Universidad Zaragoza
1542

Trabajo Fin de Grado

Estudio de viabilidad de un sistema de localización sonora mediante el dispositivo UMA-8 de miniDSP

Feasibility study of a sound source location system using the miniDSP UMA-8 device

Autor

Sonia Escorza Santos

Director

Dr. José Ramón Beltrán Blázquez

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Sonia Escorza Santos,

con nº de DNI 16630450-R en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Estudio de viabilidad de un sistema de localizacion de fuentes sonoras mediante el dispositivo UMA-8 de miniDSP,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 23 de noviembre del 2017

Fdo: Sonia Escorza Santos

Agradecimientos

En primer lugar, a mi director José Ramón Beltrán por dirigir, coordinar y realizar conmigo este proyecto. También a David Díaz-Guerra por realizar el proyecto anterior y prestarme su ayuda y consejo, al igual que a Eduardo Estopiñán por su ayuda en la resolución de algunos problemas.

A mis compañeros de clase, por hacer que estos años hayan sido mucho mejores y más fáciles gracias a ellos. Por su ayuda y apoyo en los buenos y malos momentos.

A todos esos amigos, nuevos y antiguos, que han convivido estos años conmigo, permitiéndome aprender nuevas cosas y haciéndome crecer como persona. Gracias por acompañarme en lo que fue una nueva aventura, esperando que haya muchas otras.

A mi familia, especialmente a mis padres y mi hermana por permitirme esta oportunidad, por su cariño y su apoyo incondicional.

A todos vosotros, muchas gracias

Estudio de viabilidad de un sistema de localización sonora mediante el dispositivo UMA-8 de miniDSP

RESUMEN

El objetivo de este proyecto es estudiar la viabilidad del dispositivo UMA-8, de reciente aparición, en la localización de fuentes sonoras, y estudiar la sensibilidad de este dispositivo si se modificaba la posición de las fuentes.

Primero, para poder conocer la viabilidad del dispositivo, se ha realizado un estudio sobre los elementos que forman la placa del dispositivo y la información obtenida a través del fabricante, miniDSP. Una vez realizado este estudio se han implementado una aplicación para cada modo de uso de UMA-8, modo DSP y modo RAW. Para poder implementar estas aplicaciones se han utilizado los entornos Visual Basic y openFrameworks a través del IDE Visual Studio 2015. A partir de estas aplicaciones se ha realizado la localización de las fuentes sonoras, de modos distintos según la información que se tenía en cada modo. En el primer caso (modo DSP) se ha realizado una aplicación que permite visualizar la información de localización que proporciona el propio dispositivo. En el caso del modo RAW se ha desarrollado una aplicación de localización de fuentes basado en el algoritmo SRP-PHAT.

Por último, se ha realizado un estudio de los resultados para poder definir, de esta manera, las limitaciones de la agrupación y de los algoritmos implementados para la detección de fuentes.

Feasibility study of a sound source location system using the miniDSP UMA-8 device

ABSTRACT

The aim of this work was to study the feasibility of the recently introduced UMA-8 device in the location of sound sources. Additionally, the project aimed to study the sensitivity of this device when the position of these sources is modified.

First, to be able to know the feasibility of the UMA-8 device, a study was conducted on the device plate elements and the information obtained through the manufacturer, miniDSP. Once this study was carried out, an application was implemented for each mode of use of UMA-8: DSP mode and RAW mode. In order to implement these applications, the Visual Basic and the openFrameworks environments were used through the IDE Visual Studio 2015. From these applications, the sound sources were located in different ways according to the information that was available in each mode. In the first case (DSP mode) an application has been made that allows visualizing the location information provide by the device. In the case of RAW mode, a source location application base don the SRP_PHAT algorithm has been developed.

Finally, a study of the results was conducted in order to define the limitations of the array and the analysis implemented for the source detection.

.

Índice general

Índice de figuras

Índice de tablas

Lista de abreviaturas

UMA-8: USB microphone array (8 micrófonos)

MEMS: Sistemas micro-electromecánicos

I2S: Integrated interchip sound

RTOS: Sistema operativo en tiempo real

OTP: Memoria única programable

PLL: Bucle enganchado en fase

JTAG: Grupo de acción de prueba conjunta

DAC: Conversor digital-analógico

UDP: User Datagram Protocol

TCP: Transmission Control Protocol

OSC: Open Sound Control

SRP-PHAT: Steered Response Power with the PHAse Transform

GCCs: Global Command and Control System

Capítulo 1

Introducción

1. Motivación y contexto

Este proyecto surge de la idea de desarrollar un sistema electrónico propio para el control de una agrupación de micrófonos con el objetivo de desarrollar algoritmos de localización de fuentes sonoras. Previamente, se había realizado un Trabajo Fin de Máster [1], en el cual se realizaba esta tarea mediante una agrupación de dieciséis micrófonos convencionales, que se puede ver en la figura 1.1. Puede observarse que el tamaño de la agrupación es bastante grande, lo que dificulta la posibilidad de incorporarla en aplicaciones de usos cotidianos o más dirigidos a aplicaciones portátiles. Por todo ello, se decidió plantear el desarrollo de este proyecto, para disminuir el diseño anterior y que con un tamaño reducido pudiese ser introducido en juguetes o en otras aplicaciones de realidad virtual o aumentada. En un principio se quiso modificar la agrupación de micrófonos convencionales por micrófonos MEMS (sistemas micro-electromecánicos) y para ello se comenzó el estudio sobre este tipo de micrófonos con el objetivo de poder desarrollar un diseño completamente digital, dando especial importancia a esa reducción del tamaño del diseño inicial.

Además de la consideración de utilizar micrófonos MEMS, se pensó en utilizar el bus I2S (Integrated Interchip Sound), para que así a través de un único reloj se pudiese sincronizar todos los micrófonos conectados sin necesidad del uso de multiplexores con los cuales sería más complicada la lectura de datos. A lo largo de la búsqueda de los componentes adecuados para el diseño de la placa se encontraron distintas aplicaciones y placas de recepción de audio en tiempo real, en las cuales se utilizan microcontroladores XMOS, y por ello se comenzó a reunir información acerca de este tipo de microcontroladores. Durante este proceso de búsqueda se encontró el dispositivo UMA-8, USB microphone array, de miniDSP [2]. Con el hallazgo de este dispositivo de reciente aparición, se vio que reunía muchas de las características que se buscaban para el diseño inicial y a un bajo coste. Por eso se decidió centrar el estudio en este dispositivo, que presentaba, a priori, una gran cantidad de ventajas

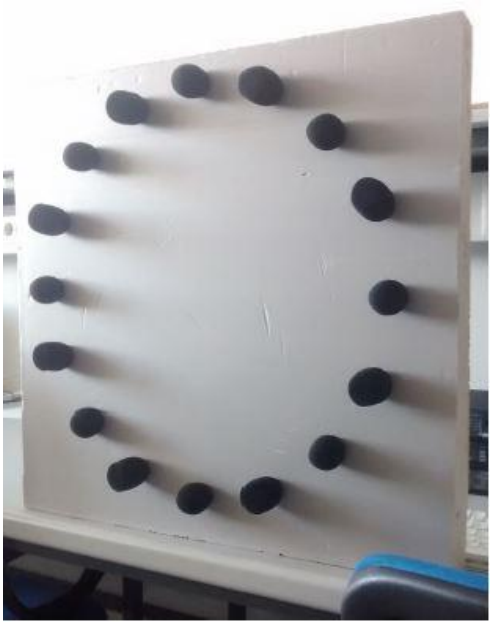

Figura 1.1: Agrupación de dieciséis micrófonos convencionales

2. Objetivos y requisitos del proyecto

El objetivo de este proyecto es realizar una aplicación de localización de fuentes sonoras basada en el dispositivo de miniDSP UMA-8, que se puede ver en la figura 1.2. Este dispositivo es una agrupación de 7 micrófonos que proporciona información espacial de la dirección de llegada de la fuente sonora. Se analiza el diseño y la tecnología de la placa UMA-8 y del microprocesador utilizado, así como los modos de trabajo y la información que proporciona. El objetivo final es valorar la utilidad de un dispositivo comercial sencillo frente a otro tipo de agrupaciones de micrófonos más complejas para la localización de fuentes sonoras o para la realización de medidas acústicas.

Figura 1.2. Dispositivo UMA-8 [3]

La placa, como se verá en el capítulo 3, incorpora una salida USB HID, a través de la cual se recogerán los datos aportados por el microcontrolador de la placa. Realización de un software de captura de la información que proporciona la placa UMA-8, para ello se utilizarán distintos entornos de programación dentro del IDE de desarrollo Visual Studio 2015 [5]. Estos entornos son openFrameworks [6] y Visual Basic [7], con los que se realizará la lectura de datos y la localización de fuentes sonoras. Al ser una placa de reciente aparición en el mercado y al no existir mucha información disponible en este momento, se estudiará la posibilidad de utilizar las salidas individuales de los micrófonos para valorar la implementación de diferentes algoritmos de localización de fuentes o de medidas acústicas.

Por lo tanto, los objetivos concretos del proyecto son:

1. Estudio tecnológico sobre los elementos más importantes del dispositivo.
2. Análisis hardware y software del sistema UMA-8.
3. Desarrollo de una aplicación de localización de fuentes sonoras modificable dependiente de los modos de trabajo del dispositivo UMA-8.

1.3. Estructura de la memoria

Este documento se divide en dos partes, la primera contiene la memoria del proyecto, en la que se puede encontrar el resumen del proyecto realizado. En la segunda parte se pueden encontrar los anexos, en ellos se encuentra detallados algunos aspectos del proyecto además de los códigos de las aplicaciones.

La memoria contiene los siguientes capítulos:

- *Capítulo 2. Estudio tecnológico*: Contiene un estudio de la funcionalidad de los elementos más importantes del dispositivo.
- *Capítulo 3. Dispositivo UMA-8:* Descripción del dispositivo UMA-8, además de los distintos modos de uso, “Modo DSP” y “Modo RAW”. También se incluye la presentación del software propio del dispositivo de miniDSP.
- *Capítulo 4. Software para UMA-8*: Se podrá encontrar la explicación del software desarrollado para la localización de fuentes sonoras, y de la

recepción y lectura de datos proporcionados por el dispositivo a través del USB.

- *Capítulo 5. Resultados obtenidos*: Se muestra la aplicación funcional creada para el dispositivo UMA-8 para sus distintos modos de uso. Demostrando el correcto funcionamiento de éstas.
- *Capítulo 6. Conclusiones y trabajos futuros*: Recoge las conclusiones del proyecto y trabajos fututos a realizar.

Los anexos del proyecto serán los siguientes:

- *Anexo A*: Disposición de los elementos de la placa y dimensiones de ésta.
- *Anexo B*: Características del microprocesador XVSM-2000 de XMOS.
- *Anexo C*: Código del software para "modo DSP" realizado en Visual Basic.
- *Anexo D*: Código del software para "modo DSP" en openFrameworks.
- *Anexo E*: Código del software para el "modo RAW" en openFramewoks.

1.4. Cronograma

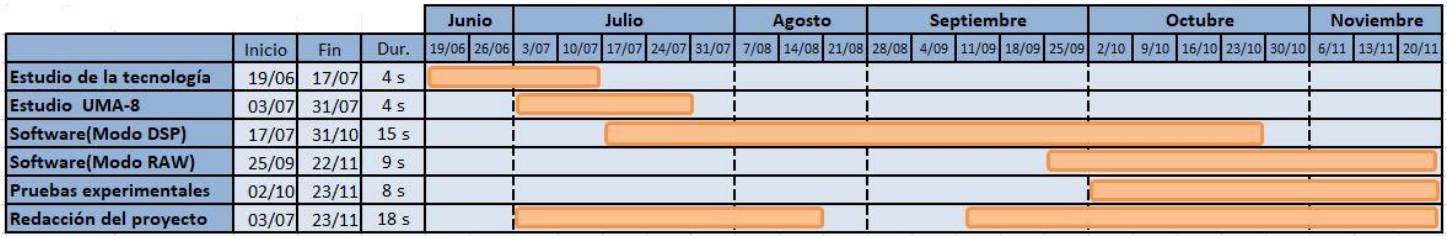

Figura 1.3. Cronograma del proyecto

Capítulo 2

Estudio tecnológico

Este capítulo se centra en el estudio de la tecnología utilizada en el dispositivo el UMA-8 de miniDSP. Primero se describe el microcontrolador XMOS utilizado, continuando con los micrófonos tipo MEMS que forman la agrupación. En este capítulo se aborda su descripción y funcionamiento como dispositivos aislados y no como parte del dispositivo UMA-8.

2.1. Microcontrolador XMOS

El microcontrolador XMOS es un microcontrolador multinúcleo capaz de realizar simultáneamente varias tareas en tiempo real, permite hacer tanto procesado digital como control de flujo de los procesos internos. Este microcontrolador fue creado por la empresa XMOS fundada en 2005 [8] y utiliza la tecnología XCORE. Esta tecnología, ofrece muchos de los elementos que se suelen utilizar en sistemas en tiempo real, donde se incluyen el programador de tareas, los temporizadores, operaciones de entradas y salidas y la comunicación de canales. Además, se eliminan interrupciones, chaches y otros recursos compartidos para que su rendimiento sea predecible, haciendo que pueda responder en periodos muy cortos de tiempo, es por ello, que son utilizados en tareas en tiempo real, ya que pueden llegar a ejecutarse ocho tareas distintas al mismo tiempo. Estos dispositivos suelen utilizarse en procesamiento de voz, audio USB, como en este caso, comunicaciones industriales y robóticas entre otras. Con la utilización de esta tecnología se consigue además un alto rendimiento a un bajo coste. Pero la razón principal de su utilización es su capacidad de trabajo en tiempo real. Es por ello que todas las opciones encontradas mientras se buscaba una solución para el diseño de la placa de captura de datos de audio a través del USB, se realizaban con un microcontrolador XMOS. En la hoja de características del dispositivo UMA-8 [3], podemos encontrar que dentro de las posibilidades ofrecidas por el fabricante se escoge para este dispositivo el modelo XMOS XSVM 2000. Para estudiar de forma más exhaustiva su funcionamiento se ha analizado su hoja de características [9]. Como se especifica en su

datasheet esta serie xCORE-200, está formada por microcontroladores multicore de 32 bits. Como ya se ha comentado, la principal diferencia con microcontroladores más convencionales es la posibilidad de ejecución simultánea de múltiples tareas en tiempo real. Estas tareas se comunican a través de una red de alta velocidad, pudiéndose, de este modo, realizarlas internamente en el propio microcontrolador sin necesidad de incorporar un hardware dedicado para ello. Algunas de las tareas más importantes se mencionan más adelante en las características principales del microcontrolador. En el anexo B se pueden encontrar datos más concretos y específicos por si se requiere más información acerca de las características de este dispositivo.

Veremos a continuación las principales características del microcontrolador XMOS utilizado (ver figura 2.1).

- *Tiles*. Los dispositivos consisten en uno o más módulos (*tiles)* xCORE. Cada uno contiene entre cinco y ocho xCOREs de 32 bits con alta integración de entradas y salidas, y memoria en el chip.
- Núcleos lógicos. Cada núcleo lógico puede realizar tareas como ejecución de código computacional, código DSP (procesador digital de señales) software de control (incluyendo decisiones lógicas y ejecución de una máquina de estados) o software de manejo de entradas y salidas.
- Planificador xTIME. Realiza funciones similares a las de un sistema operativo en tiempo real, RTOS, en hardware. Proporciona servicios de sincronización de eventos en un núcleo, por lo que no es necesario un control separado para las interrupciones. El planificador xTIME activa los núcleos mediante eventos generados por recursos de hardware como los pines de entrada y salida, y realiza también la comunicación de canales y los temporizadores. Una vez activado, un núcleo se ejecuta independientemente y de manera simultánea a otros núcleos. Al finalizar, el núcleo correspondiente se coloca en un estado de pausa para esperar nuevos eventos.
- Canales y extremos de canal. Las tareas se ejecutan en núcleos lógicos que se comunican mediante canales formados entre dos extremos del canal. Los datos se pueden pasar de forma síncrona o asíncrona entre los extremos de canal asignados a la comunicación de tareas.

- Conmutador xCONNECT y enlaces entre los *tiles*. Las comunicaciones de canal se implementan a través de una red de alto rendimiento de enlaces xCONNECT que son enrutados a través de un conmutador xCONNECT de hardware.
- Puertos. Los pines de E / S están conectados a los núcleos de procesamiento por puertos de *Hardware Response*. La lógica de control de los puertos puede actuar sobre los pines de E/S colocándolos en estado alto o bajo, o puede muestrear el valor de los pines a la espera de una condición particular.
- Bloques de reloj. Los dispositivos xCORE incluyen un conjunto de bloques de reloj programables que pueden usarse para controlar la velocidad a la que se ejecutan los códigos.
- Memoria. Cada *tile xCORE* integra una SRAM para instrucciones y datos, y un bloque de memoria única programable, OTP, que se puede configurar para integrar las funciones de seguridad de todo el sistema.
- PLL (Bucle enganchado en fase). El PLL se utiliza para crear el reloj del procesador de alta velocidad dada una velocidad baja del oscilador externo.
- USB. El Physical USB, PHY USB, puede utilizarse en modo de alta velocidad (1.0) o de velocidad "total" (2.0), actuar como dispositivo, como host, o como anfitrión. Los datos se comunican a través de puertos en el nodo digital. Se proporciona una librería software para implementar la funcionalidad del dispositivo USB [12].
- Flash. El dispositivo tiene una memoria flash incorporada de 2MB.
- JTAG (Grupo de Acción de Prueba Conjunta). El módulo JTAG puede utilizarse para cargar programas, pruebas de escaneo de límites, tiene un circuito a nivel de *debug* y programación de la memoria OTP.

Todo lo mencionado en los puntos anteriores se puede ver en la siguiente representación, figura 2.1, en la cual se pueden observar los bloques principales de los microcontroladores XMOS.

