almacenan los coeficientes del filtro calculado por la función fir1(), de Matlab, los parámetros de entrada a dicha función son:

- "**n":** orden del filtro (número de coeficientes que componen la función de transferencia del filtro FIR).
- **"fc/(fs/2)":** frecuencia de corte normalizada.
- "**low":** tipo de filtro, en este caso, paso bajo.

El uso de variables persistentes se debe a que éstas se almacenan en memoria una vez calculadas y, cuando el programa vuelva a acceder a la función del filtro no tendrá que volver a calcular sus coeficientes. Esto produce que el programa únicamente haga el cálculo de los coeficientes del filtro una vez por uso, reduciendo el tiempo de procesado.

La respuesta en frecuencia de este filtro en concreto es el que se muestra en la Figura 21.

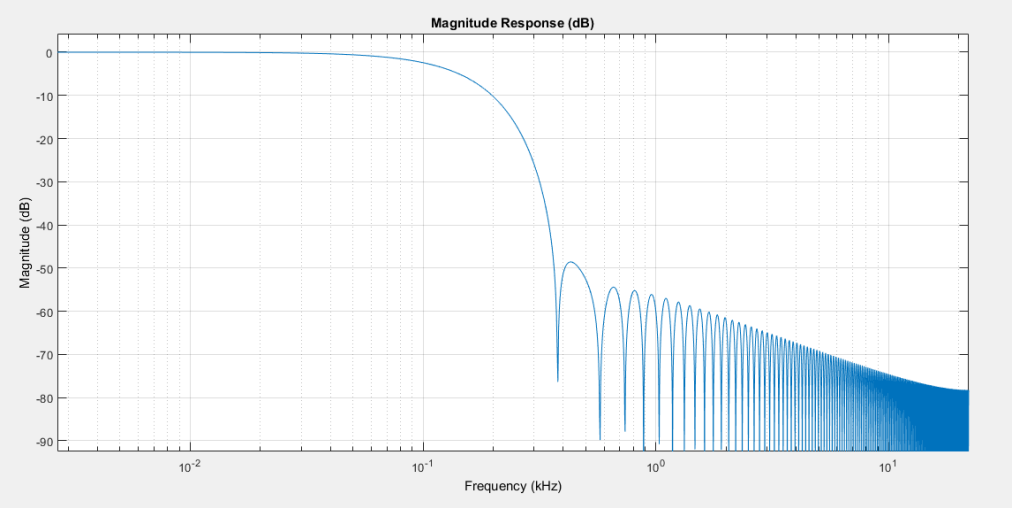


Figura 21: Respuesta en frecuencia del filtro LFE (120 Hz de frecuencia de corte)

A continuación, se calcula el sector en el que se encuentra la fuente virtual, se calcula su vector de ganancia, se normaliza y se envía a cada una de las salidas la señal que corresponda.

Por ejemplo, para el primer sector, el código sería el siguiente:

```
if ((angulo >= myPlugin.ang3) && (angulo < myPlugin.ang2))
                g = p/L32;
                a = norvec(myPlugin,g,in);
                L = a(:,1);
                R = a(:,2);
                canal_1 = ceros;
                canal_2 = R;
                canal_3 = L;
                canal_4 = ceros;
                canal_5 = ceros;
                canal_6 = ceros;
                canal_7 = ceros;
```

Por último, se crea la matriz de salida que se enviará a la interfaz de audio:

```
out = [canal_1,canal_2,canal_3,canal_4,canal_5,canal_LFE,canal_6,canal_7];
```

Una vez enviada la señal procesada a la matriz de salida se toma la siguiente trama de audio y vuelve a ejecutarse el método process.

4.8.6. Pruebas realizadas al Plug-in pluginVBAP7Canales

Una vez se ha codificado el Plug-in se procede a ejecutarlo en el entorno “Audio Test Bench” para realizar las pruebas necesarias para demostrar su funcionamiento.

El “Audio Test Bench” ofrece una interfaz gráfica que dependerá de los “Audio Plugin Parameter” que componen el proyecto, dependiendo del tipo de parámetro la apariencia será diferente. En este caso, la interfaz gráfica resultante es la siguiente:

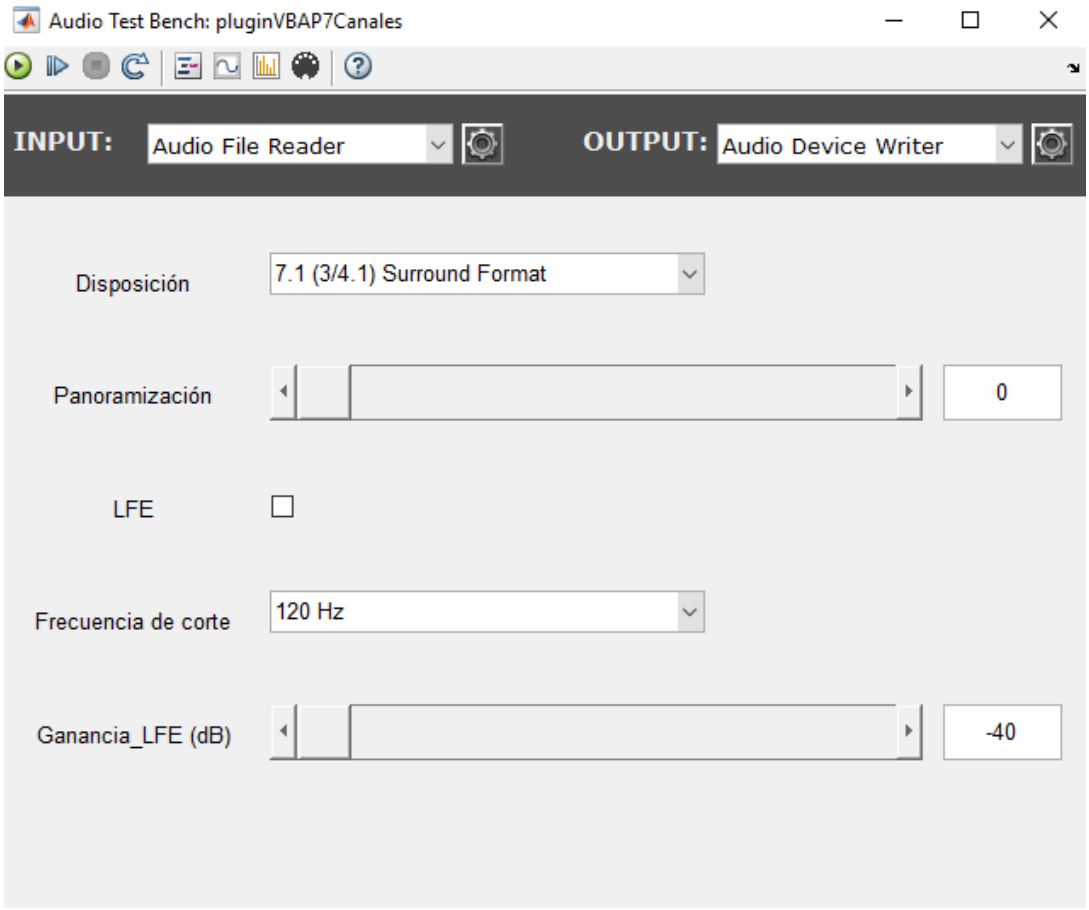


Figura 22: Audio Test Bench de pluginVBAP7Canales

Es necesario configurar las entradas y salidas de audio, los objetos encargados de realizar éstas operaciones son los siguientes: “Audio File Reader”, lee las muestras desde un archivo de audio, “Audio Device Reader”, se encarga de tomar las muestras desde la interfaz de audio (desde el conversor analógico-digital), “Audio File Writer”, convierte el audio procesado en un archivo y “Audio Device writer” escribe las muestras procesadas en el conversor analógico-digital de la interfaz de audio y permite escucharlas en tiempo real.

