ESCUELA SUPERIOR DE INGENIEROS DE SEVILLA

INGENIERIA EN TELECOMUNICACIÓN

DETECCION DE TRAMAS DE AUDIO CON SOUND BLASTER

Autor: ***RAFAEL J. VALLEJO PEREZ***

Tutor: ***JOSÉ RAMÓN CERQUIDES BUENO***

Octubre 2001

DETECCION DE TRAMAS DE AUDIO CON SOUNDBLASTER

I. MEMORIA DESCRIPTIVA.

1) INTRODUCCIÓN.

1.1) Generalidades.
1.2) Descripción del Proyecto.
1.3) Objetivos y alcance del Proyecto.

2) ANÁLISIS.

2.1) Análisis previo: Situación inicial.

2.1.1) Tramas de audio. Introducción.
2.1.2) Estructura de las tramas. El formato *WAV*.
2.1.3) Cabecera de un fichero *WAVE*.
2.1.4) Ventajas e inconvenientes.

2.2) Análisis de necesidades.

2.2.1) Métodos de comparación y detección de tramas de audio.
2.2.2) Convolución y correlación de señales. Propiedades.
2.2.3) Carga computacional de la función correlación. Métodos de bloque:

- Método de Solapamiento y Suma (Overlapp & Add)

- Método de Solapamiento y Almacenamiento (Overlapp & Save).

2.2.4) Ventajas computacionales de los métodos de bloque:

2.3) Análisis de herramientas disponibles.

2.3.1) Descripción del hardware necesario.
2.3.2) Herramientas software:

- Matlab 5.0. Descripción de aplicaciones audio.
- Wave Studio (Soundblaster).

3) COMPARACION DE TRAMAS DE AUDIO.

3.1) Introducción.
3.2) Características de la aplicación *'wavread.m'* de Matlab 5.0.
3.3) Características y particularidades del método de comparación.

- Implementación del algoritmo Overlap & Add.
- Longitud de los bloques.
- Señal correlación resultante: coeficientes de correlación útiles.
- Normalización de los coeficientes de correlación.

3.4) Evaluación del método. Resultado final.

4) OPTIMIZACION DE PARAMETROS EN LA COMPARACION DE TRAMAS DE AUDIO.

4.1) Introducción.
4.2) Optimización de parámetros.

IV. RUTINAS EMPLEADAS. CÓDIGOS.

V. BIBLIOGRAFIA.

I. MEMORIA DESCRIPTIVA.

PROYECTO DE COMPARACION Y DETECCION DE TRAMAS DE AUDIO CON TARJETA SOUND BLASTER APLICADO A LA RADIODIFUSION COMERCIAL

1) INTRODUCCION:

Hoy en día la publicidad y el marketing han cobrado una importancia desproporcionada, están presentes en todo momento y en todos los sectores, pero los medios por excelencia a través de los cuales se lleva a cabo un "bombardeo" de anuncios para el consumidor son la radio y la televisión. Debido a la relevancia de dichos anuncios en la economía de la sociedad actual, y por lo tanto a los ingresos que se generan a través de ellos, hay un complejo proceso en torno al mundo de la publicidad. En este proceso las emisiones de dichos anuncios son controladas de una forma muy estricta, ya sea la emisión en sí como la franja horaria en que se emite.

Este proyecto se centra en este campo, tratando de ser un medio o dispositivo de control de la emisión de anuncios. A continuación veremos de una forma breve y rápida cuál es el problema y la motivación de este proyecto, describiendo sus directrices principales, así cómo su ámbito de aplicación y su alcance.

1.1) Generalidades:

El reconocimiento de anuncios de radio (FM y AM) o televisión por computador en tiempo real es un amplio campo de investigación y desarrollo hoy en día. Existen multitud de empresas de marketing que se encargan de controlar las emisiones de los anuncios, y estas empresas buscan aplicaciones capaces de

comparar y reconocer tales anuncios, con el objetivo de comprobar el correcto cumplimiento de los contratos con las diferentes emisoras.

Mientras en emisoras importantes es habitual guardar un registro de las transmisiones efectuadas, esto no ocurre en emisoras de carácter local, lo que obliga a realizar un control en tiempo real de la programación emitida. En la mayoría de las ocasiones en que se varía la emisión de un anuncio no se debe a la negligencia de estas emisoras, sino a los problemas de tiempo que tienen, entrevistas, programas en directo..., esto causa que los anuncios muchas veces sean desplazados de la franja horaria concertada y ubicados en una franja diferente, en ocasiones en programas con una cuota de "share" o de audiencia diferente a la contratada.

El desarrollo de una aplicación de estas características se puede abordar desde diferentes niveles de precisión y complejidad:

- Detección y reconocimiento de la secuencia de audio.
- Detección y reconocimiento de la secuencia de vídeo.
- Ambos procesos conjuntos.

Evidentemente la mejor y más completa aplicación será aquella que reconozca tanto el sonido como las imágenes del anuncio publicitario, pero la detección y reconocimiento de secuencias de vídeo es un proceso complejo que consume muchísimos recursos, y que hace que sea muy costoso trabajar en tiempo real cuando no se dispone de los dispositivos adecuados (microprocesadores de alta velocidad, DSPs programables, ...). Además no tiene sentido trabajar con secuencias de vídeo cuando se trata de reconocer anuncios en las emisoras de FM y AM. Las señales de audio son más rápidas y fáciles de procesar, y al trabajar con ellas podremos monitorizar anuncios comerciales tanto en emisoras de televisión como en emisoras de radio.

En este proyecto se desarrolla una aplicación que detecta y reconoce únicamente secuencias de audio, debido a ello se puede utilizar para reconocer anuncios en ambos medios de radiodifusión. Además nos dará algunos datos de interés para el anunciante, como son la hora y fecha de emisión cada vez que se detecta el anuncio y una tabla en la que se recogen todas las detecciones realizadas.

Se hace notar que en ocasiones se emiten diferentes versiones del mismo anuncio, y una de las características de estas versiones es la variabilidad de la duración de la emisión, que se realiza con el objetivo de reducir el costo por parte de los anunciantes. Para este tipo de anuncios habría que realizar un estudio individual, ya que son anuncios de distinta duración y por lo tanto se considerarán anuncios distintos, aunque sean de la misma marca o producto.

1.2) Descripción del proyecto:

Como ya hemos adelantado el proyecto consiste en una aplicación cuya finalidad es la detección y reconocimiento de anuncios en FM y AM, aunque también se puede extender a la detección de anuncios de TV. El método utilizado es el de comparar la señal proveniente de la radio con el anuncio que tenemos guardado en memoria y decidir si se trata del anuncio en cuestión en función del grado de similitud que hay entre ambas. Para ello se ha implementado un algoritmo de comparación que va tomando fracciones de señal de duración muy corta de la emisora que queramos monitorizar, a través de la tarjeta de sonido Sound Blaster, y las va comparando con el anuncio que tenemos en memoria.

Para llevar a cabo dicha comparación nos ayudamos de una de las principales propiedades de la función correlación, ya sabemos que la correlación cruzada de dos señales mide cuán parecidas son éstas. Este método implica una alta carga computacional, siendo la comparación un proceso lento que dificulta la posibilidad de trabajar en tiempo real, sin embargo hay técnicas que hacen que se reduzca dicha carga computacional considerablemente. Utilizando una de estas técnicas se

consigue trabajar en tiempo real, y aunque no podamos hacerlo con varios anuncios y emisoras simultáneamente la eficiencia de este método hace que sea el más apropiado para comprobar la similitud de dos señales.

Además los coeficientes de la función correlación normalizada varían entre cero y uno, dependiendo del grado de similitud entre las muestras de las señales a comparar. Esta característica no sólo hace que dichos coeficientes sean fáciles de interpretar y manejar a la hora de implementar el algoritmo de detección, sino que nos es de gran ayuda para el análisis estadístico del proceso de reconocimiento de patrones de audio, sin el cual nos veríamos obligados a hacer multitud de pruebas para determinar cuál es el formato de la señal de audio más apropiado: frecuencia de muestreo, número de bits por muestra, tiempo de captura de señal, ...

Por lo tanto la correlación entre las dos tramas de audio, el anuncio y el trozo de señal capturado por la Sound Blaster, es una técnica con grandes ventajas para el reconocimiento de anuncios:

- Es eficiente.
- Facilita la implementación del algoritmo de detección.
- Hace factible un análisis estadístico.
- Hardware necesario: PC y tarjeta de radio FM.

De ahí que esta sea la técnica escogida para llevar a cabo la aplicación que nos ocupa.

El análisis estadístico del que hablamos consiste en encontrar una función de distribución cuyas características sean tales que podamos ajustar el proceso de correlación de dos señales a dicha función, de forma que se pueda realizar un estudio estadístico. Dicho de otro modo, buscamos una función de distribución cuyo dominio varíe entre 0 y 1, igual que el rango de valores de la función correlación, de esta manera se podrá representar estadísticamente el conjunto de

pruebas o detecciones, tanto positivas como negativas, que se realicen para cada formato de la señal que proviene de la tarjeta Sound Blaster. Así tendremos una representación estadística para cada pareja *frecuencia de muestreo-tiempo de captura*, que es la que define el formato de la señal, pues el número de bits por muestra se considerará constante e invariable, y con dicha representación se obtendrán datos, como la probabilidad de falsa alarma y la probabilidad de pérdida, que nos ayudarán a elegir el formato (*frecuencia de muestreo- tiempo de captura*) más apropiado para la detección.

Una vez escogido el formato de la señal de audio sólo nos queda aplicarlo y hacer un algoritmo decisor que sea robusto, o sea que las probabilidades de falsa alarma y de pérdida lo más pequeñas posible.

A lo largo de este documento se hablará de *detección positiva* y *negativa*, entendiéndose por detección el proceso de comparación de la trama de audio obtenida de la radio con el anuncio grabado en memoria, así como la posterior decisión de si dicha trama coincide con algún trozo del anuncio o no, y por lo tanto si forma parte de él. Si es coincidente entonces hablaremos de *detección positiva*, si no la detección será *negativa*. Hay que dejar claro que una *detección positiva* no implica que en ese momento se esté emitiendo el anuncio buscado, pues hay una pequeña probabilidad de error que hace que a veces haya *falsas alarmas* (cuando se obtiene una *detección positiva* pero no se trata de nuestro anuncio) o *pérdidas* (cuando se está emitiendo nuestro anuncio y hay una *detección negativa*). Siempre intentaremos llevar a cabo el mayor número posible de detecciones o pruebas cuando se está emitiendo el anuncio y en función de todas ellas se decidirá si es el anuncio en cuestión o no.

1.3) Objetivos y alcance del proyecto:

En la actualidad el estudio de las señales de audio está muy extendido y se aplica en diversidad de campos. El papel del audio en el contexto de las aplicaciones multimedia que incluyen vídeo es cada vez más importante. Muchos

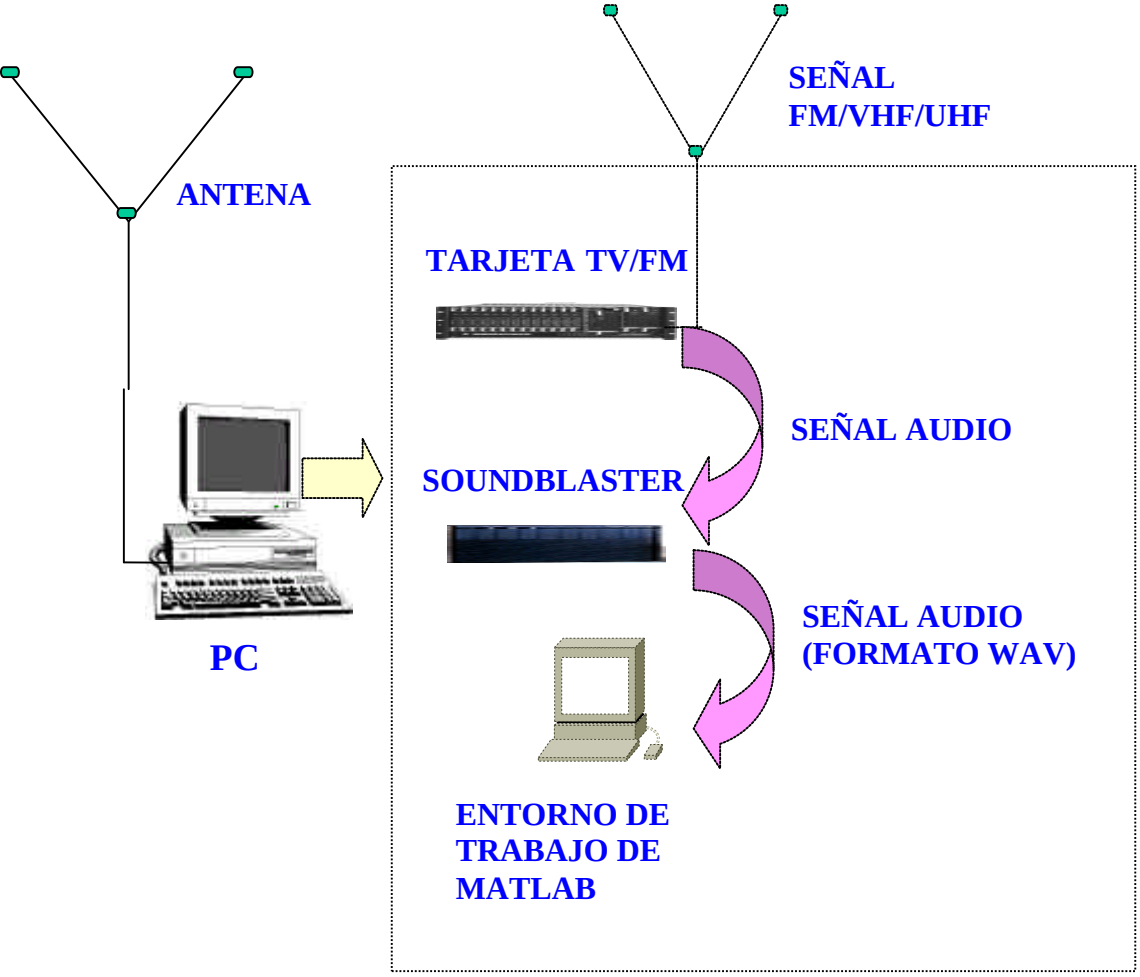
esfuerzos en este área están enfocados hacia el estudio del audio que contiene alguna estructura de información semántica incorporada, como es el caso de noticias de radiodifusión y publicidad subliminal. Otros lo enfocan desde otro punto de vista, y estudian el audio que contiene un solo tipo de sonido, ya sea música, discurso, ..., entre ellos el más extendido es el de reconocimiento de voz.

En otras muchas investigaciones se trabaja con señales de audio mezcladas, en las que nos encontramos con una mezcla de música, discurso, ruído de fondo, efectos sonoros, etc. La segmentación del audio mezclado tiene aplicaciones en la detección de bandas de publicidad para que el vídeo deje de grabar automáticamente, en sistemas de recuperación de audio, ...

Asímismo, este proyecto se basa en el estudio de señales audio formadas por componentes que proceden de diversas fuentes, pues es así como aparecen normalmente en radio y televisión. Sin embargo no estamos interesados en desgranar la señal en cada una de sus componentes y estudiarlas por separado. La señal de audio tiene unas propiedades y características que hacen que los sonidos sean casi inconfundibles, aún cuando son resultado de la suma de varias fuentes simultáneas. De esta manera no es necesario separar dichas fuentes para reconocer y detectar la emisión de patrones de audio conocidos a priori.

Por lo tanto nuestro único objetivo es, como ya hemos dicho, el reconocimiento de anuncios comerciales que se emiten tanto por radio como por televisión, y no nos vamos a preocupar en absoluto de la segmentación y clasificación de la señal de audio.

A continuación tenemos un diagrama en el que se muestra, sin entrar en detalles, el método utilizado para llevar a cabo nuestra tarea:



En el diagrama se puede observar que la señal de audio tras pasar por la tarjeta Sound Blaster adquiere un cierto formato, llamado *formato WAV*, el cual facilita enormemente el procesado de la señal de audio bajo el entorno de *Windows*, y en definitiva bajo el entorno de trabajo de la aplicación *Matlab.*

La versatilidad y capacidad de la tarjeta Sound Blaster para convertir y procesar los distintos formatos en que nos podemos encontrar la señal de audio hace de ella una candidata ideal para desarrollar este proyecto.

2) ANALISIS:

A continuación se va a realizar un estudio de la situación de la que partimos y a dónde queremos llegar, dando algunas nociones básicas sobre señales audio, de forma que se vaya comprendiendo poco a poco el alcance de este proyecto.

2.1) Análisis previo. Situación inicial:

Se va a realizar un análisis más o menos generalizado de las señales de audio, para posteriormente centrarnos en las señales con formato *WAV*, cómo se estructuran, clasificaciones, información contenida en la cabecera de las tramas, etc. De esta forma, conoceremos mejor el tipo de señal con la que trabajamos y tendremos menos dificultades cuando llegue el momento de su procesado.

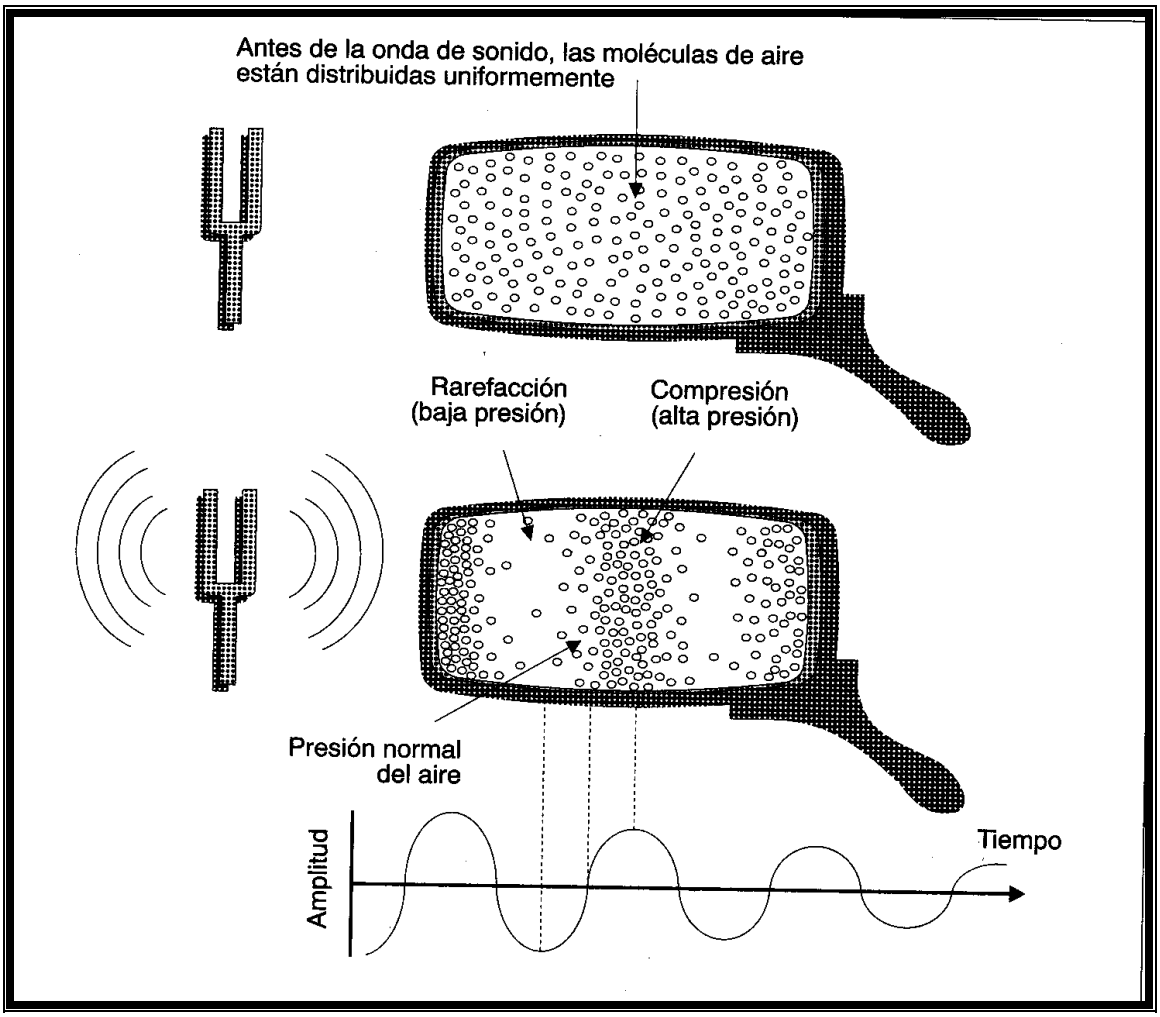
2.1.1) Señales audio. Introducción:

Este capítulo se ha concebido con el fin de proporcionar unos principios rápidos pero esenciales que permitan entender los conceptos y las tecnologías del audio (sin entrar en detalles de carácter analítico). Una vez aprendidos estos conceptos será posible trabajar con la computadora o con un dispositivo de audio con un cierto grado de confianza.

- **La naturaleza del sonido:**

El sonido es una vibración que se propaga a través del aire, gracias a que las moléculas del aire transmiten la vibración hasta que llega a nuestros oídos. Se aplican los mismos principios que cuando se lanza una piedra a un estanque, la perturbación de la piedra provoca que el agua se agite en todas las direcciones hasta que la amplitud de las ondas es tan pequeña que dejan de percibirse. La figura muestra las vibraciones físicas de un diapasón que ha sido golpeado. Las

vibraciones del diapasón fuerzan a las moléculas de aire a agruparse en regiones de mayor y menor densidad, dando lugar a que la presión del aire aumente o disminuya instantáneamente. El diapasón es un excelente ejemplo de fuente de sonido por dos razones: la primera es que puede observarse el movimiento de vaivén de sus brazos mientras se escuchan los resultados de esta vibración; la segunda es que el diapasón vibra a una frecuencia (vibraciones por segundo) constante hasta que toda su energía se ha disipado en forma de sonido. Una perturbación que viaja a través del aire se denomina onda y la forma que adopta esta se conoce como forma de onda.



- **Características de una onda sencilla:**

La forma de onda del diapasón es la más sencilla de las formas de onda, denominada onda sinusoidal.

La frecuencia de sonido se mide en unas unidades denominadas Hertzios (Hz). Un sonido que vibra una vez por segundo tiene una frecuencia de 1 Hz. Las frecuencias se escriben normalmente en Kilohercios (kHz), unidad que representa 1.000 Hz. Una persona joven con unos oídos saludables puede oír sonidos que estén en el rango de los 20 a los 20.000 Hz (20 kHz).

En la figura se observa cómo la amplitud de la onda disminuye a medida que el sonido se aleja de su fuente, extendiéndose en todas las direcciones. (La figura exagera la rapidez con la que la onda disminuye en el aire).

El oído y un micrófono incorporado en la tarjeta de sonido se comportan de manera similar. Ambos transforman pequeñas variaciones en la presión del aire en señal eléctrica que puede ser comprendida y almacenada por sus respectivos "cerebros" (ya sea el humano o la CPU de la computadora). Esta señal eléctrica puede ya ser guardada, manipulada o reproducida mediante los medios electrónicos adecuados.

- **Amplitud:**

La medida de la amplitud de una onda es importante porque informa de la fuerza, o cantidad de energía, de una onda, que se traduce en la intensidad de lo que oímos, su unidad de medida es el decibelio. Un decibelio, abreviado como dB, es una unidad de medida de la fuerza de la señal y es útil en la comparación de la intensidad de dos sonidos. La sensibilidad del oído humano es extraordinaria, con un rango dinámico o variación en intensidad muy amplio. La mayoría de los oídos humanos pueden capturar el sonido del murmullo de una hoja y, después de haberse sometido a ruidos explosivos como los de un avión, siguen funcionando.

Lo que es sorprendente es que la fuerza de la explosión en un avión es al menos 10 millones de veces mayor que el murmullo que una hoja produce con el viento.

El oído necesita un porcentaje elevado de variaciones en la fuerza de un sonido para detectar un cambio en la intensidad percibida, lo que significa que la sensibilidad del oído a la fuerza del sonido es logarítmica, de manera que el decibelio, unidad de medida logarítmica, es la elección más adecuada para medir la fuerza del sonido. El aspecto práctico de la amplitud es que un incremento de sólo 3 dB duplica la intensidad de un sonido. Por ejemplo, un sonido con 86 dB tiene, el doble de fuerza que un sonido con 83 dB y cuatro veces más que un sonido con 80 dB. Desde la perspectiva de nuestra percepción de la intensidad, un incremento de 3 dB, que da lugar a que se duplique la fuerza, provoca que el sonido se perciba sólo ligeramente más alto. Es necesario un aumento en 10 dB para que nuestros oídos perciban un sonido con el doble de intensidad.