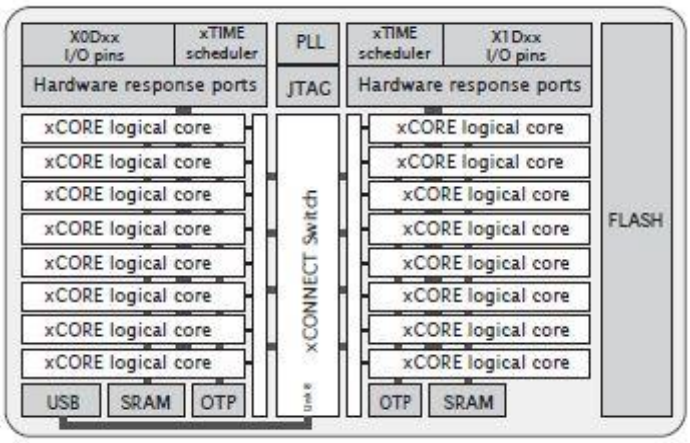

Figura 2.1: Características principales XMOS XSVM 2000 [9]

A continuación, el estudio se centrará en la parte digital del microcontrolador y su relación respecto a las entradas y salidas. En su hoja de características podemos encontrar el mapa de pines y una tabla con las características de cada uno de ellos, imprescindible si se estuviera realizando el diseño de la placa. El dispositivo tiene combinaciones de puertos de 1, 4, 8 y 16 bits, además habría que tener en cuenta que todos los pines de un puerto proporcionan o entrada o salida, que tienen diferentes direcciones asignadas por lo que no podrán ser detectadas simultáneamente en el mismo puerto. En la hoja de características se pueden encontrar diseños para cada una de las funciones que realiza el microcontrolador como, por ejemplo, la conexión con un dispositivo USB.

A modo de resumen, englobando lo dicho en este apartado y relacionándolo con el dispositivo UMA-8, este microcontrolador fue elegido por su capacidad de trabajo en tiempo real gracias a su capacidad de procesado en paralelo. En este proyecto no ha sido necesario la configuración del microcontrolador ya que el dispositivo UMA-8 consta de un driver propio, a través del cual, como se mencionará en el tercer capítulo, es posible cambiar el modo de uso.

2.2. Micrófonos MEMS

Los micrófonos son una parte fundamental en un diseño en el cual se desea realizar la localización espacial de un sonido. Durante el estudio tecnológico realizado de forma previa al

descubrimiento del dispositivo UMA-8, se les dio mucha importancia a estos elementos y se eligieron los micrófonos de tipo MEMS por su reducido tamaño y sus prestaciones. Pero estos micrófonos necesitan una soldadura muy compleja para realizarla manualmente, que sería la opción que se podría disponer para el desarrollo de un prototipo en este proyecto. Por eso, al abordar la selección de estos dispositivos se buscaba que fuesen acompañados de placas ya montadas (*breakout boards*) y de esta forma asegurar que el orificio de captación de sonido estuviese debidamente colocado. El micrófono utilizado por el dispositivo UMA-8 es el SPH1668LM4H, que al igual que muchos otros de tipo MEMS captan el sonido a través de un orificio, marcado el rojo en la figura 2.2, en la cara de soldadura del dispositivo y por ello se hace necesario que la placa en su diseño conste también de ese orificio. En el caso de un error en la posición y en la soldadura, el sonido no será captado correctamente produciendo errores en la detección, pudiéndose llegar a que la recepción del sonido sea inexistente. Por todo ello la mayoría de estos micrófonos se venden en placas ya soldadas que evitan los fallos comentados. Se puede ver una placa de este tipo en la figura 2.3.

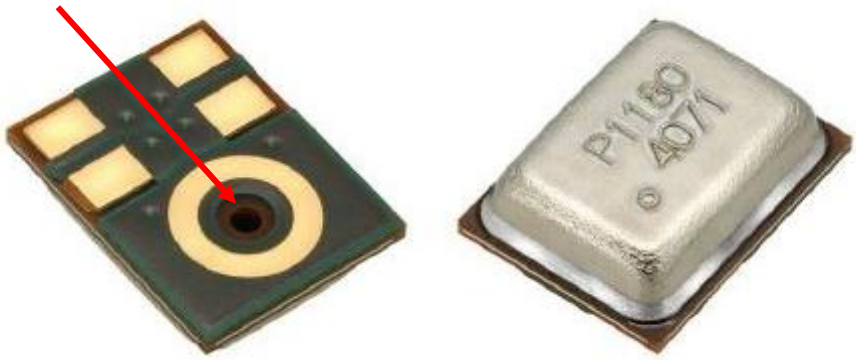

Figura 2.2: Micrófono SPH1668LM4H[10]

En la figura 2.2 se muestra el micrófono SPH1668LM4H que como se ha dicho anteriormente es el utilizado en el dispositivo UMA-8. Para realizar un pequeño estudio sobre sus características y funcionamiento se ha tomado como referencia su hoja de características [10]. Este micrófono, fabricado en silicio, ofrece una salida modulación PDM con un único bit, de esta forma la amplitud de la señal representa la densidad, el número de impulsos, en función del tiempo. Las características más importantes, que pueden verse en su hoja de característica, por las cuales fueron seleccionados estos micrófonos son:

- Baja distorsión del 1.6% a 120 dBSPL.

- Alta relación señal a ruido, SNR, de 65.5 dB.
- Bajo consumo de corriente, 230uA en modo de baja potencia.
- Respuesta de frecuencia (plana), entre 100 Hz - 10 kHz.
- Distorsión de radiofrecuencia blindada.
- Admite canales multiplexados duales.
- Múltiples modos de rendimiento, como se puede ver en la figura 2.4.
- Tamaño reducido.

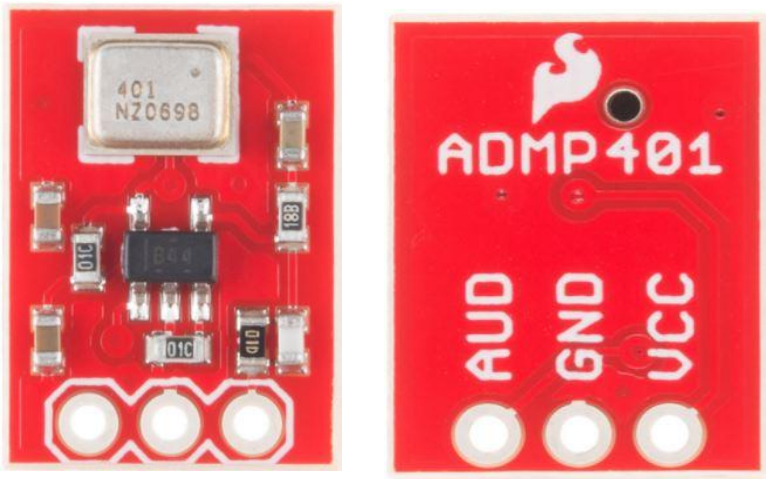

Figura 2.3: Placas soladas con micrófonos MEMS

Se continuará el estudio del micrófono conociendo un poco más a fondo su forma de trabajo. Este micrófono digital consta de un sensor acústico de actividad que, aunque siempre que esté alimentado el dispositivo esté activo, el consumo no sea el mismo cuando esté captado sonido o cuando esté en reposo. En estado de reposo el dispositivo se coloca esperando una señal con una intensidad suficientemente alta dentro de unos paramentos que dependen de la intensidad de la señal a recibir. En la figura 2.4 se puede ver la máquina de estados en la que se describen los modos de funcionamiento del micrófono.

Como se puede ver en la figura 2.4 en todo momento cuando se encuentra en uno de los tres estados de trabajo, la alimentación puede estar entre 3,6 y 1,62 V. Por otro lado, según la frecuencia de trabajo el dispositivo podrá estar en reposo (“Sleep Mode”) o en el modo de baja potencia (“Low-Power Mode”). Estos modos son zonas de trabajo bajo o de latencia donde se espera a que los datos recibidos sean los supuestos para entrar en la zona normal de

trabajo (“Performance Mode”). Como se puede ver consta de dos intervalos de frecuencias distintos para poder recibir información, las especificaciones eléctricas y acústicas en los dos modos más activos son prácticamente equivalentes ya que están relacionados frecuencialmente.

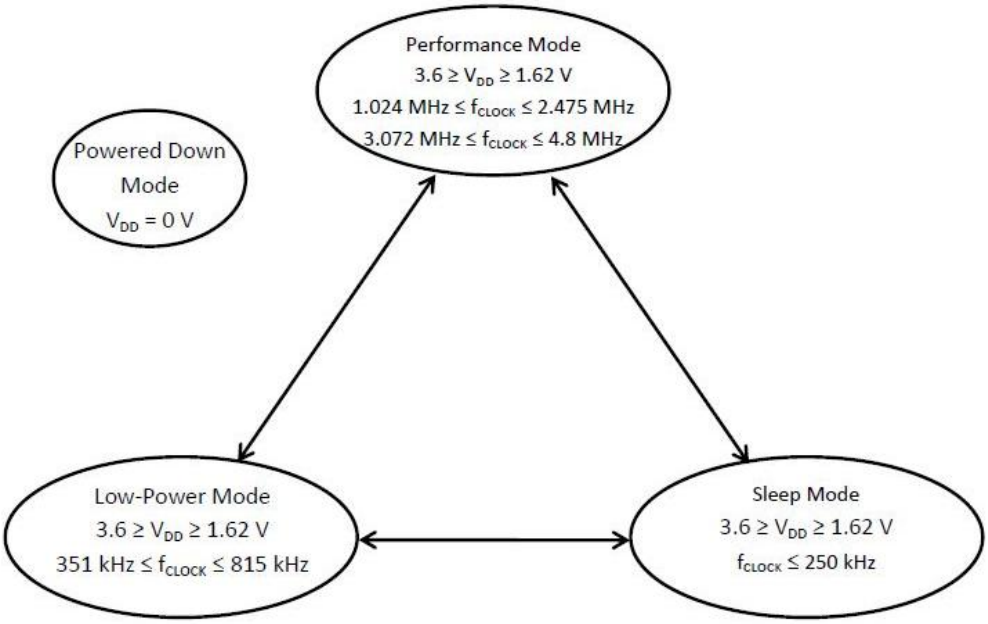

Figura 2.4: Máquina de estados SPH1668LM4H [10]

En este tipo de dispositivos es muy importante estudiar cómo se recibe la información, y para ello hay que observar con detenimiento el diagrama de la figura 2.5. En él se muestran los cambios temporales de la señal de reloj, y las señales de datos. Estos datos hay que tenerlos en cuenta ya que los instantes en los que se producen las transiciones son los que pueden impedir una correcta recepción si no se han tenido cuenta los tiempos correctos. En este diagrama de tiempos se puede ver una señal *DATA*, en ella se encontrará la información que se está recibiendo. Esta señal depende del valor que tenga la señal *SELECT*, que indica en qué modo de trabajo se está recibiendo. Todas ellas dependen de la señal de reloj, CLK, que oscila a una frecuencia de 250 kHz.

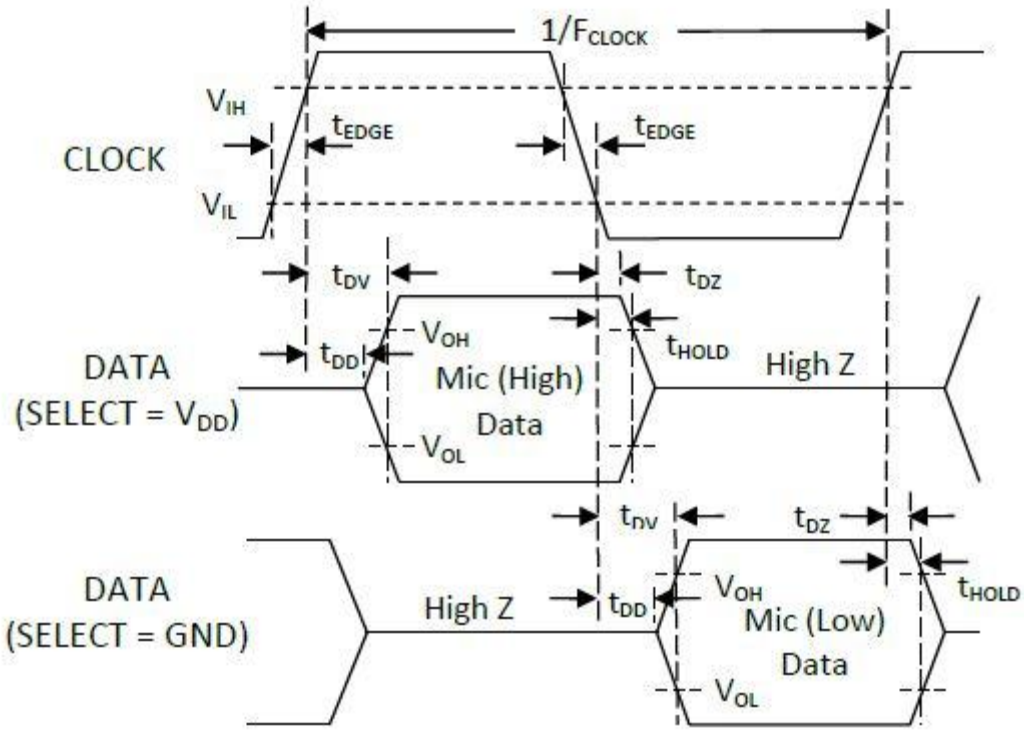

Figura 2.5: Diagrama de tiempos micrófono SPH1668LM4H [10]

Como se ha mencionado anteriormente en su hoja de características [10] se pueden encontrar otras propiedades como las dimensiones del micrófono, distintos diseños y modos de trabajo y datos referentes a tiempos de actuación, los cuales dependen de la temperatura a la que se encuentre. En el capítulo siguiente se verán las características que determinan la utilización de estos micrófonos en el dispositivo UMA-8.

Capítulo 3

Dispositivo UMA-8

En este capítulo se explica el funcionamiento del dispositivo UMA-8, USB *microphone array*, con el que se ha realizado este proyecto. El capítulo se divide en dos partes, en la primera se habla del hardware del dispositivo dando mayor importancia a las partes comentadas en el capítulo anterior, abordando también su distribución, tamaño y forma. En la segunda parte se comentan los aspectos más importantes del software propio del producto, en el cual, como se verá más adelante, a través del USB, es posible obtener los datos de posición de las fuentes de sonido. Se permite desde esta aplicación cambiar algunas de las variables para realizar pequeñas variaciones sobre el algoritmo de procesado de audio, pero no es posible modificar el algoritmo de procesado.

3.1. UMA-8

Se ha visto que la incorporación de dispositivos de localización de fuentes sonoras en aplicaciones de detección de voz para asistentes software inteligentes aumentan su rendimiento si se realiza un conformado de haz (*beamforming*) que permita filtrar el ruido ambiente y mejorar la detección de los comandos de voz. Recientemente han aparecido en el mercado varias aplicaciones de este estilo, algunas de ellas utilizando el microcontrolador de XMOS, como Alexa de Amazon [13], u otra aplicación con UMA-8 utilizando para su control y alimentación una RasPi [14], y otras como MATRIX Creator [15], que utiliza el microcontrolador de Amtel Cortex-M3 ATSAM3S2, en todos estos casos dando mucha importancia al trabajo en tiempo real.

Como ya se ha comentado anteriormente, el dispositivo UMA-8 es un producto de la marca miniDSP, el cual tuvo su primera aparición en el mercado el 17 de mayo de este año 2017, tras la cual se han realizado dos actualizaciones del software hasta la fecha.

Para empezar, se va a dar una visión global del dispositivo UMA-8 que se puede ver en la figura 3.1 y se estudiará la información que se obtiene a través de las hojas de características del propio producto [3]. Además, en el anexo A, se detallan la placa y la disposición de sus elementos principales, así como una imagen del dispositivo donde se puede ver sus dimensiones.

El dispositivo UMA-8 se puede definir como un micrófono USB multicanal. Consta de una agrupación de siete micrófonos MEMS de alto rendimiento. Estos micrófonos se encuentran distribuidos en una placa circular, seis de ellos se encuentran en la parte externa repartidos como si fuesen los vértices de un hexágono y el séptimo se encuentra en el centro de la circunferencia. Este dispositivo se centra en la tecnología multicore XCORE, que ya se ha presentado en el capítulo 2, con la que se consigue un alto rendimiento con un bajo coste. Además, con el aprovechamiento del procesamiento DSP, el dispositivo UMA-8 admite algoritmos de procesado de audio, entre los cuales se incluye el conformado de haz, la reducción de ruido, la cancelación de eco acústico y la reverberación. También se incluyen unos drivers propios del dispositivo para su utilización en distintos sistemas operativos (Windows, macOS, Linux), de los que se hablará en el siguiente apartado.

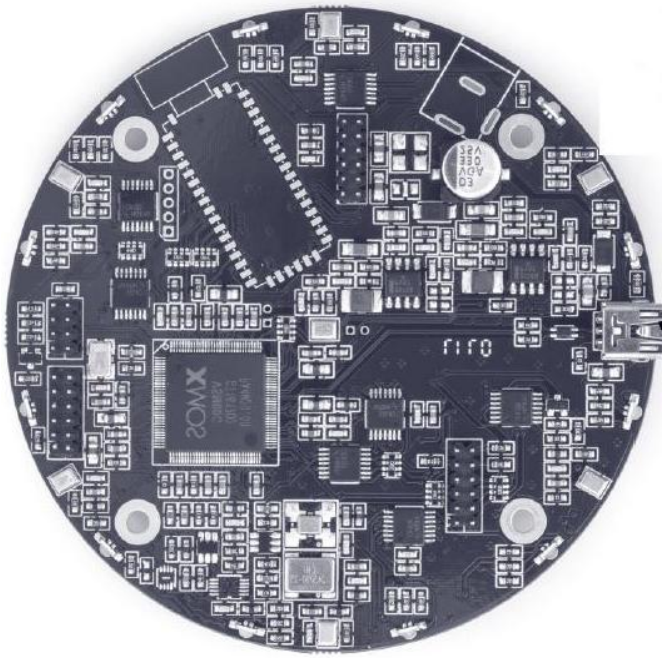

Figura 3.1. Dispositivo UMA-8 [3]

Como ya se comentó en el capítulo anterior, los micrófonos utilizados en el dispositivo UMA-8 son siete SPH1668LM4H, como se puede ver en la hoja de características de estos micrófonos se han elegido por su baja distorsión, buena SNR, bajo nivel de ruido y tener una comunicación digital de alto rendimiento.