Se llevaron a cabo, con el material del laboratorio, las siguientes pruebas:

- Se realizó un "debug" del código, paso a paso se observaron las tareas que iba ejecutando el programa. Entre otras, se comprobó que las ganancias calculadas eran las correctas, que el arco activo fuese el correcto dependiendo del valor del ángulo virtual y que las salidas de audio se enviaban a los altavoces correspondientes.
- Pruebas de percepción auditiva, se comprobó que las muestras panoramizadas se localizaban en la posición correcta, que no existen caídas de ganancia a lo largo del arco activo y se comprobó que no existe distorsión perceptible por parte del filtro del canal LFE.

Por último, se realizó el llamado Audio Plugin Workflow, que consiste en comprobar que el programa puede ser interpretado por el codificador de MATLAB con un conjunto de pruebas predefinidas que se ejecutan al llamar a la función "validateAudioPlugin". Una vez realizado dicho test, se adaptó el código para poder ser compilado.

Cuando el programa estaba listo para ser compilado, se ejecutó la función "generateAudioPlugin" que, mediante el compilador de Visual Studio 2013, genera un archivo .dll que puede ser utilizado en un software DAW.

El siguiente paso es ejecutar el Plug-in en un software DAW, en este caso, se probó en REAPER y Cubase. Debido a que "Audio System Toolbox" no permite diseñar la interfaz de usuario (únicamente permite añadir parámetros de control asociados a propiedades), es el DAW el que se encarga de dicha tarea. La interfaz del Plug-in "pluginVBAP7Canales" ejecutada en REAPER es la que se presenta en la Figura 23.

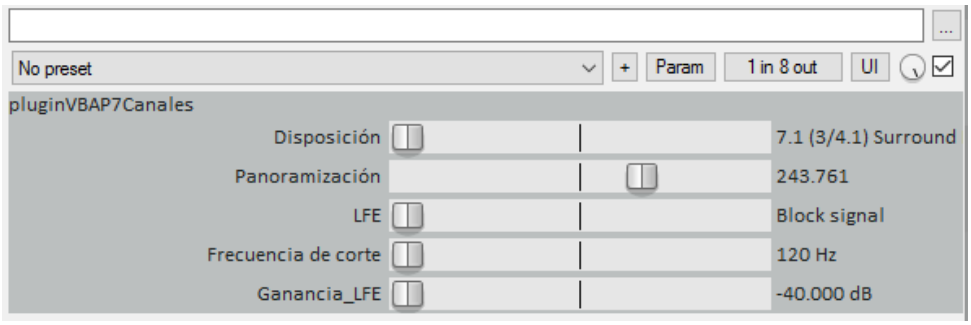


Figura 23: Interfaz de usuario de pluginVBAP7Canales en el software REAPER

4.9. Calibración

El proceso de calibración de un sistema sonoro es necesario en cualquier estudio, asegura que la mezcla que se realice en un entorno controlado tendrá las mismas características al reproducirse en otros sistemas.

Para realizar la calibración de un sistema de altavoces es necesario una fuente de ruido rosa, un generador de funciones (para generar señales senoidales) y un micrófono de medida calibrado. Éste debe estar conectado a un sistema RTA (Real-Time Analyzer) o, en su defecto, a un sonómetro configurado con ponderación C y respuesta lenta.

4.9.1. Proceso de calibración de fase del subwoofer:

1) Con un generador de funciones se envía una señal senoidal, con una frecuencia de 80 Hz, a los canales principales (L y R) y al subwoofer.
2) Con el Sonómetro en la posición central y apuntando hacia arriba se mide el valor de presión sonora y se anota.
3) Activamos el control de inversión de fase del subwoofer.
4) Volvemos a repetir los pasos 1) y 2).
5) Colocamos el control de fase en la posición en la que la ganancia es máxima.

4.9.2. Proceso de calibración de amplitud del sistema:

1) Se decide el nivel de presión sonora de referencia, éste paso dependerá del destino del material a mezclar en el estudio, se recomiendan los valores de la Tabla 3.
2) Se conecta una fuente de ruido rosa no correlado a cada una de las entradas digitales del sistema y se ajusta a un nivel de salida de -20 dBFS RMS.
3) Se envía la señal de ruido rosa a los distintos canales para comprobar que el conexionado se ha realizado correctamente.

4) Se posiciona el micrófono del equipo de medida apuntando hacia arriba en la posición de escucha (posición central), éste debe estar configurado para que el tiempo de cálculo promedio sea de entre 3 y 10 segundos.
5) Se ajusta el nivel del máster a 0 dB (nivel de referencia).
6) Se envía la señal de ruido rosa al altavoz izquierdo y se va aumentando progresivamente la ganancia del amplificador hasta que el nivel medido alcance al de referencia. Si la ganancia del altavoz no es suficiente será necesario ajustar el nivel en los controles de salida de la interfaz de audio.
7) Si las medidas se están haciendo en un RTA y las bandas de octava medidas en el RTA presentan picos y valles que superan los ±3 dB es posible que se deba modificar la acústica de la sala o aplicar una ecualización al sistema de monitores.
8) Se deben repetir los pasos anteriores para los diferentes altavoces que componen el sistema. Hay que tener en cuenta que algunos estándares recomiendan, para salas de cine, ajustar los altavoces envolventes a -3 dB con respecto de los principales y para producción musical la ganancia de dichos altavoces debe igualarse al resto. [17]

Tabla 3: Niveles de presión sonora recomendados [17]

Niveles de presión sonora recomendados	
TV	79 dB
Cine	83 dB
Música	85 dB

4.9.3. Calibración del canal LFE

Para compensar la baja sensibilidad a bajas frecuencias del oído, las bandas RTA para el canal LFE deberían dar una lectura 10 dB superiores a la banda de 1 kHz. Para un ruido rosa de -20 dBFS RMS, se reproduce en solitario por el canal LFE y se ajusta el subwoofer hasta que la lectura de la banda de 50 o 63 Hz de los resultados esperados En caso de que se realicen las medidas con un sonómetro, hay que enviar ruido rosa limitado en banda entre 20 Hz y 120 Hz y se va aumentando la ganancia del subwoofer hasta obtener una medida de 4 dB superior al del resto de monitores. [17]

4.10. Plug-in de calibración

Se ha implementado una aplicación para facilitar el proceso de calibración de sistemas multicanal 7.1. Consiste en una interfaz que permite realizar el envío individual de la señal de entrada a los diferentes canales de salida de la interfaz de audio. Además, dispone de un control de ganancia para, en caso de que sea necesario, ajustar el nivel de la salida de un canal vía software.

Es necesario insertarlo en el canal máster surround del proyecto y permite hacer "Solo" (reproducir únicamente los canales que tengan marcada la casilla "solo") a cada uno de los ocho canales y ajustar la ganancia de salida de los canales individualmente (por software) para calibrar el sistema de escucha. Si se ajusta el nivel de salida vía software, con el objetivo de calibrar el sistema de monitorización, es necesario desactivar la acción de este Plug-in en el momento en el que se vaya a volcar el proyecto en un archivo de audio.

El funcionamiento del Plug-in es el siguiente: para cada canal se deja pasar la señal de entrada a través del módulo de ganancia si está activada la casilla de Solo. En caso negativo, se envía la señal "ceros" que, al igual que en el Plug-in de panoramización VBAP, es un vector con tantos ceros como el tamaño de la señal de entrada.

La interfaz gráfica en el "Audio Test Bench" y el REAPER se muestra en la Figura 24.

