UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

DISEÑO Y DESARROLLO EN ENTORNO MATLAB DE UN SISTEMA PARA LA DETECCIÓN Y LOCALIZACIÓN DE EVENTOS SONOROS ATMOSFÉRICOS SINGULARES

Titulación: Grado en Ingeniería en Tecnologías de la Telecomunicación
Mención: Sonido e Imagen
Autor: Juan Francisco Martín Rodríguez
Tutor: Dr. Eduardo Hernández Pérez
Fecha: Julio, 2019

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

DISEÑO Y DESARROLLO EN ENTORNO MATLAB DE UN SISTEMA PARA LA DETECCIÓN Y LOCALIZACIÓN DE EVENTOS SONOROS ATMOSFÉRICOS SINGULARES

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo:_____

Vocal

Fdo:_____

Secretario

Fdo:_____

Fecha: Julio, 2019

[Esta página ha sido dejada intencionalmente en blanco]

AGRADECIMIENTOS

Agradecer en primer lugar a mi tutor Eduardo por darme la oportunidad de realizar este proyecto en el que se usan tecnologías punteras, por ayudarme cuando no encontraba salida, servirme de orientación y corregirme todo el trabajo, tanto la memoria como el código de Matlab de principio a fin.

A mi padre, que ahora no está con nosotros y a mi madre, porque siempre estaban al teléfono para escucharme cuando hablaba con ellos sobre los buenos momentos y sobre los que me servían de aprendizaje en la universidad, y por contribuir con mis estudios, también a mi hermano por ser un gran compañero.

A mis dos abuelas que tampoco están con nosotros, pero deseaban verme con una carrera "debajo del brazo".

A mi tía Juliana y mi prima Corín por su apoyo desde que entré a la universidad.

A mi tía Carmen, por ser vecina y darme buenos consejos.

A mi prima Elena que estudió Matemáticas porque siempre ha sido un ejemplo a seguir en la familia, igual que su marido Octavio.

A mi primo Lorenzo Tomás que estudió ingeniería de Telecomunicaciones y ha sido otro referente en la familia y a mi primo Humberto por ser el primo que más ha estado a mi lado.

Agradecer a los amigos de Lanzarote, como Félix, Moisés, Cristian, Kevin, A. Pérez y Adrián por los buenos momentos que hemos pasado.

De Gran Canaria a Andrés, Gabri, Mario, Luisete o Angelo por ser graciosos y por ser los mejores amigos que tengo en la isla.

A los amigos de la Universidad de Málaga, como Juanpa, gracias a él elegí la carrera y me ayudó bastante en los dos años que estuve allí. También a Antonio y a Fran 'Cata'.

A los compañeros de la Universidad de las Palmas, como Saúl, Javier Reimón, Jonay, Jaime o Lara, por toda su ayuda recibida y por lo bonito que era estar unidos y ayudarnos.

Agradecer a cada uno de los profesores que me ha dado clase desde la guardería hasta la universidad, porque cada uno de ellos ha contribuido en mi formación educativa. En especial a Don Alejandro del colegio de Mozaga, Mª de los Ángeles del colegio de San Bartolomé. Y del instituto de San Bartolomé a Manolo, Don Luis, Estéban y Óscar el jefe de estudios.

También a los profesores de la universidad, en especial a Alfonso por transmitirnos su amor por la asignatura de Microprocesadores, al principio la vi imposible, pero al final la saqué. También a Félix, Carmen Nieves y Eduardo Rovaris.

Agradecer a Eckhart Tolle, el autor del libro "El Poder del Ahora", por sus videos de YouTube y por sus enseñanzas para estar en paz y alineados con el momento presente.

Agradecer a Dios porque siempre tuve fé, mantuve la paciencia y permanecí fuerte y firme en los momentos difíciles y por terminar esta ingeniería.

Juan Francisco Martín Rodríguez

Julio, 2019

"No fracasé, solo descubrí 999 maneras de cómo no realizar la bombilla".

Thomas Alva Edison

[Esta página ha sido dejada intencionalmente en blanco]

Tabla de Contenidos

Lista de acrónimos y siglas

DCT – Discrete Cosine Transform

DEMO – DEMOstración

DFT - Discrete Fourier Transform

DOA - Direction Of Arrival

FFT - Fast Fourier Transform

FIR - Finite Impulse Response

IDFT - Inverse Discrete Fourier Transform

IEEE - Institute of Electrical and Electronics Engineers

KNN - K-Nearest Neighbor

MATLAB - MATrix LABoratory

MFCC - Mel Frequency Cepstral Cepstrum

ML - Maximum Likelihood

MUSIC - MUltiple SIgnal Classification

MV - Minimum Variance

RF - Random Forest

SISEVEMETEO - SIStema de EVEntos METEOrológicos

SVD - Singular Values Decomposition

TDOA - Time Difference of Arrival

USB - Universal Serial Bus

[Esta página ha sido dejada intencionalmente en blanco]

Índice de figuras

Índice de tablas

[Esta página ha sido dejada intencionalmente en blanco]

MEMORIA

[Esta página ha sido dejada intencionalmente en blanco]

CAPÍTULO 1: INTRODUCCIÓN

[Esta página ha sido dejada intencionalmente en blanco]

1.1 Introducción

Un evento sonoro se puede usar para describir una escena sonora, permitiendo representar, de manera simbólica, dicha escena. Por ejemplo, en el caso de una ciudad: calle concurrida, con autos pasando, bocinas de automóviles o el paso de gente. Los diferentes descriptores de nivel representan texto (calle) y eventos característicos (coche, bocina de coche, pasos) [7]. Un ejemplo de ello es SOLAR [6] donde mediante análisis espectral y técnicas probabilísticas de clasificación hacen posible la identificación de los eventos sonoros y por ende la tipología de la escena. Se trata de un sistema capaz de detectar y clasificar eventos sonoros, como el ladrido de un perro, la pita de un coche, disparos, un niño llorando…

En otras investigaciones resultantes de la combinación del Procesado de Audio y Aprendizaje Automático (Machine Learning) que se han llevado a cabo durante los últimos años, la mayoría se ha centrado en el reconocimiento de sonidos enfocados al ámbito de la voz y la música, como por ejemplo el reconocimiento de voz o del hablante (Automatic Speaker Recognition), el reconocimiento atomático del habla (Automatic Speech Recognition), o incluso la recuperación de información musical basada en Reconocimiento de Patrones (Music Information Retrieval) [1].
Un campo menos desarrollado dentro de la clasificación automática de señales de audio (Audio Signal Classification), pero actualmente en continua investigación, es el reconocimiento de ruidos ambientales o del entorno (Environmental Sound Recognition) [2], el cual tiene diversas aplicaciones y entre ellas se encuentra la identificación de eventos sonoros atmosféricos singulares [3].

Los eventos sonoros singulares son aquellos que tienen lugar dentro de una escena sonora determinada, pero no son lo habitual en dicha escena, por así decirlo no son propios del contexto de una escena sonora. Por ejemplo, en una escena sonora en la naturaleza lo normal son sonidos de aves, brisa en la vegetación, mamíferos terrestres o anfibios entre otros. La lluvia y los truenos se podrían considerar eventos singulares ya que no ocurren en el cien por cien del tiempo.
Si en la captura del evento sonoro usamos más de un micrófono tendríamos lo que se conoce como una agrupación de micrófonos. La idea general de una agrupación de micrófonos es análoga a la de un solo micrófono muy directivo al combinar adecuadamente las señales de todos los micrófonos [5]. La salida tendrá menor nivel de ruido y permitirá incluso reducir la posible reverberación que acompañe al sonido capturado. Además, en base a las señales capturadas por cada micrófono, situados en localizaciones diferentes, podemos usar la agrupación para localizar la fuente sonora. Este es un proceso de estimación de la dirección de llegada (DOA, Direction Of Arrival) de la fuente sonora [5].

El propósito principal de este trabajo fin de grado es el diseño, desarrollo e implementación en entorno Matlab de un sistema con aplicaciones principalmente meteorológicas, capaz de detectar y localizar eventos sonoros singulares, entre los que se encuentran dos atmosféricos como son el trueno y la lluvia y uno no atmosférico como es el sonido ambiente.

El funcionamiento de este sistema será en tiempo real, es decir, realizará sus tareas al mismo tiempo que recoja la señal sonora, esta señal puede ser microfónica o grabada procedente de archivos de audio.
Para su implementación en exteriores, usaremos una agrupación de micrófonos que adaptaremos a un ordenador personal que incluya el entorno Matlab, concretamente la Audio System Toolbox [4], desarrollando en ese entorno el conjunto de herramientas de procesado y análisis, junto con la correspondiente interfaz de usuario que permita la interacción con la aplicación.
Para su implementación con señal grabada, procederemos de la misma manera, pero sólo usando nuestro ordenador personal ya que las tareas se realizarán a nivel software.

Probaremos el sistema desarrollando de manera sintética una serie de escenarios sonoros que nos permitirán comprobar el funcionamiento de cada una de las funcionalidades del sistema tanto para señal grabada como para exteriores.

La tarea desde la que parte nuestro sistema es la de detección, la cual consiste en distinguir el sonido del silencio y para realizarla, se establecerá un umbral de detección previamente. Sólo realizará las tareas posteriores cuando el sistema detecte sonido, de esta manera habrá señal para trabajar.
Luego clasificará los sonidos atendiendo a sus bases de datos ya entrenadas. Por último, localizará la dirección de procedencia sólo si en la clasificación ha identificado truenos.

1.2 Objetivos

De acuerdo al anteproyecto, los objetivos establecidos en este Trabajo de Fin de Grado son los siguientes:

• Obj. 1. **Comprender el funcionamiento y las herramientas de los sistemas de detección, localización de la dirección de llegada y clasificación de eventos sonoros.** Este objetivo está planteado para realizar un estudio sobre investigaciones y desarrollos relacionados con el sistema que queremos desarrollar, así como de las herramientas disponibles para llevar a cabo su implementación. De esta manera, tendremos presentes las limitaciones y problemas que puedan surgir a la hora de empezar y así podemos solucionarlos de la manera más sencilla y rápida posible. Por último, podremos elegir las herramientas que más se adapten a nuestras necesidades para afrontar con éxito la realización de este Trabajo.

• Obj. 2. **Establecer la metodología genérica a seguir en el desarrollo de un sistema de localización de eventos sonoros en entorno Matlab.** Conociendo las herramientas que podemos usar para la localización de la dirección de procedencia, elegiremos una de ellas y estableceremos un método para llevar a cabo su funcionamiento en el sistema.

• Obj. 3. **Integrar un sistema de detección y clasificación en entorno Matlab.** Una vez entendamos las herramientas que podemos usar para la detección, construimos un sistema en entorno Matlab que pueda distinguir los silencios y los sonidos de manera que actúe solamente cuando hay sonido y seguidamente, conociendo las herramientas

para la clasificación, construiremos otro sistema en el mismo entorno que clasifique los eventos sonoros de interés. Por último, combinamos los dos sistemas en entorno Matlab realizando las tareas necesarias para ello.

• Obj. 4. **Integrar un sistema de localización en entorno Matlab.** Cuando hayamos comprendido el funcionamiento de las herramientas de la localización de la dirección de procedencia, podemos elegir la que más nos convenga para poder integrarla en nuestro sistema, luego procedemos a su desarrollo y ajustamos los parámetros para poder integrarlo con las otras dos partes.

• Obj. 5. **Integrar detección, clasificación y localización en entorno Matlab.** Una vez tenemos los tres sistemas desarrollados individualmente, nos toca integrarlos realizando las tareas necesarias para ello, así como ajustando los parámetros necesarios. Finalmente, nuestro sistema completo estará preparado para su uso.

1.3 Antecedentes al Trabajo de Fin de Grado

Existen algunos sistemas que tratan de medir la intensidad de la lluvia empleando distintos sensores acústicos (micrófonos y captadores de vibraciones) [18-19]. También existen estudios centrados en la identificación de la escena sonora, indicando cuando está presente o no la lluvia [17]. Sin embargo, en este último ámbito no hemos encontrado ninguna implementación de los mismos, como ocurre en nuestro caso.
En el caso de los truenos tampoco hemos encontrado estudios ni desarrollos o productos comerciales que usen sensores de sonido para la detección de estos eventos meteorológicos, tampoco para su localización y seguimiento. Esto es debido a que, en este caso, la detección se realiza por destellos de luz emitidos por el trueno y no por sonido [51].

1.4 Estructura y resumen de los capítulos

Este trabajo de fin de grado se ha estructurado en cuatro grandes apartados: Memoria, Planos, Pliego de Condiciones y Presupuesto.

La MEMORIA está dividida en ocho capítulos, que cubrirán los siguientes aspectos:
Capítulo 1. “Introducción”. Se realiza una introducción al trabajo, se describen los objetivos a cumplir y, por último, se habla sobre la estructura y el resumen de los capítulos.
Capítulo 2. “Algoritmos de detección, clasificación y localización”. Se explica a nivel teórico los distintos algoritmos correspondientes a las distintas funcionalidades de nuestro sistema al igual que los pasos previos a llevar a cabo antes de implementar cada una de esas técnicas.
Capítulo 3. “Descripción y montaje del sistema”. Se describe el sistema, para ello se habla de los elementos que lo componen y su interconexionado. También se habla sobre la calibración de los micrófonos junto con su respuesta en frecuencia. Por último, se

comenta la agrupación final de micrófonos, la cual detectará con mayor precisión las direcciones de llegada.
Capítulo 4. “Síntesis de los escenarios sonoros a evaluar. Diseño y explicación”. Se comenta el diseño de los escenarios sonoros para realizar las pruebas de cada una de las funcionalidades de nuestro sistema y su finalidad.
Capítulo 5. “Resultados”. Se representan y se explican los resultados obtenidos para cada una de las técnicas (detección, clasificación y localización) que utiliza el sistema de manera individual al igual que los resultados cuando las tres técnicas están unidas.
Capítulo 6. “Implementación del sistema. Interfaz de usuario”. Se habla sobre la integración de las tres técnicas en el sistema y sobre el diseño de la interfaz de usuario.
Capítulo 7. “Conclusiones”. Se presenta lo que se ha aprendido tras realizar este trabajo, los logros alcanzados y cómo se podría mejorar o ampliar.
Capítulo 8. “Bibliografía”. Se detalla todo el material (libros, artículos científicos y páginas web) que se han usado para la ejecución de este TFG y la elaboración de su documentación.

PLANOS. Se esboza el modelo a tener en cuenta para la fabricación del soporte del array triangular de micrófonos, junto con sus medidas.

PLIEGO DE CONDICIONES. Se establecen las características de los equipos a emplear para las mediciones al igual que las del software empleado.

PRESUPUESTO. Se tasa el valor del proyecto, para ello se tiene en cuenta el tiempo de realización desde el establecimiento de las condiciones del trabajo a realizar, con la elección de las medidas, estudio, cálculos y simulaciones hasta los gastos derivados de la redacción de la memoria. Junto con la amortización de los materiales hardware y software que se han utilizado.

CAPÍTULO 2: ALGORITMOS DE DETECCIÓN, CLASIFICACIÓN Y LOCALIZACIÓN

[Esta página ha sido dejada intencionalmente en blanco]

2.1 Introducción

En este capítulo se expone a nivel teórico los distintos algoritmos correspondientes a las distintas funcionalidades de nuestro sistema (detección, clasificación y localización de la dirección de procedencia) empleados para llevar a cabo su desarrollo al igual que los pasos previos antes de implementar las técnicas que dan lugar a cada una de las funcionalidades.
Además, se detallará el proceso por el que se ha pasado para poder integrar cada técnica en el sistema, exponiendo lo que fue útil y lo que no lo fue.

2.2 Algoritmos de Detección

La función de los algoritmos de detección es la de detectar una señal inmersa en un entorno con ruido ambiente característico, estableciendo un umbral para distinguir la señal del ruido ambiente, el cual consideraremos silencio, además de marcar el principio y el final de la señal. Para calcular el umbral, primero hay que segmentar la señal en trozos sobre los que es más fácil realizar operaciones, como calcular la media o la varianza.
El algoritmo que usemos debe de cumplir una serie de características para poder integrarlo en nuestro sistema. En primer lugar, debe ser fiable o lo suficientemente robusto como para no permitir excesivas pérdidas de instantes de principio-fin. También, debe ser lo más exacto posible situando las marcas de principio y de fin. Seguidamente, debe ser adaptativo para poder trabajar bien en ambientes cambiantes, especialmente cuando el ruido de fondo lo es.
Por último, la simplicidad es otra característica deseada, especialmente cuando está pensado para formar parte de un sistema complejo.

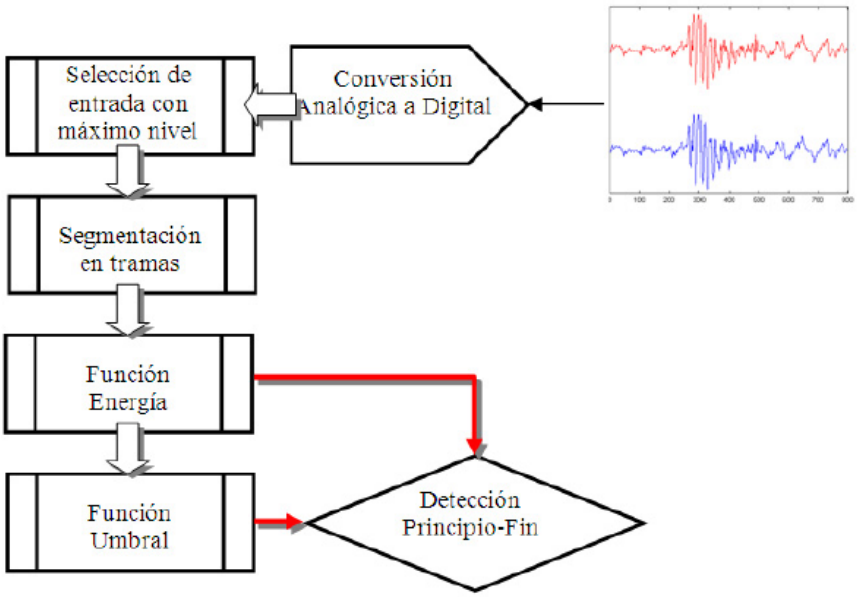


Figura 2. 1.- Esquema de un algoritmo de detección por umbrales [20].

Los algoritmos de detección mediante umbrales tienen una serie de pasos comunes. Estos son variables, pero los más completos incluyen los siguientes (figura 2.1);

conversión analógica-digital, filtrado de la señal en la banda frecuencial de interés, segmentación en tramas, función de energía, función umbral y detección principio-fin. Nosotros sólo aplicaremos detección a uno de los canales, concretamente al más energético. Se trata de un método sencillo ya existen multitud de algoritmos implementados para esta tarea. También se trata de un método efectivo puesto que cuanto mayor sea la relación señal más ruido a ruido mayor será la probabilidad de detección y menores las de falsa alarma y error [20].

2.2.1 Segmentación en Tramas

Para poder realizar un seguimiento de la evolución de la señal (p.e., mediante la función de energía) necesitamos tomar la misma en intervalos cortos llamados tramas. La segmentación en tramas es un aspecto importante del algoritmo que debe tratarse con cuidado por dos motivos. Primero, los estimadores de las funciones de detección requieren de un número suficiente de muestras. Y segundo, para obtener unas buenas medidas, las tramas deben ser tales que dentro de ellas se pueda considerar la señal estacionaria, excepto cuando haya un trueno, donde se produce un fuerte cambio de señal con incremento de su energía, momento en el que se puede aprovechar para hacer la detección. El tamaño final de las tramas estará fuertemente influido por unos resultados experimentales en los que buscamos satisfacer sobre todo la necesidad de obtener buena fiabilidad y pocas falsas alarmas [20].

2.2.2 Detección Basada en Umbrales

Esta detección consiste en hacer un análisis por tramas de la señal observada. Sobre cada trama se aplicará una función de energía a las muestras de señal y por comparación de la amplitud de la energía con un umbral establecido se decide si hay señal o silencio. Si la función rebasa el umbral se activa la detección hasta que vuelve a caer por debajo del mismo, instante en el que consideramos que ha vuelto el silencio.
Esto está reflejado en la figura 2.2.

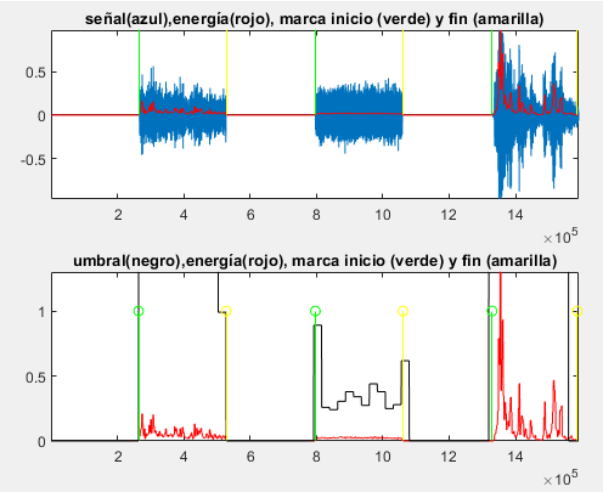


Figura 2. 2.- Comportamiento de la función de energía y de los umbrales.

El test de detección se realiza como sigue:

Sea la señal discreta s(n) un proceso de media cero. Sea n(n) un ruido discreto desconocido de media cero. Asumamos que los dos procesos s(n) y n(n) son estadísticamente independientes. La señal s(n) es observada en ruido aditivo, n(n), como y(n)=s(n)+n(n) con tramas de longitud N. El objetivo del test es detectar la presencia de señal en las observaciones ruidosas. Esta detección se hará a partir de una función de energía calculada al efecto a la que se le piden valores altos cuando hay señal y bajos cuando hay silencio, con o sin ruido.
De esta forma podremos obtener un umbral a través de los valores bajos de señal y compararlo con los valores altos para así sacar hipótesis acerca de la presencia de señal. Sin duda, una estrategia en la que se compare la función de energía con un umbral fijo puede ser adecuada en entornos donde el ruido es estacionario. Es más, tiene la virtud de utilizar un sólo umbral que una vez determinado queda fijo con lo cual todo resulta sencillo.
Tomaremos como umbral para nuestro algoritmo, la función $T_{\alpha}(t)$ definida en la ecuación 2.1. Consiste en considerar simultáneamente media y desviación estándar de la función de detección.

$$T_a(t) = Med(t) + Desv(t) \qquad \textit{Ec.2.1}$$

Donde la media y la desviación se definen en las ecuaciones 2.2 y 2.3 respectivamente.

$$Med(t) = \frac{1}{T}\sum_{t=1}^{T} T_T(t) \qquad \textit{Ec.2.2}$$

$$Desv(t) = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(T_T(t) - Med(t))^2} \qquad \textit{Ec.2.3}$$

Donde T es el número de muestras de la ventana para hacer los promedios. La elección de la longitud de la ventana de promedio *T* se ha hecho buscando una solución de compromiso entre precisión y fiabilidad.
Debemos hacer notar que nuestro algoritmo, aunque adaptativo y cambiante con el tiempo, trabaja con un único umbral. Otros algoritmos utilizan más (p.e., dos o tres) de la misma forma que utilizan más características. No hemos optado por soluciones de este tipo pues, los experimentos demuestran que se pueden obtener buenos resultados con un algoritmo más sencillo.
La primera vez que se sobrepasa el umbral anteriormente calculado, se coloca una marca de inicio y cuando la señal vuelve a ser menor que el umbral, se coloca una marca de fin y así a lo largo en toda la señal a procesar [20].

Para el sistema a desarrollar, emplearemos este algoritmo de detección, basado en umbrales, el cual ha sido de elaboración propia.

El código de la detección junto con el de todas las funciones en Matlab que se invocan para integrarse en el sistema y de esta manera, realizar la detección en tiempo real, se adjunta en el Anexo de este proyecto, a su vez en el apartado del Código de Matlab.

2.2.3 Detección Basada en la Energía

La función de detección basada en la energía es una elección razonable cuando deseamos detectar la presencia de una señal sobre un fondo de ruido estacionario. El ruido se asume de media cero y varianza σ_N^2. La señal también se asume de media cero y varianza $\rho \cdot \sigma_N^2$, donde ρ es la relación señal a ruido.
Sea una trama de la señal y(n) =s(n)+n(n) donde (n=1,2,…,N). Si se observa y(n) en un instante 't' cualquiera, la estadística del test es:

$$T_E(t) = \sum_{n=1}^{N} \frac{y^2}{\sigma_N^2} \qquad \textit{Ec.2.4}$$

Como podemos observar en la ecuación 2.4, se requiere el conocimiento de la varianza del ruido. En la práctica ésta se calcula tomando registros a partir de intervalos de silencio donde sólo hay ruido, y de éstos se obtiene una estimación de σ_N^2.
Para ello se toma H_0 como la hipótesis nula o presencia de ruido y H_1 como la alternativa, presencia de señal (ecuación 2.5).

$$H_1: y(n) = s(n) + n(n), n = 1,2, \dots, N$$
$$H_0: y(n) = n(n), n = 1,2, \dots, N \qquad \text{Ec.2.5}$$

El test de detección consiste en rechazar H_0 cuando la detección basada en la energía es mayor que un umbral, es decir, se cumple la ecuación 2.6:

$$T_E(t) > T\alpha \qquad \text{Ec.2.6}$$

donde la ecuación 2.7 es la probabilidad de falsa alarma y Tα se puede determinar a partir de experimentos.

$$\mathrm{P}(\chi_N > T\alpha | H_0) = \alpha \qquad \textit{Ec.2.7}$$

La probabilidad de detección es la probabilidad de que se cumpla la ecuación 2.8.

$$P(T_E(t)|(1+\rho)) > T\alpha \qquad \textit{Ec.2.8}$$

Atendiendo a estas dos situaciones se puede definir un algoritmo de detección con sólo hacer un seguimiento efectivo de los cambios de estado, silencio (H_0) y señal (H_1) [20].

2.2.4 Postprocesado de la Detección

Un buen método de adaptación de los umbrales no es suficiente para obtener buenos resultados. Por ejemplo, es común detectar segmentos demasiado cortos que no permitirán una buena estimación, pues no conllevan un número significativo de muestras de señal. El objetivo principal que nos marcamos con el postprocesado es eliminar pulsos cortos de señal cuya duración permita sospechar que corresponden a ráfagas de ruido [20].

2.3 Algoritmos de Clasificación

El Aprendizaje Automático (Machine Learning) tiene como objetivo desarrollar algoritmos que permitan aprender a partir de un conjunto de observaciones, de tal manera que sea posible establecer hipótesis generales o predicciones sobre nuevas observaciones, y de esta manera, tomar decisiones automáticamente. O lo que es lo mismo, asignarles a los datos de entrada una clase atendiendo a distintos patrones característicos de cada clase, dotando así a un sistema de Inteligencia Artificial.
Los métodos se denominan supervisados cuando se utiliza información a priori (como etiquetas o clases). De lo contrario se les llama no supervisados. Nosotros emplearemos un método supervisado.
El Aprendizaje Automático es utilizado en una gran variedad de ámbitos, desde el reconocimiento de voz, hasta la exploración o minería de datos, existiendo una gran variedad de algoritmos.
Este tipo de algoritmos tienen dos fases fundamentales: la fase de entrenamiento y la fase de evaluación.

La **fase de entrenamiento** consiste principalmente en proporcionar observaciones o ejemplos al sistema, de los cuales éste debe aprender.
En este trabajo se utilizará el aprendizaje supervisado, proporcionando siempre al sistema la etiqueta correcta de cada observación durante la fase de entrenamiento.
La **fase de evaluación o testing** es llevada a cabo una vez el sistema ha sido entrenado. Esta fase consiste en proporcionar al sistema un subconjunto de datos o ejemplos, que el propio sistema debe etiquetar. Conociendo las verdaderas etiquetas de los datos proporcionados sería posible verificar los resultados de la evaluación, caracterizando así al sistema en función de sus aciertos y fallos.

Una vez el sistema se ha entrenado y evaluado, obtenemos la matriz de confusión, donde comprobamos que clasifica correctamente o coloca en su lugar, las clases para las que ha sido entrenado.
Es importante mencionar que todo algoritmo de Aprendizaje Automático basado en Reconocimiento de Patrones requiere un procesado previo de la información con la cual se trabajará. Este procesado consiste en extraer las características o features relevantes de la información a tratar (Feature Extraction) [21], las cuales serán los datos para introducir al sistema, tanto en la fase de entrenamiento como en la fase de evaluación. Es decir, si se trata de una aplicación de audio, será necesario extraer características

clave del audio, al igual que si se trata de una aplicación de imagen, será necesario procesar las imágenes para obtener las características más relevantes.

En este tipo de algoritmos existen dos tipos de aciertos:
• Verdadero Positivo (True Positive): probabilidad que tiene un sistema de aceptar correctamente muestras pertenecientes a la clase considerada positiva.
• Verdadero Negativo (True Negative): probabilidad que tiene un sistema de aceptar correctamente muestras que pertenecen a la clase considerada negativa.
En este tipo de algoritmos existen dos tipos de fallos:
• Falso Positivo (False Positive): probabilidad que tiene un sistema de rechazar erróneamente muestras pertenecientes a la clase considerada positiva.
• Falso Negativo (False Negative): probabilidad que tiene un sistema de rechazar erróneamente muestras pertenecientes a la clase considerada negativa.

Para poder realizar estas observaciones debemos establecer previamente quiénes son los positivos y quienes son los negativos.
Por ejemplo, nuestro sistema está entrenado para tres clases, truenos, lluvia y ruido ambiente.
Lo positivo es lo que queremos detectar y lo negativo es lo que no queremos detectar.
Supongamos que queremos detectar truenos, estos serían los positivos y no queremos detectar ni ruido ambiente, ni lluvia, estos serían los negativos.
Para probar el sistema, sería necesario realizar un procesado de audio en tiempo real, bien por archivos de audio o bien por micrófonos (sonido en vivo), que contenga sonidos de las tres clases, extrayendo las características del mismo y normalizándolas posteriormente.
Habrá acertado en muchos casos detectando los truenos correctamente (verdadero positivo) o bien, la lluvia y el sonido ambiente correctamente (verdadero negativo). Sin embargo, habrá un determinado porcentaje de casos en los que el sistema haya fallado, clasificando los truenos como lluvia o sonido ambiente (falso negativo), y por último, otro porcentaje que haya sido aceptado como truenos cuando realmente no se trataba de ellos (falso positivo).
Generalmente, los algoritmos de Machine Learning son utilizados en dos tipos de aplicaciones, clasificación y regresión. Sin embargo, en este trabajo únicamente se tratarán los algoritmos con el objetivo de clasificar, concretamente, señales de audio.

2.3.1 Pasos previos

Queremos crear un clasificador automático para el caso supervisado, el enfoque más directo y tradicional se basa en una fase de entrenamiento en la que se construyen los modelos o plantillas de cada clase.

En la figura 2.3 se muestra el esquema general del proceso de desarrollo de un clasificador [24]. Como se puede apreciar, el primer paso tras determinar cuál es el problema a resolver es recopilar una base de datos (database). La elección de la base de datos a utilizar es un aspecto fundamental del clasificador, ya que se trata de los ejemplos de los cuales éste debe aprender. Una vez definida la base de datos, es necesario procesar cada uno de los ejemplos para extraer las características más

significativas (feature extraction). Todas las features seleccionadas conforman, para todas las observaciones de la base de datos, el conjunto de datos de entrenamiento (training dataset), entrenando así el algoritmo elegido.
Estas características se usan luego como plantillas de referencia o para construir un modelo estadístico para cada clase.

Es conveniente elegir adecuadamente el algoritmo de clasificación a utilizar, para de esta forma obtener los resultados óptimos. También se debe reservar una parte de la base de datos para definir un conjunto de datos de test o evaluación (test dataset) para caracterizar el clasificador.
Finalmente, se aceptará o no el clasificador resultante en función de los resultados obtenidos. En caso de no ser aceptado, se podrá recurrir a la adquisición de nuevos datos de entrenamiento, la búsqueda de nuevas features que permitan obtener información relevante, la elección de otro algoritmo de clasificación o la configuración de los parámetros que intervienen en el algoritmo utilizado.

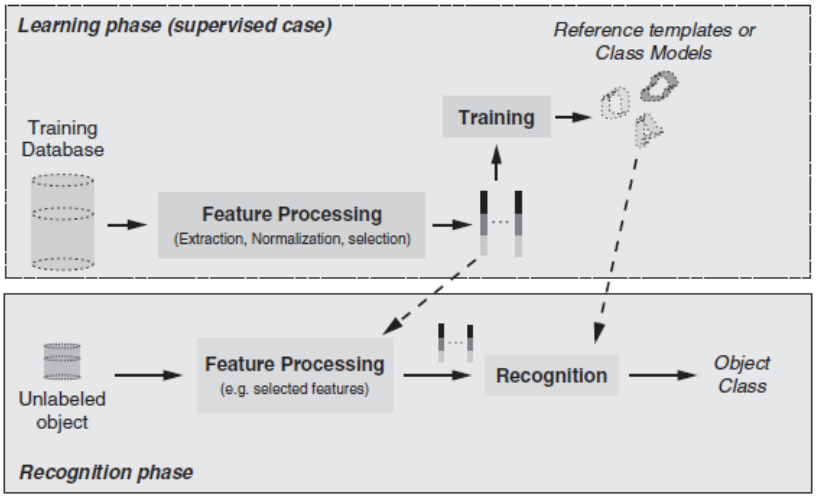


Figura 2. 3.- Arquitectura de un sistema de clasificación automático [24].

Por último, en la fase de reconocimiento, cuando se presentan objetos sin etiquetar en tiempo real al sistema, en nuestro caso archivos de audio o de sonido cuasi-real, el mismo conjunto de características se extraen, se procesan (en nuestro caso extrayendo y normalizando las características) y se compara con las plantillas o modelos ya entrenados para así determinar cuáles son las clases más probables a las que pertenece ese objeto.

2.3.1.1 Bases de Datos

Siguiendo el esquema de la figura 2.3, el primer paso sería tener una idea clara de qué tipo de información se desea clasificar. El objetivo de este trabajo consiste en clasificar señales de audio en un conjunto de clases, por lo que la información básica vendrá dada en forma de archivos de audio.

El trueno y la lluvia que hemos utilizado para entrenar al sistema, los hemos obtenido de la página 99 Sounds [38].

Nos hemos descargado una biblioteca de sonido gratuita llamada “99sounds Rain and Thunder” con 64 muestras de audio de alta calidad que, según sus especificaciones se registraron durante una tormenta de primavera en las calles de Belgrado y se grabaron a una profundidad de 24 bits utilizando un grabador de campo estéreo Edirol R-09 portátil.
Esta biblioteca contiene varios tipos de grabaciones de truenos y lluvia, aunque nosotros para entrenar y probar el sistema, hemos utilizado los truenos lejanos, truenos fuertes, lluvia cayendo en un desagüe y lluvia cayendo en la ventana.
Además, el sonido ambiente lo hemos captado desde la azotea del Edificio de Telecomunicación y Electrónica conectando el array de tres micrófonos formando un triángulo equilátero a la grabadora.

En la base de datos, las clases deben estar lo más diferenciadas posibles, o lo que es lo mismo, los archivos de audio de cada clase deben incluir sólo sonidos de esa clase para evitar posibles confusiones posteriores por parte del sistema, también los archivos de la base de datos tienen que estar en formato .wav.

Para empezar a preparar las bases de datos, debemos normalizar todos los sonidos que van a constituir nuestra base de datos. Como se puede apreciar en la ecuación 2.9, normalizar consiste en dividir toda la señal, entre el valor absoluto de su máximo, de tal manera que el máximo sea 1 para todas las señales y así no haya diferencias de amplitudes.

$$y_n = \frac{y}{máx(|y|)} \qquad \textit{Ec.2.9}$$

En la figura 2.4 se observa la diferencia entre una señal de trueno sin normalizar y otra normalizada, la gran diferencia está en las amplitudes de la señal de la derecha, que su máximo es la unidad.

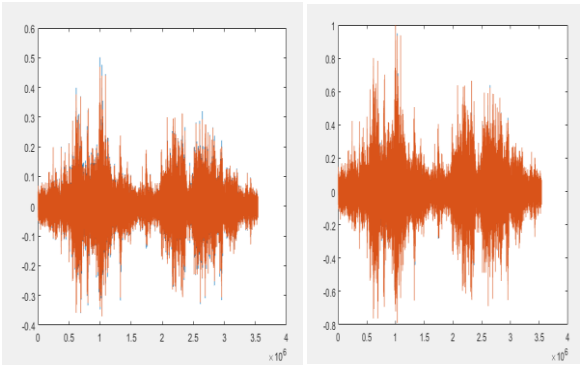


Figura 2. 4.- Señal de un trueno sin normalizar (izquierda) y normalizada (derecha).

Ahora ya podemos empezar a marcar los archivos de audio de las clases. Para ello, creamos documentos de texto en formato .txt que contengan las marcas de inicio y de final, las marcas no son otra cosa que el número de muestras de audio de las partes más significativas de cada archivo de audio perteneciente a cada clase. Posteriormente, realizamos la extracción de características sólo en estas zonas.

Para la clase truenos, la hemos marcado de manera que queden las partes más significativas de los truenos en medio de las dos marcas.
En los archivos de audio de truenos largos, aprovechamos cada parte significativa pudiendo tener una o más marcas cada archivo de audio de trueno a analizar.
En esta misma clase, debemos tener cuidado ya que los archivos de sonido de truenos pueden contener lluvia u otros sonidos, hay que intentar que estos otros sonidos no se cuelen en medio de las marcas para posteriormente realizar un correcto entrenamiento y test de nuestro sistema además de evitar posibles fallos.
En la clase lluvia, primero se escuchan los archivos de audio en un editor de audio como Audacity o similares y se eliminan otros sonidos que no sean esta clase, luego se vuelven a reconstruir.
En la clase sonido ambiente, no hace falta escucharlo porque al grabarlo nos hemos asegurado de que sólo existe sonido ambiente.
Para la clase lluvia y sonido ambiente se han dividido los archivos de audio pertenecientes a estas clases en fragmentos de 1,5 segundos con una rutina creada en Matlab.

Posteriormente, dividimos cada archivo de audio de estas dos últimas clases en partes de un segundo y medio de duración y también ponemos marcas al inicio y al final con rutinas creadas en Matlab que realizan estas dos tareas de manera automática.
Para ver exactamente el número de muestra de la marca de inicio y de final, se abren los archivos de audio en un editor de audio como Audacity y el documento de texto generado, luego nos asegurarnos de que las marcas de trueno están bien situadas, para ello se puede escuchar lo que está entre las marcas, utilizando otra pequeña rutina en Matlab que reproduce desde la muestra de inicio a la muestra de final.

Llegados hasta aquí en todas las clases, no nos importará lo que esté en medio de las marcas porque ya nos hemos asegurado de que los archivos de audio sólo contienen sonidos de esas clases.

Cuando hayamos cortado todos los archivos de audio de las tres clases y creado una carpeta para cada clase, usamos el 70% de las particiones para la fase de entrenamiento y el 30% de las particiones restantes para la fase de test. Entonces, cada carpeta correspondiente a cada una de estas dos últimas clases contendrá el 70% de sus archivos de audio para train y el 30% para test.

Después de todo lo anterior, procedemos a crear las listas de entrenamiento y de test, o benchmarks de entrenamiento y de test.
Los benchmarks son documentos de texto en formato txt que contienen los directorios donde se encuentran cada uno de los audios, estos últimos en formato wav.
Cada una de las clases a clasificar, se corresponde con dos listas o benchmarks, una de entrenamiento y otra de test que definirán las particiones que se usarán para train y las que se usarán para test.
Ahora, ya podemos extraer las características o features, que serán archivos en formato mat.

2.3.1.2 Pre-procesado

Tal y como se ha comentado anteriormente, es necesario depurar la información a clasificar para crear el denominado training dataset o conjunto de datos de entrenamiento. Para ello, es necesario realizar un pre-procesado de cada audio con el objetivo de extraer la información más relevante que permita resolver el problema en cuestión, en este caso, clasificar un conjunto de señales de audio en un determinado número de clases. Por ello, la extracción de características es una parte fundamental dentro cualquier sistema de Reconocimiento de Patrones.

El proceso de obtención de características del audio que describan al mismo en diferentes aspectos es denominado Feature Extraction. De tal forma que cada descriptor o feature aporte algún tipo de información sobre el audio. Al realizar el pre-procesado del audio, cada feature tomará un valor (feature value), y cada audio será descrito mediante un conjunto de features, es decir, cada audio será descrito con un conjunto de valores, formando así un ejemplo del training dataset, que a su vez contiene todos los audios o ejemplos de entrenamiento. Es importante destacar que a la hora de definir un training dataset, todos los ejemplos que forman el mismo deben estar descritos por exactamente las mismas features, o lo que es lo mismo, a cada audio se le debe extraer exactamente las mismas features.

Existen diferentes tipos de features [23], y cada una de ellas aporta en teoría una información distinta. Si bien, es normal que entre distintas features haya cierta correlación, lo cual no es deseable, ya que, si dos features tienen alta correlación entre sí, estarán aportando la misma información, pudiendo ser redundante una ellas. Por el contrario, cuanta menos correlación haya entre features, más información distinta se tendrá para describir el audio. Es importante evitar la redundancia entre features para que el funcionamiento del clasificador sea más eficiente, al igual que se prefieren features que proporcionen la mejor separabilidad de las clases.

El proceso de selección de features (feature selection) normalmente puede ir asociado a un conocimiento previo del problema a resolver, utilizando así features cuyo significado permita discriminar entre clases distintas. Sin embargo, es muy común realizar el proceso extrayendo todas las features posibles para posteriormente probar cuáles de ellas en conjunto proporcionan el mejor resultado posible.

2.3.1.3 Feature Extraction

Como paso previo a la extracción de dichas features, es habitual realizar algún tipo de mejora en el audio a tratar, por ejemplo, eliminar ruido y componente continua, o incluso separación de fuentes. En nuestro caso, no hemos realizado ningún algoritmo de supresión de ruido ni separación de fuentes, pero sí se ha contemplado la eliminación de componente continua o DC Offset (Direct Current).

En el proyecto que tomamos como referencia [40], se suprimía la componente continua con un filtro IIR paso alto (Infinite Pulse Response), el cual viene determinado por la ecuación 2.10, en este caso $\alpha = 0.95$.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1 - z^{-1}}{1 - \alpha z^{-1}} \qquad \textit{Ec.2.10}$$

Aunque hicimos varias pruebas con ese filtro, observamos que restándole la media a la señal original (Ec.2.11) para eliminar la componente continua, obtuvimos mejores resultados.

$$y = y - \bar{y} \qquad \textit{Ec.2.11}$$

Seguidamente, el audio de entrada es diezmado a una frecuencia de muestreo de 16000 Hz. Esto es debido a que no todos los audios de las bases de datos tienen la misma frecuencia de muestreo. Por ello, se ha elegido como frecuencia de muestreo la mínima posible para poder utilizar el máximo número de audios de las bases de datos.

En este proyecto se extraerán características en dos situaciones: cuando se introduzcan nuevos archivos de audio que sirvan para entrenar el sistema en clases ya existentes o entrenar nuevas clases y la otra situación es cuando el sistema realiza la clasificación en tiempo real.

2.3.1.4 Clasificación

Una vez extraídas todas las features, se crean otros benchmarks, pero esta vez los que contienen los directorios donde se encuentran cada una de las características o features en formato mat. Cada una de las clases a clasificar se corresponde con dos listas o benchmarks de features, una de entrenamiento y otra de test.
Seguidamente podremos elegir el clasificador para empezar a entrenar el sistema.

Entonces se construirá el training dataset, que será una matriz o una tabla que contenga todas las features para cada uno de los ejemplos de entrenamiento, incluyendo la clase a la que pertenece cada uno de los ejemplos (aprendizaje supervisado), pudiéndose realizar un entrenamiento del clasificador, según el algoritmo de clasificación de Aprendizaje Automático que hayamos elegido, en nuestro caso KNN o Random Forest como veremos a continuación. Un ejemplo de formato de training dataset podría ser el mostrado en la tabla 2.1 [22].

Tabla 2. 1.- Ejemplo de training dataset.

Audio	**Feature 1**	**Feature 2**	**…**	**Feature n**	**Clase**
1	F_1	F_2	…	F_n	Trueno
2	F_1	F_2	…	F_n	Trueno
…	F_1	F_2	…	F_n	Lluvia
200	F_1	F_2	…	F_n	Lluvia
…	F_1	F_2	…	F_n	Ambiente
400	F_1	F_2	…	F_n	Ambiente
…	F_1	F_2	…	F_n	Ambiente

Es habitual realizar la media y varianza de cada una de las columnas (features) y realizar una normalización para obtener una variable aleatoria normal estandarizada. Es decir, $X \sim N(\mu, \sigma^2)$ (variable aleatoria normal) tiene su propia media y varianza y, para normalizar esta variable a una distribución normal estándar de media cero y varianza unidad, es necesario definir la nueva variable aleatoria normal estandarizada $Z \sim N(0,1)$, la cual se muestra en la ecuación 2.12. Esta normalización será aplicada de igual forma a cada nueva instancia que se presente al clasificador, utilizando la media y varianza obtenidas en el training dataset.

$$Z = \frac{X - \mu}{\sigma^2} \qquad \textit{Ec.2.12}$$

El formato del test dataset sería igual que el dataset de entrenamiento salvo que no se incluiría la clase a la que pertenece cada ejemplo. Este test dataset determinará si el clasificador cumple con los requisitos necesarios. En caso de obtener resultados insatisfactorios, habría que revisar las características que tenemos para eliminar las que no distingan las clases, o bien dejar las que ya estaban que sí las distingan, o bien, añadir nuevas características.

Para evaluar los resultados de la clasificación (fase de test) existen diferentes técnicas. La más común de todas ellas es dividir la base de datos para tener un gran porcentaje para entrenamiento (training dataset), y el resto para test (test dataset).
En nuestro caso, hemos dividido el 70% para el test y el 30% para el train.
Los resultados de la clasificación vienen generalmente dados en forma de matriz de confusión. Consiste en una matriz cuyas filas y columnas se corresponden con las clases, concretamente las filas serían las clases reales, y las columnas las clases predichas. De esta manera, el resultado óptimo sería que cada ejemplo de test sea predicho como la clase que realmente es, lo cual situaría en la diagonal de la matriz. Un ejemplo de la matriz de confusión se refleja en la figura 2.5.