Se pueden encontrar más características y más información sobre este tipo de micrófonos en el capítulo 2 o en su hoja de características [10].

Este producto de miniDSP está alimentado a través del puerto USB, el cual envía los datos de posición de la fuente sonora captada por los micrófonos. El sonido puede recogerse a través de los propios micrófonos como si se tratase de un micrófono portátil. Existen en la placa otras entradas, pero que no se utilizaran en nuestro trabajo. Además hay salidas I2S (*Integrated Interchip Sound*), que permitirán en futuras actualizaciones ser conectadas a un DAC (Conversor digital-analógico) para así reproducir el audio que capten los micrófonos directamente sobre unos altavoces con entradas digitales. En el anexo A se puede ver el puerto USB y la posición de estas salidas en la placa.

Como apunte final se puede decir que, en la versión actual de la placa, se dispone de dos modos de funcionamiento: el modo DSP y el modo RAW. En modo DSP la interfaz está funcionando con una configuración 2xIN (2ch beamforming), 2XOUT (salida estéreo en la salida I2S del pin J3.1). Para el modo RAW, en cambio, se podrán ver 8 entradas, 8xIN, que se corresponden con las señales los siete MEMS y además una entrada PDM que permite una futura ampliación de un micrófono más, y dos salidas, 2xOUT, que será como en el caso del modo DSP, la salida estéreo I2S que como se ha dicho antes se encuentra en el pin J3.1.

3.2. MicArray. Software UMA-8

El dispositivo UMA-8, dispone de dos programas de control propios. El último apareció después de la actualización del 6 de junio, pero se comenzará con el primero de ellos. Primero comentar que en la guía [4] de instalación están los pasos de instalación muy bien explicados para que no se produzca ningún error, y añadir además que en el momento en el que se conecta el dispositivo a través de cable USB se instala de forma automática. Como se ha dicho existen dos modos de trabajo que cambian modificando el firmware del dispositivo. El firmware del modo RAW es la última actualización hasta la fecha.

3.2.1 Modo DSP

Se comenzará explicando cómo funciona y se pone en marcha el "Modo DSP". Se comienza con la puesta en marcha, tal y como se especifica en la guía del dispositivo. Ésta puede hacerse tanto desde una máquina con sistema operativo Windows, Mac o Linux. Como primer paso es necesario instalar el contenido de la carpeta "WinDrivers", en esta carpeta al igual que ocurre con el resto de material esta descargado desde la página oficial de miniDSP [2]. Continuando en esta carpeta se pueden encontrar dos archivos para que se instale uno u otro según la actualización de Windows desde la que se vaya a trabajar. Como se ha mencionado, en la guía aparecen los puntos clave para poder instalar de forma correcta el dispositivo. Una vez la instalación sea correcta, nuestro dispositivo está configurado en "Modo DSP. Durante la realización de este proyecto en una primera aplicación se descubrió que una vez el driver estaba completamente instalado, desde el entorno openFrameworks únicamente se podía acceder a este dispositivo de audio, impidiendo trabajar con la tarjeta de sonido del ordenador u otra adicional, como la Behringer UCA-202 con la que se realizó alguna prueba. El problema se solucionó finalmente instalando manualmente alguno de los paquetes de este driver, pero únicamente para este modo y no para el "Modo RAW", pero como se explicará en el capítulo siguiente se acabó instalando el driver completamente.

Tras esto, se puede realizar la instalación de la aplicación a través de la cual se obtienen los datos o las especificaciones del sonido que se recibe. Se puede encontrar, como se ha dicho antes, el ejecutable tanto para Apple como para Windows en la carpeta que lleva el nombre de "Plugins", en este caso se instalará la versión para Windows. La aplicación instalada consta de dos pantallas, que se pueden observar en las figuras 3.2 y figura 3.3. Todos estos pasos se pueden encontrar en el manual del dispositivo UMA-8 [4].

Como se puede ver en la figura 3.2, en la esquina superior derecha vemos un icono con un *tick* verde y justo debajo del *tick* la palabra "*Connected*", de esta forma se puede comprobar si el dispositivo y la aplicación están conectados, y si no lo está es necesario conectase. En el resto de la pantalla se pueden ver otras características de las cuales algunas de ellas se podrán modificar para restringir de este modo el sonido que se desea captar, la información más detalla sobre cada una de estas características se puede encontrar en el manual del dispositivo UMA-8 [4].

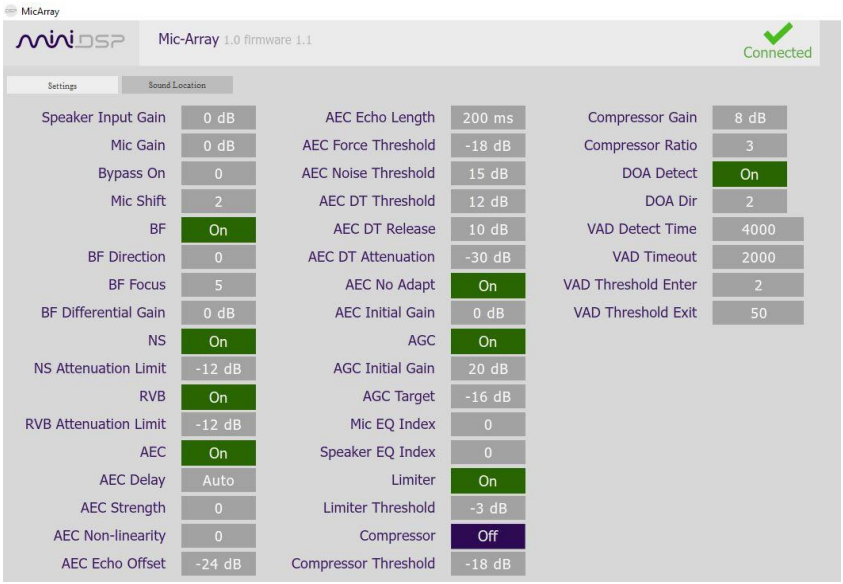

Figura 3.2. Aplicación MicArray, características del sonido

Lo que se puede ver en la figura 3.3, es la pantalla secundaria de la aplicación MicArray, en la cual aparece la disposición de los seis micrófonos. Es muy importante darse cuenta que la imagen está tomada como si se viese la placa desde abajo, ya que como se mencionó al hablar de estos micrófonos el sonido se recibe en la parte baja del dispositivo. Se puede ver también en la zona cercana al micrófono 3 unos círculos naranjas, estos aparecerán cerca del micrófono por el que se está recibiendo el sonido. Esta pantalla es simplemente una visualización, si se observa la placa en el momento en el que se recibe un sonido se puede ver que se encienden los leds más próximos al micrófono por el que se estima la llegada del sonido.

Esta aplicación únicamente se utilizó muy al principio del proyecto, para visualizar las posibilidades que daba el dispositivo, y ver la velocidad de reacción a la hora cambiar la fuente sonora rápidamente. En el capítulo siguiente se podrá ver el uso que se le ha dado finalmente al modo DSP en este proyecto.

Figura 3.3. Aplicación MicArray, localización de sonido.

3.2.2. Modo RAW

El dispositivo UMA-8 tiene una última actualización que se denominará "Modo RAW", que se puede encontrar en la carpeta "XMOS_Firmware", la cual se puede realizar de forma muy sencilla siguiendo los pasos que se encuentran en el manual del dispositivo [4]. Añadir que esta actualización únicamente puede hacerse desde un sistema operativo Windows, pero que una vez el dispositivo este actualizado es reconocido y se puede usar desde cualquier otro sistema operativo. Con esta aplicación se consigue captar la información de todos los micrófonos de forma simultánea, disponiendo así de un sistema de captura de audio multicanal. Las velocidades de muestreo a las que pueden trabajar son 11.2, 16, 32, 44.1 y 48kHz. Hay que tener en cuenta que en el modo RAW el dispositivo UMA-8, emitirá la señal RAW de los micrófonos MEMS sin ninguna ganancia digital (normalmente aplicada en modo 2ch DSP). Es posible que al trabajar con este modo sea necesario aplicar una determinada ganancia a las señales capturadas amplificándola desde la aplicación que se esté desarrollando.

Capítulo 4

Software para el UMA-8

Este capítulo se centra en el funcionamiento de las aplicaciones diseñadas para el dispositivo UMA-8. Se comentan algunas opciones que se tuvieron en cuenta hasta llegar al resultado final. Desde un principio, se decidió utilizar el IDE de desarrollo Visual Studio 2015 [5], utilizando el entorno openFrameworks [6]. Finalmente, se acabó utilizando tanto el entorno openFrameworks, como Visual Basic [7]. A lo largo del capítulo, se habla de la tarea que realizará cada uno y cómo es la comunicación entre ellos.

En este capítulo se diferencian dos secciones, en cada una se habla de la aplicación realizada para cada modo (modo DSP y modo RAW), ya que como se ve más adelante la lectura y cálculo de datos se hacen de formas diferentes.

4.1. Modo DSP

En el "modo DSP", como se habló en el capítulo anterior, se utiliza únicamente un micrófono como entrada de audio, éste será el más cercano a la fuente sonora captada. Como se comentó anteriormente, el dispositivo envía en este caso a través del USB el ángulo detectado y el micrófono activo en ese momento. Para la utilización de este modo de trabajo se han creado dos aplicaciones distintas de las que se habla a lo largo del capítulo, y que se comunican entre sí, con lo que se consigue, con el uso de la cámara de un ordenador localizar la fuente sonora activa en ese momento. Hay que tener en cuenta que si se detecta más de una fuente sonora únicamente se reconocerá la que se reciba con mayor intensidad.

Para trabajar con este modo del dispositivo UMA-8, se han utilizado los entornos Visual Basic, que se encarga de la recepción de datos a través de USB, y openFrameworks, que realiza la síntesis de los datos y los representa en un entorno gráfico. Al trabajar con dos entornos distintos de IDE Visual Studio se necesita realizar una comunicación entre las dos

aplicaciones. Para ello, se estudió las distintas posibilidades, y se decidió utilizar mensajes UDP (*User Datagram Protocol*) en vez de TCP (*Transmission Control Protocol*) ya que no necesita recibir mensajes de confirmación (ACK) en cada envío de datos. De esta forma, si un paquete se pierde la comunicación no se bloqueará, por lo que se puede seguir trabajando a tiempo real con un determinado retardo. Además, se decidió prescindir del control de errores en la comunicación, ya que esto aumentaría el retardo. Dentro de la comunicación UDP, se pensó utilizar comandos OSC (*Open Sound Control*) [21], ya que está muy desarrollado en los dos entornos, pero finalmente se decidió mandar mensajes UDP estándar ya que requieren menor cabecera por lo que la cantidad de datos enviados es menor. Se puede ver un pequeño esquema de la comunicación en la figura 4.1.

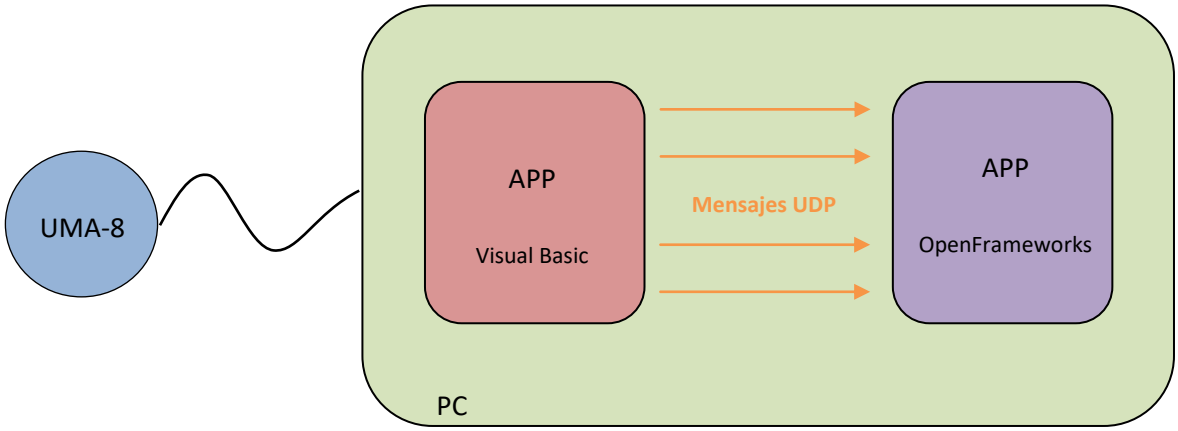

Figura 4.1: Diagrama de la comunicación, desde UMA-8 hasta su aplicación en openFrameworks.

4.1.1. Visual Basic

La aplicación desarrollada en Visual Basic se ha realizado utilizando el IDE Visual Studio 2015. Esto permitía tener el mismo IDE tanto para el desarrollo mediante Visual Basic y openFrameworks. Este entorno se ha utilizado para realizar el reconocimiento del dispositivo a través del USB, el cual desde openFrameworks, a pesar de probar y gestionar diferentes librerías, dio bastantes problemas. Para realizar este desarrollo se ha utilizado una plantilla [11]. A partir de ella, se ha desarrollado una sencilla aplicación con cinco botones.

El funcionamiento de la aplicación es muy sencillo, se puede ver su interfaz en la figura 4.3. En la parte superior aparece un mensaje, USB desconectado o USB conectado,

acompañados de un indicador rojo o verde respectivamente. El cambio entre estas dos opciones será automático en el momento en que el dispositivo se conecte, esto se debe a que reconoce un VID (&H2752) y PID (&H1C) propios de UMA-8. ES muy importante recalcar que estos identificadores del dispositivo con propios del "modo DSP" ya que si estuviera en "modo RAW" el PID cambiaría. En la figura 4.2, se puede ver un diagrama de estados, que explica el código, el cual se puede encontrar en el anexo C.

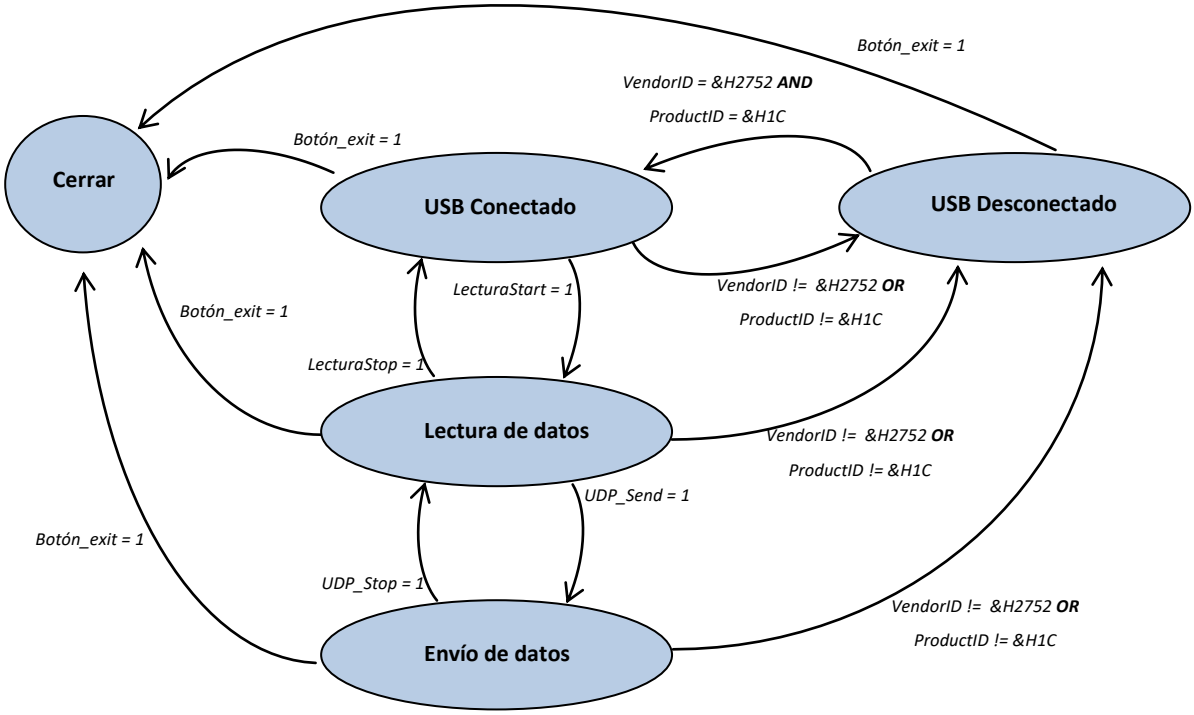

Figura 4.2: Diagrama de estado Visual Basic.

En el diagrama de estados, de la figura 4.2, se pueden ver cuatro estados, de los que se habla a continuación. Durante la explicación de estos estados se mencionarán botones de la aplicación que se pueden identificar en la figura 4.3.

- **USB Desconectado**: Permanece en este estado hasta que se reconocen los identificadores del USB. Desde este estado, únicamente es posible ir al estado "USB Conectado", si se reconocen los identificadores de UMA-8, o al estado "Cerrar" cuando se pulse el botón de apagado. A este estado es posible acceder desde cualquiera, excepto desde el de "Cierre", en el momento que UMA-8 no esté conectado.