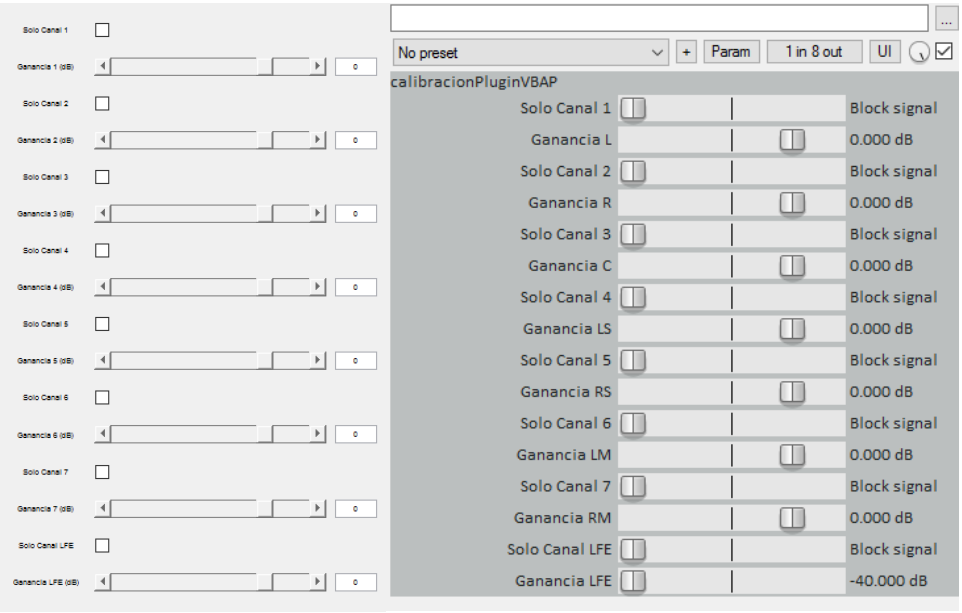


Figura 24: Audio Test Bench del Plug-in de calibración (izq.) Plug-in de calibración en REAPER

Capítulo 5. Resultados y conclusiones

En el primer capítulo de esta memoria se presentaron un conjunto de objetivos, éstos se plantearon a la hora de abordar el diseño de este Trabajo de Fin de Grado. La consecución de esos objetivos perseguía alcanzar el objetivo principal de este trabajo, que era el de desarrollar una herramienta capaz de ejecutarse en sistemas DAW, que pudiese procesar señales de audio e interactuar con las entradas y salidas de una interfaz de audio. Además, se planteó que la funcionalidad de dicha herramienta estuviese orientada a sistemas de audio multicanal.

Las conclusiones que se han obtenido tras el desarrollo de este trabajo son las siguientes:

- Se ha implementado el algoritmo VBAP en sistemas Plug-in para realizar la panoramización de una muestra monofónica en varios sistemas de altavoces, concretamente, en sistemas estéreo, 5.1 Surround y 7.1 surround.
- Los sistemas Plug-in VST que se han desarrollado con Maltab y “Audio System Toolbox” son compatibles con sistemas DAW profesionales como REAPER o Cubase.
- El algoritmo VBAP que se ha implementado permite posicionar las muestras sonoras en el espacio bidimensional de manera acertada.
- Haciendo uso únicamente de la herramienta VBAP es posible localizar la dirección de varias fuentes sonoras virtuales para generar un escenario sonoro artificial, pero, para potenciar el efecto de inmersión, es recomendable combinarlo con otro tipo de panoramizadores y efectos de reverberación.
- Se creó también un Plug-in para simplificar el proceso de calibración que permite aislar las señales de los canales de salida de manera independiente y ajustar su ganancia de salida.

Capítulo 6. Líneas futuras

Existe un gran número de posibilidades de ampliación para este proyecto. Se han considerado las siguientes líneas futuras:

- Implementar el algoritmo VBAP en otro lenguaje, como C++, que permite profundizar más en la programación, ya que Matlab está orientado al diseño de prototipos y, tareas como la reserva de espacio en memoria son transparentes para el programador. Redefinir el código, entre otras cosas, permitiría hacerlo más eficiente y daría la posibilidad de incorporarle una interfaz gráfica propia.
- El algoritmo VBAP también es compatible con configuraciones de altavoces en tres dimensiones. Como futura línea de desarrollo se podría diseñar un nuevo Plug-in que permita posicionar muestras monofónicas de audio en espacios tridimensionales.
- Añadir módulos de efectos de reverberación o delays al Plug-in desarrollado, el "pluginVBAP7Canales".

Capítulo 7. Bibliografía

[1] E. Pla Vall and K. Torrent Fuentes, “El cine como recurso didáctico. Módulo 3a - Historia del cine: Cine sonoro,” 2012. [Online]. Available: http://www.ite.educacion.es/formacion/materiales/24/cd/m3_1/. [Accessed: 03-May-2017].

[2] T. Holman, *Surround Sound - Up and Running*, Second Edi. 2008.

[3] Wrkshop, “The History of Fantasia and Fantasound!” [Online]. Available: https://vimeo.com/153064375. [Accessed: 01-Jun-2017].

[4] Steinberg Media Technologies, “VST3: New Standard for Virtual Studio Technology.” [Online]. Available: https://www.steinberg.net/en/company/technologies/vst3.html. [Accessed: 01-May-2017].

[5] Unión Internacional de Telecomunicaciones, “Sistema de sonido estereofónico multicanal con y sin acompañamiento de imagen. Recomendación UIT-R BS.775-3,” vol. 3, 2012.

[6] V. Pulkki, “Virtual Sound Source Positioning Using Vector Base Amplitude Panning,” 1997.

[7] Unión Internacional de Telecomunicaciones, “Multichannel sound technology in home and broadcasting applications. BS.2159,” *ITU-R Rep.*, 2010.

[8] Dolby Laboratories Inc., “7.1 Speaker Placement.” [Online]. Available: https://www.dolby.com/us/en/guide/surround-sound-speaker-setup/7-1-setup.html.

[9] B. Owsinski, *The Mixing Engineer’s Handbook, Second Edition*. 2006.

[10] Dolby Laboratories Inc., “Dolby Atmos Cinema Technical Guidelines.” .

[11] Unión Internacional de Telecomunicaciones, “Parámetros para el intercambio internacional de grabaciones sonoras multicanal. Recomendación UIT-R BR.1384,” pp. 1–6, 2001.

[12] A. V. Oppenheim and A. S. Willsky, *Señales y Sistemas*, 2ª Edición. Prentice Hall, 1997.

[13] M. F. Richard, *Elements Of Computer Music*. Prentice Hall, 1990.

[14] F. Rumsey, *Spatial Audio*, vol. 33. 2001.

[15] “Audición Binaural y Monoaural,” 2016. [Online]. Available: http://www.cochlea.eu/es/sonido/psicoacustica/localizacion. [Accessed: 06-Jun-

2017].

[16] Mathworks, *Audio System Toolbox Getting Started Guide*. 2016.

[17] J. P. Fisher, *Instant Surround Sound*. San Francisco: CMP Books.

PARTE II PRESUPUESTO

P.1. Presupuesto detallado

El presupuesto se ha desglosado en varias secciones en las que se han separado los distintos costes asociados al desarrollo del proyecto. Estos costes se dividen en:

- Trabajo Tarifado por Tiempo Empleado.
- Recursos Materiales.
- Material Fungible.
- Aplicación de Impuestos.

P.2. Trabajo tarifado por tiempo empleado:

Para realizar el cálculo de los honorarios por tiempo empleado se han tenido en cuenta todos los factores económicos que influyen en el ejercicio de la profesión de proyectista independiente (costes de seguros, transporte, impuestos, etc.).

Las tareas que se han realizado durante el proceso del trabajo y que se incluyen en este apartado del presupuesto son:

- Desarrollo de la herramienta software: 230 horas
- Redacción del proyecto: 70 horas

Por lo tanto, en la elaboración de este trabajo se han empleado 300 horas, todas en horario laboral y se considera que el coste por hora de trabajo es de 32 €.

El cálculo de los honorarios resulta:

$$H = 32 \cdot 300 = 9600 \text{ €}$$

Por lo tanto, los honorarios totales por tiempo dedicado, libres de impuestos, ascienden a **nueve mil seiscientos euros (9600 €)**.

P.3. Recursos materiales

Para la realización de este trabajo ha sido necesario hacer uso de ciertos recursos hardware y software. Para realizar el cálculo de la amortización de dichos recursos, se ha considerado que la vida útil de los mismos es de tres años (36 meses) y la duración del trabajo es de 4 meses.

$$Cuota\ anual = \frac{Valor\ de\ adquisición - Valor\ residual}{Número\ de\ años\ de\ vida\ útil}$$

El valor residual se refiere al valor que se supone que tendrá el recurso una vez concluida su vida útil.