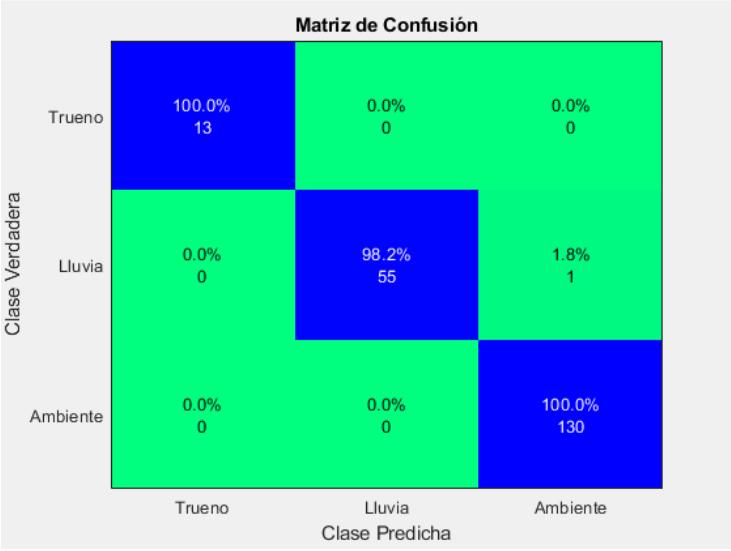


Figura 2. 5.- Matriz de confusión.

2.3.2 Parametrización

En este apartado, definiremos cada una de las características que se han elegido. Para ello, nos toca hablar de la parametrización, consiste en crear vectores que se componen de las características extraídas en cada uno de los segmentos de audio marcados. De esta manera, podremos entrenar el sistema y construir los clasificadores según los patrones que caracterizan a cada clase, así también podremos realizar la clasificación en tiempo real.
La extracción de características se realiza en segmentos de tiempo cortos y superpuestos (o en ventanas) a las que luego se realiza la FFT, para por último obtener las características espectrales, cepstrales o temporales que constituirán los vectores de características. Esto se refleja en la figura 2.6 [24].

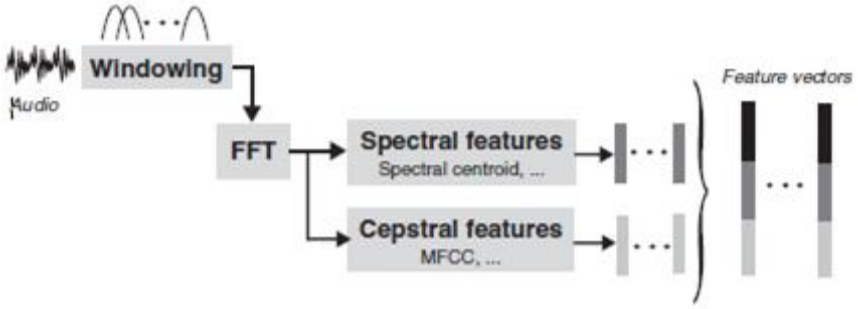


Figura 2. 6.- Extracción de características en forma de vector [24].

En nuestro caso, la elección de las características fue la tarea a la que más tiempo y trabajo le dedicamos ya que se hicieron pruebas con gran variedad de estas, no fue hasta que se nos ocurrió representar las características sobre la gráfica que nos dimos cuenta cuáles resultaban útiles y cuáles no.
En los siguientes subapartados se enumerarán las características descartadas y las que finalmente se eligieron.
Las únicas características que han sido representadas son las que finalmente se eligieron, las cuales se encuentran en su subapartado correspondiente. Las que no nos sirvieron no tiene sentido representarlas en la gráfica ya que no aportan información válida.

2.3.2.1 Características descartadas

Es importante reducir el número de features, ya que, si tenemos muchas, puede existir alta correlación entre ellas, redundancia, aportando la misma información o pueden tener amplitudes insignificantes. Estas condiciones no son deseables.

A continuación, enumeraré las características que se probaron pero que no fueron útiles para este proyecto:

· **Dispersión Espectral (Spectral Spread):** El spread o dispersión espectral define la concentración de energía alrededor de la media [21], es decir, del centroide. En términos estadísticos se corresponde con la varianza, que se obtiene según la ecuación 2.13,

siendo su raíz la desviación típica σ, la cual se utiliza para el cálculo de parámetros de orden superior (como la asimetría o la curtosis espectral).

$$\sigma^2 = \int (x-\mu)^2 \cdot p(x)\delta x \qquad \textit{Ec.2.13}$$

· **Curtosis Espectral (Spectral Kurtosis):** La curtosis espectral describe el grado de planicie o "picudez" del espectro alrededor del centroide. La curtosis es calculada según la ecuación 2.14 [21].

$$\begin{cases} m_4 = \int (x-\mu)^4 \cdot p(x)\delta x \\ \gamma_2 = \dfrac{m_4}{\sigma_4}\ (kurtosis) \end{cases} \qquad \textit{Ec.2.14}$$

· **Roll-off Espectral** (**Spectral Roll-off):** El roll-off también considera el espectro como una distribución de probabilidad, siendo éste el valor para el cual se supera una determinada probabilidad en la distribución de probabilidad acumulada [21]. En términos espectrales, el roll-off sería aquella frecuencia para la cual se tiene el 85 % de la energía por debajo de la misma (distribución de energía acumulada), tal y como se indica en la ecuación 2.15.

$$\sum_0^{fc} a^2(f) = 0{,}85 \sum_0^{fs/2} a^2(f) \qquad \textit{Ec.2.15}$$

· Pendiente Espectral (Spectral Slope): El slope o pendiente espectral describe el grado de decaimiento de la amplitud espectral. Se obtiene directamente de realizar una regresión lineal, obteniendo así la pendiente de la recta [21]. El cálculo se obtiene directamente de la ecuación 2.16, donde a(k) es la amplitud correspondiente al bin frecuencial k, y f(k) la frecuencia correspondiente a dicho bin frecuencial.

$$\begin{cases} \hat{a}(f) = slope \cdot f + constante \\ slope = \dfrac{1}{\sum_k a(k)} \cdot \dfrac{N\sum_k f(k)\cdot a(k) - \sum_k f(k) \cdot \sum_k a(k)}{N\sum_k f^2(k)\cdot a(k) - ((\sum_k f(k))^2} \end{cases} \qquad \textit{Ec.2.16}$$

· Decaimiento Espectral (Spectral Decrease): Al igual que el slope, el decaimiento o decrease describe el grado de decaimiento de la amplitud espectral. Sin embargo, el cómputo de esta feature tiene mayor similitud con la percepción humana [21]. El cálculo del spectral decrease viene dado por la ecuación 2.17.

$$decrease = \frac{1}{\sum_{k=2}^{K} a(k)} \cdot \sum_{k=2}^{K} \frac{a(k) - a(1)}{k - 1} \qquad \textit{Ec.2.17}$$

· **Variación Espectral (Spectral Flux):** La variación espectral, también denominada spectral flux, aporta información acerca de la variación espectral que existe entre frames consecutivos. Esta feature es útil para diferenciar entre música y sonidos del entorno [37]. Se calcula según la ecuación 2.18 [21], obteniendo el flux correspondiente a una trama t, que depende de la amplitud del espectro de la trama anterior a(t-1, k). En ocasiones, también es calculado como la diferencia cuadrática entre espectros [37].

$$flux(t) = 1 - \frac{\sum_k a(t-1,k)a(t,k)}{\sqrt{\sum_k (t-1,k)^2} \cdot \sqrt{\sum_k (t,k)^2}} \qquad \textit{Ec.2.18}$$

· **Audio Spectrum Centroid (MPEG-7):** El Audio Spectrum Centroid (ASC) de MPEG-7 describe el centroide espectral en una escala de frecuencia logarítmica del espectro, definiendo así el centro de gravedad del mismo de una forma más perceptual [37]. Antes de proceder al cálculo del mismo, se realiza una suma de aquellas frecuencias por debajo de 62.5 Hz, estableciendo como frecuencia central de dicha banda 31.25 Hz. De esta forma, se evita que el cálculo del ASC proporcione demasiado peso a la componente continua y las bandas de muy baja frecuencia. El cálculo se obtiene según la ecuación 2.19 [1] tras haber obtenido el espectro en potencia mediante la FFT.

$$ASC = \frac{\sum_{k'=0}^{\frac{N_{FFT}}{2}-K_{low}} \log_2\left(\frac{f'(k)}{1000}\right) P'(k')}{\sum_{k'=0}^{\frac{N_{FFT}}{2}-K_{low}} P'(k')} \qquad \textit{Ec.2.19}$$

Donde $Klow$=⌊62.5 $Hz/\Delta F⌋$, siendo ΔF la resolución espectral ($\Delta F = fs/NFFT$) y f'(k') y P'(k'), frecuencias y potencias para los nuevos valores de los índices de las frecuencias discretas (k') una vez unificada la baja frecuencia (es decir, f(k'=0)=31,25 Hz).

· **Audio Spectrum Spread (MPEG7):** Al igual que el ASC, el Audio Spectrum Spread aporta la misma información que el spectral spread, salvo que utiliza la escala de frecuencias logarítmica, indicando la distribución del espectro alrededor del ASC [1]. El cálculo se obtiene mediante la ecuación 2.20.

$$ASS = \sqrt{\frac{\sum_{k'=0}^{\frac{N_{FFT}}{2}-K_{low}} \left[\log_2\left(\frac{f'(k)}{1000}\right) - ASC\right]^2 P'(k')}{\sum_{k'=0}^{\frac{N_{FFT}}{2}-K_{low}} P'(k')}} \qquad \textit{Ec.2.20}$$

· **Audio Spectrum Flatness (MPEG7):** El Audio Spectrum Flatness (ASF) es otra feature de bajo nivel de MPEG-7, la cual realiza una comparación del espectro de la señal con un espectro totalmente plano. Esta comparación se realiza por bandas de frecuencia, para posteriormente realizar una media de todas las bandas, obteniendo un único escalar que describe la planicie total. La planicie espectral de una banda viene dada como la ratio entre la media geométrica y la media aritmética de los coeficientes del espectro en potencia de dicha banda [37]. Valores bajos de ASF implicarán presencia de componentes armónicas, mientras que valores altos implicarán ruido o señales impulsivas.

· **Zero Crossing Rate (ZCR):** El Zero-Crossing Rate es un tipo de feature obtenida en el dominio temporal, mide el número de cruces por cero de una señal. Los sonidos que presentan periodicidad suelen tener menor valor de ZCR que aquellos con gran presencia de ruido [21]. El cálculo del ZCR puede obtenerse según la ecuación 2.21 [1], donde N es el número de muestras de la señal s(n) y fs su frecuencia de muestreo.

$$ZCR = \frac{1}{2}\left(\sum_{n=1}^{N-1} |signo\big(s(n)\big) - signo(s(n)-1)|\right)\frac{fs}{N} \qquad \textit{Ec.2.21}$$

· **Pitch:** El pitch es una de las percepciones básicas de un sonido, describe la frecuencia percibida en una escala de altura, grave-agudo, esta feature es obtenida en el dominio temporal. Si bien, también puede entenderse como la frecuencia fundamental de un sonido. Esta feature puede venir dada en forma de histograma (Pitch Histogram), donde el análisis del pitch viene determinado generalmente por notas musicales.

En este caso, el pitch ha sido orientado hacia la búsqueda de la frecuencia fundamental, para ello, existen distintos métodos, como la autocorrelación, el dominio espectral y el dominio cepstral, o una combinación de estos [23]. Por lo general, la obtención de la frecuencia fundamental comienza por el cálculo de la autocorrelación, precisando posteriormente con alguna otra técnica de las mencionadas. El pitch que se probó en este trabajo viene dado únicamente por la autocorrelación.

La autocorrelación permite obtener patrones repetitivos dentro de una misma señal. Ésta consiste en una multiplicación de la señal original por ella misma desplazada un intervalo de tiempo, por ello, cuando este intervalo coincide con un periodo de la señal, la autocorrelación toma un valor alto, ya que, si la señal es pseudo-periódica, dos periodos consecutivos serán muy parecidos entre sí. En el caso de una señal de ruido blanco, la autocorrelación únicamente tendrá un valor alto con desplazamiento nulo, ya que la señal es completamente aleatoria y no existirá ningún proceso repetitivo. La autocorrelación se define según la ecuación 2.22.

$$R_x(m) = E\{x[n]x[n-m]\} \qquad \textit{Ec.2.22}$$

· **Índice de modulación AM (AM Index):** El índice de modulación se ha venido utilizando en acústica como índice de degradación de la señal acústica a la hora de evaluar la

inteligibilidad del habla en un recinto. Para calcular este índice, se obtiene la envolvente de la señal, extrayendo su valor máximo y su valor medio por cada segundo. El cálculo se realiza para cada segundo según la ecuación 2.23, promediando los resultados obtenidos. Esta feature es obtenida en el dominio temporal [40].

$$m(\%) = 100 \cdot \frac{I_{max} - I_0}{I_{max}} \qquad \textit{Ec.2.23}$$

· **Tiempo de ataque logarítmico (Log-Attack Time):** El tiempo de ataque es un tipo de feature obtenida en el dominio temporal. Describe el tiempo de ataque de una señal (aplicando después un logaritmo).

En este trabajo, únicamente se ha probado la extracción del ataque, ya que es un descriptor perceptual importante [21]. Para ello, se determinan los puntos de comienzo y fin del incremento de la energía durante el ataque, estableciendo como *threshold* de comienzo el 25 % de su valor RMS (*Root Mean Square)* de la señal, aplicando posteriormente la ecuación 2.24.

$$LogAttackTime = \log_{10}(stop_{attack} - start_{attack}) \qquad \textit{Ec.2.24}$$

A parte de las comentadas anteriormente, probamos con nuevas características, las cuales tampoco fueron útiles, estas se enumeran a continuación:

· **Características que parten de los momentos de onda temporales:** Estos momentos permiten representar diferentes características de la forma de onda en el dominio del tiempo. Se definen desde los cuatro primeros momentos centrales resultantes de la ecuación 2.25 [24].

$$\mu_i = \frac{\sum_{n=0}^{N-1} n^i x(n)}{\sum_{n=0}^{N-1} x(n)} \qquad \textit{Ec.2.25}$$

Entre ellas se encuentran:

1) **Centroide Temporal (Temporal centroid):** Describe el centro de gravedad de la forma de onda y viene dado por la ecuación 2.26.

$$Sc = \mu_1 \qquad \textit{Ec.2.26}$$

2) **Ancho temporal (Temporal width):** Describe la dispersión alrededor del valor medio y viene dado por la ecuación 2.27.

$$Sc = \sqrt{\mu_2 - \mu_1^2} \qquad \textit{Ec.2.27}$$

3) **Asimetría temporal (Temporal Assimetry):** Representa la asimetría temporal de la forma de onda sobre su media y su definición parte de la asimetría temporal (temporal skewness), viene dada por la ecuación 2.28.

$$Sa = \frac{2\mu_1^3 - 3\mu_1\mu_2 + \mu_3}{S_w^3} \qquad \textit{Ec.2.28}$$

4) **Planicidad temporal (Temporal Flatness):** Representa la planicidad en conjunto de la forma de onda en el dominio del tiempo y viene definida por la curtosis espectral (ecuación 2.29).

$$Sf = \frac{-3\mu_1^4 + 6\mu_1\mu_2 - 4\mu_1\mu_3 + \mu_4}{S_w^4} - 3 \qquad \textit{Ec.2.29}$$

· **Factor de cresta (Crest factor):** Se usa mayormente en la corriente alterna o en el sonido y muestra la relación entre el valor de pìco y el valor RMS de la forma de onda (ecuación 2.30) [10].

$$FC = \frac{|\max(x)|}{x_{RMS}} \qquad \textit{Ec.2.30}$$

Para convertirlo a dB se usa la ecuación 2.31.

$$FC_{dB} = 20\log(FC) \qquad \textit{Ec.2.31}$$

· **Dinámica de la señal (Signal Dynamic Range):** Es la diferencia de nivel de presión sonora existente entre los sonidos más bajos y los sonidos más altos que podemos escuchar. Se mide en decibelios y su ecuación es la 2.32 [9].

$$DI_{dB} = DI_{máx\ dB} - DI_{mín\ dB} \qquad \textit{Ec.2.32}$$

· **Percentiles 10, 50 y 90:** Nivel de presión sonora ponderado en frecuencia y ponderado en el tiempo superado en el 10,50 y 90% respectivamente del intervalo de tiempo considerado [8].

· **Slew Rate:** Se define como el rango máximo de cambio de la tensión de salida para las máximas señales de entrada posibles (ecuación 2.33), por lo que limita la velocidad de funcionamiento, es decir la frecuencia máxima a la que puede funcionar el amplificador para un nivel de señal de salida dado [35].

$$SR = \left.\frac{dV_0(t)}{dt}\right|_{máx} \qquad \textit{Ec.2.33}$$

Transformada discreta del Coseno (DCT): Al principio se valoró realizar operaciones a la DCT tales como la DCT al cuadrado o el valor absoluto de la DCT, sin usar los MFCC (Mel

Frequency Cepstral Coefficients), aunque obtuvimos mejores resultados usándolos. La DCT se explicará con más detalle en el siguiente subapartado, ya que finalmente se utilizó para obtener los MFCC de los cuales también hablaremos a continuación.

· **La primera y segunda derivada de los MFCC**: Consiste en aplicarle la primera derivada(Δ-MFCC) o la segunda derivada (ΔΔ-MFCC) a los coeficientes MFCC los cuales se explicarán a continuación, en el siguiente subapartado.

2.3.2.2 Características elegidas

Finalmente, atendiendo a las características elegidas, las dividimos en dos categorías:

• **Dominio Cepstral**: Se obtienen mediante la Transformada Discreta de Fourier Inversa (IDFT – Inverse Discrete Fourier Transform).

o Mel-Frequency Cepstral Coefficients (MFCC)

Estas características se reflejan en la figura 2.7 y observamos que en cada frecuencia se distinguen las amplitudes medias de los MFCC, es decir, existe separación entre los puntos que se encuentran a una misma frecuencia.

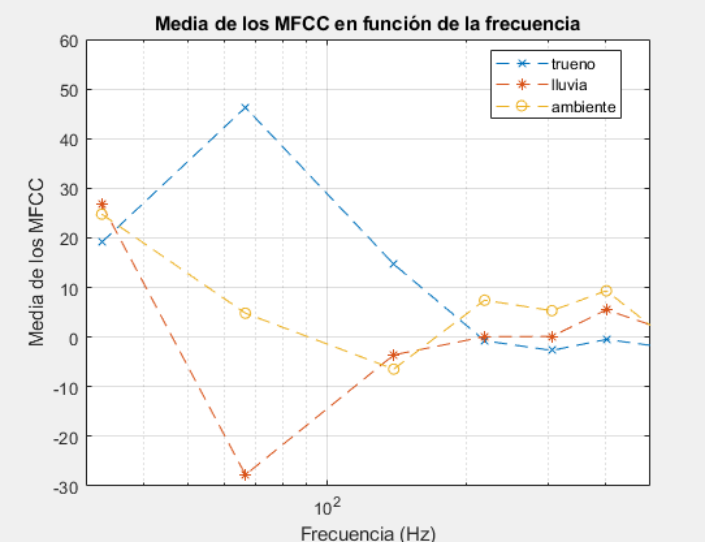


Figura 2. 7.- Amplitudes MFCC de las clases en función de la frecuencia.

• **Dominio Espectral**: Aquellas que se obtienen en el dominio de la frecuencia y que definen la forma del espectro.

o Asimetría Espectral (Spectral Skewness)
o Centroide Espectral (Spectral Centroid)

Estas features se pueden observar en la figura 2.8, la feature 41 se corresponde con el skewness y la 42 con el centroid.

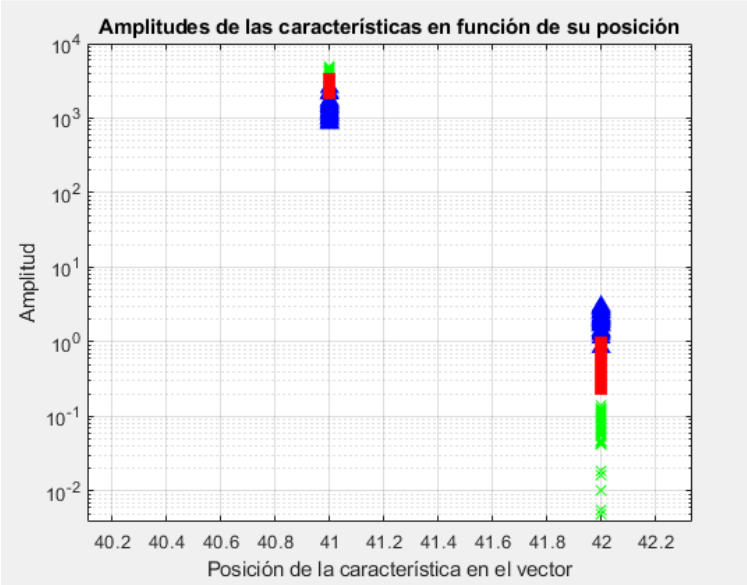


Figura 2. 8.- Amplitudes de las características Skewness y Centroid.

También observamos que se distinguen estas dos características en las tres clases, pues se perciben las fronteras de separación entre los símbolos de tres colores correspondientes a las tres clases.
El azul corresponde a la clase truenos, el verde a la clase lluvia y el rojo a la clase sonido ambiente.

· Mel-Frequency Cepstral Coefficients (MFCC)

Los MFCC [25,26] son features obtenidas en el dominio cepstral, el cual se obtiene como la Transformada Inversa de Fourier del logaritmo natural de la magnitud del espectro (Spectrum), dando lugar al dominio Cepstrum. Estas features han demostrado ser muy útiles en el ámbito del reconocimiento de voz (ASR – Automatic Speaker Recognition) aunque como hemos visto en la figura 2.7, a nosotros también nos funciona para los eventos sonoros atmosféricos ya que existe distinción entre las tres clases.
El algoritmo de extracción conlleva un conjunto de operaciones, tal y como se muestra en la figura 2.9.

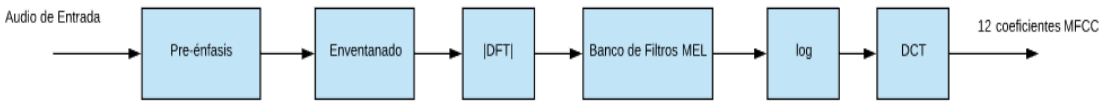


Figura 2. 9.- Proceso de extracción de los MFCC [25].

En realidad, los MFCC son features en forma de vector [25], es decir, se trata de más de un valor, pudiendo obtenerse tantos MFCC como coeficientes de la Transformada Discreta del Coseno (DCT), además de sus derivadas.
En primer lugar, se realiza un preénfasis (pre-emphasis) de la señal, lo que equivale a un filtrado tipo FIR (Finite Impulse Response) para realzar la alta frecuencia, el cual se muestra en la ecuación 2.34. Este realce proporciona una mejor relación señal a ruido (SNR – Signal to Noise Ratio) [26], además de introducir aproximadamente +6 dB/octava para compensar la atenuación de alta frecuencia.

$$H(z) = 1 - \alpha z^{-1} \quad 0{,}9 \leq \alpha \leq 1 \qquad \textit{Ec.2.34}$$

Posteriormente, se divide la señal en bloques o frames (división en frames) de menor duración con cierto solapamiento entre sí. En la figura 2.10 se puede observar este proceso, dividiendo la señal en bloques de tamaño L (frame length), con cierto solapamiento entre sí (overlapping).
Todo esto, para hacer a continuación, un enventanado, aplicando una ventana (Hamming, Hanning…) a cada frame.
Cada frame o bloque se multiplica por una ventana muestra a muestra (en este caso Hamming, ya que introduce menos distorsión que otras ventanas), de tal forma que las muestras de los extremos de cada bloque tengan menor peso que las muestras centrales, ya que, de lo contrario, se producirían efectos indeseados en alta frecuencia. Además, puesto que existe solapamiento, no se tiene pérdida de información por el efecto de atenuación de los extremos de cada bloque.

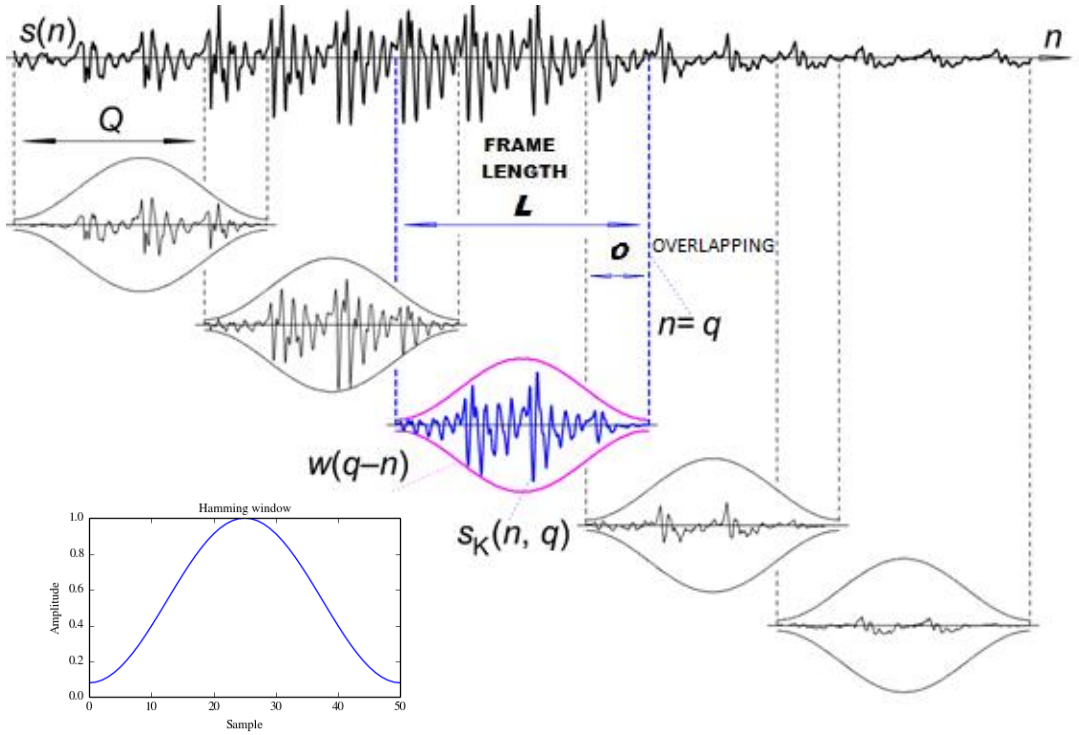


Figura 2. 10.- Enventanado [40].

De cada frame se obtiene la Transformada Discreta de Fourier (DFT – Discrete Fourier Transform), la cual se obtiene de la ecuación 2.35 obteniendo así la información de cada bloque en el dominio espectral. En la implementación práctica, este proceso es realizado mediante al algoritmo FFT (Fast Fourier Transform) o Transformada Rápida de Fourier**.**

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \quad 0 \leq k \leq N-1 \qquad \textit{Ec.2.35}$$

Donde k es un punto determinado de la DFT de los N posibles, siendo N mayor o igual a L (longitud de cada bloque de muestras).
Posteriormente, se aplica un banco de filtros Mel sobre cada frame para reducir el espectro en potencia (al cuadrado) a un conjunto de bandas espectrales. Este filtrado está basado en la percepción humana, teniendo así mayor resolución en baja frecuencia y menor en alta frecuencia. A diferencia de la escala lineal de frecuencias utilizada en el cómputo de la FFT, la escala Mel es una escala perceptual de tonalidades equidistantes, es decir, es proporcional al logaritmo de las frecuencias lineales, asemejándose así a la percepción humana [26]. En la ecuación 2.36 se presenta la ecuación que permite pasar de frecuencias a frecuencias Mel.

$$Mel(f) = 2595 \cdot log_{10}(1 + \frac{f}{700}) \qquad \textit{Ec.2.36}$$

El banco de filtros Mel es por tanto un conjunto de filtros triangulares equiespaciados entre sí en la escala Mel, da tal forma que, al volver a la escala lineal de frecuencias, las frecuencias centrales de dichos filtros se distribuyen de forma logarítmica en el espectro, dejando de ser filtros equiespaciados. Para devolver estas frecuencias Mel a frecuencias lineales, basta con aplicar la función inversa de la ecuación 2.36, la cual se muestra en la ecuación 2.37.

$$Mel(f) = 700 \cdot (10^{\frac{Mel(f)}{2595}} - 1) \qquad \textit{Ec.2.37}$$

En la figura 2.11 se puede observar cómo quedarían estos filtros distribuidos en el espectro. Se trata de un ejemplo en cual se ha utilizado una frecuencia de muestreo de 16000 Hz, con un tamaño de FFT de 1024 puntos y 34 filtros Mel.

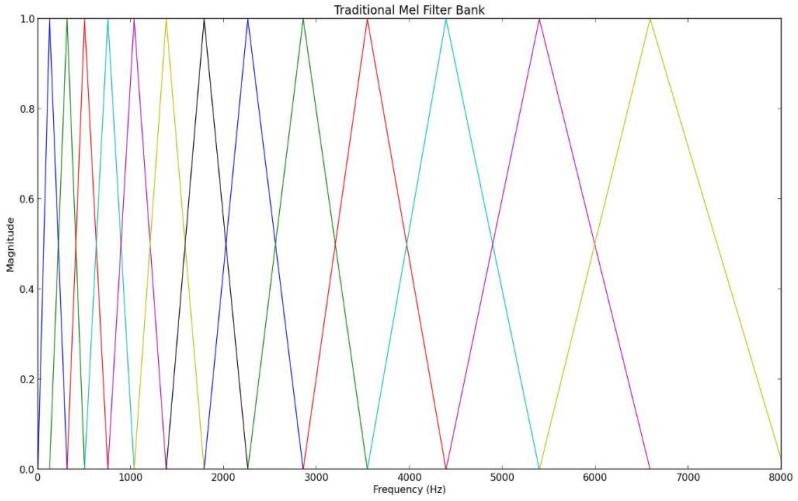


Figura 2. 11.- Banco de filtros MEL tradicional [45].

Este conjunto de filtros viene dado según la ecuación 2.38 [26], donde k es un punto ó bin de la FFT, m es el número del filtro, siendo f(m) el bin correspondiente a la frecuencia central del filtro. Por lo tanto, para cada filtro, se tiene un vector de ponderaciones de cada punto de la FFT, ponderando con gran peso aquellos puntos que se encuentren

alrededor de la frecuencia central, y con valor nulo para el resto del espectro. En la figura 2.12 se muestra el resultado de cada filtro para cada punto de la FFT.

$$Hm(k) = \begin{cases} 0, & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)}, & f(m-1) \leq k \leq f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)}, & f(m) \leq k \leq f(m+1) \\ 0, & k > f(m+1) \end{cases} \qquad \textit{Ec.2.38}$$

Nótese que los filtros se encuentran perfectamente solapados, da tal manera que el peso total de cada bin distribuido en dos filtros es igual a la unidad, satisfaciendo la ecuación 2.39 (no necesariamente para el último filtro).

$$\sum_{m=0}^{M-1} H_m(k) = 1 \qquad \textit{Ec.2.39}$$

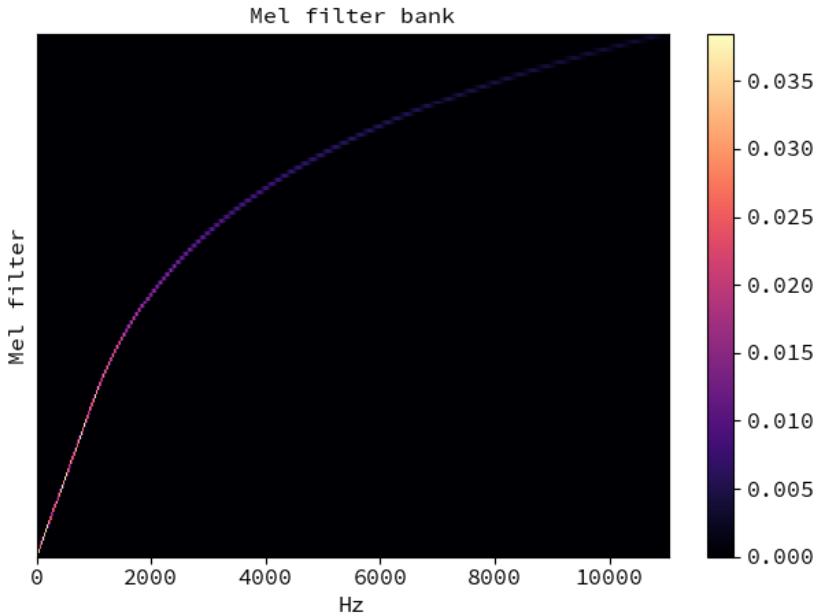


Figura 2. 12.- Banco de filtros a la frecuencia Mel [32].

Este filtrado se aplica a cada frame, obteniendo M valores del espectro, correspondientes a cada una de las bandas espectrales definidas por los filtros triangulares. Una vez se tiene la magnitud de cada una de las bandas, se aplica el logaritmo natural, ya que, la percepción humana de la sonoridad también es procesada de una forma logarítmica. En la ecuación 2.40 se muestra la ecuación que describe este proceso [22].

$$S(m) = \log\left[\sum_{k=0}^{N-1} |X(k)^2| \cdot H_m(k)\right] \quad 0 \leq m \leq M \qquad \textit{Ec.2.40}$$

Finalmente, se aplica la Transformada Discreta del Coseno o DCT (Discrete Cosine Transform) para obtener el cepstrum. El término cepstrum proviene de las cuatro primeras letras de spectrum al revés, ya que, se trata del espectro del logaritmo del espectro de la señal, es decir, trata el logaritmo del espectro de la señal como una nueva señal en sí misma. En realidad, el dominio cepstral se obtiene mediante la Transformada Discreta de Fourier Inversa (IDFT – Inverse Discrete Fourier Transform). Sin embargo, puesto que en el proceso de cálculo de los MFCC no se contempla la parte imaginaría, es habitual realizar este proceso mediante la DCT, la cual se aplica directamente sobre la magnitud logarítmica, según la ecuación 2.41:

$$C(k) = \sum_{m=0}^{M-1} S(m) \cdot \cos(\pi k(m + 1/2)/M) \quad 0 \leq k \leq K \qquad \textit{Ec.2.41}$$

Donde K es el número máximo de coeficientes de la DCT, determinando k el número del MFCC a extraer. M determina el número máximo de puntos sobre los que se aplicará la DCT, que coincide con el número de filtros Mel empleados, siendo m el indicador de cada filtro.
La primera componente de la DCT (k=0) equivale a la energía de la señal, ya que, la ponderación para cada filtro es la unidad. La segunda componente es una relación entre baja y alta frecuencia. A medida que aumenta el orden de las componentes de la DCT aumenta el número de ciclos del coseno. Puesto que se trata de un dominio logarítmico, las sumas y restas equivalen a multiplicaciones y divisiones, respectivamente, por lo que conceptualmente la DCT realiza una serie de ratios entre bandas espectrales. Esto hace que la información más relevante se concentre en los primeros coeficientes cepstrales (MFCC), decorrelando así la información y eliminando redundancia.

· Centroide Espectral (Spectral Centroid)
El centroide espectral es un tipo de feature estadístico que aporta información acerca de la forma del espectro, concretamente define el centro de gravedad del espectro, es decir, considera el espectro como una distribución de probabilidad, la cual viene determinada por la amplitud del espectro para cada frecuencia, obteniendo así la frecuencia media ponderada en amplitud [21], tal y como está representado en la figura 2.13.

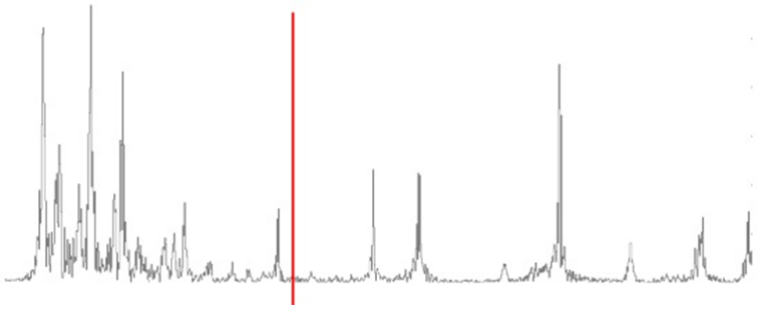


Figura 2. 13.- Centroide espectral [30].

El centroide se calcula según la ecuación 2.42, donde *x* es la frecuencia y *p(x)* la amplitud (o probabilidad) del espectro normalizado, en la frecuencia *x*.

$$\begin{cases} \mu = \int x \cdot p(x)\delta x \\ p(x) = \dfrac{amplitud(x)}{\sum_x amplitud(x)} \end{cases} \qquad \textit{Ec.2.42}$$

Existen otros tipos de centroide, como el centroide armónico, el cual puede ser muy útil a la hora de discriminar entre instrumentos musicales. Sin embargo, éste implica la extracción previa de la nota fundamental, así como sus armónicos.
El centroide está muy correlado con el pitch de la señal, ya que la tonalidad percibida tendrá mucha parte de la energía del espectro.

· Asimetría Espectral (Spectral Skewness)
El skewness o asimetría [21], aporta información de la asimetría energética del espectro respecto a su centroide (media). Esta asimetría se calcula según la ecuación 2.43.

$$\begin{cases} m_3 = \int (x-\mu)^3 \cdot p(x)\delta x \\ \gamma_1 = \dfrac{m_3}{\sigma^3}\ (skewness) \end{cases} \qquad \textit{Ec.2.43}$$

En función del valor del skewness, el espectro será simétrico o asimétrico con mayor energía hacia la izquierda (menor frecuencia) o hacia la derecha (mayor frecuencia) del centroide (ecuación 2.44):

$$\begin{cases} \gamma_1 = 0 \rightarrow \text{ Distribución simétrica} \\ \gamma_1 < 0 \rightarrow \text{Distribución asimétrica hacia la derecha} \\ \gamma_1 > 0 \rightarrow \text{Distribución asimétrica hacia la izquierda} \end{cases} \qquad \textit{Ec.2.44}$$

2.3.3 Random Forest

Random Forest es un algoritmo de clasificación basado en árboles de decisión. Con este algoritmo introducimos un nuevo concepto, el ensamblado. Se produce al utilizar, en este caso, más de un árbol de decisión para una misma clasificación. Sin embargo, para entender este algoritmo es necesario analizar previamente un árbol de decisión [27].
El árbol de decisión es un tipo de algoritmo muy utilizado en Inteligencia Artificial. Su estructura está basada en nodos (nodes), ramas (branches) y hojas (leaves). En la figura 2.14 se muestra la estructura del algoritmo. El primer nodo es denominado nodo raíz (root node), correspondiéndose cada nodo a una única feature. Cada rama representa un rango de valores de dicha feature para dividir (split) los datos, indicando el camino entre dos nodos. Los nodos finales se denominan hojas, que son nodos a partir de los cuales no se continúa subdividiendo los datos, obteniendo así un histograma de los

datos existentes en cada hoja, el cual será utilizado como resultado para una predicción futura, siendo la clase ganadora aquella con mayor presencia en la hoja.

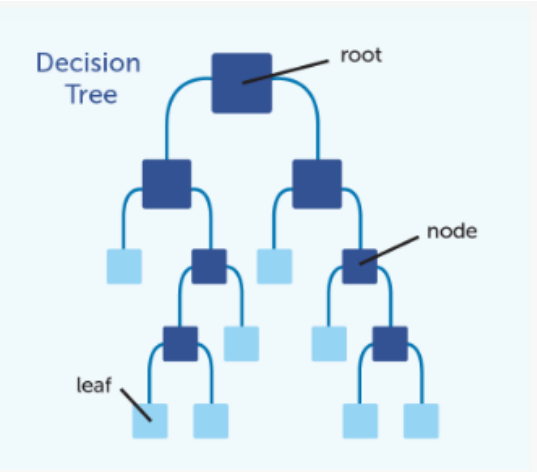


Figura 2. 14.- Árbol de decisión [28].

En la figura 2.15 se muestran los procesos de entrenamiento y test del clasificador. En la primera fase, se determinan los umbrales de cada nodo y se obtienen los histogramas de probabilidad de cada hoja o lo que es lo mismo, se 'dibuja' el árbol. En la fase de test, se realiza el camino raíz-hoja obteniendo así el resultado de la clasificación.

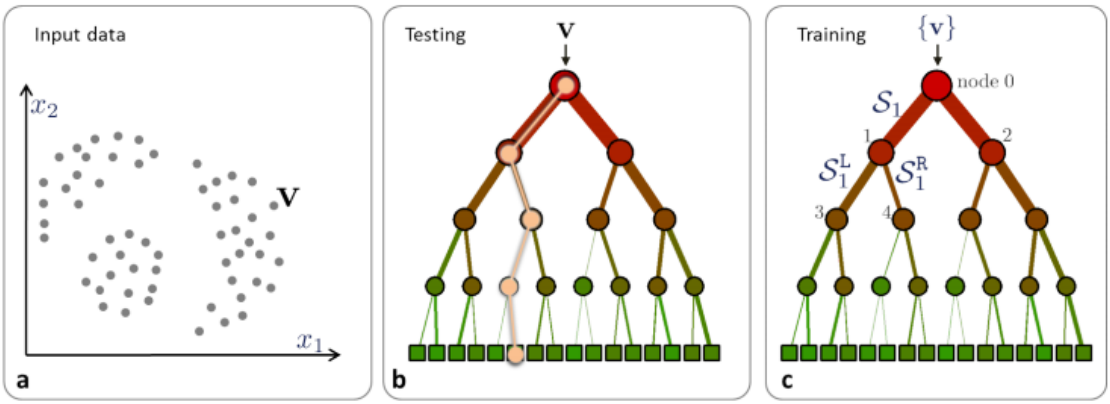


Figura 2. 15.- Ejemplo de entrenamiento y test con árboles de decisión [29].

En la fase de entrenamiento, tal y como se muestra en la figura 2.16, se tiene el training dataset (ver figura 2.16.a) a la entrada del nodo raíz, teniendo todo el conjunto de datos de N dimensiones (features) con los que será entrenado el árbol (ver figura 2.16.b), donde cada punto en el espacio de características es un vector de valores para cada feature: vn = (x1, x2, …, xn). Puesto que cada nodo se corresponde con una feature, y cada feature con una dimensión de las N iniciales, el proceso consiste en ir subdividiendo los datos en cada dimensión, de tal manera que cada nodo presente un histograma de clases de los datos que han tomado dicho camino.

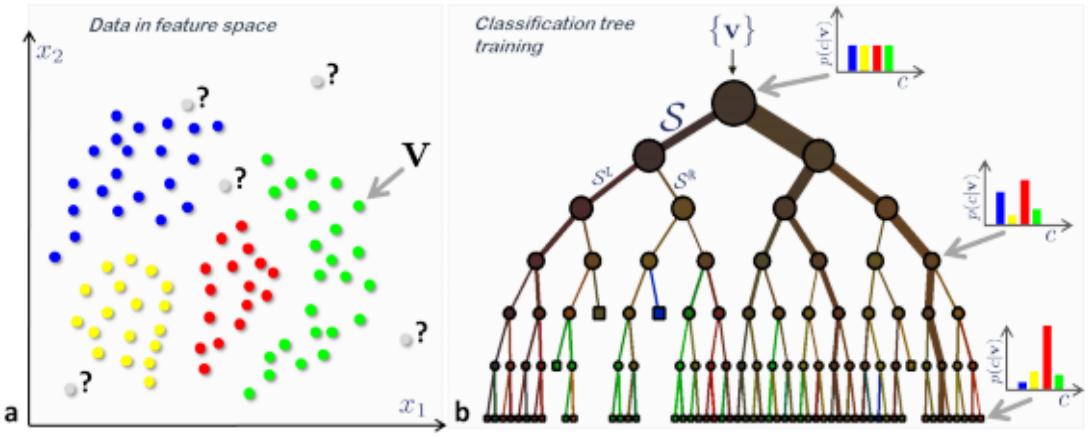


Figura 2. 16.- a) Training Dataset b) Fase de entrenamiento del árbol de decisión [29].

En primer lugar, se divide la información en el nodo raíz, separando los datos en función de un umbral específico que mejor separe la información de la feature asociada a dicho nodo (existen distintos criterios para determinar este umbral: Gini index, variance reduction, information gain… [31]). Una vez dividida la información, se separa en distintos nodos en función del umbral. En cada uno de estos nodos se produce otra subdivisión de los datos de la misma forma que en el nodo anterior. Finalmente, aquellos nodos que no pueden subdividir más la información quedan como hojas de una única clase, y, aquellos nodos que superen la profundidad del árbol o tree depth (parámetro configurable, representa el máximo número de subdivisiones), quedarán como hojas representando un histograma con el número de datos que contienen de cada clase.

En la fase de test, mostrada en la figura 2.17, el clasificador utiliza todos los umbrales determinados en la fase de entrenamiento, de tal manera que el vector de test en cuestión seguirá un camino desde el nodo raíz hasta una hoja en función de los valores de las features. La hoja resultante representará un histograma de probabilidad para cada una de las clases, es decir, la probabilidad que tiene el ejemplo o vector v de pertenecer a cada una de las clases, tal y como muestra la ecuación 2.45 [29], siendo la clase ganadora aquella con mayor probabilidad (es decir, mayor número de ejemplos de una clase durante la fase de entrenamiento en dicha hoja).

$$p_t(\mathrm{c}|\mathrm{v}) \qquad \textit{Ec.2.45}$$

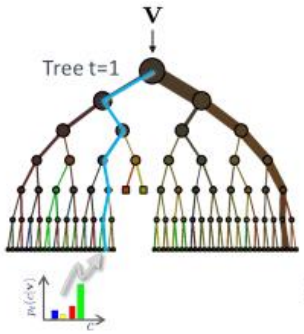


Figura 2. 17.- Fase de test de un árbol de decisión [29].