- **USB Conectado**: En el momento en que se reconozcan los identificadores del dispositivo se accede a este estado. Desde él, se puede llegar a "Cierre", al pulsar el botón de apagado, a "USB Desconectado", como se ha comentado en el punto anterior, y a "Lectura de dato", al pulsar el botón de "LecturaStart".
- **Lectura de datos**: Se produce a través de un evento, el cual, lee los datos del USB cada vez que estén disponibles. Desde este estado como ocurría en el anterior, es posible llegar a "USB Desconectado" y a "Cierre", además si se pulsase el botón "LecturaStop" se retrocede a "USB Conectado" y si se pulsa "UDP_Send" se cambia al estado "Envío de dato". Comenzando así el intercambio de mensajes con la aplicación desarrollada en openFrameworks. Más adelante se habla de cómo son los datos que se leen a través del USB.
- **Envío de datos**: En el momento que se llega a este estado comienza el envío de mensajes UDP a la aplicación de openFrameWorks. Desde este estado, se puede llegar a "Cierre" y a "USB Desconectado", como en el caso anterior. Además, al pulsar el botón "UDP_Stop" se vuelve al estado "Lectura de datos" y se cancela el envío de mensajes UDP.
- **Cierre**: A este estado se accede al pulsar el botón de apagado que se encuentra en la esquina inferior derecha, figura 4.3, una vez se llegue a este estado la aplicación se cerrará.

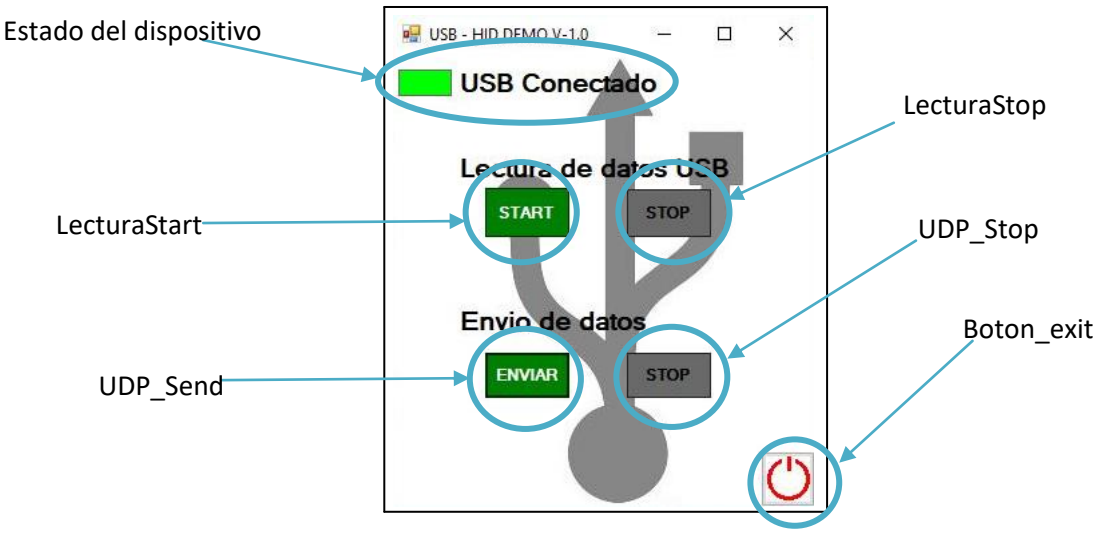

Figura 4.3: Distribución de los botones.

Esta aplicación envía cada vez que recibe datos un buffer de siete bytes, de los cuales los tres primeros son constantes por lo que dan información de control. El cuarto byte varía de

cero a uno o de uno a cero en cada nuevo mensaje, este byte será utilizado para comprobar que el sonido recibido desde una posición concreta sigue produciéndose viendo el cambio en cada repetición. Y por último los tres últimos bytes que nos indican la posición del sonido, los bytes cinco y seis nos dan el ángulo del que proviene el sonido y el último byte recibido indica que micrófono lo está recibiendo. Esta disposición de los datos se puede ver más clara en la figura 4.4, que representa la parte trasera del dispositivo, ella se encuentra la disposición de ángulos, el micrófono al que pertenece y ejemplos de tramas de datos.

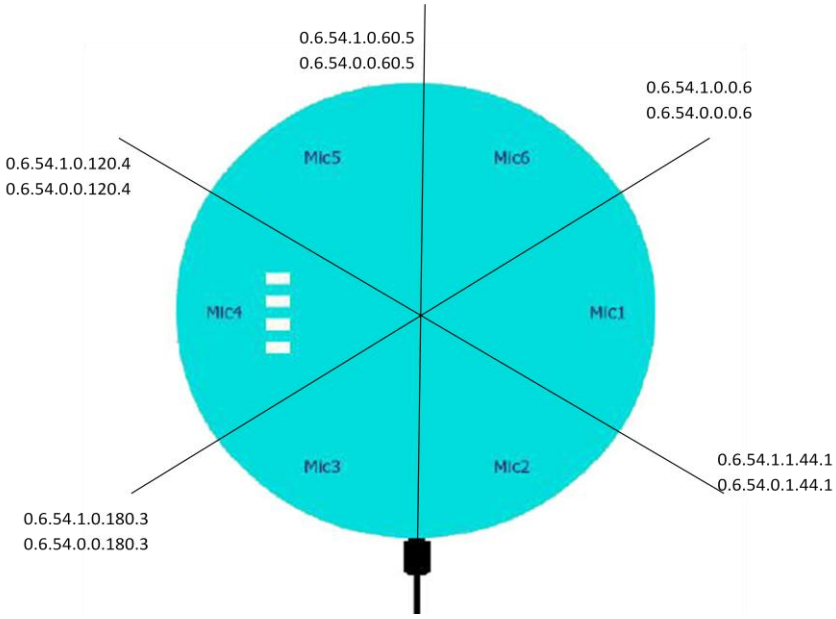

Figura 4.4: Distribución de los datos [4]

4.1.1. OpenFrameworks.

A lo largo de este punto, se hablará de cómo funciona el entorno gráfico de la aplicación. Desde un principio se eligió openFrameworks, ya que proporciona muchas posibilidades a nivel gráfico. Como se explicó anteriormente, la aplicación recibirá los mensajes UDP desde la aplicación de Visual Basic, como se puede ver en la figura 4.1.

Esta aplicación, trabajará con dos tipos de datos, los que viene de la comunicación con Visual Basic y lo procedentes de los micrófonos. El mensaje UDP que se recibe contiene, como se mencionó en el punto anterior, información de posición, tras un estudio de los resultados, que aparece en el capítulo cinco, únicamente se trabajará con el byte que indica qué micrófono está activo en ese momento. Por otro lado, se recoge en un buffer el sonido

recibido por la agrupación de micrófonos, en este caso el audio estéreo de uno de ellos, a partir de ellos se realizará una representación del espectro del sonido.

A continuación, en la figura 4.5, se ve el diagrama de estados, en el que se explica el funcionamiento de la aplicación, pudiendo encontrar en el anexo D, el código completo. Como se puede ver aparecen seis estados que se explican brevemente:

- **Recepción audio**: este estado engloba todas tareas encargadas de trabajar con el sonido recibido desde la agrupación. Se utiliza para ello funciones y clases propias del entorno, como "ofSoundStream". A partir del audio recibido se realizará la representación del mismo, como se explicará en el estado siguiente.
- **Representación audio**: se utiliza para representar las muestras recibidas en cada momento. Esta representación solo será visible si está activo "controlFFT", si el "botonFFT" ha sido activado.
- **Pantalla principal**: dentro de este estado puede ocurrir dos cosas, que la cámara se encuentre activa o no, en ambos modos se podrá ver la representación del micrófono activo por la fuente sonora.
- **Pantalla secundaria, PANDA**: en esta pantalla aparece la representación de un panda, el cual, con la mirada sigue la localización de la fuente sonora.
- **Recepción de datos**: representa la recepción de datos a través de los mensajes UDP, y el tratamiento de estos una vez recibidos. Únicamente se utilizará el identificador del micrófono activo, dibujando la sección afectada en la pantalla principal o la dirección de la mirada en la pantalla secundaria.

La interfaz gráfica de esta aplicación consta de tres botones que controlan lo que se quiere visualizar, el paso de un estado a otro. En la figura 4.6 se puede ver la pantalla principal en el momento en el que percibe una fuente sonora.

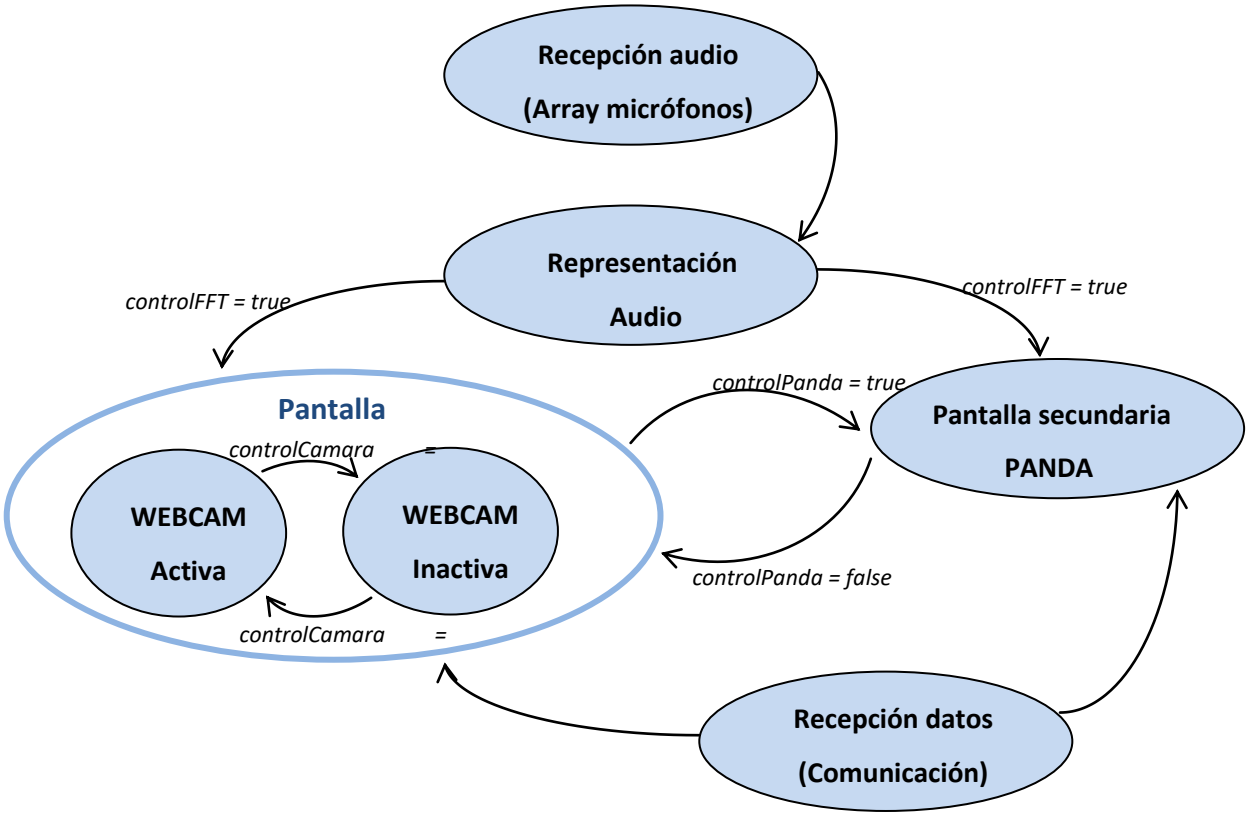

Figura 4.5: Diagrama de estados openFrameworks

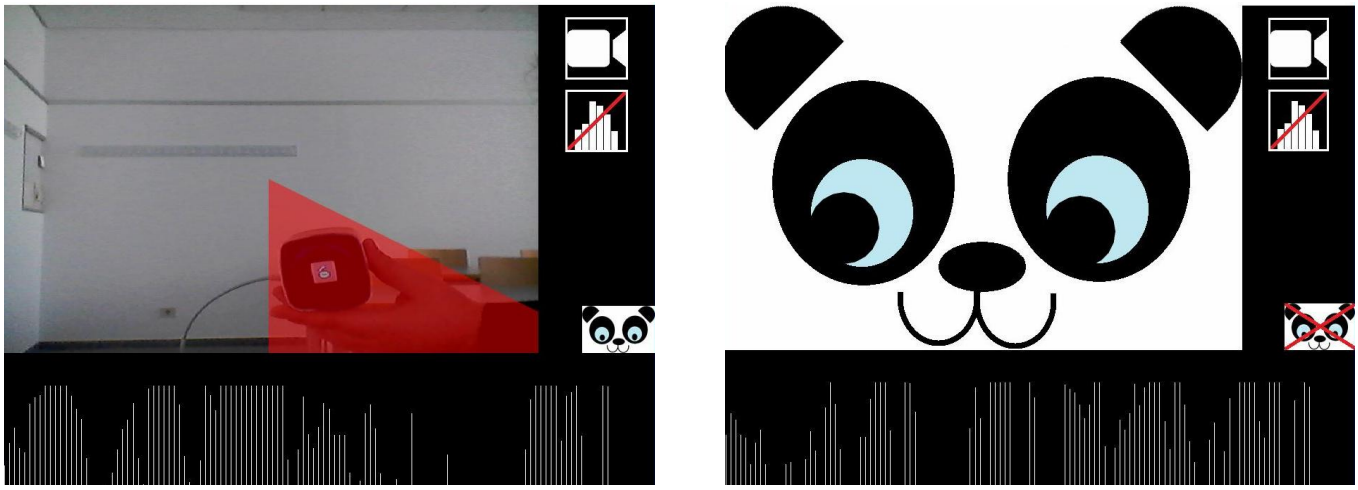

Figura 4.6: Interfaz openFrameworks para el "modo DSP", distribución de botones

4.2. Modo RAW

A lo largo de este apartado se habla sobre el segundo modo de uso del dispositivo UMA-8, el "modo RAW". Esta actualización apareció a principios del mes de junio, y como se

explica en el tercer capítulo, trabaja con todos los micrófonos de forma simultánea. Con relación a este proyecto, esta actualización fue muy interesante, ya que como se verá en siguiente capítulo, los resultados espaciales obtenidos con el "modo DSP" no son precisos. Para trabajar con el "modo RAW" se ha implementado una aplicación, desde la cual se recogerán los datos captados por los seis micrófonos externos, con los que se trabaja conseguir la localización de la fuente sonora y, se realizará un algoritmo de localización de fuentes sonoras.

Para el "modo RAW" se ha implementado una única aplicación a través del entorno openFrameworks. El objetivo de esta aplicación es representar un mapa de potencias estimadas a partir de técnicas basadas en *beamforming* superpuesto a la imagen procedente de la cámara, en este caso la webcam del ordenador. Las técnicas de *beamforming* se utilizan para localizar una dirección determinada de un conjunto de datos recibidos procedentes de una agrupación de micrófonos. Una vez se identifica la dirección se estima su potencia y aplicando distintos filtros se obtiene un mapa de potencias, esta técnica se conoce como SRP. En un Trabajo Fin de Master, realizado por David Díaz-Guerra [1], se escogió, en función de sus ventajas y desventajas, el algoritmo SRP-PHAT (*Steered Response Power with the PHAse Transform*), que es capaz de combinar las técnicas SRP con la resolución que permite obtener la transformación PHAT con un coste computacional aceptable.

Este algoritmo puede escribirse en términos de GCCs (*Global Command and Control System*), de forma que la transformación utilizada en la GCCs es igual al producto de los filtros usados en el *beamforming* para cada señal:

$$P(\theta_0) = \int_{-\infty}^{+\infty} \left| \sum_{n=1}^{M} G_n(\omega) X_n(\omega) e^{-j\omega\,\tau_n(\theta_0)} \right|^2 d\omega = 2\pi \sum_{n=1}^{N} \sum_{n=1}^{N} R_{nm}\left(\Delta\tau_{nm}\ (\theta_0)\right)$$

$$\psi(\omega) = G_n(\omega) G_m^*(\omega)$$

Este algoritmo puede implementarse con un coste computacional relativamente bajo realizando las FFTs de las señales capturadas por cada micrófono, aplicando la transformación de fase y normalizando el módulo de cada *bin* frecuencial, multiplicando todas entre todas y realizando la inversa de FFTs para obtener las correlaciones cruzadas. De esta forma se consigue estimar la potencia de cada dirección, sumando las componentes adecuadas de cada

GCC. La componente que se buscaba estará determinada por $\Delta\tau_{nm}(\theta_0)$. Según el estudio realizado en el proyecto anterior lo resultados son satisfactorios si se elige la muestra más cercana sin necesidad de interpolar debido a que se trabaja con una agrupación de micrófonos circular. En la figura 4.7 se puede observar un mapa de potencias obtenido mediante la utilización de este algoritmo.