- **Rango dinámico:**

La calidad de los sonidos musicales grabados no es demasiado importante, ya que nunca son comparables a los reales. La razón principal es que el equipo estéreo no puede duplicar el rango dinámico completo de una orquesta o de un concierto de rock. Una orquesta puede alcanzar los 110 dB en su clímax y en el punto más suave bajar hasta los 30 dB, dando lugar a un rango dinámico de 80 dB. Este rango es superior al rango dinámico de un sistema estéreo típico y, de hecho, superior a la capacidad de grabación de medios tales como un disco de vinilo y una cinta de audio.

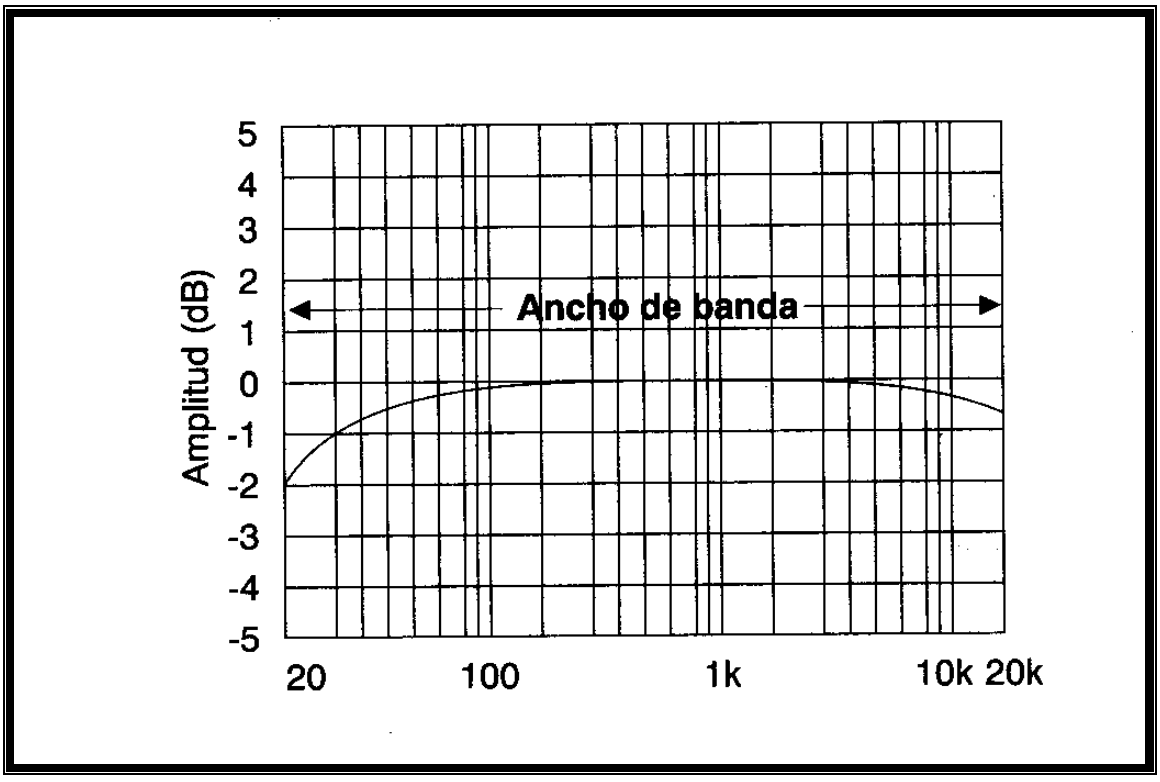
- **Ancho de banda:**

Profundizamos ahora en aspectos prácticos, como el rango de frecuencia con el que es capaz de trabajar un reproductor CD, nuestro oído o nuestra voz. El ancho de banda es muy importante para disfrutar de la música y es un criterio básico a la hora de seleccionar un equipo de audio para utilizar con la tarjeta de sonido. Por

ejemplo, el ancho de banda teórico de la radio FM es aproximadamente tres veces el ancho de banda de la radio AM, por lo que la FM será capaz de reproducir frecuencias que no entran dentro del campo de trabajo de la AM.

A menudo el ancho de banda se simboliza mediante un único número cuando la frecuencia baja está bastante próxima a cero. Por ejemplo, el ancho de banda de una voz femenina se sitúa en torno a los 9 kHz, aunque realmente puede estar en el rango que va desde los 200 Hz hasta los 9 kHz.

Existe una medida estándar para definir el ancho de banda: el rango de frecuencias sobre el que la amplitud de la señal no difiere del promedio en más de 3 dB, es decir la diferencia de las frecuencias en la que se produce una caída de 3 dB, ya es el punto donde su amplitud cayó a la mitad, y éste es el mínimo cambio en la fuerza de la señal que puede ser percibido como un cambio real en la intensidad por la mayoría de los oídos.



- **Ancho de banda del sistema:**

Es importante tener en cuenta que el ancho de banda de un equipo de sonido depende del enlace más débil del canal, que normalmente no es la tarjeta de sonido. La calidad del sonido producido por la computadora refleja el esfuerzo de muchas componentes, y la salida no será mejor que la interpretación del miembro menos capacitado de un grupo. En el caso del sistema de sonido de la computadora, una señal debe pasar por muchas fases de transformación de audio y por diferentes dispositivos. Por ejemplo, consideremos el sonido grabado mediante un micrófono y que luego es reproducido. La tarjeta de sonido transforma el sonido recogido del micrófono en una señal eléctrica que, posteriormente, se transforma en audio digital y se almacena en disco. El audio digital del disco es transformado de nuevo en una señal eléctrica y reproducido a través de los cascos o de los altavoces. El ancho de banda efectivo del sistema de sonido está limitado por el dispositivo con el ancho de banda más estrecho de todos los dispositivos que procesan el sonido.

El enlace más débil en grabación suele ser el micrófono, que tiene probablemente un ancho de banda aproximadamente de 12 kHz.

- **Ruido:**

Del mismo modo que perturban los ruidos y ecos en una habitación, también puede generarse ruido y distorsión en la tarjeta de sonido, en los altavoces y en el micrófono. El ruido (sonidos aleatorios que transforman y enmascaran el sonido deseado) se mide también en decibelios. Dado que es tan poco probable disponer de un entorno de audio digital en perfecto silencio, lo que interesa realmente es saber la cantidad de ruido en relación con la señal que se introduce en el equipo de sonido, especialmente en la tarjeta de sonido. La fuerza de la música, del habla o de cualquier otro sonido, comparada con la fuerza promedio del ruido, se conoce como relación señal/ruido. A medida que aumenta la relación s/n, es mejor el trabajo realizado en grabación. La relación señal ruido de una tarjeta digital sencilla es del orden nada despreciable de 85 dB. Esto significa que la fuerza de la

señal es 85 dB mayor que la fuerza del ruido. Una relación de 70 dB se considera válida para propósitos musicales y una relación de 65 dB está en el límite de aceptación.

- **Grabación y reproducción de audio. Bases del audio digital:**

Antes de que la computadora pueda grabar, manipular y reproducir sonido, debe transformarse el sonido de una forma analógica audible a una forma digital aceptable por la computadora, mediante un proceso denominado conversión analógica-digital (ADC).

Una vez que los datos de sonido se han almacenado como bytes en la computadora, puede hacerse uso de la potencia de la CPU de la computadora para transformar este sonido de miles de modos. Con el software adecuado es posible, por ejemplo, añadir reverberación o eco a la música o a la voz, pueden eliminarse trozos de sonido grabado, pueden mezclarse archivos de sonido, ajustarse el tono de la voz de manera que no pueda reconocerse y muchas cosas más.

Finalmente, cuando se está dispuesto a escuchar el resultado, el proceso de conversión digital-analógica (DAC) transforma de nuevo los bytes de sonido a una señal eléctrica analógica que emiten los altavoces.

- **Muestreo. Conversión analógica-digital y digital-analógica:**

Comenzaremos con la captura del sonido haciendo uso del micrófono. Cuando las ondas de sonido llegan al micrófono, el movimiento mecánico se traduce en una señal eléctrica. Esta señal se denomina señal analógica porque es una señal continua en el tiempo, análoga al sonido original.

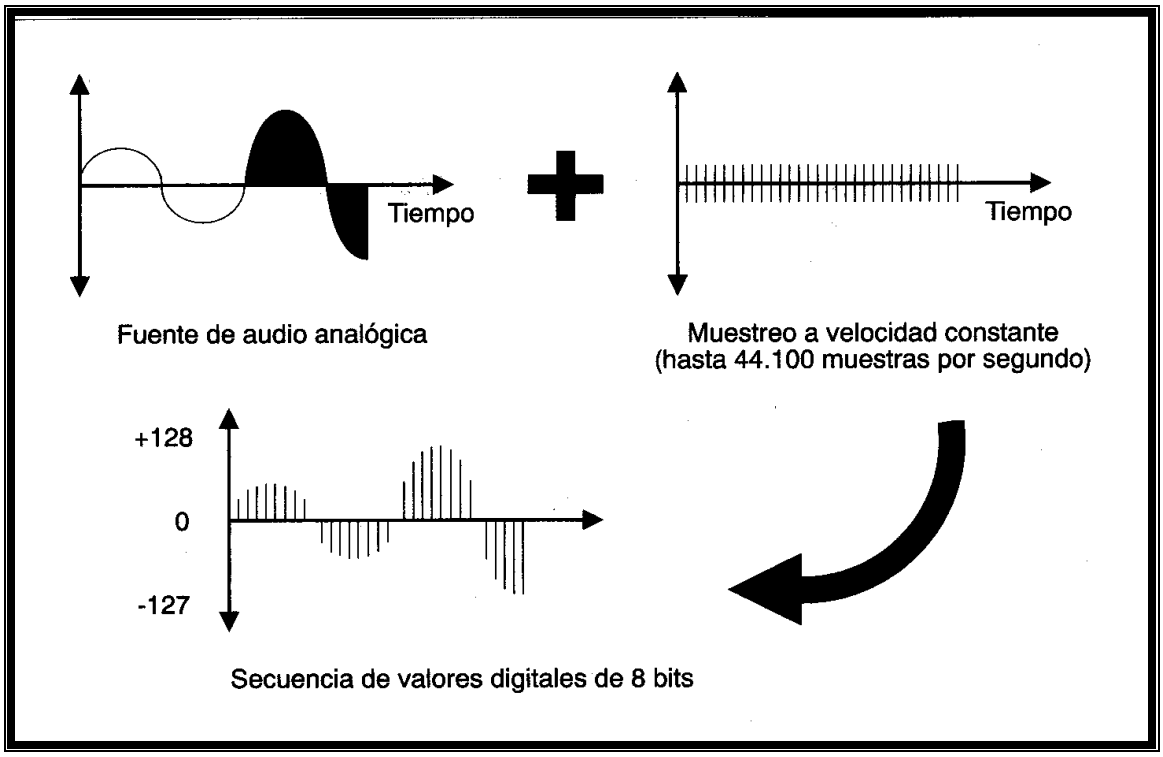
CONVERSION ANALOGICA-DIGITAL (ADC):

Dada una señal analógica, se van tomando valores discretos de su amplitud a intervalos de tiempo pequeños, evidentemente será más fiable la reproducción cuantas más muestras por segundo se tomen. A estos valores obtenidos se les asigna un valor digital que el computador puede entender y procesar como se requiera.

A cada muestra obtenida se le asigna un equivalente binario ya que es en este sistema en el que trabajan los computadores, su unidad de información es el bit. Un bit sólo puede tomar dos posibles valores, "1" ó "0", es lógico pensar que será necesario ampliar esta unidad de información para así poder asignar a cada valor de muestra tomada un equivalente binario. Por esta razón y dependiendo de la fidelidad con que queramos trabajar podemos utilizar palabras de 8 ó 16 bits, pudiendo obtener así 256 ó 65536 combinaciones distintas y obtener mayor resolución.

CONVERSION DIGITAL-ANALOGICA (DAC):

El proceso inverso es mucho menos complejo ya que solo se trata de ir poniendo los valores de las muestras en el mismo orden que fueron tomados y unos filtros electrónicos se encargan de convertir esa señal resultante de valores discretos en una señal analógica.



- **Velocidad y tamaño de muestra:**

La fidelidad, terminología empleada por los entusiastas del audio para expresar la exactitud en la réplica de la música original del sonido audio digital, depende de la selección de la correcta frecuencia de muestreo y del correcto tamaño de muestra, siendo este último el número de bytes utilizados para almacenar cada muestra.

FRECUENCIA DE MUESTRA:

La frecuencia de muestra (también denominada frecuencia de muestreo) debe ser lo suficientemente alta para que los sonidos de alta frecuencia, como el sonido del cristal de una copa de vino o el del arqueo de un violín, puedan recogerse con precisión. Según el teorema de Nyquist, es posible repetir con exactitud una forma de onda si la frecuencia de muestreo es como mínimo el doble de la frecuencia de la componente de mayor frecuencia. La frecuencia más alta que puede percibir el oído humano está cercana a los 20 kHz, de modo que la frecuencia de muestreo de

44.1 kHz de las tarjetas de sonido es más que suficiente. Este valor es el utilizado hoy en día por los reproductores de audio CD.

Los archivos de audio digital pueden grabarse seleccionando la frecuencia de muestreo. A medida que aumenta la frecuencia de muestreo, aumenta la calidad del sonido. Por ejemplo, una velocidad de 6.000 Hz (6.000 muestras por segundo) es buena para una voz masculina típica, pero no lo es para una voz femenina típica, que tiene componentes con una frecuencia más alta. Una frecuencia de muestreo de 8.000 Hz proporciona una grabación de la voz femenina de mayor calidad. La siguiente tabla ofrece una lista de algunas tarjetas de sonido Sound Blaster y de sus frecuencias de muestreo:

Tarjeta Grabación-Reproducción

Sound Blaster 16	44.100 (mono o estéreo)	44.100 (mono o estéreo)
Sound Blaster Pro	22.050 (estéreo) ó 44.100 (mono)	22.050 (estéreo) ó 44.100 (mono)
Sound Blaster 2.0	15.000 (mono)	44.100 (mono)
Sound Blaster 1.0/1.5	13.000 (mono)	23.000 (mono)

La Sound Blaster 16 puede grabar en estéreo, grabando hasta 44.100 muestras por segundo, con un canal izquierdo y otro derecho que producen una frecuencia de muestreo combinada de 88.200 muestras por segundo. Las tarjetas Sound Blaster Pro y la Sound Blaster 16 son capaces también de trabajar en estéreo con una velocidad máxima de reproducción de 22.050. Ambas tarjetas, la Sound Blaster y la Sound Blaster Pro, toman muestras de sonido de 8 bits (1 byte); cada medida consume 1 byte de almacenamiento de la memoria de la computadora o del disco. La Sound Blaster 16 maneja muestras de 16 bits (2 bytes), emitiendo voz y música con una fidelidad equivalente a los reproductores CD actuales.

Existen varias razones para no utilizar las frecuencias de muestreo más altas. En primer lugar, las frecuencias de muestreo altas necesitan gran capacidad de almacenamiento.

TAMAÑO DE MUESTRA:

El tamaño de muestra es la otra componente de mayor influencia en la fidelidad del audio digital. Las tarjetas de sonido de 16 bits ofrecen la posibilidad de elegir entre un tamaño de muestra de audio digital de 8 bits (1 byte) o de 16 bits (2 bytes).

El tamaño de muestra controla el rango dinámico que puede grabarse. Por ejemplo, las muestras de 8 bits limitan el rango dinámico a 256 pasos (rango de 50 dB). Por el contrario, una muestra de 16 bits tiene un rango dinámico de 65.536 pasos (rango de 90 dB), lo que supone una mejora sustancial. El oído humano percibe todo un mundo de diferencias entre estos dos tamaños de muestra. Los oídos son más sensibles a la detección de diferencias en el tono que en la intensidad, pero son aún más sensibles a la fuerza del sonido. Los oídos humanos, que están acostumbrados a detectar sonidos con variaciones de varios órdenes de magnitud en la fuerza, perciben el sonido de 8 bits con un tono apagado o desafinado si se compara con el sonido de audio digital de 16 bits.

COMPROMISOS EN EL MUESTREO:

Se podría asumir que todo lo que hay que hacer para obtener buen sonido es grabar a la velocidad límite de 44,1 kHz con muestras de 16 bits (2 bytes). El único problema que aparece si se graba en estéreo, tomando muestras simultáneamente en los canales izquierdo y derecho a 44,1 kHz, es que una muestra de sonido de un minuto necesita un espacio para almacenarse de 10,58 MB.

Lo aconsejable es usar la frecuencia de muestreo más baja posible. Por ejemplo, supongamos que planeamos grabar una conversación telefónica. El ancho de banda de un teléfono es de sólo 3 kHz. De acuerdo con el teorema de Nyquist, la grabación será acertada si la frecuencia de muestreo es de 6 kHz o mayor.

Cuando se elige la frecuencia de muestreo, también hay que considerar el ancho de banda de todo el sistema. Por ejemplo, no existe ningún problema en la grabación de audio digital a 44,1 kHz si el micrófono utilizado funciona a 12 kHz y la fuente de sonido es una voz masculina grave que no supera los 7 kHz.

- **Compresión de audio digital:**

Como hemos visto anteriormente, cada muestra tomada es almacenada en el ordenador ocupando uno o dos bytes de memoria dependiendo de si se ha tomado una resolución de 8 ó 16 bits. Esto conlleva emplear grandes espacios de disco para almacenar estos ficheros de sonido. Se han desarrollado muchos formatos de archivos comprimidos que permiten realizar grabaciones de alta calidad sin necesidad de utilizar tanto espacio de disco. Estas técnicas suelen incluir un método para codificar secuencias largas de bytes repetidos. Evidentemente existe una contrapartida y es que una vez comprimido el archivo de sonido, este no puede ser editado para su modificación.

- **Formatos de archivos de sonido:**

A continuación se muestran algunos de los formatos más utilizados para archivar audio:

- **Archivos AU:** es el formato audio estándar en ordenadores Sun. Por lo general son de 8 bits y poseen menor calidad que otros formatos de sonido.

- **IFF:** es común en ordenadores Mac. Pueden ser de 8 ó 16 bits, soportan frecuencias de muestreo de hasta 44,1 kHz y por lo general tienen una buena calidad de sonido. IFF-C es un formato de archivo IFF comprimido.

- **Audio WAV:** es el formato propio de Windows. Pueden ser de 8 ó 16 bits con índices de muestreo de 8 kHz, 11,025 kHz, 22,05 kHz o 44,1 kHz y por lo general tienen buena calidad de sonido.

- **VOC:** similares a los archivos WAV con la salvedad de que los VOC incluyen unos marcadores de sincronización que los programas de presentación multimedia pueden utilizar para sincronizar la reproducción de archivos VOC con imágenes o vídeos u otros sonidos.

- **RA:** (real audio). Es un formato utilizado para reproducir sonidos en Internet, y ofrece unas características especiales que lo hacen muy interesante en este campo. Si utilizáramos archivos del tipo *wav*, podemos imaginarnos el tiempo necesario que haría falta para acceder a un archivo cuya capacidad puede llegar a ser de varios Mb. Real Audio permite trabajar con ficheros comprimidos en tiempo real descomprimiendo el archivo a la vez que lo reproduce, evidentemente la calidad del sonido no será igual que la obtenida de un CD, pero sirve perfectamente a sus fines.

- **MIDI:** el Musical Instrument Digital Interface (MIDI) es el protocolo de comunicaciones estándar utilizado para intercambiar datos entre sintetizadores, ordenadores, software, procesadores de efectos y otros dispositivos MIDI. Los archivos MIDI son significativamente diferentes de los archivos de audio digital como los VOC o los WAV. Los archivos de audio digital contienen sonido real grabado de forma digital a partir de la captura de miles de muestras por segundo. La música MIDI, por otro lado, contiene sólo instrucciones sobre la manera de tocar un instrumento. Los archivos de audio digital pueden necesitar millones de

bits para almacenar unos minutos de música, mientras que los MIDI con unos pocos miles de bits pueden reproducir música durante horas.
Un archivo MIDI es completamente editable, pudiéndose manipular el instrumento que se toca, la duración de una nota, la intensidad del sonido, etc.

2.1.2) Estructura de las tramas. El formato *WAV*:

El nombre WAVE proviene de *'Waveform Audio File Format'*, y es el estándar 'de facto' desarrollado por Microsoft e IBM como parte del RIFF (*'Resource Interchange File Format'*, formato de ficheros para intercambio de recursos) para Windows. Su ámbito de aplicación es en el entorno multimedia y define una serie de formatos y protocolos para el intercambio de información multimedia audio, de hecho forma parte de la especificación '*Microsoft's Multimedia Programming Interface and Data'*.

El formato WAVE es un subconjunto de las especificaciones RIFF de Microsoft, el cual puede incluir una gran diversidad de tipos de datos. Originalmente fue desarrollado para archivos multimedia, pero las especificaciones, están aún abiertas de forma que se permita que en tales ficheros se puedan incluir otros tipos de datos aún no definidos sin que por ello haya que cambiar los programas que actualmente leen este formato.

La descripción que vamos a hacer a continuación no va a ser muy exhaustiva, simplemente se va a definir el formato de este tipo de ficheros a '*grosso modo*'. Sin embargo incluímos una descripción mucho más detallada en el *Apéndice A*, en ella se ve la compleja estructura que presentan los ficheros *'.WAV '*.

- **El formato RIFF:**

RIFF es un formato capaz de almacenar muchos tipos de datos, principalmente datos multimedia como audio y vídeo. Está basado en conjuntos y subconjuntos

(*chunks and sub-chunks*). Cada conjunto tiene un tipo (*chunk type*), representado por una etiqueta de cuatro caracteres. Lo primero que nos encontramos en un fichero RIFF es el tipo de conjunto, seguido por el tamaño de dicho conjunto y a continuación su contenido (*chunk contents*).

El fichero RIFF no es más que un gran contenedor que contiene al resto de conjuntos. El primer campo en el contenido de cada conjunto es el tipo de formato (*form type*), que describe los tipos de datos que nos vamos a encontrar. Vemos un ejemplo de cómo es esta estructura:

Offset (hex)	**Contents**
0000	**'R', 'I', 'F', 'F'**
0004	**Length of the entire file - 8 (32-bit unsigned integer)**
0008	**form type (4 characters)**
000B	**first chunk type (4 character)**
0010	**first chunk length (32-bit unsigned integer)**
0014	**first chunk's data**
...	**...**

Todos los enteros se almacenan con formato “Little-Endian”, que consiste en ordenar por bytes siguiendo un orden de menor a mayor.

- **El formato WAVE:**

El formato WAVE es uno de los subconjuntos del formato RIFF usado para almacenar datos de audio exclusivamente. Su tipo de formato es “WAVE” y requiere dos tipos de subconjuntos:

- El subconjunto ***fmt (format)***, que describe la tasa de muestreo, el formato utilizado para los datos de audio, etc.

- El subconjunto ***data***, que contiene las muestras de audio.

WAVE puede contener también otros tipos de subconjuntos permitidos por el formato RIFF, entre ellos el subconjunto tipo *LIST*, que se utiliza para contener opcionalmente tipos de datos como la fecha de copyright, el nombre del autor, ... Los subconjuntos pueden aparecer en cualquier orden.

La especificación WAVE soporta muchos de los diferentes algoritmos de compresión de datos existentes. La etiqueta de formato (*format tag*), perteneciente al subconjunto de formato, es la encargada de indicar el tipo de compresión utilizada. Un valor de "1" significa que no se utiliza ningún tipo de compresión en la codificación de las muestras o que se utiliza compresión por codificación directa de las muestras, PCM (*Pulse Code Modulation*). Otros valores indican que existe algún tipo de compresión no directa.

Como se puede ver los ficheros WAVE son una herramienta muy potente, ya que nos podemos encontrar con multitud de variantes, sin embargo para la realización de este proyecto vamos a considerar una de ellas solamente, conocida como "Formato canónico WAVE". Ésta consiste básicamente en dos subconjuntos, el subconjunto de formato y el de datos por este orden, y en el uso del formato PCM para codificar las muestras de audio. Esta es la forma más simple de escribir y leer datos de ficheros WAVE, además de ser compatible con cualquiera de los programas creados para el procesado de ficheros WAVE.