El algoritmo Random Forest es un método de ensamblado (ensemble method) basado en árboles de decisión, es decir, en lugar de entrenar un único árbol de decisión, se

entrenan un gran número de estos, de tal manera que entre todos ellos se obtenga una solución más precisa. A cada uno de estos árboles de decisión dentro del algoritmo de ensamblado se le denomina weak learner, y cada uno de ellos es entrenado con un subconjunto de features de las N posibles. La selección de este subconjunto de features entre las N que describen cada ejemplo se realiza de forma aleatoria, entrenando cada árbol con el mismo número de features, pero siendo estas diferentes. A este proceso se le denomina Bagging o Bootstrap Aggregation.
El proceso de entrenamiento se realiza para T árboles (parámetro configurable), y, de la misma forma, el test se realiza sobre T árboles, aportando cada uno de ellos un resultado distinto, tal y como se muestra en la figura 2.18 [29].

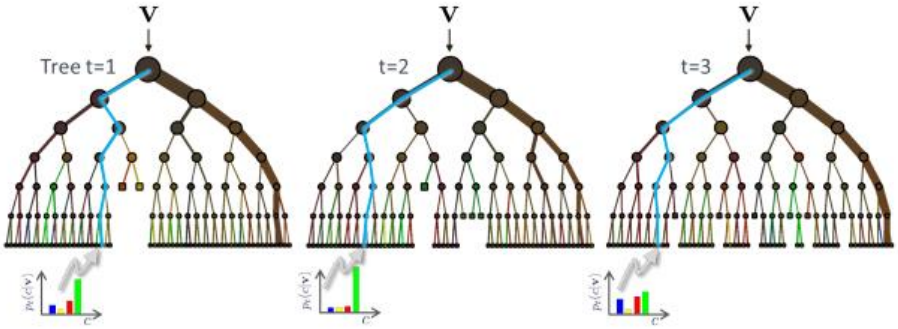


Figura 2. 18.- Random Forest: fase de test [29].

El resultado de la clasificación vendrá dado por el resultado conjunto de todos los árboles, el cual puede realizarse mediante un promediado, o bien mediante multiplicación de los resultados, según se muestra en la ecuación 2.46 [29].

$$\begin{cases} \mathrm{p(c|v)} = \dfrac{1}{T}\displaystyle\sum_{i=1}^{n} p_t(\mathrm{c|v}) \\ \mathrm{pt(c|v)} = \dfrac{1}{T}\displaystyle\prod_{t=1}^{T} p_t(\mathrm{c|v}) \end{cases} \qquad \textit{Ec.2.46}$$

Este proceso de ensamblado tiene la ventaja de seleccionar los resultados con mayor confianza, proporcionando así robustez frente a árboles "ruidosos" [29].

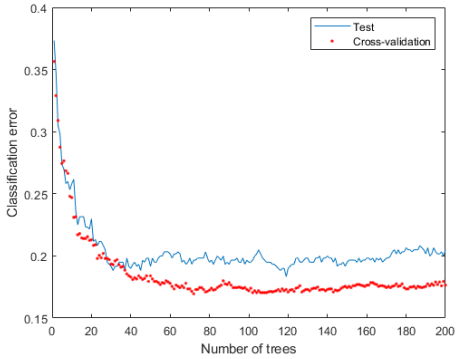


Figura 2. 19.- El error de la clasificación disminuye con el número de árboles [29].

Como se muestra en el ejemplo de la figura 2.19, la tasa de error disminuye de forma exponencial a medida que aumenta el número de árboles, hasta obtener un valor constante aproximadamente.

Sin embargo, no ocurre lo mismo con la profundidad de árbol, ya que, a medida que la información se subdivide, el modelo se ajusta cada vez más a los datos, lo que se denomina overfitting, produciendo que el clasificador no sea capaz de generalizar una nueva entrada. Si en la etapa de entrenamiento se subdivide la información hasta que todas las hojas sean puras, es decir, que únicamente contengan una clase, el clasificador estará sobreentrenado. Para solucionar esto, se realiza una poda o pruning de las hojas del árbol, limitando la profundidad del árbol, obteniendo así mayor grado de generalización.
Una particularidad interesante de este algoritmo es que permite clasificar un ejemplo para cada uno de los árboles por separado, es decir, en paralelo o de forma distribuida, uniendo posteriormente los resultados, lo cual puede resultar muy útil para aplicaciones de tiempo real.

2.3.4 K-Nearest Neighbors

El algoritmo k-NN es un tipo de algoritmo estadístico basado en los ejemplos de entrenamiento (instance-based learning) [22]. Al contrario que otros algoritmos, éste requiere menos tiempo de cómputo durante la fase de entrenamiento y por muy atractivo que resulte, este algoritmo presenta también varios inconvenientes. El primero es un aumento significativo de la capacidad de almacenamiento requerido para obtener un buen rendimiento del sistema, lo cual es un inconveniente salvable, pero dificulta la implementación final. Y también se incrementa, consecuentemente, el tiempo necesario para realizar la detección, puesto que la decisión se toma entre todas las muestras de las que se dispone. Sin embargo, la mayor debilidad que presenta es que puede ser engañado por la información de características que no aportan información real para la detección, eclipsando a las que sí lo hacen.
Está basado en el principio de cercanía de los ejemplos con características similares, siendo el algoritmo más básico el vecino más cercano (nearest neighbor). En este caso, el algoritmo se basa en los k vecinos más cercanos para predecir la clase de la muestra en cuestión.
El algoritmo consiste en determinar un espacio de características n-dimensional (n features) en el cual se encuentran todas las muestras de entrenamiento, todas ellas etiquetadas con su clase correspondiente. Durante el proceso de clasificación de una nueva muestra, ésta se proyecta en dicho espacio n-dimensional y se obtiene la distancia relativa entre la muestra y= (y_1, …, y_n) y cada uno de los ejemplos o muestras de entrenamiento: x= (x_1, …, x_n). Esta distancia se puede medir con diferentes métricas, siendo la más habitual la distancia euclídea, la cual se muestra en la ecuación 2.47. Una vez se identifican los k ejemplos con menor distancia relativa a la nueva muestra, se selecciona la clase con mayor probabilidad, es decir, aquella con mayor número de ejemplos entre los k vecinos [22].

$$\text{Euclidean:} \qquad D(x,y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \qquad \textit{Ec.2.47}$$

Sin embargo, existen otros métodos para medir dicha distancia que tratan de minimizar la distancia entre muestras pertenecientes a la misma clase, y maximizar aquellas con distinta clase. Algunas de estas métricas se muestran en las expresiones mostradas la ecuación 2.48 [22]. En la distancia de Minkowsky, *r* representa el orden del método (distancia euclídea generalizada).

$$\begin{cases} Minkowsky:\ D(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^r\right)^{1/r} \\ Manhattan:\ D(x,y) = \sum_{i=1}^{n} |x_i - y_i| \\ Chebyshev: D(x,y) = \max|x_i - y_i| \\ Camberra: D(x,y) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{|x_i + y_i|} \end{cases} \qquad \textit{Ec.2.48}$$

El valor de k influye en gran medida en el resultado de la clasificación. Sin embargo, no existe un método que permita elegir un k adecuado, salvo la realización de pruebas con distintos valores de k.

2.4 Algoritmos de Localización

La estimación de la dirección de procedencia (DOA), en este caso del sonido, consiste en identificar la posición relativa de la fuente emisora del sonido con respecto al array de micrófonos. Entre sus aplicaciones más frecuentes se encuentra la localización del hablante [33] o la separación de fuentes de sonido [36]. Con los algoritmos DOA, prácticamente la posición exacta no puede ser obtenida y en tal caso, la dirección de llegada para las ondas acústicas es sólo estimada. En condiciones ideales, la estimación DOA es un problema directo y determinista. Sin embargo, en condiciones reales, existen factores que pueden influir en la señal, como el ruido de fondo que causa distorsiones, y otros factores como señales reverberantes que interfieren con el objetivo, lo que hace que el problema sea más complejo y no determinista.

Es por ello que, para resolver estos problemas, se han implementado una serie de métodos. Los más utilizados para obtener la dirección de procedencia del sonido se basan en el algoritmo del retardo temporal (TDOA) [42], entre señales provenientes de una serie de micrófonos a los que le aplican técnicas de correlación cruzada [34]. O bien, otros métodos subespaciales como la clasificación de señales múltiples (MUSIC) [12]. Por último, existen otros métodos que desarrollaremos a continuación como el del

Periodograma que parte de la transformada discreta de Fourier (DFT), el de la Mínima Varianza cuyo desarrollo parte de un filtro FIR, además de el de la Máxima Verosimilitud.[20].

Estas técnicas se originan a partir de la psicoacústica, la cual justifica que la percepción de la directividad del sonido por el sistema binaural humano se basa en las siguientes dos diferencias interaurales [49]:

•Diferencia de nivel interaural: diferencia de intensidades de las formas de onda en el oído izquierdo y derecho.
• Diferencia de tiempo interaural: diferencia de tiempos de llegada de formas de onda en ambos oídos, que es equivalente a una diferencia de fase de estas formas de onda.

El método que adoptaremos para la localización de la dirección de procedencia es el del retardo temporal o TDOA. Cuyo código de partida lo hemos obtenido de la página web de Matlab [44] adaptándolo a nuestras necesidades, modificando el número de pares, la posición de los micrófonos, además del tamaño del buffer y la longitud de los frames de audio a analizar en tiempo real.
Este código se adjunta en el Anexo de este proyecto, a su vez en el apartado del Código de Matlab.

2.4.1 Método del Periodograma

Este estimador espectral denominado Periodograma [15] consiste en elevar al cuadrado la Transformada de Fourier Discreta (DFT) de las observaciones y multiplicarla por 1/N, tal y como se observa en la ecuación 2.49.

$$P(f) = \left| \frac{1}{N} \sum_{n=0}^{N-1} y(n) \cdot e^{-j \cdot 2 \cdot \pi \cdot f \cdot n} \right|^2 = \frac{1}{N} |Y(f)|^2 \qquad \textit{Ec.2.49}$$

Donde y(n) son los snapshots en cada instante de tiempo, f es la frecuencia, N el tamaño de la muestra y Y(f) es la DFT de y(n).
Este estimador espectral es sesgado e inconsistente, es decir, proporciona una respuesta inestable. En la literatura dedicada a la estimación estadística de espectros hay quién afirma: *Como una función del tamaño de la muestra, el periodograma es "asintóticamente insesgado".* Además, el periodograma es inconsistente debido a que su varianza $\mathrm{E}\{\mathrm{P}(\mathrm{f})\}^2$ no disminuye con el tamaño de la muestra. No obstante, lo anterior puede ser una buena base para la estimación DOA. Además, permite una aproximación bastante intuitiva.

La figura 2.20 muestra el organigrama de un algoritmo con dos alternativas una vez se dispone del snapshot, al cual denominaremos y(n). La elección de la vía dependerá del dominio en el que vayamos a trabajar (en el del tiempo o en el de la frecuencia), para así aproximarnos al estimador considerado y así obtener luego la matriz de covarianzas, Ryy. En sistemas prácticos con señales reales la única vía válida será la que utiliza la

Transformada de Fourier (TF) calculada mediante un algoritmo para el cálculo de la FFT (Fast Fourier Transform).

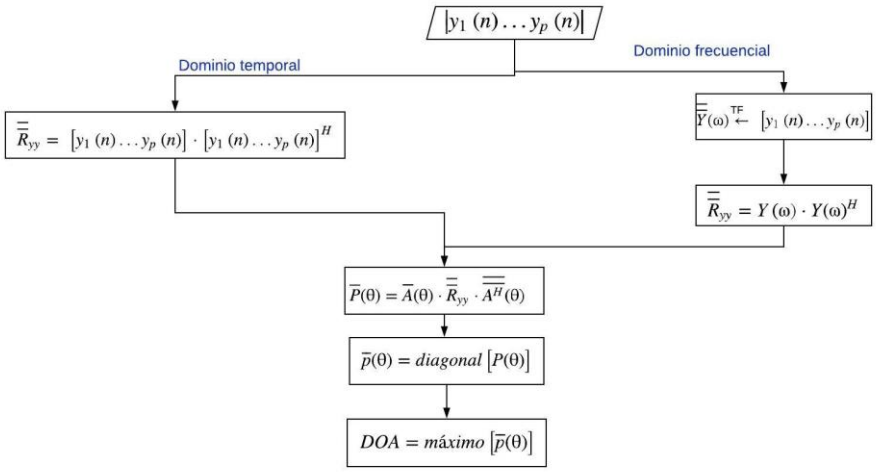


Figura 2. 20.- Organigrama del algoritmo del Periodograma [20].

Una vez hecha la estimación de la matriz de covarianzas Ryy (figura 2.20), procedemos al cálculo de la potencia para cada una de las direcciones deseadas, por ejemplo, podríamos realizar un barrido en pasos de 1° entre 90° y –90°. Una vez terminado este proceso dispondremos la matriz P(θ) y en su diagonal se encuentran los productos de nuestro interés p(θ). Por último, el valor o valores máximos de p(θ) nos señalará la dirección o direcciones de procedencia de la fuente o fuentes [20].

2.4.2 Método de la Mínima Varianza

Este estimador fue propuesto por Capon en 1969 [13] para aplicarlo en la estimación del número de onda en detectores sísmicos. Más tarde, Lacoss en 1971 [14], demostró que el método proporciona una estima insesgada de mínima varianza (MV) de las componentes espectrales de la señal.
El desarrollo de Lacoss parte de un filtro FIR de coeficientes a_k a determinar, donde $0 \leq k \leq p$.
En este caso no se restringe a_0 a la unidad. La respuesta del filtro para una entrada s(n), donde $0 \leq n \leq N-1$ sería la ecuación 2.50.

$$y(n) = \sum_{k=0}^{p} a_k s(n-k) \equiv s(n)^T \cdot \bar{a} \qquad \text{Ec.2.50}$$

Donde $s(n)$ es la entrada, el superíndice $(\cdot)^T$ indica traspuesta, T_s es el periodo de muestreo en segundos y $\bar{a}$ es el vector de coeficientes del filtro.

Luego, serán necesarias algunos cálculos intermedios para obtener el estimador adaptándolo a nuestras necesidades. Con este fin la obtenemos la ecuación 2.51.

$$P_{SS}^{MV}(\theta) = \frac{1}{\bar{A}^T(\theta) \cdot \bar{\bar{R}}_{ss}^{-1} \cdot \bar{A}^*(\theta)} \qquad \text{Ec.2.51}$$

Donde $\bar{\bar{R}}_{ss}^{-1}$es la matriz inversa de autocorrelación de entrada y A(θ) representa la matriz de fases de las fuentes al incidir sobre los sensores con i=1,….,p y k=1,…,q y los superíndices * y T usados en la matriz de fases indican conjugada y traspuesta respectivamente**.**

Entonces, se puede decir que este método es una implementación de banco de filtros, diferenciándose con otros estimadores que usan la misma técnica en que aquí se optimizan los coeficientes del filtro. Pero nuestro interés no es el cálculo de la potencia o densidad de potencia espectral, sino la potencia o densidad de potencia espacial o angular.

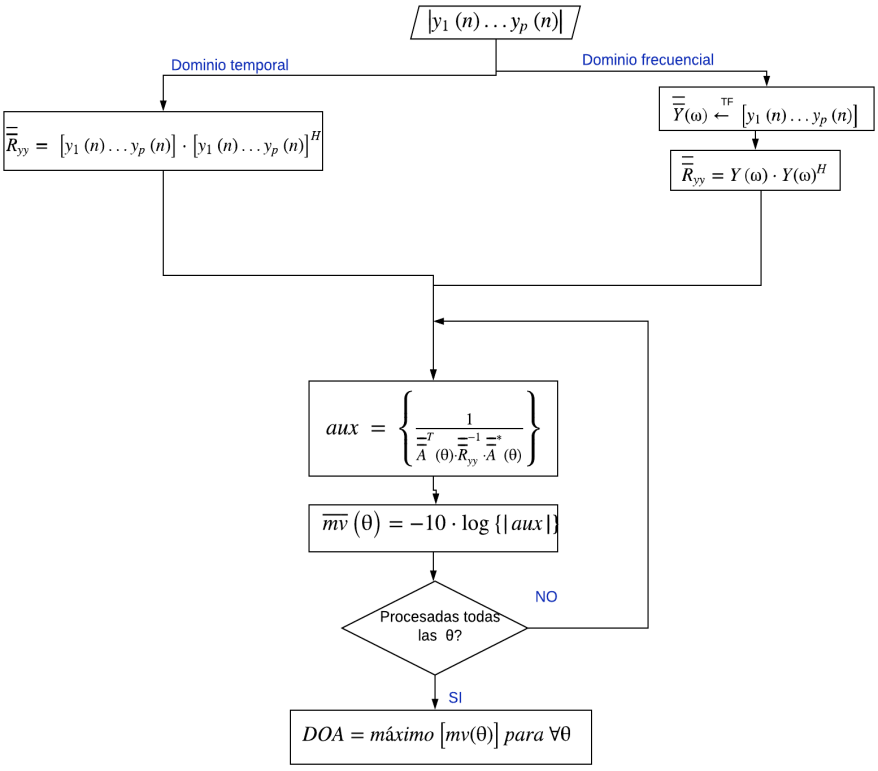


Figura 2. 21.- Organigrama del algoritmo de la Mínima Varianza [20].

La figura 2.21 muestra el organigrama de un algoritmo con dos alternativas una vez se dispone del snapshot, al cual denominaremos y(n). La elección de la vía dependerá del dominio en el que vayamos a trabajar (en el del tiempo o en el de la frencuencia), para así aproximarnos al estimador considerado y luego obtener la matriz de covarianzas, Ryy. En sistemas prácticos con señales reales la única vía válida será la que utiliza la Transformada de Fourier (TF) calculada mediante un algoritmo para el cálculo de la FFT (Fast Fourier Transform).

Seguidamente, se calcula la estimación de la matriz de covarianzas (Ryy) a partir de una instantánea tomada por la agrupación, también aquí podrá hacerse en el dominio

temporal o en el frecuencial. A continuación, nos introducimos en un bucle de longitud igual al conjunto de las {θ}. Para cada θ aplicamos el estimador memorizando la "potencia" recogida en esa dirección en la variable temporal aux. Luego, se calcula el valor en decibelios de aux, se cambia de signo y se guarda en el vector mv. El cambio de signo permite transformar un problema de búsqueda de mínimo en otro de búsqueda del máximo. Al terminar y salir de este bucle buscamos de entre todos los valores de mv el máximo o máximos que constituirán las estimaciones de la localización o localizaciones de las fuentes [20].

2.4.3 Método MUSIC

El método MUSIC (MUltiple SIgnal Characterization) es el algoritmo más popular para estimar la dirección de procedencia de una o varias fuentes sonoras, está basado en la descomposición en valores singulares (SVD) [12] de la matriz de covarianzas, Ryy de y(t). A los métodos que utilizan esta descomposición se les denomina métodos Basados en Subespacios. Originalmente esta descomposición se utilizó en métodos espectrales, aunque de manera implícita. El punto de partida es la matriz de covarianza de la salida, que puede expresarse como la ecuación 2.52.

$$R_{yy} \equiv E\{y(t) \cdot y^{+}(t)\} = A(\theta) \cdot E\{s(t) \cdot s^{+}(t)\} \cdot A^{+}(\theta) + \sigma^{2} \cdot \bar{\bar{I}} \qquad \textit{Ec.2.52}$$

Donde s(t) es la señal, A(θ) es la matriz de direcciones, g(θi) contenido en A(θ) es el vector de las respuestas de cada sensor en la dirección θi, τm(θi) contenido en g(θi) el retardo sobre cada sensor tomando como referencia al primer sensor, σ^2 es la varianza del ruido presente en la agrupación e I es la matriz identidad.

Previamente haremos un inciso sobre el problema que nos ocupa, esto es determinar las direcciones de L ondas planas incidiendo en una agrupación lineal y uniforme de M sensores. Además, hemos supuesto que el ruido es espacial y temporalmente incorrelado, que las ondas planas no son totalmente coherentes entre sí y que el número de muestras es mucho mayor que el número de sensores.

Después de varios cálculos intermedios, llegamos a la ecuación 2.53:

$$\bar{\bar{F}}(\theta) = \bar{\bar{A}}^{H}(\theta) \cdot \bar{\bar{E}} \cdot \bar{\bar{E}}^{H} \cdot \bar{\bar{A}}(\theta) \qquad \textit{Ec.2.53}$$

Donde $\bar{\bar{E}}$ es la matriz de autovectores y el superíndice H implica que la matriz es conjugada y traspuesta.

La estimación MUSIC de los θ_i se obtiene seleccionando los *q* valores de θ para los cuáles ***F(***θ*)* es mínima.

El algoritmo que hemos implementado se corresponde con el organigrama de la figura 2.22, igual que en los casos anteriores, observamos que existen dos alternativas una vez se dispone del snapshot (y(n)) según el dominio en el que vayamos a trabajar (en el del

tiempo o en el de la frencuencia), elegiremos una vía u otra para aproximarnos al estimador considerado y luego obtener la matriz de covarianzas, Ryy.

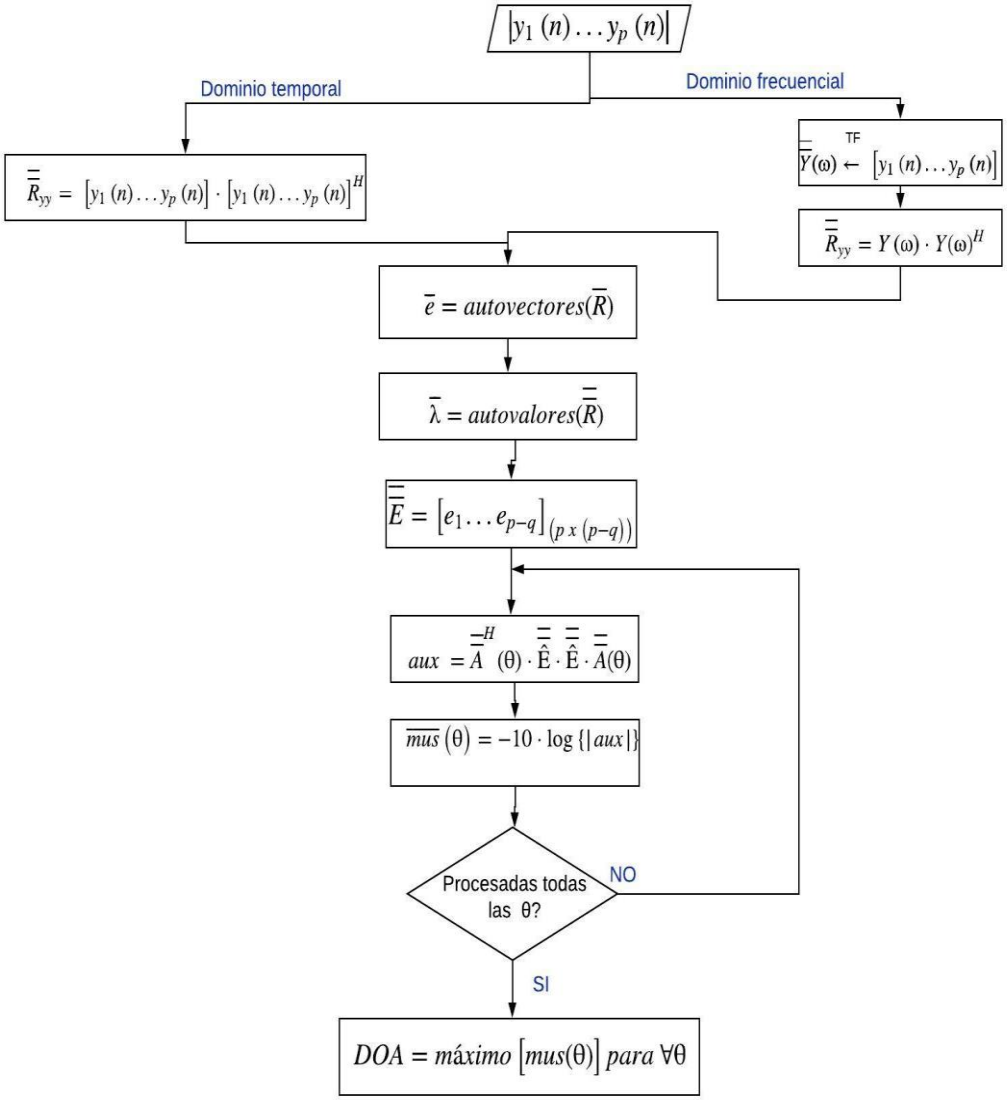


Figura 2. 22.- Organigrama del algoritmo MUSIC [20].

En sistemas prácticos con señales reales la única vía válida (figura 2.22) será la que utiliza la Transformada de Fourier (TF) calculada mediante un algoritmo para el cálculo de la FFT (Fast Fourier Transform).
Seguidamente, calculamos los autovectores y autovalores de la matriz de covarianzas R. De entre los autovalores buscamos el menor y a continuación formamos la matriz E que contiene los autovectores del espacio de ruido. Ahora procede aplicar el estimador sobre los que supuestamente son los autovectores del espacio de ruido. El resultado de la estimación para la dirección θ se almacenará en la variable temporal aux. El valor en decibelios de la energía estimada en cada dirección θ, aux, se memoriza en el vector mus. Cuando hayamos terminado la realización del proceso descrito para todas las direcciones θ de interés, pasaremos a buscar el máximo o máximos de mus. El valor o valores de θ donde se encuentre el máximo o máximos constituirán la DOA de cada fuente [20].

2.4.4 Método de la Máxima Verosimilitud

El estimador de máxima verosimilitud (Maximum Likelihood) posee algunas propiedades interesantes: Consistencia, eficiencia asintótica y, además, es asintóticamente gausiano. Por otro lado, estas propiedades nos adelantan la necesidad de gran cantidad de datos, para alcanzar estimaciones con cierta credibilidad. Por lo tanto, influirá en el tiempo de cálculo de la dirección de procedencia.

Este método parte de la selección de una función de coste, que será minimizada o maximizada según el caso. Supongamos que intentamos estimar un parámetro de naturaleza aleatoria, Θ, como ocurre en el caso de la búsqueda de DOA a partir de un conjunto de datos Y=y. En estas condiciones a la función densidad de probabilidad condicional p(Θ|Y=y) se le conoce como la densidad a posteriori, ya que la estimación está condicionada a una observación previa [11]. La obtención de p(Θ|Y=y) se efectúa por el teorema de Bayes (ecuación 2.54).

$$p(\Theta|Y) = P(Y|\Theta) \cdot \frac{p(\Theta)}{p(Y)} \qquad \textit{Ec.2.54}$$

Finalmente, obtenemos la expresión de este estimador que constituye la base para el algoritmo implementado (ecuación 2.55), cuyo diagrama de flujo se presenta en la figura 2.23.

$$ML(\theta) = tr\left\{\frac{A(\theta)^H \cdot R_{yy}^{-1} \cdot A(\theta)}{\left[R_{yy}^{-1} \cdot A(\theta)\right]^H \cdot \left[R_{yy}^{-1} \cdot A(\theta)\right]}\right\}_{pxp} \qquad \textit{Ec.2.55}$$

La figura 2.23 comienza con la adquisición de un snapshot de la agrupación como en los casos anteriores. Seguidamente y según el dominio en que vayamos a trabajar (en el del tiempo o en el de la frencuencia), tendremos dos formas de obtener la matriz de covarianzas, Ryy.
Los dos siguientes pasos constituyen el desarrollo del estimador dado y permiten una simplificación del cálculo. Este proceso se repetirá tantas veces como direcciones, θ, se quieran comprobar. Ahora que ya disponemos de todos los elementos del estimador asignamos a la variable *aux* el resultado, escalar, de aplicar la traza a la operación con matrices tal y como lo define la ecuación del estimador ML. En el paso siguiente calculamos el logaritmo base diez del escalar *aux* asignándolo al vector *ml* que tendrá tantos elementos como direcciones θ estemos probando. Además, en este paso cambiamos de signo el resultado del logaritmo y así el problema de búsqueda del mínimo se convierte en la búsqueda de un máximo.
Este último paso sólo se realizará cuando hayamos procesado todas las direcciones θ [20].

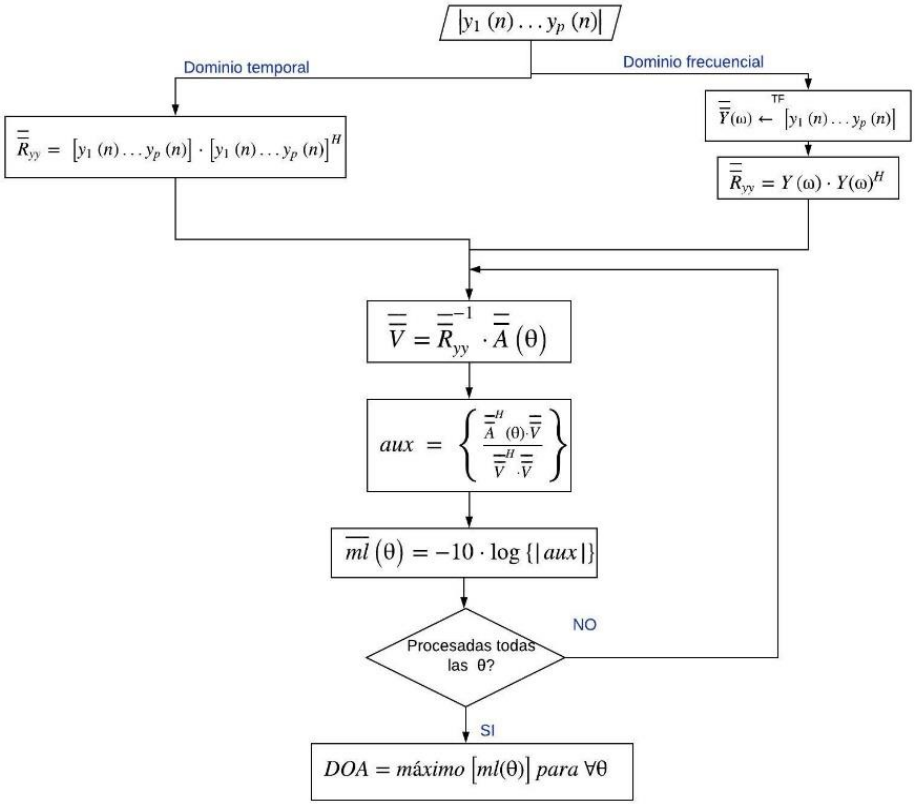


Figura 2. 23.- Organigrama del algoritmo de la Máxima Verosimilitud [20].

2.4.5 Método del Retardo Temporal. TDOA

El método del retardo temporal o la diferencia de tiempos de llegada es la segunda técnica de localización del sonido más popular, por detrás del MUSIC.
Este método no requiere saber el momento en el que se envió la señal desde el objetivo, solo el momento en que se recibió la señal y la velocidad a la que viaja el sonido. Una vez que la señal se recibe en dos puntos de referencia, la diferencia de tiempos de llegada se usa para calcular la diferencia de distancias entre el objetivo y los dos puntos de referencia [41].

Considerando varios micrófonos (M) y una fuente (S) en un mismo plano. Existe una ecuación (2.56) para cada uno de ellos.

$$x_i(t) = s(t - \tau_i) + n_i(t), \qquad \forall \mathrm{i}, 1 \leq \mathrm{i} \leq \mathrm{M} \qquad \textit{Ec.2.56}$$

Así como un retardo temporal (TDOA) entre cada par de micrófonos (ecuación 2.57).

$$\tau_{i,j} = \tau_i - \tau_j \qquad \textit{Ec.2.57}$$

Donde:

- x(t) es la señal recibida por el micrófono que se atenúa en él y retrasada por un retardo τ.
- s(t) es la señal de la fuente.

- n(t) es ruido ambiental.
- τ_i, τ_j es el tiempo de llegada desde la fuente hasta el micrófono.
- $\tau_{i,j}$ es la diferencia de tiempos de llegada.

Entonces, dado un par de micrófonos, calculamos su retardo temporal atendiendo a la ecuación 2.58:

$$\tau_i, \tau_j = \frac{\|\boldsymbol{S} - \boldsymbol{M}_i\|}{v} - \frac{\|\boldsymbol{S} - \boldsymbol{M}_j\|}{v} \qquad \textit{Ec.2.58}$$

Donde:

- $\|\cdot\|$ es la norma Euclídea
- v es la velocidad del sonido, 340 m/s.

Esta ecuación (2.58) representa una hipérbola, para cada par de receptores o micrófonos hay una única hipérbola como se muestra en la figura 2.24. En nuestro caso existen tres debido a que hay tres pares.

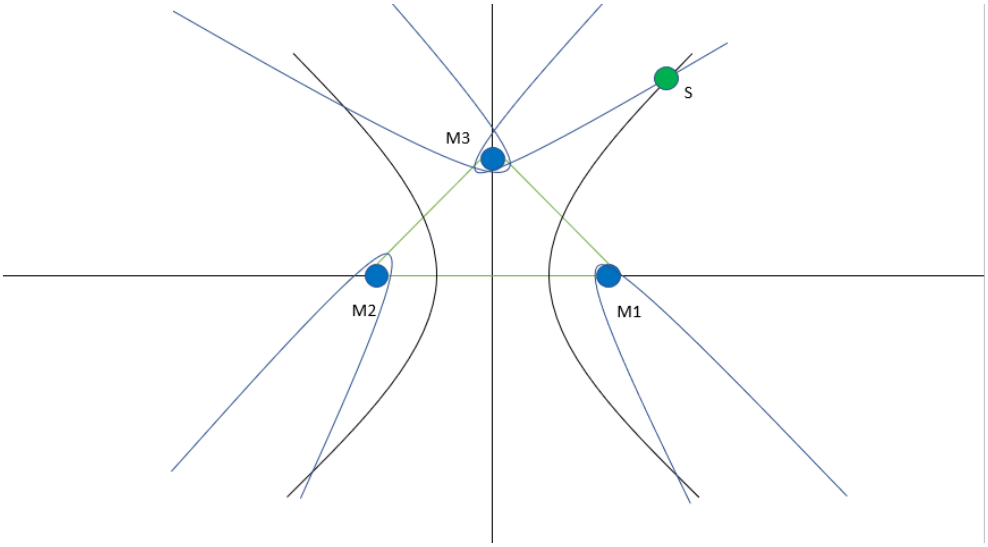


Figura 2. 24.- Hipérbolas correspondientes a cada par de micrófonos.

A cada hipérbola o par de micrófonos(i,j), se le aplica correlación cruzada en conjunto, que se corresponde con el máximo de la correlación de fase entre los micrófonos, como se observa en la ecuación 2.59 [42]:

$$\Delta T_{i,j} = \arg\max_{\tau}(R_{i,j}(\tau)) \qquad \textit{Ec.2.59}$$

Después, aplicamos un filtro FIR (Filtro de Respuesta Finita) cuyo valor estimado se obtiene de la interpolación de los coeficientes del filtro con el objetivo de obtener una mejor estimación del TDOA [43] o diferencia de tiempos de llegada. Esta operación se realiza para cada uno de los tres pares para obtener tres retardos.

El conjunto de los tres retardos obtenidos se convierte a ángulos en radianes, por lo tanto, tendremos tres ángulos de los cuales elegimos el mínimo en valor absoluto, por equivaler al mínimo retardo y con lo cual, de mayor cercanía a la fuente. A este ángulo

elegido se le sumará una cantidad en grados aplicando reglas heurísticas y así obtendremos una estimación de la dirección de procedencia del sonido.

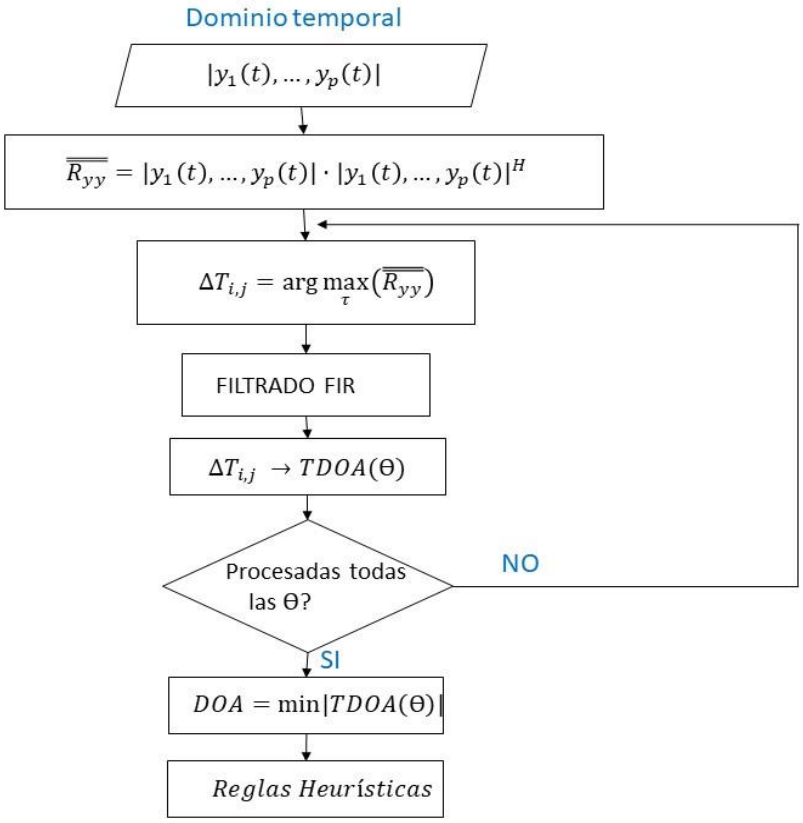


Figura 2. 25.- Organigrama del algoritmo del TDOA.

El organigrama del algoritmo TDOA se muestra en la figura 2.25, comienza con la adquisición de un snapshot de la agrupación como en los casos anteriores para obtener la matriz de covarianzas, pero esta vez trabajamos en el dominio del tiempo. Luego, los siguientes pasos se han comentado anteriormente.

[Esta página ha sido dejada intencionalmente en blanco]

CAPÍTULO 3: DESCRIPCIÓN Y MONTAJE DEL SISTEMA

[Esta página ha sido dejada intencionalmente en blanco]

3.1 Introducción

En este capítulo se describirá el sistema y su funcionamiento, para ello, se describirán tanto los elementos que lo componen como su interconexionado. Asimismo, se detallan los procedimientos necesarios para calibrar los micrófonos junto con su respuesta en frecuencia. Luego, se comentan las distintas configuraciones de micrófonos posibles antes de elegir la agrupación final de micrófonos, la cual detectará con mayor precisión las direcciones de llegada. Finalmente, se habla sobre el área de cobertura de cada par de micrófonos sobre la circunferencia.

3.2 Descripción del sistema

Nuestro sistema se compone de un ordenador personal desde donde lanzamos la aplicación que procesa el sonido recogido por el array de tres micrófonos. La agrupación de micrófonos (array) junto con la interfaz de audio Scarlett Focusrite 18i20 forman la otra parte de nuestro sistema.
Su principal funcionamiento es el de una aplicación redundante o complementaria para los sistemas existentes en meteorología.
En la mayoría de los sistemas de vigilancia de cualquier tipo, se suelen usar aplicaciones redundantes por si fallan los sistemas principales, en ese caso, este sistema sería el adecuado para entrar en funcionamiento, facilitando información meteorológica al usuario dentro de los límites y funcionalidades para las que está diseñado.
El sistema primero “escuchará” el medio en el que se quiere poner en funcionamiento para posteriormente establecer umbrales que distingan el sonido del silencio y así poder realizar las detecciones de los eventos sonoros, luego los clasificará, más concretamente los truenos y la lluvia y en el caso de los truenos, localizará la dirección de procedencia.
El sistema también se ha entrenado con sonido ambiente para que cuando no se detecte ni truenos ni lluvia, se clasifique como sonido ambiente (donde se combina el sonido del tráfico, del viento que mueve las hojas de los árboles y el canto de pájaros). O si, por el contrario, no se detecta ningún evento sonoro atmosférico, informar al usuario de que sólo detecta sonido ambiente.
Como hemos visto en el capítulo 2, cada tipo de sonido que reconoce nuestro sistema se denomina clase y decidimos restringirlo sólo a tres clases, de lo contrario sería más complejo, ya que surgirían nuevas dificultades como el aumento del tamaño de las bases de datos de entrenamiento, el aumento de características o features para distinguir entre clases o, el uso de algoritmos o técnicas de reconocimiento de sonidos más complejas.

3.2.1 Descripción de los elementos del sistema

El sistema consta de los siguientes elementos:

Array de 3 micrófonos: Un conjunto de tres micrófonos que se distribuyen formando un triángulo equilátero. De esta manera, se pueden recoger las máximas direcciones de

llegada posibles ya que es lo más parecido al modelo omnidireccional, como veremos al final de este capítulo.
Para formar el triángulo equilátero, hemos diseñado un soporte para los micrófonos utilizando dos láminas de aluminio, estas dos láminas se unen por su centro formando una cruz y ésta a su vez, se introduce en el tornillo del pie de micrófono, luego se atornillan con una tuerca de mariposa de manera que este soporte para los micrófonos en forma de cruz quede lo más fijo posible al pie. Las láminas, contienen unos raíles que nos servirán para variar la longitud de los lados del triángulo, sobre ellos, se atornillan los soportes de micrófono, también con sus tuercas de mariposa. Posteriormente, se colocan los tres micrófonos en sus respectivos soportes (figura 3.1).
El diseño del soporte descrito se encuentra reflejado en el apartado de planos de este proyecto, más concretamente en los planos 1 y 2.
El plano 1 contiene las 4 vistas de una de las láminas con sus medidas, es decir, el alzado, perfil, y planta además de una vista 3D y el plano 2 se corresponde con una vista 3D de las dos láminas cruzadas que constituyen el soporte, junto con las tuercas de mariposa colocadas por debajo para poder atornillar los soportes de micrófonos.

Por último, cada uno de los tres micrófonos se conectan por cable de audio con conexión XLR en ambos extremos a tres entradas distintas de la Scarlett Focusrite 18i20.

Figura 3. 1.- Agrupación de micrófonos (array) con soporte para los micrófonos mediante dos láminas de aluminio.

Scarlett Focusrite 18i20: Es una interfaz de audio de alta velocidad que se conecta por USB a nuestro ordenador y a nosotros nos funciona como preamplificador de micrófono o como tarjeta de sonido, de manera que se pueda trabajar simultáneamente con las tres señales recogidas por los micrófonos. Con esta interfaz es posible ajustar la señal de salida de cada micrófono por separado.

Ordenador Personal (PC): Ordenador Lenovo, modelo Ideapad 320 con 12GB de RAM y procesador Intel Core i5 séptima generación. Cumple los requisitos para poder implementar nuestro sistema.

Matlab R2017b: Va instalado en el ordenador personal, nos sirve para desarrollar los scripts informáticos que posteriormente invocará nuestra aplicación. La aplicación desarrollada se iniciará ejecutándola desde este entorno.

Monitor M-Audio BX5 D2: Es nuestra fuente sonora y la usamos para reproducir los escenarios que contienen los eventos sonoros de interés, es decir, truenos, lluvia o sonido ambiente, de manera que, a la hora de realizar las pruebas en espacio exterior, nuestro sistema capture, detecte y clasifique. En el caso concreto de los truenos, este monitor se colocará a distintas orientaciones de los micrófonos para permitir comprobar la capacidad de localización de la dirección de procedencia.
Es un elemento externo a nuestro sistema, se ha añadido únicamente para realizar las pruebas en exteriores.

3.2.2 Interconexionado de los elementos del sistema.

Este sistema puede implementarse de dos maneras: con un archivo de audio de grabación como entrada, donde sólo será necesario nuestro ordenador con Matlab o bien, con entrada microfónica, en este caso será necesario interconectar los elementos que lo componen, de manera que se nos permita recoger por medio de los micrófonos la señal emitida por el altavoz, para así procesarla y obtener la información deseada.
Los elementos son los siguientes:

- Matlab R2017b
- Ordenador portátil Lenovo Ideapad 320
- Interfaz de audio Focusrite Scarlett 18i20
- Array de tres micrófonos
- Monitor M-Audio BX5 D2

Se conectarán como está reflejado en la figura 3.2:

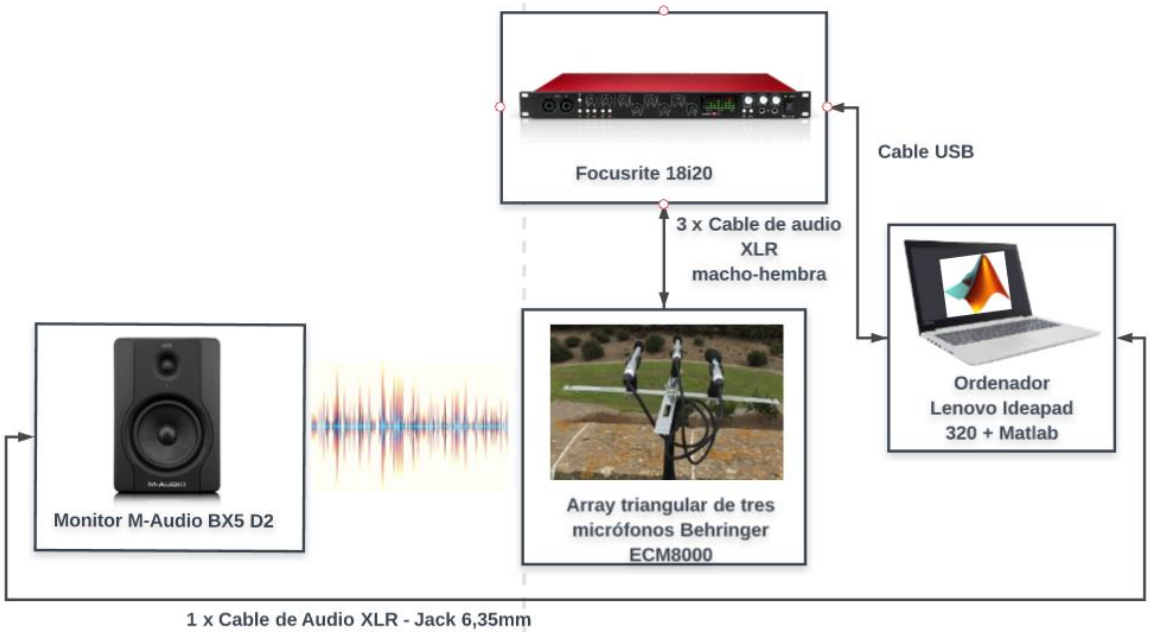


Figura 3. 2. – Interconexionado de los elementos del sistema.