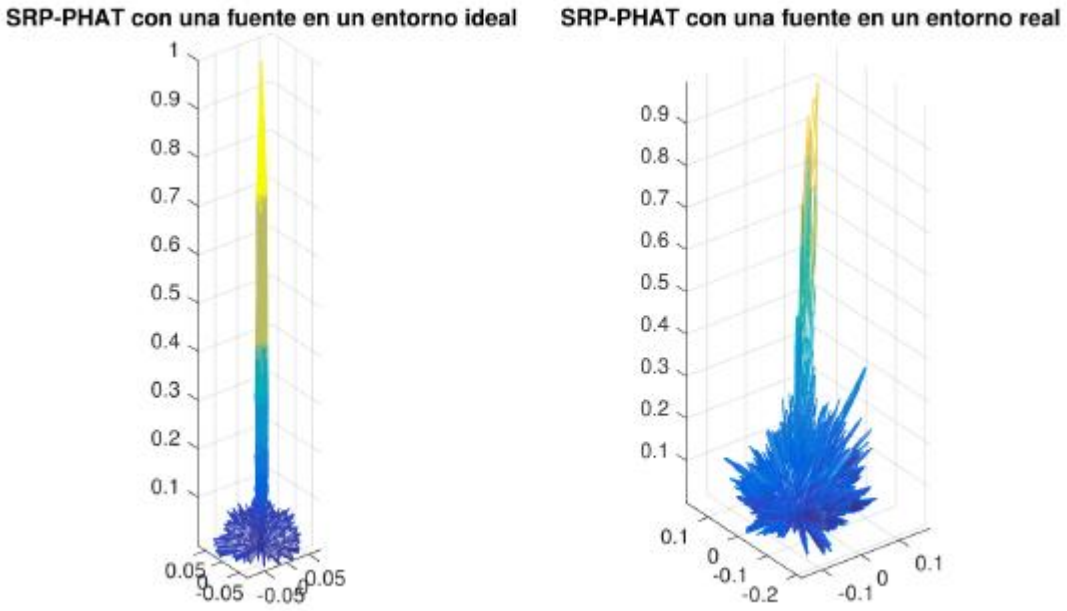

Figura 4.7: Mapa de potencia estimada mediante el algoritmo SRP-PHAT, en el lado izquierdo par a un entorno ideal y en la derecha para un entorno real [1]

Para implementar este algoritmo en el entorno openFrameworks se ha utilizado una función realizada para el Trabajo Fin de Máster [1] al que se hace referencia anteriormente. Para ver la resolución real del algoritmo es muy importante que el mapa de potencias que se crea se superponga correctamente al vídeo, intentando evitar los desplazamientos. Para conseguirlo, es necesario calcular las coordenadas esféricas de cada punto de la imagen. Por otro lado, la forma de conseguir que el mapa de potencias se superponga correctamente con la imagen del vídeo se consigue calculando previamente el valor de las coordenadas de cada punto de la imagen, para que posteriormente únicamente haya que calcular la potencia y aplicar el algoritmo para esos puntos.

Este algoritmo se ha implementado en el entorno openFrameworks y se ha realizado una aplicación de la que se puede encontrar ver su diagrama de estados en la figura 4.8. Desde esta aplicación se consiguen visualizar los canales que se deseen, la imagen procedente de la webcam y el mapa de potencias del sonido recogido por la agrupación, para lo cual se utiliza el algoritmo de SRP_PHAT.

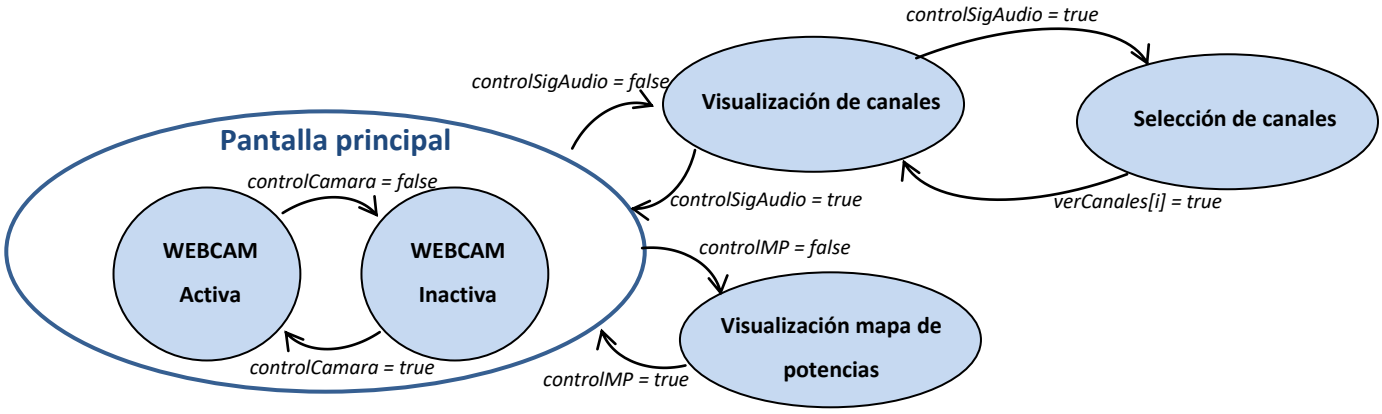

Figura 4.8: Diagrama de estado, aplicación "modo RAW"

En la figura 4.8, se explica mediante un diagrama de estados como funciona la aplicación, a continuación, se comentan brevemente todos ellos:

- **Pantalla principal**: dentro de este estado se pueden encontrar los estados de webcam activa e inactiva, desde los cuales se controla la visualización de las imágenes captadas por webcam o no.
- **Visualización de canales y selección de canales**: los canales de audio se pueden visualizar al pulsar el botón que los controla. Una vez activado este botón aparecerán siete botones que controlar la visualización de cada una de las señales recogidas por la agrupación.
- **Visualización mapa de potencias**: con este modo activado se visualizará el mapa de potencias superpuesto a la imagen de vídeo, en el caso de que este no esté activo no se podrá visualizar el mapa de potencias.

Una vez explicado el funcionamiento se puede encontrar el código completo en el anexo E, no se ha incluido en este anexo el código del algoritmo SRP_PHAT ya que no es de creación propia. Se puede ver el entorno gráfico en la figura 4.9, donde se puede ver la

distribución de los botones, además se podrán ver los resultados obtenidos en el capítulo siguiente.

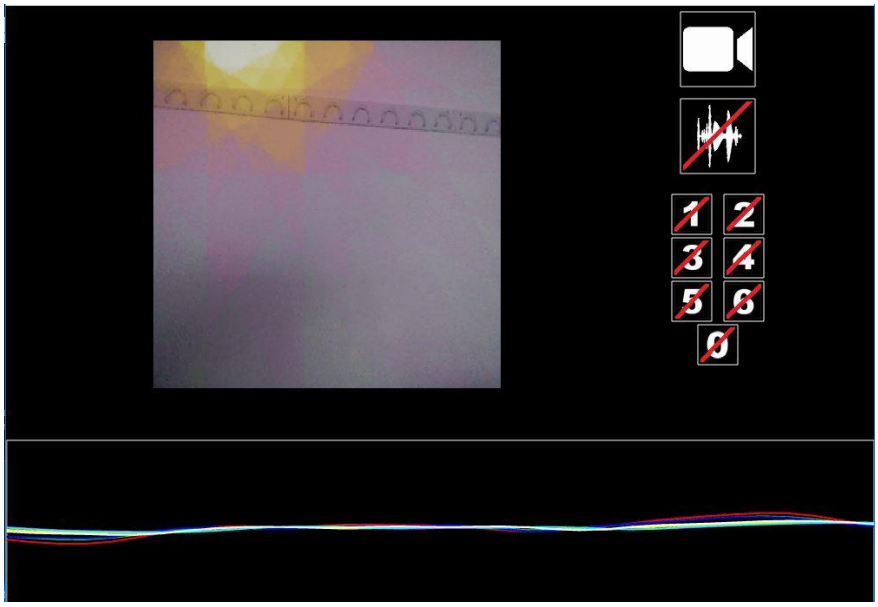

Figura 4.9: Interfaz openFrameworks para el "modo RAW", distribución de botones

Capítulo 5

Resultados obtenidos

En este capítulo se explica los resultados obtenidos para los dos modos de uso de la placa, se comenza con el modo DSP continuando con el modo RAW. Para la realización de todos los experimentos, se ha querido minimizar el error existente entre la posición de la agrupación y la cámara, por lo que se decidió colocar la agrupación sobre la cámara minimizando la distancia entre ellas, ya que la posición ideal sería la cámara en el centro de la agrupación. Se puede ver la posición de la agrupación en la figura 5.1.

Figura 5.1: Posición del dispositivo UMA-8 para su utilización

5.1. Modo DSP

Para este modo de uso del dispositivo UMA-8, se añadieron en el software de Visual Basic los comandos necesarios para poder escribir los datos recibidos a través de USB en un archivo de texto. De esta forma, se podía evitar la posible pérdida de paquetes de la

comunicación UDP, ya que como se explica en el cuarto capítulo, se ha decidido no hacer corrección de errores para no incrementar el retraso entre la imagen y el sonido.

Para realizar el estudio de los resultados obtenidos se decidió hacer diferentes pruebas, de las que se hablará a continuación. Para estas pruebas se utilizaron tres tipos de sonidos, ruido blanco, una melodía y ruidos cortos, como podrían ser los tonos de un teléfono. Para todas las pruebas que se realizaron se vio que los sensores trabajan mejor detectando sonidos cortos desde silencio. En el caso del ruido blanco, únicamente se detecta la posición del sonido en el momento en el que se activa la fuente por primera vez, si se modifica su posición sin detener el sonido el microcontrolador descarta la información. Por otro lado, en las pruebas realizadas con la melodía, se recogían más datos a través del USB, pero si la melodía no presenta ningún cambio brusco, aunque se modifique su posición, no siempre es detectada.

Por todo lo comentado anteriormente, los resultados representados a continuación son los realizados con audios de corta duración con silencios entre cada tono. La primera prueba realizada es la modificación de la posición de la fuente sonora en todo el rango de posibles valores que se reciben a través del USB. Para realizar este experimento se fue modificando la posición angular de la fuente sonora a una distancia de la agrupación de micrófonos de un metro aproximadamente. Al estudiar de los datos obtenidos se pudo ver qué posiciones están controlados por cada micrófono, por lo que, realizando las pruebas manualmente, se consideró acierto si la posición donde se coloca la fuente activa el ángulo correcto. Se realizaron 20 mediciones por cada posición, consiguiendo un total de 240 muestras. A continuación, en la tabla 5.2, se representa el porcentaje de muestras recibidas por cada micrófono en cada posición. En la tabla 5.2, se encuentra remarcado en verde los porcentajes máximos en cada posición, comprobando que todos ellos se encuentran en el micrófono correcto.

A partir de los resultados de la tabla 5.2, se puede representar la probabilidad de acierto y error, según la posición donde se encuentre la fuente sonora, como se representa en la figura 5.3. Se considera acierto que se active el micrófono él cual controla la posición donde se encuentra la fuente, y fallo la activación de cualquier otro micrófono. Se puede ver que los resultados obtenidos a esa distancia son muy buenos, sufriendo en el peor caso un 20% de error.

		PORCENTAJE DE ACIERTO Y FALLO											
		Posición donde se encuentra la fuente [°]											
		0	30	60	90	120	150	180	210	240	270	300	330
Micrófono	6	100,00	90,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	10,00
	5	0,00	10,00	100,00	100,00	10,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	4	0,00	0,00	0,00	0,00	90,00	100,00	0,00	10,00	0,00	0,00	0,00	0,00
	3	0,00	0,00	0,00	0,00	0,00	0,00	100,00	80,00	10,00	0,00	0,00	0,00
	2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	10,00	90,00	100,00	10,00	0,00
	1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	90,00	90,00
% por pos.		100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00

Tabla 5.2: Porcentaje de aparición de cada micrófono según la posición de la fuente sonora, en el modo DSP.

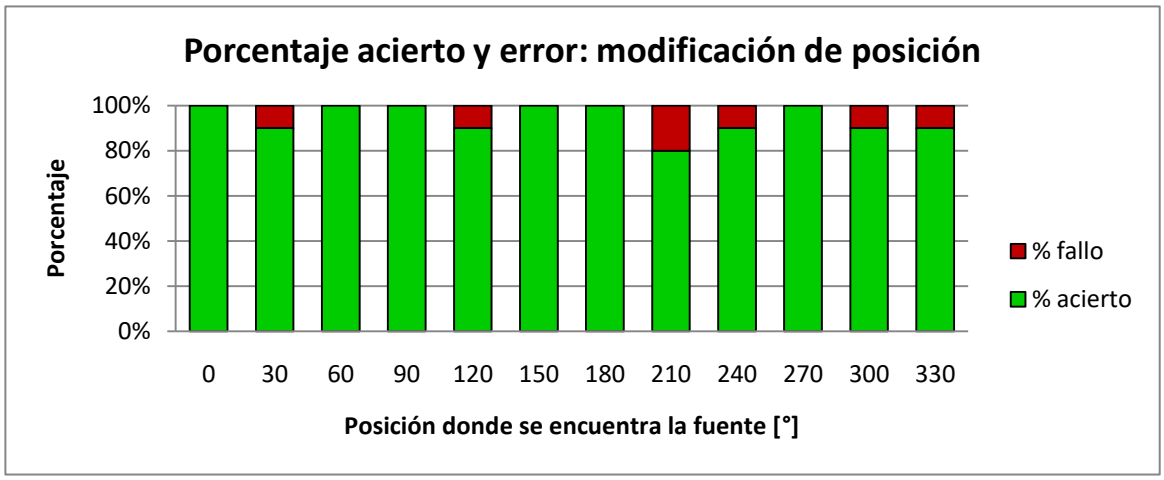

Figura 5.3: Porcentaje de acierto y error según la posición angular de la fuente sonora, en el modo DSP.

Se ha realizado otro experimento, esta vez teniendo en cuenta distancia entre la fuente sonora y la agrupación de micrófonos. Para ello, se cola una fuente fija y tras la recepción de 20 muestras se modifica la distancia de esta con respecto a la placa, únicamente se modifica la distancia con respecto a agrupación y no su posición, para esta prueba se ha realizado un barrido desde 0 a 4 metros utilizando únicamente el micrófono 1, modificando la posición medio metro cada vez. En cada posición se han tomado 20 muestras, recogiendo un total de 200 muestras. En la tabla 5.4, se presentan los resultados obtenidos en porcentaje

para cada posición. Se ha calculado también el porcentaje de aparición de cada micrófono en todo el barrido.

		PORCENTAJE DE ACIERTO Y FALLO									
		Distancia entre la fuente y la agrupación [m]									
		0	0,5	1	1,5	2	2,5	3	3,5	4	TOTAL %
Micrófono	6	0,00	20,00	20,00	30,00	70,00	80,00	40,00	60,00	20,00	37,78
	5	0,00	0,00	0,00	0,00	0,00	10,00	0,00	10,00	70,00	10,00
	4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	10,00	0,00	1,11
	3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	10,00	1,11
	2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	1	100,00	80,00	80,00	70,00	30,00	10,00	60,00	20,00	0,00	50,00
% por posición		100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00

Tabla 5.4: Porcentaje de aparición de cada micrófono según la distancia en metros entre la fuente y los micrófonos, en el modo DSP.

Lo que se puede ver en la tabla 5.4, es que, como era esperado, la probabilidad de fallo, de que no se active el micrófono 1 que se ha elegido como micrófono de referencia, aumenta. Al aumentar la distancia entre sensor y fuente sonora, se selecciona el micrófono según la posición global, en este caso se podría decir que estaba “alto”, por eso se detecta el micrófono 6, que se encuentra sobre el micrófono 1, vemos que si se toma como acierto la posición en la que se recogen más muestras se puede decir que el error en un rango de 0 a 4 metros no es tan grande. De esta forma y a partir de los datos de la tabla 5.4, se obtiene la figura 5.5, donde se representan el porcentaje de acierto y error en función de la distancia que separa la fuente sonora y la agrupación. En este caso se considera acierto que se active el micrófono 1, ya que es la referencia elegida y error que se active cualquier otro. En la figura 5.5, se puede ver, como ya se ha mencionado, que el porcentaje de error a distancias mayores de 2 metros es muy alto, por lo que se podría concluir que este dispositivo está diseñado para distancias cortas.

Las pruebas que se han realizado tienen como resultado el valor del micrófono que se lee a través del USB, como se menciona en el capítulo 3, además del número del micrófono afectado también se indica que posición es recibida. En estas pruebas se ha decidido no utilizar dicho valor ya que, únicamente puedes recibir 12 posiciones de 360 posibles, un 3,33%, es por esta baja exactitud que se ha decido dar los resultados por micrófono, en vez por ángulo.

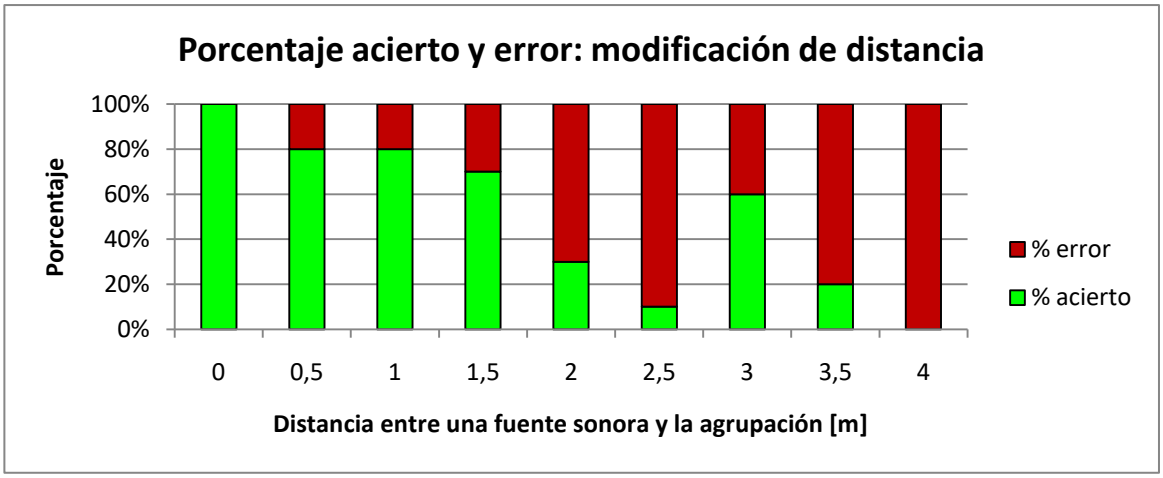

Figura 5.5: Porcentaje de acierto y error según la distancia entre la fuente sonora y la agrupación, en el modo DSP.