2.1.3) Cabecera de un fichero *WAVE*:

Seguidamente vemos la estructura completa de la cabecera del fichero WAVE (formato canónico o estándar), junto con la cabecera del fichero RIFF que lo contiene:

	Posición	Tamaño	Nombre	Descripción
Cabecera RIFF	0	4	**ChunkID**	Contiene las letras ‘RIFF’ en formato ASCII (0x52494646 en formato big-endian).
	4	4	**ChunkSize**	36+SubChunk2Size, o con más precisión 4+(8+SubChunk1Size)+(8 + SubChunk2Size) Tamaño del resto del conjunto a partir de este byte. Es el tamaño del fichero entero en bytes menos los ocho bytes no incluídos en esta cuenta: ChunkID y ChunkSize
	8	4	**Format**	Contiene las letras ‘WAVE’. (0x57415645 en formato big-endian).
Subconjunto	12	4	**Subchunk1ID**	Contiene las letras ‘FMT’. (0x57415645 en formato big-endian).
	16	4	**Subchunk1Size**	16 para PCM. Este es el tamaño del resto del subconjunto a partir de este byte.
	20	2	**AudioFormat**	PCM = 1 (Cuantización Lineal). Otros valores indican otro tipo de cuantización.
	22	2	**NumChannels**	Mono = 1, Estéreo = 2, etc.

	24	4	**SampleRate**	8000, 11025, 44100, etc.
	28	4	**ByteRate**	SampleRate*NumChannels*BitsPerSample/8
	32	2	**BlockAlign**	NumChannels*BitsPerSample/8
	34	2	**BitsPerSample**	8 bits=8, 16 bits=16, etc.
		2	**ExtraParamSize**	No existe este campo para PCM.
		X	**ExtraParams**	Espacio para parámetros extra.
Subconjunto de datos	36	4	**Subchunk2ID**	Contiene las letras ‘DATA’. (0x64617461 en formato big-endian).
	40	4	**Subchunk2Size**	NumSamples*NumChannels*BitsPerSample/8
	44	*	**Data**	Datos audio.

- La ordenación de bytes asumida por defecto por los fichero WAVE es “Little-Endian”. Los archivos que utilicen la ordenación “Big-Endian” tendrán el identificador RIFX en vez de RIFF.
- Las muestras de datos deben terminar en un número de byte impar, sea cual sea.
- Las muestras de 8 bits son almacenadas como bytes sin signo, con rango de 0 a 255. Las muestras de 16 bits se almacenan como enteros con signo en complemento a 2, con rango de –32768 a 32767.

- Puede haber subconjuntos adicionales en un flujo de datos WAVE. Si es así cada uno tendrá un identificador de subconjunto de cuatro caracteres, asímismo el campo SubChunkSize tendrá formato largo sin signo y la longitud de los datos vendrá definida por SubChunkSize.

Aunque, como se puede observar, la estructura de los ficheros WAVE está muy bien definida y parece que no es probable que obtengamos errores a la hora de extraer los verdaderos datos de audio (PCM) tenemos que ser precavidos, pues a veces se van introduciendo algunas modificaciones o puede que sea el dispositivo de captura y grabación, en nuestro caso la Sound Blaster, el que dé lugar a dichos cambios. De hecho, en el análisis que se llevó a cabo durante la realización del proyecto se comprobó que los ficheros WAVE con los que trabajamos tienen una ligera variación en la cabecera respecto de la que acabamos de ver. Por este motivo hubo que programar un lector para poder pasar los datos de audio al entorno de trabajo de Matlab. Más adelante veremos este problema con más detenimiento.

2.1.4) Ventajas e inconvenientes del formato WAVE:

Los ficheros WAVE han sido muy utilizados para reproducir audio durante los últimos años, de forma que su uso ha sido y es muy extendido. Este predominio sólo se está viendo truncado en los dos últimos años por la aparición de nuevos formatos de audio comprimido.

Debido al crecimiento de Internet y al uso masivo de redes de datos se hacía necesario un formato que permitiera el traspaso de información multimedia, y por lo tanto datos de audio, con cierta velocidad. Para ello se han creado varías estructuras de almacenamiento de datos de forma comprimida, todas basadas en el método de *Codificación Perceptual* (codificación de sub-bandas), que permite reducir a una décima parte el espacio necesario para almacenar un sonido o el ancho de banda para transmitirlo.

Un ejemplo es la compresión de sonido contemplada en el estándar ISO MPEG 1 - Layer 3, más conocida como MP3. Incluso a un oído muy bien adiestrado le cuesta mucho diferenciar la señal original en formato PCM (un WAV por ejemplo) de la convertida en MP3, que ocupa de diez a doce veces menos espacio. Y precisamente ahí es donde esta el truco, estos sistemas funcionan "con pérdida", pero esta pérdida no puede ser percibida por el oído humano.

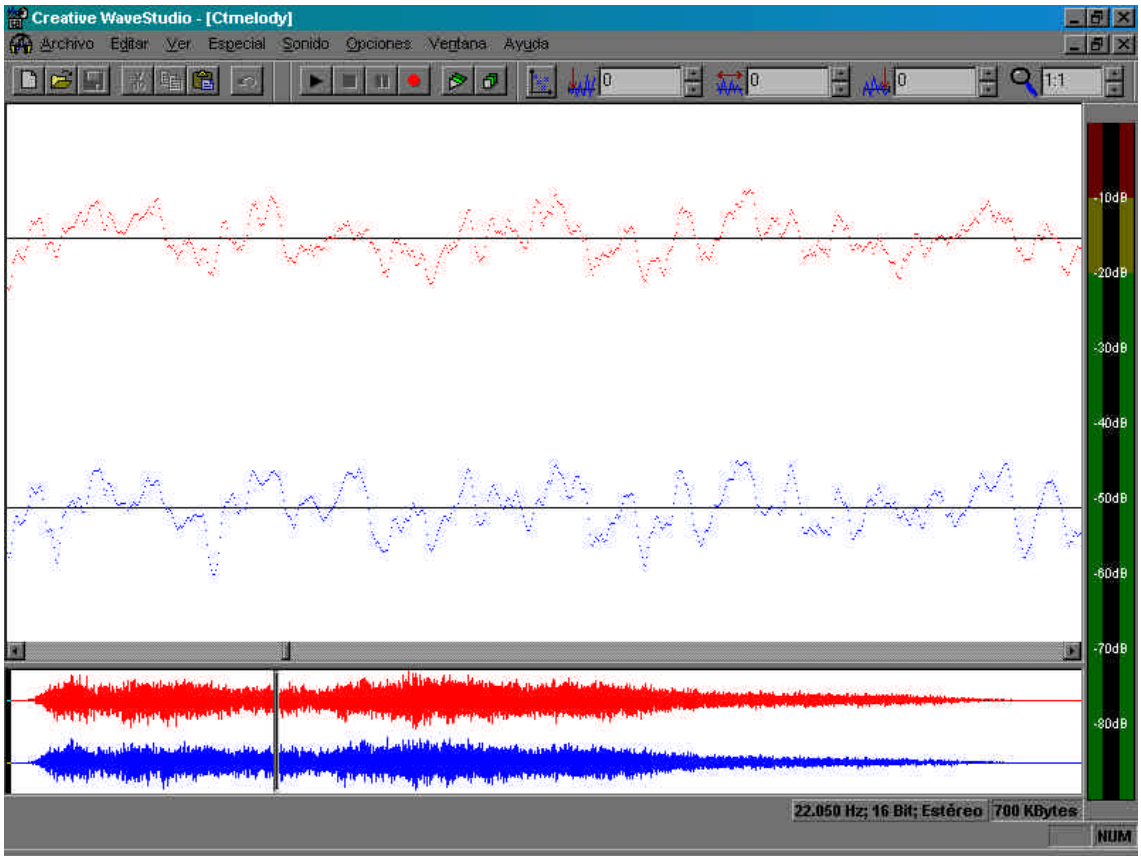
Sin embargo para nuestro estudio no nos vale ninguno de los formatos de audio que conlleven compresión no lineal, pues sería mucho más difícil detectar y reconocer cualquier tipo de sonido debido a la distorsión que introducen. Es por ello por lo que elegimos el formato PCM, y por lo tanto el formato WAVE, cuyo único inconveniente, que es el espacio que ocupa, no tiene mayores consecuencias.

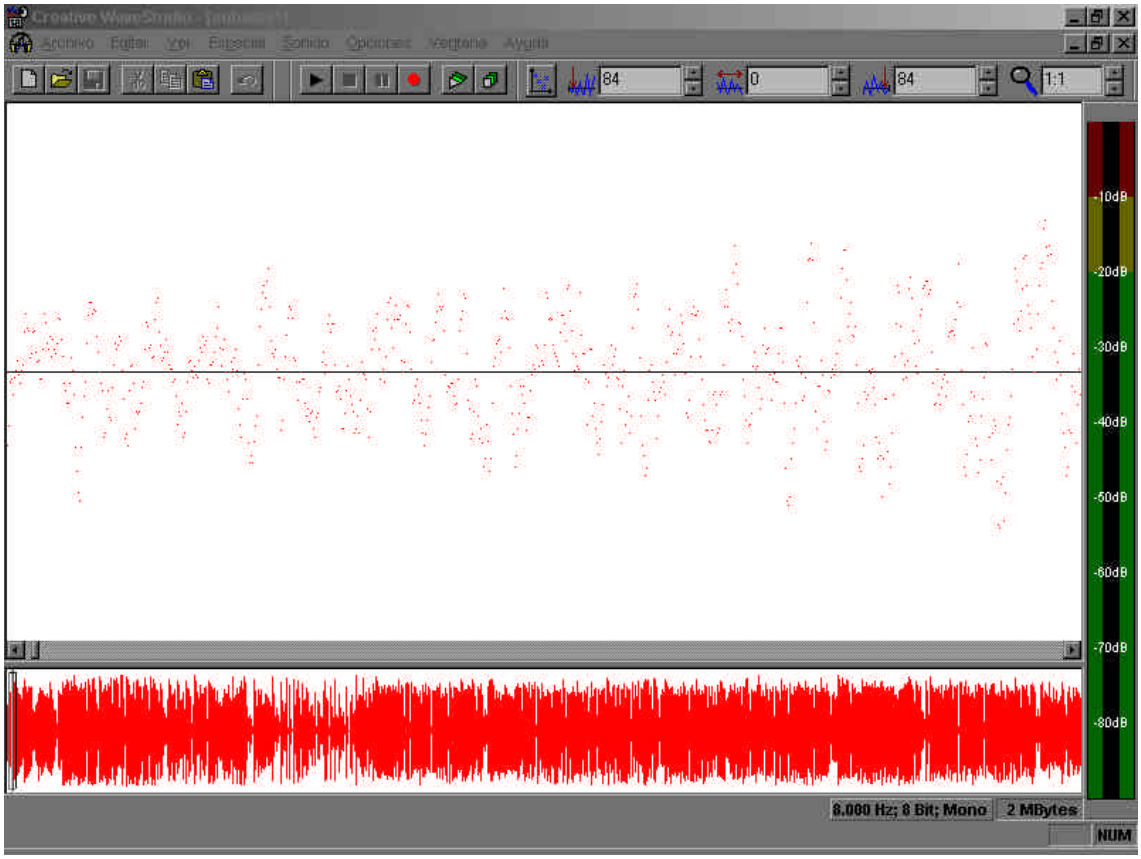
En cuanto a las ventajas:

- Compresión lineal, muy adecuada para nuestros propósitos (detección y reconocimiento de tramas de audio), ya que introduce una distorsión mínima en la señal original.
- Simplicidad a la hora de leer el contenido del fichero y manejar los datos audio.
- Versatilidad del formato RIFF, capaz de almacenar y transportar una gran variedad de datos: mapas de bits, audio, vídeo, información de control de los dispositivos periféricos, ...

Para finalizar con este análisis mostramos dos señales WAVE, la primera de ellas estéreo, con dos canales, y la segunda mono. Se puede observar que en la primera figura hay una mayor densidad de muestras, esto es debido a que la frecuencia de muestreo de esta señal es casi tres veces mayor (22050/8000= 2.756). Esto significa que por cada muestra de la segunda señal habrá aproximadamente tres muestras en cada uno de los canales de la primera señal, o sea, seis muestras, y por lo tanto mayor calidad de sonido. A cambio, la segunda

señal ocupa una sexta parte del espacio que ocupa la primera y su procesado es mucho más rápido.





2.2) Análisis de necesidades:

Tras analizar la estructura de los ficheros con los que vamos a trabajar haremos lo propio con el método a utilizar para el reconocimiento del audio. De esta manera estudiaremos las características y propiedades de las funciones convolución y correlación en esta sección.

2.2.1) Métodos de comparación y detección de tramas de audio:

En el capítulo anterior nos hemos centrado en los fundamentos del modelo de señal de audio, y consecuentemente en las propiedades de estas señales. Basándose en dichas propiedades, los algoritmos de procesamiento de señal se encargan de extraer algún tipo de información. Esto ocurre para distintos tipos de señales y diferentes aplicaciones.

En concreto, para aplicaciones como el RADAR, el SONAR, o incluso el procesamiento de *arrays* de señales se utiliza la misma técnica que vamos a emplear nosotros. Se trata de captar una señal buscada y tratar de reconocerla mediante la comparación de dicha señal con diferentes patrones. Aunque el tipo de señal y sus características y propiedades cambien, el objetivo principal es el mismo, el reconocimiento y detección de señales.

La intención de la *teoría de la detección* es la de proporcionar técnicas racionales para determinar cuál de los distintos modelos de generación y medida de datos es más consistente con un conjunto dado de datos. En comunicaciones digitales la señal recibida debe ser procesada para determinar si representa un “0” binario o un “1”; en RADAR o SONAR, la presencia o ausencia de un blanco debe ser determinada por medidas de los campos de propagación; en problemas sísmicos, la presencia de depósitos de petróleo debe ser deducida a través de medidas de la propagación del sonido a través de la tierra. Usando la *teoría de la detección,* podemos hallar algoritmos de procesamiento de señal que dan buena respuesta a problemas en los que la señal que transporta la información está adulterada por otras señales supérfluas, o sea ruído.

En todos los ejemplos vistos anteriormente, RADAR, SONAR, *arrays* de señales, estudios sísmicos y del terreno, ..., se utiliza una técnica muy conocida para la detección y reconocimiento de señales, lo cual indica que se trata de una técnica bastante buena, dado que es válida para trabajar con señales de diferentes características. Esta técnica no es otra que la de comparar dos señales mediante el estudio de la correlación cruzada de ambas. Así, de todas las aplicaciones mencionadas las que más se asimilan a la que ahora nos ocupa son las del RADAR y el SONAR, ya que en ellas se captura un fragmento de señal intermitentemente y se analiza si existe un blanco en función de la información que nos facilita la correlación de dicha señal. Además, en caso que haya blanco, el procesador puede decirnos de qué clase de blanco se trata mediante una análisis comparativo que se hace a posteriori.

El RADAR trabaja con un receptor de correlación, que implementa una operación muy intuitiva, que puede ser consolidada teóricamente de muchas

formas. Es importante, sin embargo, distinguir dos criterios separados: detección de un blanco y estimación de su rango y Doppler. En el problema de detección el receptor de correlación se utiliza para generar una estadística, que es la base para la comparación con un umbral y tomar la decisión acerca de la presencia de un blanco dentro de un cierto rango y una región Doppler. El funcionamiento del detector está normalmente presente en las características operativas de un receptor que toma la probabilidad de detección como la probabilidad de falsa alarma. En el problema de la estimación el receptor de correlación es utilizado para generar un estadístico que es función de los posibles valores de rango y Doppler del blanco. Entonces este estadístico es maximizado en función a estos valores para hacer la estimación. El funcionamiento del estimador se representa normalmente como la varianza de los parámetros estimados.

En ambos casos, detección de un blanco y estimación de sus características principales, el método a utilizar es el de la comparación a través de la correlación, y éste será también el que nosotros utilicemos en nuestro proyecto.

La principal razón para utilizarlo es que está presente prácticamente en todos las aplicaciones en las que haya que comparar señales, cualquiera que sea la naturaleza de dichas señales, analógicas o digitales, imágenes, voz, comunicaciones, ... Este hecho se debe principalmente a que la correlación es un proceso sencillo y rápido de implementar y muy fiable a la hora de comparar señales.

La otra razón para tomar esta decisión, es que hay muy diversas formas de implementar el proceso de correlación, algunas de las cuales nos permitirán trabajar en tiempo real, y por lo tanto conseguir el objetivo de este proyecto.

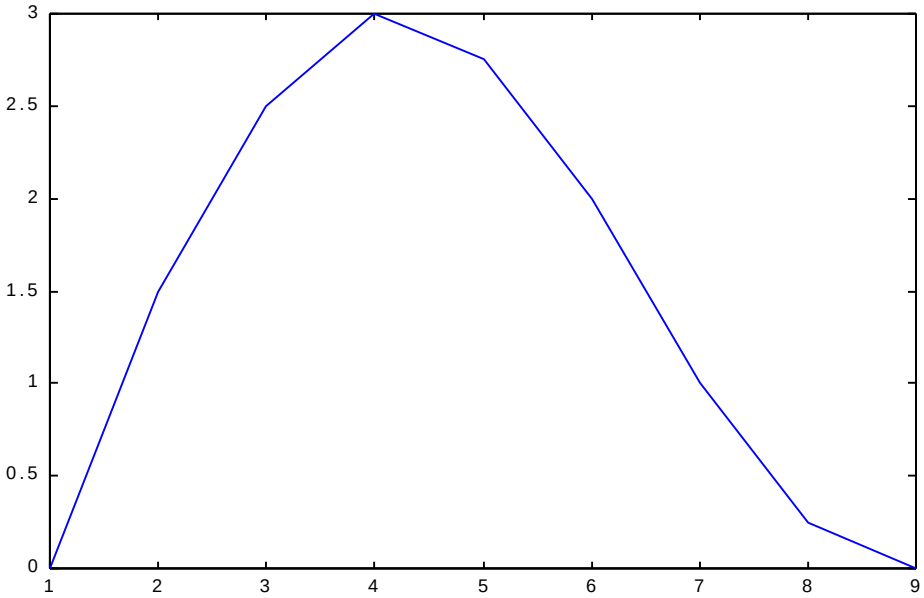
A continuación estudiamos la función correlación con más detalle.

2.2.2) Convolución y correlación de señales. Propiedades:

A continuación vamos a definir el proceso de correlación, así como algunas de sus propiedades, para ello es preciso describir la función convolución:

- **Descripción de la función convolución:**

El término convolución describe, entre otras cosas, cómo interactúa la entrada a un sistema con ese mismo sistema para producir la salida. Generalmente la salida del sistema será una versión retrasada y amplificada o atenuada de la entrada. Particularmente hay un caso muy útil, es cuando la entrada es la función impulso unidad, entonces a la salida se obtiene la respuesta temporal del sistema, o sea la transformada inversa de la función de transferencia del sistema. Ya sabemos que cualquier señal se puede representar como una secuencia de impulsos con distintas amplitudes, por lo que si el sistema es LTI y conocemos su respuesta temporal se puede obtener de forma inmediata la salida del sistema ante cualquier señal de entrada. A continuación se muestra un ejemplo de la salida de un sistema LTI que tiene como entrada la función impulso unidad.



Esta figura muestra que en el instante de muestreo '*m*' la salida debida al impulso unidad aplicado en el instante de muestreo '*0*' es h(m), conocida como *respuesta impulso* del sistema.

Si se utiliza como entrada al sistema una secuencia de impulsos, x(m), aplicados en los instantes de muestreo '*m*', entonces la salida en el instante '*0*' viene dada por:

$$y(0) = h(0)x(0)$$

En el instante de muestreo m=1 la salida vendrá dada por h(0)x(1), el efecto de la entrada actual x(1), más el efecto retardado h(1)x(0) de la entrada aplicada en el instante de muestreo m=0. Esto es:

$$y(1) = h(0)x(1) + h(1)x(0)$$

De la misma manera, las siguientes salidas vendrán dadas por:

$$y(2) = h(2)x(0) + h(1)x(1) + h(0)x(2)$$
$$y(3) = h(3)x(0) + h(2)x(1) + h(1)x(2) + h(0)x(3)$$
$$\vdots$$
$$y(n) = h(n)x(0) + h(n-1)x(1) + \ldots + h(0)x(n)$$

De esta forma se puede escribir la salida para cada instante de muestreo como una combinación lineal de la entrada en los instantes de muestreo anteriores si el sistema es lineal. Las ecuaciones anteriores describen la salida de un sistema lineal de primer orden.

Si se observan las expresiones anteriores se ve que la salida se obtiene de multiplicar la secuencia de entrada por los puntos correspondientes a la respuesta impulso invertida en el tiempo o viceversa, ya que también se puede escribir como:

$$y(n) = h(0)x(n) + h(1)x(n-1) + \ldots + h(n)x(0)$$

Esto no es otra cosa que la suma de convolución, que es equivalente a la correlación cruzada de una secuencia con la versión invertida en el tiempo de la segunda secuencia. Si escribimos estas ecuaciones de forma compacta nos queda:

$$\mathbf{y(n)} = \sum_{m=0,n} h(n-m)x(m) = \mathbf{x(n) * h(n)}$$

y

$$\mathbf{y(n)} = \sum_{m=0,n} h(m)x(n-m) = \mathbf{x(n) * h(n)}$$

Estas ecuaciones se conocen como las sumas de convolución de las entradas por la función respuesta impulso, y la salida viene dada por la convolución de la señal de entrada y la respuesta impulso del sistema.

Las ecuaciones anteriores se pueden extender para señales de duración infinita:

$$\mathbf{y(n)} = \sum_{m=0,\infty} h(n-m)x(m)$$

y

$$\mathbf{y(n)} = \sum_{m=0,\infty} h(m)x(n-m)$$

Si la entrada consiste en una secuencia continua entonces se cambian los sumatorios por integrales y tenemos la integral de convolución, que se extiende desde -∞ hasta ∞:

$$y(t) = \int_{\infty} x(\lambda)h(t-\lambda)\, d\lambda$$

Hasta ahora hemos utilizado el término convolución para describir la respuesta de un sistema ante una secuencia de entrada cualquiera, sin embargo la idea de convolución se puede extender a cualquier conjunto de datos de forma que el término debe ser considerado en un sentido más general.

- **Propiedades de la convolución:**

Algunas de las propiedades de la función convolución son:

- ✓ Propiedad conmutativa:

$$x1(t) * x2(t) = x2(t) * x1(t)$$

- ✓ Propiedad distributiva:

$$x1(t) * [x2(t) + x3(t)] = x1(t) * x2(t) + x1(t) * x3(t)]$$

- ✓ Propiedad asociativa:

$$x1(t) * [x2(t) * x3(t)] = [x1(t) * x2(t)] * x3(t)]$$

- **Multiplicación de dos DFTs y convolución circular:**

Supongamos que tenemos dos secuencias de duración finita de longitud N, $x_1(n)$ y $x_2(n)$. Sus respectivas DFTs de N puntos son:

$$X_1(k) = \sum_{n=0,N-1} x_1(n)\, e^{-j2\pi nk/N}; \qquad k = 0, 1, ..., N-1$$

$$X_2(k) = \sum_{n=0,N-1} x_2(n)\, e^{-j2\pi nk/N}; \qquad k = 0, 1, ..., N-1$$

Si multiplicamos las dos DFTs, el resultado es una DFT, digamos, $X_3(k)$, de una secuencia $x_3(n)$ de longitud N. Determinemos la relación entre $x_3(n)$ y las secuencias $x_1(n)$ y $x_2(n)$. Tenemos:

$$X_3(k) = X_1(k)X_2(k); \qquad k = 0, 1, \ldots, N-1$$

La IDFT de $\{X_3(k)\}$ es:

$$x_3(m) = (1/N) \sum_{k=0,N-1} X_3(k)\, e^{j2\pi km/N}$$

$$= (1/N) \sum_{k=0,N-1} X_1(k)\, X_2(k)\, e^{j2\pi km/N}$$

Supongamos que sustituimos $X_1(k)$ y $X_2(k)$ usando las DFTs calculadas anteriormente. Por lo tanto, obtenemos:

$$x_3(m) = (1/N) \sum_{k=0,N-1} [\sum_{n=0,N-1} x_1(n)\, e^{-j2\pi nk/N}][\sum_{l=0,N-1} x_2(l)\, e^{-j2\pi lk/N}]\, e^{j2\pi km/N}$$
$$= (1/N) \sum_{n=0,N-1} x_1(n) \sum_{l=0,N-1} x_2(l) [\sum_{k=0,N-1} e^{j2\pi k(m-n-l)/N}]$$

El sumatorio interior entre corchetes tiene la forma:

$$\sum_{k=0,N-1} (a^k) = \begin{cases} N; & a = 1 \\ (1-a^N)/(1-a); & a \neq 1 \end{cases}$$

donde a se define como:

$$a = e^{j2\pi k(m-n-l)/N}$$

Observamos que a = 1 cuando *m-n-l* es un múltiplo de N. Por otra parte, $a^N = 1$ para cualquier valor de $a \neq 0$. En consecuencia, la ecuación anterior se reduce a:

$$\sum_{k=0,N-1} (a^k) = \begin{cases} N; & l = m - n + pN = ((m-n))_N, \quad p \text{ entero} \\ 0; & \text{en el resto} \end{cases}$$

Sustituyendo obtenemos la expresión deseada de $x_3(m)$ de la forma:

$$\mathbf{x_3(m) = \sum_{n=0,N-1} x_1(n)\, x_2((m-n))_N = x_1(n) \otimes x_2(n) \qquad m = 0, 1, ..., N-1}$$

Esta expresión tiene la forma de una convolución. Sin embargo, no es la convolución lineal ordinaria que se ha visto, que relaciona la secuencia de salida y(n) de un sistema lineal con la secuencia de entrada x(n) y la respuesta impulsional h(n). A diferencia de ésta, la convolución de la expresión contiene el índice $((m-n))_N$ y se denomina convolución circular. Por lo tanto, concluímos que el producto de las DFTs de dos secuencias es equivalente a la convolución circular de dos secuencias en el dominio del tiempo.