El **funcionamiento** del sistema en **exteriores** será como sigue:

1. El altavoz, conectado al ordenador por el cable de audio tipo XLR a Jack 6.35mm, emite sonidos que contienen los eventos sonoros atmosféricos de interés o el sonido ambiente.
2. Se recoge la señal por el array de tres micrófonos conectados por cable de audio tipo XLR macho-hembra a la Scarlett Focusrite 18i20, cuyas funciones son las de preamplificar el sonido y trabajar simultáneamente con las tres señales de micrófonos.
3. La Scarlett Focusrite 18i20 envía la señal al ordenador, se conecta con este mediante puerto USB.
4. Matlab reconoce la interfaz de audio una vez que se han instalado los drivers de esta, y es capaz de recoger los datos obtenidos gracias a la Audio System Toolbox que incorpora. Será necesario crear un objeto de sistema de Matlab, encargados de leer las muestras de audio a partir de la entrada. En este caso, como la entrada es microfónica y se quiere probar el sistema en exteriores, se llama audioDeviceReader.
5. Por último, se ponen en funcionamiento los distintos algoritmos que realizan las tareas dependiendo de las funcionalidades de nuestro sistema. Esto es, establecer umbrales a partir del ruido de fondo de la sala en la que esté implementado para distinguir entre el silencio y el sonido, clasificar los sonidos en función de las bases de datos realizadas en el entrenamiento y si se trata de truenos, localizar la dirección de procedencia. Mostrando por pantalla en todo momento, a través de la aplicación la información correspondiente, en función de la operación que se esté realizando.

El siguiente esquema (Figura 3.3) presenta cómo se desarrolla el flujo de datos dentro del programa:

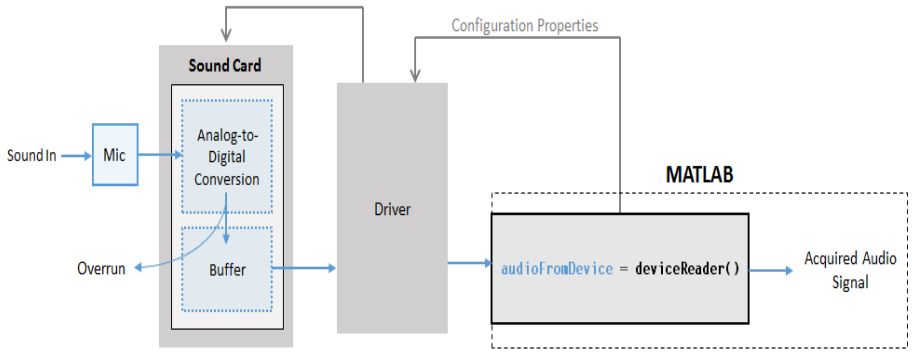


Figura 3. 3.- Flujo de datos en audioDeviceReader [47].

El objeto audioDeviceReader [47] recoge los datos desde la interfaz de audio (en nuestro caso Focusrite Scarlett 18i20), la cual se comunica con Matlab mediante sus drivers que deben estar instalados para poder trabajar con la interfaz, en nuestro caso ha sido necesario descargar los drivers conocidos como ASIO, que son los drivers más extendidos para trabajar con audio. De esta manera, Matlab puede recibir las muestras de audio para realizar las operaciones que correspondan.

El **funcionamiento** del sistema cuando la entrada es un **archivo de audio** que contiene una grabación de lo que se quiere procesar será como sigue:

1. En este caso, el objeto encargado de leer las muestras de audio a partir de la entrada se llamará audioFileReader.
2. Una vez que Matlab tiene los datos de entrada de la señal de audio, es capaz de trabajar sobre las muestras, realizando DSP sobre ellas.
3. Por último, se ponen en funcionamiento los distintos algoritmos que realizan las tareas dependiendo de las funcionalidades de nuestro sistema. Esto es, establecer umbrales a partir de los silencios del archivo de audio para distinguir entre el silencio y el sonido, clasificar los sonidos en función de las bases de datos realizadas en el entrenamiento y si se trata de truenos, localizar la dirección de procedencia. Mostrando por pantalla en todo momento, a través de la aplicación la información correspondiente, en función de la operación que se esté realizando.

3.2.3 Calibración del sistema. Respuesta en frecuencia

La calibración se realiza para evitar que ciertos factores influyan en las medidas y las distorsionen, estos factores pueden ser las desviaciones en la fabricación que hacen que las señales recogidas por los micrófonos del array difieran entre sí o bien, las condiciones ambientales como la temperatura ambiente, la presión del aire o la humedad.
Calibrando el sistema todos los micrófonos devolverían el mismo nivel de salida al estar conectados a un preamplificador. Para proceder a calibrar el sistema, usamos el calibrador de B&K 4231 que nos proporciona un tono de 1 kHz con un nivel de presión sonora de 94 dB. Para compensar las posibles derivas se procede al ajuste de la ganancia de cada canal.

La respuesta en frecuencia de cada uno de los micrófonos que usaremos (Behringer ECM8000) se muestra en la figura 3.4 y abarca todo el rango audible (20 Hz – 20 kHz).

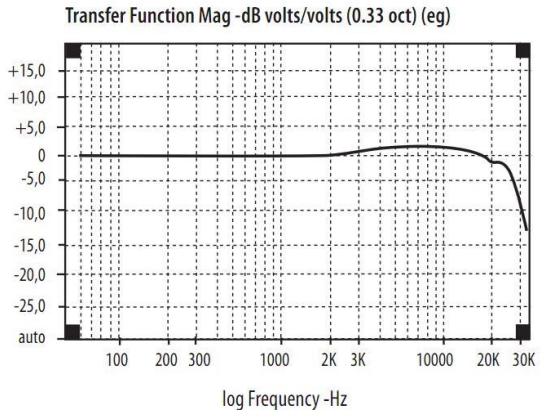


Figura 3. 4.- Respuesta en frecuencia del micrófono Behringer *ECM8000.*

3.3 Agrupación de Sensores para la Detección de la Dirección de Llegada

Hemos pensado en realizar un sistema mínimo y para ello, el funcionamiento tiene que ser lo más sencillo posible y adaptarse a nuestras necesidades.
Lo que buscamos es que parámetros como el tiempo de cómputo o el flujo de datos sea el menor posible además de que el número de micrófonos sea el mínimo. Estas tres condiciones están relacionadas ya que un menor número de micrófonos implica un menor flujo de datos y, por lo tanto, un menor tiempo de cómputo. Un mayor número de micrófonos supondría una mayor precisión, por lo tanto, mayor flujo de datos y seguidamente, mayor tiempo de cómputo, pero eso no es lo que estamos buscando.
En la Figura 3.5 se muestran los diagramas de directividad de cuatro distribuciones distintas de arrays de micrófonos en función de los ángulos que forman los lados que constituyen el triángulo.
Para nuestro sistema, el número mínimo de micrófonos que debe contener nuestro array y que se adapta a nuestras necesidades es tres.

Las cuatro distribuciones en función de los ángulos de los vértices del triángulo y que se muestran en la figura 3.5 son las siguientes:

a) Dos micrófonos separados una distancia en la horizontal.
b) Tres micrófonos formando un triángulo equilátero.
c) Tres micrófonos formando un triángulo isósceles.
d) Tres micrófonos formando un triángulo escaleno.

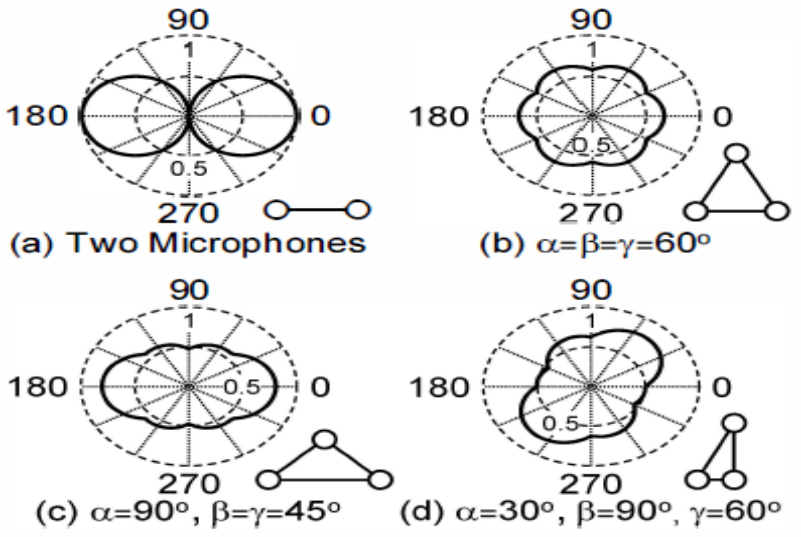


Figura 3. 5.- Directividad de cuatro distribuciones distintas de micrófonos [46].

Donde alfa, beta y gamma son los ángulos correspondientes a cada vértice del triángulo (figura 3.6):

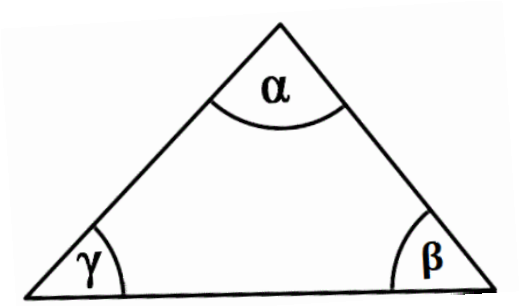


Figura 3. 6.- Ángulos correspondientes a los vértices del triángulo.

Escogemos el modelo b, el cual se corresponde con tres micrófonos formando un triángulo equilátero, esto es debido a que con dos micrófonos separados una distancia en la horizontal podemos detectar la llegada de sonido desde direcciones de -60 a 60 grados como veremos en el capítulo de los resultados, pero no podemos determinar el resto de direcciones, es decir, no podemos detectar si el sonido procede de la zona delantera o trasera de la línea que separa los dos micrófonos. Con tres micrófonos además de captar de qué lado procede el sonido, también podemos determinar si el sonido procede de la zona delantera o trasera y, además, elegimos el caso de un triángulo equilátero ya que es lo que más se aproxima al modelo omnidireccional, de esta manera podemos captar la dirección de llegada del sonido procedente del mayor número de direcciones posibles usando las técnicas DOA con una precisión aceptable.
Nuestro triángulo tiene 17 cm de lado o de separación entre micrófonos, ya que se corresponde con media longitud de onda de tomando como frecuencia 1kHz y como velocidad del sonido 340 m/s.

$$\lambda = \frac{c}{f} = \frac{340}{10^3} = 0{,}34m \rightarrow \frac{\lambda}{2} = 0{,}17m$$

Con un triángulo equilátero, tendremos tres sub-agrupaciones (sub-arrays) iguales o tres pares de micrófonos unidos por la línea que une cada lado del triángulo y que podemos analizar individualmente, dependiendo del ángulo del que provenga el sonido se seleccionará automáticamente el par que corresponda.

Los ángulos que recogen los pares de micrófonos se presentan en coordenadas meteorológicas, la relación con las coordenadas cartesianas se puede apreciar visualmente en la figura 3.7.

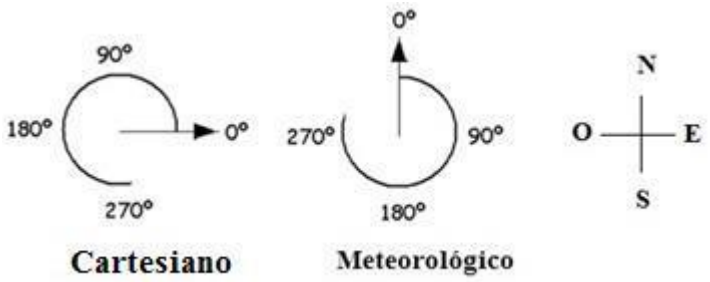


Figura 3. 7.- Relación entre coordenadas cartesianas y meteorológicas [50].

Observamos que los ángulos en el eje cartesiano se encuentran en sentido antihorario y en el meteorológico se encuentran en sentido horario. Hemos decidido usar este sistema debido a que nos movemos en el ámbito de la meteorología y resulta más adecuado usarlo.

En la figura 3.8, tenemos el triángulo equilátero formado por los tres micrófonos y los tres pares. El par 1 está formado por los micrófonos 1 y 2, seguidamente, el par 2 está formado por los micrófonos 2 y 3 y por último, el par 3 está formado por los micrófonos 1 y 3.

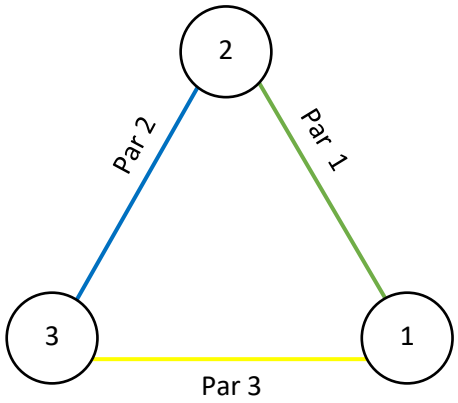


Figura 3. 8.- Triángulo equilátero con los tres micrófonos y los tres pares.

Cada par se puede usar tanto por la zona delantera como por la zona trasera del array formado por los dos micrófonos.
Además, observamos que cada una de las tres líneas que representan a cada par tiene un color diferente, usaremos los mismos colores posteriormente en la figura 3.9.

En la figura 3.9 se refleja que por cada uno de estos pares se recogen aproximadamente 60 grados tanto por la zona delantera como por la zona trasera, en total cada par captura aproximadamente 120 grados. Cada color de la leyenda representa un par distinto. Al sumar las cantidades que recoge cada par tanto por delante como por detrás, se cubre la totalidad del área de la circunferencia, es decir los 360 grados.

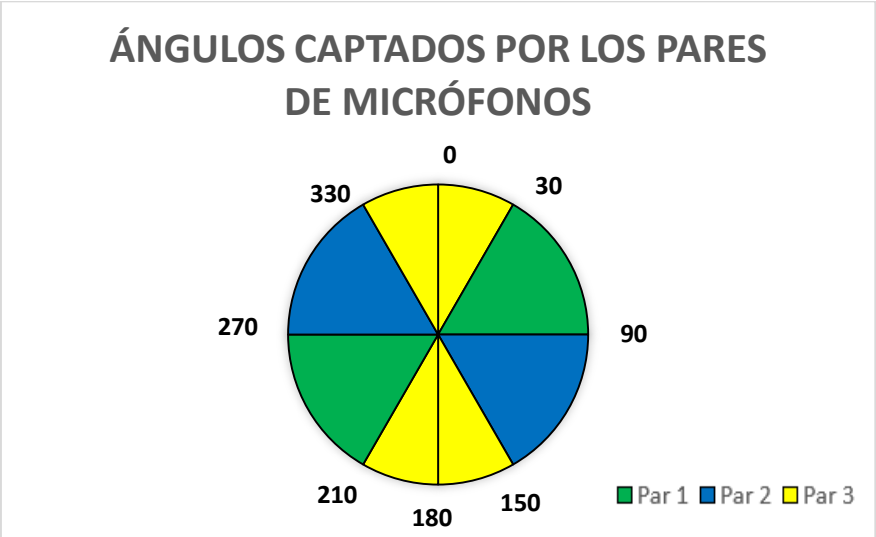


Figura 3. 9 . - Aproximación del área de cobertura captada por cada par de micrófonos sobre la circunferencia.

Esta figura (3.9) se ha tenido en cuenta para el diseño del sistema, pero la realidad es diferente, ya que en algunas secciones de la circunferencia existen varios grados de diferencia. Una mayor aproximación de lo que recoge cada par se presenta en la tabla 3.1:

Tabla 3. 1.- Mayor aproximación del rango de ángulos captados por cada par de micrófonos.

	Zona delantera (°)	Zona trasera (°)
Par 1	(30,88)	(210,274)
Par 2	(89,150)	(275,329)
Par 3	(330,29)	(151,209)

[Esta página ha sido dejada intencionalmente en blanco]

CAPÍTULO 4: SÍNTESIS DE LOS ESCENARIOS SONOROS A EVALUAR. DISEÑO Y EXPLICACIÓN

[Esta página ha sido dejada intencionalmente en blanco]

4.1 Introducción

En este capítulo se explica el diseño de los distintos escenarios sonoros y su finalidad. Además, se enumera y se describe cada escenario que ha sido creado.
Entre estos escenarios se encuentran los simulados desde el software y los simulados desde el exterior en recinto cerrado.

4.2 Diseño y objetivos de los escenarios sonoros

Los escenarios sonoros se crean con una aplicación de grabación o edición de audio, como por ejemplo el Audacity. Se han creado teniendo en cuenta cuatro elementos, dónde se van a simular (en software o en exteriores), cuántos canales contendrán (1,2 o 3), qué sonidos se van a introducir en ellos, en nuestro caso estarán compuestos de eventos sonoros singulares, estos pueden ser, sonidos de lluvia, de truenos y de sonido ambiente y cuántos silencios contendrán.
Se crean con el objetivo de realizar pruebas tanto en software como en exteriores para simular las condiciones reales, es decir simular que existen los sonidos de manera natural. Esto es debido a que probar nuestro sistema en condiciones reales sería difícil ya que habría que esperar a que sucedan esos fenómenos atmosféricos de manera natural.
Pueden estar compuestos sólo de uno o de más de un evento sonoro de interés, dependiendo de las pruebas que se quieran realizar. Las pruebas que se realizarán con estos escenarios sonoros son, comprobar las funcionalidades para las que ha sido diseñado el sistema, es decir, detectar y clasificar los eventos sonoros atmosféricos y, por último, localizar la dirección de procedencia solo cuando se trate de truenos.
En nuestro caso, hemos creado tantos escenarios sonoros como pruebas distintas hemos realizado, tanto en software como en exteriores, y de esa manera comprobamos la correcta actuación de nuestro sistema para las funcionalidades para las que ha sido diseñado. Podemos comprobar cada funcionalidad del sistema tanto por separado como en conjunto.
Entonces, crearemos escenarios sonoros para realizar las pruebas individuales, es decir, para la detección, para la clasificación y para la localización de la dirección de procedencia, sólo en el caso de los truenos. Y por último, crearemos escenarios sonoros para probar todo el sistema en conjunto, con las tres funcionalidades realizadas una después de otra. Los resultados numéricos obtenidos en las pruebas en conjunto no variarán respecto a las pruebas individuales ya que configuramos el sistema con los mismos parámetros.
El trueno y la lluvia que hemos utilizado para incluirlos en los escenarios sonoros, los hemos obtenido de la página 99 sounds [38].
Nos hemos descargado una biblioteca de sonido gratuita llamada “[99sounds] Rain and Thunder” con 64 muestras o archivos de audio de alta calidad que, según sus especificaciones se registraron durante una tormenta de primavera en las calles de Belgrado y se grabaron a una profundidad de 24 bits utilizando un grabador de campo estéreo Edirol R-09 portátil.

Esta biblioteca contiene varios tipos de grabaciones de truenos y lluvia, aunque nosotros para entrenar y probar el sistema, hemos utilizado los truenos lejanos, truenos fuertes, lluvia en desagüe y lluvia en ventana.
Además, el sonido ambiente lo hemos captado desde la azotea del Edificio de Telecomunicación y Electrónica conectando el array de tres micrófonos formando un triángulo equilátero a la grabadora (figura 4.1).

Figura 4. 1.– Array de tres micrófonos conectado a la grabadora para recoger el sonido ambiente.

En este trabajo se han implementado dos tipos de escenarios sonoros: los simulados desde el software y los simulados desde el exterior en recinto cerrado.

4.3 Escenarios sonoros simulados desde el software

Estos escenarios se usan para simular el funcionamiento de nuestro sistema de manera interna, es decir, desde nuestro ordenador y los archivos se leen desde el Matlab. Para usar los escenarios no nos hace falta el sistema completo, sólo nuestro ordenador.
Con estos escenarios se pueden ajustar los parámetros que usan los programas que son los responsables de cada una de las funcionalidades del sistema, así como configurar los modos de funcionamiento, de manera que sean los más adecuados para que el sistema funcione de la mejor manera posible tanto en las tareas individuales como en conjunto. Esto es, primero detectar que existe sonido, luego clasificar los distintos eventos sonoros y por último localizar la dirección de procedencia, sólo en el caso de los truenos antes de probar el sistema en exteriores.

I. ESCENARIO PARA LAS PRUEBAS DE DETECCION SIMULADAS DESDE EL SOFTWARE

En primer lugar, para las pruebas de *detección* simuladas desde el software, en cuanto a los sonidos, queremos que nuestro escenario sonoro contenga los tres eventos sonoros. Además, tiene que existir silencio antes de que se produzca el evento sonoro, ya que, por un lado, en la detección se tienen que escuchar los silencios del archivo de audio para establecer los umbrales y por otro lado, tenemos que asegurarnos de que el sistema diferencia bien entre silencios y sonidos, actuando cuando existan sonidos y dejando de actuar cuando detecte silencios. En cuanto al número de canales, siempre se va a seleccionar un canal sobre el que se mirará si existe detección. Por lo tanto, con un canal es suficiente.
Se pueden apreciar los retardos de este escenario en la figura 4.2.

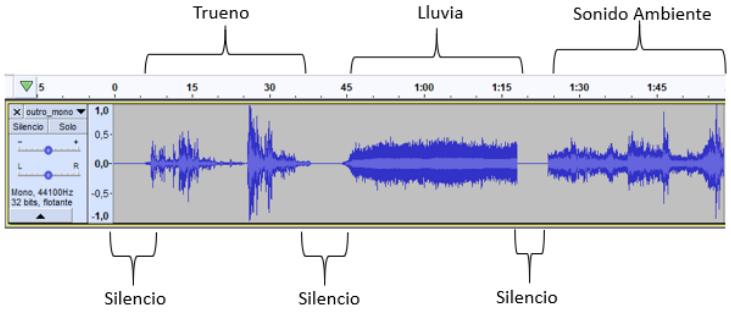


Figura 4. 2.– Escenario sonoro para las pruebas de detección simuladas desde el software.

II. ESCENARIO PARA LAS PRUEBAS DE CLASIFICACION SIMULADAS DESDE EL SOFTWARE

En segundo lugar, para las pruebas de *clasificación* simuladas desde el software, nos es indiferente el número de canales que contengan los archivos de audio, ya que existe un algoritmo que selecciona el más energético, por lo que en este caso siempre se trabajará con señales monofónicas. Pero sí es importante que contenga todos los sonidos o las tres clases para asegurarnos de que las distingue correctamente. Para poder calcular la precisión en la clasificación, hemos realizado un archivo de 1 minuto y 47 segundos de duración, debido a que con este periodo de tiempo se puede dividir cada una de las tres clases en fragmentos de 34 segundos. Además, este periodo de tiempo es divisible por la longitud del frame de audio, (75.000 muestras), de este modo podemos contabilizar los aciertos del sistema de una manera sencilla.

$$34s * 3\ (clases) = 102\ s = \ 1\ \mathrm{min}\ 47\ s$$

Si trabajamos a una frecuencia de muestreo de 44.100 Hz, las 75.000 muestras corresponden a 1,7 segundos de sonido los que procesamos del archivo de audio y usamos para clasificar. A su vez, cada fragmento del escenario sonoro correspondiente a cada clase se dividirá entre 20 particiones o iteraciones del bucle, resultantes de dividir 34 segundos entre 1.7 segundos, con lo que el archivo de audio o escenario sonoro tendrá en total 60 iteraciones (20 por cada una de las 3 clases contempladas).
Para conocer el porcentaje de aciertos, se dividen las clases que el sistema coloca correctamente, entre los aciertos reales (20 como máximo por clase) y ese resultado se multiplica por 100. Esto se realiza en cada fragmento, con la misma duración correspondiente a cada una de las clases. Este escenario está representado en la figura 4.3.

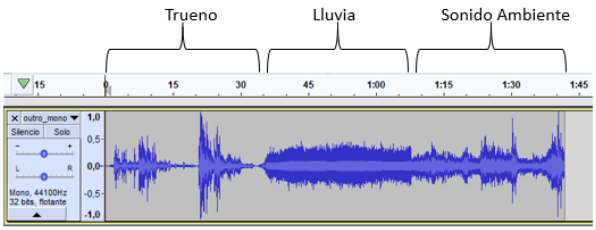


Figura 4. 3.– Escenario sonoro para las pruebas de clasificación simuladas desde el software.

III. ESCENARIO PARA LAS PRUEBAS DE DOA SIMULADAS DESDE EL SOFTWARE

En tercer lugar, para las pruebas de *localización de la dirección de procedencia*, simuladas desde el software, sí que es importante el número de canales que posean los archivos de audio, ya que cada canal simulará el sonido recogido por cada uno de los micrófonos con sus respectivos retardos, estos serán mayores o menores dependiendo del ángulo de incidencia del sonido que dependerá de la posición de la fuente sonora.
Los retardos de cada canal simularán la posición (dirección angular) en la que se encuentra la fuente con respecto del centro del array de micrófonos dado que, teniendo los retardos de los distintos canales, con las técnicas de DOA podemos localizar la dirección de procedencia del sonido emitido por la fuente. En concreto con el algoritmo TDOA.
Así, la señal multicanal puede ser descrita por la siguiente ecuación (4.1):

$$\begin{cases} x_1(t) = a_1 \cdot h_1(t) * s(t) + n_1(t) \\ \vdots \\ x_i(t) = a_i \cdot h_i(t) * s(t - \tau_i) + n_i(t) \\ \vdots \\ x_L(t) = a_L \cdot h_L(t) * s(t - \tau_L) + n_L(t) \end{cases} \qquad \textit{Ec.4.1}$$

Donde $x_i(t)$ denota una señal recibida por el i-ésimo micrófono y retrasada por un retardo τ_i con respecto al primer micrófono (de referencia), mientras que $s(t)$ representa una señal fuente que se atenúa en el mismo micrófono por $a_i(t)$ y se distorsiona por un ruido ambiental $n_i(t)$ recibido por este micrófono. Por último, $h_i(t)$ se conoce como una respuesta de impulso del canal [16].

Entonces, se crean archivos de simulación con tantos canales como micrófonos queramos simular. Siendo dos o tres el número de micrófonos.
Cuando queramos simular dos micrófonos, se crea un archivo de audio estéreo y cuando queramos simular tres, se crea un archivo de audio de tres canales pudiendo captar más direcciones con tres micrófonos como hemos podido comprobar en el capítulo anterior.

Si tenemos dos canales, tenemos la siguiente representación (Figura 4.4), donde:

- S(n) es la onda incidente.
- d es la distancia de la fuente al centro del array, en nuestro caso una constante de valor 0,17cm o lo que es lo mismo, la distancia entre los dos micrófonos.
- θ es el ángulo de inclinación de la fuente respecto a la línea que separa los dos micrófonos, puede variar, dependiendo del ángulo el cual queramos calcular su retardo. Como es en un eje, solo se calcula su seno. Siempre trabajaremos en radianes.

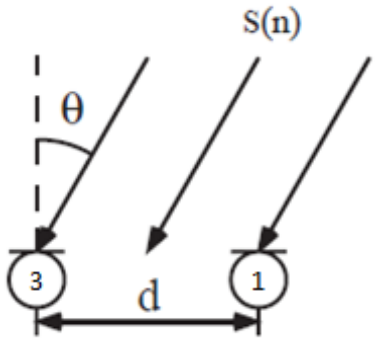


Figura 4. 4 – Onda incidente para dos canales [48].

Los retardos serán como máximo de 0,5ms, resultantes de la división de la distancia que resulta ser de 0,17cm multiplicada por el seno de θ cuando es máximo, es decir vale uno o lo que es lo mismo, incide desde 90º, todo eso entre la velocidad del sonido, c, 340m/s, por lo que no se percibirá mucho desfase entre los canales.

$$\tau máx = \frac{d \cdot sen(\theta)}{c} = \frac{d}{c} = \frac{0.17 \cancel{m}}{340\ \cancel{m}/s} = 0{,}5ms$$

Para calcular el retardo de uno de los canales con respecto al otro, hemos aplicado la fórmula de la ecuación 4.2.

$$\tau = \frac{d \cdot sen(\theta)}{c} \qquad \textit{Ec.4.2}$$

Donde c es la velocidad del sonido, en nuestro caso constante de valor 340m/s.
Como se puede apreciar en la figura 4.5 en el caso de dos canales, los retardos no serán muy grandes y por lo tanto, para observarlos hay que hacer bastante zoom en la señal de audio. En este caso, primero le llega al canal 1 y luego al 2.

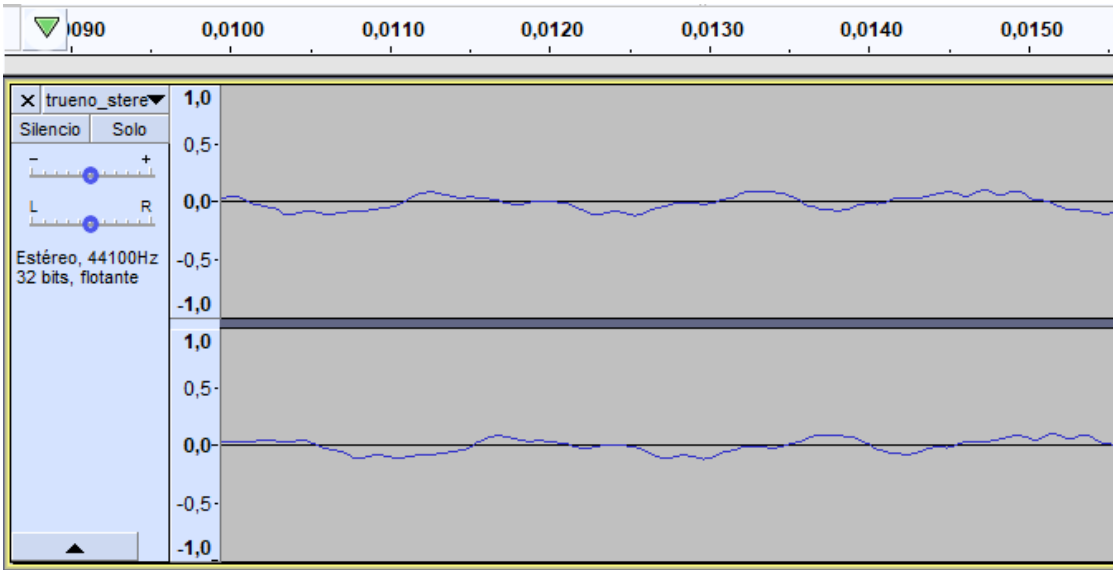


Figura 4. 5 .-Escenario sonoro para localización de la dirección de procedencia con dos canales y distintos retardos para cada canal.

En el caso de tres canales, tenemos la siguiente representación (Figura 4.6), donde:

- d es la distancia entre dos micrófonos, en nuestro caso una constante de valor 0,17cm o lo que es lo mismo, la longitud de los lados del triángulo.
- θ es el ángulo de inclinación de la fuente respecto a la línea vertical del centro de cada uno de los micrófonos, puede variar, dependiendo del ángulo el cual queramos calcular su retardo. Se calcula en radianes cuando se usa en el Matlab.
- d1, d2 y d3 son las distancias de la fuente a cada uno de los micrófonos.

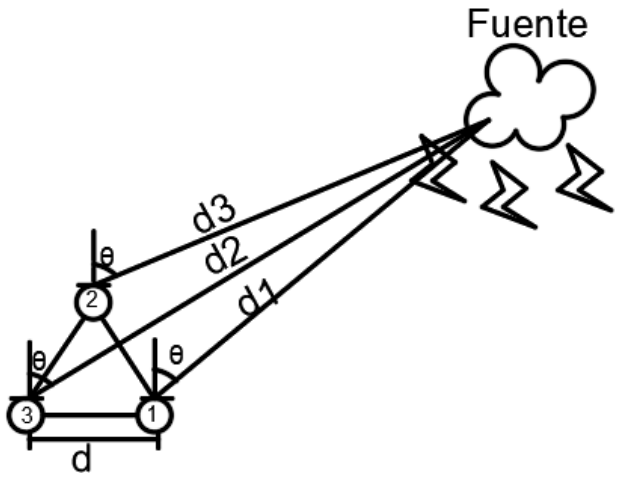


Figura 4. 6.– Representación de la distancia euclídea para tres canales.

Para calcular cada una de las distancias (d_1, d_2 y d_3) hemos aplicado la fórmula de la distancia euclídea (Ecuación 4.3)

$$d_N = \sqrt{(posFuente_x - posmicN_x)^2 + (posFuente_y - posmicN_y)^2} \qquad Ec.4.3$$

Luego, para calcular el retardo, dividimos esa distancia entre la velocidad del sonido que resulta ser de 340 m/s (Ecuación 4.4).

$$\tau = \frac{d_N}{c} \qquad Ec.4.4$$

Para ello, antes de aplicar las fórmulas anteriores hay que definir las posiciones de los micrófonos en el array con sus coordenadas x e y.

$$posmic_1 = (0'085, 0)$$

$$posmic_2 = (0, 0'15)$$

$$posmic_3 = (-0'085, 0)$$

Después, definimos la posición de la fuente sonora, en nuestro caso, un trueno. Con sus coordenadas x e y.

$$posFuente = (\mathrm{d} \cdot sen(\theta), \mathrm{d} \cdot \cos(\theta))$$

Donde:

- d es la distancia de la fuente al centro del array de micrófonos, en nuestro caso constante de 1 metro.
- θ es el ángulo de localización de la fuente respecto de cada uno de los micrófonos, puede variar, dependiendo del ángulo el cual queramos calcular su procedencia. Se calcula en radianes cuando se usa en el Matlab.

Por último, ya podemos sustituir en las ecuaciones 4.2 y 4.3.

Como se puede apreciar en la figura 4.7, para el caso de tres canales los retardos tampoco serán muy grandes y hay que hacer mucho zoom en el Audacity, en este ejemplo se percibe que primero le llega al canal 3, luego al 2 y luego al 1.

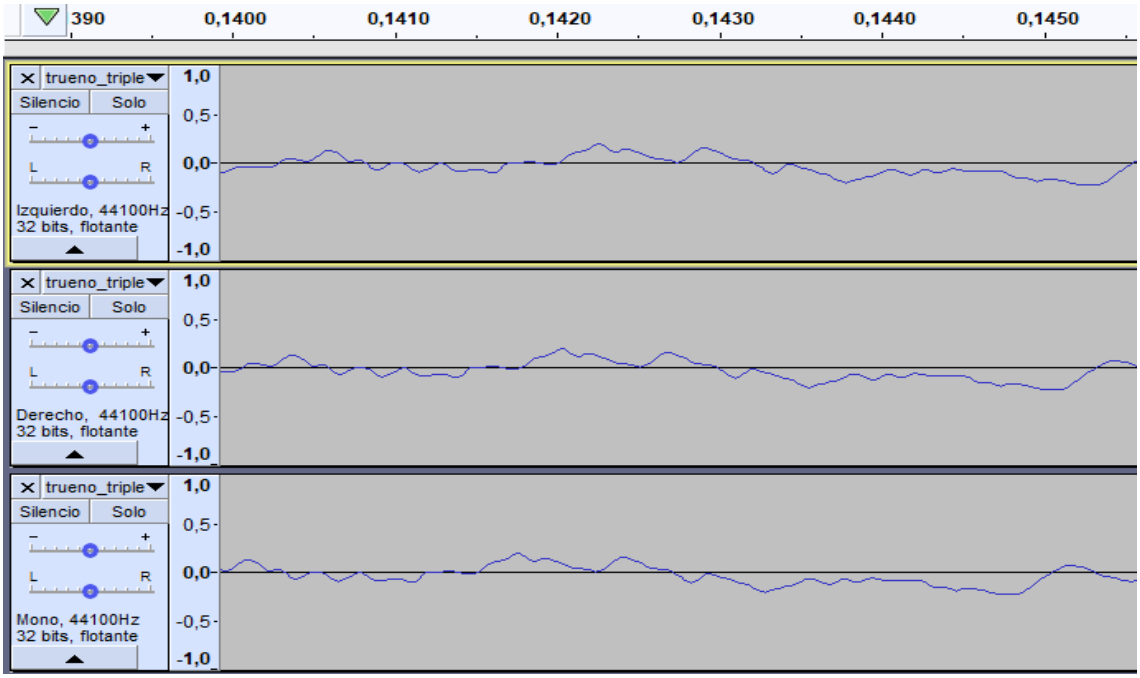


Figura 4. 7 .– Escenario sonoro para localización de la dirección de procedencia con tres canales y distintos retardos para cada canal.

Para crear los dos últimos escenarios de dos y tres canales respectivamente, relacionados con las pruebas de localización de la dirección de procedencia, se ha utilizado el programa Matlab ya que podemos implementar un algoritmo que al introducirle un archivo de truenos mono nos cree otro archivo con dos o tres canales y que a cada canal le asigne su respectivo retardo.

IV. ESCENARIO PARA LAS PRUEBAS EN CONJUNTO SIMULADAS DESDE EL SOFTWARE

Una vez se han realizado las pruebas con el escenario sonoro correspondiente a cada funcionalidad, se prueba el sistema completo, de esa manera se prueban las tres funcionalidades, una detrás de otra, para las que nuestro sistema está diseñado.
Para probarlo en software se usa el escenario de la figura 4.8, con tres canales para usarlos cuando se detecten truenos, ya que con el algoritmo TDOA, al localizar la dirección de procedencia hay que emplear los retardos de cada canal. En la figura 4.8 se

puede apreciar el silencio inicial que está unido a los retardos de cada canal. En primer lugar, la señal le llega al micrófono 1, luego al 2 y por último al 3, este caso concreto es para 90°.

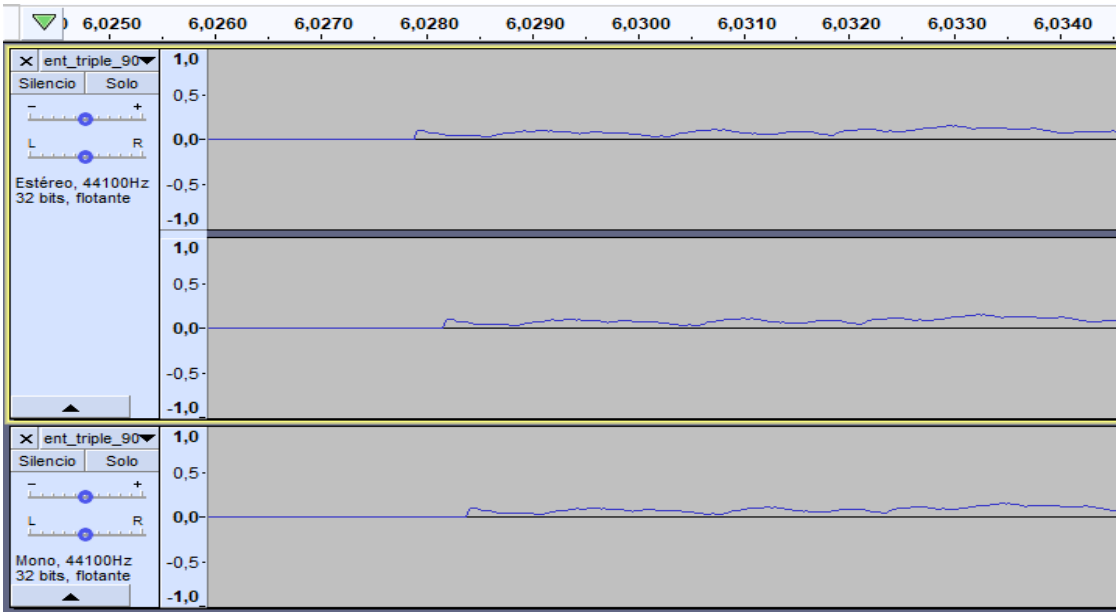


Figura 4. 8.- Escenario sonoro para probar el sistema completo simulando desde el software.

4.4 Escenarios sonoros simulados desde el exterior en recinto cerrado

Estos escenarios se usan para simular el funcionamiento de nuestro sistema en exteriores en recinto cerrado, es decir, emitiendo el sonido de estos escenarios desde el monitor y recogiendo la señal por el array de micrófonos conectado al Scarlett Focusrite 18i20 que a su vez está conectada al ordenador para así comprobar que nuestro sistema detecta el sonido y clasifica los distintos eventos sonoros. Por último, si se quiere localizar la dirección de procedencia, se emiten truenos a distintos ángulos de procedencia, y se comprueba que también funciona correctamente.
Se han probado en el lugar donde hemos hecho las pruebas en exteriores, esto es en el laboratorio de electroacústica del edificio de Telecomunicación y Electrónica.
El momento para poner en funcionamiento estos escenarios será después de haber ajustado los parámetros (de cada algoritmo correspondiente a cada funcionalidad) simulando desde el software.

I. ESCENARIO PARA LAS PRUEBAS DE DETECCION SIMULADAS DESDE EL EXTERIOR

Para las pruebas de *detección* simuladas desde el exterior, se han reproducido desde el altavoz los distintos eventos sonoros atmosféricos a modo de escenario, pero dejando un tiempo de silencio antes para que el sistema calcule el umbral de detección.
Este escenario se realiza para que el sistema sepa distinguir entre los periodos de silencio, en los que se encontrará en reposo y los periodos de sonido, en los que trabajará.

II. ESCENARIO PARA LAS PRUEBAS DE CLASIFICACION SIMULADAS DESDE EL EXTERIOR

Para las pruebas de *clasificación* simuladas desde el exterior se ha empleado el mismo escenario que para estas mismas pruebas simuladas desde el software.
Este escenario se realiza para que el sistema clasifique cada sonido, es decir, que cada vez que, entre un sonido nuevo, se le asigne una clase teniendo como referencia la base de datos que ya conoce.

III. ESCENARIO PARA LAS PRUEBAS DE DOA SIMULADAS DESDE EL EXTERIOR

Para realizar las pruebas de *localización de la dirección de procedencia* en exteriores, se usa un altavoz que nos sirve de fuente sonora, desde donde reproducimos truenos continuamente, esta vez lo movemos de manera real y lo colocamos a distintos ángulos del centro del array de tres micrófonos. Al ser únicamente un altavoz, la señal debe de ser monofónica.
Este escenario se realiza para que el sistema calcule la dirección de procedencia del sonido.

IV. ESCENARIO PARA LAS PRUEBAS EN CONJUNTO SIMULADAS DESDE EL EXTERIOR

Una vez se han realizado las pruebas en con el escenario sonoro correspondiente a cada funcionalidad, se prueba el sistema completo, pero esta vez desde el exterior en recinto cerrado y comprobamos que funciona correctamente para las tres funcionalidades, una detrás de otra, para las que nuestro sistema está diseñado.
Para probar el sistema completo en exteriores, se reproducen desde el altavoz los distintos eventos sonoros con silencios previos, se comprueba que el sistema distingue entre silencios y sonidos. Se comprueba que el sistema clasifica las clases para las que ha sido diseñado y por último, situamos el altavoz a distintos ángulos de los micrófonos para que si se trata de truenos, comprobar que localiza la dirección a la que se encuentra la fuente sonora que emite el trueno, en nuestro caso el altavoz (figura 4.9).

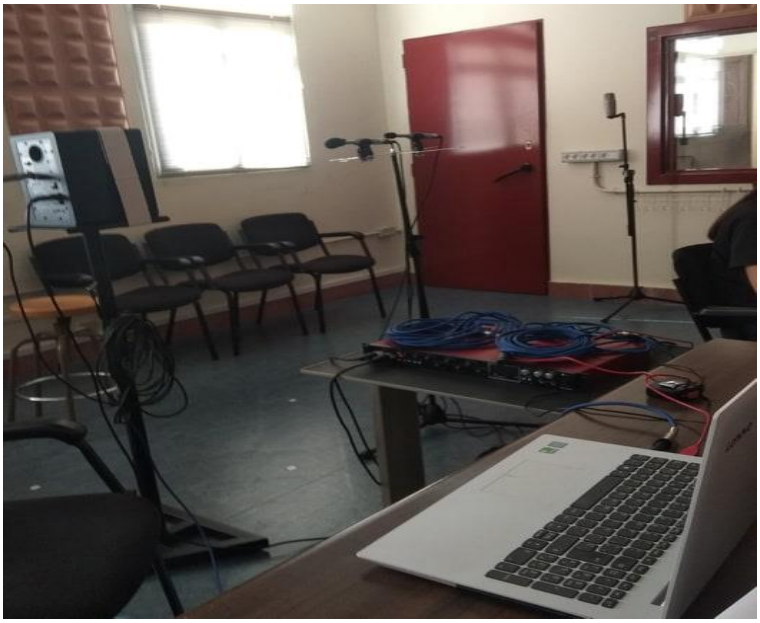


Figura 4. 9.– Sistema completo para la realización de pruebas en exteriores.

[Esta página ha sido dejada intencionalmente en blanco]

CAPÍTULO 5: RESULTADOS

[Esta página ha sido dejada intencionalmente en blanco]

5.1 Introducción

En este capítulo se presentan los resultados obtenidos para cada una de las técnicas o funcionalidades (detección, clasificación y localización) que utiliza nuestro sistema, en algunos casos para más de un algoritmo por funcionalidad como por ejemplo en la clasificación para comparar resultados o en la localización de la dirección de procedencia (DOA) que se han utilizado más de dos. Antes de obtener los resultados, se han ajustado los parámetros, para que nuestro sistema tenga el mejor funcionamiento cuando se combinen las tres técnicas.

Los parámetros que se han establecido en las técnicas son los siguientes:
El primero lo utilizan los objetos audioFileReader y audioDeviceReader en las tres funcionalidades, se denomina audioFrameLength y determina la cantidad de muestras de audio que procesa el sistema en tiempo real.
El segundo, es el tamaño en muestras de la ventana de energía(L), este parámetro se utiliza en la detección e indica el número de muestras que forman la ventana a usar para calcularle la energía a la señal.
El tercero y último, se utiliza en el caso del DOA (Direction of Arrival) y se denomina bufferLength. Determina la longitud del buffer o de pequeños almacenes de memoria intermedia donde se guardan los paquetes de muestras a procesar. Se utilizan para poder procesar la señal más fácilmente, al obtenerse mejores resultados que tomando todas las muestras que se recogen en tiempo real (audioFrameLength). Ya que tomando todas las muestras, el tamaño del buffer será demasiado grande para realizar los procesos y cálculos necesarios.