5.2. Modo RAW

El modo RAW, como se ha mencionado anteriormente, consta de ocho entradas de audio de las cuales se utilizan para la localización de la fuente sonora únicamente las seis que hacen referencia a los micrófonos periféricos, ya que la clase que se utiliza está preparada para trabajar con agrupaciones circulares.

Las pruebas que se han realizado para este modo de uso son las mismas que para el modo DSP. Antes de la realización de estas pruebas se ha estudiado con qué tipo de sonido se obtenían mejores resultados. En la figura 5.6 y la figura 5.7, se muestran dos imágenes en la que se puede observar el mapa de potencias superpuesta a la imagen procedente de la cámara. La figura 5.6 muestra la recepción de un sonido de baja frecuencia, y la figura 5.7, la recepción de un sonido de alta frecuencia. Se puede ver en la figura 5.7, que el foco sonoro se detecta con bastante exactitud, en comparación con la figura 5.6, en la que no se consigue marcar la zona correctamente. Es por todo esto por lo que para la realización de todos los experimentos se han utilizados sonidos con frecuencias altas, ya que el mapa de potencias tiene mayor resolución. De esta forma, podemos concluir que por la utilización de estos micrófonos y dada su baja sensibilidad no son demasiado exactos con una señal de voz, para conseguir un buen resultado en este ámbito se necesitaría realizar un *beamforming* más selectivo para ese rango de frecuencias.

Figura 5.6: localización, modo RAW con baja frecuencia.

Figura 5.7: localización, modo RAW con alta frecuencia.

Se realizaron pruebas con distintos sonidos dando como resultado, que a una distancia cercana los resultados son muy satisfactorios, se consigue con sonidos de frecuencias altas una buena resolución de la malla. Previamente, se había pensado que por la distancia a la que están los micrófonos, no se iban a conseguir buenos resultados, ya que se pensaba que los canales se iban a mezclar, pero siempre que se trabaje con una frecuencia adecuada los resultados son bastante exactos. Es por esto, que para repetir las pruebas que se realizaron para el modo DSP, se ha elegido un sonido de frecuencia alta. En la tabla 5.8, se encuentra los resultados obtenidos al modificar la posición angular a una distancia de 1 metro de la agrupación, para obtener que micrófono se activa en cada momento, para realizar estas mediciones se añade una función que representa en la imagen procedente de la cámara que zona controla cada uno de los micrófonos y visualmente se va recogiendo los datos.

Si se observa en la tabla 5.8, como también ocurre en el modo DSP, tabla 5.2, que en cada posición el porcentaje de activación es más alto en micrófono que controla dicha zona. Se representa en la figura 5.9, el porcentaje de acierto y fallo en cada posición, donde se considera acierto que el micrófono de activación sea el más cercano, y fallo cualquier otro.

		PORCENTAJE DE ACIERTO Y FALLO											
		Posición donde se encuentra la fuente [°]											
		0	30	60	90	120	150	180	210	240	270	300	330
Micrófono	6	85,00	90,00	40,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	30,00
	5	0,00	10,00	60,00	90,00	15,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	4	0,00	0,00	0,00	10,00	85,00	100,00	0,00	0,00	0,00	0,00	0,00	0,00
	3	0,00	0,00	0,00	0,00	0,00	0,00	100,00	80,00	10,00	0,00	0,00	0,00
	2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	20,00	100,00	100,00	10,00	0,00
	1	15,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	100,00	70,00
% por pos.		100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	110,00	100,00	110,00	100,00

Tabla 5.8: Porcentaje de aparición de cada micrófono según la posición de la fuente sonora, en el modo RAW.

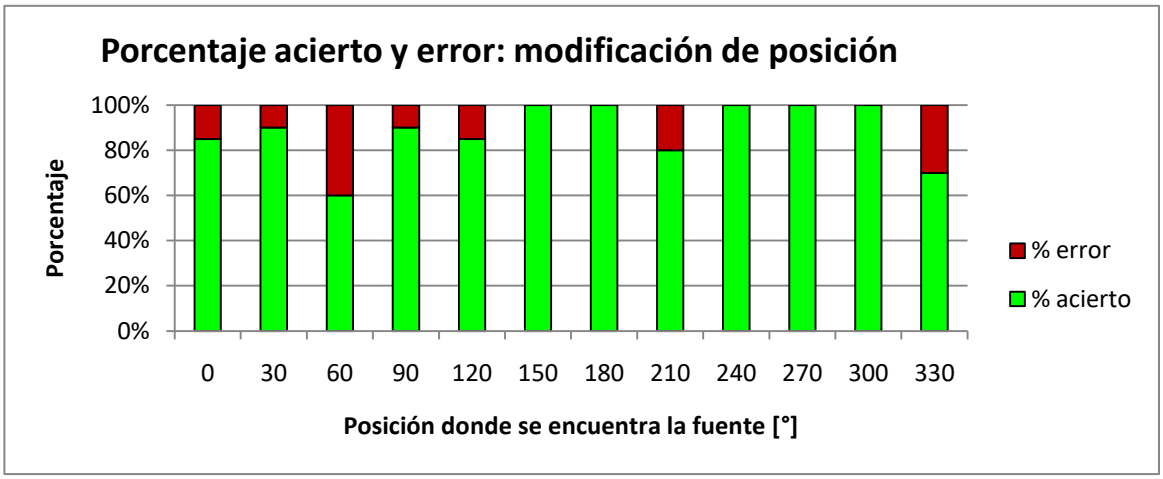

Figura 5.9: Porcentaje de acierto y error según la posición angular de la fuente sonora, en el modo RAW.

Aparentemente los resultados obtenidos son peor que en el caso del modo DSP, figura 5.3, pero esto se debe a que las mediciones no son numéricas y se hacen de forma visual. Ya que si esta prueba se realiza moviendo la fuente sonora se puede ver que en el modo RAW se consigue seguir la fuente con un error muy pequeño, y con el modo DSP es más costoso.

Por otro lado, si se realizan pruebas modificando la distancia entre la agrupación y la fuente sonora, tabla 5.10, se ha intentado realizar el mismo barrido que en el caso del modo DSP, tabla 5.4, pero los resultados, en este caso en distancias mayores de 3 metros

manteniendo el volumen constante, no son concluyentes. La fuente sonora se ha dejado fija apuntando al micrófono 1, cuando se encuentran agrupación y fuente a cero metros.

		Distancia entre la fuente y la agrupación [m]									
		0	0,5	1	1,5	2	2,5	3	3,5	4	TOTAL %
Micrófono	6	0,00	20,00	40,00	55,00	65,00	25,00	30,00	-	-	33,57
	5	0,00	0,00	0,00	0,00	0,00	15,00	0,00	-	-	2,14
	4	0,00	0,00	0,00	0,00	0,00	50,00	60,00	-	-	15,72
	3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	-	-	0,00
	2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	-	-	0,00
	1	100,00	80,00	60,00	45,00	35,00	10,00	10,00	-	-	48,57
% por posición		100,00	100,00	100,00	100,00	100,00	100,00	100,00	0,00	0,00	100,00

Tabla 5.10: Porcentaje de aparición de cada micrófono según la distancia en metros entre la fuente y los micrófonos, en el modo RAW.

A partir de la tabla 5.10, se ha realizado una gráfica, figura 5.11, donde se muestra el porcentaje de acierto y error, donde se considera acierto la activación del micrófono 1, y fallo si se activa cualquier otro.

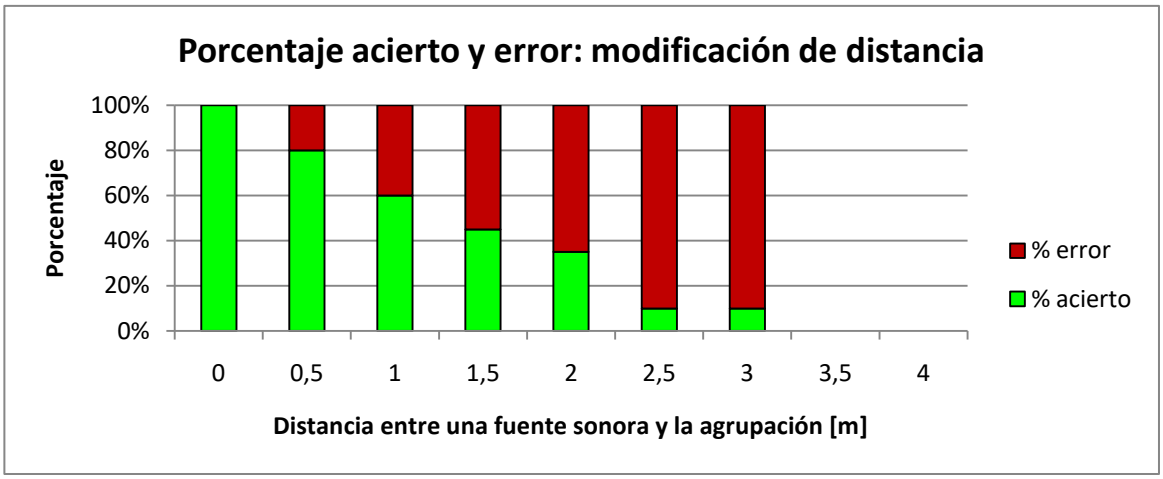

Figura 5.11: Porcentaje de acierto y error según la distancia entre la fuente sonora y la agrupación, en el modo DSP.

En la figura 5.11, como ya ocurría en el modo DSP, figura 5.5, se puede ver cómo a medida que crece la distancia entre fuente sonora y agrupación el porcentaje de error aumenta.

Capítulo 6

Conclusiones y líneas futuras

En este último capítulo se resumen las conclusiones recogidas tras la realización del proyecto y las posibles líneas futuras a implementar.

6.1. Conclusiones

El dispositivo UMA-8, es un dispositivo muy interesante para poder reconocer fuentes sonoras de una forma muy sencilla, pero es insuficiente si se desea precisión, ya que para trabajar de forma exacta sería necesario que la cámara se encontrase en el centro de la agrupación, como se ha explicado en el capítulo anterior, al ser esto imposible se ha intentado minimizar esta distancia, pero no se pueden alinear de forma precisa para todo el espacio, únicamente podría solucionarse en un plano a una distancia fija. Además para obtener resultados más exactos sería muy interesante poder modificar el código del microcontrolador. Aunque para aplicaciones sencillas donde no se busque exactitud el modo DSP es muy interesante ya que toda la información necesaria vendrá desde el dispositivo. Si se busca un resultado más preciso es conveniente implementar el mapa de potencias sonoras utilizando el "modo RAW" en tiempo real. A partir de los experimentos realizados los resultados son mucho mejor de lo esperado, ya que debido a las dimensiones de la placa y con ello a la distancia a la que se encuentra cada micrófono, se pensó que los canales iban a mezclarse. Pero tras las pruebas realizadas se puede decir que, los resultados obtenidos a partir de modo RAW con una frecuencia y a distancias cortas son, en localización de fuentes, bastante buenos. Se puede decir que, mucha parte de los errores vienen causados por la baja sensibilidad de los micrófonos MEMS, que es mucho menor que la de los micrófonos convencionales. Sería recomendable realizar una mayor cantidad de pruebas para así conseguir una mayor adaptación entre la cámara y el mapa de potencias, ajustando así el error en la localización. Como se mencionará en las líneas futuras, si se utilizase la cámara de la Kinect [17] en vez del ordenador, se podría estudiar la profundidad, distancia entre la agrupación y la fuente sonora,

pudiendo mejorar de este modo la localización de la fuente, ya que se conseguiría alinear de forma más exacta la agrupación y la imagen de la cámara. Mencionar que gracias a la cantidad de factores del dispositivo que se puede visualizar y modificar a través del modo DSP y su aplicación de miniDSP, se puede precisar de forma más sencilla el análisis realizado en el modo RAW.

Por lo que se puede concluir que los resultados obtenidos con el modo RAW son muchos mejores, que los obtenidos en el modo DSP, debido a que no se puede modificar las tareas realizadas por en microcontrolador XMOS, ya que el análisis que se realiza en modo RAW es en tiempo real y como se realiza todo desde el mismo entorno existe menor retardo y no existe la posibilidad de que haya una pérdida de datos.

6.2. Líneas futuras

Como forma de continuar este proyecto, se han pensado distintas formas para mejorar su precisión y posibilitar la detección de distintas fuentes. Las líneas futuras más inmediatas son las siguientes:

- Modificar el algoritmo que se ha utilizado para realizar el mapa de potencias, de tal modo que se utilice el micrófono central, y así mejorar el análisis.
- Realizar un algoritmo para poder hacer separación de fuentes y poder diferenciar de esta forma varias fuentes sonoras.
- Conseguir la presión sonora de forma analítica, partiendo de las especificaciones de los micrófonos utilizados.

Continuando con las líneas futuras, se buscaría realizar pequeñas modificaciones en el hardware, que serían las siguientes:

- Modificar la colocación de la agrupación respecto a la cámara, para que la cámara se encuentre en el centro de la agrupación y de esta forma minimizar el desfase entre ellos. Para implementar esto se necesitaría una cámara externa de tamaño reducido para no perder rango el rango de operación de los micrófonos.

- Utilización de una Kinect. Como ya se ha mencionado, se podría detectar con su cámara la "profundidad", la distancia entre agrupación y fuente, mejorando de esta forma la exactitud.

Po último, para continuar con el desarrollo de este proyecto, se podría utilizar más de una agrupación de este tipo, y de esta forma, conseguir las siguientes cosas:

- Utilización de varias agrupaciones, de esta forma se formaría un *array* de *arrays*, obteniendo mucha más información de sensores más separados, por lo que se podría calcular mucho mejor la posición de la que proviene el sonido, especialmente para bajas frecuencias.
- Detección de varias fuentes, si se parte de un *array* de *arrays* se podría conseguir detectar más fuentes sonoras sin perder información ya que se podrá eliminar analíticamente el solape entre los datos recogidos.

Bibliografía

[1] Díaz-Guerra, David. Trabajo Fin de Máster Localización de fuentes sonoras mediante agrupaciones de micrófonos, Universidad de Zaragoza, 2017.

[2] UMA-8, web oficial miniDSP: https://www.minidsp.com/products/usb-audio-interface/uma-8-microphone-array.

[3] Datasheet dispositivo UMA-8: https://www.minidsp.com/images/documents/Product%20Brief-UMA-8.pdf

[4] Manual dispositivo UMA-8: https://www.minidsp.com/images/documents/UMA-8%20User%20Manual.pdf

[5] Visual Studio IDE: https://www.visualstudio.com/es/vs/?rr=https%3A%2F%2Fwww.google.es%2F

[6] OpenFrameworks: http://openframeworks.cc/

[7] Visual Basic: https://www.microsoft.com/en-us/download/details.aspx?id=9639

[8] Web oficial de XMOS: http://www.xmos.com/

[9] Datasheet XMOS XSVM 2000: http://www.xmos.com/download/private/XUF216-512-TQ128-Datasheet%281.14%29.pdf

[10] Datasheet micrófonos MEMS, SPH1668LM4H: https://media.digikey.com/pdf/Data%20Sheets/Knowles%20Acoustics%20PDFs/SPH1668LM4H-1_Rev_A.pdf

[11] Plantilla HID visual basic: http://helmpcb.com/software/usb-hid-template-for-visual-basic-2005

[12] WikipediaUSB: https://es.wikipedia.org/wiki/Universal_Serial_Bus

[13] Alexa Amazon: https://developer.amazon.com/alexa

[14] Uma-8 Raspi: https://www.minidsp.com/applications/usb-mic-array/uma-8-rpi-diy-amazon-echo

[15] MATRIX Creator: https://creator.matrix.one/

[16] Addons adicionales FFTW: https://github.com/kylemcdonald/ofxFft

[17] Kinect de Xbox: https://www.xbox.com/es-ES/xbox-one/accessories/kinect

[18] Tamai, Yuki, et al. Three ring microphone array for 3d sound localization and separation for mobile robot audition. En Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on. IEEE, 2005. p. 4172-4177.

[19] Goseki, Masafumi; Takemura, Hiroshi; Mizoguchi, Hiroshi. Visualizing sound pressure distribution by kinect and microphone array. En *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*. IEEE, 2011. p. 1243-1248.

[20] Alexandridis, Anastasios, et al. Development and evaluation of a digital MEMS microphone array for spatial audio. En *Signal Processing Conference (EUSIPCO), 2016 24th European*. IEEE, 2016. p. 612-616.

[21] Open Sound Control, web oficial: http://opensoundcontrol.org/

ANEXOS

Anexo A

Disposición de los elementos de la placa y dimensiones de ésta

Se realiza un reconocimiento de los elementos de los que se ha hablado a lo largo del proyecto, para con ayuda de este esquema, figura A.1, poder reconocerlos en el dispositivo, además se podrá distinguir mejor la disposición de los micrófonos.