En el cálculo de la convolución circular tienen lugar básicamente los mismos cuatro pasos que en la convolución lineal ordinaria: reflexión (en el tiempo) de una secuencia, desplazamiento de la secuencia reflejada, multiplicación de las dos secuencias para obtener la secuencia producto y, finalmente, suma de los valores de la secuencia producto. La diferencia básica entre ambos tipos de convolución es que en la convolución circular las operaciones de reflexión y desplazamiento se realizan de forma circular calculando el índice de una de las secuencias módulo N. En la convolución lineal no existe ninguna operación módulo N.

- **Convolución lineal rápida:**

Se ha visto que los resultados de la convolución lineal y la convolución circular de dos secuencias son distintos, así que buscaremos una metodología en el dominio de la frecuencia equivalente a la convolución lineal. Ya se ha visto que si tenemos una secuencia de entrada x(n), de longitud L, a un sistema h(n), de longitud M, donde h(n) es la respuesta implusional del sistema. La secuencia de salida en el dominio del tiempo y(n) puede expresarse como la convolución de x(n) y h(n):

$$y(n) = \sum_{k=0,M-1} h(k)x(n-k)$$

Dado que h(n) y x(n) son secuencias de duración finita, su convolución es también de duración finita. De hecho, la duración de y(n) es L+M-1.

El equivalente a la ecuación anterior en el dominio de la frecuencia es;:

$$Y(w) = X(w)H(w)$$

Si la secuencia y(n) ha de representarse de forma única en el dominio de la frecuencia mediante muestras de su espectro Y(w) en un conjunto de frecuencias discretas, el número de muestras distintas debe ser igual o superior a L+M-1. Por lo tanto se requiere una DFT de tamaño $N \geq L+M-1$, para representar {y(n)} en el dominio de la frecuencia. Ahora, si:

$$Y(k) = Y(w)|_{w=2\pi k/N} = X(w)H(w)|_{w=2\pi k/N}; \qquad k = 0, 1, ..., N-1$$

entonces:

$$Y(k) = X(k)H(k); \qquad k = 0, 1, ..., N-1$$

donde {X(k)} y {H(k)} son las DFTs de N puntos de las secuencias x(n) y h(n), respectivamente. Dado que las secuencias x(n) y h(n) son de duración menor que N, rellenamos estas secuencias con ceros para hacerlas de longitud N. Este aumento de la longitud de las secuencias no modifica sus espectros X(w) y H(w), que son espectros continuos, dado que las secuencias son aperiódicas. Sin embargo, al muestrear sus espectros en N frecuencias equiespaciadas (calculando las DFTs de N puntos), hemos aumentado el número de muestras que representan a estas secuencias en el dominio de la frecuencia por encima del mínimo (L ó M, respectivamente).

Dado que la DFT de N = L+M-1 puntos de la secuencia de salida y(n) es suficiente para representar a y(n) en el dominio de la frecuencia, tenemos que la multiplicación de las DFTs de N puntos X(k) y H(k), seguida del cálculo de la IDFT de N puntos, debe dar la secuencia {y(n)}. Por otra parte, esto implica que la convolución circular de N puntos de x(n) con h(n) es equivalente a la convolución lineal de x(n) con h(n). En otras palabras, aumentando la longitud de las secuencias x(n) y h(n) hasta N puntos (añadiendo ceros), y convolucionando circularmente las secuencias resultantes, obtenemos el mismo resultado que habríamos obtenido con la convolución lineal. Por lo tanto, con el relleno con ceros, podemos usar la DFT para realizar la convolución lineal de dos secuencias.

Esta característica nos será de gran ayuda a la hora de realizar el algoritmo de correlación para la comparación de señales, ya que el algoritmo de convolución no es computacionalmente efectivo, pues requiere un alto numero de operaciones. Sin embargo sí que se puede trabajar en tiempo real si se multiplican las DFTs de ambas señales y se antitransforma el resultado del producto. A esto se le llama *convolución lineal rápida*.

- **Ventajas computacionales de la convolución lineal rápida:**

Hemos visto que la convolución lineal se puede expresar de la siguiente forma:

$$x1(l) * x2(r) = F_D^{-1}[X1(k)X2(k)]$$

en donde F_D^{-1} denota la transformada discreta inversa de Fourier (IDFT), X1(k) es la transformada discreta de Fourier (DFT) de x1(l), y X2(k) es la transformada discreta de Fourier de x2(r). Consideraremos que la longitud tanto de x1(l) como de x2(r) es N.

También decíamos que el método de la convolución lineal rápida ofrece la ventaja de una mayor velocidad computacional si lo comparamos con el método directo, sin embargo esto sólo es cierto si el número de valores a convolucionar es suficientemente alto. Para obtener una medida de la eficiencia computacional relativa de ambos métodos se compara el número de multiplicaciones requeridas por cada uno de ellos para llevar a cabo la convolución.

Para el método directo tenemos que para obtener la convolución lineal de dos secuencias de N puntos, h(n-m) y x(m), es necesario multiplicar cada valor de h(n-m) por cada valor de x(m), entonces tendremos que multiplicar cada uno de los N valores de h(n-m) por cada uno de los N valores de x(m), por lo tanto tendremos un total de NxN = N^2 multiplicaciones.

Si estudiamos el método de la convolución rápida para el mismo par de secuencias de longitud N tenemos que considerar el relleno de ceros que hay que llevar a cabo para obtener la convolución lineal. Por lo tanto tendremos dos secuencias con longitud 2N-1, nosotros consideraremos que la longitud es de 2N para simplificar los cálculos. También supondremos $N \geq 8$, siendo N una potencia de 2 ($N = 2^d$, d entero), ya que entonces se puede utilizar el algoritmo eficaz de la FFT. El número de multiplicaciones complejas para calcular la FFT de una señal con N puntos viene dado por $(N/2)\log_2 N$, para señales con 2N puntos serán $(N)\log_2 2N$ multiplicaciones complejas. Se requiere el cálculo de dos DFTs y una IDFT, por lo que el número de multiplicaciones complejas necesarios es de $(3N)\log_2 2N$, además hay que tener en cuenta el producto X1(k)X2(k), ambas secuencias de 2N puntos, así tendremos $(3N)\log_2 2N+2N$ multiplicaciones complejas. Cada multiplicación compleja de la forma (A+jB)(C+jD) requiere cuatro multiplicaciones reales, por lo tanto el número de multiplicaciones para obtener la convolución lineal de dos señales mediante el método rápido viene dado por $(12N)\log_2 2N+8N$.

Para comparar la eficiencia computacional de ambos métodos mostramos una tabla en la que se compara el número de multiplicaciones reales para ambos métodos y para distintos valores de N:

N	Método Directo	Convolución Rápida	Relación Directo/Rápido
8	64	448	0.1429
16	256	1088	0.2353
32	1024	2560	0.4000
64	4096	5888	0.6957
128	16384	13312	1.2308
256	65536	29696	2.2070
512	262144	65536	4.0000
1024	1048576	143360	7.3153
2048	4194304	311296	13.4771

La tabla demuestra que la convolución rápida requiere menos operaciones que el método directo para secuencias con más de N=128 puntos, siendo aproximadamente diez veces más rápida para secuencias con N=1024 puntos. Esta misma conclusión se puede aplicar a la correlación directa y la correlación rápida.

Las secuencias con las que trabajaremos tendrán por lo general longitudes largas, como ejemplo supongamos que la frecuencia de muestreo escogida es de 8 Kbit/s y que vamos tomando de la radio trozos de señal de 1/8 de segundo, entonces tendremos secuencias de longitud N=1000. Por ello resultará muy beneficioso utilizar el método de correlación rápida, pues tenemos que trabajar en tiempo real. Aún así es posible modificar el algoritmo rápido de forma que se reduzca la carga computacional considerablemente, se trata de métodos en los que no se trabaja con la señal entera, sino que dicha señal se secciona y se van procesando las distintas secciones. Estos métodos se verán más adelante.

- **Correlación de señales discretas:**

Una operación matemática muy parecida a la convolución es la correlación. Al igual que en el caso de la convolución la correlación es una operación entre dos secuencias. Pero al contrario que la convolución, el objetivo de la correlación es medir el parecido que existe entre dos señales y así extraer información que dependerá de la aplicación concreta considerada. La correlación de señales es una operación que se realiza con frecuencia en distintas áreas de la ingeniería y la ciencia como radar, sonar, comunicaciones digitales, geología, etc.

En concreto, supongamos que tenemos dos secuencias x(n) e y(n) que queremos comparar. En radar y sonar, por ejemplo, x(n) puede representar muestras de la señal que transmitimos e y(n) muestras de la señal que recibimos a la salida del conversor analógico-digital (A/D). Si existe un blanco en el espacio explorado por el radar o sonar, la señal recibida y(n) es una versión retardada de la señal transmitida, reflejada por el blanco y corrompida por ruido aditivo.

La secuencia recibida se puede representar como:

$$y(n) = \alpha x(n-D) + w(n)$$

donde α es un factor de atenuación que contiene las pérdidas debidas al trayecto de ida y vuelta recorrido por la señal x(n), D es el retardo debido al trayecto de ida y vuelta y supone que es un múltiplo entero del intervalo de muestreo, y w(n) representa el ruido aditivo captado por la antena y cualquier otra componente de ruido generada por los componentes electrónicos y los amplificadores del receptor. Por otra parte, si no existe ningún blanco en el espacio explorado por el radar o el sonar, la señal recibida constará únicamente de ruído.

Teniendo las dos secuencias, x(n), que se denomina señal de referencia o señal transmitida, e y(n), la señal recibida, el problema radar o sonar consiste en determinar si existe algún blanco comparando x(n) con y(n) y, si es así, determinar el retardo en el tiempo, D, y a partir de él la distancia a la que se encuentra el blanco. En la práctica, la señal x(n-D) se encuentra fuertemente corrompida por ruido, de manera que la observación visual de y(n) no permite determinar la presencia o ausencia de un blanco. La correlación nos proporciona una forma de extraer esta información de y(n).

Otra de las áreas en la que la correlación se usa con frecuencia es la de las comunicaciones digitales. En comunicaciones digitales la información que se va a transmitir de un punto a otro se convierte a forma binaria, es decir, se transforma en una secuencia de unos y ceros que es transmitida hacia el receptor. Para transmitir un 0 podemos enviar una secuencia $x_0(n)$ para $0 \leq n \leq L-1$, y para transmitir un 1 la secuencia $x_1(n)$ para $0 \leq n \leq L-1$, donde L es un entero que indica el número de muestras en cada una de las dos secuencias. Muy a menudo se selecciona $x_1(n)$ como el valor negativo de $x_0(n)$. La señal recibida por el receptor se puede representar como:

$$y(n) = x_i(n) + w(n), \qquad i = 0,1 \;\; 0 \leq n \leq L-1$$

donde ahora lo que hay que determinar es si es $x_0(n)$ o $x_1(n)$ la señal contenida en y(n), y w(n) representa el ruido aditivo y otras interferencias propias de cualquier sistema de comunicación. Otra vez, parte del ruido tiene su origen en los distintos componentes del receptor. En cualquier caso, el receptor conoce las dos posibles secuencias transmitidas y su tarea consiste en compararlas con la señal recibida y(n) para determinar cuál de las dos se parece más a ésta. Esta comparación se realiza mediante la correlación que se describe a continuación.

- **Autocorrelación y correlación cruzada:**

Suponiendo que tenemos dos secuencias reales x(n) e y(n), ambas de energía finita, la *correlación cruzada* de las secuencias x(n) e y(n) es:

$$\mathbf{r_{xy}(l) = \sum_{n=-\infty,\infty} x(n)y(n-l)} \qquad \mathbf{l = 0, \pm 1, \pm 2, ...}$$

o también

$$\mathbf{r_{xy}(l) = \sum_{n=-\infty,\infty} x(n+l)y(n)} \qquad \mathbf{l = 0, \pm 1, \pm 2, ...}$$

El índice *l* es el parámetro de desplazamiento o retardo en el tiempo y los subíndices *xy* de la secuencia de autocorrelación indican las señales que han sido correladas. El orden de los subíndices, con *x* precediendo a *y* indica la dirección en que una secuencia es desplazada con respecto a la otra. Es decir, la secuencia x(n) no se desplaza y la secuencia y(n) se desplaza *l* muestras hacia la derecha si *l* es positivo y *l* muestras a la izquierda si *l* es negativo, esto es para la primera de las ecuaciones. De forma análoga, en la segunda ecuación la secuencia que se desplaza es x(n), y lo hace *l* muestras hacia la izquierda si *l* es positivo y *l* muestras a la derecha si *l* es negativo. De ahí que ambas ecuaciones produzcan idénticas secuencias de correlación cruzada.

Si invertimos los papeles de x(n) e y(n), y por lo tanto invertimos también los subíndices, obtenemos otra secuencia de correlación cruzada:

$$\mathbf{r_{yx}(l) = \sum_{n=-\infty,\infty} y(n)x(n-l)} \qquad \mathbf{l = 0, \pm 1, \pm 2, ...}$$

o también

$$\mathbf{r_{yx}(l) = \sum_{n=-\infty,\infty} y(n+l)x(n) \qquad l = 0, \pm 1, \pm 2, ...}$$

Por tanto, $r_{yx}(l)$ es simplemente la versión reflejada de $r_{xy}(l)$, donde la reflexión se hace con respecto a l=0. De aquí, que ambas secuencias de correlación nos proporcionen exactamente la misma información con respecto a la similitud entre x(n) e y(n).

Las similitudes entre el cálculo de la correlación cruzada y la convolución de dos secuencias son evidentres. En el cálculo de la convolución, una de las señales se refleja, se desplaza y entonces se multiplica por la otra secuencia; finalmente se suman todos los valores de la secuencia producto. Con excepción de la operación de reflexión, el cálculo de la correlación cruzada supone exactamente las mismas operaciones: desplazamiento de una de las secuencias, multiplicación de ambas y suma de todos lo términos de la secuencia producto. En definitiva, si tenemos un programa para el cálculo de la convolución, podemos usarlo para obtener la correlación cruzada proporcionando como entrada x(n) y la secuencia reflejada y(-n). Así, la convolución de x(n) con y(-n) nos da la correlación cruzada $r_{xy}(l)$, esto es:

$$\mathbf{r_{xy}(l) = x(l) * y(-l)}$$

En el caso especial de que y(n) = x(n), tenemos la autocorrelación de x(n), que se define como:

$$\mathbf{r_{xx}(l) = \sum_{n=-\infty,\infty} x(n)x(n-l)}$$

o también

$$\mathbf{r_{xx}(l) = \sum_{n=-\infty,\infty} x(n+l)x(n)}$$

Al tratar con señales de duración finita es costumbre expresar tanto la autocorrelación como la correlación cruzada mediante sumatorios finitos. En particular, si x(n) e y(n) son secuencias causales de longitud N (es decir, x(n) =

y(n) = 0 para n < 0 y n ≥ N), la autocorrelación y la correlación cruzada pueden expresarse como:

$$r_{xy}(l) = \sum_{n=i,N-|k|-1} x(n)y(n-l)$$

y

$$r_{xx}(l) = \sum_{n=i,N-|k|-1} x(n)x(n-l)$$

donde i=l, k=0 para l ≥ 0, e i=0, k=l para l < 0.

- **Propiedades de las secuencias de autocorrelación y correlación cruzada:**

Las secuencias de autocorrelación y correlación cruzada tienen varias propiedades importantes, que presentamos a continuación. Para desarrollar dichas propiedades consideremos que tenemos dos secuencias x(n) e y(n) de energía finita que combinamos linealmente para obtener

$$ax(n) + by(n-l)$$

donde a y b son constantes arbitrarias y l es un desplazamiento en el tiempo. La energía de esta señal es:

$$\sum_{n=-\infty,\infty} |ax(n) + by(n-l)|^2 = a^2 \sum_{n=-\infty,\infty} x^2(n) + b^2 \sum_{n=-\infty,\infty} y^2(n-l) + 2ab \sum_{n=-\infty,\infty} x(n)y(n-l)$$
$$= a^2 r_{xx}(0) + b^2 r_{yy}(0) + 2ab\, r_{xy}(l)$$

En primer lugar, observamos que $r_{xx}(0) = E_x$ y $r_{yy}(0) = E_y$, que son las energías de x(n) e y(n) respectivamente. Es obvio que:

$$a^2 r_{xx}(0) + b^2 r_{yy}(0) + 2ab\, r_{xy}(l) \geq 0$$

ahora, suponiendo que b ≠ 0 podemos dividir por b^2 y considerar la ecuación como una ecuación cuadrática de coeficientes $r_{xx}(0)$, $r_{yy}(0)$, y 2 $r_{xy}(l)$. Dado que siempre es no negativa el discriminante debe ser no positivo, es decir:

$$|r_{xy}^2(l) - r_{xx}(0)r_{yy}(0)| \leq 0$$

Por tanto la correlación cruzada verifica que:

$$\mathbf{|r_{xy}(l)|} \text{ £ } (r_{xx}(0)r_{yy}(0))^{1/2} = \mathbf{(E_x E_y)^{1/2}}$$

En el caso especial de que y(n) = x(n) entonces tendremos que:

$$\mathbf{|r_{xx}(l)|} \text{ £ } r_{xx}(0) = \mathbf{E_x}$$

Esto significa que la autocorrelación es una secuencia que alcanza su valor máximo para el retardo cero. Este resultado es consistente con el hecho de que una señal se adapta consigo misma para un retardo cero. En el caso de la correlación cruzada la ecuación constituye una cota superior de sus posibles valores.

Observamos que si una de las dos secuencias implicadas en el cálculo de la correlación cruzada se escala, la forma de la correlación no cambia, simplemente se produce un escalado de la misma de acuerdo con el escalado realizado sobre las secuencias originales. Así pues, el escalado carece de importancia y es a menudo conveniente, en la práctica, normalizar las secuencias de autocorrelación y correlación cruzada al rango entre –1 y 1. En el caso de la autocorrelación, simplemente dividiremos por $r_{xx}(0)$, así la autocorrelación normalizada se define como

$$\mathbf{r_{xx}(l) = r_{xx}(l)/r_{xx}(0)}$$

de forma similar definimos la correlación cruzada normalizada como:

$$\mathbf{r_{xy}(l) = r_{xy}(l)/(r_{xx}(0)r_{yy}(0))^{1/2}}$$

Ahora $|\rho_{xx}(l)| \leq 1$ y $|\rho_{xy}(l)| \leq 1$, y de aquí que estas secuencias sean independientes del escalado de la señal.

Finalmente, como ya hemos demostrado, la correlación cruzada satisface la siguiente propiedad:

$$\mathbf{r_{xy}(l) = r_{yx}(-l)}$$

Con y(n) = x(n), esta relación se traduce en la siguiente propiedad de la autocorrelación:

$$\mathbf{r_{xx}(l) = r_{xx}(-l)}$$

De aquí que la autocorrelación sea una función par. En consecuencia, es suficiente calcular $r_{xx}(l)$ para $l \geq 0$.

2.2.3) Carga computacional de la función correlación. Métodos de bloque:

Anteriormente vimos las diferencias, en cuanto a carga computacional se refiere, entre el método directo para el cálculo de la convolución de dos secuencias y el método de la convolución rápida. Asimismo se dijo que tales conclusiones se podían aplicar de la misma manera a la correlación, ya que convolucionar dos señales o hallar su correlación cruzada requiere el mismo número de cálculos.

Hasta ahora se ha asumido que las dos funciones a correlar son de duración finita. Sin embargo puede que no sea éste el caso. Podemos considerar los datos de entrada con duración infinita debido a que se trata de una señal continua en el tiempo, o con más probabilidad nos encontraremos con que la memoria disponible no es suficientemente grande para almacenar todos los datos. En nuestro caso particular nos encontramos con que para comparar señales de audio en tiempo real no nos es suficiente aprovechar la carácterística del método de la correlación rápida, pues este método aún es lento para nuestro propósito. Debido a ello hay que buscar otros métodos que sean más rápidos, o lo que es lo mismo más eficientes computacionalmente.

En estos casos es necesario implementar la correlación por etapas, dividiendo los datos de entrada en secciones separadas, realizando los cálculos para cada una de las secciones de entrada, y por último, combinando los resultados. Estos son los métodos de bloque, que se describen a continuación. Son dos, el Método de Solapamiento y Suma (Overlapp & Add) y el Método de Solapamiento y Almacenamiento (Overlapp & Save).

- **Método de Solapamiento y Suma (Overlapp & Add):**

En este método, el tamaño de las secciones o bloques de datos de entrada es de L puntos y el tamaño de las DFTs y las IDFTs es de N = L + M – 1. A cada bloque de datos se le añaden M-1 ceros y se calcula la DFT de N puntos. Por lo tanto, cada bloque de datos puede representarse como:

$$x_1(n) = \{x(0), x(1), ..., x(L-1), \underbrace{0, 0, ..., 0}_{\text{M-1 ceros}}\}$$

$$x_2(n) = \{x(L), x(L+1), ..., x(2L-1), \underbrace{0, 0, ..., 0}_{\text{M-1 ceros}}\}$$

$$x_3(n) = \{x(2L), x(2L+1), ..., x(3L-1), \underbrace{0, 0, ..., 0}_{\text{M-1 ceros}}\}$$

Y así sucesivamente. Las dos DFTs de N puntos se multiplican para formar:

$$Y_m(k) = H(k)X_m(k), \qquad k = 0, 1, \ldots, N-1$$

La IDFT da como resultado bloques de longitud N que no se encuentran afectados por aliasing, dado que, tanto el tamaño de las DFTs como el de las

IDFTs, es N = L + M – 1 y a las secuencias se les aumentó el tamaño hasta N puntos, añadiendo ceros a cada bloque.

Como cada bloque de datos termina con M-1 ceros, los M-1 últimos puntos de cada bloque de salida deben de solaparse y sumarse a los primeros M-1 puntos del siguiente bloque. De aquí, que este método se llame de solapamiento y suma. Este solapamiento y suma da como resultado la secuencia de salida:

$$y(n) = \{y_1(0), y_1(1), ..., y_1(L-1), y_1(L) + y_2(0), y_1(L+1) + y_2(1), ..., \quad y_1(N-1) + y_2(M-1), y_2(M), ...\}$$

- **Método de Solapamiento y Almacenamiento (Overlapp & Save):**

En este método el tamaño de los bloques de entrada es N = L + M – 1 y la longitud de cada una de las DFTs y las IDFTs, es N. Cada bloque de datos contiene al menos M-1 puntos del bloque de datos anterior, seguidos de L nuevos puntos, de manera que la longitud de la secuencia así formada es N = L + M – 1. Se calcula la DFT de N puntos de cada bloque de datos. La respuesta impulsional del filtro se aumenta en longitud añadiendo L-1 ceros, se calcula la DFT de N puntos y se almacena. La multiplicación de las dos DFTs de N puntos {H(k)} y $\{X_m(k)\}$ correspondiente al bloque m-ésimo de datos da lugar a:

$$\ddot{Y}_m(k) = H(k)X_m(k) \quad k = 0, 1, \ldots, N-1$$

Por lo tanto, la IDFT de N puntos da como resultado:

$$\ddot{y}_m(n) = \{\ddot{y}_m(0)\ddot{y}_m(1)...\ddot{y}_m(M-1)\ddot{y}_m(M)...\ddot{y}_m(N-1)\})$$

Dado que la longitud del registro de datos es N, los primeros M-1 puntos de $\ddot{y}_m(n)$ están distorsionados por el aliasing, y deben ser desechados. Los últimos L puntos de $\ddot{y}_m(n)$ son exactamente los mismos que resultarían de la convolución lineal y, como consecuencia:

$$\ddot{y}_m(n) = y_m(n), \qquad n = M, M+1, \ldots, N-1$$

Para evitar la pérdida de datos debida al aliasing, se almacenan los últimos M-1 puntos de cada registro de datos y estos puntos se convierten en los primeros M-1 puntos del registro siguiente, tal como se indicó arriba. Para empezar el procesado, los M-1 primeros puntos del primer registro se hacen iguales a cero. Por tanto, los bloques de datos son:

$$x_1(n) = \{\underbrace{0, 0, ..., 0}_{\text{M-1 ceros}}, x(0), x(1), ..., x(L-1)\}$$

$$x_2(n) = \{\underbrace{x(L-M+1), ..., x(L-1)}_{\text{M-1 puntos de datos de } x_1(n)}, \underbrace{x(L), ..., x(2L-1)}_{\text{L nuevos puntos de datos}}\}$$

$$x_3(n) = \{\underbrace{x(2L-M+1), ..., x(2L-1)}_{\text{M-1 puntos de datos de } x_2(n)}, \underbrace{x(2L), ..., x(3L-1)}_{\text{L nuevos puntos de datos}}\}$$

y así sucesivamente. Las secuencias de datos resultantes de la IDFT están dadas por $\ddot{y}_m(n)$, rechazándose los M-1 primeros puntos debido al aliasing, de manera que los L puntos restantes constituyen el resultado deseado de la convolución lineal.