Para poder ver los resultados, se pone a prueba el sistema usando los distintos escenarios sonoros que han sido creados y que se han explicado en el capítulo anterior.

5.2 Resultados en la detección

El escenario software que se ha empleado para realizar estas pruebas es el que contiene silencios antes y después del evento sonoro de interés. Y para el escenario exterior se han realizado las pruebas dejando un silencio antes de reproducir el evento sonoro por el altavoz. Con el escenario software, se han ajustado los parámetros que luego se usarán en el escenario exterior.

Sólo hemos realizado pruebas con un algoritmo de detección, y es el de la detección basada en umbrales.
Los parámetros más importantes que se han establecido están reflejados en la tabla 5.1 y son, la longitud del frame (audioFrameLength) y la longitud de la ventana de energía (L).
El primer parámetro tiene un valor de 85.000 muestras debido a que estábamos buscando un valor mayor a 75.000 muestras, para luego poder quedarnos justamente con 75.000 muestras para la clasificación y la localización de la dirección de procedencia, entre la muestra de inicio y la de final. La longitud de la ventana promedio ha resultado ser de 1.000 muestras.

La elección de estos parámetros se ha realizado buscando una solución de compromiso entre precisión y fiabilidad.

Normalmente, la longitud de la ventana promedio (L) se corresponde con el valor de la marca inicial y la longitud del frame de audio, se corresponde con la marca final de la detección. Entonces, en nuestro caso, en la mayoría de las veces obtenemos detecciones de 84.000 muestras entre marca final y marca inicial, de las cuales solo cogemos las primeras 75.000 para la clasificación y si se trata de truenos, para realizar DOA.
Hay que añadir que solo aceptaremos detecciones mayores a 75.000 muestras, el resto serán desechadas ya que no es suficiente para realizar la clasificación o localizar la procedencia de la fuente.

Tabla 5. 1.- Parámetros importantes en la detección.

Duración del Frame de Audio (AudioFrameLength)	**Longitud de la ventana Promedio (L)**
85.000 muestras	1.000 muestras

5.2.1 Escenario software

RESULTADOS PARA LA DETECCIÓN BASADA EN UMBRALES

Como se comentó anteriormente, usando el escenario para las pruebas de detección simuladas desde el software se han ajustado los parámetros de detección que también usará nuestro sistema en el escenario exterior en recinto cerrado.
En el escenario software, cada silencio dura 6 segundos y cada sonido dura 34 segundos como se indicó en el capítulo anterior. Al realizar las pruebas, observamos que sólo se deja una detección fuera, la que va justo después del primer silencio.
Los resultados se muestran en la tabla 5.2. Nuestro sistema tiene un porcentaje de acierto en la detección del 95% y un porcentaje de fallo del 5%. Acertando en 57 detecciones de 60 totales y fallando en las tres detecciones restantes que las considera silencio.

Tabla 5. 2.- Resultados para la detección basada en umbrales, escenario software.

Porcentaje de acierto	**Porcentaje de fallo**
95% = $\frac{57}{60}x100$	5% = $\frac{3}{60}x100$

5.2.2 Escenario exterior en recinto cerrado

RESULTADOS PARA LA DETECCIÓN BASADA EN UMBRALES

Para realizar estas pruebas se deja un tiempo de 6 segundos antes de reproducir el sonido correspondiente a clasificar, para que el sistema calcule el umbral.
Los resultados se muestran en la tabla 5.3. Obteniendo los mismos que en el escenario software.

Tabla 5. 3.- Resultados para la detección basada en umbrales, escenario exterior.

Porcentaje de acierto	Porcentaje de fallo
95% = $\frac{57}{60}x100$	5% = $\frac{3}{60}x100$

5.3 Resultados en la clasificación

Para ver estos resultados usaremos el escenario sonoro que nos sirve para realizar las pruebas de clasificación desde el software, contiene todos los sonidos de interés y un solo canal.
Compararemos los resultados de KNN con los de Random Forest, y para ello observaremos tres factores: los porcentajes de acierto, fallo y falsa alarma, el tiempo que tarda en analizar el escenario sonoro que hemos usado para la simulación de software y la matriz de confusión.
Donde los porcentajes de acierto (PF), fallo (PF) y falsa alarma (PFA) se presentan en la ecuación 5.1:

$$PA(\%) = \frac{prediciones\ acertadas\ de\ esa\ clase}{suma\ de\ fragmentos\ de\ esa\ clase\ en\ todo\ el\ archivo}x100$$

$$PF(\%) = \frac{predicciones\ falladas\ de\ esa\ clase}{suma\ de\ fragmentos\ de\ esa\ clase\ en\ todo\ el\ archivo}x100 \qquad \textit{Ec.5.1}$$

$$PFA(\%) = \frac{falsos\ positivos\ de\ esa\ clase}{suma\ total\ de\ fragmentos\ de\ las\ clases\ en\ todo\ el\ archivo}x100$$

El tiempo en analizar el archivo de audio no siempre es constante, depende de las otras actividades que se estén realizando en el ordenador o lo que es lo mismo, del uso de la CPU, cuanto más libre esté más rápido irá el programa.

El único parámetro importante (tabla 5.4) que se ha utilizado para la clasificación es una longitud de frame de 75.000 muestras de audio (aproximadamente 1,7 segundos si hablamos de una frecuencia de muestreo de 44100 Hz). Un valor similar a la duración de la mayoría de los archivos de entrenamiento, la cual es de 1,5s y si ponemos el valor de 1,7 segundos, el clasificador presenta un mayor número de aciertos.

Tabla 5. 4.- Parámetros importantes en la clasificación.

Duración del Frame de Audio (AudioFrameLength)
75.000 muestras

5.3.1 Escenario software

RESULTADOS PARA KNN
Uno de los parámetros más importantes del algoritmo KNN es el valor de k o número de vecinos que influye en gran medida en el resultado de la clasificación. Sin embargo, no

existe un método que permita elegir un k adecuado, es decir, salvo la realización de pruebas con distintos valores de k.
Queremos elegir un k de manera que en las tres clases salga un buen porcentaje de aciertos, hemos considerado un buen porcentaje a partir del 60%.
Por este motivo, hemos implementado un algoritmo que realice esta tarea y nos represente al final el porcentaje en función del número de vecinos, usando el escenario sonoro que nos sirve para realizar las pruebas de clasificación desde el software. Esto se ha representado en la figura 5.1.

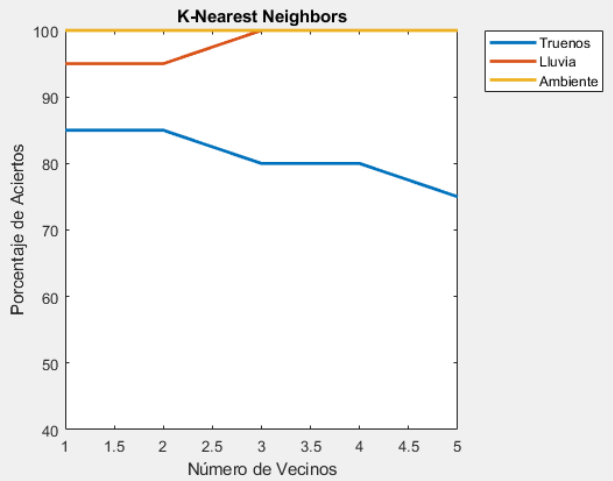


Figura 5. 1.- Porcentaje de aciertos en función del número de vecinos para KNN – Escenario Software..

Entonces, k puede ser de 1 o 2 vecinos, se puede apreciar en la figura 5.1 que esos son los valores donde nuestro sistema presenta el mayor porcentaje de aciertos para las tres clases en conjunto, sin embargo, elegimos 2 vecinos ya que el programa tarda menos tiempo en procesar el escenario sonoro, como veremos a continuación (Figura 5.2).

El siguiente parámetro que se ha valorado para ver la calidad del algoritmo KNN, es el tiempo que tarda en analizar el archivo de audio de pruebas o el escenario sonoro, en función del número de vecinos.

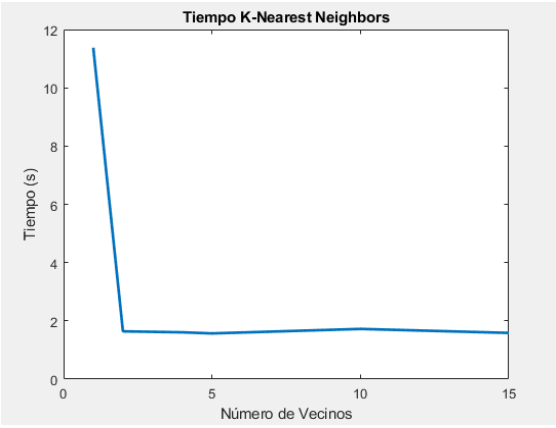


Figura 5. 2.- Tiempo en función del número de vecinos para KNN – Escenario Software.

En este caso, para 2 vecinos tarda un tiempo de 1,74 segundos en analizar el escenario sonoro que hemos usado para la simulación de software frente a los 10,5 segundos que tarda para 1 vecino.

Es un algoritmo rápido, ya que su única tarea es calcular distancias como se explicó en el capítulo 2.

Entonces, atendiendo a la ecuación 5.1, para el algoritmo KNN con 2 vecinos, usando el escenario software, tenemos los siguientes resultados, presentados en la tabla 5.5:

Tabla 5. 5.- Porcentaje de acierto, fallo y falsa alarma para KNN con k = 2 – Escenario Software.

	PA	PF	PFA
Truenos	$85\% = \frac{17}{20}x100$	$15\% = \frac{3}{20}x100$	$1{,}7\% = \frac{1}{60}x100$
Lluvia	$95\% = \frac{19}{20}x100$	$5\% = \frac{1}{20}x100$	$0\% = \frac{0}{60}x100$
Sonido ambiente	$100\% = \frac{20}{20}x100$	$0\% = \frac{0}{20}x100$	$0{,}05\% = \frac{3}{60}x100$

Referente a la tabla 5.5, se ha comprobado que los tres falsos negativos de la clase trueno, son confundidos con la clase sonido ambiente y el falso negativo de la clase lluvia, es confundido con la clase truenos.

Por último, obtenemos la matriz de confusión (representada en la figura 5.3). Es la resultante para el algoritmo KNN con 2 vecinos. Nos indica que, de los 13 truenos para test, los acierta todos. De los 56 archivos para lluvia, acierta 55 y falla en 1, que lo coloca en la clase de ruido ambiente. En la clase de ruido ambiente, los acierta todos.

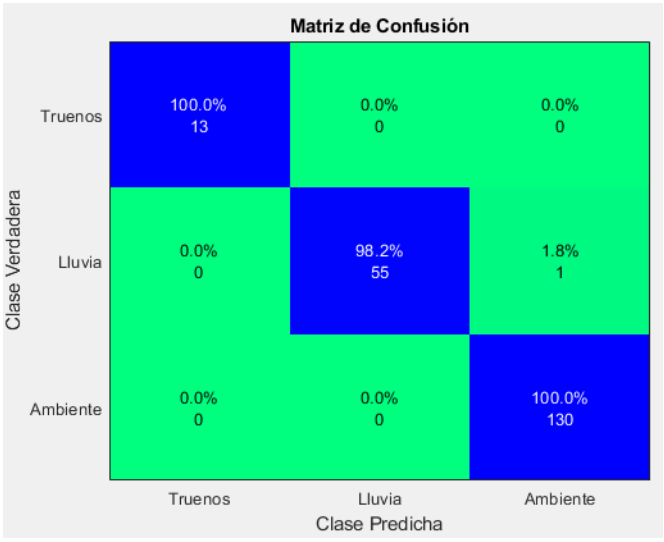


Figura 5. 3.- Matriz de confusión para 2 vecinos con KNN.

RESULTADOS PARA RANDOM FOREST

En cuanto a Random Forest, nos hemos dado cuenta que, una vez que entrenas el modelo, aunque mantengas los mismos parámetros, dibuja un árbol diferente y por lo tanto diferente número de aciertos como se detalló en el capítulo 2.

El motivo de los resultados diferentes es debido a que cada vez que se entrena, realiza las divisiones de los datos de manera diferente, escoge conjuntos de características

diferentes y el número de ramas o splits también es diferente cada vez que se entrena, estando dentro de un rango, que va de 1 al número máximo de ramas.
Para dibujar los árboles, hay dos variables que se pueden cambiar y son, el número de árboles y el máximo número de splits, también llamado divisiones o ramas.
Se ha optado por probar con un conjunto reducido de árboles para que vaya más rápido el programa y porque presenta mejores resultados.
En cuanto al número máximo de splits, también hemos optado por ponerlo pequeño para que el rango de números sea menor y la cantidad de árboles distintos al ejecutarlo también sea menor. En nuestro caso, hemos puesto un número de 10 ramas como máximo, es decir que cada vez que lo entrenas puede construir un árbol con 10 o menos ramas.
Para elegir el número de árboles, también se ha implementado un algoritmo que pruebe con distinto números de árboles y nos represente al final la precisión en función del número de árboles. Esto está representado en la siguiente figura (5.4).

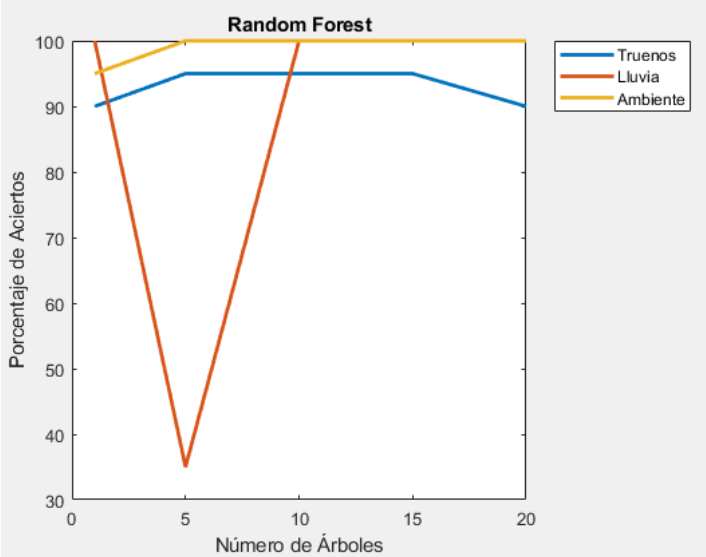


Figura 5. 4.- Porcentaje de aciertos en función del número de árboles para Random Forest – Escenario Software.

El número de árboles más adecuado para las tres clases resulta ser de 15 árboles, ya que es el que mejor precisión presenta en conjunto. La clase trueno presenta un 95%, la clase lluvia un 100% y la clase de ruido ambiente un 100%.

Llegados a este punto, ya tenemos los dos parámetros importantes para el algoritmo Random Forest cuyos valores se presentan en la tabla 5.6.

Tabla 5. 6.- Parámetros importantes para el algoritmo Random Forest.

Número de árboles	**Máximo número de ramas (MaxNumSplits)**
15	10

En la tabla 5.7, se encuentran los porcentajes de acierto, de fallo y de falsa alarma para el algoritmo Random Forest con 15 árboles, atendiendo a la ecuación 5.1, obtenidos para el escenario software.

Tabla 5. 7.- Porcentajes de acierto, fallo y falsa alarma para Random Forest con 15 árboles – Escenario Software.

	PA	PF	PFA
Truenos	$95\% = \frac{19}{20}x100$	$5\% = \frac{1}{20}x100$	$0\% = \frac{0}{60}x100$
Lluvia	$100\% = \frac{20}{20}x100$	$0\% = \frac{0}{20}x100$	$0\% = \frac{0}{60}x100$
Sonido ambiente	$100\% = \frac{20}{20}x100$	$0\% = \frac{0}{20}x100$	$1{,}67\% = \frac{1}{60}x100$

Referente a esta misma tabla (5.7), se ha comprobado que el falso negativo de la clase truenos, es confundido con la clase sonido ambiente.
El siguiente parámetro que se ha valorado para ver la calidad del algoritmo Random Forest es el tiempo que tarda en analizar el archivo de audio de pruebas, en función del número de árboles (Figura 5.5).

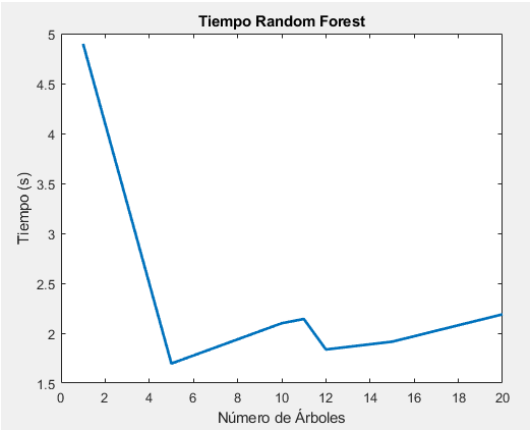


Figura 5. 5.- Tiempo en función del número de árboles para Random Forest – Escenario Software.

En este caso, para 15 árboles tarda un tiempo de 1,92 segundos en analizar el escenario sonoro que hemos usado para la simulación de software. Tarda más tiempo que el anterior algoritmo en realizar las predicciones en el archivo de audio completo ya que una vez el árbol está dibujado en la fase de entrenamiento, la fase de test consiste en crear un camino desde el nodo raíz hasta una hoja y obtener un histograma con las clases ganadoras como se explicó en el capítulo 2.

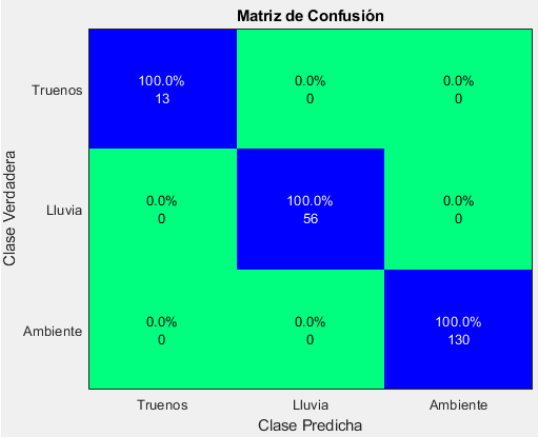


Figura 5. 6.- Matriz de confusión para 11 árboles con Random Forest.

Por último, cuando ya tenemos todos los parámetros establecidos, obtenemos la matriz de confusión (representada en la figura 5.6), nos indica que coloca todos los archivos de test en cada una de las clases correspondientes, lo cual realiza su tarea correctamente.

5.3.2 Escenario exterior en recinto cerrado

Estos resultados en vivo se han conseguido en una sala que absorbe bastante las reflexiones como se puede observar en la figura 5.7, además de con un nivel sonoro considerable emitido por la fuente y en absoluto silencio.

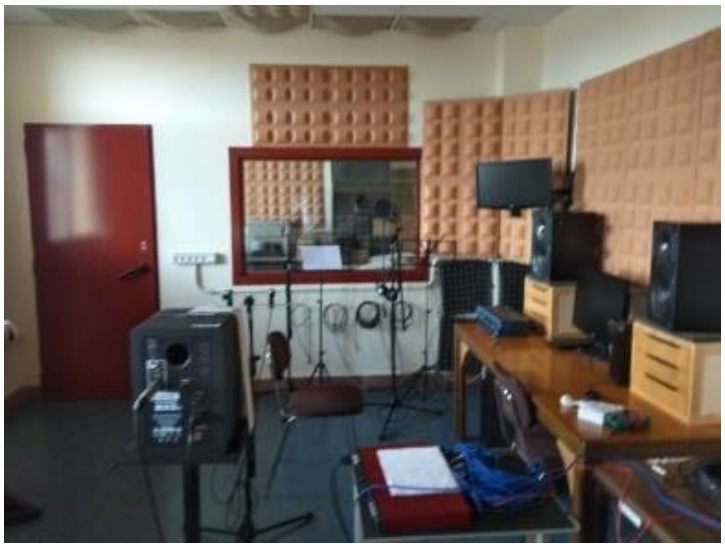


Figura 5. 7 .- Sala en la que se han realizado las pruebas exteriores.

Con los parámetros ya establecidos en el escenario anterior, para k – Nearest Neighbor obtenemos los siguientes porcentajes de acierto, fallo y falsa alarma para el escenario exterior en recinto cerrado, atendiendo a la ecuación 5.1. Se muestran en la tabla 5.8.

Referente a la tabla 5.8, se ha podido comprobar que los dos falsos negativos obtenidos en la clase trueno se han confundido con la clase sonido ambiente. Lo cual se relaciona en cierta manera con los resultados obtenidos con este mismo algoritmo con el escenario en Software.

Tabla 5. 8.- Porcentajes de acierto, fallo y falsa alarma para KNN con k=2 – Escenario en Exteriores.

	PA	**PF**	**PFA**
Trueno	$90\% = \frac{18}{20} x100$	$10\% = \frac{2}{20} x100$	$0\% = \frac{0}{60} x100$
Lluvia	$100\% = \frac{20}{20} x100$	$0\% = \frac{0}{20} x100$	$0\% = \frac{0}{60} x100$
Sonido Ambiente	$100\% = \frac{20}{20} x100$	$0\% = \frac{0}{20} x100$	$0{,}3\% = \frac{2}{60} x100$

Seguidamente, obtenemos los resultados para Random Forest, están representados en la tabla 5.9.

Tabla 5. 9.- Porcentaje de acierto, fallo y falsa alarma para RF con 15 árboles – Escenario en Exteriores.

	PA	PF	PFA
Trueno	$100\% = \frac{20}{20}x100$	$0\% = \frac{0}{20}x100$	$0\% = \frac{0}{60}x100$
Lluvia	$100\% = \frac{20}{20}x100$	$0\% = \frac{0}{20}x100$	$13{,}3\% = \frac{8}{60}x100$
Sonido Ambiente	$60\% = \frac{12}{20}x100$	$40\% = \frac{8}{20}x100$	$0\% = \frac{0}{60}x100$

Referente a esta tabla (5.9), se ha podido comprobar que los ocho falsos negativos obtenidos en la clase sonido ambiente se han confundido con la clase lluvia. Lo cual no se relaciona con los resultados obtenidos con este mismo algoritmo en el escenario software, esta es una peculiaridad de Random Forest.

5.4 Resultados en la localización (DOA)

En este apartado, primeramente, se ha realizado un estudio para ver el rendimiento de tres algoritmos, estos son el Periodograma, la Máxima Verosimilitud y MUSIC, analizando parámetros como el error y la desviación estándar del error o el tiempo de cómputo en función de la relación señal a ruido y de la dirección aunque no los hemos integrado en el sistema ni hemos hecho pruebas reales con ellos en escenario software o exteriores. Las pruebas que se presentan son únicamente teóricas, pues el objetivo era estudiar su comportamiento antes de decidirnos cual era el algoritmo más adecuado para nuestro sistema.
Finalmente, hemos elegido el algoritmo TDOA (Time Delay) por su sencillez a la hora de implementarlo en nuestro código, además de por su precisión y rapidez, a pesar de que MUSIC es considerado, por muchos investigadores, el mejor algoritmo DOA [39].

Estas pruebas de localizar la dirección de procedencia sólo se llevan a cabo con sonidos de truenos debido a que con el resto de las clases no tiene sentido ya que no vamos a localizar la dirección de procedencia de la lluvia o del sonido ambiente.
Para implementar el TDOA, se ha requerido de un estudio inicial, siguiendo el mismo procedimiento que en las funcionalidades anteriores de nuestro sistema, en lo referente al uso de un escenario de simulación software antes de pasar a un escenario en exteriores. Esto nos ha permitido estudiar las características y los parámetros de control de cada uno de los algoritmos, haciendo más eficiente su uso práctico.
Para proceder, primero simulamos con archivos de sonido de dos canales y de distintos retardos de creación propia y comprobamos que el ángulo localizado se aproxima al ángulo de emisión. Ajustamos dos parámetros importantes; la longitud del buffer (bufferLength) y la longitud de los frames de audio (AudioFrameLength).
Luego verificamos, con los mismos parámetros, en el laboratorio que usando un array de sólo dos micrófonos separados una distancia horizontal, y emitiendo truenos por el altavoz, el sistema también funciona correctamente.
Por último, cuando el sistema funciona bien con dos canales o dos micrófonos tanto en escenario software como en escenario exterior, pasamos a crear los archivos de tres canales y probamos el sistema de tres micrófonos y tres arrays, en principio simulando

sólo en Matlab y ajustando los mismos parámetros que en el caso anterior. Cuando funcionó volvimos a pasar nuevamente al laboratorio, ahora ya con los tres micrófonos y nuevamente altavoz desde distintas direcciones emitiendo truenos.

5.4.1 Resultados teóricos

Como comentamos previamente en la introducción de este apartado, hemos analizado varios algoritmos DOA, con los cuales no hemos llegado a realizar pruebas reales, ya que no los hemos integrado en la aplicación. Para ello, se ha realizado un proceso iterativo con el objetivo de obtener los valores de la precisión y tiempo de cómputo del algoritmo, esta tarea se realiza simulando dos sensores y barriendo varios valores de SNR y direcciones DOA que repetimos 100 veces.

RESULTADOS PARA EL ALGORITMO DEL PERIODOGRAMA

La gráfica superior de la figura 5.8 nos muestra el error absoluto promedio en grados. La mejor zona, en la que no existe error, se encuentra entre -80 ° y 80 ° con aproximadamente 15 dB de relación señal a ruido, además la desviación del valor absoluto (gráfica inferior) en esta zona es de 0°, entonces al no haber error en esta zona, el ángulo localizado será el ángulo al que se emite sonido. Si se permite un error de 1°o 2°, con una relación señal a ruido de 10 dB es posible cubrir direcciones que van de -70 ° a 70°, en este caso la desviación como mucho entre estimaciones consecutivas será de ±1°, esto último se manifiesta siempre para direcciones en el extremo del rango, en torno a los -70° o 70°. Si admitimos de 3° a 4° de error con una relación señal a ruido de 5 dB sería posible abarcar las direcciones que van de -70° a 70°.
Por debajo de esa zona el error será superior a los 5°.

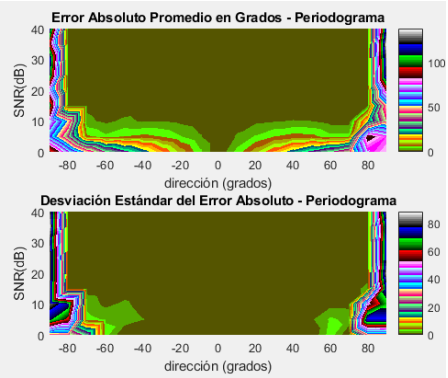


Figura 5. 8.- Error Absoluto (superior) y Desviación Estándar (inferior) del método Periodograma.

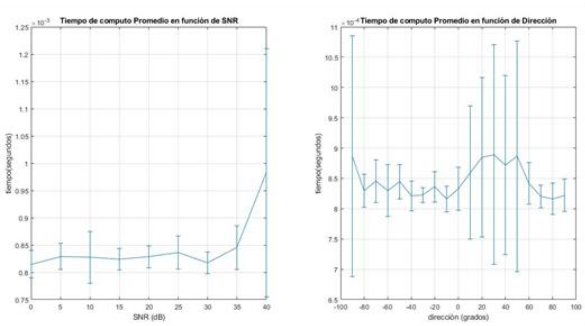


Figura 5. 9.- Tiempo de cómputo promedio en función de SNR (izquierda) y de la Dirección(derecha) – Periodograma.

Observamos en la figura 5.9 que dependiendo del valor de la relación señal a ruido o del ángulo captado, el algoritmo irá más rápido o más lento. La gráfica de la izquierda nos muestra el tiempo de cómputo promedio junto a su desviación estándar, en función de la Relación Señal a Ruido. De aquí podemos deducir que para 30 dB de SNR, es para lo que el algoritmo funciona más rápido (0,82 ms) y funciona más lento a partir de 35 dB (0,84 ms).
La gráfica de la derecha indica el tiempo de cómputo promedio en función de la dirección. El algoritmo funciona más rápido para -10˚(0,81ms) y más lento para 30˚ (0,88ms). En las dos gráficas el tiempo de cómputo promedio ronda los 0,82ms.

RESULTADOS PARA EL ALGORITMO DE LA MÁXIMA VEROSIMILITUD

La gráfica superior de la figura 5.10 nos muestra el error absoluto promedio en grados. La mejor zona, en la que no existe error y con 0˚ de desviación, se encuentra entre -80 y 80 grados con aproximadamente 30 dB de relación señal a ruido. Por debajo de esa zona existe error. Observamos que, de 30 dB hacia abajo en la zona de -70˚ a 70˚existe más error en los ángulos positivos(0˚ a 80˚) que en los ángulos negativos(-80˚ a 0˚).
La gráfica inferior nos muestra la desviación estándar del error absoluto. Es decir, lo que puede variar de forma positiva o negativa el error promedio y observamos que cuando se calculan ángulos positivos varía más el error que en el caso de los negativos.

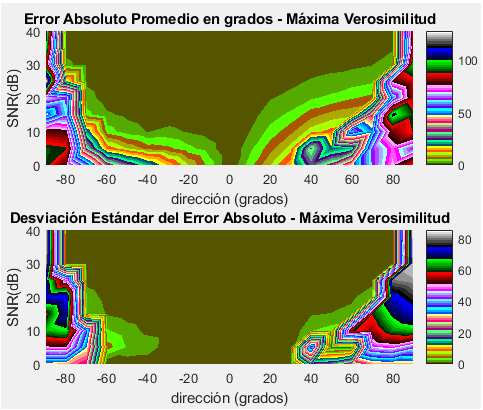


Figura 5. 10.- Error Absoluto (superior) y Desviación Estándar (inferior) del método Máxima Verosimilitud.

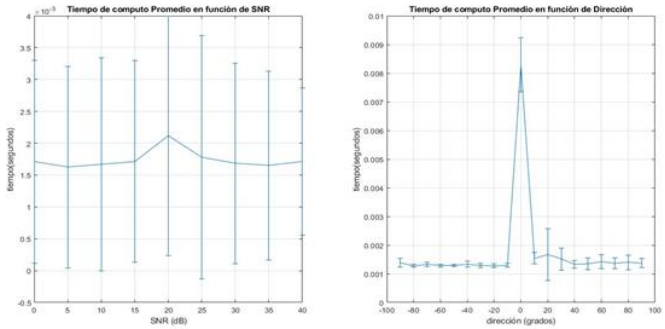


Figura 5. 11.- Tiempo de cómputo promedio en función de SNR (izquierda) y de la Dirección(derecha) – Máx. Ver.

A la vista de la figura 5.11 y su tendencia, tanto en la gráfica de la derecha como la de la izquierda, suponemos que estos picos son sobrecargas del sistema, ya que en condiciones normales, no tiene sentido que existan esas variaciones empleando este algoritmo.

RESULTADOS PARA EL ALGORITMO MUSIC(Multiple Signal Classification)

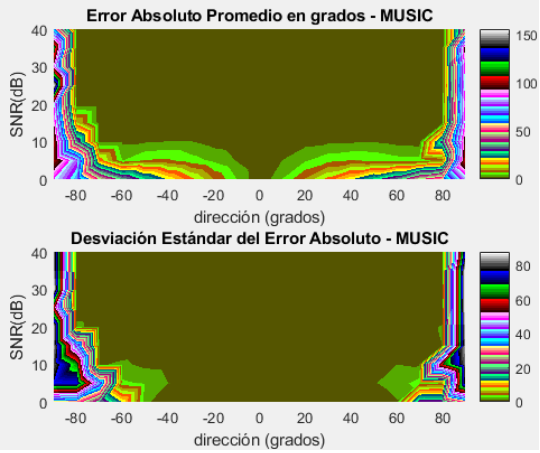


Figura 5. 12.- Error Absoluto (superior) y Desviación Estándar (inferior) del método MUSIC.

La gráfica superior de la figura 5.12 nos muestra el error absoluto promedio en grados. Existe una zona en la que no se produce error, es entre -80˚ y 80˚ con 20 dB de relación señal a ruido. Además, la desviación del valor estimado (gráfica inferior) en esta zona es de 0dB. Si se permite un error de 1˚o 2˚, con una relación señal a ruido de 10 dB es posible cubrir direcciones que van de -80˚ a 80˚, en este caso la desviación entre estimaciones consecutivas será de ±1˚, esto último se manifiesta siempre para direcciones en el extremo del rango. Si admitimos de 3˚ a 4˚ de error con una relación señal a ruido de 5 dB sería posible abarcar las direcciones que van de -70˚a 70˚.

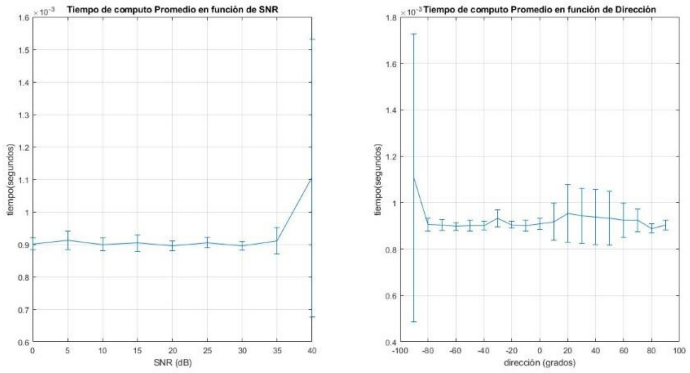


Figura 5. 13.- Tiempo de cómputo promedio en función de SNR (izquierda) y de la Dirección(derecha) – MUSIC.

La gráfica de la izquierda de la figura 5.13 nos muestra el tiempo de cómputo promedio en función de la Relación Señal a Ruido. De aquí podemos deducir que para 10,20 o 30 dB es el nivel para el cual el algoritmo funciona más rápido con un tiempo de procesado de 0,9ms y el algoritmo funciona más lento para 40 dB con un tiempo de 1,1 ms.
La gráfica de la derecha muestra el tiempo de cómputo promedio en función de la dirección. Este tiempo también ronda los 0,9ms. El algoritmo funciona más rápido en 80˚(0,87ms) y funciona más lento en -90˚ (1,1ms). En las dos gráficas el tiempo de cómputo ronda los 0,9 ms.
En resumen, existen dos algoritmos que prácticamente nos dan los mismos resultados de los tres analizados, presentando ligeramente mejores resultados el Periodograma, para la estimación de la dirección de llegada. El método del Periodograma tiene el límite

de error a menor relación señal a ruido (15 dB) frente a los 20 dB del MUSIC. Aunque MUSIC en 10 dB cubre más rango. En cuanto a la desviación del error, es menor en el Periodograma y en cuanto al tiempo de cómputo, es mejor en el Periodograma, con un tiempo de cómputo medio de 0,82ms frente a los 0,9ms de MUSIC.
Sin embargo, MUSIC es más utilizado debido a que tiene mayor resolución espacial y es más resistente al ruido [39].

5.4.2 Escenario software

RESULTADOS PARA EL ALGORITMO TIME DELAY

Como se comentó en la introducción de este apartado, el escenario sonoro empleado para realizar estas pruebas sólo contiene sonidos de truenos y el número de canales varía si se quieren simular dos micrófonos, donde tendremos dos canales o si son tres, tendremos tres canales. Estas pruebas se realizaron simulando varias direcciones de llegada o lo que es lo mismo, creando archivos de audio de truenos con distintos retardos entre canales.
Los parámetros que finalmente hemos elegido están reflejados en la tabla 5.10. Esto es, una longitud del frame de 75.000 y un tamaño del buffer de 7.500. Se han realizado varias pruebas y hemos comprobado que cuanto más grande sea el frame, en conjunto con el buffer, se presenta mejor precisión. También hemos tenido en cuenta que el tamaño del buffer sea divisible entre la longitud del frame.

Tabla 5. 10 .-Parámetros importantes para el cálculo de la DOA.

Duración del Frame de Audio (AudioFrameLength)	**Tamaño del buffer (bufferLength)**
75.000 muestras	7.500

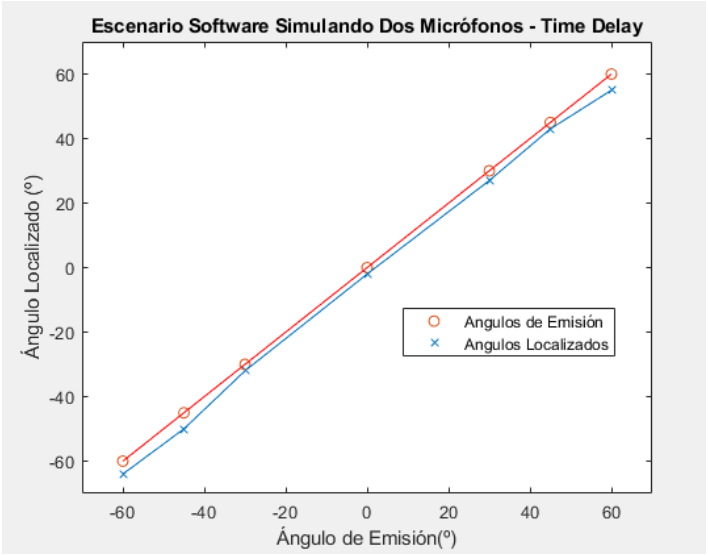


Figura 5. 14.- Precisión en escenario software con dos micrófonos para el Algoritmo Time Delay.

Simulando con dos micrófonos sólo podemos detectar de direcciones de -60 a 60 grados y hemos observado la precisión que presenta el sistema, comparando los ángulos que localiza el sistema junto con los ángulos reales, como se muestra en la figura 5.14.

Observamos que, con dos micrófonos, los ángulos detectados están por debajo de los ángulos reales con un error de 5 grados como máximo (tabla 5.11).

Tabla 5. 11.- Precisión simulada con dos micrófonos para el Algoritmo Time Delay.

Ángulo de emisión (˚)	Ángulo Localizado (˚)	|Error| (˚)
-60	-64	4
-45	-50	5
-30	-32	2
0	-2	2
30	27	3
45	43	2
60	55	5

Simulando con tres micrófonos podemos detectar sonidos en los 360 grados de la circunferencia y hemos calculado su precisión, comparando los ángulos que localiza el sistema junto con los ángulos localizados, como se muestra en la figura 5.15.

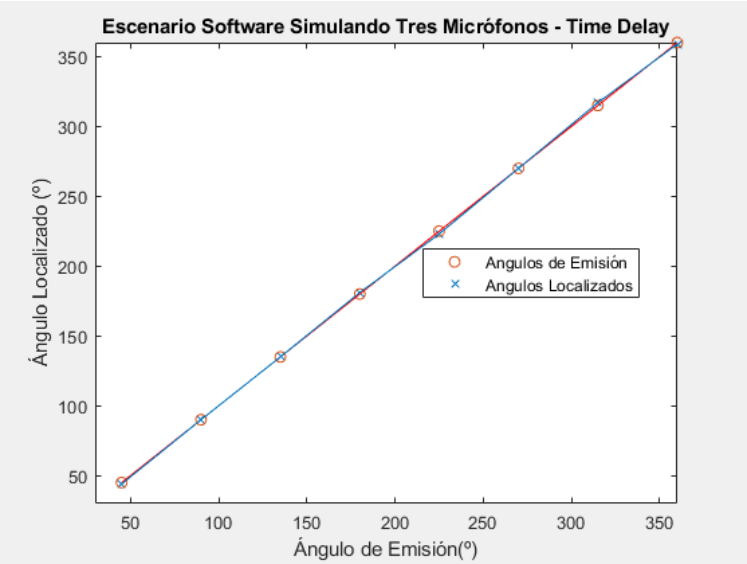


Figura 5. 15 .-Precisión en escenario software con tres micrófonos para el Algoritmo Time Delay.

Observamos que existe más precisión que con dos micrófonos ya que el error se reduce, las dos líneas casi coinciden. En este caso tenemos un error máximo de 2 grados (Tabla 5.12).

Tabla 5. 12.- Precisión simulada con tres micrófonos para el Algoritmo Time Delay.

Ángulo de emisión (˚)	Ángulo Localizado (˚)	|Error| (˚)
360	359	1
315	317	2
270	270	0
225	223	2
180	181	1
135	135	0
90	90	0
45	44	2

5.4.3 Escenario exterior en recinto cerrado

RESULTADOS PARA EL ALGORITMO TIME DELAY

Para ver el comportamiento del sistema con dos micrófonos en vivo se han realizado las pruebas en el laboratorio de electroacústica, emitiendo sonidos mediante una caja acústica autoamplificada desde los ángulos que se quiere comprobar la dirección de procedencia. Para ver los resultados, se ha representado en una gráfica en la figura 5.16, la cual contiene una línea con los ángulos de emisión y otra con los ángulos localizados junto a su desviación estándar.

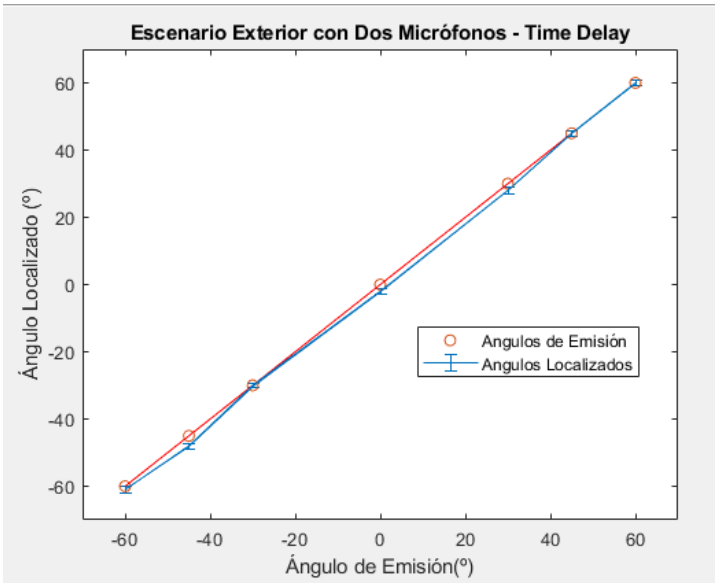


Figura 5. 16.- Precisión en escenario exterior con dos micrófonos para el Algoritmo Time Delay.

Observamos que existe más precisión con dos micrófonos en escenario exterior que en escenario software, la línea está más unida. Además, existe un error de 3 grados como máximo y una desviación estándar de aproximadamente 1 (tabla 5.13).

Tabla 5. 13.- Precisión en escenario exterior con dos micrófonos para el Algoritmo Time Delay.

Ángulo de emisión (˚)	Ángulo Localizado (˚)	|Error| (˚)	Desviación estándar
-60	-61	1	0,91
-45	-48	3	0,85
-30	-30	0	0,77
0	-2	2	0,75
30	28	2	0,86
45	45	0	0,88
60	60	0	0,89

Por último, se han realizado las pruebas para tres micrófonos en escenario exterior con el algoritmo TDOA, las cuales están representadas sobre una gráfica en la figura 5.17.

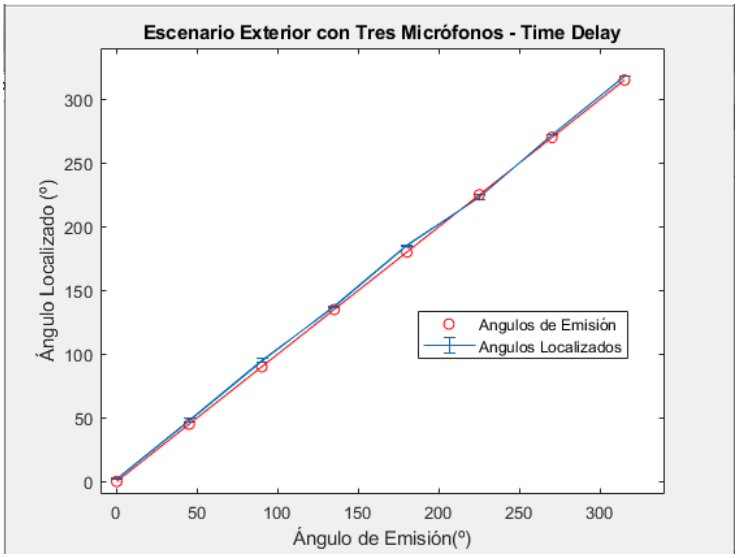


Figura 5. 17.- Precisión en escenario exterior con tres micrófonos para el Algoritmo Time Delay.

Observamos que el peor caso se da cuando se quieren detectar fuentes localizadas en 90 y 180 grados, donde existe un error de 5 grados, además en 180 grados existe una desviación estándar de 2. (Tabla 5.14)

Tabla 5. 14.- Precisión en escenario exterior con tres micrófonos para el Algoritmo Time Delay.

Ángulo de emisión (˚)	**Ángulo Localizado (˚)**	**|Error| (˚)**	**Desviación estándar**
315	318	3	1,5
270	272	2	0,8
225	223	2	0,65
180	185	5	2
135	137	2	0,29
90	95	5	0
45	48	3	0,39
0	2	2	1,2

CAPITULO 6: IMPLEMENTACIÓN DE LA INTERFAZ DE USUARIO

[Esta página ha sido dejada intencionalmente en blanco]

6.1 Introducción

En este capítulo, se habla sobre la integración de las tres técnicas en el sistema junto con los algoritmos que se han utilizado en cada una de ellas.
También hablaremos sobre las variables que lo controlan y la tipología de usuario. Después, sobre cómo se ha diseñado la interfaz de usuario. Y, por último, sobre los accesos a la interfaz, entre los que se encuentran el básico y el profesional.

6.2 Integración del sistema y algoritmos específicos

Teniendo los resultados de cada una de las técnicas que realiza nuestro sistema, mostrados en el capítulo anterior y habiendo ajustado los parámetros necesarios, podemos unir todos los algoritmos para integrarlos en nuestro sistema.
El sistema se ha integrado atendiendo al diagrama de bloques representado en la figura 6.1 y estableciendo los parámetros que se comentaron en el capítulo anterior.