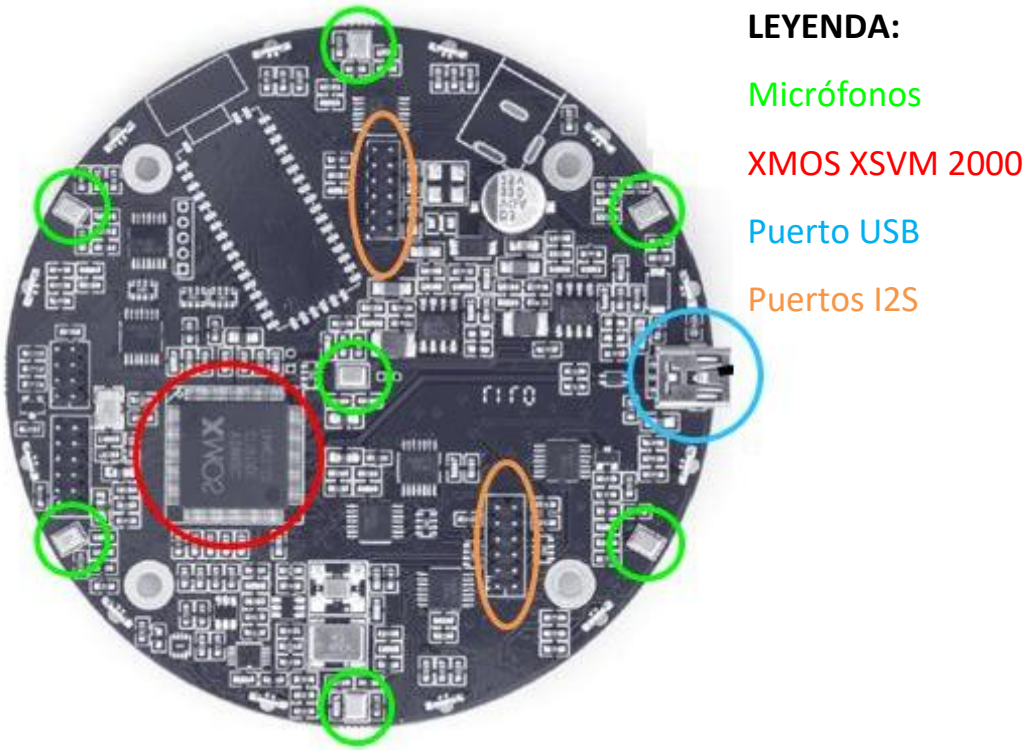

Figura A.1: Disposición de los elementos de la placa [3]

Además, es muy importante conocer el tamaño real de dispositivo para saber sus posibilidades de uso en otros campos, es por ello, que en la figura A.2, se realiza una medición de la placa.

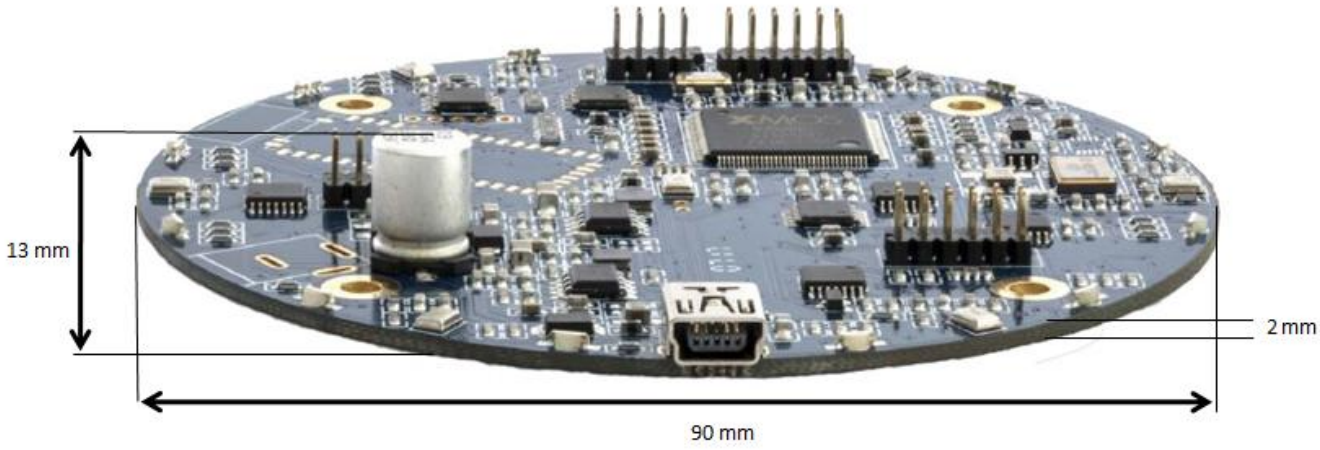

Figura A.2: Dimensiones, dispositivo UMA-8 [3]

Anexo B

Características del microprocesador XVSM-2000 de XMOS

- Microcontrolador multicore con una avanzada arquitectura RISC multicore:
    - 16 núcleo (core) lógicos en tiempo real, 2 xCORE por (tile).
    - Los núcleo (core) comparten 1000 MIPS: hasta 2000 MIPS en modo de doble emisión.
    - Cada núcleo (core) lógico tiene:
        - Rendimiento garantizado entre 1/5 y 1/8 de los (TILES) MIPS.
        - Registros dedicados 16x32 bit.
    - 167 instrucciones de alta densidad de 16/32 bits:
        - Todos tienen ejecución de ciclo de reloj único, excepto dividir.
        - 32x32 instrucciones MAC de 64 bits para DSP, aritmética y funciones criptográficas definibles por el usuario.
- USB PHY, totalmente compatible con las especificaciones USB 2.0.
- I/O programables:
    - 81 pines de entrada/salida de uso general, configurables como entradas o salidas.
        - Hasta 25 puertos de 1 bit, 12 puertos de 4 bits, 8 puertos de 8 bits y 4 puertos de 16 bits.
        - 4 enlaces xCONNECT.
    - Velocidades de muestreo de puertos de hasta 60 MHz con respecto al reloj externo.
    - 64 canales (endss) (32 per tile) para la comunicación con otros núcleos, chips de on u off.
- Memoria
    - 512KB SRAM internas de ciclo único para el almacenamiento de código y datos.
    - 16KB OTP interna para el código de inicio de la aplicación.
    - 2MB de flash interno para el código e aplicación y superposición.
- RecursoKB s de hardware:
    - 12 bloques de relojes, 6 per tile.

  - 20 timers, 10 per tile.
  - 8 locks, 4 per tile.
- Micrófono opcional y DSP acústico.
- Módulo JTAG para el chip de depuración.
- Características de seguridad:
  - El bloqueo de programación deshabilita la depuración e impide la lectura de los contenidos de la memoria.
  - El gestor de arranque de AES garantiza el secreto de la IP en la memoria flash externa.
- Rango de temperaturas ambiente:
  - Comercial: 0 °C a 70 °C.
  - Industria: -45°C a 85 °C.
- Grados de velocidad: 20: 1000 MIPS.
- Consumo de energía: valor típico 570 mA.
- 128 pins con encapsulado cuadrado plano de paso de 0,4mm (TQFP package -> igal que QFP package con diferencia en la separación de pines).

Anexo C

Código del software para el "modo DSP" realizado en Visual Basic.

```
Imports System.Net  'UDP
Imports System.Text.Encoding 'UDP
Imports System.Threading

Public Class Form1
    ' Identificadores de UMA-8: VID y PID
    Private Const VendorID As Integer = &H2752 'igual en los dos modos de uso
    Private Const ProductID As Integer = &H1C  'DSP_&H1C 'RAW_&H1D


    Private Const BufferInSize As Integer = 6     'Solo se envían 6 bytes útiles.
    Dim BufferIn(BufferInSize) As Byte            'Almacenamientos datos
recibidos
    Private mut As New Mutex()

    'Conexión UDP
    Dim UDPsocket As New Sockets.UdpClient()

    Dim lectura_Act As Boolean = False
    Dim OSC_Act As Boolean = False
    Dim conect_Act As Boolean = False

    ' Buffers de Escritura
    Dim bufferData(BufferInSize) As Byte
    Dim bufferChar(BufferInSize) As Char

    Dim contado As Integer = 0
    Dim pHandle As Integer



    Private Sub boton_exit_Click(sender As Object, e As EventArgs) Handles
boton_exit.Click
        Close()
    End Sub

    Private Sub LecturaStart_Click(sender As Object, e As EventArgs) Handles
LecturaStart.Click
        If (conect_Act) Then
            LecturaStart.BackColor = Color.Green
            LecturaStart.ForeColor = Color.White
            LecturaStop.BackColor = Color.DimGray
            LecturaStop.ForeColor = Color.Black
            lectura_Act = True

        End If
    End Sub


    Private Sub LecturaStop_Click(sender As Object, e As EventArgs) Handles
LecturaStop.Click
```

```
        LecturaStart.BackColor = Color.DimGray
        LecturaStart.ForeColor = Color.Black
        LecturaStop.BackColor = Color.Red
        LecturaStop.ForeColor = Color.White
        lectura_Act = False

    End Sub

    Private Sub UDP_Send_Click(sender As Object, e As EventArgs) Handles
UDP_Send.Click

        If (conect_Act) Then
            If (lectura_Act) Then
                UDPsocket.Connect("localhost", 11999)

                UDP_Send.BackColor = Color.Green
                UDP_Send.ForeColor = Color.White
                UDP_Stop.BackColor = Color.DimGray
                UDP_Stop.ForeColor = Color.Black
                OSC_Act = True
            End If
        End If
    End Sub

    Private Sub UDP_Stop_Click(sender As Object, e As EventArgs) Handles
UDP_Stop.Click
        UDP_Send.BackColor = Color.DimGray
        UDP_Send.ForeColor = Color.Black
        UDP_Stop.BackColor = Color.Red
        UDP_Stop.ForeColor = Color.White
        OSC_Act = False

    End Sub

    ' ****************************************************************
    ' Se carga es el Formulario e iniciamos la conexión HID
    '****************************************************************
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ConnectToHID(Me)
    End Sub

    '****************************************************************
    ' Desconectamos el HID, cuando cerramos el formulario
    '****************************************************************
    Private Sub Form1_FormClosed(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosedEventArgs) Handles Me.FormClosed
        DisconnectFromHID()
    End Sub

    '****************************************************************
    ' Cuando se conecta el USB
    '****************************************************************
    Public Sub OnPlugged(ByVal pHandle As Integer)
        If hidGetVendorID(pHandle) = VendorID And hidGetProductID(pHandle) =
ProductID Then
            Label1.Text = "USB Conectado"
            ledConect.BackColor = Color.Lime
            conect_Act = True
        End If
```

```
    End Sub

    '*****************************************************************
    ' Cuando se desconecta el USB
    '*****************************************************************
    Public Sub OnUnplugged(ByVal pHandle As Integer)
        If hidGetVendorID(pHandle) = VendorID And hidGetProductID(pHandle) =
ProductID Then
            hidSetReadNotify(hidGetHandle(VendorID, ProductID), False)
            Label1.Text = "USB Desconectado"
            ledConect.BackColor = Color.Red
            conect_Act = False
            lectura_Act = False
            OSC_Act = False
            LecturaStart.BackColor = Color.DimGray
            LecturaStart.ForeColor = Color.Black
            LecturaStop.BackColor = Color.Red
            LecturaStop.ForeColor = Color.White
            UDP_Send.BackColor = Color.DimGray
            UDP_Send.ForeColor = Color.Black
            UDP_Stop.BackColor = Color.Red
            UDP_Stop.ForeColor = Color.White
        End If
    End Sub


    Public Sub OnChanged()
        ' Evento para saber si hay datos que leer
        pHandle = hidGetHandle(VendorID, ProductID)
        hidSetReadNotify(hidGetHandle(VendorID, ProductID), True)
    End Sub

    '*****************************************************************
    ' Es el evento o interrupción que nos permite Leer los datos
    '*****************************************************************
    Public Sub OnRead(ByVal pHandle As Integer)

        If (lectura_Act) Then
            If hidRead(pHandle, BufferIn(0)) Then
                Dim i As Integer
                For i = 0 To BufferInSize
                    bufferData(i) = Val(BufferIn(i))
                Next

                If (OSC_Act) Then
                    'Envio de datos UDP
                    UDPsocket.Send(bufferData, bufferData.Length)

                    Dim j As Integer
                    For j = 0 To BufferInSize
                        SendKeys.Send(bufferData(j).ToString + "{.}")
                    Next

                    SendKeys.Send("{Enter}")

                End If
            End If
        End If
    End Sub
End Class
```

Anexo D

Código del software para el “modo DSP” en openFrameworks

```
#include "ofMain.h"
#include "ofApp.h"

//========================================================================
int main() {
	// El tamaño, 900x700, fue elegido por el tamaño de video que se utilizará
	// más las franjas adicionales, la del lateral para los botones y la
	// inferior para la representación del audio.
	ofSetupOpenGL(900, 700, OF_WINDOW);


	ofRunApp(new ofApp());
}
```

```
#pragma once

#include "ofMain.h"
#include "ofxNetwork.h" // Comunicación UDP
#include "ofxFFTBase.h" // Representación del sonido


class ofApp : public ofBaseApp {

public:
	void setup();
	void update();
	void draw();
	void mousePressed(int x, int y, int button);
	void sectionDraw();
	void drawSamples(vector<float> samples);
	void audioIn(float * input, int bufferSize, int nChannels);

	// Control de camara
	ofVideoGrabber video;

	// Imagenes entorno gráfico
	ofImage cam, noCam;
	ofImage pandaIcon, pandaIconNO;
	ofImage FFT, noFFT;
	ofImage  panda;

	// Conexión UDP
	ofxUDPManager udpConnection;

	// SONIDO
	// Recepción de sonido
	ofSoundStream soundStreamInput;
	vector <float> audio;
	// Representación audio
	ofxFFTBase fftChannel;

	// Pantalla secundaria: PANDA
	// Vectores que contienen la posición de los ojos, según el microfon
activo
	vector<int> rightEyeX = { 200, 225, 225, 175, 175, 175, 225 };
	vector<int> rightEyeY = { 290, 290, 310, 310, 290, 270, 270 };
	vector<int> leftEyeX = { 535, 560, 560, 510, 510, 510, 560 };
	vector<int> leftEyeY = { 285, 285, 315, 315, 285, 265, 265 };

	// Micrófono activo
	int micro;
};
```

OpenFrameworks: ofApp.cpp

```
#include "ofApp.h"

//Inicialización
BOOLEAN videoAct = true;
BOOLEAN controlCamara = true;
BOOLEAN controlPanda = false;
BOOLEAN controlFFT = false;
int micro = 0;
char udpMessage[7];

//--------------------------------------------------------------
void ofApp::setup() {

	ofBackground(0, 0, 0);

	// Carga de imagenes
	cam.load("cam.jpg");
	noCam.load("noCam.jpg");
	panda.load("panda.jpg");
	pandaIcon.load("pandaIcon.jpg");
	pandaIconNO.load("pandaNo.jpg");
	FFT.load("fftIcon.jpg");
	noFFT.load("nofftIcon.jpg");

	// VIDEO:
	// El tamaño fue seleccionado por la resolución de la cámara utilizada,
	// si se utilizase una cámara con mayor calidad podría aumentarse.
	video.setup(740, 480);

	// SONIDO:
	// Velocidad de muestreo: 16000 Hz
	// 256 muestras por cada buffer
	// 4 buffers para evitar latencia

	//Listado de dispositivos
	ofSoundStreamListDevices();

	// Buffer completo, contiene el lado derecho y el izquierdo
	int bufferSize = 512;
	// Buffer de entrada
	audio.assign(bufferSize, 0.0);


	// UMA-8
	soundStreamInput.setDeviceID(0);
	soundStreamInput.setup(this, 0, 2, 16000, bufferSize, 4);

	// Conexión UDP
	udpConnection.Create();  // Creación del puerto cliente
	udpConnection.Bind(11999); // Conexión al puerto 11999
	udpConnection.SetTimeoutReceive(0);

}

//--------------------------------------------------------------
void ofApp::update() {
	// WEBCAM
	video.update();

	// SONIDO
```

```
	fftChannel.update();

	// COMUNICACIÓN UDP
	udpConnection.Receive(udpMessage, 10); // No es bloqueante
	if (udpMessage != "") {
		// Conversión a entero del valor del micrófono
		micro = (int)udpMessage[6];
	}
}

//--------------------------------------------------------------
void ofApp::draw() {

	// Control de video
	if (videoAct) {
		ofSetColor(255, 255, 255);
		video.draw(0, 0);
	}
	else {
		ofSetColor(0, 0, 0);
		video.draw(0, 0);
		if (controlPanda) {
			ofSetColor(255, 255, 255);
			panda.draw(0, 0, 740, 480);
		}
	}

	// Icono de cámara
	if (controlCamara) {
		ofSetColor(255, 255, 255);
		ofDrawRectangle(777, 17, 86, 86);
		cam.draw(780, 20, 80, 80);
	}
	else {
		ofSetColor(255, 255, 255);
		ofDrawRectangle(777, 17, 86, 86);
		noCam.draw(780, 20, 80, 80);
	}

	// Representación del audio
	if (controlFFT) {
		ofPushMatrix();
		ofTranslate(0, 700);
		drawSamples(fftChannel.getFftNormData());
		ofPopMatrix();
		ofSetColor(255, 255, 255);
		ofDrawRectangle(777, 117, 86, 86);
		noFFT.draw(780, 120, 80, 80);
	}
	else {
		ofSetColor(255, 255, 255);
		ofDrawRectangle(777, 117, 86, 86);
		FFT.draw(780, 120, 80, 80);
	}

	// Pantalla secundaria: PANDA
	if (controlPanda) {
		ofSetColor(255, 255, 255);
		ofDrawRectangle(800, 415, 100, 65);
		pandaIconNO.draw(800, 415, 100, 65);
```

```
			ofSetColor(0, 0, 0);
			ofDrawCircle(leftEyeX[micro], leftEyeY[micro], 50);
			ofSetColor(0, 0, 0);
			ofDrawCircle(rightEyeX[micro], rightEyeY[micro], 50);
		}
		else {
			// Representación de la sección en la que trabaja el
			// micrófono afectado por la fuente sonora.
			ofSetColor(255, 255, 255);
			ofDrawRectangle(800, 415, 100, 65);
			pandaIcon.draw(800, 415, 100, 65);
			sectionDraw();
		}
}

//--------------------------------------------------------------------------
-------------
void ofApp::mousePressed(int x, int y, int button) {
		ofPoint pt;
		pt.set(x, y);

		if (button == 0) {
			if (((780 <= x) && (860 >= x)) && ((10 <= y) && (90 >= y))) {
				if (controlPanda) {
					controlPanda = false;
					controlCamara = false;
					videoAct = false;
				}
				else {
					controlCamara = !controlCamara;
					videoAct = !videoAct;
				}
				ofApp::draw();
			}
			else if (((780 <= x) && (860 >= x)) && ((110 <= y) && (190 >= y)))
{
				if (controlPanda) {
					controlFFT = false;
				}
				else {
					controlFFT = !controlFFT;
				}
				ofApp::draw();
			}
			else if (((800 <= x) && (900 >= x)) && ((415 <= y) && (480 >= y)))
{
				if (controlCamara) {
					controlPanda = !controlPanda;
					if (controlPanda) {
						videoAct = false;
					}
					else {
						videoAct = true;
					}
				}
				else {
					controlCamara = true;
					controlFFT = false;
					controlPanda = true;
					videoAct = false;
```

```
                        }
                        ofApp::draw();
                }
        }
}


// Función de audio
//--------------------------------------------------------------------------------
-------------
void ofApp::audioIn(float * input, int bufferSize, int nChannels) {

        for (int i = 0; i < bufferSize; i++) {
                audio[i] = input[i];
        }
        float * data = &audio[0];
        fftChannel.audioIn(data);
}

void ofApp::sectionDraw() {
        // Esta función representa con un nivel de transparencia del 50% la zona
        // en la que se ha detectado la fuente sonora. Por construcción solo se
        // verá la sección de la que viene el sonido en la pantalla del video,
        // esté activo o no.

        if (micro == 1) {
                ofEnableAlphaBlending();
                ofSetColor(255, 0, 0, 127);
                ofDrawTriangle(0, 57, 370, 240, 0, 423);
        }
        else if (micro == 2) {
                ofEnableAlphaBlending();
                ofSetColor(255, 0, 0, 127);
                ofDrawTriangle(0, 423, 370, 240, 370, 423);
                ofEnableAlphaBlending();
                ofSetColor(255, 0, 0, 127);
                ofDrawRectangle(0, 423, 370, 57);
        }
        else if (micro == 3) {
                ofEnableAlphaBlending();
                ofSetColor(255, 0, 0, 127);
                ofDrawTriangle(370, 423, 370, 240, 740, 423);
                ofEnableAlphaBlending();
                ofSetColor(255, 0, 0, 127);
                ofDrawRectangle(370, 423, 370, 57);
        }
        else if (micro == 4) {
                ofEnableAlphaBlending();
                ofSetColor(255, 0, 0, 127);
                ofDrawTriangle(740, 57, 370, 240, 740, 423);
        }
        else if (micro == 5) {
                ofEnableAlphaBlending();
                ofSetColor(255, 0, 0, 127);
                ofDrawTriangle(370, 57, 370, 240, 740, 57);
                ofEnableAlphaBlending();
                ofSetColor(255, 0, 0, 127);
                ofDrawRectangle(370, 0, 370, 57);
        }
        else if (micro == 6) {
```

```
			ofEnableAlphaBlending();
			ofSetColor(255, 0, 0, 127);
			ofDrawTriangle(0, 57, 370, 240, 370, 57);
			ofEnableAlphaBlending();
			ofSetColor(255, 0, 0, 127);
			ofDrawRectangle(0, 0, 370, 57);
		}
}


void ofApp::drawSamples(vector<float> samples) {
		//Representación de audio recogido por los micrfonos
		int sampleWidth = ofGetWidth() / (2*samples.size());
		int sampleHeight = ofGetHeight() / 2;
		int numOfSamples = samples.size()/2;
		// Se ha reducido todo un factor 2 para adaptarlo a la ventana utilizada

		for (int i = 0; i<numOfSamples; i++) {
			int x = ofMap(i, 0, numOfSamples - 1, 0, ofGetWidth() -
sampleWidth);
			int y = 0;
			int w = sampleWidth;
			int h = (-samples[i] * sampleHeight)*0.5;

			ofRect(x, y, w, h);
		}
}
```