P.1.1 Recursos hardware

Los recursos hardware en los que se apoya este proyecto son:

- Ordenador portátil con procesador Intel Core i5-5200U, 4GB de memoria RAM y sistema operativo de 64 bits.
- Interfaz de audio Focusrite Scarlett 18i20 con 10 salidas de audio y 8 previos de micrófono.
- Monitores de estudio M-Audio BX5 D2 (pareja), autoamplificados.
- M-Audio BX Subwoofer, autoamplificado.
- Cables de audio balanceados (3 metros).
- Soportes para monitores de estudio (pareja).
- Sonómetro RION NL-18
- Calibrador B&K 4231
- Prolongadores de 3 metros para realizar la conexión eléctrica.
- Bases múltiples para realizar la conexión eléctrica.

A continuación, se detallan los costes de los recursos materiales utilizados en la realización de este Trabajo Fin de Grado:

Tabla 4: Coste de Herramientas Hardware

Coste de Herramientas Hardware						
	Cantidad	**Coste por unidad**	**Coste total**	**Valor Residual**	**Coste/mes**	**Valor Amortización (4 meses)**
Ordenador portátil	1	700 €	700 €	0 €	19,44 €	77,76 €
Interfaz de audio Focusrite Scarlett 18i20	1	500 €	500 €	0 €	13,89 €	55,56 €
Monitores de estudio M-Audio BX5 D2 (pareja)	4	274 €	1096 €	0 €	30,44 €	121,76 €
M-Audio BX Subwoofer	1	405 €	405 €	0 €	11,25 €	45 €
Sonómetro RION NL - 128	1	4230 €	4230 €	0 €	117,5 €	470 €
Calibrador B&K 4231	1	1300 €	1300 €	0 €	36,1 €	144,4 €
Cables de audio balanceados (3 metros)	8	24,4 €	195,2 €	0 €	5,42 €	21,68 €
Soportes para monitores de estudio (pareja)	4	48 €	192 €	0 €	5,33 €	21,32 €
Prolongadores (3 metros)	4	5,5 €	22 €	0 €	0,61 €	2,44 €
Bases múltiples	4	8 €	32 €	0 €	0,89 €	3,56 €
Coste total:						963,48 €

El coste total de herramientas hardware, libre de impuestos, es de **novecientos sesenta y tres con cuarenta y ocho céntimos (963,48 €).**

P.1.2 Recursos Software

Se han usado las siguientes herramientas software:

- Entorno de programación técnica Matlab.
- "Audio System Toolbox".
- Visual Studio 2013, necesario para compilar el Plug-in,
- Microsoft Office 2016.
- Sistemas DAW Reaper y Cubase Elements 9.

Los costes de los recursos hardware se presentan en la tabla 6:

Tabla 5: Coste de Herramientas Software

Coste de Herramientas Software				
	Coste Total	**Valor Residual**	**Coste/ mes**	**Valor Amortización**
Matlab	2000 €	0 €	55,55 €	222,2 €
Audio System Toolbox	1250 €	0 €	34,72 €	138,88 €
Microsoft Office 2016	279 €	0 €	7,75 €	31 €
Reaper	0 €	0 €	0 €	0 €
Cubase Elements 9	100 €	0 €	2,78 €	11,12 €
Visual Studio 2013	0 €	0 €	0 €	0 €
Coste total:				403,2 €

El coste total de los recursos software, libre de impuestos, da un total de **cuatrocientos tres euros con dos céntimos (403,2€).**

P.4. Material fungible

Además de los ya citados materiales hardware y software, se han empleado materiales fungibles como folios y tinta de impresora para la entrega de este trabajo. En la tabla 7 se presentan los costes de estos recursos:

Tabla 6: Costes del material fungible

Costes del material fungible	
Descripción	Costes
Folios	3 €
Tóner de la impresora	43 €
Encuadernación	11 €
CDs	8 €
Total	65 €

P.5. Costes totales del TFG

Para calcular los costes totales del presente trabajo fin de grado es necesario sumar el total de los costes y añadir el 7% de impuestos correspondiente.

Tabla 7: Costes totales del TFG

Costes totales del TFG	
Descripción	Costes
Trabajo tarifado por tiempo empleado	9600 €
Recursos materiales	963,48 €
Material fungible	65 €
Subtotal	10628,48 €

Aplicación de impuestos (IGIC 7%)	743,99 €
Total	11372,47 €

El coste total del trabajo de fin de grado será, por tanto, de **once mil trescientos setenta y dos euros con cuarenta y siete céntimos (11372,47 €)**

El autor del trabajo:

Sergio Falcón Castellano

Las Palmas de Gran Canaria a de junio de 2017

PARTE III PLIEGO DE CONDICIONES

C.1. Pliego de condiciones

El equipo que se ha utilizado para probar y testear el funcionamiento de los Plug-in que se han desarrollado en este trabajo fin de título es el siguiente:

Hardware:

- Ordenador:
  - Microprocesador Intel Core i5 o superior.
  - Memoria RAM $\geq$ 4 GBytes.
  - Sistema operativo Windows de 64 bytes.
- Interfaz de audio: Focusrite 18i20.
  - 10 salidas de línea/monitores: respuesta en frecuencia: 20 Hz – 20 kHz, $\pm$0,5dB; rango dinámico: 108 dBA; THD+N = 0.001%.
  - 8 entradas de micrófono/línea: respuesta en frecuencia: 20 Hz – 20 kHz, +0,5/-1,5dB; rango dinámico: 109 dBA. THD+N = 0.003%.
- Monitores de escucha (x7): M-Audio BX5 D2.
  - Respuesta en frecuencia: 56 Hz – 22 kHz, rango dinámico: 100 dBA
- Subwoofer: M-audio BX Subwoofer.
  - Respuesta en frecuencia: 20 Hz - 200 Hz
- Sonómetro RION NL-18
- Calibrador B&K 4231
- Cableado (x8): Cables de audio de línea balanceados.

Software:

- Entorno de programación Matlab con “Audio System Toolbox”.
- Software DAW: Cubase Elements 9 y REAPER.
- Visual Studio 2013, únicamente necesario para compilar el Plug-in desde Matlab.
- Microsoft Office 2016.

C.2. Pliego de Condiciones Legales

C.2.1. Concesión de licencia

Este programa es propiedad de la Universidad de Las Palmas de Gran Canaria y cualquier usuario debe estar de acuerdo y cumplir los términos y condiciones establecidas en esta licencia del programa, aceptando todas sus cláusulas. El uso de este software o de una copia en un PC, será bajo la autorización expresa del autor, tutor del proyecto y de la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria.

C.2.2. Derechos de autor

Este programa, junto con la documentación, está protegido por las leyes de la propiedad intelectual que le sean aplicables, así como las disposiciones de los tratados internacionales. En consecuencia, el usuario debe utilizar el programa como cualquier producto protegido por derechos de autor. Sin embargo, el usuario podrá usar una copia y utilizar los códigos fuente de la programación y la documentación siempre bajo la autorización del autor, el tutor y de la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria.

C.2.3. Restricciones

El usuario podrá transferir el programa a un tercero, siempre que no tenga copias del programa, incluyendo posibles actualizaciones o retener material escrito adicional que acompañe al programa.

C.2.4. Limitación de responsabilidad

En ningún caso serán el autor ni el tutor, ni la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria responsables de los perjuicios directos, indirectos, incidentales o consiguientes, gastos, lucro cesante, pérdida de ahorros, interrupción de negocios, pérdida de información comercial o de negocio, o cualquier otra pérdida que resulte del uso o de la incapacidad de usar el programa o la documentación. El usuario conoce y acepta que los derechos de licencia reflejan esta asignación de riesgo como

el resto de cláusulas y restricciones. El autor y los tutores rechazan cualquier otra garantía que no haya sido indicada anteriormente.

C.2.5. Varios

En el supuesto de que cualquier disposición de esta licencia sea declarada total o parcialmente inválida, la cláusula afectada será modificada convenientemente de manera que sea ejecutable una vez modificada, plenamente eficaz, permaneciendo el resto de este contrato en vigencia.

Este contrato se rige por las leyes de España. El usuario acepta la jurisdicción exclusiva de los tribunales de este país en relación con cualquier disputa que pudiera derivarse de la presente licencia.