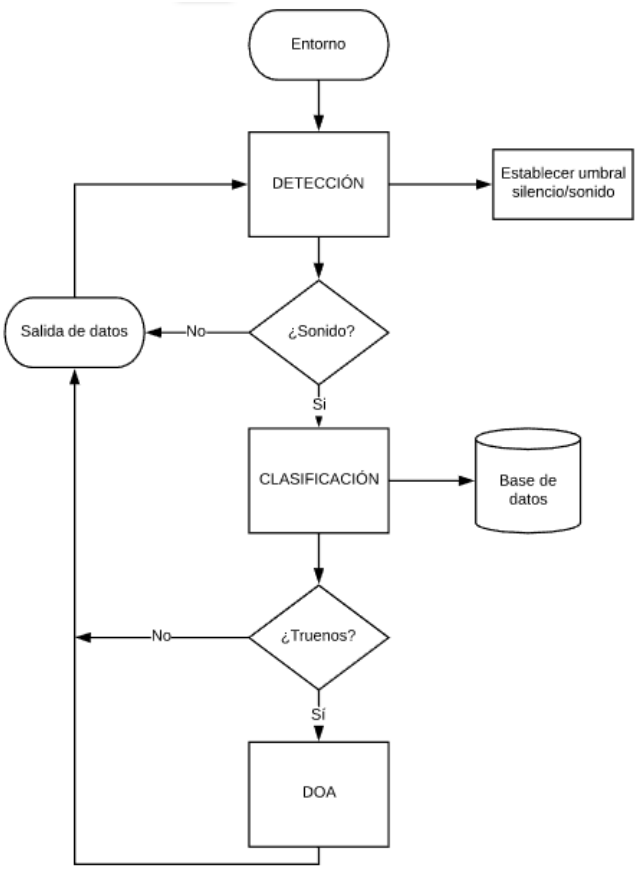


Figura 6. 1.- Diagrama de bloques.

1. El diagrama de bloques comienza en el entorno en el que se sitúa el sistema, es decir, en la sala desde donde ejecutamos la aplicación o desde el software.
2. Esperamos a que el sistema calcule el umbral utilizando el algoritmo de detección basada en umbrales, para ello, si se encuentra en exteriores “escucha” el ruido de fondo

del entorno donde se encuentra, así podrá distinguir entre el ruido de fondo, el cual considerará silencio, del sonido emitido por la fuente. En el caso del escenario Software, esta operación se realizará a nivel de software, realizando el cálculo del umbral sobre los silencios del archivo.

3. Cuando se haya calculado el umbral, el sistema mostrará en pantalla si detecta silencio o sonido. Detectará silencio cuando el nivel sonoro sea menor o igual al umbral calculado y detectará sonido cuando suceda lo contrario.

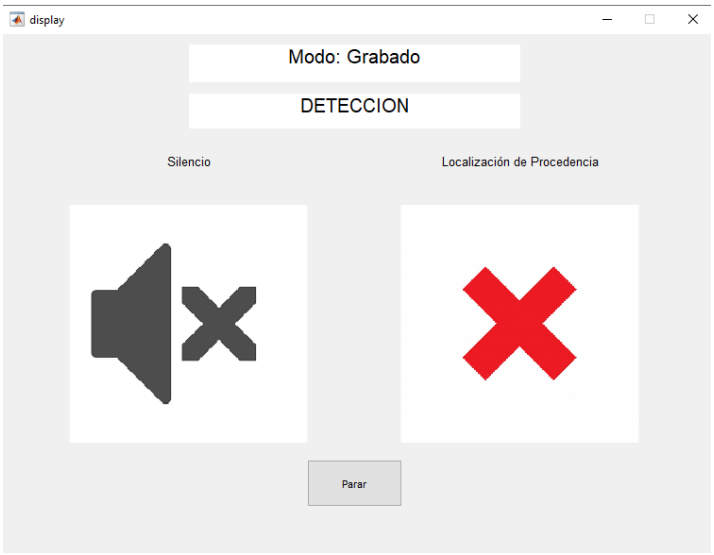


Figura 6. 2.- Detección de silencio.

4. Cuando detecte sonidos, los clasificará atendiendo a tres clases (truenos, lluvia y sonido ambiente) mostrando la clasificación por pantalla (Figura 6.3), según las plantillas de entrenamiento que se encuentran en la base de datos. Para clasificarlos podremos elegir el algoritmo KNN (K-Nearest Neighbor) o Random Forest.

Figura 6. 3.- Sonido ambiente en la clasificación.

5. Sólo cuando se detecten truenos, se mostrará por pantalla la dirección de procedencia detectada (figura 6.4). Para ello, usa el algoritmo TDOA (Time Delay). En las clases donde no se calcula la dirección de procedencia, aparecerá la imagen con la equis roja, como hemos visto en las figuras 6.2 y 6.3.

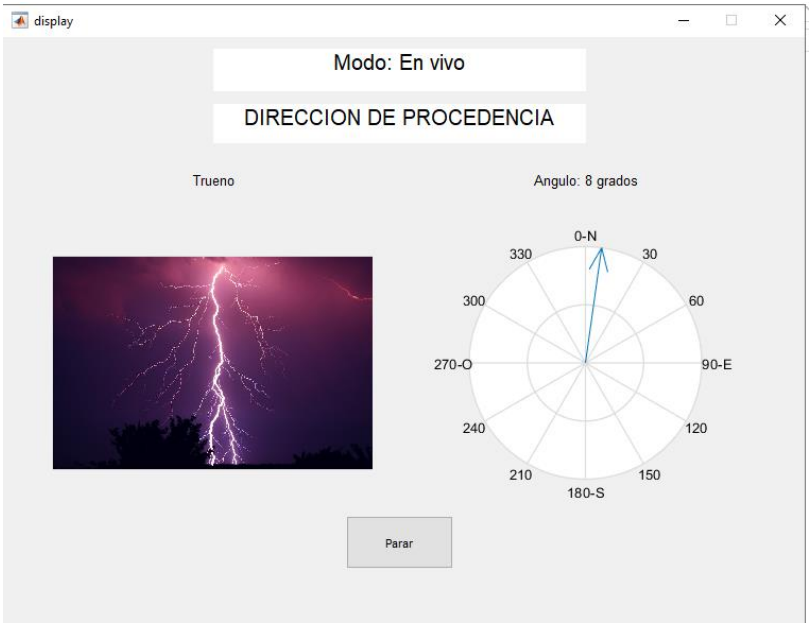


Figura 6. 4.- Localización de la dirección de procedencia.

6. Al final, cuando el sistema haya acabado, por distintas razones: porque le hayamos introducido un tiempo finito, porque hayamos introducido un archivo de audio que ha finalizado de procesarse, o bien porque hemos pulsado el botón 'Parar', se generará automáticamente un archivo txt con los resultados obtenidos, cuyo nombre será diferente si el sistema se prueba con señal grabada o con señal microfónica. El archivo con los resultados se creará en la carpeta 'Resultados', siguiendo el modelo de la tabla 6.1:

Tabla 6. 1.- Tabla con los resultados.

Detecciones	**Clases**	**Ángulo (°)**	**Fecha**	**Hora**
Silencio	----	NaN	10 – May – 2019	11:06:14
Sonido	Sonido Ambiente	NaN	10 – May – 2019	11:06:15
Silencio	----	NaN	10 – May – 2019	11:06:16
Sonido	Truenos	45	10 – May – 2019	11:06:17

Como observamos en la tabla 6.1, en las detecciones se imprimirá si es silencio o sonido. Si es sonido, escribirá la clase detectada, esto es, sonido ambiente, lluvia o truenos.
Y si es silencio, mostrará un espacio vacío
Solamente cuando sea truenos, imprimirá su dirección de procedencia en ángulos, de lo contrario imprimirá NaN (Not a Number).
En la siguiente columna, anotará la fecha, en el formato; día con números, mes con letras con y año con números de cuatro cifras y en la última columna la hora, en el formato horas, minutos y segundos, todas con dos dígitos, de los eventos que han sido detectados.

6.3 Variables que controlan el sistema y la tipología del usuario

Los botones y variables de la interfaz, que a su vez controlarán el sistema según el acceso que el usuario haya elegido, básico o profesional, se enumerarán a continuación:

Al ejecutar el script SISEVEMETEO.m accedemos a la interfaz inicial, llamada SISEVEMETEO, tendremos dos botones de acceso para poner en marcha el sistema, el acceso profesional y el acceso básico. La imagen de la interfaz SISEVEMETEO la veremos dentro de dos apartados en la figura 6.6.
Además, en la parte superior izquierda de la interfaz habrá un botón de instrucciones, donde se explica al usuario los pasos que deberá seguir antes de entrar en alguno de los dos accesos y acompañando a este botón por su lado derecho se encuentra un botón de ayuda, donde se explica cada uno de los dos accesos.

Se complementará esta información de los accesos con los apartados 6.5.1 y 6.5.2 de este capítulo.
Dentro de los botones y variables comunes a los dos accesos nos encontramos:
· Un botón llamado 'Empezar' para que el sistema entre en funcionamiento.
· Un botón llamado 'Atrás' que nos dirige a la interfaz SISEVEMETEO, donde se encontrarán los accesos a la interfaz.
· Un grupo de botones para elegir el algoritmo de clasificación (KNN o Random Forest).
· Cuando comenzamos en cualquiera de los dos accesos, el sistema lanzará otra interfaz llamada 'display', ésta contiene un botón para parar el sistema en cualquier momento, siempre y cuando no aparezca "Espere…" en pantalla.
En esta interfaz aparece el modo en el que estamos trabajando y justo debajo, la tarea que se está realizando en ese momento. Sólo habrá dos modos para mostrar, en vivo o grabado. El acceso DEMO o Básico funciona con el modo grabado.
Cuando aparezca "Espere…" (Figura 6.5) en la interfaz 'display', el programa se estará iniciando y no se podrá parar.

Figura 6. 5.- Mensaje "Espere…" en el display.

Además, la interfaz 'display' mostrará en pantalla la información al usuario en cada momento, dependiendo de las tareas que esté realizando el sistema.
Cada interfaz de la aplicación (excepto la que muestra la información al usuario, llamada 'display'), tendrá su botón de ayuda en la parte superior izquierda donde habrá información sobre el modo de funcionamiento, de los algoritmos de clasificación y de los botones exteriores.

En el acceso profesional, cuya imagen veremos posteriormente en la figura 6.8, nos encontramos:
· Un grupo de botones para seleccionar el modo de funcionamiento (En vivo o Grabado), cada uno con su papel correspondiente.
· En el modo en vivo, tenemos que introducir el tiempo. O bien, seleccionando Infinito, donde la aplicación funcionará con un tiempo infinito o bien, seleccionando Tiempo. En ese caso se nos habilitará el texto editable para introducir manualmente el tiempo en segundos, no importa el tiempo que introduzcamos ya que el sistema se podrá parar en cualquier momento, siempre y cuando no aparezca "Espere…" en la interfaz 'display'.
· Además en el modo en vivo de este acceso, habrá un botón llamado 'Calibrar' para realizar la calibración de refuerzo del sistema.
· Si hemos elegido el modo grabado, aparte de elegir el algoritmo de clasificación, tendremos que cargar un archivo de audio pulsando el botón 'Cargar'. Cuando carguemos el archivo observaremos su información en el panel del modo grabado antes de empezar. El tiempo que dura el sistema procesando el archivo, es aproximadamente la duración del archivo de audio que hayamos cargado.
· Cuando se selecciona el modo en vivo, se ocultan todas las funcionalidades del modo grabado y viceversa.

Por otra parte, tenemos el acceso básico, cuya imagen veremos posteriormente en la figura 6.7, al cual se ha denominado DEMO. Su función es que el usuario tenga una demostración de cómo funciona el sistema.
En este acceso tenemos:
· Un menú desplegable para seleccionar un archivo que queramos cargar de los dos disponibles y también su información antes de empezar.

6.4 Diseño de la Interfaz de Usuario

Para el desarrollo de la parte gráfica se hizo uso del entorno de programación visual GUIDE, además se consultó la documentación de MATLAB y algunos tutoriales en línea. Es muy importante que, al lanzar la interfaz principal, el usuario se encuentre dentro de la carpeta donde está ubicado el script llamado SISEVEMETEO.m. Al ejecutar este script, se añadirán al patch todas las subcarpetas que se encuentran junto al SISEVEMETEO.m y que a su vez contienen los archivos de funcionamiento del sistema.

6.5 Accesos a la interfaz

Esta interfaz aparece al ejecutar SISEVEMETEO.m. Como se comentó al principio, tendremos dos accesos al sistema: el básico, también llamado DEMO y el profesional.

En la figura 6.6, tenemos la interfaz inicial, la cual tiene escrito el nombre del sistema, SISEVEMETEO, el cual proviene de Sistema de Eventos Meteorológicos.

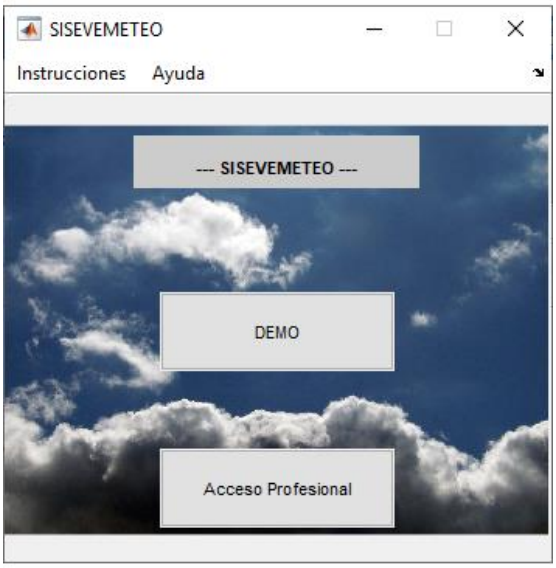


Figura 6. 6.- Accesos a la interfaz.

6.5.1 Acceso Básico

Este acceso, al cual se ha denominado DEMO sería el adecuado para que lo usara un usuario que quiera saber de manera rápida cómo funciona el sistema, porque no tenemos que cargar un archivo como en el modo Grabado del acceso profesional, sino que existen dos archivos ya cargados. Cuando pulsemos sobre el botón 'Empezar' podremos ver la información que se muestra en pantalla observando las tareas que se realizan en cada momento o escuchar los sonidos que clasifica el sistema, ya que se escuchan a la vez que se clasifican, antes de probar el acceso profesional, su interfaz se muestra en la figura 6.7:

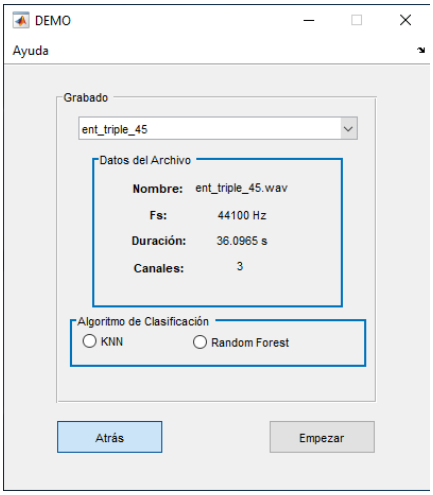


Figura 6. 7.- Acceso básico (DEMO).

Como vemos, al principio existe un desplegable para seleccionar uno de los dos archivos de audio posibles que queramos elegir, cuando hayamos elegido el archivo, saldrán sus datos, en concreto, el nombre, la frecuencia de muestreo, la duración en segundos y el número de canales.

Por último, debemos que elegir un algoritmo de clasificación.
Al pulsar sobre el botón 'Empezar' nos conducirá a la interfaz 'display', donde veremos la información proporcionada por el sistema. Si se nos olvida elegir algo de lo anteriormente comentado, se nos mostrará una advertencia en forma de cuadro de diálogo, para que elijamos lo que nos falta antes de empezar.

6.5.2 Acceso Profesional

El acceso profesional (Figura 6.8) sería el adecuado para usuarios que necesiten comprobar el funcionamiento de otros parámetros en el sistema, así como revisar su funcionamiento. Este acceso tiene dos modos de funcionamiento, En vivo, donde el sistema recoge señal de audio de los micrófonos y Grabado, donde el sistema recoge señal de audio de archivos que se cargan. En el modo 'En Vivo' se puede realizar una calibración de refuerzo al sistema, para cuando se vaya a usar en exteriores y de esta manera, exista más precisión en el cálculo de la Localización de la Dirección de Procedencia. Esta tarea, junto con la calibración normal, debería realizarse periódicamente y cada vez que se lanza la aplicación en un nuevo recinto.
La otra utilidad que tiene este acceso es en el modo grabado, donde se puede cargar cualquier archivo de audio para procesarlo.

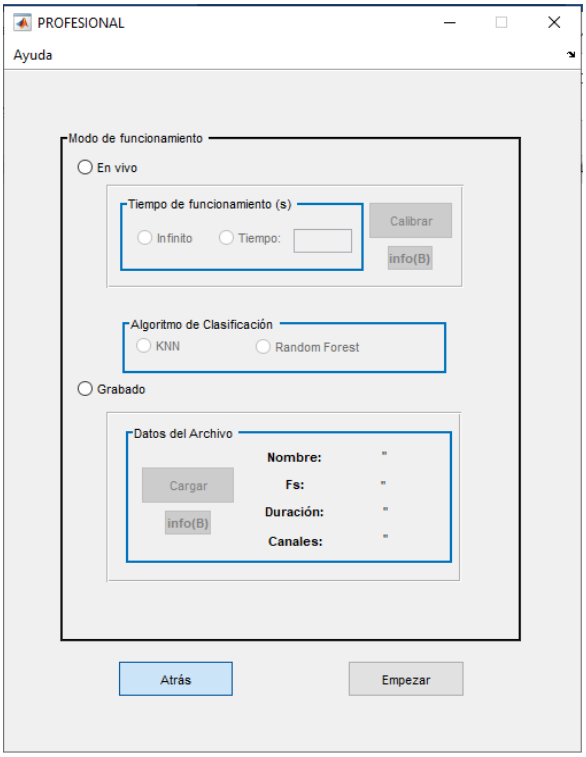


Figura 6. 8.- Acceso profesional.

Como vemos, existen dos modos para elegir, en vivo o grabado.
Se ha programado la aplicación para que cuando se seleccione un modo, se oculten todas las funcionalidades del otro, es decir, se nos active su panel y se oculte el panel del otro modo.

Si seleccionamos el modo En vivo, tendremos que introducir el tiempo. O bien, seleccionando infinito, donde la aplicación funcionará por un tiempo infinito o bien, seleccionando 'Tiempo' donde introduciremos manualmente el tiempo en segundos en el texto editable, no importa el tiempo que introduzcamos ya que se podrá parar en cualquier momento, siempre y cuando no aparezca "Espere…" en la interfaz 'display'.
Es importante que después de pulsar sobre el botón 'Empezar' en este modo, para que el sistema funcione en exteriores, debemos esperar un tiempo sin emitir sonidos (aproximadamente 6 segundos), para que el cálculo del umbral no se vea alterado. Entonces, el sistema "escuchará" el ruido de fondo de la sala, calculará el umbral y sabrá distinguir entre el silencio el sonido.
Además, el modo En vivo tendrá un botón llamado 'Calibrar', al pulsarlo se realiza una calibración de refuerzo, generando un archivo mat que contiene tres factores de multiplicación, cada uno de los factores se multiplicará por la amplitud de cada canal de la señal cuando empiece a funcionar el sistema en exteriores, de esta manera los tres canales tienen la misma ganancia y existirá más precisión en la Localización de la Dirección de Procedencia.
Antes de pulsar el botón 'Calibrar', tenemos que asegurarnos de que los micrófonos están calibrados realizando la calibración normal, tal y como se detalló en el apartado de calibración del capítulo 3.
Cuando se pulsa el botón 'Calibrar', aparecerá el question dialog que se muestra en la figura 6.9 y la tarea del usuario es la de volver a calibrar cada micrófono antes de pulsar el botón 'Empezar' para obtener el archivo mat con los tres factores de calibración.

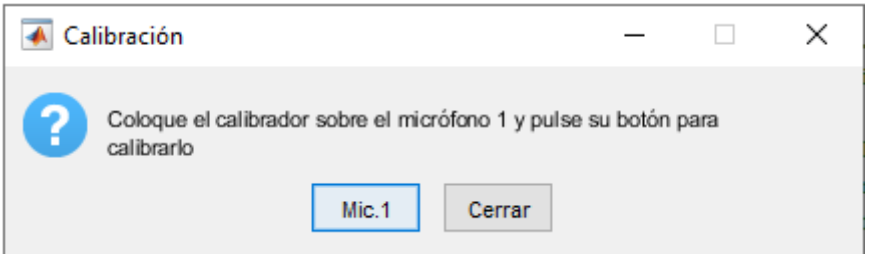


Figura 6. 9.- Question dialog 'Calibración'.

Para calibrar el sistema se colocará el calibrador sobre cada uno de los micrófonos (figura 6.10) y se pulsará el botón del calibrador para emitir un tono 1 kHz con un nivel de presión sonora de 94 dB, cuando esté sonando el tono, pulsaremos sobre el botón del micrófono correspondiente en la aplicación (en la figura 6.9 sobre el botón 'Mic.1'), si pulsamos sobre el otro botón del question dialog, llamado 'Cerrar' nos conducirá al acceso Profesional.
Entonces esperaremos a que el programa lance el mensaje informando que el micrófono está calibrado, realizaremos este proceso hasta calibrar los tres micrófonos. Calibrar el sistema es importante, tanto de la manera normal como la de refuerzo para que el cálculo de la Dirección de Procedencia salga lo más aproximado posible.

Figura 6. 10.- Calibración.

Habrá más información sobre la calibración en el botón complementario ‘info(B)’, donde también se mostrará una imagen de cada micrófono numerado sobre los vértices del triángulo equilátero.

Si hemos elegido el modo Grabado, tenemos que cargar un archivo de audio pulsando el botón cargar para que el sistema calcule la DOA. Este archivo debe contener tres canales. Si no es así, el sistema realizará sus tareas excepto la de calcular la DOA, y cada vez que se haya detectado un trueno, mostrará un mensaje para que el usuario cargue un archivo de tres canales si se quiere realizar. Igual que en el modo DEMO, cuando cargamos un archivo saldrán sus datos, en concreto, el nombre, la frecuencia de muestreo, la duración en segundos y el número de canales.
Este modo es muy parecido al acceso básico o DEMO, la única diferencia es que en el modo grabado se puede introducir cualquier archivo de audio, mientras que el modo DEMO lo hemos limitado solo a dos archivos de audio, para que el usuario tenga una demostración del funcionamiento del sistema.

El botón cargar también tendrá un botón complementario llamado ‘info(B)’ donde se explica al usuario la tarea que realiza.

Después de seleccionar el modo de funcionamiento e introducir las variables de entrada (el tiempo en el modo ‘En vivo’ o el archivo de audio en el modo ‘Grabado’), se habilitará el panel para seleccionar uno de los dos algoritmos de clasificación (KNN o Random Forest).

Al pulsar sobre al botón ‘Empezar’ nos conducirá a la interfaz ‘display’, donde veremos la información proporcionada por el sistema. Si se nos olvida elegir algo de lo

anteriormente comentado, se nos mostrará una advertencia para que elijamos lo que nos falta antes de empezar.

Como nota final, los archivos de audio más adecuados para cargarlos en el sistema se encuentran en la carpeta llamada ‘Grabado’, al igual que , los archivos de audio más adecuados para probar el sistema en exteriores se encuentran en la carpeta llamada ‘En vivo’.

CAPÍTULO 7: CONCLUSIONES

[Esta página ha sido dejada intencionalmente en blanco]

7.1 Conclusiones

La realización de este trabajo me ha permitido, sobre todo, adquirir conocimientos sobre las tres técnicas empleadas para construir el sistema, las cuales usan sus respectivos algoritmos. Fue interesante realizar este proyecto usando tecnologías punteras para, en primer lugar, diferenciar el silencio de los sonidos de interés, y solo actuar cuando exista un sonido determinado, en segundo lugar, la clasificación de eventos sonoros usando herramientas de Machine Learning, una rama de la inteligencia artificial y, en tercer lugar, la localización de la dirección de procedencia (DOA), una herramienta que podemos utilizar en cualquier aplicación en la que queramos localizar la fuente sonora.

También, a medida que hemos realizado este trabajo, hemos podido experimentar los pasos previos que conlleva programar y estudiar cada una de las tareas que realiza nuestro sistema, es decir, el proceso de detección de una señal inmersa en ruido de fondo pasando por el cálculo del umbral, seguidamente, el proceso de clasificación de señales de audio, pasando por la extracción de las características y la obtención de los modelos entrenados y, por último, el proceso de localización de la dirección de procedencia del sonido pasando por el uso de las reglas heurísticas. Además, se han establecido y ajustado los parámetros necesarios para cada una de las funcionalidades anteriores al igual que se han buscado los algoritmos que más se adaptan al sistema para cada una de las técnicas.

En la detección solo se ha probado con un algoritmo, el de detección basada en umbrales, con el cual obtenemos resultados satisfactorios, detectando el 95% de los sonidos en los casos que la señal entra al sistema por archivo de audio y cuando entra por micrófonos, situando correctamente las marcas de inicio y fin. Para ello, fue importante ajustar el tamaño del buffer y la duración de los frames de audio. Hemos considerado sonido cuando las marcas de inicio y fin son mayores a 75.000 muestras de frames de audio y silencio cuando sucede lo contrario.

En la clasificación, hemos concluido que las bases de datos que tienen que estar lo más diferenciadas posibles para que no existan confusiones entre clases, de esta manera, los modelos estarán bien entrenados. Por otro lado, tenemos que contar con buenas características, en nuestro caso los MFCC han demostrado su gran capacidad para aportar información útil, siendo estas las features principales en la clasificación realizada. Por otro lado, las features espectrales como el centroid o el skewness también han demostrado ser útiles para la clasificación.
En esta técnica, destaca el algoritmo K-Nearest Neighbor por su rapidez y fiabilidad, cuyos resultados han sido mejores que los de Random Forest en cada una de las evaluaciones, ya que este último algoritmo cada vez que se entrena, 'dibuja' un bosque diferente y cada vez que se prueba, 'dibuja' un camino diferente desde el nodo raíz hasta la hoja final, por lo tanto es más lento y con resultados más dispersos que el primer algoritmo.

En la Localización de la Dirección de Procedencia, el algoritmo diferencia de tiempos de llegada (TDOA) es con el que obtenemos resultados admisibles, con 5 grados de error como máximo, además de ser el más rápido.

El sistema necesita ser calibrado en el escenario donde sea instalado, al objeto de que las estimaciones DOA sean correctas dentro del orden en que las hemos considerado como tales, para que todos los micrófonos devuelvan el mismo nivel a la salida y, por lo tanto, que no haya variaciones entre niveles sonoros recogidos por los micrófonos.

Cabe destacar que los objetivos del proyecto se han cumplido con éxito, realizando una aplicación versátil y de fácil manejo, sirviéndole al usuario para detectar y clasificar los eventos sonoros y localizar la dirección de procedencia solo en el caso de los truenos en tiempo real, proporcionándole información y ofreciéndole al final un archivo de texto con los resultados obtenidos, cuando se pare el sistema automáticamente o cuando el usuario quiera pararlo.

7.2 Mejoras y líneas de ampliación

Podemos mejorar nuestro sistema en varios aspectos:

- En aspectos de software:

1) En la detección, conseguir que cuando el sistema funcione en modo 'En vivo', el nivel sonoro al que se encuentra el umbral disminuya, y de esta manera se puedan captar sonidos a menor nivel sonoro. Por otro lado, disminuir el tiempo en establecer el umbral de detección en este mismo modo.

2) En la clasificación, mejorar la respuesta del algoritmo Random Forest incorporando nuevas features o características que permitan discriminar mejor entre clases, al tiempo que reducir la redundancia y correlación entre features. Además, siguiendo con esta dinámica, podremos entrenar el sistema para que reconozca más clases.

Asimismo, integrar redes neuronales en el sistema, que es una de las distintas técnicas existentes en el campo de machine learning, para verificar si se obtienen mejores resultados. Este algoritmo imita las redes neurales del cerebro humano, por lo que, si una persona es capaz de distinguir entre distintos tipos de señales de audio utilizando estas redes, cabe pensar que una máquina con una algoritmica similar podría realizar este cometido.

3) En la localización de la dirección de procedencia (DOA), integrar el algoritmo MUSIC o ESPRIT, ya que son los métodos que mejor resolución espacial poseen o lo que es lo mismo, mayor precisión en la localización de fuentes en el espacio. Luego verificar si mejora la estimación de la dirección de llegada en términos de precisión y rapidez.

4) En la aplicación, crear un diseño gráfico más atractivo para el usuario, y añadirle mejoras visuales como, por ejemplo, un visor de la señal que se está procesando en

tiempo real. También, convertir la aplicación en independiente de MATLAB para ejecutarse en cualquier ordenador sin necesidad de tenerlo instalado. Con esto mejoraría la portabilidad del programa al facilitar su uso independientemente de MATLAB.

-En aspectos de hardware:

1) Integrar los elementos del sistema en un solo dispositivo más cómodo o intentar minimizar el número de elementos a conectar, así ocupará menos espacio y, por lo tanto, será más práctico. Sin olvidarnos de incluir una pantalla en la que se muestre la información al usuario en cada momento.

2) Realizar la calibración automática para ahorrarnos revisiones periódicas.

[Esta página ha sido dejada intencionalmente en blanco]

CAPÍTULO 8: BIBLIOGRAFÍA

[Esta página ha sido dejada intencionalmente en blanco]

[1] M. M Al-Maathidi and F. F. Li, “Audio Content Feature Selection and Classification” in *IEEE International Conference on Progress in Informatics and Computing (PIC)*, Nanjing, China, December 2015.
[2] S. Chachada and C.-C. J. Kuo, “Environmental sound recognition: A survey” in *Proc. Asia Pac. Signal Inf. Process. Assoc. Annu. Summit Conf.*, Kaohsiung, Taiwan, pp. 1–9, 2013.
[3] R. Gil-Pita, D. Ayllón, J. Ranilla, C. Llerena-Aguilar and I. Díaz,“A computationally efficient sound environment classifier for hearing aids” in *IEEE Transactions on Biomedical Engineering*, Vol. 62, n. 10, pp. 2358-2368, October 2015.
[4] MathWorks. Audio System Toolbox , consultado el 19 de enero de 2017 en https://es.mathworks.com/help/audio/
[5] I. J. Tashev, *Sound Capture and Processing: Practical Approaches*.Wiley, 2009.
[6] D. Hoiem, Y. Ke, and R. Sukthankar, “SOLAR: Sound object localization and retrieval in complex audio environments”, in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. V, 2005.
[7] T. Heittola, A. Mesaros, T. Virtanen, and A. Eronen, *Sound Event Detection in Multisource Environments Using Source Separation*. CHIME 2011. Septiembre, 2011.
[8] *Acústica. Descripción, medición y evaluación del ruido ambiental*. Parte 1: Magnitudes Básicas y Métodos de Evaluación, Norma UNE-ISO 1996-1:2003.
[9] Geekland, “¿Qué es el rango dinámico de un audio y que ventajas proporciona?”, consultado el 20/05/2019 en https://geekland.eu/que-es-el-rango-dinamico-audio-ventajas/
[10] Yokogawa Test & Measurement Corporation, “Tutorial-Power Meter Calibration”, consultado el 20/05/2019 en https://www.yokogawa.com/ymi/tutorial/tm-tutorial_wt_08.htm
[11] D. Peña-Sánchez, *Estadística Modelos y Métodos*. Alianza Editorial, 2ª edición, Madrid, 1986.
[12] B.D. Rao, and K.S. Arun, “Model based processing of signals: a state space approach”, *Proceedings of the IEEE* , Volume 80, Issue 2, Page(s):283 – 309, Feb. 1992.
[13] J. Capon,"High-resolution frequency-wavenumber spectrum análisis”,*Proceedings of the IEEE,* Volume 57, Issue 8,Page(s):1408 – 1418, August, 1969.
[14] R.T. Lacoss, "Data adaptive Spectral Analysis Methods," *Geophysics,* Volume 36, 661, August 1971.
[15] A.R Schuster, *The periodogram of magnetic declination*, Camb. Phil. Trans. 18 (1900), pp. 107–135.
[16] Czerniawski J., Czyżewski A., Królikowski R., “Neural computation of direction-of-arrival of sound”, Gdansk, POLAND, February 2001.
[17] M. Ferroudj, A. Truskinger, M. Towsey, L. Zhang, J. Zhang and P. Roe, “Detection of Rain in Acoustic Recordings of the Environment”, In: *Pham DN.*, Park SB. (eds) PRICAI 2014: Trends in Artificial Intelligence. PRICAI 2014. Lecture Notes in Computer Science, vol 8862. Springer, Cham
[18] Discovery of Sound in the Sea (2017), “Acoustic Rain Gauge (ARG)”. Consultado el 03/03/2019 en https://dosits.org/galleries/technology-gallery/observing-ocean-currents-and-temperature/acoustic-rain-gauge-arg/
[19] N. van de Giesen, C. Degen and R. Hut. “Affordable Acoustic Disdrometer: Design, Calibration, Tests” in *AGU Fall Meeting Abstracts*, 2009.

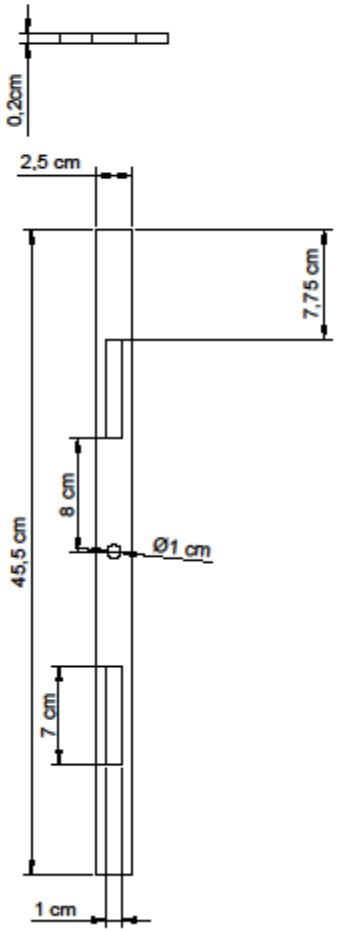
[20] E. Hernández Pérez and J.L. Navarro Mesa, "Nuevos Algoritmos de Localización, Seguimiento e Identificación de Fuentes en Escenarios con Agrupaciones de Sensores Concentradas y Distribuidas", Tesis doctoral, Universidad de Las Palmas de Gran Canaria, España, 2010.
[21] G. Peeters, "A large set of audio features for sound description (similarity and classification) in the CUIDADO project", Technical report, IRCAM, 2004.
[22] S. B. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques", *in Proceedings of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, June 10, 2007, pp.3-24.
[23] D. Mitrovic, M. Zeppelzauer, and C. Breiteneder,"Features for content based audio retrieval*". Advances in Computers*, vol. 78, pp. 71–150, 2010.
[24] R. Troncy, B. Huet, S. Schenk, *Multimedia semantics: metadata, analysis and interaction*, United Kingdom, John Wiley & Sons Ltd, 2011.
[25] D. Jurafsky and J. H. Martin, "Feature Extraction: MFCC vectors" (Sección 9.3) in *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*, Pearson Prentice Hall, Upper Saddle River, N.J., Second edition, 2009.
[26] S. C. Joshi, and A. N. Cheeran, "MATLAB Based Feature Extraction Using Mel Frequency Cepstrum Coefficients for Automatic Speech Recognition", *International Journal of Science, Engineering and Technology Research* (IJSETR), vol. 3, Issue 6, June 2014, pp. 1820-1823.
[27] L. Breiman, J. Friedman, C. J. Stone and R. A. Olshen, *Classification and Regression Trees*, London, United Kingdom, Chapman and Hall/CRC, 1984.
[28] "Decision Trees: An Overview", Aunalytics, 2018 consultado el 25/05/2019 en https://www.aunalytics.com/2015/01/30/decision-trees-an-overview/
[29] A. Criminisi, J. Shotton and E. Konukoglu, "Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning", Microsoft Research technical report TR-2011-114 , 2011.
[30] P. Nikolov, "Machine Learning for Music", April 8, 2015, Faculty of Mathematics and Informatics, SU consultado el 10/05/2019 en https://www.slideshare.net/PetkoNikolov/machine-learning-for-music
[31] X. Liu, M. Song, D. Tao, Z. Liu, L. Zhang, C. Chen, and J. Bu, "Semi-supervised node splitting for random forest construction" in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 492-499, 2013.
[32] Librosa development team, librosa.filters.mel consultado el 20/05/19 en https://librosa.github.io/librosa/generated/librosa.filters.mel.html
[33] G. Arslan and F. Sakarya, "A Unified Neural Network-Based Speaker Localization Technique", *IEEE Trans. on Neural Networks*, Vol. 11, No. 4, pp. 997-1002, July, 2000.
[34] M. Brandstein, "A Pitch-Based Approach to Time-Delay Estimation of Reverberant Speech", *Proc. of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, NY, USA, 1997.
[35] A. Castillo García, "Diseño de un Convertidor de Corriente en Tecnología CMOS 0.35 µm", Proyecto de Fin de Carrera, Universidad de Las Palmas de Gran Canaria, España, 2009.

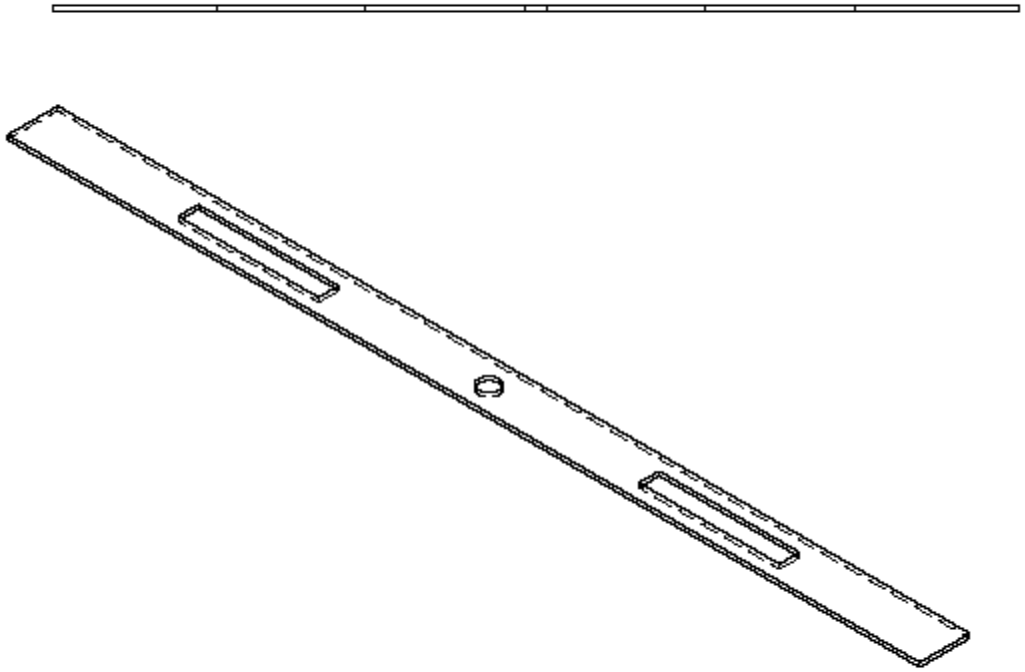
[36] A. Czyzewski, B. Kostek and R. Krolikowski, "Neural Networks Applied to Sound Source Localization", *Proc. of the 110th Audio Engineering Society Convention*, Amsterdam, Holland, 2001.
[37] S. Hariharan, "Audio Signal Classification", *M.Tech. Credit Seminar Report, Electronic Systems Group*, EE. Dept, IIT Bombay, November 2004.
[38] 99Sounds, Free Rain And Thunder Sounds, consultado el 20/05/19 en http://99sounds.org/rain-and-thunder/
[39] H. Tang, "DOA estimation based on MUSIC algorithm", Bachelor's degree thesis, Institutionen för Fysik och Elektroteknik, Växjö, Sweden, 2014.
[40] F. Aguirre Martín, "Desarrollo y análisis de clasificadores de señales de audio", Trabajo de Fin de Máster, Universidad Politécnica de Valencia, Gandía, España, Julio de 2017.
[41] B. O'Keefe, "Finding Location with Time of Arrival and Time Difference of Arrival Techniques", ECE Senior Capstone Project, 2017.
[42] R.Horaud, "Binaural Hearing for Robots", Part 3: Sound-Source Localization, Institut National de Recherche en Informatique et en Automatique, France, March 16,2015, consultado el 20/05/19 en https://www.canal-u.tv/producteurs/inria/cours_en_ligne/binaural_hearing_for_robots/3_sound_source_localization
[43] P.C. Ching, H.C. So and S.Q. Wu, "On Wavelet Denoising and its Applications to Time Delay Estimation", in *IEEE Transactions on Signal Processing*, Volume: 47 , Issue: 10 , pp.2879 – 2882, Oct 1999.
[44] MATLAB - MathWorks, Live Direction Of Arrival Estimation with a Linear Microphone Array, consultado el 20/05/19 en https://es.mathworks.com/help/audio/examples/live-direction-of-arrival-estimation-with-a-linear-microphone-array.html
[45] Ltd. VOCAL Technologies, "Perceptual Noise Reduction for Voice Quality Enhancement" consultado el 20/05/19 en https://www.vocal.com/noise-reduction/perceptual-noise-reduction/
[46] A. Karbasi and A. Sugiyama,"A DOA Estimation Method For an Arbitrary Triangular Microphone Arrangement" in *2006 14th European Signal Processing Conference*, Florence, Italy, 4-8 Sept. 2006.
[47] MATLAB - MathWorks Spain, Record from sound card, consultado el 20/05/2019 en https://es.mathworks.com/help/audio/ref/audiodevicereader-system-object.html
[48] Y. Hioka and N. Hamada, "DOA Estimation of Speech Signal Using Microphones Located at Vertices of Equilateral Triangle" in *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, E87A(3):559-566, March 2004.
[49] W. Hartmann, "How We Localize Sound, Physics Today", Vol. 11, pp. 24-29, November 1999.
[50] M. Cazorla and M. Bravo, "Instructivo del Taller de Elaboración de Gráficos de Meteorología Operacional en MATLAB", Congreso Anual de Meteorología y Calidad del Aire (CAMCA), Instituto de Investigaciones Atmosféricas de la Universidad San Francisco de Quito, Quito, Ecuador,Abril, 2015.
[51] BTD-300 Thunderstorm Detector consultado el 20/05/19 en https://www.biral.com/product/btd-300-thunderstorm-detector/

[Esta página ha sido dejada intencionalmente en blanco]

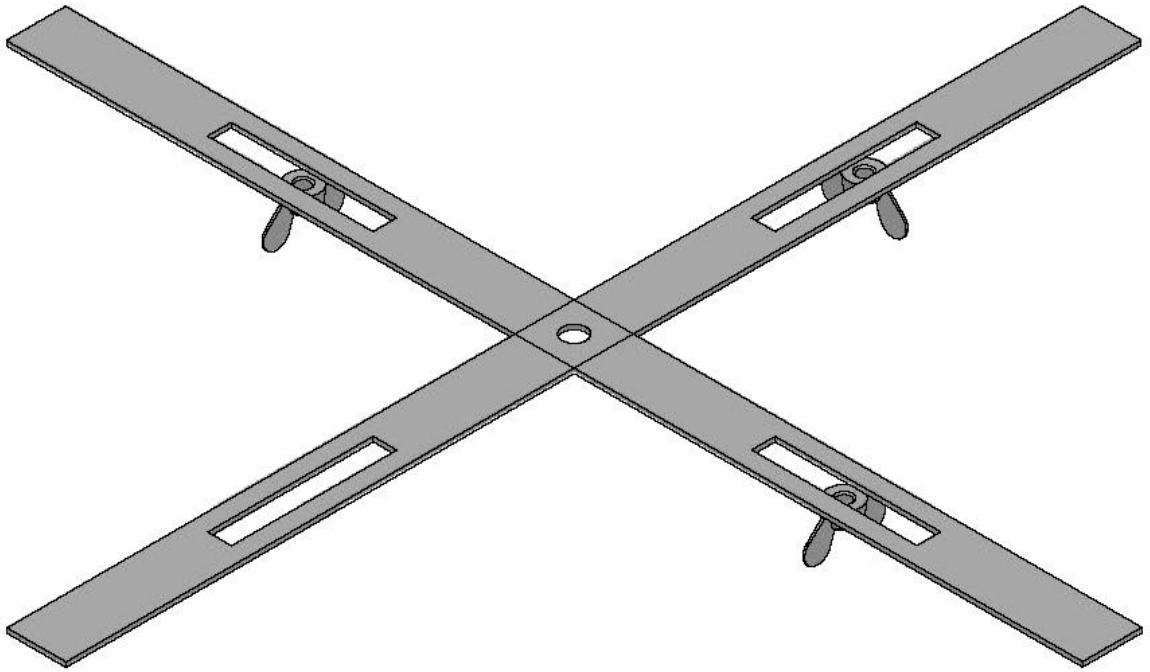
PLANOS

[Esta página ha sido dejada intencionalmente en blanco]





TRABAJO DE FIN DE GRADO		
JUAN FRANCISCO MARTIN RODRIGUEZ	4 VISTAS	07/06/2019
	LAS PALMAS	N°1



TRABAJO DE FIN DE GRADO		
JUAN FRANCISCO MARTIN RODRIGUEZ	DIBUJO 3D	07/06/2019
	LAS PALMAS	N°2

PLIEGO DE CONDICIONES

[Esta página ha sido dejada intencionalmente en blanco]

1. Introducción

El pliego de condiciones general contiene una descripción del contenido del Trabajo Fin de Grado, esto es, la legislación de obligado cumplimiento para la elaboración del apartado técnico (en nuestro caso la Norma UNE-EN ISO 3382-2) y para la utilización y distribución del software creado en el trabajo. En cuanto a las condiciones técnicas particulares se recogerán los materiales, equipos y software utilizados en el proyecto.

2. Condiciones de uso del Software

El software se proporciona tal y como es, sin garantía de ningún tipo, expresa o implícita. No está limitado a garantías de comercialización ni idoneidad para un propósito particular. En ningún caso los autores o titulares del copyright serán responsables de ninguna reclamación, daños u otras responsabilidades, ya sea en una acción de contrato, agravio o cualquier otro motivo, que surja de conexión con el software o el uso u otro tipo de acciones en el software.