2.2.4) Ventajas computacionales de los métodos de bloque:

Hemos visto que el esfuerzo computacional de la función correlación puede ser evitado siguiendo alguno de los dos métodos vistos, o sea, componiendo cada sección de forma de onda para correlarla. No es necesario analizarlo para darse cuenta de que los requerimientos computacionales de ambos métodos son similares, de forma que sólo vamos a considerar para este estudio el Método de Solapamiento y Suma, que es también el que utilizaremos para la realización de

nuestras rutinas de comparación y detección. Vamos a suponer que la secuencia x(n) de longitud N se divide en N/N_1 secciones, cada una de longitud N_1, que la secuencia h(n) tiene longitud N_2, y que las longitudes de las secuencias para la convolución lineal son $N^1 = 2^d \geq N_1 + N_2 - 1$. También se ha visto que para llevar a cabo la convolución o correlación rápida de dos secuencias de N puntos se requieren $12N^1\log_2 2N^1 + 8N^1$ multiplicaciones reales. Para implementar el algoritmo de solapamiento y suma tenemos (N/N_1) secciones, por lo que el número de multiplicaciones será:

$$\mathbf{(N/\ N_1)\ (12N^1 log_2 2N^1 + 8N^1) = R_m(S)}$$

Esto nos muestra que la longitud de las secuencias a correlar, N^1, debería ser pequeña, mientras que la longitud de las secciones de datos de x(n), N_1, debería ser aproximadamente igual a N^1. Idealmente $N^1 = 2^d \geq N_1 + N_2 - 1$. El número de multiplicaciones para la secuencia de N puntos original es:

$$\mathbf{12N\ log_2 2N + 8N = R_m(N)}$$

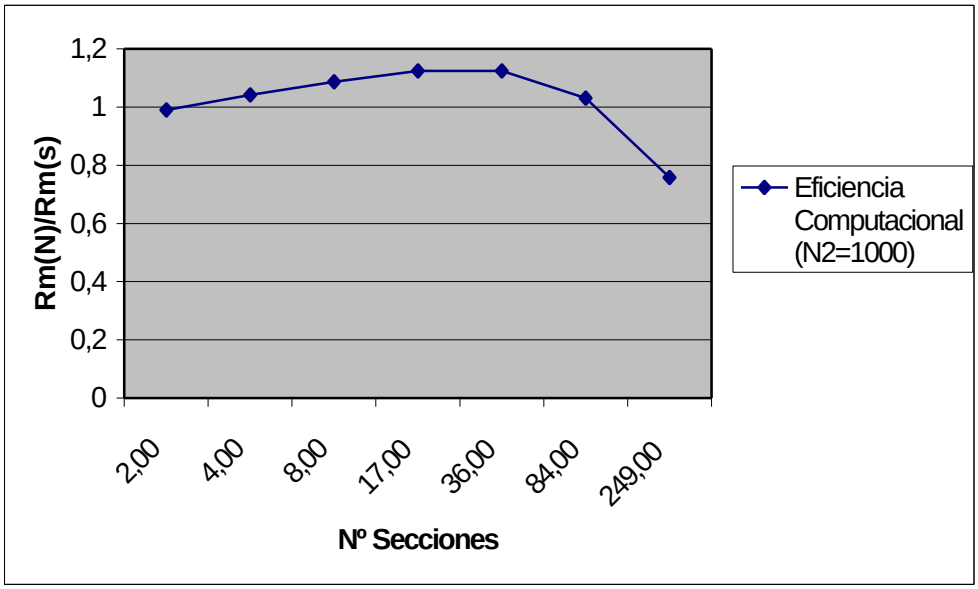
A continuación mostramos una tabla indicativa con estos valores teóricos:

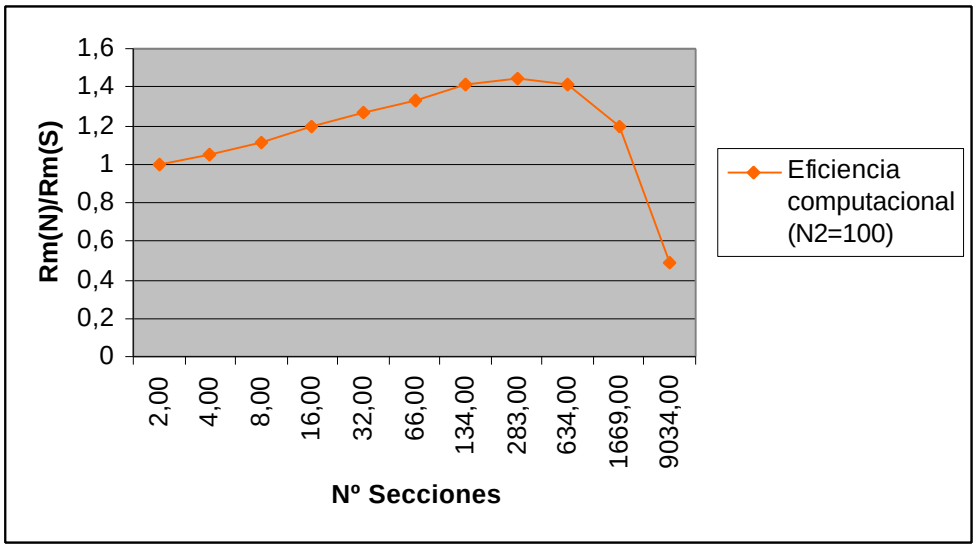
N	N^1	N_1	N/N_1	N_2	$R_m(S)/R_m(N)$
262000	2^{17}	130073	2	1000	1.01
262000	2^{16}	64537	4	1000	0.96
262000	2^{15}	31769	8	1000	0.92
262000	2^{14}	15385	17	1000	0.89
262000	2^{13}	7193	36	1000	0.89
262000	2^{12}	3097	84	1000	0.97
262000	2^{11}	1049	249	1000	1.32
262000	2^{10}	25	10480	1000	25.6
262000	2^{17}	130973	2	100	1.00
262000	2^{16}	65437	4	100	0.95
262000	2^{15}	32669	8	100	0.9
262000	2^{14}	16285	16	100	0.84

262000	2^{13}	8093	32	100	0.79
262000	2^{12}	3997	66	100	0.75
262000	2^{11}	1949	134	100	0.71
262000	2^{10}	925	283	100	0.69
262000	2^{9}	413	634	100	0.71
262000	2^{8}	157	1669	100	0.84
262000	2^{7}	29	9034	100	2.05

Se puede observar que se ha tomado un valor elevado para N, pues de lo contrario no tendría sentido este estudio, ya que se trata de analizar la eficiencia de los métodos de bloque a la hora de calcular la convolución de señales con un elevado número de muestras. N^1 toma siempre el valor de una potencia de dos, ya que representa la longitud de las DFTs que se van a calcular. La principal variable de este estudio es N/N1, número de bloques en que se divide la señal original para implementar el algoritmo de Solapamiento y Suma. Aunque en la tabla varía también el valor de N_2 no se le puede considerar como una verdadera variable para este análisis, ya que su valor vendrá determinado por otros criterios, aún así vemos como los resultados son más favorables cuanto menor es el valor de este parámetro.

De esta tabla se deduce que se puede conseguir una relación $R_m(S)/R_m(N) \leq 1$, siendo posibles ahorros computacionales del orden de hasta el 30%. Se observa que el valor óptimo se obtiene para un valor intermedio del número de bloques en que se divide la señal, de forma que cuando se aumenta demasiado N/N_1 ya no se obtiene tal ahorro. A continuación se muestran dos gráficas con los valores teóricos obtenidos:





Como se ha dicho se puede observar que el máximo para la eficiencia computacional se da para un valor intermedio del número de bloques. Hay que tener en cuenta que en estas gráficas se considera que el método de Solapamiento y Suma supone un ahorro computacional cuando el valor de Rm(N)/Rm(S) es mayor que uno.

2.3) Análisis de herramientas disponibles:

Como última parte del análisis que se está realizando de la situación vamos a estudiar las herramientas, tanto hardware como software, que se necesitan para la realización del proyecto. Vamos a hablar sólo y exclusivamente de los dispositivos y programas que realmente se han utilizado. Esto no quiere decir que no se pueda prescindir de alguno de estos elementos, pero probablemente el tiempo empleado en la realización del proyecto se vería incrementado de forma significativa.

2.3.1) Descripción del hardware necesario:

En realidad los equipos o harware necesario para llevar a cabo la realización del proyecto no son demasiados, además cualquier persona puede disponer de ellos hoy en día. A continuación veremos por encima qué elementos hemos utilizado. En el *Apéndice B* se verá con mayor detalle las características técnicas de cada uno de los elementos empleados.

- Ordenador personal (PC), Pentium II-450 MHz, 64 Mbytes SDRAM: con estas características técnicas se consigue tener los recursos necesarios para que nuestra aplicación funcione en tiempo real, aunque cabe la posibilidad de obtener buenos resultados con características algo más bajas.

- Tarjeta con dispositivo de captura de TV/FM: en realidad nos bastaría con una tarjeta sintonizadora de FM, sin necesidad de dispositivo capturador ni de sintonizador de televisión. Por supuesto se necesitará también una antena de FM, que normalmente viene acompañando a la tarjeta sintonizadora.

- Tarjeta de sonido Sound Blaster AudioPCI 128: puede grabar y reproducir archivos de sonido en multitud de formatos y con calidad de CD. Además, y gracias a la aplicación *'Recordsound.m'* de Matlab, permite grabar el sonido

directamente al entorno de trabajo de Matlab, sin lo cual habría sido mucho más difícil realizar el reconocimiento de anuncios en tiempo real.

- Altavoces: podemos utilizar cualquier tipo y modelo de altavoz, siempre que funcione y sea compatible con la salida de la tarjeta de sonido, caso habitual.

- Tarjeta gráfica: también nos da igual el modelo y la velocidad de la tarjeta gráfica, ya que no hay que ejecutar ningún programa o aplicación que requiera un procesado de imágenes importante. Aún así también se muestran sus características en el apéndice mencionado. En el caso de reconocimiento de video si tiene relevancia la resolución gráfica y la capacidad de procesado de imágenes.

2.3.2) Herramientas software:

Al igual que hemos hablado del hardware necesario también se hará un breve inciso en el software utilizado. Particularmente nos centraremos en dos programas, que han sido los más utilizados, uno de ellos es el Matlab, en concreto la versión 5.0; el otro es el Wave Studio, programa que viene junto con la Sound Blaster, por lo que su uso es recomendable si se trabaja con dicha tarjeta de sonido.

❑ Matlab 5.0. Descripción de aplicaciones audio:

Matlab es un programa interactivo que ayuda a la computación numérica y a la visualización de datos. Fundamentalmente Matlab está construído sobre un software basado en sofisticadas matrices que permite analizar sistemas de ecuaciones lineales. Estas herramientas nos permiten resolver problemas en Matemáticas Aplicadas, Física, Química, Ingeniería, Finanzas, etc, o sea, casi todas las áreas de aplicación que traten con cálculos numéricos complejos. De

hecho, trataremos las tramas de audio como vectores de datos, lo cual hace ideal el uso de Matlab.

Además posee multitud de librerías, cada una de las cuales contiene funciones y aplicaciones para un campo en concreto. Así, la librería que más utilizaremos será el *Toolbox* de Matlab, que entre otras contiene aplicaciones para el procesado de señales digitales.

Este programa es necesario para programar las rutinas y hacer las pruebas de comparación y detección de anuncios. De hecho la interfaz escogida para ejecutar nuestra aplicación será el entorno de trabajo del Matlab.

A continuación mostramos algunas de las funciones que más utilizaremos, vienen incluídas en la librería *Datafun* de Matlab. Son funciones para el procesado de sonidos y audio:

```
Correlation.
  corrcoef    - Correlation coefficients.
  cov         - Covariance matrix.
  subspace    - Angle between subspaces.

Filtering and convolution.
  filter      - One-dimensional digital filter.
  filter2     - Two-dimensional digital filter.
  conv        - Convolution and polynomial multiplication.
  conv2       - Two-dimensional convolution.
  convn       - N-dimensional convolution.
  deconv      - Deconvolution and polynomial division.

Fourier transforms.
  fft         - Discrete Fourier transform.
  fft2        - Two-dimensional discrete Fourier transform.
  fftn        - N-dimensional discrete Fourier Transform.
  ifft        - Inverse discrete Fourier transform.
  ifft2       - Two-dimensional inverse discrete Fourier transform.
  ifftn       - N-dimensional inverse discrete Fourier Transform.
  fftshift    - Move zeroth lag to center of spectrum.

Sound and audio.
  sound       - Play vector as sound.
  soundsc     - Autoscale and play vector as sound.
  speak       - Convert input string to speech (Macintosh only).
  recordsound - Record sound (Macintosh only).
  soundcap    - Sound capabilities (Macintosh only).
  mu2lin      - Convert mu-law encoding to linear signal.
  lin2mu      - Convert linear signal to mu-law encoding.

Audio file inport/export.
  auwrite     - Write NeXT/SUN (".au") sound file.
  auread      - Read NeXT/SUN (".au") sound file.
  wavwrite    - Write Microsoft WAVE (".wav") sound file.
  wavread     - Read Microsoft WAVE (".wav") sound file.
  readsnd     - Read SND resources and files (Macintosh only).
  writesnd    - Write SND resources and files (Macintosh only).
```

- **Wave Studio (Sound Blaster):**

WaveStudio es una aplicación para Windows que permite llevar a cabo fácilmente las siguientes funciones de edición de sonido:

- Reproducir, editar y grabar datos de onda de 8 bits (calidad de cinta) y 16 bits (calidad de CD).

- Mejorar los datos de onda o crear sonidos únicos mediante el empleo de diversos efectos especiales (como el rap, la inversión, el eco, el silencio o la panorámica) y operaciones de edición (como cortar, copiar o pegar).

- Abrir y editar varios archivos de onda de forma simultánea.

- Importar y exportar archivos de datos sin formato (.RAW).

Hay otras aplicaciones o programas capaces de realizar las mismas o más funciones que el Wave Studio, sin embargo con las que tiene este programa nos basta. Además el hecho de venir junto con la tarjeta Sound Blaster ha hecho que nos decantemos por él.

3) Comparación de tramas de audio:

Tras lo visto hasta ahora se puede deducir que el proceso en torno al cual va a girar el proyecto es el de la comparación de tramas de audio principalmente. Además dicho proceso debe ejecutarse en tiempo real, por lo que su estudio nos llevará algún tiempo. A continuación haremos un análisis minucioso de dicho proceso.

3.1) Introducción:

Hasta ahora hemos visto prácticamente propiedades generales del audio y de los archivos de sonido, así como un análisis introductorio de las herramientas y métodos que van a ser utilizados. A partir de ahora particularizaremos y ahondaremos mucho más, exponiendo con mayor grado de detalle cada uno de los pasos a dar en la realización del comparador y detector de anuncios comerciales.

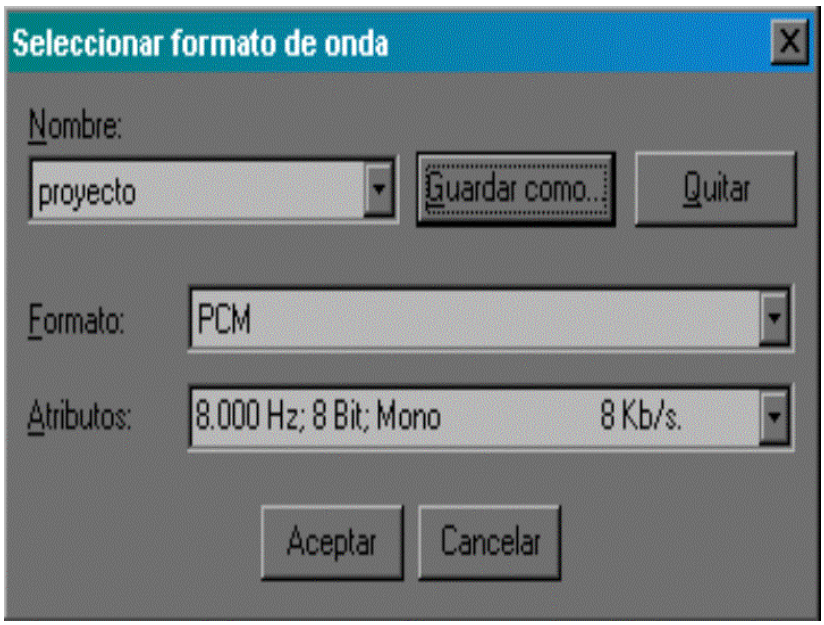
El primer paso es conseguir una rutina o aplicación que sea capaz de comparar tramas de audio. Además hay que programarla considerando que queremos trabajar en tiempo real, por lo que no se puede escapar ningún detalle y habrá que implementarla intentando optimizar el tiempo de ejecución.

Como hemos dicho antes haremos estas pruebas en el entorno de trabajo de Matlab. En un principio no vamos a trabajar con los datos directamente tomados de la Soundblaster, sino que para mayor comodidad se harán grabaciones de anuncios emitidos en la misma o distintas emisoras. La duración de estas grabaciones puede ser aleatoria, sin embargo hemos considerado un tiempo medio de 30 segundos, tiempo promedio que suele durar un anuncio. Además, intentaremos tener varias grabaciones del mismo anuncio, dos tomadas de la misma emisora y una tercera grabada de otra emisora distinta. De esta manera se tendrán en cuenta todos los factores negativos que nos pueden afectar a la hora de la comparación, pues así tendremos grabaciones de un mismo anuncio que puede haber sido emitido con distinta fase, potencia (amplitud) e incluso frecuencia, ya

que a veces dos emisiones diferentes de un anuncio pueden tener duraciones desiguales.

Un aspecto muy importante a la hora de realizar las grabaciones es que deben ser completamente aleatorias en todos los aspectos relacionados con la emisión: emisora, hora y día de emisión, etc. Así que debemos ser objetivos y grabar los anuncios sin atender a ninguna referencia, de lo contrario estaríamos falseando los datos que a posteriori nos servirán para hallar los umbrales de detección.

Para llevar a cabo esta tarea se ha optado, como se dijo anteriormente, por el Wave Studio, ya que además de poseer una interfaz gráfica amigable nos ofrece recursos muy interesantes a la hora de preparar la forma de onda y de definir su formato para posteriormente trabajar con ella. Así, cada grabación tendrá la duración exacta de cada emisión y tendremos varias grabaciones para cada anuncio. Además el formato por el que nos hemos decantado para hacer estas pruebas previas es el siguiente:



Este hecho no quiere decir que sea el formato definitivo de la aplicación, pues aunque probablemente el formato final sea codificar las muestras con PCM, 8 bits y un solo canal, aún no se sabe si la frecuencia de muestreo óptima será la de 8 KHz. Si utilizáramos dos canales, estéreo, o 16 bits para codificar las muestras necesitaríamos mucha más memoria para almacenar dichas muestras y se tardaría mucho más tiempo en procesarlas, de ahí que estos dos atributos sean conocidos a priori.

No hay que olvidar que para realizar el estudio estadístico habrá que hacer las pruebas pertinentes para todos las frecuencias de muestreo disponibles, por ello se ha optado por realizar las grabaciones de los anuncios a 44100 Hz. De esta manera se puede convertir el formato de la forma de onda a 22050, 11025, 8000 y 4000 Hz paulatinamente y sin producir distorsión en la forma de onda. Si lo hiciéramos a la inversa, por ejemplo pasar de un archivo grabado a 8000 Hz a un archivo con una frecuencia de muestreo de 22050 Hz, se acabaría produciendo distorsión en las muestras. Esto es debido a que los métodos utilizados por el Wave Studio para la conversión de frecuencias son el diezmado para pasar a una frecuencia más baja y la interpolación para la conversión a una frecuencia superior. Ambos métodos son efectivos pero el segundo de ellos tiene el inconveniente de distorsionar la señal de audio.

En definitiva, se sugiere hacer las grabaciones a 44100 Hz y guardarlas como tal, según se vayan haciendo pruebas necesitaremos convertir a otras frecuencias, entonces se hará una copia de la grabación original y convertiremos la frecuencia de la copia. Por ahora ya hemos dicho que vamos a trabajar a 8000 Hz para hacer las pruebas iniciales, posteriormente se harán pruebas a todas las frecuencias.

Por último comentaremos un pequeño matiz que puede darnos algún que otro problema si no lo consideramos. Aunque la interfaz que veamos al realizar las grabaciones sea la del Wave Studio, realmente el dispositivo que se encarga de almacenar los datos es la Sound Blaster. Eso quiere decir que habrá que configurarla adecuadamente para que grabe las muestras procedentes de la tarjeta de FM, tarea que se lleva a cabo con el Mezclador de la Sound Blaster, en concreto en el menú *"Control de Grabación"* hay que seleccionar como entrada

la pestaña correspondiente a *"Línea"*, pues esta es la entrada de la Sound Blaster donde se conecta la tarjeta de TV/FM:

Si no lo hacemos así oiremos la emisora que tengamos sintonizada en ese momento, pero nos será imposible grabar lo que escuchamos, ya que la Soundblaster estará grabando datos de otro dispositivo que no es la tarjeta de FM.

3.2) Características de la aplicación 'wavread.m' de Matlab:

Una vez almacenados los datos de audio en ficheros sólo nos queda pasar estas muestras al entorno de trabajo de Matlab para comenzar a procesarlas e ir avanzando. Dichas muestras se tratarán como vectores, de forma que cada grabación o anuncio pasará a formar un vector con tantos elementos como muestras o bytes (ya que cada muestra está codificada por 8 bits) tenga el anuncio. Por supuesto hay que tener en cuenta los bytes de cabecera, que son fundamentales para aportarnos información pero que no van a formar parte del vector con el que trabajaremos en Matlab. En este vector habrá sólo y exclusivamente muestras de audio.

Para realizar este proceso se puede utilizar una función de las librerías de Matlab, esta función se denomina *'wavread.m'*, y es bastante compleja, su código se muestra en el Apartado IV de este documento. Básicamente, lo que hace es leer todos los campos de la cabecera del fichero WAV, los cuales vimos anteriormente. Una vez leídos dichos campos, que contienen información acerca del formato y del tipo de datos audio que hay en el fichero, ya pueden leerse también las muestras de audio y almacenarlas en un vector en el entorno de Matlab.

No obstante no utilizaremos dicha función por un motivo ya expuesto anteriormente. Si recordamos la estructura de la cabecera de un fichero canónico WAV, en la posición 34 teníamos los siguientes campos:

34	2	**BitsPerSample**	8 bits=8, 16 bits=16, etc.
	2	**ExtraParamSize**	No existe este campo para PCM.
	X	**ExtraParams**	Espacio para parámetros extra.

Sin embargo nos encontramos con que este formato varía ligeramente si los ficheros WAV se crean bajo Windows95/98 con la Sound Blaster PCI 128, como es nuestro caso, de forma que la cabecera con la que realmente nos vamos a encontrar es:

34	2	**BitsPerSample**	8 bits=8, 16 bits=16, etc.
	2	**ExtraParamSize**	Para PCM tendremos '00'
	X	**ExtraParams**	Espacio para parámetros extra (no hay ninguno)

Por ello, cuando la función *'wavread.m'* termina de leer el campo ***'BitsPerSample'***, a continuación espera leer la posición 36, con lo que hay un

desfase en el proceso de lectura de la cabecera del fichero WAV, y nos devuelve un error, siendo incapaz de leer los datos de audio.