PARTE IV ANEXOS

A.1. Proyecto “StereoVBAP”

Panoramización estéreo con el algoritmo VBAP

```
% VBAP (Vector Base Amplitude Panning) estéreo

[y,Fs] = audioread('MUESTRA_DE_AUDIO.wav');


% Parámetros de las fuentes físicas según UIT-R BS.775-3
% Altavoz1 -> L (-60º y distancia = 1)
% Altavoz2 -> R (60º y distancia = 1)

ang1 = ((-60)*pi/180); % -60º en radianes
ang2 = ((60)*pi/180);  % 60º en radianes


% Matriz L12 (matriz compuesta por las componentes de los vectores
% que apuntan a las fuentes físicas)

L12 = [cos(ang1), sin(ang1); cos(ang2), sin(ang2)];


% Parámetros de la fuente virtual

angV = input('Teclee el angulo de la fuente virtual (entre -60 y
60):');
angV = ((angV)*pi/180); % En radianes


% Vector p (vector unitario en la dirección y el sentido de la
fuente   % virtual):

p = [cos(angV), sin(angV)];


% Vector g con los valores de ganancia de cada uno de los canales

g = p/L12;


% Normalización del vector g

c = 1; % Constante 'c'
gs = (sqrt(c)/sqrt(g(1)^2+g(2)^2))*g; % gs = gain scaled

% Se multiplica la señal de entrada por los valores de ganancia
calculados:
```

```
a = y*gs;

subplot(2,1,1);
plot(a(:,1));
axis([0 40000 -1 1])
subplot(2,1,2);
plot(a(:,2));
axis([0 40000 -1 1])


% Se genera un archivo .wav estéreo con la panoramización realizada

audiowrite('AUDIO_PANORAMIZADO.wav',a, Fs);
```

A.2. Proyecto “pluginVBAPStereo”

Implementación del proyecto VBAP Stereo en formato Plug-in

```
% Plugin VBAP estéreo

classdef pluginStereoVBAP < audioPlugin

    properties
        angV = 0; % Valor por defecto = 0º
    end

     properties (Access = private)
        ang1;
        ang2;
    end

    properties (Constant)
        PluginInterface =
audioPluginInterface(audioPluginParameter(...
            'angV', 'DisplayName', 'Panoramización', 'Mapping',
{'lin',-60,60}), 'InputChannels',1, 'OutputChannels', 2);
    end


    method

% Constructor:
% Parámetros de las fuentes físicas según UIT-R BS.775-3
% Altavoz1 -> L (-60º y distancia = 1)
% Altavoz2 -> R (60º y distancia = 1)

        function myPlugin = pluginStereoVBAP
            myPlugin.ang1 = ((-60)*pi/180);
            myPlugin.ang2 = ((60)*pi/180);
        end

        function out = process(myPlugin, in)


% Angulo de la fuente virtual en radianes

            angulo = myPlugin.angV;
            angulo = ((angulo)*pi/180);


% Matriz L12 (matriz compuesta por las componentes de los vectores
% que apuntan a las fuentes físicas)
```

```
            L12 = [cos(myPlugin.ang1), sin(myPlugin.ang1); ...
cos(myPlugin.ang2), sin(myPlugin.ang2)];


% Vector p (vector unitario en la dirección y el sentido de la
% fuente virtual):

            p = [cos(angulo), sin(angulo)];


% Vector g con los valores de ganancia de cada uno de los canales

            g = p/L12;


% Normalización del vector g

            c = 1;
            gs = (sqrt(c)/sqrt(g(1)^2+g(2)^2))*g;
            Lg = gs(1);
            Rg = gs(2);
            L = in*Lg;
            R = in*Rg;

            out = [L R];
        end

        function reset(myPlugin)
        end
    end

end
```

A.3. Proyecto “VBAP5Canales”:

Implementación del algoritmo VBAP en un sistema de 5 canales, sin LFE, en formato Matlab.

```
% VBAP (Vector Base Amplitude Panning) 5 canales

[y,Fs] = audioread('606CHAT.WAV');

dim = size(y);
sal = zeros(dim(1), 5);


% Ángulos de las fuentes físicas (por parámetros)

ang1 = input('Teclee el angulo de la fuente 1: ');
ang2 = input('Teclee el angulo de la fuente 2: ');
ang3 = input('Teclee el angulo de la fuente 3: ');
ang4 = input('Teclee el angulo de la fuente 4: ');
ang5 = input('Teclee el angulo de la fuente 5: ');


ang1 = ((ang1)*pi/180);  % En radianes
ang2 = ((ang2)*pi/180);  % En radianes
ang3 = ((ang3)*pi/180);  % En radianes
ang4 = ((ang4)*pi/180);  % En radianes
ang5 = ((ang5)*pi/180);  % En radianes


% Matriz L12 (matriz compuesta por las componentes de los vectores
% que apuntan a las fuentes físicas)

L12 = [cos(ang1), sin(ang1); cos(ang2), sin(ang2)];
L23 = [cos(ang2), sin(ang2); cos(ang3), sin(ang3)];
L34 = [cos(ang3), sin(ang3); cos(ang4), sin(ang4)];
L45 = [cos(ang4), sin(ang4); cos(ang5), sin(ang5)];
L51 = [cos(ang5), sin(ang5); cos(ang1), sin(ang1)];


% Parámetros de la fuente virtual

angV = input('Teclee el angulo de la fuente virtual: ');
angV = ((angV)*pi/180); % En radianes


% Vector p (vector unitario en la dirección y el sentido de la
fuente virtual):

p = [cos(angV), sin(angV)];
```

```
% Vector g con los valores de ganancia de cada uno de los canales

if (angV >= ang1) && (angV <= ang2)
   g = p/L12;
 else if angV>ang2 && angV<ang3
    g = p/L23;
 else if angV>ang3 && angV<ang4
    g = p/L34;
 else if angV>ang4 && angV<ang5
    g = p/L45;
 else if angV>ang5 && angV<2*pi
    g = p/L51;
     end
     end
     end
     end
end


% Normalización del vector g

c = 1; % Constante 'c' igual a '1'
gs = (sqrt(c)/sqrt(g(1)^2+g(2)^2))*g; % gs = gain scaled


% Se multiplica la señal de entrada por los valores de ganancia
% calculados:

a = y*gs;

% Se selecciona el sector (en el que se encuentra el ángulo virtual
% y se sustituyen los ceros de la matriz de salida por la señal
% panoramizada en los canales que corresponda

if (angV >= ang1) && (angV <= ang2)
   sal(:,1) = a(:,1);
   sal(:,2) = a(:,2);
 else if angV>ang2 && angV<ang3
   sal(:,2) = a(:,1);
   sal(:,3) = a(:,2);
 else if angV>ang3 && angV<ang4
   sal(:,3) = a(:,1);
   sal(:,4) = a(:,2);
 else if angV>ang4 && angV<ang5
   sal(:,4) = a(:,1);
   sal(:,5) = a(:,2);
 else if angV>ang5 && angV<2*pi
```

```
   sal(:,5) = a(:,1);
   sal(:,1) = a(:,2);
     end
     end
     end
     end
end
```

A.4. Proyecto “pluginVBAP5Canales”