3. Equipamiento Hardware

Scarlett Focusrite 18i20

La interfaz de audio Scarlett 18i20 proporciona los medios para la conexión de micrófonos, instrumentos musicales, señales de audio de nivel de línea y señales de audio digitales en los formatos ADAT y S/PDIF a un ordenador que ejecute las versiones compatibles de Mac OS o Windows mediante uno de los puertos USB del ordenador.
Se puede enrutar las señales de las entradas físicas a la estación de trabajo de audio digital (DAW) a un máximo de 24 bits y 192 kHz de resolución; y, de modo similar, las señales de salida grabadas o de monitor de la DAW pueden configurarse para aparecer en las salidas físicas de la unidad.
Las salidas pueden conectarse a amplificadores y altavoces, monitores activos, auriculares, un mixer de audio o cualquier otro equipo de audio analógico o digital que desee usar. A pesar de que todas las entradas y las salidas de Scarlett 18i20 se encuentran enrutadas directamente a y desde su DAW para la grabación y reproducción, podrá configurar el enrutamiento en su DAW de modo que se adapte a sus necesidades particulares.

Figura PL. 1.– Interfaz de audio Roland Octacapture.

Grabadora zoom H6

Esta grabadora se ha usado para recoger el sonido ambiente. Sus características más importantes son las siguientes:
- Grabación simultánea de seis pistas.
- Cuatro entradas de micro/línea sobre conectores combo XLR/TRS.
- Controles de ganancia (knobs reales) y atenuadores (pads) de -20dB para cada entrada.
- Alimentación Phantom en todas las entradas principales.
- Gran pantalla LCD a todo color, en ángulo para que pueda leerse fácilmente en cualquier entorno.
- Graba directamente a tarjetas SD, SDHC y SDXC de hasta 128GB.
- Soporta formatos de audio de hasta 24-bit /96 kHz del tipo WAV conforme-con-BWF o distintos tipos de MP3.
- Multipistas a mezcla estéreo interna.
- Puerto USB para intercambio de datos hacia y desde el ordenador.
- Más de 20 horas de autonomía con 4 pilas alcalinas AA.

Figura PL. 2.- Grabadora zoom H6.

Micrófono Behringer ECM8000

Se han usado para recoger la señal y poder realizar las pruebas de detección, clasificación y localización de los eventos sonoros se han usado tres micrófonos Behringer ECM8000. Este micrófono de condensador es omnidireccional y tiene una respuesta en frecuencia ultra-plana, lo que lo hace apropiado para obtener medidas con una gran fidelidad.

Figura PL. 3.- Micrófono Behringer ECM8000.

Su respuesta en frecuencia abarca todo el rango audible (20 Hz – 20 kHz), tiene una impedancia interna de 200 Ω y posee una sensibilidad de 70 dB ref. 1 V/Pa.

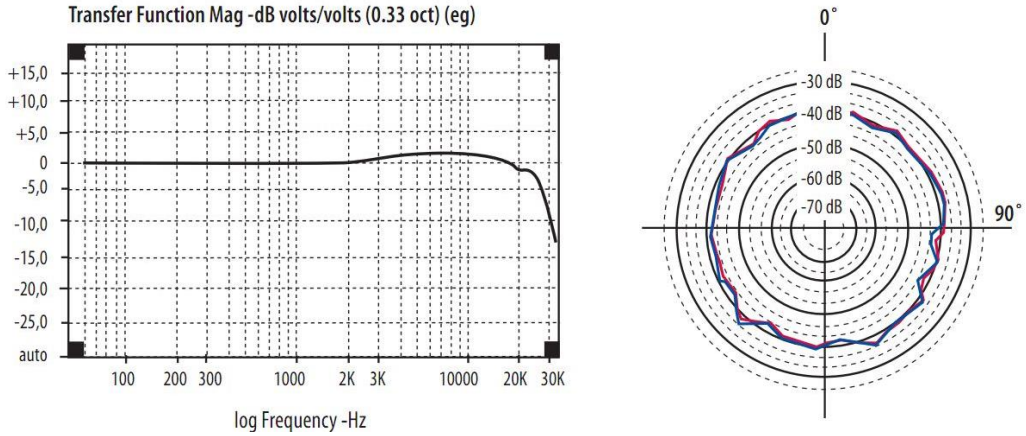


Figura PL. 4.- Izquierda, respuesta en frecuencia en el eje. Derecha, su diagrama de directividad del ECM8000.

Monitor M-Audio BX5 D2

Se usado este monitor para simular los escenarios sonoros desde el exterior.

Figura PL. 5.- Monitor M-Audio BX5.

Su respuesta de frecuencia abarca el rango de 20 Hz a 22 kHz, tiene una frecuencia de cruce 3 kHz y tiene una impedancia de entrada: 20 k Ω balanceadas y 10 k Ω no balanceada.

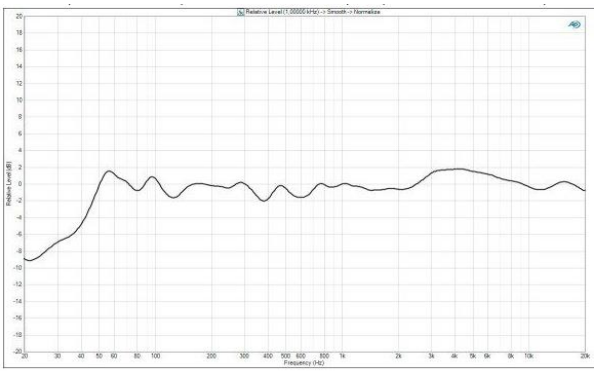


Figura PL. 6.- Respuesta en Frecuencia Monitor M-Audio BX5.

Calibrador Bruel & Kjaer Type 4231

El calibrador de nivel sonoro 4231 se utiliza para que todos los micrófonos capten el mismo nivel de entrada.
Al pulsar el botón “on”, el calibrador se pone en marcha, de forma que se emite un tono de 1 kHz con un nivel de presión sonora de 94 dB. El calibrador posee además un botón “+20 dB”, que hace que el nivel de presión sonora se incremente en 20 dB sobre su valor normal.
El micrófono se calibra colocando el 4231 sobre el micrófono, de forma que lo que se tiene que apreciar en el analizador de espectros son los 94 dB que se están emitiendo desde el calibrador.

Figura PL. 7.- Calibrador Bruel & Kjaer Type 4231.

Ordenador Personal Lenovo Ideapad 320

Sus características son las siguientes:

- Sistema operativo: Windows 10 Home de 64 bits, procesador x64
- 12 Gb de memoria RAM
- Procesador Intel Core i5-7200U @ CPU 2.50GHz 2.71 GHz

Figura PL. 8.- Ordenador Personal Lenovo Ideapad 320.

En él se han instalado los distintos programas necesarios para la realización de este proyecto, entre los que destacamos Microsoft Word para escribir este documento llamado memoria y Matlab 2017b, con el cual hemos desarrollado los scripts de nuestro sistema y la aplicación con su editor de interfaces de usuario.

4. Equipamiento Software

Matlab R2017b

MATLAB es el nombre abreviado de "MATrix LABoratory". Es una herramienta de software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows, Mac OS X y GNU/Linux.
Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, estas son, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolboxes); y las de Simulink con los paquetes de bloques (blocksets).

Entre los toolboxes, destacamos la Audio System Toolbox ya que nos ha servido de ayuda para realizar este proyecto.
Audio Toolbox [4], proporciona herramientas para el procesamiento de audio, el análisis de la voz y la medición acústica. Incluye algoritmos para el procesamiento de señales de audio (como la ecualización y el control del rango dinámico) y la medición acústica (como la estimación de la respuesta a impulso, el filtrado de octavas y la ponderación perceptiva). También proporciona algoritmos para la extracción de características de audio y voz (como MFCC y tono) y la transformación de señales de audio (como el banco de filtros gammatono y el espectrograma con espaciado de Mel).

Las apps de esta toolbox soportan las pruebas de algoritmos en tiempo real, la medición de la respuesta a impulso y el etiquetado de señales de audio. La toolbox proporciona interfaces de streaming para tarjetas de audio y dispositivos MIDI de ASIO, WASAPI, ALSA y CoreAudio, así como herramientas para generar y alojar complementos de audio estándar como VST y Audio Units.

Con Audio Toolbox, se puede importar, etiquetar y aumentar conjuntos de datos de audio, además de extraer características y transformar señales para machine learning y deep learning. Es posible prototipar algoritmos de procesamiento de audio en tiempo real mediante el streaming de audio de baja latencia mientras se ajustan los parámetros y se visualizan las señales. También se puede validar el algoritmo mediante su conversión en un complemento de audio ejecutarlo en aplicaciones de alojamiento externas como, por ejemplo, estaciones de trabajo de audio digital. El alojamiento de

complementos permite utilizar complementos de audio externos como objetos normales para procesar arrays de MATLAB. La conectividad para tarjetas de sonido permite ejecutar mediciones personalizadas en señales de audio y sistemas acústicos reales.

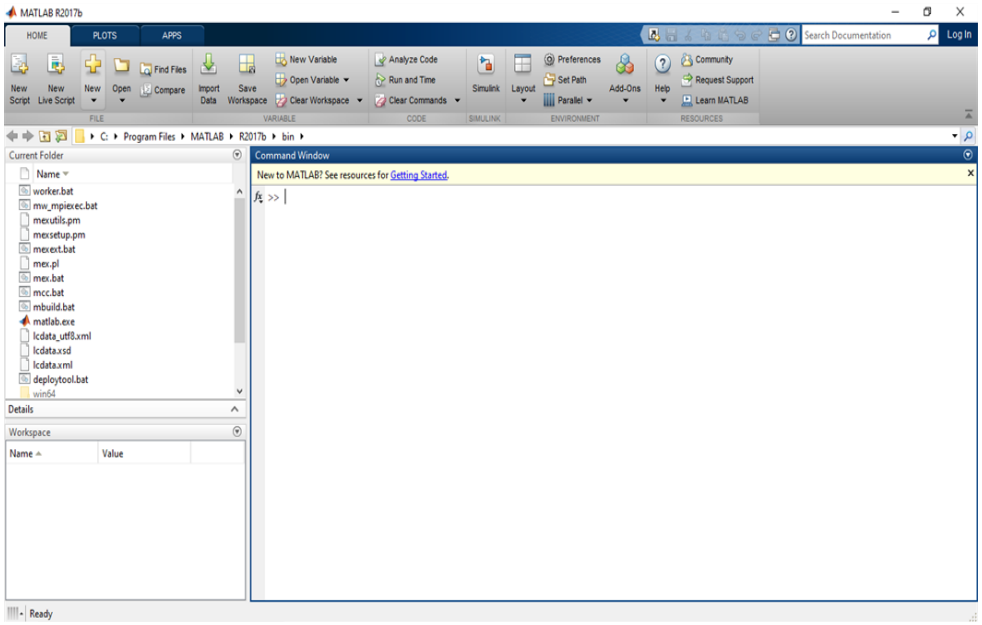


Figura PL. 9.- Interfaz de Matlab (Versión R2017b).

El software matemático MATLAB se ha utilizado para la elaboración de todos los algoritmos y la aplicación y para ejecutar el programa.

Audacity 2.3.1

Audacity es un editor y grabador de audio multipista, fácil de usar y gratuito para Windows, Mac OS X, GNU / Linux y otros sistemas operativos. La interfaz está traducida a muchos idiomas.

El audacity se puede usar para:

- Grabar audio en vivo.
- Grabar la reproducción de la computadora en cualquier máquina.
- Convertir cintas y registros en grabaciones digitales o CD.
- Editar los archivos de sonido WAV, AIFF, FLAC, MP2, MP3 u Ogg Vorbis.
- AC3, M4A / M4R (AAC), WMA y otros formatos compatibles con bibliotecas opcionales.
- Cortar, copiar, unir o mezclar sonidos juntos.
- Numerosos efectos que incluyen cambiar la velocidad o el tono de una grabación.
- Escribir sus propios efectos de complemento con Nyquist.
- Audacity es software libre, desarrollado por un grupo de voluntarios y distribuido bajo la Licencia Pública General de GNU (GPL).

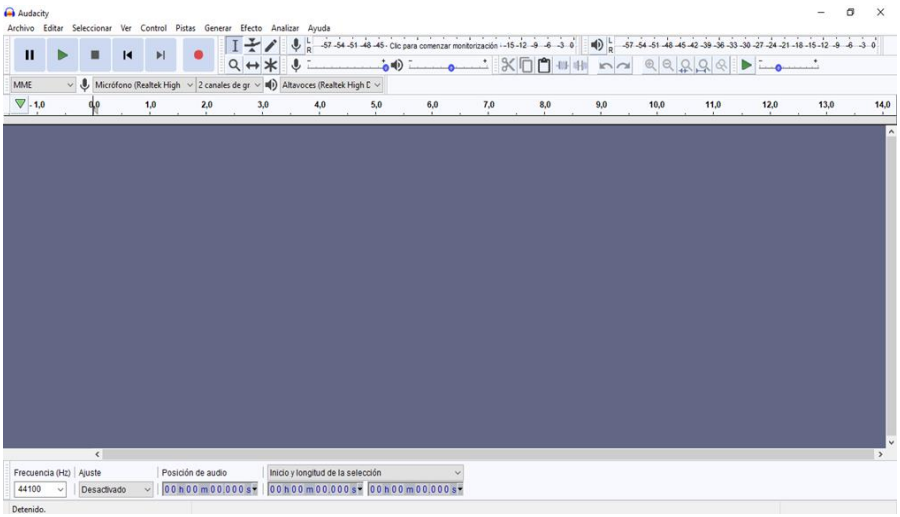


Figura PL. 10. - Interfaz de Audacity (Versión 2.3.1).

El Audacity se ha utilizado para la creación de los distintos escenarios sonoros, para cortar archivos de audio de entrenamiento, además de para ver algunas características de la señal de audio como la frecuencia predominante, niveles en decibelios, el número de muestras o la forma de onda.

Autocad

AutoCAD® es un software de diseño asistido por ordenador (CAD) en el que arquitectos, ingenieros y profesionales de la construcción confían para producir dibujos y documentación en 2D.

Entre sus funciones se encuentran:

- Dibujar y editar geometrías 2D y modelos 3D con caras sólidas, superficies y objetos de malla
- Incluir anotaciones en dibujos con texto, cotas, directrices y tablas
- Personalizar tareas con ayuda de las API y aplicaciones complementarias.

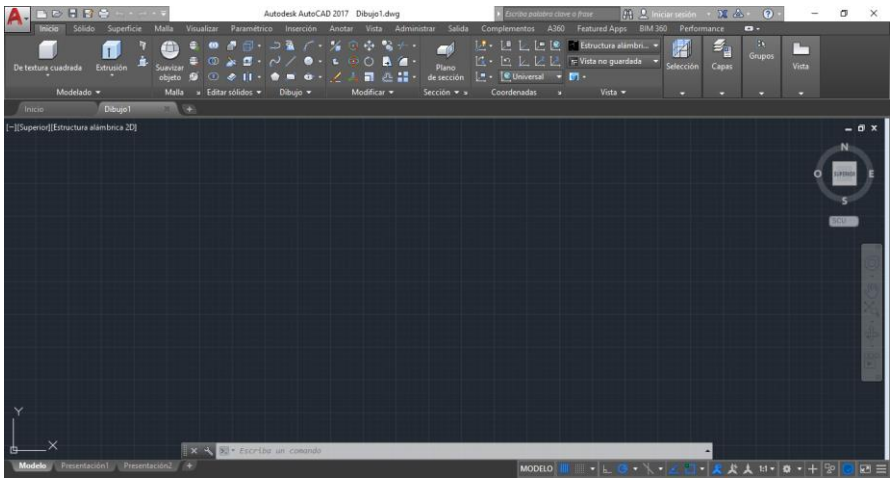


Figura PL. 11. - Interfaz del Autocad 2017.

El Autocad se ha utilizado para la creación de los planos que se encuentran en el apartado de Planos de este proyecto además de algunas figuras de creación propia.

PRESUPUESTO

[Esta página ha sido dejada intencionalmente en blanco]

1. Introducción

El presupuesto que se presenta en este capítulo abarca todo el periodo de realización del trabajo técnico desde el establecimiento de las condiciones del trabajo a realizar, con la elección de medidas, estudio, cálculos, simulaciones, etc.
Se divide en los siguientes apartados:

- Tiempo empleado en la realización del proyecto técnico.
- Amortización del material instrumental e informático.
- Seguros.
- Presupuesto final.

2. Tiempo empleado en la realización del trabajo

Este concepto cubre los gastos de mano de obra, tomando como referencia la recomendación del COIT (Colegio Oficial de Ingenieros de Telecomunicación) de 2015 con la hora de trabajo, a la que se le ha aplicado el incremento del IPC (Índice de Precios al Consumidor) del año 2016, tenemos:

$$H = C \cdot 76{,}07 \cdot H_n + C \cdot 98.26 \cdot H_e[€] \qquad \textit{Ec.PR.1}$$

donde:

C: es un factor de corrección que depende de las horas trabajadas.
H: son los honorarios por tiempo.
Hn: son las horas trabajadas dentro de la jornada laboral normal.
He: son las horas especiales trabajadas, nocturnas y/o festivos.

Tabla PR. 1 .- Factor de corrección por horas de trabajo.

Número de horas	**Factor de corrección**
Menos de 36	1
36-72	0.9
72-108	0.8
108-144	0.7
144-180	0.65
180-360	0.6
360-510	0.55
510-720	0.5

Para la realización de este Trabajo de Fin de Grado se estima un total de 500 horas laborables, con ausencia de horas especiales. Por lo tanto, el factor de corrección a utilizar es de 0.55. Luego, la resolución de la fórmula será:

$$H = 0.55 \cdot 76{,}07 \cdot 500 = \ 20.919{,}25 \ €$$

La tarifa por tiempo de ejecución es de VEINTE MIL NOVECIENTOS DIECINUEVE EUROS CON VEINTICINCO CÉNTIMOS (20.919,25 €).

3. Amortización del material instrumental e informático

En este concepto se toman en consideración tanto la amortización del material hardware, así como del material software empleado en la ejecución del trabajo de fin de grado. Se estipula el coste de amortización del hardware para un periodo de 5 años, y del software por un periodo de 3 años, usando un sistema de amortización lineal. En este sistema, se toma que el inmovilizado material se deprecia de forma constante a lo largo de su vida útil. La cuota de amortización anual se calcula haciendo uso de la fórmula:

$$CUOTA\ DE\ AMORTIZACION\ ANUAL = \frac{Valor\ de\ adquisición - Valor\ Residual}{Número\ de\ años\ de\ vida\ útil} \qquad Ec.PR.2$$

El valor de adquisición es el valor invertido en la adquisición del instrumento o herramienta considerada. El valor residual es el valor que se estima que tendrá el instrumento o herramienta al final de su vida, el valor típico fijado es de un 5% de su valor de adquisición. La vida útil es el número de años durante el que se estima que el instrumento o herramienta conserva un provecho pleno y competitivo.
Una vez obtenida la cuota anual, se dividirá entre el número de días totales en un año para obtener la cantidad equivalente por día. Por último, esa cantidad se multiplicará por el número de días que se ha utilizado el equipo.

Tabla PR. 2.- Precios y costes de la amortización del Hardware.

Elementos Hardware	**Valor de Adquisición (€)**	**Valor Residual (5 Años (€)**	**Coste de Amortización (1 año)**	**Tiempo usado (días)**	**Coste de amortización por tiempo de utilización (días)**
Ordenador Portátil Lenovo ideapad 320	750 €	37,5 €	142,50 €	90	36,00 €
3 x Micrófono	3x60 €	9 €	34,20 €	7	0,70 €

Behringer ECM8000					
Scarlett Focusrite 18i20	560 €	28€	106,40 €	7	2,10 €
1 pie de micrófono Millenium MS-2003	10 €	0,50 €	1,90 €	7	0,07 €
3 x soporte de micrófono K&M 85055	3x3,5 €	0,53 €	1,99 €	7	0,07 €
2 x Láminas de aluminio - 0,25kg	2x1 €	0,10 €	0,38 €	7	0,00 €
3 x Cable XLR the sssnake DMX-Cable 1000/3	3x10 €	1,50 €	5,70 €	7	0,14 €
2 x Cable Jack Cordial CCI 3 PP	20 €	1,00 €	3,80 €	7	0,07 €
Grabadora zoom H6	310 €	15,50 €	58,90 €	1	0,16 €
Monitores M-Audio BX5 D2	200 €	10,00 €	38,00 €	7	0,77 €
Calibrador Bruel & Kjaer type 4230	245 €	12,25 €	46,55 €	3	0,39 €
4x ADAPTADOR ROSCA PARA MICROFONO OQAN AMS-AD35	4x1 €	0,20 €	0,76 €	7	0,00 €
Tuercas de mariposa	1 €	0,05 €	0,19 €	7	0,00 €
				TOTAL	40,47 €

Por tanto, el coste total del hardware empleado asciende a la cantidad CUARENTA EUROS CON CUARENTA Y SIETE CÉNTIMOS (40,47 €).

Tabla PR. 3.- Precios y costes de la amortización del Software.

Elementos Software	**Valor de Adquisición**	**Valor Residual (3 años)**	**Coste de Amortización (1 año)**	Tiempo usado (días)	**Coste de amortización por tiempo de utilización (meses)**
Matlab 2017b	2.000 €	100,00 €	633,33 €	90	158,34 €

Microsoft Office	149 €	12,50 €	79,17 €	90	11,70 €
Autocad 2017	2.123,55 €	106,18 €	672,46 €	90	168,30 €
				TOTAL	338,34 €

Por tanto, el coste total del software empleado asciende a la cantidad TRESCIENTOS TREINTA Y OCHO EUROS CON TREINTA Y CUATRO CÉNTIMOS (338,34 €).

La amortización del material instrumental e informático asciende a TRESCIENTOS SETENTA Y OCHO EUROS CON OCHENTA Y UN CÉNTIMOS (378,81 €).

4. Seguros (Cotización Seguridad Social)

En este apartado se incluye la parte correspondiente a este trabajo de la cotización a la Seguridad Social como trabajador autónomo.
La base de cotización aplicada es de 875,70 € y la cuota mensual corresponde al 29,8% de dicha base, por lo que la cantidad a pagar al mes es de 260,96 €.
La realización del Trabajo ha llevado un total de 90 días, por lo que el importe correspondiente asciende a exactamente tres mensualidades, es decir, SETECIENTOS OCHENTA Y DOS EUROS CON OCHENTA Y OCHO CÉNTIMOS (782,88 €).

5. Presupuesto final

Para el cálculo del presupuesto final, en primer lugar, obtendremos "el presupuesto antes de impuestos" como resultado de la sumatoria de la totalidad de los apartados anteriores.

Tabla PR. 4.- Presupuesto antes de impuestos.

CONCEPTO	SUBTOTAL (€)
Trabajo tarifado por tiempo	20.919,25 €
Amortización de equipos y software	378,81 €
Seguro	782, 88 €
TOTAL	22.080,94 €

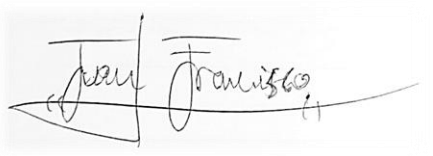
Al "presupuesto antes de impuestos" se le debe aplicar un cargo por impuestos en la Comunidad Autónoma de Canarias, esto es, el I.G.I.C (Impuesto General Indirecto de Canarias) que asciende a un 7%. Con esto el presupuesto final queda de la siguiente forma:

Tabla PR. 5.- Presupuesto después de impuestos.

CONCEPTO	SUBTOTAL (€)
Presupuesto antes de impuestos	22.080,94 €
IGIC 7%	1.545,66€
TOTAL	23.626,60€

El presupuesto total asciende a VEINTITRES MIL SEISCIENTOS VEINTISÉIS EUROS CON SESENTA CÉNTIMOS (23.626,60€).

Las Palmas de Gran Canaria, a 9 de Junio de 2019



Fdo: Juan Francisco Martín Rodríguez

[Esta página ha sido dejada intencionalmente en blanco]

ANEXOS

[Esta página ha sido dejada intencionalmente en blanco]

1. Código Matlab

INTERFACES GUI

SISEVEMETEO.m

```
function varargout = SISEVEMETEO(varargin)
% SISEVEMETEO MATLAB code for SISEVEMETEO.fig
%      SISEVEMETEO, by itself, creates a new SISEVEMETEO or raises the existing
%      singleton*.
%
%      H = SISEVEMETEO returns the handle to a new SISEVEMETEO or the handle to
%      the existing singleton*.
%
%      SISEVEMETEO('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in SISEVEMETEO.M with the given input arguments.
%
%      SISEVEMETEO('Property','Value',...) creates a new SISEVEMETEO or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before SISEVEMETEO_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to SISEVEMETEO_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help SISEVEMETEO

% Last Modified by GUIDE v2.5 18-May-2019 18:47:42

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @SISEVEMETEO_OpeningFcn, ...
                   'gui_OutputFcn',  @SISEVEMETEO_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before SISEVEMETEO is made visible.
function SISEVEMETEO_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to SISEVEMETEO (see VARARGIN)

% Choose default command line output for SISEVEMETEO
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes SISEVEMETEO wait for user response (see UIRESUME)
addpath(genpath('./'));
imshow('cielo.jpg');
```

```
var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
end
end

% --- Outputs from this function are returned to the command line.
function varargout = SISEVEMETEO_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles) % Acceso Profesional
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(SISEVEMETEO);
PROFESIONAL;

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles) % DEMO
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(SISEVEMETEO);
DEMO;


% --------------------------------------------------------------------
function Ayuda_Callback(hObject, eventdata, handles)
% hObject    handle to Ayuda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function acceso_Callback(hObject, eventdata, handles)
% hObject    handle to acceso (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
end
end
message = sprintf('Acceso Básico:  Adecuado para usuarios que necesiten saber de manera
rápida cómo funciona el sistema, ya que existen dos archivos de audio cargados para
procesar de los cuales elegiremos uno. Podremos saber cómo funciona el sistema, antes de
probar el acceso profesional.\n\n Acceso Profesional: Adecuado para usuarios que
necesiten comprobar el funcionamiento de otros parámetros en el sistema. Este acceso
tiene dos modos de funcionamiento, En vivo, donde el sistema recoge señal de audio de
los micrófonos y Grabado, donde el sistema recoge señal de audio de archivos que se
cargan.En el modo en vivo, se puede calibrar el sistema (debería de realizarse
periódicamente, o cada vez que se lanza la aplicación en un nuevo recinto) antes de
lanzarlo recogiendo la señal por micrófonos. En el modo grabado  se puede cargar
cualquier archivo de audio.\n');
f = msgbox(message, 'Accesos al Sistema');
setappdata(0,'f',f);
% --------------------------------------------------------------------
function instrucciones_Callback(hObject, eventdata, handles)
% hObject    handle to instrucciones (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
```

```
if ishandle(f)
    close(f);
    delete(f);
end
end
message = sprintf('Al lanzar la interfaz principal (SISEVEMETEO) con el script
SISEVEMETEO.m:\n\n 1. El usuario debe encontrarse dentro de la carpeta principal, la que
contiene el script SISEVEMETEO.m \n 2. Ejecutar SISEVEMETEO.m desde donde se encuentra y
se añadirán automáticamente todas las carpetas al patch.');
f = msgbox(message, 'Instrucciones');
setappdata(0,'f',f);
```

DEMO.m

```
function varargout = DEMO(varargin)
% DEMO MATLAB code for DEMO.fig
%      DEMO, by itself, creates a new DEMO or raises the existing
%      singleton*.
%
%      H = DEMO returns the handle to a new DEMO or the handle to
%      the existing singleton*.
%
%      DEMO('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in DEMO.M with the given input arguments.
%
%      DEMO('Property','Value',...) creates a new DEMO or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before DEMO_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to DEMO_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help DEMO

% Last Modified by GUIDE v2.5 18-May-2019 11:39:40

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @DEMO_OpeningFcn, ...
                   'gui_OutputFcn',  @DEMO_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before DEMO is made visible.
function DEMO_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to DEMO (see VARARGIN)

% Choose default command line output for DEMO
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
% UIWAIT makes DEMO wait for user response (see UIRESUME)
popupmenu4_Callback(hObject, eventdata, handles);
buttons = [handles.radiobutton3, handles.radiobutton2]; %Deseleccionar
set(buttons, 'Value', 0);

var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
end
end

% --- Outputs from this function are returned to the command line.
function varargout = DEMO_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton5. Empezar
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
var = [];


guidata(hObject,handles);
algoritmo = getappdata(0,'B'); %Obtener knn o rf
if size(getappdata(0,'B')) == size(var)
    warndlg('Seleccione un algoritmo de clasificación', 'Advertencia');
else
    handles.algoritmo = algoritmo;
    guidata(hObject,handles);
    InputAudioExampleDentro;
end

% --- Executes when selected object is changed in uibuttongroup1.
function uibuttongroup1_SelectionChangedFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in uibuttongroup1
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
B = get( hObject, 'String' ); % para solo seleccionar KNN o Random Forest . Mensaje para
elegir algoritmo
setappdata(0,'B',B);
guidata(hObject,handles);


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(DEMO);
SISEVEMETEO;


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2 contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on selection change in popupmenu4.
function popupmenu4_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu4 contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from popupmenu4

allItems = handles.popupmenu4.String;
selectedIndex = handles.popupmenu4.Value;
filename = strcat(allItems{selectedIndex}, '.wav');

workpath = '.';
pathname=[workpath '/Grabado/'];
setappdata(0,'C',filename);
[~, Fs]=audioread([pathname,filename]);
info = audioinfo(filename);
nombre = filename;
set(handles.text5, 'String', nombre);
fs = Fs;
textLabel2 = sprintf([num2str(fs), ' Hz']);
set(handles.text4, 'String', textLabel2);
duracion = info.Duration;
textLabel3 = sprintf([num2str(duracion), ' s']);
set(handles.text3, 'String', textLabel3);
canales = info.NumChannels;
textLabel4 = sprintf('%0.0f', canales);
set(handles.text2, 'String', textLabel4);
handles.filename = filename; %<--- Ojo con esto
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function popupmenu4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


% --------------------------------------------------------------------
function Ayuda_Callback(hObject, eventdata, handles)
% hObject    handle to Ayuda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function modo_Callback(hObject, eventdata, handles)
% hObject    handle to modo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
end
end
message = sprintf('Grabado: Para recoger la señal desde un archivo de audio que simula
la señal recogida por los micrófonos, el número de canales simulará el número de
micrófonos. Habrá que elegir uno de los dos archivos de audio, cuya información se
mostrará.\n');
f = msgbox(message, 'Modo de Funcionamiento');
setappdata(0,'f',f);
% --------------------------------------------------------------------
function algoritmo_Callback(hObject, eventdata, handles)
% hObject    handle to algoritmo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
end
end
message = sprintf('KNN: K-Nearest Neighbors, calcula distancias desde la muestra
desconocida a las clases ya formadas para predecir la clase de la muestra en cuestión,
basándose en los k vecinos más cercanos.\n\nRandom Forest: Se basa en árboles de
decisión, en la fase de train se dibuja el árbol con sus nodos y en la fase de test, se
dibuja el camino, que va desde el nodo raiz al nodo hoja de cada árbol, donde saldrá un
histograma con la clase ganadora. Este algoritmo crea bosques(conjuntos de árboles),
dividiendo los datos de manera aleatoria según el número de árboles, la clase ganadora
será la más votada en todos los árboles.\n');
f = msgbox(message, 'Algoritmos de Clasificación');
setappdata(0,'f',f);
% --------------------------------------------------------------------
function botones_Callback(hObject, eventdata, handles)
% hObject    handle to botones (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
end
end
message = sprintf('Atrás: Vuelve a la interfaz inicial, SISEVEMETEO.\n\nEmpezar:
Comienza a funcionar del sistema, con los parámetros de entrada (algoritmo y archivo)
que haya elegido el usuario.\n');
f = msgbox(message, 'Botones Exteriores');
setappdata(0,'f',f);
```

PROFESIONAL.m

```
function varargout = PROFESIONAL(varargin)
% PROFESIONAL MATLAB code for PROFESIONAL.fig
%      PROFESIONAL, by itself, creates a new PROFESIONAL or raises the existing
%      singleton*.
%
%      H = PROFESIONAL returns the handle to a new PROFESIONAL or the handle to
%      the existing singleton*.
%
%      PROFESIONAL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PROFESIONAL.M with the given input arguments.
%
%      PROFESIONAL('Property','Value',...) creates a new PROFESIONAL or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before PROFESIONAL_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to PROFESIONAL_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help PROFESIONAL

% Last Modified by GUIDE v2.5 25-May-2019 12:42:05

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @PROFESIONAL_OpeningFcn, ...
    'gui_OutputFcn',  @PROFESIONAL_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before PROFESIONAL is made visible.
function PROFESIONAL_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to PROFESIONAL (see VARARGIN)

% Choose default command line output for PROFESIONAL
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes PROFESIONAL wait for user response (see UIRESUME)
% uiwait(handles.figure1);

buttons = [handles.radiobutton3, handles.radiobutton4, handles.radiobutton7,
handles.radiobutton11,handles.radiobutton22,handles.radiobutton23]; % Crear esto para
todo lo demás y solo habilitar botones cuando se seleccione
set(buttons, 'Value', 0);
set(handles.pushbutton9,'Enable','off')
set(handles.pushbutton6,'Enable','off')
set(handles.pushbutton17,'Enable','off')
set(handles.pushbutton18,'Enable','off')
set(handles.edit1,'Enable','off')
set(handles.radiobutton3,'Enable','off')
set(handles.radiobutton4,'Enable','off')
set(handles.radiobutton22,'Enable','off')
set(handles.radiobutton23,'Enable','off')
```

```
% Limpia las variables cada vez que empieza. Se resetea todo
if size(getappdata(0,'A')) ~= 0 % Algoritmo de clas
    rmappdata(0,'A');
end
if size(getappdata(0,'B')) ~= 0 % Modo de funcionamiento
    rmappdata(0,'B');
end

if size(getappdata(0,'C')) ~= 0 % Filename
    rmappdata(0,'C');
end

if size(getappdata(0,'D')) ~= 0 % Tiempo o infinito
    rmappdata(0,'D');
end


set(handles.text23, 'String', """");
set(handles.text22, 'String', """");
set(handles.text21, 'String', """");
set(handles.text20, 'String', """");

var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
end
end


ayuda = getappdata(0,'ayuda');
if length(ayuda) ~= length(var)
if ishandle(ayuda)
    handles2 = getappdata(AYUDA);
    close(handles2.UsedByGUIData_m.figure1);
end
end


% --- Outputs from this function are returned to the command line.
function varargout = PROFESIONAL_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;




% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1


% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2


% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox1 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox1



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(PROFESIONAL);
SISEVEMETEO;


% --- Executes when selected object is changed in uibuttongroup1.
function uibuttongroup1_SelectionChangedFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in uibuttongroup1
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.radiobutton3,'Enable','on')
set(handles.radiobutton4,'Enable','on')
B = get( hObject, 'String' ); % para solo seleccionar en vivo o grabado
setappdata(0,'B', B);
if strcmp( B, 'Grabado' )
    set(handles.pushbutton9,'Enable','on');
    set(handles.pushbutton6,'Enable','off');
    set(handles.pushbutton17,'Enable','off')
    set(handles.pushbutton18,'Enable','on')
    set(handles.edit1,'Enable','off');
    set(handles.radiobutton22,'Enable','off')
    set(handles.radiobutton23,'Enable','off')
elseif strcmp( B, 'En vivo' ) %Activar solo cuando seleccione en tiempo
    set(handles.pushbutton6,'Enable','on');
    set(handles.radiobutton22,'Enable','on')
    set(handles.radiobutton23,'Enable','on')
    set(handles.pushbutton17,'Enable','on')
    set(handles.pushbutton18,'Enable','off')
    set(handles.pushbutton9,'Enable','off');
end



% --- Executes on button press in pushbutton5. Empezar
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
var = [];

if ~isempty(findobj('Name','AYUDA')); % Si ayuda está abierto lo cerramos
    hf = findobj('Name','AYUDA');
    close(hf);
end

f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
end
end

if size(getappdata(0,'B')) == size(var)
    warndlg('Seleccione el Modo de Funcionamiento', 'Advertencia');
else
    if size(getappdata(0,'A')) == size(var)
        warndlg('Seleccione un algoritmo de clasificación', 'Advertencia');
    else
        if strcmp( getappdata(0,'B'), 'En vivo' )
            tiempoVal = getappdata(0,'D');
            if strcmp(tiempoVal, 'Tiempo:')
                tiempo = str2double(get(handles.edit1, 'string'));
                if isnan(tiempo) % Si tiempo no es un número sale display
                    set(handles.edit1, 'String','');
                    warndlg('Inserte un tiempo numérico o selecciona infinito',
'Advertencia');
                else
                    handles.algoritmo = getappdata(0,'A');
                    handles.tiempo = tiempo;
                    guidata(hObject,handles);
                    InputAudioExampleMic;
                end
            elseif strcmp(tiempoVal, 'Infinito') % Si el tiempo es infinito
                handles.algoritmo = getappdata(0,'A');
                handles.tiempo = Inf;
                guidata(hObject,handles);
                InputAudioExampleMic;
            else
                warndlg('Inserte un tiempo numérico o selecciona infinito',
'Advertencia');
            end
        elseif strcmp(getappdata(0,'B'), 'Grabado' )
            if size(getappdata(0,'A')) == size(var)
                warndlg('Seleccione un algoritmo de clasificación', 'Advertencia');
            else
                if size(getappdata(0,'C')) == size(var)
                    warndlg('Seleccione un archivo para cargar', 'Advertencia');
                else
                    handles.filename = getappdata(0,'C');
                    handles.algoritmo = getappdata(0,'A');
                    guidata(hObject,handles);
                    InputAudioExampleDentro;
                end
            end
        end
    end

end

% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(PROFESIONAL);
MiCalibracion;


% --- Executes on button press in pushbutton7.
%function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[filename, pathname] = uigetfile( {'*.wav'}, 'Pick a file');

if filename ~=0
setappdata(0,'C', filename);
[y, Fs]=audioread([pathname,filename]);
info = audioinfo(filename);
nombre = filename;
set(handles.text23, 'String', nombre);
fs = Fs;
textLabel2 = sprintf([num2str(fs), ' Hz']);
set(handles.text22, 'String', textLabel2);
duracion = info.Duration;
textLabel3 = sprintf([num2str(duracion), ' s']);
set(handles.text21, 'String', textLabel3);
canales = info.NumChannels;
textLabel4 = sprintf('%0.0f', canales);
set(handles.text20, 'String', textLabel4);
handles.filename = filename;
guidata(hObject,handles)
end

% --- Executes when selected object is changed in uibuttongroup2.
function uibuttongroup2_SelectionChangedFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in uibuttongroup2
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
A = get( hObject, 'String' );% Obtener en KNN o RF
setappdata(0,'A', A);
guidata(hObject,handles);



% --- Executes when selected object is changed in uibuttongroup8.
function uibuttongroup8_SelectionChangedFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in uibuttongroup8
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

D = get( hObject, 'String' );% Obtener tiempo
setappdata(0,'D', D);
guidata(hObject,handles);

if strcmp( D, 'Tiempo:' )
    set(handles.edit1,'Enable','on')
else
    set(handles.edit1,'Enable','off')
end



% --------------------------------------------------------------------
function Ayuda_Callback(hObject, eventdata, handles)
% hObject    handle to Ayuda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function modo_Callback(hObject, eventdata, handles)
% hObject    handle to modo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
```

```
end
end

ayuda = getappdata(0,'ayuda')
if length(ayuda) ~= length(var)
if ishandle(ayuda)
    handles2 = getappdata(AYUDA);
    close(handles2.UsedByGUIData_m.figure1);
end
end

message = sprintf('En vivo: Para recoger la señal por micrófonos y probar el sistema en
exteriores. Habrá que introducirle el tiempo de funcionamiento.\n\nGrabado: Para recoger
la señal por un archivo de audio que simula la señal recogida por los micrófonos, el
número de canales de este archivo simulará el número de micrófonos. Habrá que cargar el
archivo de audio, cuya información se mostrará.\n');
f = msgbox(message, 'Modos de Funcionamiento');
setappdata(0,'f',f);
% --------------------------------------------------------------------
function algoritmo_Callback(hObject, eventdata, handles)
% hObject    handle to algoritmo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
end
end
ayuda = getappdata(0,'ayuda')
if length(ayuda) ~= length(var)
if ishandle(ayuda)
    handles2 = getappdata(AYUDA);
    close(handles2.UsedByGUIData_m.figure1);
end
end

message = sprintf('KNN(K-Nearest Neighbors): calcula distancias desde la muestra
desconocida a cada grupo de clases ya formadas, basándose en los k vecinos más cercanos
para predecir la clase a la que pertenece la muestra en cuestión.\n\nRandom Forest: Se
basa en árboles de decisión, en la fase de train se dibuja el árbol con sus nodos,
estableciendo los umbrales en cada rama, y en la fase de test, se dibuja el camino, que
va desde el nodo raiz al nodo hoja de cada árbol, cada hoja contendrá un histograma con
la clase ganadora. Este algoritmo crea bosques(conjuntos de árboles) aleatorios,
dividiendo los datos de manera aleatoria según el número de árboles, la clase ganadora
será la más votada en todos los árboles.\n')
f = msgbox(message, 'Algoritmo de Clasificación');
setappdata(0,'f',f);
% --------------------------------------------------------------------
function botones_Callback(hObject, eventdata, handles)
% hObject    handle to botones (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
end
end

ayuda = getappdata(0,'ayuda')
if length(ayuda) ~= length(var)
if ishandle(ayuda)
    handles2 = getappdata(AYUDA);
    close(handles2.UsedByGUIData_m.figure1);
end
end

message = sprintf('Atrás: Vuelve a la interfaz inicial, SISEVEMETEO.\n\nEmpezar: Empieza
el funcionamiento del sistema, con los parámetros de entrada (algoritmo y archivo en el
modo grabado o algoritmo y tiempo en el modo en vivo) que haya introducido el
usuario.\n(IMPORTANTE MODO EN VIVO) --> Al pulsar sobre este botón, tenemos que esperar
```

```
un tiempo (6 segundos) a que se calcule el umbral sin emitir sonidos que alteren el
cálculo, este umbral distinguirá entre el sonido y el silencio');
f = msgbox(message, 'Botones Exteriores');
setappdata(0,'f',f);
% --- Executes on button press in pushbutton17.
function pushbutton17_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%msgbox('Botón para calibrar cada uno de los tres micrófonos')
var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
end
end

handles2 = getappdata(AYUDA);
set(handles2.UsedByGUIData_m.text5,'String', "CALIBRACION");
ayuda = imshow('mics.JPG');
setappdata(0,'ayuda', ayuda);
message = sprintf('Antes de pulsar el botón Calibrar, debemos asegurarnos de que los
micrófonos están calibrados, realizando la calibración normal, donde habrá que ajustar
los niveles sonoros en la Scarlett Focusrite 18i20 para que a todos los micrófonos le
llegue el mismo nivel sonoro.\n\nCalibrar: Este botón sirve para generar un archivo .mat
con tres factores de multiplicación, los cuales se multiplican a la amplitud de cada
canal de la señal cuando empiece a funcionar el sistema en exteriores, de manera que
exista el mismo nivel sonoro en los tres canales, siendo el sistema más preciso en la
Localización de la Dirección de Procedencia.\n\nCada micrófono está numerado en cada
vértice del triángulo y representado en la imagen adjunta.\nEsta calibración es de
refuerzo y se realiza de la misma manera que la calibración normal, esto es, colocando
el calibrador sobre cada micrófono y esperando a que el programa lance el mensaje
informando que está calibrado, debemos realizar este proceso hasta calibrar los tres
micrófonos');
f = msgbox(message, 'Calibración');
setappdata(0,'f',f);
% --- Executes on button press in pushbutton18.
function pushbutton18_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
var = [];
f = getappdata(0,'f'); %Se cierra la ventana de info si está abierta
if length(f) ~= length(var)
if ishandle(f)
    close(f);
    delete(f);
end
end

ayuda = getappdata(0,'ayuda')
if length(ayuda) ~= length(var)
if ishandle(ayuda)
    handles2 = getappdata(AYUDA);
    close(handles2.UsedByGUIData_m.figure1);
end
end

message = sprintf('Cargar: Botón para cargar el archivo de audio con la señal grabada de
tres canales a procesar, cuyos datos saldrán en el panel (Datos del Archivo). Es
importante que tenga tres canales para poder calcular la localización de procedencia con
el algoritmo TDOA (Time Delay.)\n');
f = msgbox(message, 'Boton Cargar');
setappdata(0,'f', f);
```

AYUDA.m

```
function varargout = AYUDA(varargin)
% AYUDA MATLAB code for AYUDA.fig
%      AYUDA, by itself, creates a new AYUDA or raises the existing
%      singleton*.
```

```
%
%      H = AYUDA returns the handle to a new AYUDA or the handle to
%      the existing singleton*.
%
%      AYUDA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in AYUDA.M with the given input arguments.
%
%      AYUDA('Property','Value',...) creates a new AYUDA or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before AYUDA_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to AYUDA_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help AYUDA

% Last Modified by GUIDE v2.5 14-May-2019 22:22:56

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @AYUDA_OpeningFcn, ...
                   'gui_OutputFcn',  @AYUDA_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before AYUDA is made visible.
function AYUDA_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to AYUDA (see VARARGIN)

% Choose default command line output for AYUDA
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes AYUDA wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = AYUDA_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

display.m

```
function varargout = display(varargin)
```

```
% DISPLAY MATLAB code for display.fig
%      DISPLAY, by itself, creates a new DISPLAY or raises the existing
%      singleton*.
%
%      H = DISPLAY returns the handle to a new DISPLAY or the handle to
%      the existing singleton*.
%
%      DISPLAY('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in DISPLAY.M with the given input arguments.
%
%      DISPLAY('Property','Value',...) creates a new DISPLAY or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before display_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to display_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help display

% Last Modified by GUIDE v2.5 08-May-2019 18:15:12

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @display_OpeningFcn, ...
                   'gui_OutputFcn',  @display_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before display is made visible.
function display_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to display (see VARARGIN)