Anexo E

Código del software para el “modo RAW” en openFrameworks

```
#include "ofMain.h"
#include "ofApp.h"

//========================================================================
int main() {
	// El tamaño, 900x700, fue elegido por el tamaño de video que se utilizará
	// más las franjas adicionales, la del lateral para los botones y la
	// inferior para la representación del audio.
	ofSetupOpenGL(900, 700, OF_WINDOW);


	ofRunApp(new ofApp());
}
```

```
#pragma once
#include "ofMain.h"
#include "headers.h"
#include "hanning1024.h"
#include <vector>
#include <mutex>
#include "srp_phat.h"

class ofApp : public ofBaseApp {

public:

	void setup();
	void update();
	void draw();
	void mousePressed(int x, int y, int button);
	void drawIconsChannels();
	void audioIn(float * input, int bufferSize, int nChannels);
	void colorsSetup();
	void sectionDraw();

	// Iconos para la interfaz
	ofVideoGrabber video;
	ofImage cam, noCam;
	ofImage sigAudio, noSigAudio;
	ofImage ChannelsIcons0, ChannelsIcons1, ChannelsIcons2, ChannelsIcons3,
		ChannelsIcons4, ChannelsIcons5, ChannelsIcons6;
	ofImage noChannelsIcons0, noChannelsIcons1, noChannelsIcons2,
		noChannelsIcons3,noChannelsIcons4, noChannelsIcons5,
		noChannelsIcons6;

	// Matrices de audio
	// audio contiene los 8 canales
	// x conitne los canales de los micrófonos periféricos.
	float audio[8][1024];
	float x[6][1024];

	// Se utiliza para la visualización de canales
	int colors[7][3];

	// Para evitar la sección crítica
	mutex mutexChannels;

	ofSoundStream soundStreamInput;

	// Para la utilización de la matriz srp_phat
	srp_phat srp;

	// Matriz para la representación del mapa
	float P[129][129];
	ofImage imageResult;

	// Posición del máximo
	float maxP;

	// Paleta de colores para crear degradados según la poténcia

	int thermal_palette[512][3] = {…};

};
```

OpenFrameworks: ofApp.cpp

```
#include "ofApp.h"
//Inicialización
BOOLEAN videoAct = true;
BOOLEAN controlCamara = true;
BOOLEAN controlSigAudio = true;
BOOLEAN controlFFT = false;
BOOLEAN verCanales[8];
int bufferSize = 1024;
int nChannels = 8;
// Solo se utilizarán los 6 micrófonos periféricos en SRP_PHAT
int nSensors = 6;
// Contador para pruebas
int contRepet = 0;
int contEscrb = 0;


//--------------------------------------------------------------
void ofApp::setup() {

	ofSetVerticalSync(true);
	ofSetCircleResolution(80);
	ofBackground(0, 0, 0);

	// Inicialización del vídeo
	video.setup(400, 400);

	imageResult.allocate(129, 129, OF_IMAGE_COLOR_ALPHA);
	imageResult.setColor(ofColor::black);

	// Inicialización de la clase srp_phat
	srp.setup((float*)x, (float*)P);

	// Inicialización de imagenes
	cam.load("cam.jpg");
	noCam.load("noCam.jpg");
	sigAudio.load("sigAudio.jpg");
	noSigAudio.load("noSigAudio.jpg");
	ChannelsIcons0.load("num0.jpg");
	ChannelsIcons1.load("num1.jpg");
	ChannelsIcons2.load("num2.jpg");
	ChannelsIcons3.load("num3.jpg");
	ChannelsIcons4.load("num4.jpg");
	ChannelsIcons5.load("num5.jpg");
	ChannelsIcons6.load("num6.jpg");
	noChannelsIcons0.load("noNum0.jpg");
	noChannelsIcons1.load("noNum1.jpg");
	noChannelsIcons2.load("noNum2.jpg");
	noChannelsIcons3.load("noNum3.jpg");
	noChannelsIcons4.load("noNum4.jpg");
	noChannelsIcons5.load("noNum5.jpg");
	noChannelsIcons6.load("noNum6.jpg");

	colorsSetup();

	// 0 output channels,
	// 8 input channels
	// 16000 samples per second
	// 1024 samples per buffer
	// 4 num buffers (latency)
	ofSoundStreamListDevices();
```

```
	soundStreamInput.setDeviceID(0);
	soundStreamInput.setup(this, 0, 8, 44100, bufferSize, 4);

	// Inicialización matriz de audio
	for (int i = 0; i < 8; i++)
	{
		for (int j = 0; j < bufferSize; j++) {
			audio[i][j] = 0.0;
		}
	}
}

//--------------------------------------------------------------
void ofApp::update() {
	// WEBCAM
	video.update();

	mutexChannels.lock();
	//Actualiza la imágen con los últimos valores de SRP
	for (int x = 0; x < 129; x++) {
		for (int y = 0; y < 129; y++) {
			int level = maxP>0 ? (int)(P[y][x] / maxP * 511) : 0;
			if (level < 0) level = 0;
			int alpha = level / 4;
			imageResult.setColor(129 - x - 1, y,
ofColor(thermal_palette[level][0], thermal_palette[level][1],
thermal_palette[level][2], alpha));
		}
	}
	imageResult.update();
	mutexChannels.unlock();
}

//--------------------------------------------------------------
void ofApp::draw() {
	// Control de video
	if (videoAct) {
		ofSetColor(255, 255, 255);
		//video.draw(0, 0);
		video.draw(170, 40);
	}
	else {
		ofSetColor(0, 0, 0);
		//video.draw(0, 0);
		video.draw(170, 40);
	}
	if (controlCamara) {
		ofSetColor(255, 255, 255);
		ofDrawRectangle(777, 7, 86, 86);
		cam.draw(780, 10, 80, 80);
	}
	else {
		ofSetColor(255, 255, 255);
		ofDrawRectangle(777, 7, 86, 86);
		noCam.draw(780, 10, 80, 80);
	}
	if (controlSigAudio) {
		// Representación de los iconos para la selecciones de los canales
		drawIconsChannels();
```

```
                ofSetColor(255, 255, 255);
                ofDrawRectangle(777, 107, 86, 86);
                noSigAudio.draw(780, 110, 80, 80);

                ofNoFill();
                ofPushStyle();
                ofPushMatrix();

                ofSetLineWidth(1);
                ofDrawRectangle(1, 500, 999, 200);
                mutexChannels.lock();

                for (int j = 0; j < 7; j++)
                {
                        // Representación de los canales seleccionados, superpuestos
                        if (verCanales[j]) {
                                ofSetColor(colors[j][0], colors[j][1], colors[j][2]);
                                ofSetLineWidth(2);
                                ofBeginShape();
                                for (unsigned int i = 0; i < bufferSize / 8; i++) {
                                        ofVertex((i*30) + 1, 600 - audio[j][i]*180.0f);
                                }
                                ofEndShape(false);
                        }
                }
                mutexChannels.unlock();
                ofPopMatrix();
                ofPopStyle();
        }
        else {
                ofSetColor(255, 255, 255);
                ofDrawRectangle(777, 107, 86, 86);
                sigAudio.draw(780, 110, 80, 80);
        }

        //Representación de la matriz de mapa de potencias.
        imageResult.draw(170, 40, 400, 400);

        // Función utilizada para realizar las medidas
        // sectionDraw();
}
//--------------------------------------------------------------
void ofApp::mousePressed(int x, int y, int button) {
        ofPoint pt;
        pt.set(x, y);

        if (button == 0) {
                if (((780 <= x) && (860 >= x)) && ((10 <= y) && (90 >= y))) {
                        controlCamara = !controlCamara;
                        videoAct = !videoAct;
                        ofApp::draw();
                }
                else if (((780 <= x) &&(860 >= x)) && ((110 <= y) && (190 >= y))) {
                        controlSigAudio = !controlSigAudio;
                        ofApp::draw();
                }
                // CANALES
                else if (((770 <= x) &&(810 >= x)) && ((220 <= y) && (260 >= y))) {
                        // Micro 1
                        if (controlSigAudio) {
```

```
                verCanales[1] = !verCanales[1];
        }
        else {
                verCanales[1] = false;
        }

        ofApp::draw();
}
else if (((830 <= x) &&(870 >= x)) && ((220 <= y) && (260 >= y))) {
        // Micro 2
        if (controlSigAudio) {
                verCanales[2] = !verCanales[2];
        }
        else {
                verCanales[2] = false;
        }
        ofApp::draw();
}
else if (((770 <= x) &&(810 >= x)) && ((270 <= y) && (310 >= y))) {
        // Micro 3
        if (controlSigAudio) {
                verCanales[3] = !verCanales[3];
        }
        else {
                verCanales[3] = false;
        }
        ofApp::draw();
}
else if (((830 <= x) &&(870 >= x)) && ((270 <= y) && (310 >= y))) {
        // Micro 4
        if (controlSigAudio) {
                verCanales[4] = !verCanales[4];
        }
        else {
                verCanales[4] = false;
        }
        ofApp::draw();
}
else if (((770 <= x) &&(810 >= x)) && ((320 <= y) && (360 >= y))) {
        // Micro 5
        if (controlSigAudio) {
                verCanales[5] = !verCanales[5];
        }
        else {
                verCanales[5] = false;
        }
        ofApp::draw();
}
else if (((830 <= x) &&(870 >= x)) && ((320 <= y) && (360 >= y))) {
        // Micro 6
        if (controlSigAudio) {
                verCanales[6] = !verCanales[6];
        }
        else {
                verCanales[6] = false;
        }
        ofApp::draw();
}
else if (((800 <= x) &&(840 >= x)) && ((370 <= y) && (410 >= y))) {
        // Micro 0
```

```
                        if (controlSigAudio) {
                                verCanales[0] = !verCanales[0];
                        }
                        else {
                                verCanales[0] = false;
                        }
                        ofApp::draw();
                }
        }
}
//--------------------------------------------------------------
void ofApp::audioIn(float * input, int bufferSize, int nChannels) {
        // Recepción de audio.
        mutexChannels.lock();
        for (int i = 0; i < nChannels; i++) {
                for (int j = 0; j < (bufferSize); j++) {
                        // Matriz con los 7 canales, uno por micrófono, el 8
                        // no contendrá datos.
                        audio[i][j] = input[i + j*nChannels];

                        if (i < 6) {
                                // La matriz x contiene la información de los 6
                                // periféricos. Esta matriz es la que se utilza
                                // en el análisis.
                                x[i][j] = input[j*nChannels + (i + 1)];
                        }
                }
        }
        // Se ejecuta la clase srp_phat().
        maxP = srp.execute();
        mutexChannels.unlock();
}
//--------------------------------------------------------------
void ofApp::drawIconsChannels() {
        // Representación de los botones de los numeros, según se esté
        // visualizando el canal o no.

        // Canal 1
        ofSetColor(255, 255, 255);
        ofDrawRectangle(767, 217, 46, 46);
        if (verCanales[1]) {
                noChannelsIcons1.draw(770, 220, 40, 40);
        }
        else {
                ChannelsIcons1.draw(770, 220, 40, 40);
        }
        // Canal 2
        ofSetColor(255, 255, 255);
        ofDrawRectangle(827, 217, 46, 46);
        if (verCanales[2]) {
                noChannelsIcons2.draw(830, 220, 40, 40);
        }
        else {
                ChannelsIcons2.draw(830, 220, 40, 40);
        }
        // Canal 3
        ofSetColor(255, 255, 255);
        ofDrawRectangle(767, 267, 46, 46);
        if (verCanales[3]) {
                noChannelsIcons3.draw(770, 270, 40, 40);
```

```
		}
		else {
			ChannelsIcons3.draw(770, 270, 40, 40);
		}
		// Canal 4
		ofSetColor(255, 255, 255);
		ofDrawRectangle(827, 267, 46, 46);
		if (verCanales[4]) {
			noChannelsIcons4.draw(830, 270, 40, 40);
		}
		else {
			ChannelsIcons4.draw(830, 270, 40, 40);
		}// Canal 5
		ofSetColor(255, 255, 255);
		ofDrawRectangle(767, 317, 46, 46);
		if (verCanales[5]) {
			noChannelsIcons5.draw(770, 320, 40, 40);
		}
		else {
			ChannelsIcons5.draw(770, 320, 40, 40);
		}
		// Canal 6
		ofSetColor(255, 255, 255);
		ofDrawRectangle(827, 317, 46, 46);
		if (verCanales[6]) {
			noChannelsIcons6.draw(830, 320, 40, 40);
		}
		else {
			ChannelsIcons6.draw(830, 320, 40, 40);
		}// Canal 0
		ofSetColor(255, 255, 255);
		ofDrawRectangle(797, 367, 46, 46);
		if (verCanales[0]) {
			noChannelsIcons0.draw(800, 370, 40, 40);
		}
		else {
			ChannelsIcons0.draw(800, 370, 40, 40);
		}
}
//--------------------------------------------------------------
void ofApp::colorsSetup() {
		// Esto se utiliza para visualizar todos los canales con colores
		// disitintos, si se dejaba al azar los colores se parecían mucho y no se
		// distinguían bien.

		// Canal 0
		colors[0][0] = 255;
		colors[0][1] = 0;
		colors[0][2] = 0;
		// Canal 1
		colors[1][0] = 0;
		colors[1][1] = 255;
		colors[1][2] = 0;
		// Canal 2
		colors[2][0] = 0;
		colors[2][1] = 0;
		colors[2][2] = 255;
		// Canal 3
		colors[3][0] = 255;
		colors[3][1] = 255;
```

```
        colors[3][2] = 0;
        // Canal 4
        colors[4][0] = 255;
        colors[4][1] = 0;
        colors[4][2] = 255;
        // Canal 5
        colors[5][0] = 0;
        colors[5][1] = 255;
        colors[5][2] = 255;
        // Canal 6
        colors[6][0] = 255;
        colors[6][1] = 255;
        colors[6][2] = 255;
}
void ofApp::sectionDraw() {
        // Función utilizada para la realización de las medidas.
        ofSetColor(255, 0, 0, 127);
        ofDrawLine(270, 40, 370, 240);
        ofDrawLine (170, 240, 370, 240);
        ofDrawLine(470, 40, 370, 240);
        ofDrawLine(570, 240, 370, 240);
        ofDrawLine(270, 440, 370, 240);
        ofDrawLine(470, 440, 370, 240);
}
```