Implementación del algoritmo VBAP en un sistema de 5 canales, sin LFE, en formato Plug-in.

```
classdef pluginVBAP5Channels < audioPlugin

    properties
        angV = 0;
    end

     properties (Access = private)
        ang1;
        ang2;
        ang3;
        ang5;
        ang6;
    end

    properties (Constant)
        PluginInterface =
audioPluginInterface(audioPluginParameter(...
            'angV', 'DisplayName', 'Panoramización', 'Mapping',
{'lin',0,359}), 'InputChannels',1, 'OutputChannels', 5);
    end


    methods

% Constructor

        function myPlugin = pluginVBAP5Channels
            myPlugin.ang1 = ((300)*pi/180); % Canal L, -60º en
radianes
            myPlugin.ang2 = ((60)*pi/180);  % Canal R, 60º en
radianes
            myPlugin.ang3 = ((0)*pi/180);   % Canal C, 0º en
radianes
            myPlugin.ang5 = ((240)*pi/180); % canal LS, -120º en
radianes
            myPlugin.ang6 = ((120)*pi/180); % canal RS 120º en
radianes
        end

        function out = process(myPlugin, in)

% Ángulo virtual en radianes
```

```
            angulo = myPlugin.angV;
            angulo = ((angulo)*pi/180);

% Matrices que caracterizan los distintos sectores

            L32 = [cos(myPlugin.ang3), sin(myPlugin.ang3);
cos(myPlugin.ang2), sin(myPlugin.ang2)];
            L26 = [cos(myPlugin.ang2), sin(myPlugin.ang2);
cos(myPlugin.ang6), sin(myPlugin.ang6)];
            L65 = [cos(myPlugin.ang6), sin(myPlugin.ang6);
cos(myPlugin.ang5), sin(myPlugin.ang5)];
            L51 = [cos(myPlugin.ang5), sin(myPlugin.ang5);
cos(myPlugin.ang1), sin(myPlugin.ang1)];
            L13 = [cos(myPlugin.ang1), sin(myPlugin.ang1);
cos(myPlugin.ang3), sin(myPlugin.ang3)];


            p = [cos(angulo), sin(angulo)];
            c = 1;


% Se selecciona el sector (en el que se encuentra el ángulo virtual,
% se calcula el vector de ganancia, se normaliza y se envía la señal
% a los canales que corresponda, en el resto se envía el vector
% "ceros"

            if ((myPlugin.angV >= myPlugin.ang3) && (myPlugin.angV <
myPlugin.ang2))
                g = p/L32;
                a = norvec(c,g,in);
                L = ceros;
                R = a(:,2);
                C = a(:,1);
                LS = ceros;
                RS = ceros;
            elseif ((myPlugin.angV>=myPlugin.ang2) &&
(myPlugin.angV<myPlugin.ang6))
                g = p/L26;
                a = norvec(c,g,in);
                L = ceros;
                R = a(:,1);
                C = ceros;
                LS = ceros;
                RS = a(:,2);
            elseif ((myPlugin.angV>=myPlugin.ang6) &&
(myPlugin.angV<myPlugin.ang5))
                g = p/L65;
                a = norvec(c,g,in);
                L = ceros;
```

```
                R = ceros;
                C = ceros;
                LS = a(:,2);
                RS = a(:,1);
            elseif ((myPlugin.angV>=myPlugin.ang5) &&
(myPlugin.angV<myPlugin.ang1))
                g = p/L51;
                a = norvec(c,g,in);
                L = a(:,2);
                R = ceros;
                C = ceros;
                LS = a(:,1);
                RS = ceros;
            elseif ((myPlugin.angV>=myPlugin.ang1) &&
(myPlugin.angV<2*pi))
                g = p/L13;
                a = norvec(c,g,in);
                L = a(:,1);
                R = ceros;
                C = a(:,2);
                LS = ceros;
                RS = ceros;
            end


% Matriz que envía la salida a la interfaz de audio

            out = [L,R,C,LS,RS];

        end

        function reset(myPlugin)
        end


% Normalización del vector g

        function a = norvec(c,g,in)
        gs = (sqrt(c)/sqrt(g(1)^2+g(2)^2))*g;   % gs = gain scaled


% Se multiplica la señal de entrada por los valores de ganancia
calculados:

        a = in*gs;
        end

    end
en
```

A.5. Proyecto “pluginVBAP7Canales”

Implementación del algoritmo VBAP en un sistema de 7 canales, con canal LFE, en formato Plug-in.

Está formado por los siguientes scripts:

- Función principal:
  - pluginVBAP7Canales
- Funciones secundarias:
  - LFE_80_44100
  - LFE_80_48000
  - LFE_100_44100
  - LFE_100_48000
  - LFE_120_44100
  - LFE_120_48000
  - desplegableFormato
  - desplegableFrecuenciaCorte