% Choose default command line output for display
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes display wait for user response (see UIRESUME)
set(handles.pushbutton1,'Enable','off');
set(handles.axes3,'visible', 'off');
imshow('equis.jpg');
if size(getappdata(0,'X')) ~= 0
    rmappdata(0,'X');
end

% --- Outputs from this function are returned to the command line.
function varargout = display_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes2


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
parar = 1;
setappdata(0,'X', parar);
guidata(hObject,handles);
```

CALIBRACIÓN

MiCalibracion.m

```
%% MiCalibracion ---------------------------------------------------------%
% Script que realiza la calibración de refuerzo, al final se generará un  %
% archivo .mat que contendrá tres factores de multiplicación, cada uno de %
% ellos se multiplicara por la amplitud de la señal recogida por cada     %
% micrófono, de esta manera habrá mayor precisión en el cálculo de la     %
% Localización de la Dirección de Procedencia(DOA).                       %
%                                                                         %
%.........................................................................%
%                                                                         %
%                                          Juan Francisco Martín Rodríguez %
%                                                                    2019 %
%_________________________________________________________________________%
clear all; close all; clc;
NumMic = 3;  % número de micrófonos que calibrar
ch     = [1,2,3]; % canales a calibrar
aFL = 16000; %audioFrameLength aFL
fs  = 16000;
audioInput = audioDeviceReader(...
    'Driver', 'ASIO', ...
    'SampleRate', fs, ...
    'NumChannels', 1,...
    'OutputDataType','double',...
     'ChannelMappingSource','Property',...
     'SamplesPerFrame', aFL);

%%
for n=1:NumMic
    switch n
        case 1
            respuesta = questdlg('Coloque el calibrador sobre el micrófono 1 y pulse su
botón para calibrarlo', ...
                'Calibración', ...
                'Mic.1','Cerrar','Mic.1');
        case 2
            respuesta = questdlg('Coloque el calibrador sobre el micrófono 2 y pulse su
botón para calibrarlo', ...
                'Calibración', ...
                'Mic.2','Cerrar','Mic.2');
        case 3
            respuesta = questdlg('Coloque el calibrador sobre el micrófono 3 y pulse su
botón para calibrarlo', ...
                'Calibración', ...
                'Mic.3','Cerrar','Mic.3');
    end
        % Calibración del micrófono 1. Se calcula la energía de los 3 canales.
        release(audioInput);
        audioInput.ChannelMapping = ch(n);
```

```
        for p=1:20
            x  = audioInput();
        end
        EE   = 1/aFL*sum(x.^2,1);
        dBx(n) = 10*log10(sqrt(EE));
        f = msgbox(['Micrófono ',num2str(ch(n)),' Calibrado']);
        waitfor(f);
end

if exist('dBx')
    maximo = max(dBx);
    mult  = maximo./dBx;
    % nivelB = [dBx1*f1 dBx2*f2 dBx3*f3];
    file1  = strcat(pwd,'\DOA\factoresT','.mat');
    %Guardar Factores de mult igualados
    save(file1,'mult');
    %disp([nivelB]);
    f = msgbox('Micrófonos Calibrados');
    waitfor(f);
    release(audioInput);
    PROFESIONAL;
end
```

DETECCION

InputAudioExampleDentro.m

```
%% InputAudioExampleDentro ------------------------------------------------%
% Script inicial que se invoca cuando la entrada de la señal se lleva a   %
% cabo por archivos de audio.  En este script se integran las tres        %
% técnicas.                                                               %
%                                                                         %
%.........................................................................%
%                                                                         %
%                                        Juan Francisco Martín Rodríguez %
%                                                                   2019 %
%_________________________________________________________________________%

% Inicializar variables
display;
audioFrameLength = 85000;
sourceChoice = 'recorded';
FL   = 1;
Wvar = 0.25;
L    = 1000;

detect = {};
clasi = {};
doa = [];
fecha = {};
hora = {};
modo = ' ';

li= [];
te=[];
pa=[];

workpath = '.';
temp_path=[workpath '/tmp/'];


guidata(hObject,handles);
if strcmp(get(handles.figure1,'Name'),'PROFESIONAL') % Si está abierto profesional, se
cierra ventana

modo = 'profesional';
set(handles.figure1,'Visible','off')
```

```
elseif strcmp(get(handles.figure1,'Name'),'DEMO') % Si está abierto demo, se cierra
ventana
modo = 'demo';
set(handles.figure1,'Visible','off')
end


B = handles.algoritmo;
if strcmp( B, 'KNN' )
    config.kNN_classifier = 1;
    config.RF_classifier = 0;
elseif strcmp( B, 'Random Forest' )
    config.RF_classifier = 1;
    config.kNN_classifier = 0;
end



if config.kNN_classifier
    if exist([temp_path 'kNN_Classifier.mat'],'file')
        % Cargar clasificador
        load([temp_path 'kNN_Classifier.mat']);
        classifier=trained_classifier.classifier;
    end
elseif config.RF_classifier
    if exist([temp_path 'RandomForest_Classifier.mat'],'file')
        % Cargar clasificador
        load([temp_path 'RandomForest_Classifier.mat']);
        classifier=trained_classifier.classifier;
    end
end

%Cargar normalización
load([temp_path 'normalization.mat']);


%Creación del objeto para recoger señal de audio
audioFileName = handles.filename;
audioInput = dsp.AudioFileReader(...
    'OutputDataType','double',...
    'Filename', audioFileName,...
    'PlayCount', 1, ...
    'SamplesPerFrame', audioFrameLength);
fs = audioInput.SampleRate;

y = audioInput();

handles2 = getappdata(display);
set(handles2.UsedByGUIData_m.pushbutton1,'Enable','on');
strr = sprintf('Modo: Grabado');
set(handles2.UsedByGUIData_m.text7,'String',strr);drawnow;
strr = sprintf('DETECCION');
set(handles2.UsedByGUIData_m.text808,'String',strr);drawnow;
str = sprintf('Calculando umbral...');
set(handles2.UsedByGUIData_m.text69,'String',str);drawnow;
pause(1);

%Cálculo del umbral
Th_ini = umbral_ini(y',L,FL,Wvar);



while ~isDone(audioInput)
    y = audioInput();

    if getappdata(0,'X') == 1 % Parar
        break;
    end

    % Cálculo de las marcas con el minimo umbral
    [markini,markfin] = Al_detector(y(:,1)',L,mean(Th_ini));
    for p =1:length(markini)

        if getappdata(0,'X') == 1 % Parar
            break;
        end
```

```
        if ((markfin(p)-markini(p))>74999) %75k muestras mínimo


            detectVar = sprintf('Sonido');
            pause(1);

            %Almacenar datos
            t = datestr(datetime('now'));
            partido = split(t," ");
            fecha{numel(fecha)+1}  = char(partido(1));
            hora{numel(hora)+1} = char(partido(2));
            detect{numel(detect)+1} = detectVar;

            %Tomamos solo la señal entre las marcas
            y = y(markini(p):markfin(p),:);

            %Clasificación
            [detectado,DD] =  clasificacion(config,y(1:75000,:),norm, classifier, fs);

            strr = sprintf('CLASIFICACION');
            set(handles2.UsedByGUIData_m.text808,'String',strr);drawnow;
            clasiVar = sprintf(detectado);
            set(handles2.UsedByGUIData_m.text69,'String',clasiVar);drawnow;
            if strcmp( detectado,'Trueno')
                axes(handles2.UsedByGUIData_m.axes3);
                imshow('thunder.jpg');
            elseif strcmp( detectado, 'Lluvia')
                str1 = sprintf(['Localización de Procedencia']);
                set(handles2.UsedByGUIData_m.text4,'String',str1);drawnow;

                %limpiar im4
                delete(li);
                delete(te);
                delete(pa);
                axes(handles2.UsedByGUIData_m.axes4);
                imshow('equis.jpg');
                axes(handles2.UsedByGUIData_m.axes3);
                imshow('rain2.jpg');
            elseif strcmp( detectado, 'Sonido Ambiente')
                str1 = sprintf(['Localización de Procedencia']);
                set(handles2.UsedByGUIData_m.text4,'String',str1);drawnow;

                %limpiar im4
                delete(li);
                delete(te);
                delete(pa);
                axes(handles2.UsedByGUIData_m.axes4);
                imshow('equis.jpg');
                axes(handles2.UsedByGUIData_m.axes3);
                imshow('ambient.jpg');
            end

            %Almacenar datos
            clasi{numel(clasi)+1} = clasiVar;

            %Sonar
            sound(y(1:75000,1),fs);
            pause(2);


            if DD == 1
                strr = sprintf('DIRECCION DE PROCEDENCIA');
                set(handles2.UsedByGUIData_m.text808,'String',strr);drawnow;
                [m,n] = size(y);
                if getappdata(0,'X') == 1 %Parar
                    doa(numel(doa)+1) = NaN;
                    break;
                end

                if n == 3
                    % Localización de la dirección de Procedencia
                    [angulo, li, te, pa] = AudioArrayDOAEstimation(y(1:75000,:),
sourceChoice, handles2);
                    doa(numel(doa)+1) = angulo;
                    if getappdata(0,'X') == 1 %Parar
                        break;
                    end
```

```
                else
                    str = sprintf('No se puede hacer la DOA, cargue archivo de tres
canales');
                    set(handles2.UsedByGUIData_m.text4,'String',str);drawnow;
                    pause(1);

                    %Almacenar datos
                    doa(numel(doa)+1) = NaN;
                end
            else
                %Almacenar datos
                doa(numel(doa)+1) = NaN;
            end
        else


            strr = sprintf('DETECCION');
            set(handles2.UsedByGUIData_m.text808,'String',strr);drawnow;
            detectVar = sprintf('Silencio');
            set(handles2.UsedByGUIData_m.text69,'String',detectVar);drawnow;
            axes(handles2.UsedByGUIData_m.axes3);
            imshow('silencio2.png');

            str1 = sprintf(['Localización de Procedencia']);
            set(handles2.UsedByGUIData_m.text4,'String',str1);drawnow;
            delete(li);
            delete(te);
            delete(pa);
            axes(handles2.UsedByGUIData_m.axes4);
            imshow('equis.jpg');
            pause(1);

            %Almacenar datos
            t = datestr(datetime('now'));
            partido = split(t," ");
            fecha{numel(fecha)+1}  = char(partido(1));
            hora{numel(hora)+1} = char(partido(2));
            doa(numel(doa)+1) = NaN;
            detect{numel(detect)+1} = detectVar;
            clasi{numel(clasi)+1} = '----';

        end
    end

end

%Crear tabla y exportar en txt
newChr = strrep(datestr(datetime('now')),':', '.');
newChr = strrep(newChr,' ', '_');
nombre = strcat(workpath,'\Resultados\',handles.filename(1:end-4),'__',newChr,'.txt');
T = table(detect',clasi',doa',fecha',hora');
T.Properties.VariableNames = {'Detecciones' 'Clases' 'DOA' 'Fecha' 'Hora'};
writetable(T,nombre,'Encoding','UTF-8');

%Liberar canales de audio
release(audioInput);


strr = sprintf('FIN');
set(handles2.UsedByGUIData_m.text808,'String',strr);drawnow;
pause(1);
close(display);

if strcmp(modo,'profesional') %volver a abrir
    PROFESIONAL;
elseif strcmp(modo,'demo')
    DEMO;
end
```

InputAudioExampleMic.m

```
%% InputAudioExampleMic ------------------------------------------------%
% Script inicial que se invoca cuando la entrada de la señal se lleva     %
```

```
% a cabo por micrófonos.  En este script se integran las tres técnicas.   %
%                                                                          %
%..........................................................................%
%                                                                          %
%                                          Juan Francisco Martín Rodríguez %
%                                                                     2019 %
%__________________________________________________________________________%

% Inicializar variables
display;
audioFrameLength = 85000;
sourceChoice = 'live';
guidata(hObject,handles);
fs  = 44100;
FL   = 1;
Wvar = 0.25;
L    = 1000;

detect = {};
clasi = {};
doa = [];
fecha = {};
hora = {};
modo = ' ';

li= [];
te=[];
pa=[];


p    = 1;
Th_ini = 0;

workpath = '.';
temp_path=[workpath '/tmp/'];

if strcmp(get(handles.figure1,'Name'),'PROFESIONAL') % Si está abierto profesional, se
cierra ventana
modo = 'profesional';
set(handles.figure1,'Visible','off')
end

B = handles.algoritmo;
if strcmp( B, 'KNN' )
    config.kNN_classifier = 1;
    config.RF_classifier = 0;
elseif strcmp( B, 'Random Forest' )
    config.RF_classifier = 1;
    config.kNN_classifier = 0;
end



if config.kNN_classifier
    if exist([temp_path 'kNN_Classifier.mat'],'file')
        % Cargar clasificador
        load([temp_path 'kNN_Classifier.mat']);
        classifier=trained_classifier.classifier;
    end
elseif config.RF_classifier
    if exist([temp_path 'RandomForest_Classifier.mat'],'file')
        % Cargar clasificador
        load([temp_path 'RandomForest_Classifier.mat']);
        classifier=trained_classifier.classifier;
    end
end

%Cargar normalización
load('./tmp/normalization.mat');

%Creación del objeto para recoger señal de audio
audioInput = audioDeviceReader(...
    'Driver', 'ASIO', ...
    'SampleRate', fs, ...
    'NumChannels', 3,...
    'OutputDataType','double',...
    'SamplesPerFrame', audioFrameLength);
```

```
handles2 = getappdata(display);
strr = sprintf('Modo: En vivo');
set(handles2.UsedByGUIData_m.text7,'String',strr);drawnow;
set(handles2.UsedByGUIData_m.pushbutton1,'Enable','on');

% 6 segundos calculando el umbral
tic
while toc<6
    if getappdata(0,'X') == 1 %Parar
        break;
    end

     y = audioInput();

    strr = sprintf('DETECCION');
    set(handles2.UsedByGUIData_m.text808,'String',strr);drawnow;
    str = sprintf('Calculando umbral...');
    set(handles2.UsedByGUIData_m.text69,'String',str);drawnow;

    %Cálculo del umbral
    Th_ini(p) = umbral_ini(y',L,FL,Wvar);
    p=p+1;
end

tic

while toc < handles.tiempo
    if getappdata(0,'X') == 1 %Parar
        break;
    end

    y = audioInput();

    % Cálculo de las marcas con el umbral medio
    [markini,markfin] = Al_detector(y(:,1)',L,mean(Th_ini));

    if isempty(markini)
        axes(handles2.UsedByGUIData_m.axes3);
        imshow('silencio2.png');
        strr = sprintf('DETECCION');
        set(handles2.UsedByGUIData_m.text808,'String',strr);drawnow;
        detectVar = sprintf('Silencio');
        set(handles2.UsedByGUIData_m.text69,'String',detectVar);drawnow;


        %Almacenar datos
        t = datestr(datetime('now'));
        partido = split(t," ");
        fecha{numel(fecha)+1}  = char(partido(1));
        hora{numel(hora)+1} = char(partido(2));
        doa(numel(doa)+1) = NaN;
        detect{numel(detect)+1} = detectVar;
        clasi{numel(clasi)+1} = '----';
    end

for p =1:length(markini)
    if getappdata(0,'X') == 1 %Parar
        break;
    end

    if ((markfin(p)-markini(p))>74999) %75k muestras mínimo

        detectVar = sprintf('Sonido');
        %Almacenar datos
        t = datestr(datetime('now'));
        partido = split(t," ");
        fecha{numel(fecha)+1}  = char(partido(1));
        hora{numel(hora)+1} = char(partido(2));
        detect{numel(detect)+1} = detectVar;

        %Tomamos solo la señal entre las marcas
        y = y(markini(p):markfin(p),:);
        %Clasificación
        [detectado,DD] =  clasificacion(config,y(1:75000,:),norm, classifier, fs);
```

```
    if strcmp( detectado,'Trueno')
        axes(handles2.UsedByGUIData_m.axes3);
        imshow('thunder.jpg');
        elseif strcmp( detectado, 'Lluvia')
            %limpiar im4
            str1 = sprintf(['Localización de Procedencia']);
            set(handles2.UsedByGUIData_m.text4,'String',str1);drawnow;
            delete(li);
            delete(te);
            delete(pa);
            axes(handles2.UsedByGUIData_m.axes4);
            imshow('equis.jpg');
            axes(handles2.UsedByGUIData_m.axes3);
            imshow('rain2.jpg');
        elseif strcmp( detectado, 'Sonido Ambiente')
            %limpiar im4
            str1 = sprintf(['Localización de Procedencia']);
            set(handles2.UsedByGUIData_m.text4,'String',str1);drawnow;
            delete(li);
            delete(te);
            delete(pa);
            axes(handles2.UsedByGUIData_m.axes4);
            imshow('equis.jpg');
            axes(handles2.UsedByGUIData_m.axes3);
            imshow('ambient.jpg');
        end

    strr = sprintf('CLASIFICACION');
    set(handles2.UsedByGUIData_m.text808,'String',strr);drawnow;

    clasiVar = sprintf(detectado);
    set(handles2.UsedByGUIData_m.text69,'String',clasiVar);drawnow;


    %Almacenar datos
    clasi{numel(clasi)+1} = clasiVar;

    if DD == 1
        if getappdata(0,'X') == 1 %Parar
            doa(numel(doa)+1) = NaN;
            break;
        end

        %Localización de la Dirección de Procedencia
        [angulo] = AudioArrayDOAEstimation(y(1:75000,:), sourceChoice, handles2);
        doa(numel(doa)+1) = angulo;
    else
        %Almacenar datos
        doa(numel(doa)+1) = NaN;
    end

else
    if getappdata(0,'X') == 1 %Parar
        break;
    end

    strr = sprintf('DETECCION');
    set(handles2.UsedByGUIData_m.text808,'String',strr);drawnow;
    detectVar = sprintf('Silencio');
    set(handles2.UsedByGUIData_m.text69,'String',detectVar);drawnow;
    axes(handles2.UsedByGUIData_m.axes3);
    imshow('silencio2.png');
    str1 = sprintf(['Localización de Procedencia']);
    set(handles2.UsedByGUIData_m.text4,'String',str1);drawnow;
    delete(li);
    delete(te);
    delete(pa);
    axes(handles2.UsedByGUIData_m.axes4);
    imshow('equis.jpg');



    %Almacenar datos
    t = datestr(datetime('now'));
    partido = split(t," ");
    fecha{numel(fecha)+1}  = char(partido(1));
```

```
        hora{numel(hora)+1} = char(partido(2));
        doa(numel(doa)+1) = NaN;
        detect{numel(detect)+1} = detectVar;
        clasi{numel(clasi)+1} = '----';
    end
end
end

%Crear tabla y exportar en txt
newChr = strrep(datestr(datetime('now')),':', '.');
newChr = strrep(newChr,' ', '_');
nombre = strcat(workpath,'\Resultados\Live__',newChr,'.txt');
T = table(detect',clasi',doa',fecha',hora');
T.Properties.VariableNames = {'Detecciones' 'Clases' 'DOA' 'Fecha' 'Hora'};
writetable(T,nombre,'Encoding','UTF-8');


%Liberar canales de audio
release(audioInput);

strr = sprintf('FIN');
set(handles2.UsedByGUIData_m.text808,'String',strr);drawnow;
close(display);
PROFESIONAL;


if strcmp(modo,'profesional') %volver a abrir
    PROFESIONAL;
end
```

umbral_ini.m

```
function Th = umbral_ini(x,L,FL,Wvar)
%% Función para calcular umbral inicial del sistema en streaming -------- %
% Para ello se dividide la señal en segmentos, a los cuales se le calcula %
% la energía. Seguidamente obtiene la media de la energía y la desviación %
% estándar, el umbral es el resultado de la media más la desviación       %
% estándar multiplicados por su factor de ajuste.                         %
%                                                                         %
% Variables de entrada                                                    %
% - x: vector de señal                                                    %
% - L: tamaño en muestras de la ventana de energía                        %
% - FL: factor multiplicativo para número de muestras de umbral           %
% - Wvar: factor multiplicativo para la varianza en el umbral             %
%                                                                         %
% Variables de salida                                                     %
% - Th: umbral de detección                                               %
%                                                                         %
%.........................................................................%
%                                                                         %
%                                        Juan Francisco Martín Rodríguez %
%                                                                   2019 %
%_________________________________________________________________________%

N = length(x);
e      = zeros(1,N);
ind    = [1:L];
Nseg   = fix((N-L)/L);
for i=1:Nseg
   acu    = x(1,ind)*x(1,ind).'/Nseg;
   e(1,ind) = acu.*ones(1,L);
   ind    = ind + L;
end
media     = mean(e);
var       = std(e);
Th        = (media*FL+var*Wvar);
```

AI_detector.m

```
function [markini,markfin] = Al_detector(x,L,Th)
```

```
%% Función detector para el sistema en streaming ------------------------ %
% Esta función calcula las marcas de inicio y fin, para ello se crean     %
% segmentos de la señal a los cuales se le calcula su energía, luego según%
% la amplitud de la energía que se esté analizando en cada momento se     %
% compara con el umbral para obtener las marcas de inicio y fin. La marca %
% de inicio se colocará antes de superarse el umbral y la marca de fin se %
% colocará cuando la señal vuelva a su estado original, pudiendo tener    %
% varias marcas de inicio y de fin.                                       %
%                                                                         %
% Variables de entrada                                                    %
% - x: señal                                                              %
% - L: tamaño en muestras de la ventana de energía                        %
% - Th: umbral de detección                                               %
%                                                                         %
% Variables de salida                                                     %
% - markini: Marcas del principio de la detección o detecciones           %
% - markfin: Marcas del final de la detección o detecciones               %
%                                                                         %
%.........................................................................%
%                                                                         %
%                                        Juan Francisco Martín Rodríguez %
%                                                                   2019 %
%_________________________________________________________________________%
N = length(x);
e      = zeros(1,N);
ind    = [1:L];
Nseg   = fix((N-L)/L);
for i=1:Nseg
   acu    = x(1,ind)*x(1,ind).'/Nseg;
   e(1,ind) = acu.*ones(1,L);
   ind    = ind + L;
end
dete    = zeros(1,N);
bandera = 0;
j=0;
markini = [];
markfin = [];
for i=2:N-2
   if ((e(i)-e(i-1)) >= Th(end)) & bandera==0
      bandera=1;
      pri=i;
   end
   if bandera==1
      dete(i)=1;
   end
   if e(i)< Th(end) & bandera==1,
      bandera=0;
      fin=i;
      if (fin-pri)>5,
         j=j+1;
         markini(j)=pri;
         markfin(j)=fin;
      end
   end
end
```

CLASIFICACION

clasificacion.m

```
function [detectado, Yfit] =  clasificacion(config,y,norm,classifier,fs)
%% CLASIFICADOR ----------------------------------------------------------- %
% Función que realiza la clasificación en tiempo real                     %
%                                                                         %
% Variables de entrada                                                    %
% - config: Vector para elegir KNN o Random Forest                        %
% - y: Señal de entrada                                                   %
% - norm: Archivo para normalizar, contiene la media y la varianza de las %
%   características                                                       %
% - classifier: Clasificador ya entrenado cuya función es la de predecir  %
%   las clases                                                            %
```

```
% - fs: Frecuencia de muestreo                                          %
%                                                                       %
% Variables de salida                                                   %
% - detectado: Clase detectada                                          %
% - Yfit: Número de la clase detectada                                  %
%                                                                       %
% Proceso:                                                              %
%  - Extracción de características                                      %
%  - Normalización                                                      %
%  - Clasificación                                                      %
%                                                                       %
%.......................................................................%
%                                                                       %
%                                        Juan Francisco Martín Rodríguez %
%                                                                  2019 %
%_______________________________________________________________________%

%% ------------------------------ EXTRACCIÓN DE CARACTERÍSTICAS --------%
[features] = feature_extraction(y,fs);
data = features';
[~ , n_feats] = size(data);
data(:,n_feats+1)=0;
data(:,1:end-1)=bsxfun(@minus,data(:,1:end-1),norm.feats_mean);
data(:,1:end-1)=bsxfun(@rdivide,data(:,1:end-1),norm.feats_var);

if config.RF_classifier || config.kNN_classifier
    [Yfit] = predict(classifier,data(1,:));
    if Yfit == 1
        detectado = 'Trueno';
    elseif Yfit ==2
        detectado = 'Lluvia';
    elseif Yfit == 3
        detectado = 'Sonido Ambiente';
    end
end
end
```

feature_extraction.m

```
function [ features ] = feature_extraction(y, fs)
%% Extracción de Características ------------------------------------- %
% Función que realiza el cálculo de todas las features.                 %
% Su función es leer la señal junto a su frecuencia de muestreo, realizar %
% un control de canal seleccionando aquel con mayor energía y           %
% posteriormente un cambio en la frecuencia de muestreo si es necesario  %
% (a 16000 Hz) para que toda la extracción se procese a la misma         %
% frecuencia de muestreo. También se elimina la componente continua      %
% de la señal.                                                          %
%                                                                       %
% Variables de entrada:                                                 %
% - y: señal de entrada                                                 %
% - fs: frecuencia de muestreo                                          %
% Variables de salida:                                                  %
% - features (vector)                                                   %
% * spectral_features                                                   %
%                                                                       %
%.......................................................................%
%                                                                       %
%                                        Juan Francisco Martín Rodríguez %
%                                                                  2019 %
%_______________________________________________________________________%

% Selección de canal con mayor energia
[~,n]=size(y);

if n > 1
    Energy=sum(y.^2,1);
    y=y(:,Energy==max(Energy));
    y=y(:,1);
end
%% -------------------------- DC OFFSET ------------------------------ %
% - Eliminar la componente continua (DC) de la señal
y = y-mean(y);
```

```
% Remuestreo de la señal para realizar la extracción de características a
% la mínima frecuencia de muestreo
if fs ~= 16000
    Tx=(0:length(y)-1)./fs;
    fs=16000; % New fs
    y=resample(y,Tx,fs);
end

Nparam = 42;
%_____________________________________________________________________%
%% ---------------------- FEATURE EXTRACTION -------------------------- %
features=zeros(Nparam,1);
%% ---- SPECTRAL FEATURES ------------------------------------------------ %
[spectral_features]=spectral_feats(y,fs);
% -> spectral_features.mfcc:
%
%   Frame1     Frame2   Frame3      Framei
% [ MFCC1      MFCC1    MFCC1    ... MFCC1 ]
% [ MFCC2      MFCC2    MFCC2    ... MFCC2 ]
% [   .          .        .            .   ]
% [   .          .        .            .   ]
% [   .          .        .            .   ]
% [ MFCCn      MFCCn    MFCCn    ... MFCCn ]
%
% -> spectral_features.centroid:
%
%    Frame1    Frame2  Frame3       Framei
% [ CENTROID CENTROID CENTROID ... CENTROID ]
% -> spectral_features.skewness
%
%    Frame1    Frame2  Frame3       Framei
% [ SKEWNESS SKEWNESS SKEWNESS ... SKEWNESS ]
%
%     Valores medios
       features(1:40)=mean(spectral_features.mfcc,2);
       features(41)=mean(spectral_features.centroid,2);
       features(42)=mean(spectral_features.skewness,2);

%              Audio
%            [ MFCC1  ]
%            [ MFCC2  ]
%            [   .    ]
%            [   .    ]
%            [   .    ]
%            [ MFCCn  ]
%            [CENTROID]
%            [SKEWNESS]
end
```

spectral_feats.m

```
function [ spectral_features ] = spectral_feats(x,fs)
%% Extracción de Características Espectrales ---------------------------- %
% Función que realiza la extracción de características                    %
% (Feature Extraction) de todas aquellas características o "features"     %
% cuyo procedimiento de extracción se realice a través de la FFT,         %
% es decir, en el dominio de la frecuencia.                               %
%                                                                         %
% Variables de entrada:                                                   %
% - x: señal de audio                                                     %
% - fs: frecuencia de muestreo                                            %
% Variables de salida:                                                    %
% - características espectrales                                          %
% * MFCC                                                                  %
% * CENTROID + SKEWNESS                                                   %
%.........................................................................%
%                                                                         %
% - Proceso:                                          Variables           %
%    - Eliminar componente continua (DC)                                  %
%    - Pre-émfasis                                                        %
%    - División en frames                                                 %
%            * Tamaño de ventana:                          W              %
```

```
%             * Solape:                                      Porcentaje        %
%     - Enventanado                                                           %
%             * Ventana (hamming, hanning...):                  win           %
%     - DFT                                                                   %
%             * Tamaño FFT:                                     nfft          %
%                                                                             %
%     %% Extracción MFCC                                                     %
%     - Banco de filtros a la frecuencia Mel                                  %
%             * Filtros Mel :                                    M            %
%     - DCT                                                                   %
%             * Coeficientes DCT :                               C            %
%                                                                             %
%     %% CENTROID/SKEWNESS                                                   %
%                                                                             %
%                                          Juan Francisco Martín Rodríguez %
%                                                                       2019 %
%_____________________________________________________________________________%
%% -------------------- Parámetros de Configuración ---------------------- %
L=length(x); % - Longitud del Audio
W=round(fs*60e-3); % - Tamaño de ventana
win=hamming(W); % - Ventana
percent=50; % - Solapamiento
nfft=1024; % - Tamaño FFT
dB_blwmax=15; % - Control de Energía (dB) (dB a MAX) Por defecto: 10dB
% - Parámetros MFCC
M=35; % - Número de Filtros MEL
C=40; % - Número de Coeficientes de la DCT
%_____________________________________________________________________________%
if length(x)<W; x(W)=0; end
if length(x)<2*W; x=[x; x; x]; L=3*W; end
%% ------------------------- PRE-ÉNFASIS ------------------------------ %
% *Opcional
% Realza la alta frecuencia
h=[1 -0.9]; % Filtro FIR
x=filter(h,1,x);
%_____________________________________________________________________________%
%% ------------------------ División en Frames ------------------------- %
ovlap=percent*W/100; % - Muestras de Solape
m=1:ovlap:L-W; % - Desplazamiento
N=length(m); % - Número de ventanas
% - Vector de división en bloques
ind_vector=(0:W-1)'*ones(1,N) + ones(W,1)*m;
% - Señal dividida en bloques
x=x(ind_vector);
%_____________________________________________________________________________%
%% ---------------------- Control de la Energía de los Frames ----------- %
energy=sum(abs(x),1); % Envolvente
framesW = (energy >= max(energy)./(10.^(dB_blwmax/10)) & energy~=0);
x=x(:,framesW); % x [W,N]
N=sum(framesW);
%_____________________________________________________________________________%
%% ------------------------- ENVENTANADO ------------------------------ %
% Enventanado
xwin=bsxfun(@times,x,win);
%_____________________________________________________________________________%
%% --------------------- Magnitud Espectral DFT ------------------------- %
X=fft(xwin,nfft); % FFT
X=abs(X(1:nfft/2,:)); % Coger solo el valor absoluto de la mitad de la FFT
X2=X.^2; % Energía
%_____________________________________________________________________________%
%% CEPSTRAL: MFCCs ------------------------------------------------------ %
[mfcc] = mfcc_extraction( X2, nfft, fs , M, C);
%_____________________________________________________________________________%
%% CENTROID / SKEWNESS -------------------------------------------------- %
[centroid, skewness] = shape_extraction(X, nfft, fs, N);
%_____________________________________________________________________________%
% MFCC
spectral_features.mfcc=mfcc;
% Descriptores de Forma
spectral_features.centroid=centroid;
spectral_features.skewness=skewness;
end
```

shape_extraction.m

```
function [centroid, skewness]=shape_extraction(X,nfft, fs,N)
%% Extraccion de descriptores de forma ---------------------------------- %
% Función que realiza la extracción de las principales features de forma  %
% espectral:                                                              %
% - Centroid (Centro de gravedad del espectro)                            %
% - Spread (Expansión de la energía respecto al centroide)                %
% - Skewness (Asimetría)                                                  %
%                                                                         %
% Variables de entrada:                                                   %
% - X: Señal enmarcada (framed)                                           %
% - nfft: Tamaño del FFT                                                  %
% - fs: Frecuencia de muestreo                                            %
% - N: N frames                                                           %
%                                                                         %
% Variables de salida:                                                    %
% - Centroid                                                              %
% - Skewness                                                              %
%                                                                         %
%.........................................................................%
%                                                                         %
%                                                                         %
%                                          Juan Francisco Martín Rodríguez %
%_________________________________________________________________________%
if length(X(:,1))>nfft/2;
    X=X(1:nfft/2,:);
end
% Vector de frecuencias
f=(0:nfft/2-1).*fs/nfft;
%% --------------------------- CENTROID ---------------------------------- %
num=f*X;
% Frame1 Frame2 FrameN
% f*X =[f0 f1 f2 ... f_n]*[X_k0 X_k0 ... X_k0] =
% [X_k1 X_k1 ... X_k1]
% [ . . . ]
% [X_kn X_kn ... X_kn]
%
% Frame1 Frame2 Frame3 Frame4 FrameN
% f*X = [ fX1 fX2 fX3 fX4 ... fXN ]
%
denom=sum(X,1);
% Frame1 Frame2 Frame3 Frame4 FrameN
% X = [X_k0 X_k0 X_k0 X_k0 ... X_k0 ]
%     [X_k1 X_k1 X_k1 X_k1 ... X_k1 ]
%     [ .    .    .    .   .    .   ]
%     [X_kn X_kn X_kn X_kn ... X_kn ]
%
% Frame1 Frame2 Frame3 Frame4 FrameN
% sum_X = [X_1 X_2 X_3 X_4 ... X_N ]
%
centroid=num./denom; % Media
%_________________________________________________________________________%
f_nor=bsxfun(@minus,f'*ones(1,N),centroid);
%% ---------------------------- SPREAD ---------------------------------- %
num=sum((f_nor.^2).*X,1);
spread=num./denom; % Varianza
%_________________________________________________________________________%
%% --------------------------- SKEWNESS ---------------------------------- %
num=sum((f_nor.^3).*X,1);
m3=num./denom;
skewness=m3./(spread.^(3/2)); % Asimetría Espectral
%_________________________________________________________________________%
end
```

mfcc_extraction.m

```
function [ mfcc ] = mfcc_extraction( X2, nfft, fs , M, C)
%% Extracción de los Coeficientes Cepstrales a la Frecuencia Mel -------- %
%                                                                         %
% Variables de entrada:                                                   %
% - X2: Espectro X.^2                                                     %
% - nfft: tamaño de FFT size                                              %
% - fs: Frecuencia de muestreo                                            %
```

```
% - M: Número de filtros Mel                                              %
% - C: Número de coeficientes DCT                                         %
%                                                                         %
% Variables de salida:                                                    %
% - MFCC                                                                  %
%                                                                         %
% - Pasos a seguir para el cálculo de los mfcc:                           %
%   1) Pre-émfasis (hecho anteriormente)                                  %
%   2) Eliminar la componente continua (DC Offset) (hecho anteriormente)  %
%   3) Dividir la señal en frames (hecho anteriormente)                   %
%   4) Enventanado de la señal (hecho anteriormente)                      %
%   5) DFT (hecho anteriormente)                                          %
%   6) Banco de filtros de frecuencias Mel                                %
%   7) DCT                                                                %
%                                                                         %
%.........................................................................%
%                                                                         %
%                                        Juan Francisco Martín Rodríguez %
%                                                                   2019 %
%_________________________________________________________________________%
% ------------------------ Energía -> MFCC0 -------------------------------%
%                                                                         %
% - MFCC0 es aproximadamente la energía de la señal                       %
energy=10*log10(sum(X2,1));
%_________________________________________________________________________%
%% -------------------- BANCO DE FILTROS DE LA FRECUENCIA MEL ------------%
%
% Cargar o crear el Banco de Filtros Mel para un número específico de
% filtros
path_ini                = pwd;
if ~exist([path_ini,'\MelFB_',num2str(M) 'x' num2str(nfft/2) '.mat'],'file')
    mel=create_MelFB(fs, nfft, M);
    save([path_ini,'\MelFB_',num2str(M) 'x' num2str(nfft/2)], 'mel');
else
    load(['MelFB_' num2str(M) 'x' num2str(nfft/2)]) % cargar variable: mel
end
% Filtrado Mel
X2=log(mel*X2);
%.........................................................................%
%     BANCO DE FILTROS MEL              FFT
% [ 1_1 1_2 1_3 1_4 1_5 ... 1_Nfft]   [ 1 ] = Mx128 * 128x1 = Mx1
% [ 2_1 2_2 2_3 2_4 2_5 ... 2_Nfft]   [ 2 ]    _____________
% [ 3_1 3_2 3_3 3_4 3_5 ... 3_Nfft]   [ 3 ]   |             |
% [               .               ] * [ 4 ]   |  40xFRAMES  |
% [               .               ]   [ 5 ]   |_____________|
% [               .               ]   [ . ]
% [ M_1 M_2 M_3 M_4 M_5 ... M_Nfft]   [ . ]
%                                     [ . ]
%                                     [Nfft]
%_________________________________________________________________________%
%% ------------------------- DCT -> MFCCs ------------------------------- %
%
% Cargar o crear la Matriz DCT para un número específico de coeficientes DFT
% y filtros a la frecuencia Mel
%
if ~exist([path_ini,'\DCT_',num2str(C) 'x' num2str(M) '.mat'],'file')
    dct=create_DCT(C,M);
    save([path_ini,'\DCT_',num2str(C) 'x' num2str(M)],'dct');
else
    load(['DCT_' num2str(C) 'x' num2str(M)]) % Cargar variable: DCT (matriz)
end
%
% - - - - - - - - -> Paso final en la extracción MFCC <- - - - - - - - - -%
% (COEFICIENTES CEPSTRALES) -> DOMINIO HOMOMÓRFICO
mfcc=dct*X2;
mfcc(1,:)=energy;
%
%.........................................................................%
% DCT1 DCT2 DCT3           DCT_N   MEL_FFT
% [ 1_1 1_2 1_3 1_4 1_5 ... 1_M]   [ 1 ] = CxM * Mx1 = Mx1
% [ 2_1 2_2 2_3 2_4 2_5 ... 2_M]   [ 2 ]    _____________
% [ 3_1 3_2 3_3 3_4 3_5 ... 3_M]   [ 3 ]   |             |
% [           .               ] * [ 4 ]   | CxFRAMES(N) |
% [           .               ]   [ 5 ]   |_____________|
% [           .               ]   [ . ]
% [ C_1 C_2 C_3 C_4 C_5 ... C_M]   [ . ]
%                                  [ . ]
```

```
%                                   [Nfft]
%_____________________________________________________________________________%
end
```

LOCALIZACIÓN DE LA DIRECCIÓN DE PROCEDENCIA (DOA)

AudioArrayDOAEstimation.m

```
function [angulo, li, te, pa] = AudioArrayDOAEstimation(audioInput, sourceChoice,
handles2)
%% Función para la estimación de la dirección de llegada (DOA)----------- %
% En esta función se utilizan múltiples pares de micrófonos dentro de una %
% matriz lineal adquiriendo y procesando audio multicanal en tiempo real. %
%                                                                         %
% Variables de entrada                                                    %
% - audioInput: Señal de audio a procesar                                 %
% - sourceChoice: Entrada de señal por archivo de audio o por micrófonos  %
% - handles2: Para modificar la interfaz display                          %
%                                                                         %
%                                                                         %
% Variables de salida                                                     %
% - angulo: Ángulo calculado en grados                                    %
% - li, te, pa: Para eliminar plot de tipo line, text y patch             %
%                                                                         %
%.........................................................................%
%                                                                         %
%                                        Juan Francisco Martín Rodríguez %
%                                                                   2019 %
%_____________________________________________________________________________%


fs = 44100;
bufferLength = 7500;% Tamaño del buffer
micPairs = [2 1;3 2;1 3]; %Tres pares de micrófonos
micPositions = [[0.085; 0],[0; 0.15],[-0.085; 0]]; % Posiciones de los micrófonos
numPairs = size(micPairs, 1);

% Objeto auxiliar para adquirir la señal de entrada y reajustarla
% según el par de micrófonos seleccionado
preprocessor = audioexample.PairArrayPreprocessor(...
    'MicPositions', micPositions,...
    'MicPairs', micPairs,...
    'BufferLength', bufferLength);
micSeparations = getPairSeparations(preprocessor);

% El algoritmo principal es un correlador cruzado que se usa en conjunto
% con un filtro FIR interpolador para asegurar una mejor resolución DOA
XCorrelator = dsp.Crosscorrelator(...
    'Method', 'Frequency Domain');
interpFactor = 8;
b = interpFactor * fir1((2*interpFactor*8-1),1/interpFactor);
groupDelay = median(grpdelay(b));
interpolator = dsp.FIRInterpolator(...
    'InterpolationFactor',interpFactor,...
    'Numerator',b);

% Bucle para adquirir y procesar la señal de entrada en cada iteración
% y obtener la estimación del DOA

for idx = 1:2
    % Adquirir la señal de entrada multicanal
    switch sourceChoice
        case 'recorded'
            multichannelAudioFrame = audioInput();
        case 'live'
            multichannelAudioFrame = audioInput();
            %Factores de multiplicación para calibrar y que todos los
            %micrófonos reciban la misma señal
            load('factoresT.mat');
            multichannelAudioFrame = mult.*multichannelAudioFrame;
    end
```

```
    % Reorganizar la muestra adquirida en un array 4-D de tamaño
    % bufferLength x numBuffers x 2 x numPairs, donde 2 es el número de canales
    % por cada par de micrófonos
    bufferedFrame = preprocessor(multichannelAudioFrame);

    % Inicializar variables según los pares disponibles
    numBuffers = size(bufferedFrame, 2);
    delays = zeros(1,numPairs);
    anglesInRadians = zeros(1,numPairs);
    xcDense = zeros((2*bufferLength-1)*interpFactor, numPairs);

    % Bucle para realizar las operaciones con cada par disponible.

    for kPair = 1:numPairs
        % Primero, se estima el retardo entre micrófonos para
        % cada buffer de dos canales
        delayVector = zeros(numBuffers, 1);
        for kBuffer = 1:numBuffers
            % Correlación Cruzada
            xcCoarse = XCorrelator( ...
                bufferedFrame(:,kBuffer,1,kPair), ...
                bufferedFrame(:,kBuffer,2,kPair));

            % Interpolador para incrementar la resolución espacial
            xcDense = interpolator(flipud(xcCoarse));

            % Se extrae la posición del máximo, igual al retardo en unidades de tiempo
de muestra
            % incluyendo el retardo en grupo del filtro de interpolación
            % Se construye el vector de retardos para cada par
            [~,idxloc] = max(xcDense);
            delayVector(kBuffer) = ...
                (idxloc - groupDelay)/interpFactor - bufferLength;
        end

        % Se obtiene el vector de retardos definitivo para cada par
        delays(kPair) = median(delayVector);

        % Se pasa el retardo a ángulos (radianes)
        anglesInRadians(kPair) = HelperDelayToAngle(delays(kPair), fs, ...
            micSeparations(kPair));
    end

    % Se pasa de radianes a ángulos
    angleinGrados = (anglesInRadians*180)/pi;

    % Se elige el par con el mínimo ángulo ya que es el que más cerca está.
    [~, ind]= min(abs(angleinGrados));

    % Se obtiene el ángulo captado por ese par y se le suma una cantidad
    %determinadas para obtener el ángulo estimado
    if ind == 1 && angleinGrados(2)<0 && angleinGrados(3)>0
        angulo = round(56-angleinGrados(1));
    elseif ind == 1 && angleinGrados(2)>0 && angleinGrados(3)<0
        angulo = round(angleinGrados(1)+245);
    elseif ind == 2 && angleinGrados(1)<0 && angleinGrados(3)>0
        angulo = round(angleinGrados(2)+120);
    elseif ind == 2 && angleinGrados(1)>0 && angleinGrados(3)<0
        angulo = round(300-angleinGrados(2));
    elseif ind == 3 && angleinGrados(1)<0 && angleinGrados(2)>0
        angulo = round(-angleinGrados(3)+178);
    elseif ind == 3 && angleinGrados(1)>0 && angleinGrados(2)<0
        angulo = round(angleinGrados(3)+2);
    end

    %Si no existe la variable ángulo, se pasa a la siguiente iteración
    A = exist('angulo');
    if A == 1
        if angulo <0
            angulo = 360+angulo;
        end
        if angulo >360
            angulo = angulo - 360;
        end

    end
```

```
end


strr = sprintf('CLASIFICACION');
set(handles2.UsedByGUIData_m.text808,'String',strr);drawnow;
str = sprintf('Trueno');
set(handles2.UsedByGUIData_m.text69,'String',str);drawnow;
axes(handles2.UsedByGUIData_m.axes3);
imshow('thunder.jpg');

strr = sprintf('DIRECCION DE PROCEDENCIA');
set(handles2.UsedByGUIData_m.text808,'String',strr);drawnow;
str = sprintf(['Angulo: ', num2str(angulo), ' grados']);
set(handles2.UsedByGUIData_m.text4,'String',str);drawnow;
axes(handles2.UsedByGUIData_m.axes4);
%Polar plot
ang_Y = angulo;
vel_Y =1;
ang_rad = ang_Y* pi/180;
[x,y] = pol2cart(ang_rad,vel_Y);
axis off, axis equal
h1 = compass(y,x);
t = findall(gcf,'type','text');
ht1=findall(t,'String','0');
ht2=findall(t,'String','30');
ht3=findall(t,'String','60');
ht4=findall(t,'String','90');
ht5=findall(t,'String','120');
ht6=findall(t,'String','150');
ht7=findall(t,'String','180');
ht8=findall(t,'String','210');
ht9=findall(t,'String','240');
ht10=findall(t,'String','270');
ht11=findall(t,'String','300');
ht12=findall(t,'String','330');
set(ht1,'String','  90-E');
set(ht2,'String','60');
set(ht3,'String','30');
set(ht4,'String','0-N');
set(ht5,'String','330');
set(ht6,'String','300');
set(ht7,'String','270-O  ');
set(ht8,'String','240');
set(ht9,'String','210');
set(ht10,'String','180-S');
set(ht11,'String','150');
set(ht12,'String','120');
delete(t(end-7:end));
refreshdata(h1,'caller');
drawnow;

%Eliminar plot
li = findall(gcf,'type','line');
te = findall(gcf,'type','text');
pa = findall(gcf,'type','patch');
%----
end
```

[Esta página ha sido dejada intencionalmente en blanco]

[Esta página ha sido dejada intencionalmente en blanco]