Para resolver este problema la solución adoptada es la de crear una rutina con un código muy simple que lea de forma correcta la cabecera de los ficheros con los que trabajamos. El nombre de dicha rutina va a ser *'waveload.m'* y también podemos ver su código en el Apartado IV del documento. Para evitar que haya desfases en la lectura de los datos de la cabecera del fichero WAV se implementa un bucle, con el operador *'while'*, en el que entra justo después de leer el campo ***'BitsPerSample'***. Dicho bucle está preparado para que se ignoren los campos '***ExtraParamSize'*** y ***'ExtraParam',*** de forma que no sale del bucle hasta que no se encuentre con los caracteres ***'DATA'*** codificados en ASCII. La razón es que el siguiente campo válido que tiene que leer es el que tiene la posición 36, y éste es:

36	4	**Subchunk2ID**	Contiene las letras 'DATA'. (0x64617461 en formato big-endian).

El hecho de ignorar los campos '***ExtraParamSize'*** y ***'ExtraParam'*** no tiene ninguna importancia, puesto que, como ya hemos dicho, en el primero de ellos nos vamos a encontrar el valor '0', y este valor indica el tamaño que ocupan los datos contenidos en el segundo de ellos.

3.3) Características y particularidades del método de comparación:

Nuestro objetivo es comparar dos señales de audio de forma eficaz, o sea, la comparación debe ser efectiva y dar resultados positivos si ambas señales son iguales o muy parecidas, y negativos cuando no lo sean; además el algoritmo tiene que estar preparado para que el tiempo de comparación sea lo suficientemente corto como para trabajar en tiempo real.

Más adelante, cuando se vea la rutina principal de detección y reconocimiento de anuncios que va a hacer llamadas a las subrutinas se comprobará que la subrutina que más problemas nos puede dar para trabajar en tiempo real es la de comparación, de hecho se verá que actúa como cuello de botella. Esto se debe a que el proceso de correlación de dos señales es complejo y necesita algún tiempo de ejecución, por lo que habrá que afinar bastante esta subrutina.

Como ya hemos dicho el método de comparación a utilizar es la correlación de ambas señales y para conseguir que este método sea computacionalmente eficiente se va a implementar el algoritmo de Solapamiento y Suma, más conocido como Overlap & Add.

- **Implementación del algoritmo Overlap & Add:**

Anteriormente estudiamos este algoritmo y se vio su metodología, así como sus ventajas computacionales. Para implementar este algoritmo en Matlab lo primero a tener en cuenta son las dos señales a correlar.

La señal de mayor longitud, x(n), va a ser el anuncio que elijamos en cada momento para hacer las pruebas, y su longitud *'Lx'* depende de la duración del anuncio. Como ya hemos dicho haremos estas pruebas con grabaciones de anuncios con formato PCM, un solo canal, muestras codificadas con 8 bits y frecuencia de muestreo igual a 8000 Hz. Por lo tanto si los anuncios tienen una duración media de 30 segundos la longitud media de la señal x(n) será:

$$Lx = (8000 \text{ muestras/seg}) \times (30 \text{ seg}) = 240000 \text{ muestras}$$

Este número de muestras es bastante alto en comparación con el de la otra señal a correlar. h(n) será la señal corta, y no es más que un trozo tomado aleatoriamente de cualquiera de los anuncios que tenemos grabados, si el anuncio del que se toma h(n) es x(n) entonces la correlación nos debe dar un resultado

positivo, pues estaríamos comparando un trozo de un anuncio con ese mismo anuncio.

Por lo tanto la señal corta, h(n), será la que iremos desplazando muestra a muestra con respecto a x(n), para realizar la comparación de las dos señales. Su longitud será variable, la que nosotros elijamos en cada momento para cada prueba, pues la función que implementemos (*'overadd1.m'*) para realizar las pruebas tendrá dicha variable como parámetro de entrada.

A continuación vamos a estudiar la función *'overadd1.m'*, pero antes hay que decir que esta no será la función que utilicemos en la rutina principal de detección y reconocimiento, sino que es una versión muy parecida que utilizaremos para hacer las pruebas. Estas pruebas servirán tanto para conseguir hacer la comparación de forma correcta, como para ir conociendo el tiempo empleado en la detección y las longitudes óptimas de las señales. La función que se utilizará en la rutina principal es *'overadd.m'*. Se puede ver el código de ambas en el Apartado IV del documento.

Los parámetros de entrada de la función *'overadd1.m'* son:

1. ***Lh***, longitud de la señal h(n). Es una de las principales variables con las que vamos a trabajar, ya que de ella dependerá más adelante el tiempo de captura de muestras de la emisora. Por ejemplo, si obtenemos un valor óptimo para Lh = 250 muestras significa que cuando trabajemos en tiempo real habrá que capturar datos de la radio durante intervalos de:

   T = Lh / Fs = (250 muestras) / (8000 muestras/seg) = 1 / 32 seg

2. ***x***, señal x(n). Como hemos dicho se trata de cualquiera de las grabaciones realizadas, que se corresponde con una emisión de un anuncio.

3. ***y***, señal de la cual se obtendrá h(n). y(n) puede ser el mismo anuncio que x(n) u otro distinto. h(n) será un trozo de longitud Lh tomado aleatoriamente de y(n).

4. ***M***, longitud de las secciones en que se divide x(n) para realizar el algoritmo de solapamiento y suma por bloques. Dividiremos x en L secciones de longitud M. Normalmente Lx no será múltiplo de M, por lo que habrá que rellenar con ceros la última de las L secciones, de esta forma todas las secciones tendrán longitud M.

En cuanto a los parámetros de salida tenemos:

1. ***s***, señal resultado del valor absoluto de la correlación normalizada entre x(n) y h(n). Contiene el valor absoluto de los coeficientes de correlación normalizados. Después se hablará de dicha normalización con más detenimiento. El hecho de trabajar con valores absolutos no influye en nuestros objetivos y hace que se simplifiquen bastante los cálculos.

2. ***noper***, número de operaciones empleado en realizar la correlación cruzada de las señales x(n) y h(n). Nos da una idea de la eficiencia computacional del algoritmo.

3. ***ymax***, máximo de la señal de correlación s(n), varía entre 0 y 1. Este es otro de los parámetros principales, pues se determinará que la comparación ha sido positiva o negativa en función de si su valor está por encima o por debajo del umbral de detección, respectivamente.

4. ***maxindex***, posición en la que se encuentra ymax dentro de la señal de correlación s(n).

Aunque no aparezca como parámetro de salida, también se utilizan las funciones de Matlab *'tic'* y *'toc'*, cuyo uso nos facilita el tiempo de ejecución a la salida de la función *'overadd1'*. Este parámetro será el que nos diga si se puede

trabajar en tiempo real o no, por lo que el objetivo principal es reducir este tiempo al mínimo y conseguir que ymax tenga el valor adecuado en cada momento, cercano a la unidad para detecciones positivas y cercano a cero para detecciones negativas.

Para finalizar, vemos los pasos a seguir en la implementación del algoritmo de una forma breve, después se comentarán con más detalle los aspectos más relevantes:

1. Seleccionamos la longitud de las secciones para que el cálculo de la DFT sea más eficiente, esto es:

$$N = M + Lh - 1$$

$$N = 2^{\,ceil(\log2(N))} \equiv 2^{d} \geq M + Lh - 1$$

   NOTA: la función ‘ceil’ redondea hacia más infinito.

2. Se calcula también el número de secciones de longitud M en que se divide x(n):

$$L = ceil(Lx/M)$$

3. Se invierte la señal h(n) y calculamos su DFT.

4. Multiplicamos la DFT de h(n) con las DFTs de cada una de las secciones en que se ha dividido x(n).

5. A cada una de las secciones resultantes se le calcula la IDFT.

6. Solapamos adecuadamente dichas secciones y se suman.

7. Por último, tomamos sólo las muestras que nos interesan, pues las colas de la correlación no nos sirven para nuestro estudio, y normalizamos los coeficientes.

Estos son los pasos a seguir para llevar a cabo la correlación de dos señales utilizando un método de convolución por bloques, en nuestro caso el método Overlap & Add.

- **Longitud de los bloques:**

Cuando se analizó anteriormente el método Overlap & Add se pudo comprobar que la eficacia del método depende de varios parámetros, como son las longitudes de las señales involucradas en la correlación, Lx y Lh, pero hay un parámetro que es decisivo en cuanto a la eficiencia del algoritmo. Dicho parámetro es la longitud de los bloques en que se divide la señal x(n) o anuncio grabado, o sea, el parámetro M. Si su valor es muy bajo entonces trabajaremos con un elevado número de DFTs muy cortas y rápidas de calcular; y por el contrario, si su valor es muy alto tendremos que trabajar con algunas DFTs largas y cuyo cálculo es muy lento. De ahí que haya que llegar a una solución de compromiso que nos dé el valor más adecuado para nuestros objetivos.

En realidad la longitud de las DFTs necesarias para el cálculo de la función correlación viene dada por el parámetro N, cuya expresión hemos visto antes:

$$N = 2^{ceil(log2(N))} \circ 2^{d} \geq M + Lh - 1$$

Con esta fórmula lo que se hace es redondear el valor de N hasta el valor inmediatamente superior que sea potencia de dos, de esta manera las longitudes de las DFTs son tales que se puede utilizar el algoritmo eficiente para el cálculo de la DFT, lo cual ahorra muchas operaciones y por lo tanto tiempo. Sin embargo este redondeo de N no afecta demasiado al valor original de N, que es:

$$N = M + Lh - 1$$

Si tenemos en cuenta que se va a trabajar con anuncios que tienen una duración media de 30 seg., la longitud media de dichos anuncios será aproximadamente Lx = 240000 muestras. Antes vimos que para que el algoritmo sea realmente efectivo N no debe tener un valor muy alto y además N y M tendrán valores muy aproximados, sin embargo en nuestro caso la primera condición es difícil de cumplir, ya que si tomamos un valor demasiado bajo para M tendremos que trabajar con un alto número de secciones, y hay un gran retardo al realizar el cálculo de la DFT de cada uno de los bloques. No ocurre lo mismo con la segunda condición, pues al tener M un valor muy elevado con respecto a Lh, entonces N ≅ M. A continuación se muestran los valores de M y N que se han tomado, en función de la longitud de Lx:

Lx > 1572000 ⇒ M = (2^19) - Lh + 1 ⇒ **N = (2^19) Þ L @ 5 secciones**

1572000 > Lx > 786000 ⇒ M = (2^18) - Lh + 1 ⇒ **N = (2^18) Þ L @ 5 secciones**

786000 > Lx > 393000 ⇒ M = (2^17) - Lh + 1 ⇒ **N = (2^17) Þ L @ 5 secciones**

393000 > Lx > 196000 ⇒ M = (2^16) - Lh + 1 ⇒ **N = (2^16) Þ L @ 5 secciones**

196000 > Lx > 98000 ⇒ M = (2^15) - Lh + 1 ⇒ **N = (2^15) Þ L @ 5 secciones**

Lx < 98000 ⇒ M = (2^14) - Lh + 1 ⇒ **N = (2^14) Þ L @ 6 secciones**

Se puede comprobar que siempre se escoge un valor para M tal que N sea una potencia de 2 sin necesidad de redondeo, así se asegura que siempre vamos a trabajar con aproximadamente 5 secciones. De esta manera, como se podrá comprobar posteriormente, se consigue trabajar en tiempo real. Si por el contrario se utilizan más secciones (tomando M más pequeño) el cálculo de las DFT sería algo más rápido pero habría que procesar demasiados bloques, y ocurre justo lo contrario si se utilizan menos secciones, pues entonces el cálculo de cada una de las DFT se hace extraordinariamente largo. Así, para el caso más común, 393000> Lx > 196000, se trabajará con aproximadamente 5 secciones de longitud

M = (2^16) - Lh + 1. Suponiendo Lx = 240000 muestras y Lh = 250 muestras se tendría M = 65287 muestras, entonces:

$$Lx / M = 3.6760 \Rightarrow L = ceil(Lx / M) = \textbf{4 secciones}$$

Al finalizar este apartado se hará un análisis del método Overlap & Add y se evaluará su eficiencia con los valores mostrados.

- **Señal correlación resultante: coeficientes de correlación útiles**.

Cuando se realiza la correlación para comparar dos señales en realidad lo que se está haciendo es superponer la más pequeña sobre la de mayor longitud e ir desplazando muestra a muestra la señal superpuesta sobre la inferior para comprobar si "encaja" más o menos bien la señal corta en la mayor, como si se tratase de un puzzle.

La primera superposición que se realiza, la cual da como resultado la primera muestra de la señal correlación, sería la de la última muestra de la señal corta sobre la primera muestra de la señal larga; a continuación se desplaza una muestra la señal corta sobre la larga, por lo que ya estarían las dos últimas muestras de la señal corta sobre las dos primeras de la señal larga; y así sucesivamente hasta que se llega al final de la señal larga. La última muestra de la señal correlación se corresponde a la comparación de la primera muestra de la señal corta con la última muestra de la señal larga.

Por lo tanto se puede observar claramente que la señal correlación tiene dos colas, al principio y al final de dicha señal, que no son útiles a efectos prácticos, pues dichas colas van a ser siempre valores próximos a cero, dado que son resultado de comparaciones no completas de ambas señales. Consideraremos una comparación completa cuando todas las muestras de la señal corta se comparen con muestras de la señal larga.

La señal correlación tiene una longitud de Lx+Lh-1 muestras, pero no necesitamos ni las primeras Lh ni las últimas Lh muestras de la señal correlación, por lo que la señal útil va desde la muestra número Lh hasta la número Lx, o sea, tendrá una longitud de Lx-Lh muestras.

De esta forma eliminamos muestras que no son útiles con el consiguiente ahorro en recursos, ya que no volveremos a procesar dichas muestras.

```
s=abs(sum(MAT));
s=s(Lh:Lx);     %Tomamos sólo las muestras que nos
                interesan, pues las colas de la correlación
                no nos sirven para nuestro estudio.
```

- **Normalización de los coeficientes de correlación:**

Por último y para que la evaluación de los resultados obtenidos sea correcta hay que normalizar los coeficientes de correlación, de forma que tengamos un valor máximo equivalente a la unidad para una comparación de dos señales idénticas, y dicho valor se aproxime a cero para la comparación de señales que no sean parecidas, por ejemplo dos señales aleatorias. De esta manera, aplicando la normalización adecuada a los coeficientes de correlación, se obtendrá tras la comparación una serie de resultados muy explícitos, ya que con sólo echar un vistazo a la señal correlación resultante se sabrá si se trata de la misma señal o no en la mayor parte de los casos. Habrá otros casos en los que no sea tan fácil tal apreciación, por lo que como se verá posteriormente habrá que analizar la comparación de ambas señales como un proceso estadístico para después implementar un algoritmo decisor.

Cuando se estudiaron las propiedades de la función correlación se pudo observar que la correlación cruzada verifica que:

$$|\mathbf{r_{xy}(l)}| \leq (r_{xx}(0)r_{yy}(0))^{1/2} = \mathbf{(E_x E_y)^{1/2}}$$

Y en el caso especial de que y(n) = x(n) entonces:

$$|\mathbf{r_{xx}(l)}| \leq r_{xx}(0) = \mathbf{E_x}$$

Esto significa que el valor máximo de la correlación cruzada es igual a la raíz cuadrada del producto de las energías de las señales a comparar.

Además también se vio que si una de las dos secuencias implicadas en el cálculo de la correlación cruzada se escala, la forma de la correlación no cambia, simplemente se produce un escalado de la misma de acuerdo con el escalado realizado sobre las secuencias originales. Así pues, el escalado carece de importancia y es a menudo conveniente, en la práctica, normalizar las secuencias de autocorrelación y correlación cruzada al rango entre –1 y 1. De hecho, para nuestro objetivos, nos conviene normalizar la correlación cruzada entre 0 y 1, lo cual se consigue trabajando con el valor absoluto. Por lo tanto la correlación cruzada normalizada se define como:

$$r_{\mathbf{xy}}\mathbf{(l)} = \mathbf{r_{xy}(l)/(r_{xx}(0)r_{yy}(0))^{1/2}}$$

Ahora $|r_{\mathbf{xy}}\mathbf{(l)}| \leq \mathbf{1}$, y de aquí que esta secuencia sea independiente del escalado de la señal.

Para realizar este escalado en Matlab nos basta con tomar ambas señales, x(n) y h(n) y sumar todas sus muestras al cuadrado:

$$\mathbf{Ex} = \sum_{\mathbf{n=-\infty,\infty}} \mathbf{x^2(n)}$$
$$\mathbf{Eh} = \sum_{\mathbf{n=-\infty,\infty}} \mathbf{h^2(n)}$$

Para el caso de h(n) si nos basta con tomar los valores de la señal, multiplicarlos por sí mismos y sumarlos, de forma que se obtiene el valor de su energía, pero no podemos hacerlo para el caso de x(n), pues al tratarse de una señal tan larga se tardaría demasiado tiempo en hacerlo y no se podría trabajar en tiempo real. Si pensamos que vamos a tratar de detectar siempre el mismo anuncio

podría realizarse el cálculo una vez y almacenarlo, de forma que fuese un parámetro de entrada más a la función que implementa el algoritmo de comparación. Sin embargo sería un algoritmo muy particularizado y sin modularidad y además para realizar las pruebas este método no nos vale. Por ello lo que se ha hecho es implementar de nuevo el algoritmo "*overlap and add*", de forma que calcularemos la energía de la señal x(n) por bloques. Basta con calcular la correlación cruzada de $x^2(n)$ con una secuencia de unos:

$$r_{xy}(l) = \sum_{n=-\infty,\infty} x(n)y(n-l) \qquad l = 0, \pm1, \pm2, \ldots$$

y(n) = 1 **n = 0, ±1, ±2, ...**

x(n) = x²(n) Þ **$r_{xy}(l) = \sum_{n=-\infty,\infty} x^2(n)$**

Para realizar estos cálculos se ha implementado la función '*energía.m*', cuyo código se puede ver en el Apartado IV del documento.

```
Eh=sum(h.^2);
Ex=energia(Lh,x,M);
s=s./sqrt(Ex*Eh); %Normalizamos cada uno de los
                  coeficientes de correlación por la
                 energía de las señales utilizadas para
                  su cálculo. El vector Ex se corresponde
                  con los parámetros de normalización
                  calculados con la función 'energía.m'.
```

De esta manera se consigue la normalización buscada con un consumo de recursos no excesivo, de forma que trabajamos en tiempo real.

3.4) Evaluación del método. Resultado final:

A continuación realizaremos algunas pruebas con un determinado anuncio para evaluar y comprobar la eficiencia computacional del método de comparación escogido.

Ya vimos para el método *'overlap & add'* que la secuencia x(n) de longitud Lx se divide en Lx/M secciones, cada una de longitud M, que la secuencia h(n) tiene longitud Lh, y que las longitudes de las secuencias para la convolución lineal son $N = 2^d \geq M + Lh - 1$. También se ha visto que para llevar a cabo la convolución o correlación rápida de dos secuencias de N puntos se requieren $12N\log_2 2N + 8N$ multiplicaciones reales. Por lo tanto para implementar el algoritmo de solapamiento y suma tendremos el siguiente número de multiplicaciones reales:

$$\mathbf{(Lx/\ M)\ (12N\log_2 2N + 8N) = R_m(S)}$$

Esto nos muestra que la longitud de las secuencias a correlar, N, debería ser pequeña, mientras que la longitud de las secciones de datos de x(n), M,debería ser aproximadamente igual a N. Idealmente $N = 2d \geq M + Lh - 1$.

A continuación se muestran algunas tablas en las que se recoge el número de operaciones necesario (incluyendo productos, sumas y todo tipo de operación matemática) y el tiempo empleado para realizar la comparación de dos anuncios. En sucesivas pruebas se irán cambiando los valores de N, M, Lh, ... para realizar un estudio comparativo. Cada uno de estos parámetros tendrá la función de variable en cada una de las tablas, los valores de dicha variable estarán resaltados. Estas pruebas se han realizado con el algoritmo *'overadd1.m'*, éste tiene algunas modificaciones con respecto al algoritmo que utilicemos finalmente para la detección en tiempo real, como ya se ha dicho antes. Por ello los valores que aparecen en las tablas no son indicativos de los que manejaremos en realidad, especialmente los referentes al tiempo empleado para cada comparación.

Así para Lh como variable tendremos:

Lx	Lh	M	N	Lx/M (Nº Secciones)	Número de Operaciones	Tiempo Empleado (seg.)
207767	**20**	65517	65536	4	**43426750**	**3.07**
207767	**200**	65337	65536	4	**43420090**	**3.07**
207767	**2000**	63537	65536	4	**43353490**	**3.19**
207767	**5000**	60537	65536	4	**43242490**	**3.13**

Vemos que al aumentar Lh no se obtiene ningún resultado apreciable en el número de operaciones a realizar, y por lo tanto y dado que no se modifica ningún otro parámetro el tiempo empleado en la comparación es aproximadamente el mismo. Esto se debe a los valores que toma cada uno de los parámetros que intervienen en la comparación. La contribución de Lh a la longitud de la DFT con la que se va a trabajar (N) es muy pequeña:

$$N = M + Lh - 1;\ M >> Lh;\ \Rightarrow\ \mathbf{N} \text{ » } \mathbf{M}$$

$$N = 2^d \geq M, \text{ aproximadamente.}$$

Por lo tanto, y como se puede observar en la tabla, tanto el número de secciones en que se divide el anuncio (Lx/M) como la longitud de la DFT a emplear (N) son los mismos para todos los casos, independientemente del valor de Lh. Si se calculara la correlación de ambas señales de forma directa en el tiempo el parámetro Lh si tendría una influencia significativa en el número de operaciones a realizar, pero al implementar el algoritmo de convolución rápida y trabajar con DFTs calculadas de forma eficiente (N igual a potencia de dos) nos encontramos con que la longitud de la DFT a calcular apenas depende de Lh.

Sin embargo a medida que aumenta Lh la calidad de la comparación es mucho mayor, dado que se están comparando más muestras, el bloque es cada vez un

orden de magnitud mayor. Por lo tanto será mucho más fiable la comparación cuanto mayor sea Lh.

Por lo tanto se llega a la conclusión de que la variable Lh no afecta en gran medida al número de operaciones realizadas al utilizar este método.

Si tomamos como variable M, entonces:

Lx	Lh	M	N	Lx/M (Nº Secciones)	Número de Operaciones	Tiempo Empleado (seg.)
207767	200	**133000**	262144	2	**98162591**	**121.82**
207767	200	**130873**	131072	2	**47616580**	**13.78**
207767	200	**65337**	65536	4	**43420093**	**3.23**
207767	200	**32569**	32768	7	**36528655**	**4.19**
207767	200	**16185**	16384	13	**34241303**	**144.54**

Podemos observar que al ir modificando el valor de M, también lo hacen N (N = 2d ≥ M + Lh – 1), y por supuesto el número de secciones con el que se trabaja en cada caso (Lx/M).

En este caso sí que hay una dependencia importante del número de operaciones realizadas respecto de la variable M, ya que según la longitud de las secciones, y por lo tanto el número de éstas, se realizarán más o menos operaciones. De hecho nuestro algoritmo está basado fundamentalmente en esta idea, dividir las señales muy largas en muchos bloques de pequeña longitud, de forma que el cálculo de las DFTs de estos bloques sea rápido.

De esta manera, si observamos la columna del número de operaciones veremos que conviene tomar M lo más pequeño posible, ya que cuanto menor es M, menor es el número de operaciones. Sin embargo si atendemos a la columna donde se

muestra el tiempo empleado en realizar la comparación nos encontramos con unos valores contradictorios, de los cuales se deduce que no hay una dependencia directa y proporcional entre M y el tiempo empleado, al igual que lo hay entre M y el número de operaciones.

Buscando una respuesta para explicar este hecho, se han analizado el número de operaciones y el tiempo empleado necesarios para realizar cada una de las operaciones internas del algoritmo de comparación, más concretamente aquellas operaciones que intervienen en el bucle *FOR* con el que implementamos el algoritmo de Solapamiento y Suma. Éstas son tres (se muestran en azul):

```
MAT=zeros(L,M*(L-1)+N);

for i=1:L
   DFTX=fft(x(M*(i-1)+1:min(M*i,Lx)),N);
   Y=DFTX.*DFTH;
   MAT(i,M*(i-1)+1:M*(i-1)+N)=ifft(Y,N)';
end
```

En la primera de ellas se calcula la DFT de los bloques en que se divide el anuncio a detectar, en la segunda se realiza el producto de las DFTs de ambas señales (la DFT del trozo de anuncio que se va extrayendo de la emisora se ha calculado con anterioridad, fuera del bucle *FOR*), y por último se calcula la IDFT de dicho producto y se hace una asignación de valores: en esta asignación se va guardando (en una matriz, por filas) el resultado de correlar cada uno de los bloques en que se divide el anuncio con la señal extraída de la emisora, además se hace de forma que en la matriz queden almacenados los datos con el solapamiento adecuado para después sumar las filas de dicha matriz, y así obtener la correlación deseada.