A.5.1. pluginVBAP7Canales

```
% Plug-in VBAP (Vector Base Amplitude Panning) para sistemas
% Surround 7.1


classdef pluginVBAP7Canales < audioPlugin

    properties

% Inicialización de los ángulos (virtual y físicos) con sus valores
por % defecto

        angV = 0;
        ang1 = ((330)*pi/180);
        ang2 = ((30)*pi/180);
```

```
        ang3 = ((0)*pi/180);
        ang4 = ((250)*pi/180);
        ang5 = ((110)*pi/180);
        ang6 = ((290)*pi/180);
        ang7 = ((70)*pi/180);


% Inicialización de los menús desplegables y de ganancia con sus
valores % por defecto

        Formato = desplegableFormato.Standard;
        Fc = desplegableFrecuenciaCorte.Cientoveinte;
        LFE = false;
        Ganancia_LFE = -40;

    end

    properties (Access = private)
        c = 1;
    end

    properties (Constant)


% Elementos que conforman la interfaz de usuario:

        PluginInterface =
audioPluginInterface(audioPluginParameter(...

'Formato','DisplayName','Disposición','Mapping',{'enum','7.1 (3/4.1)
Surround Format',’Dolby 7.1 Surround Format’,'7.1 SDDS Surround
Format'}),...
            audioPluginParameter('angV', 'DisplayName',
'Panoramización', 'Mapping', {'lin',0,359}),...
            audioPluginParameter('LFE', 'Mapping', {'enum','Block
signal','Pass through'}),...
            audioPluginParameter('Fc','DisplayName','Frecuencia de
corte','Mapping',{'enum','120 Hz','100 Hz','80 Hz'}),...
            audioPluginParameter('Ganancia_LFE','Label',
'dB','Mapping',{'pow', 1/3, -40, 3}),...
            'InputChannels',1, 'OutputChannels', 8);
    end


    methods

% Constructor

        function myPlugin = pluginVBAP7Canales
```

```
        end

        function out = process(myPlugin, in)

% Se da valor a los ángulos de las fuentes físicas en función de la
% selección realizada por el usuario en el menú desplegable

            switch (myPlugin.Formato)
                case desplegableFormato.Standard
% Canal L, -30º en radianes
                    myPlugin.ang1 = ((330)*pi/180);
% Canal R, 30º en radianes
                    myPlugin.ang2 = ((30)*pi/180);
% Canal C, 0º en radianes
                    myPlugin.ang3 = ((0)*pi/180);
% canal LS, -110º en radianes
                    myPlugin.ang4 = ((250)*pi/180);
% canal RS 110º en radianes
                    myPlugin.ang5 = ((110)*pi/180);
% canal LM, -70º en radianes
                    myPlugin.ang6 = ((290)*pi/180);
% canal RM 70º en radianes
                    myPlugin.ang7 = ((70)*pi/180);

                case desplegableFormato.Dolby
% Canal L, -26º en radianes
                    myPlugin.ang1 = ((334)*pi/180);
% Canal R, 26º en radianes
                    myPlugin.ang2 = ((26)*pi/180);
% Canal C, 0º en radianes
                    myPlugin.ang3 = ((0)*pi/180);
% canal LS, -135º en radianes
                    myPlugin.ang4 = ((2255)*pi/180);
% canal RS 135º en radianes
                    myPlugin.ang5 = ((135)*pi/180);
% canal LM, -100º en radianes
                    myPlugin.ang6 = ((260)*pi/180);
% canal RM 100º en radianes
                    myPlugin.ang7 = ((100)*pi/180);

                case desplegableFormato.SDDS
% Canal LC, -22.5º en radianes
                    myPlugin.ang1 = ((337.5)*pi/180);
% Canal RC, 22.5º en radianes
                    myPlugin.ang2 = ((22.5)*pi/180);
% Canal C, 0º en radianes
                    myPlugin.ang3 = ((0)*pi/180);
% Canal LS, -135º en radianes
                    myPlugin.ang4 = ((225)*pi/180);
```

```
% Canal RS 135º en radianes
                    myPlugin.ang5 = ((135)*pi/180);
% Canal L, -45º en radianes
                    myPlugin.ang6 = ((315)*pi/180);
% Canal R 45º en radianes
                    myPlugin.ang7 = ((45)*pi/180);
end

            angulo = myPlugin.angV;
            angulo = ((angulo)*pi/180);
            gananciaLFE = 10^(myPlugin.Ganancia_LFE/20);

% Matrices Lxx (matrices compuestas por las componentes de los
% vectores que apuntan a las fuentes físicas), determina los
% posibles sectores

            L32 = [cos(myPlugin.ang3), sin(myPlugin.ang3);
cos(myPlugin.ang2), sin(myPlugin.ang2)];
            L27 = [cos(myPlugin.ang2), sin(myPlugin.ang2);
cos(myPlugin.ang7), sin(myPlugin.ang7)];
            L75 = [cos(myPlugin.ang7), sin(myPlugin.ang7);
cos(myPlugin.ang5), sin(myPlugin.ang5)];
            L54 = [cos(myPlugin.ang5), sin(myPlugin.ang5);
cos(myPlugin.ang4), sin(myPlugin.ang4)];
            L46 = [cos(myPlugin.ang4), sin(myPlugin.ang4);
cos(myPlugin.ang6), sin(myPlugin.ang6)];
            L61 = [cos(myPlugin.ang6), sin(myPlugin.ang6);
cos(myPlugin.ang1), sin(myPlugin.ang1)];
            L13 = [cos(myPlugin.ang1), sin(myPlugin.ang1);
cos(myPlugin.ang3), sin(myPlugin.ang3)];


% Vector p (vector unitario en la dirección y el sentido de la
% fuente virtual):

            p = [cos(angulo), sin(angulo)];


% Vector de ceros con el mismo tamaño que la señal de entrada
% (depende del tamaño del enventanado seleccionado por el usuario en
% la configuración del proyecto de la DAW):

            ceros = in*0;


% Si está activada la casilla LFE se filtra la señal, en caso
% contrario se envía la señal ceros a la salida correspondiente

            if myPlugin.LFE == true
```

```
            LFE_signal = filtrar(myPlugin, in);
            canal_LFE = LFE_signal*gananciaLFE;
        else
            canal_LFE = ceros;
        end


% Se selecciona el sector (en el que se encuentra el ángulo virtual,
% se calcula el vector de ganancia, se normaliza y se envía la señal
% a los canales que corresponda, en el resto se envía el vector
% "ceros"

        if ((angulo >= myPlugin.ang3) && (angulo <
myPlugin.ang2))
            g = p/L32;
            a = norvec(myPlugin,g,in);
            L = a(:,1);
            R = a(:,2);
            canal_1 = ceros;
            canal_2 = R;
            canal_3 = L;
            canal_4 = ceros;
            canal_5 = ceros;
            canal_6 = ceros;
            canal_7 = ceros;
        elseif ((angulo >= myPlugin.ang2) && (angulo <
myPlugin.ang7))
            g = p/L27;
            a = norvec(myPlugin,g,in);
            L = a(:,1);
            R = a(:,2);
            canal_1 = ceros;
            canal_2 = L;
            canal_3 = ceros;
            canal_4 = ceros;
            canal_5 = ceros;
            canal_6 = ceros;
            canal_7 = R;
        elseif ((angulo >= myPlugin.ang7) && (angulo <
myPlugin.ang5))
            g = p/L75;
            a = norvec(myPlugin,g,in);
            L = a(:,1);
            R = a(:,2);
            canal_1 = ceros;
            canal_2 = ceros;
            canal_3 = ceros;
            canal_4 = ceros;
            canal_5 = R;
```

```
                canal_6 = ceros;
                canal_7 = L;
            elseif ((angulo >= myPlugin.ang5) && (angulo <
myPlugin.ang4))
                g = p/L54;
                a = norvec(myPlugin,g,in);
                L = a(:,1);
                R = a(:,2);
                canal_1 = ceros;
                canal_2 = ceros;
                canal_3 = ceros;
                canal_4 = R;
                canal_5 = L;
                canal_6 = ceros;
                canal_7 = ceros;
            elseif ((angulo >= myPlugin.ang4) && (angulo <
myPlugin.ang6))
                g = p/L46;
                a = norvec(myPlugin,g,in);
                L = a(:,1);
                R = a(:,2);
                canal_1 = ceros;
                canal_2 = ceros;
                canal_3 = ceros;
                canal_4 = L;
                canal_5 = ceros;
                canal_6 = R;
                canal_7 = ceros;
            elseif ((angulo >= myPlugin.ang6) && (angulo <
myPlugin.ang1))
                g = p/L61;
                a = norvec(myPlugin,g,in);
                L = a(:,1);
                R = a(:,2);
                canal_1 = R;
                canal_2 = ceros;
                canal_3 = ceros;
                canal_4 = ceros;
                canal_5 = ceros;
                canal_6 = L;
                canal_7 = ceros;
            elseif ((angulo >= myPlugin.ang1) && (angulo < 2*pi))
                g = p/L13;
                a = norvec(myPlugin,g,in);
                L = a(:,1);
                R = a(:,2);
                canal_1 = L;
                canal_2 = ceros;
                canal_3 = R;
```

```
                canal_4 = ceros;
                canal_5 = ceros;
                canal_6 = ceros;
                canal_7 = ceros;
            else
                canal_1 = ceros;
                canal_2 = ceros;
                canal_3 = ceros;
                canal_4 = ceros;
                canal_5 = ceros;
                canal_6 = ceros;
                canal_7 = ceros;
            end


% Matriz que envía la salida a la interfaz de audio

            out =
[canal_1,canal_2,canal_3,canal_4,canal_5,canal_LFE,canal_6,canal_7];

        end

        function reset(myPlugin)
        end

% Normalización del vector g

        function [a] = norvec(myPlugin,g,in)

% Constante 'c'

            constant = myPlugin.c;

% gs = gain scaled

            gs = (sqrt(constant)/sqrt(g(1)^2+g(2)^2))*g;

% Se multiplica la señal de entrada por los valores de ganancia
% calculados:

            a = in*gs;
        end

        function [LFE_signal] = filtrar(myPlugin,in)


% Pedimos al entorno que nos de la frecuencia de muestreo del
proyecto
```

```
            fMuestreo = myPlugin.getSampleRate;


% Necesario inicializar la variable signal para el codificador

            signal = in;


% Dependiendo de la frecuencia de muestreo y de la frecuencia de
% corte seleccionamos el filtro correspondiente

            switch (fMuestreo)
                    case 44100
                        if(myPlugin.Fc ==
desplegableFrecuenciaCorte.Cientoveinte)
                            signal = LFE_120_44100(in);
                        elseif(myPlugin.Fc ==
desplegableFrecuenciaCorte.Cien)
                            signal = LFE_100_44100(in);
                        elseif(myPlugin.Fc ==
desplegableFrecuenciaCorte.Ochenta)
                            signal = LFE_80_44100(in);
                        end

                    case 48000
                        if(myPlugin.Fc ==
desplegableFrecuenciaCorte.Cientoveinte)
                            signal = LFE_120_48000(in);
                        elseif(myPlugin.Fc ==
desplegableFrecuenciaCorte.Cien)
                            signal = LFE_100_48000(in);
                        elseif(myPlugin.Fc ==
desplegableFrecuenciaCorte.Ochenta)
                            signal = LFE_80_48000(in);
                        end

                otherwise
                    signal = in;

            end

            LFE_signal = signal;

        end
    end

end
```

A.5.2. LFE_80_44100

```
function y = LFE_80_44100(x)

persistent FILTER_LFE_80_44100;

if isempty(FILTER_LFE_80_44100)
    n = 500;
    fc = 80;
    fs = 44100;

    FILTER_LFE_80_44100 = fir1(n,fc/(fs/2),'low');

end


y = filter(FILTER_LFE_80_44100,1,x);
```

A.5.3. LFE_80_48000

```
function y = LFE_80_48000(x)

persistent FILTER_LFE_80_48000;

if isempty(FILTER_LFE_80_48000)
    n = 500;
    fc = 80;
    fs = 48000;

    FILTER_LFE_80_48000 = fir1(n,fc/(fs/2),'low');

end


y = filter(FILTER_LFE_80_48000,1,x);
```