A continuación mostramos una tabla con los tiempos que se emplean en ejecutar cada uno de estos comandos, así como los de los comandos anteriores y posteriores al bucle:

	Operación 1 (DFT)	Operación 2 (Producto)	Operación 3 (IDFT y Asignación)	Suma por filas	Total bucle	Operaciones iniciales	Operaciones finales	Total
M = 133000	22,63	13,18	35,60	71,41				
	2,58	0,05	13,29	15,92	87,33	20,59	12,03	**119,95**
M = 130873	4,61	0,72	1,09	6,42				
	3,19	0,00	0,93	4,12	10,54	0,39	1,10	**12,03**
M = 65337	0,22	0,00	0,55	0,77				
	0,16	0,00	0,44	0,60				
	0,16	0,00	0,44	0,60				
	0,16	0,00	0,44	0,60	2,57	0,22	1,05	**3,84**
M = 32569	0,06	0,00	0,21	0,27				
	0,06	0,00	0,16	0,22				
	0,06	0,00	0,16	0,22				
	0,06	0,00	0,16	0,22				
	0,06	0,00	0,16	0,22				
	0,06	0,00	0,16	0,22				
	0,06	0,00	0,16	0,22	1,59	0,16	0,44	**2,19**
M = 16185	0,44	0,00	72,89	73,33				
	0,77	0,00	0,22	0,99				
	0,44	0,00	0,11	0,55				
	0,44	0,00	0,11	0,55				
	0,44	0,00	0,05	0,49				
	0,44	0,00	0,11	0,55				
	0,44	0,00	0,49	0,93				
	0,44	0,00	1,49	1,93				
	0,49	0,00	0,06	0,55				
	0,49	0,00	3,02	3,51				
	0,44	0,00	0,11	0,55				
	0,44	0,00	1,81	2,25				
	0,39	0,00	1,10	1,49	87,67	16,20	28,50	**132,37**

Antes de pasar a comentar los resultados hay que decir que esta tabla es el resultado de decenas de simulaciones. Los valores que se muestran son indicativos del proceso, ya que en la mayoría de las simulaciones se han obtenido valores aproximados en cada una de las celdas. Sin embargo estos tiempos no son totalmente estables y fijos, como ocurre con el número de operaciones que siempre toma el mismo valor, sino que dependen en cierta medida del PC con el que trabajemos: tamaño de la memoria virtual, potencia del microprocesador,

tamaño del sistema de archivos, ... Este hecho se pone especialmente de manifiesto para la operación 3 del bucle *FOR* (IDFT y asignación de valores), ya que incluye la asignación de un vector de grandes dimensiones a una matriz. O sea, es una operación de escritura en memoria, y por lo tanto su tiempo de ejecución es una variable no controlable que a veces toma valores muy altos, independientemente del tamaño del vector.

Se puede observar que los tiempos totales más bajos se obtienen para M = 65337 y M = 32569, o sea, para un número de secciones entre 4 y 7, como ya se vio anteriormente. Para M = 133000 y M = 130873 el tamaño de las secciones es muy grande, pues la señal tan sólo se divide en dos secciones, por lo tanto los tiempos de cálculo de DFT e IDFT son altos, sobre todo en el caso de M = 133000, ya que en este caso el tamaño de N es doble:

$$N = 2^d \geq M + Lh - 1;$$

$$M = 133000 \Rightarrow \mathbf{N = 262144}$$

$$M = 130873 \Rightarrow \mathbf{N = 131072}$$

Para M = 16185 comienza a aumentar de forma desproporcionada el tiempo total, ya que el número de veces que se ejecuta el bucle es muy superior y por lo tanto también lo es el número de accesos a memoria que implica la operación 3 del bucle.

Por lo tanto al evaluar el tiempo total empleado para realizar una comparación se observa que siempre se obtiene un mínimo al utilizar un valor de M tal que el número de secciones varíe entre 4 y 7, de ahí que para nuestro algoritmo final de detección se haya optado por tomar los siguientes valores para M en función de la longitud del anuncio (ya vimos estos valores anteriormente):

Lx > 1572000 $\Rightarrow$ M = (2^19) - Lh + 1 Þ **L @ 5 secciones**

1572000 > Lx > 786000 $\Rightarrow$ M = (2^18) - Lh + 1 Þ **L @ 5 secciones**

786000 > Lx > 393000 ⇒ M = (2^17) - Lh + 1 Þ **L** @ **5 secciones**

393000 > Lx > 196000 ⇒ M = (2^16) - Lh + 1 Þ **L** @ **5 secciones**

196000 > Lx > 98000 ⇒ M = (2^15) - Lh + 1 Þ **L** @ **5 secciones**

Lx < 98000 ⇒ M = (2^14) - Lh + 1 Þ **L** @ **6 secciones**

4) Optimización de parámetros en la comparación de tramas de audio:

Una vez que hemos visto en líneas generales el proceso a llevar a cabo para realizar la comparación de dos señales audio, estudiaremos algunos de los parámetros implicados en tal proceso, de forma que lleguemos a fijar unos criterios en los que apoyarnos para decidir cuál es el valor óptimo para cada uno de estos parámetros. Tales criterios se corresponderán con nuestros objetivos principales, la comparación efectiva de las tramas de audio con un margen de error pequeño y conseguir que la comparación y la detección se hagan en tiempo real.

Para conseguir que la probabilidad de error en la comparación de las señales sea pequeña se hace necesario un estudio estadístico, más o menos complejo, en el que analizaremos la representación estadística del proceso correlación cruzada de dos señales a través de una función de distribución.. A través del estudio de dicha función de distribución estadística y del ajuste de sus valores conseguiremos llegar al valor óptimo de los parámetros implicados en la comparación de tramas de audio, por este motivo se incluirá dicho análisis en este capítulo.

Además, una vez conocidos los valores óptimos que buscamos podremos implementar un algoritmo de detección con un decisor. En este algoritmo se tratará de afinar todavía aún más la probabilidad de error de la comparación y detección. Por último se realizarán pruebas para comprobar la eficacia real de dicho algoritmo.

4.1) Introducción.

Como ya hemos dicho, una vez estudiado el método de comparación que vamos a emplear sólo nos queda hacerlo óptimo, y esta optimización vendrá marcada por dos criterios. El primero de ellos es el de conseguir una comparación eficaz, de forma que cuando se comparen dos señales que sean iguales el comparador nos dé una alarma positiva (en realidad los anuncios nunca van a ser exactamente iguales, sino que nos referimos a emisiones distintas del mismo

anuncio publicitario, y por lo tanto habrá algunas distorsiones y variaciones en la amplitud y fase de las señales), y cuando sean distintas nos dé una alarma negativa, o simplemente no dé ninguna alarma (con señales distintas nos referimos a anuncios publicitarios diferentes). Por lo tanto estamos buscando reducir el margen de error al mínimo.

El segundo criterio es conseguir hacer las comparaciones en un tiempo suficientemente pequeño como para hacer el seguimiento de una emisora de radio o televisión, o sea, trabajar en tiempo real.

Uno de los parámetros implicados en este proceso de optimización es la frecuencia de muestreo de las señales audio con las que trabajamos; una vez decidido qué valor es el mejor para la frecuencia de muestreo se fijará dicho valor para la captura de la señal de radio o televisión. El otro parámetro a estudiar es el tiempo durante el cual se capturarán muestras, o lo que es lo mismo el número de muestras necesario para hacer una buena detección.

A priori puede parecer que deberíamos encontrar primero el valor óptimo para el número de muestras (o tiempo de captura) a utilizar en la comparación, con lo cual se cumpliría el criterio de obtener una comparación eficaz. Así, posteriormente y una vez fijado el número de muestras, se buscaría el valor adecuado para la frecuencia de muestreo, de forma que también se cumpla el criterio de trabajar en tiempo real. Sin embargo no es tan trivial, ya que ambos parámetros, número de muestras y frecuencia de muestreo están íntimamente relacionados entre sí, y por lo tanto no se puede hacer una correspondencia unívoca entre parámetros y criterios:

Nº de muestras = Frecuencia de Muestreo x Tiempo de Captura

Lh = Fs x Tc

Aquí vemos que el número de muestras no viene determinado sólo por el tiempo de captura, sino también por la frecuencia de muestreo, por lo tanto no se puede fijar primero el número de muestras para cumplir un criterio y después variar la frecuencia de muestreo para cumplir el otro, pues al cambiar la

frecuencia de muestreo cambiaríamos de nuevo el número de muestras. Por ello habrá que hacer un análisis con detenimiento y llegar a una solución de compromiso.

Nos interesa conseguir un valor alto para el número de muestras, de forma que la calidad de la comparación sea buena, por otra parte también nos interesa obtener un valor bajo para tal parámetro, de manera que el tiempo de procesado sea lo menor posible. Esto mismo ocurre con la frecuencia de muestreo, pues vemos que son parámetros directamente proporcionales.

4.2) Optimización de parámetros.

A continuación analizaremos cualitativamente los principales parámetros que intervienen en el algoritmo de comparación de tramas de audio. Se hará un análisis por separado de cada uno de ellos en la medida de lo posible, ya que ambos están íntimamente relacionados, como se ha podido comprobar anteriormente.

Hay otros parámetros comos son el número de canales a utilizar para la señal audio, un solo canal (mono) o dos canales (estéreo), y el número de bits a utilizar para codificar cada muestra, ocho bits (un byte) o dieciseis bits (dos bytes), que influyen decisivamente en la calidad de la forma de onda. Sin embargo para nuestra aplicación no ofrecen ninguna ventaja, sino todo lo contrario como se ha comentado con anterioridad. De ahí que estos parámetros se hayan fijado a priori, utilizando un solo canal y ocho bits por muestra.

4.2.1) Estudio de la frecuencia de muestreo:

Los archivos de audio digital pueden grabarse seleccionando la frecuencia de muestreo deseada. A medida que aumenta la frecuencia de muestreo, aumenta la calidad del sonido, de forma que debe ser lo suficientemente alta para que los

sonidos de alta frecuencia puedan recogerse con precisión. Sin embargo, para aplicaciones como la que nos ocupa, existe una razón para no utilizar las frecuencias de muestreo más altas, y ésta es que las frecuencias de muestreo altas necesitan gran capacidad de almacenamiento y por lo tanto un mayor tiempo de procesado. Además la degradación en la calidad no afecta a la probabilidad de error en la comparación, pues no nos interesa escuchar el anuncio, sino detectar su emisión.

Según el teorema de Nyquist, es posible repetir con exactitud una forma de onda si la frecuencia de muestreo es como mínimo el doble de la frecuencia de la componente de mayor frecuencia. Si no se cumple el teorema de Nyquist se produce un fenómeno denominado '*aliasing'*, que consiste en el solapamiento de muestras en el dominio de la frecuencia. Este solapamiento del espectro se traduce en una fuerte distorsión en el dominio temporal y en una degradación irreversible de la calidad. Por lo tanto para reproducir la voz humana con un mínimo de calidad se exige al menos una frecuencia de muestreo de 8 KHz.

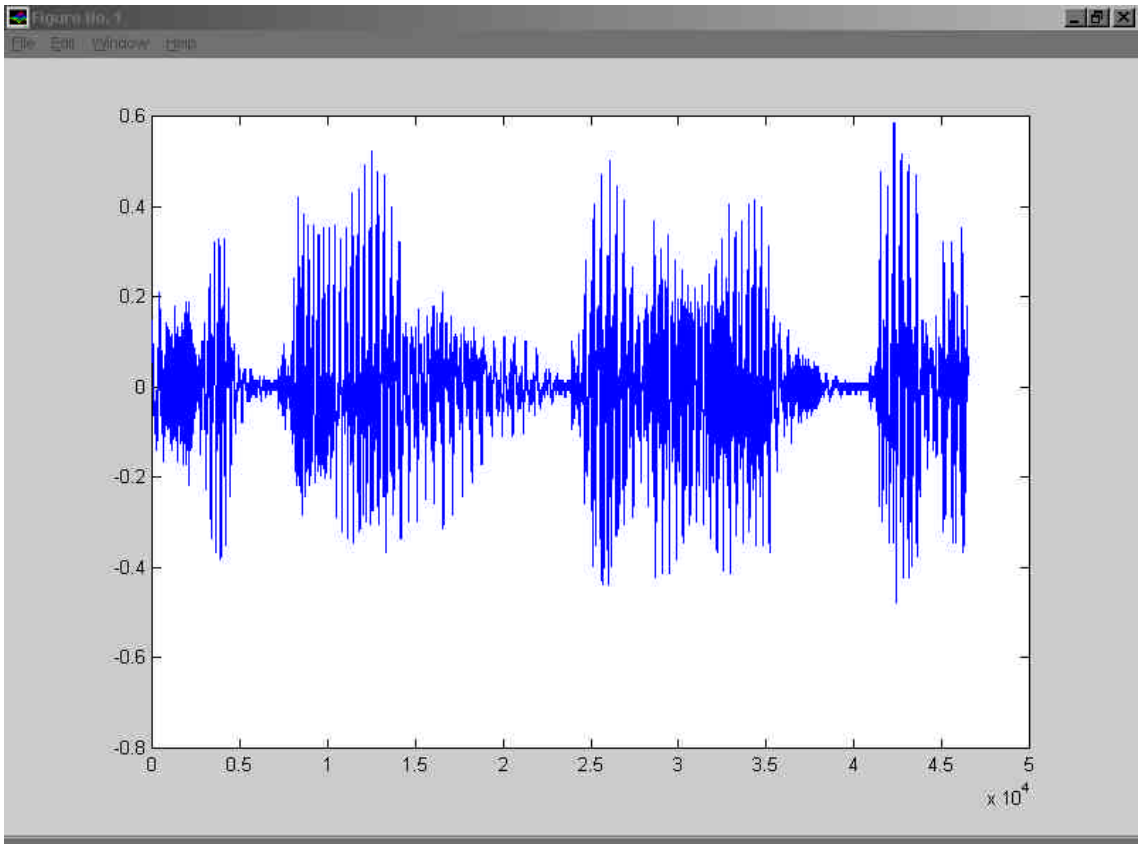
Sin embargo para la comparación de muestras de audio nos da igual que haya '*aliasing'*, por lo tanto no hay nada que nos impida trabajar con frecuencias de muestreo inferiores a 8 KHz. De hecho las pruebas realizadas para encontrar la frecuencia de muestreo óptima se han llevado a cabo para los siguientes valores de frecuencia: 4 KHz, 8 KHz, 11,025 KHz, 22,050 KHz y 44,1 KHz. Todos los valores, excepto el de 4 KHz, son valores normalizados, que se pueden configurar en la Sound Blaster y en el Wave Studio sin ninguna dificultad, ya que son valores muy habituales en la reproducción de señales audio.

Para tomar muestras a una velocidad de 4000 Hz ha sido necesario crear una rutina, que no hace otra cosa que tomar una de cada dos muestras procedentes de una señal muestreada a 8000 Hz. Este proceso se conoce con el nombre de '*diezmado*', y las operaciones implicadas apenas consumen recursos, consiguiendo reducir a la mitad el número de muestras del anuncio almacenado y por lo tanto disminuyendo de manera considerable el tiempo de procesado. En contrapartida el tiempo de captura de muestras de la emisora aumentará al doble, lo cual apenas incrementa el tiempo de comparación. Por lo tanto dicha frecuencia

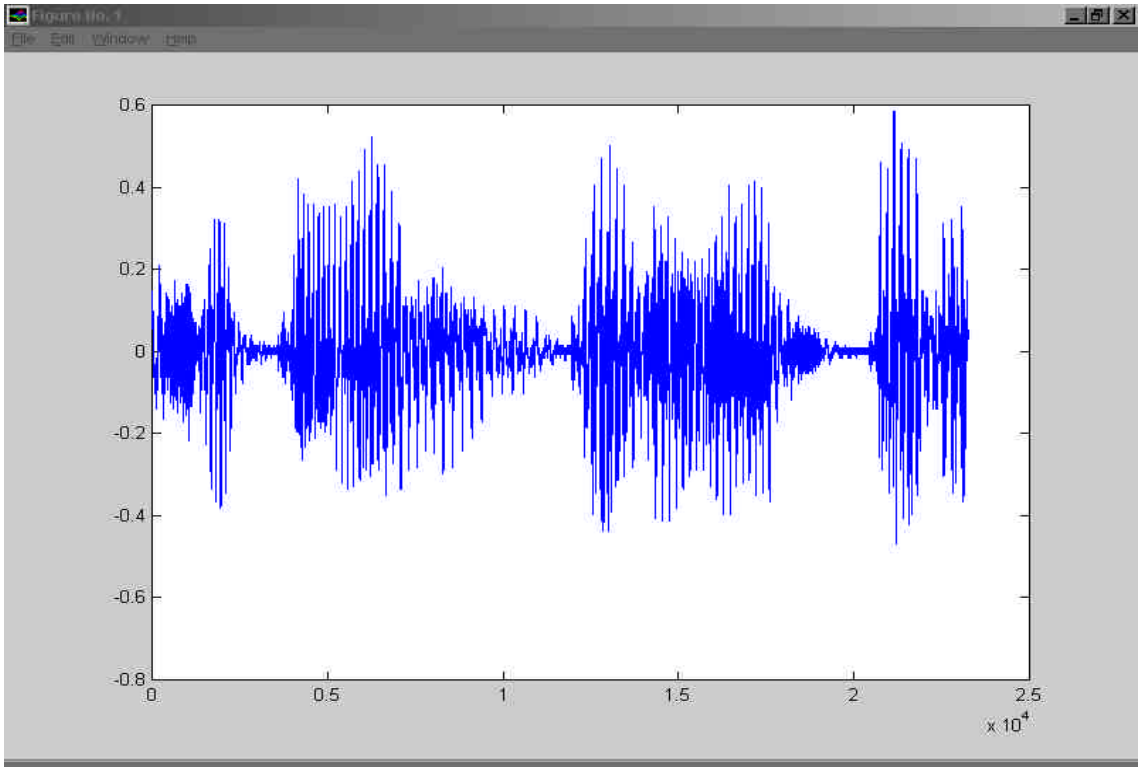
de muestreo nos ofrece una gran ventaja, lo cual quedará demostrado más adelante, cuando se analicen de forma cuantitativa los resultados de las pruebas llevadas a cabo para cada pareja de valores '*frecuencia de muestreo – tiempo de captura*'.

A continuación se muestran las gráficas del mismo anuncio grabado con distintas frecuencias de muestreo, de forma que se puede comprobar cómo varía el número de muestras en función de la frecuencia:

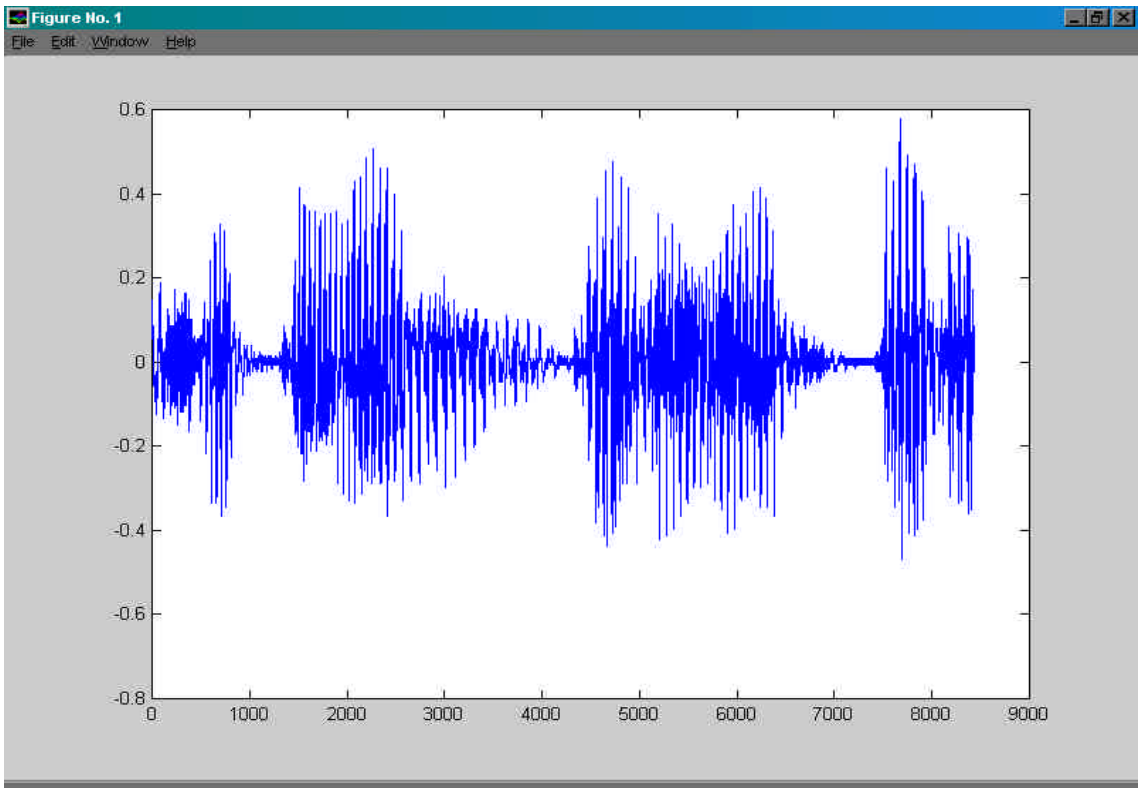
a) Fs = 44.1 KHz



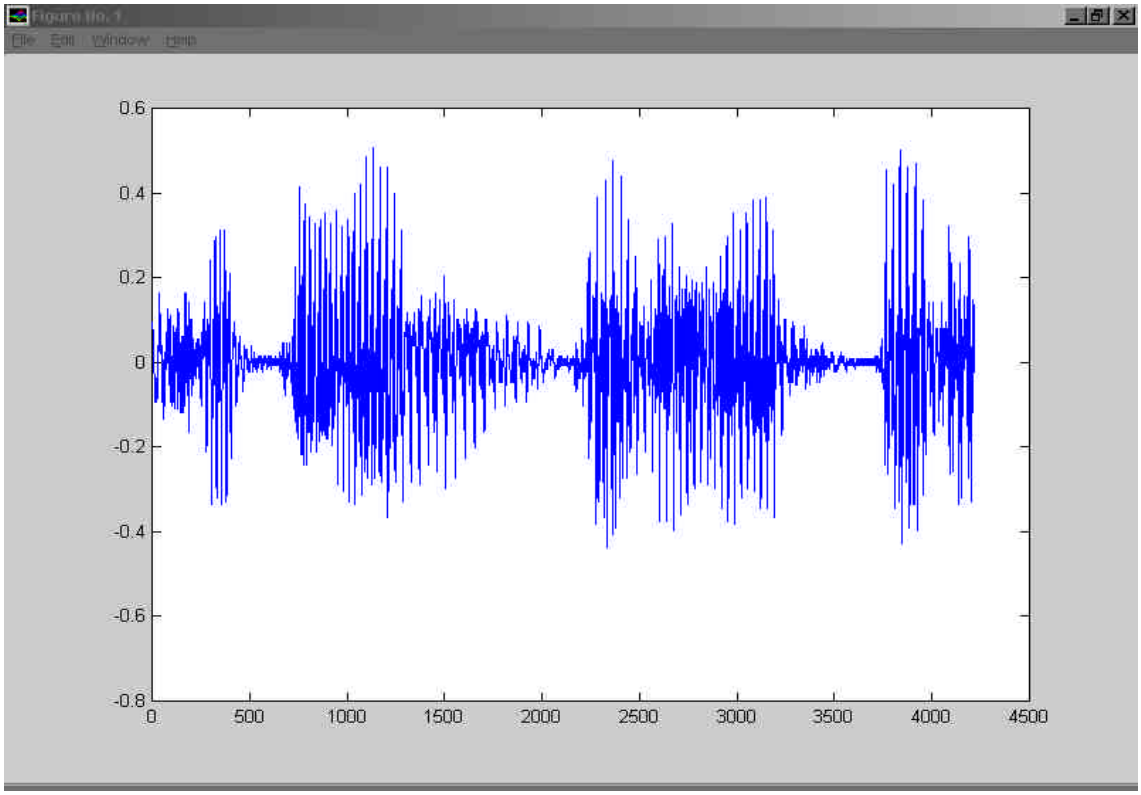
b) Fs = 22.05 KHz



c) Fs = 8.0 KHz



d) Fs = 4.0 KHz



Se puede comprobar cómo disminuye el número de muestras a medida que bajamos la frecuencia de muestreo, sin embargo se sigue conservando la forma de onda para las frecuencias de muestreo bajas (4 KHz), y por lo tanto sigue siendo útil para trabajar con ella.