A.5.4. LFE_100_44100

```
function y = LFE_100_44100(x)

persistent FILTER_LFE_100_44100;

if isempty(FILTER_LFE_100_44100)
    n = 400;
```

```
    fc = 100;
    fs = 44100;

    FILTER_LFE_100_44100 = fir1(n,fc/(fs/2),'low');

end

y = filter(FILTER_LFE_100_44100,1,x);
```

A.5.5. LFE_100_48000

```
function y = LFE_100_48000(x)

persistent FILTER_LFE_100_48000;

if isempty(FILTER_LFE_100_48000)
    n = 400;
    fc = 100;
    fs = 48000;

    FILTER_LFE_100_48000 = fir1(n,fc/(fs/2),'low');

end

y = filter(FILTER_LFE_100_48000,1,x);
```

A.5.6. LFE_120_44100

```
function y = LFE_120_44100(x)

persistent FILTER_LFE_120_44100;

if isempty(FILTER_LFE_120_44100)
    n = 300;
    fc = 120;
    fs = 44100;

    FILTER_LFE_120_44100 = fir1(n,fc/(fs/2),'low');

end


y = filter(FILTER_LFE_120_44100,1,x);
```

A.5.7. LFE_120_48000

```
function y = LFE_120_48000(x)

persistent FILTER_LFE_120_48000;

if isempty(FILTER_LFE_120_48000)
    n = 300;
    fc = 120;
    fs = 48000;

    FILTER_LFE_120_48000 = fir1(n,fc/(fs/2),'low');

end


y = filter(FILTER_LFE_120_48000,1,x);
```

A.5.8. desplegableFormato

```
classdef desplegableFormato < int8
    enumeration
        Standard  (0)
        SDDS      (1)
    end
end
```

A.5.9. desplegableFrecuenciaCorte

```
classdef desplegableFrecuenciaCorte < int8
    enumeration
        Cientoveinte (0)
        Cien  (1)
        Ochenta  (2)
    end
end
```

A.6. Proyecto calibracionPlugin

Implementación de un Plug-in para la calibración de un sistema multicanal, compuesto por 7 canales surround y un canal LFE.

```
classdef calibracionPlugin < audioPlugin

    properties

% Se inicializan las ganancias y las casillas de Solo

        Ganancia1 = 0;
        Ganancia2 = 0;
        Ganancia3 = 0;
        Ganancia4 = 0;
        Ganancia5 = 0;
        Ganancia6 = 0;
        Ganancia7 = 0;
        GananciaLFE = 0;
        Solo1 = false;
        Solo2 = false;
        Solo3 = false;
        Solo4 = false;
        Solo5 = false;
        Solo6 = false;
        Solo7 = false;
        SoloLFE = false;
    end

    properties (Access = private)
    end

    properties (Constant)

% Interfaz compuesta por 8 casillas de Solo y 8 faders de ganancia

        PluginInterface = audioPluginInterface(...
            audioPluginParameter('Solo1','DisplayName', 'Solo Canal
1', 'Mapping', {'enum','Block signal','Pass through'}),...
            audioPluginParameter('Ganancia1','DisplayName','Ganancia
1', 'Label','dB','Mapping',{'pow', 1/3, -40, 3}),...
            audioPluginParameter('Solo2','DisplayName', 'Solo Canal
2', 'Mapping', {'enum','Block signal','Pass through'}),...
            audioPluginParameter('Ganancia2','DisplayName','Ganancia
2','Label', 'dB','Mapping',{'pow', 1/3, -40, 3}),...
            audioPluginParameter('Solo3','DisplayName', 'Solo Canal
3', 'Mapping', {'enum','Block signal','Pass through'}),...
```

```
            audioPluginParameter('Ganancia3','DisplayName','Ganancia
3','Label', 'dB','Mapping',{'pow', 1/3, -40, 3}),...
            audioPluginParameter('Solo4','DisplayName', 'Solo Canal
4', 'Mapping', {'enum','Block signal','Pass through'}),...
            audioPluginParameter('Ganancia4','DisplayName','Ganancia
4','Label', 'dB','Mapping',{'pow', 1/3, -40, 3}),...
            audioPluginParameter('Solo5','DisplayName', 'Solo Canal
5', 'Mapping', {'enum','Block signal','Pass through'}),...
            audioPluginParameter('Ganancia5','DisplayName','Ganancia
5','Label', 'dB','Mapping',{'pow', 1/3, -40, 3}),...
            audioPluginParameter('Solo6','DisplayName', 'Solo Canal
6', 'Mapping', {'enum','Block signal','Pass through'}),...
            audioPluginParameter('Ganancia6','DisplayName','Ganancia
6','Label', 'dB','Mapping',{'pow', 1/3, -40, 3}),...
            audioPluginParameter('Solo7','DisplayName', 'Solo Canal
7', 'Mapping', {'enum','Block signal','Pass through'}),...
            audioPluginParameter('Ganancia7','DisplayName','Ganancia
7','Label', 'dB','Mapping',{'pow', 1/3, -40, 3}),...
            audioPluginParameter('SoloLFE','DisplayName', 'Solo
Canal LFE', 'Mapping', {'enum','Block signal','Pass through'}),...

audioPluginParameter('GananciaLFE','DisplayName','Ganancia
LFE','Label', 'dB','Mapping',{'pow', 1/3, -40, 3}),...
            'InputChannels',1, 'OutputChannels', 8);
    end


    methods

        function myPlugin = calibracionPlugin     % Constructor
        end

        function out = process(myPlugin, in)

% Vector de ceros del mismo tamaño que la señal de entrada

            ceros = in*0;

% Inicializamos los canales para que el codificador pueda
interpretarlos

            canal_1 = ceros;
            canal_2 = ceros;
            canal_3 = ceros;
            canal_4 = ceros;
            canal_5 = ceros;
            canal_6 = ceros;
            canal_7 = ceros;
            canal_LFE = ceros;
```

```
% Tomamos el valor en dB de la ganancia y lo pasamos a lineal

            gananciaCanal1 = 10^(myPlugin.Ganancia1/20);
            gananciaCanal2 = 10^(myPlugin.Ganancia2/20);
            gananciaCanal3 = 10^(myPlugin.Ganancia3/20);
            gananciaCanal4 = 10^(myPlugin.Ganancia4/20);
            gananciaCanal5 = 10^(myPlugin.Ganancia5/20);
            gananciaCanal6 = 10^(myPlugin.Ganancia6/20);
            gananciaCanal7 = 10^(myPlugin.Ganancia7/20);
            gananciaCanalLFE = 10^(myPlugin.GananciaLFE/20);

% Multiplicamos la señal de entrada por la ganancia del canal
% correspondiente

            signal1 = in*gananciaCanal1;
            signal2 = in*gananciaCanal2;
            signal3 = in*gananciaCanal3;
            signal4 = in*gananciaCanal4;
            signal5 = in*gananciaCanal5;
            signal6 = in*gananciaCanal6;
            signal7 = in*gananciaCanal7;
            signalLFE = in*gananciaCanalLFE;

% Si la casilla Solo de un canal concreto está activada se envía la
% señal correspondiente, en caso contrario se envía la señal ceros

            if myPlugin.Solo1 == true
                canal_1 = signal1;
            else
                canal_1 = ceros;
            end
            if myPlugin.Solo2 == true
                canal_2 = signal2;
            else
                canal_2 = ceros;
            end
            if myPlugin.Solo3 == true
                canal_3 = signal3;
            else
                canal_3 = ceros;
            end
            if myPlugin.Solo4 == true
                canal_4 = signal4;
            else
                canal_4 = ceros;
            end
            if myPlugin.Solo5 == true
                canal_5 = signal5;
```

```
            else
                canal_5 = ceros;
            end
            if myPlugin.Solo6 == true
                canal_6 = signal6;
            else
                canal_6 = ceros;
            end
            if myPlugin.Solo7 == true
                canal_7 = signal7;
            else
                canal_7 = ceros;
            end
            if myPlugin.SoloLFE == true
                canal_LFE = signalLFE;
            else
                canal_LFE = ceros;
            end

% Matriz que envía la salida a la interfaz de audio

            out =
[canal_1,canal_2,canal_3,canal_4,canal_5,canal_LFE,canal_6,canal_7];

        end

        function reset(myPlugin)
        end

    end

end
```