4.2.2) Análisis de la longitud de trama:

Este es el principal parámetro de nuestro estudio, el cual va a estar muy ligado con la frecuencia de muestreo y con el tiempo de captura como hemos dicho antes, pues su valor viene dado por el de estas dos variables. Además el número de muestras con el que trabajemos va a determinar tanto la calidad de las comparaciones como el tiempo necesario para cada una de ellas, y por lo tanto el tiempo total para una detección.

En los algoritmos que se han implementado ésta variable está representada por Lh, que es la longitud de la señal corta que interviene en la correlación. En el algoritmo final Lh se define mediante una expresión que ya hemos visto:

Lh = Fs * Tc ; **Fs** ≡ Frecuencia de muestreo.

Tc ≡ Tiempo de captura de muestras de la emisora.

Sin embargo para realizar las pruebas no se trabaja adquiriendo muestras de la emisora, sino que se trabaja con formas de onda grabadas y preparadas para realizar este tipo de análisis, por lo tanto el valor Lh no viene definido por otras dos variables, sino que es una variable con la que jugar hasta que se encuentre un valor que se aproxime al óptimo, ya que es muy difícil encontrar el óptimo de forma exacta. Dicho valor será el menor número de muestras necesarias para obtener una correlación o comparación con la calidad suficiente como para decidir si dicha comparación es positiva o no. Esto es una correlación con una calidad suficiente para fijar un umbral máximo, tal que cuando algún coeficiente de correlación supere dicho umbral se deba a que se están comparando las mismas formas de onda. Por supuesto será tanto más fácil calcular dicho umbral cuanto mayor sea el número de muestras a comparar, lo cual es contraproducente con la idea de procesado en tiempo real.

Las comparaciones nunca se realizan con los mismos anuncios, sino con diferentes emisiones de estos, por lo tanto nunca dos formas de onda a comparar serán totalmente iguales, en cuyo caso se debería obtener un máximo igual a la unidad, sino que habrá variaciones en la amplitud y fase de las muestras. Este hecho hace que no sea nada fácil encontrar el valor óptimo para la longitud de la trama a comparar, y por lo tanto que haya un margen de error superior al que exige la aplicación que tratamos de desarrollar.

Este es el motivo por el que habrá que buscar alguna solución en el algoritmo decisor, y ésta no es otra que hacer que el tiempo de comparación sea muy corto, de forma que a lo largo de la emisión de un anuncio se puedan realizar varias

comparaciones. De esta manera se consigue reducir muchísimo el margen de error.

4.3) Necesidad de un estudio estadístico:

Para poder medir este error del que estamos hablando no basta con hacer algunas comparaciones y contabilizar el número de éxitos y fracasos del algoritmo de comparación, sino que habrá que llevar a cabo cientos de comparaciones, o lo que es lo mismo, hacer un estudio estadístico del proceso, de forma que la función error quede bien definida. Así, su valor será muy aproximado a la realidad y podremos decidir qué frecuencia de muestreo y qué tiempo de captura son los que más nos convienen en función de los resultados obtenidos tras el análisis estadístico.

Por lo tanto todas las pruebas que hemos mencionado hasta ahora no tienen otro fin que el de desarrollar un método estadístico que nos permita valorar de forma cuantitativa el proceso de comparación y en definitiva la aplicación que se lleva a cabo.

Así el objetivo que se persigue ahora es ajustar los valores de la correlación a una función de distribución, de forma que posteriormente se tenga un proceso monitorizado del que se puedan obtener las probabilidades de detección correcta, detección incorrecta, fallo, ...

Para ajustar dicha función de distribución haremos múltiples pruebas con varios anuncios. Así, se grabarán diferentes emisiones de una misma emisora, y también de distintas emisoras, del mismo anuncio, de manera que nos acerquemos lo más posible al caso real, y que la varianza del proceso estadístico que estamos calculando sea pequeña.

Además esto lo haremos para distintas frecuencias de muestreo y distintas longitudes de la señal a comparar, de manera que para cada pareja *frecuencia de*

muestreo-tiempo de captura de señal se obtendrá una curva de de distribución de probabilidad distinta, y consecuentemente diferentes probabilidades de detección correcta e incorrecta. Así sabremos cuál es la longitud de trama de audio mínima necesaria para obtener la calidad deseada en el proceso de correlación. Con este valor y examinando las distintas curvas obtenidas será muy sencillo escoger la pareja de valores *frecuencia-tiempo de captura* más apropiada a nuestro objetivo.

4.4) Representación estadística del proceso de correlación:

A continuación se va a hacer un breve repaso de algunas de las características del proceso correlación, más particularmente del valor absoluto de la correlación normalizada, pues es con este valor con el que estamos trabajando. Por lo tanto se estudiará el valor de los coeficientes de correlación para decidir qué función de distribución probabilística puede ajustarse a dichos valores, y por lo tanto ser representativa del proceso de comparación de dos tramas de audio.

4.4.1) Introducción. Análisis del proceso correlación:

En los capítulos 2 y 3 de este documento se ha hablado del proceso de correlación entre dos señales y de sus propiedades, y se ha hecho un estudio pormenorizado de su cálculo. Sin embargo lo que más nos interesa en este momento es el valor que alcanza cada uno de los coeficientes de correlación, de forma que obtengamos los límites superior e inferior de dichos coeficientes.

Ya vimos que se hacía necesario normalizar los coeficientes de correlación, de forma que se obtenga un valor máximo equivalente a la unidad para una comparación de dos señales idénticas, y dicho valor se aproxime a cero para la comparación de señales que sean desiguales. Así, aplicando la normalización adecuada a los coeficientes de correlación se conseguirá que los límites inferior y superior sean 0 y 1 respectivamente.

También se vio que si una de las dos secuencias implicadas en el cálculo de la correlación cruzada se escala, la forma de la correlación no cambia, sino que simplemente se produce un escalado de la misma de acuerdo con el escalado realizado sobre las secuencias originales. Así pues, es conveniente normalizar la secuencia de correlación cruzada al rango entre –1 y 1.

Cuando se estudiaron las propiedades de la función correlación se pudo observar que la correlación cruzada verifica que:

$$|\mathbf{r_{xy}(l)}| \leq (r_{xx}(0)r_{yy}(0))^{1/2} = \mathbf{(E_xE_y)^{1/2}}$$

Y en el caso especial de que y(n) = x(n) entonces:

$$|\mathbf{r_{xx}(l)}| \leq r_{xx}(0) = \mathbf{E_x}$$

Entonces el valor máximo de la correlación cruzada es igual a la raíz cuadrada del producto de las energías de las señales a comparar. De esta forma se consigue que los coeficientes de correlación estén comprendidos entre –1 y 1. Llegados a este punto nos encontramos con multitud de funciones de distribución probabilística válidas para representar al proceso, pues el único requisito que hay que cumplir es que el dominio de la distribución sea (-1,1).

Recordemos, sin embargo, que para comparar dos señales no nos conviene trabajar con números negativos, ya que el proceso de comparación y detección se complica demasiado. Por lo tanto, lo ideal es trabajar con un dominio comprendido entre 0 y 1. Por esta razón, para nuestro objetivos, es adecuado normalizar la correlación cruzada entre 0 y 1, lo cual se consigue trabajando con el valor absoluto de la correlación cruzada normalizada, que se define como:

$$|\mathbf{r_{xy}(l)}| = |\mathbf{r_{xy}(l)/(r_{xx}(0)r_{yy}(0))^{1/2}}| \leq \mathbf{1}$$

Entonces estamos buscando una función de distribución probabilística que se ajuste a las condiciones vistas, o sea, cuyo dominio venga dado por el valor (0,1).

Así, la función que se ajusta a estos requisitos es la función Beta, que no siendo de las funciones más comunes o utilizadas sí que es muy adecuada para representar estadísticamente el valor absoluto del proceso correlación cruzada normalizada. A continuación se analizarán la función Beta y sus parámetros y características, de forma que quede claro cómo se ajusta el proceso de comparación o correlación a la función Beta y cómo se pueden calcular las probabilidades de acierto y de error de manera monitorizada por el proceso estadístico.

4.4.2) Función de distribución Beta. Carácterísticas y ajuste.

La función de distribución Beta está relacionada con otras distribuciones continuas que se definen en un intervalo infinito, en concreto con la distribución Gamma (de hecho esta relación se define a través de la función Gamma de Euler). Por este motivo vamos a definir en primer lugar la distribución Gamma, y después pasaremos a hablar de la función de distribución Beta:

- **Distribución Gamma:**

Esta distribución depende de dos parámetros λ y k denominados parámetros de escala y de forma, respectivamente. Es decir, al variar k varía la forma de la distribución, mientras que al variar λ sólo varía la escala de la distribución.

$$f(x) = \frac{\lambda^k}{\Gamma(k)} x^{k-1} e^{-\lambda x} \qquad 0 < x < \infty, \lambda > 0, k > 0$$

Donde la función Gamma de p>0, Γ (p), viene dada por:

$$\Gamma(p) = \int_0^\infty x^{p-1} e^{-x} dx$$

Si p = 1/2 entonces se tiene que $\Gamma(1/2) = \sqrt{\pi}$.

Si p > 1 entonces se tiene que $\Gamma(p) = (p-1)\Gamma(p-1)$.

Si p es entero, entonces se cumple que $\Gamma(p)=(p-1)!$.

Si definimos el valor del parámetro λ en función del parámetro k y del parámetro μ según la expresión $\lambda = k/\mu$, se tiene que la función de densidad se escribe:

$$f(x) = \left(\frac{1}{\Gamma(k)}\right)\left(\frac{k}{\mu}\right)^{k} x^{k-1} e^{-\frac{k}{\mu}x}$$

En esta expresión, el parámetro μ determina la localización de la distribución (μ es la media de la distribución Gamma), y el cociente μ^2/k determina la forma de la distribución (μ^2/k es la variancia de la distribución Gamma).

Esta distribución es usada para modelar datos que presentan asimetría positiva. Veamos algunos casos particulares:

Para **k = 1** se tiene la **distribución exponencial**.

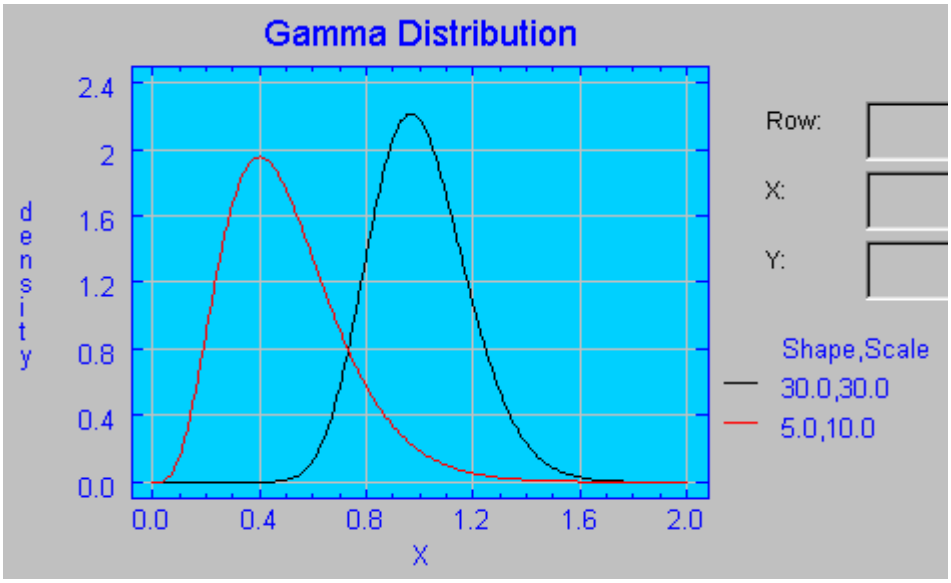
Para **k = 1 y** m **= 1** se tiene la **distribución exponencial estándar**.

Para **k = n/2 y** m **= n** se tiene la **distribución ji-cuadrado con n grados de libertad**.

Esta distribución se ha aplicado a los tiempos de vida de sistemas eléctricos y mecánicos, a la abundancia de especies animales, a períodos de incubación de enfermedades infecciosas, etc.

En el siguiente gráfico se muestran dos funciones de densidad Gamma, una con parámetros de escala (Scale) y de forma (Shape) igual a 30 y otra

con parámetro de escala 10 y parámetro de forma 5.



- **Distribución Beta:**

Las distribuciones de probabilidad continuas normalmente se definen como funciones distintas de cero en un intervalo infinito, pero es muy útil disponer de otra clase de distribuciones que se puedan usar para modelar fenómenos restringidos a un intervalo finito de valores posibles. Una distribución de esta clase, la distribución Beta, es muy útil para modelar el comportamiento probabilístico de determinadas variables aleatorias, como las proporciones, o como la correlación normalizada de dos tramas de audio, limitadas a caer en el intervalo (0,1).

Como hemos dicho anteriormente la distribución Beta está muy relacionada con la función Gamma de Euler, veamos su forma funcional:

$$f(x) = \frac{G(a+b)}{G(a)\,G(b)}\, x^{a-1}(1-x)^{b-1} \qquad 0 < x < 1$$

$$= 0 \qquad x < 0,\ x > 1$$

donde α y β son constantes positivas.

La distribución Beta depende de dos parámetros, p (forma) y q (escala), ambos positivos. Se denomina B(p,q).

$$f(x) = \frac{1}{B(p,q)} x^{p-1}(1-x)^{q-1} \qquad 0 < x < 1$$

Donde la función Beta B(p,q) viene dada, para p y q positivos, por:

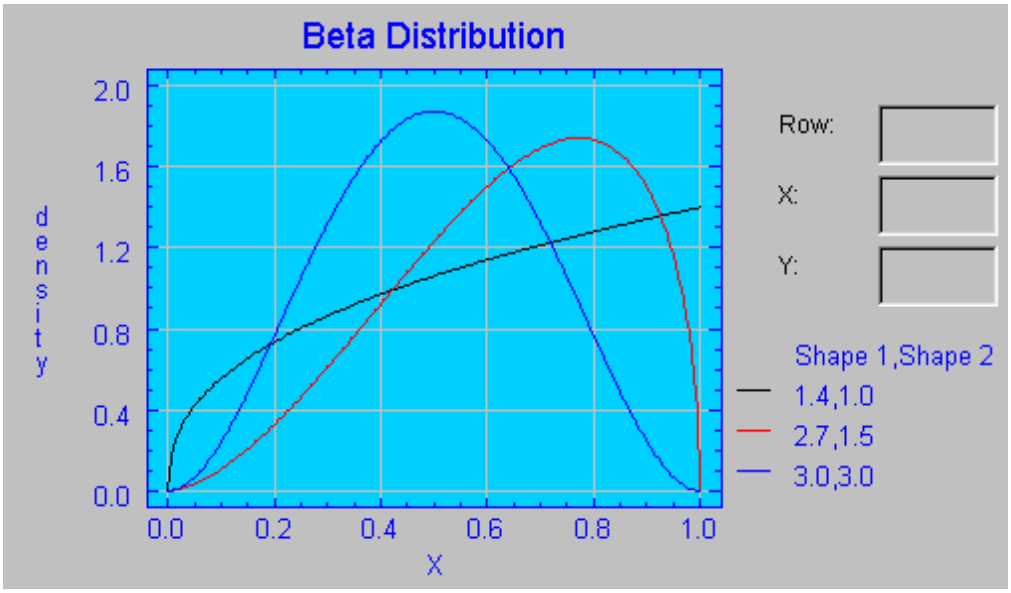
$$B(p,q) = \int_0^1 x^{p-1}(1-x)^{q-1}dx$$

La función Beta tiene la siguiente propiedad, a través de la cual se relaciona con la función Gamma.

$$B(p,q) = \frac{\Gamma(p)\,\Gamma(q)}{\Gamma(p+q)}$$

La distribución B(1,1) equivale a la distribución Uniforme U(0,1).

En la siguiente figura se presentan tres funciones de densidad Beta, cada una de ellas con parámetros p (Shape 1) y q (Shape 2) distintos.



En los últimos años la distribución beta ha encontrado importantes aplicaciones en la *inferencia bayesiana*, donde los parámetros se consideran como variables aleatorias y existe la necesidad de tener una densidad de probabilidad lo suficientemente flexible para el parámetro θ de la distribución binomial, que toma valores distintos de cero sólo en el intervalo de 0 a 1. Por flexible nos referimos a que la densidad de probabilidad puede tomar una amplia variedad de formas diferentes, así puede tomar las formas vistas en la gráfica, la forma de una distribución Uniforme, U(0,1), ..., como se podrá comprobar en el siguiente capítulo.

Este hecho hace que prácticamente se pueda modelar cualquier proceso cuyos valores varíen entre cero y uno con la función de distribución Beta.

Después de ver algunas características de la función de distribución Beta pasaremos a estudiar sus parámetros y la forma de modelar nuestro proceso a través de dicha función. Para ello vamos a estudiar antes las rutinas existentes en Matlab para trabajar con la función Beta:

- **BETAFIT.**

Calcula los parámetros (p y q) y sus intervalos de confianza para una serie de datos que tengan una distribución probabilística Beta.

Betafit(x) devuelve la estimación de máxima verosimilitud de los parámetros de la distribución beta, cuyos datos se pasan como vector de entrada, x.

[phat,pci] = betafit(x,alpha) da la estimación de máxima verosimilitud de los parámetros, así como un intervalo de confianza para cada parámetro dado por un porcentaje del 100(1-alpha). Los datos se dan en el vector de entrada, x. El parámetro alpha es opcional, y por defecto su valor es 0.05, correspondiendo a un intervalo de confianza del 95 %.

- **BETAPDF.**

Calcula la función densidad de probabilidad Beta.

Y = Betapdf(x,a,b) devuelve la función densidad de probabilidad Beta, de parámetros a y b, para los valores de abcisas definidos en x. El tamaño de Y es el mismo que el del vector x.

- **BETACDF**.

Calcula la función de distribución acumulativa Beta.

P = Betacdf(x,a,b) devuelve la función de distribución acumulativa para una función Beta de parámetros a y b en los puntos designados por el vector x. El tamaño de P es el mismo que el del vector x.

- **BETAINV.**

Calcula la inversa de la función de distribución acumulativa Beta.

X = Betainv(p,a,b) devuelve la inversa de la función de distribución acumulativa para una función Beta de parámetros a y b en los puntos designados por el vector x. El tamaño de P es el mismo que el del vector x.

BETAINV utiliza el método de Newton para converger a esta solución.

- **BETARND.**

Genera matrices con números aleatorios que siguen una distribución Beta.

R = Betarnd(a,b) genera una matriz con números aleatorios que siguen una distribución Beta con parámetros a y b.

- **BETASTAT.**

Calcula la media y la varianza para una distribución Beta dada.

[M,V] = Betastat(a,b) devuelve la media y la varianza de la función de distribución Beta de parámetros a y b.

Para este proyecto no es necesario emplear todas las rutinas vistas, sino algunas de ellas, especialmente aquella que nos permite ajustar unos determinados datos a una función de distribución Beta ("*betafit*"), devolviéndonos como argumentos de salida los parámetros de la distribución. También se obtiene como argumento de salida un intervalo de confianza, que podemos elegir en cierto modo con el parámetro de entrada "*alpha*", aunque dicho dato no será necesario.

A continuación se va a describir el proceso a seguir para ajustar los datos obtenidos en alguna de las pruebas de comparación de tramas de audio a una función de probabilidad Beta, en el próximo capítulo se hablará con más detenimiento de cada una de estas pruebas, así como de sus resultados.

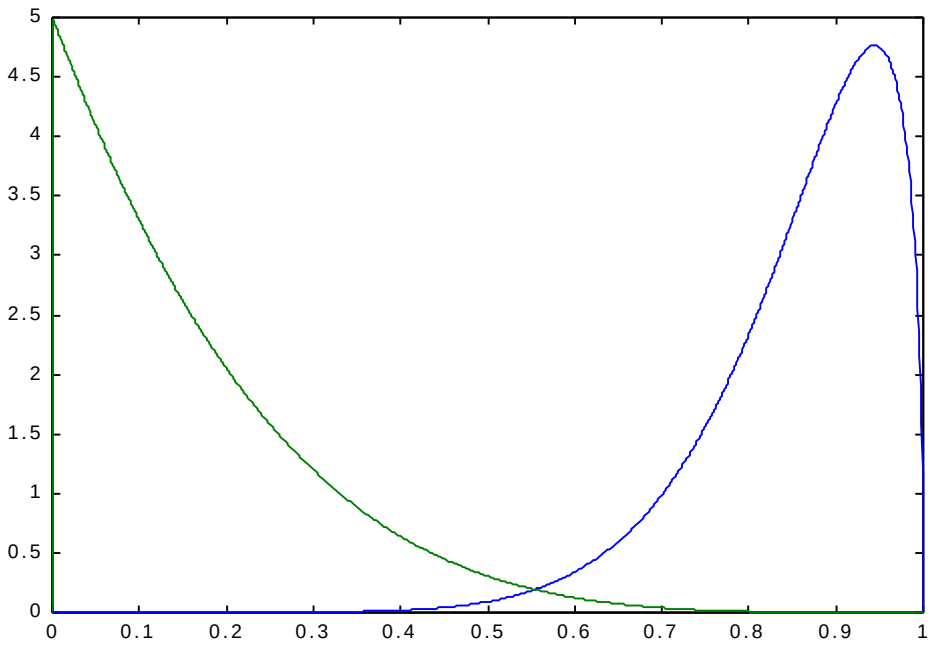
Supongamos que realizamos varias comparaciones con distintas emisiones del mismo anuncio publicitario, todas con la misma frecuencia de muestreo, y que para todas las comparaciones se utiliza el mismo número de muestras a comparar. Entonces tras hacer cada comparación, con la función "*overadd1*", almacenaremos el máximo coeficiente de correlación por un lado y cincuenta coeficientes tomados aleatoriamente de la función correlación por otro lado (en estos cincuenta valores nunca se incluirá el máximo coeficiente de correlación ni ninguno de los valores adyacentes, ya que esto podría conducir a errores).

Si realizamos veinte comparaciones obtendremos dos vectores, el primero de ellos formado por los veinte máximos coeficientes de correlación. Este vector es representativo de las comparaciones positivas y nos da una ligera idea de cual debe ser el umbral de detección que marque el criterio de comparación positiva o negativa. El segundo vector estará formado por 1000 valores aleatorios, todos distintos del máximo coeficiente de correlación de cada una de las comparaciones.

Este vector representa a las comparaciones negativas, ya que está formado por valores correspondientes a comparaciones de dos trozos de señal distintos.

Así, con cada uno de estos vectores se obtendrá una función de distribución Beta con la función Betafit, a la cual se le pasará como argumento de entrada cada uno de los vectores y nos devolverá como argumentos de salida los parámetros de cada una de las dos distribuciones.

Una vez conocidos los parámetros de una función de distribución Beta es muy fácil obtener su forma con la función Betapdf. La forma de ambas funciones de distribución es fácil de imaginar, pues la correspondiente a los máximos coeficientes de correlación tendrá un máximo cercano a la unidad del eje de abscisas, y la correspondiente a los coeficientes de correlación aleatorios tendrá dicho máximo próximo al origen. Seguidamente se muestra una de estas gráficas como ejemplo:



Una vez obtenidas dichas gráficas sólo hay que saber leer en ellas, pues para un umbral dado y con la función Betacdf se calcula de forma rápida y sencilla

cualquier probabilidad, entre ellas la probabilidad de falsa alarma (probabilidad de que no se emita el anuncio y el detector contabilice una detección) y la probabilidad de pérdida (probabilidad de que se emita el anuncio y nuestro algoritmo no lo detecte). Con estas probabilidades, para las que nos fijaremos unos valores objetivo, y eligiendo el umbral de decisión adecuado, será muy sencillo llegar a una conclusión en cuanto a la frecuencia de muestreo y el tiempo de captura óptimos.

4.4.3) Pruebas realizadas. Probabilidad de pérdida y de falsa alarma.

A continuación se verán los resultados de los experimentos realizados, mostrándose cada una de las gráficas obtenidas para cada pareja frecuencia de muestreo-tiempo de captura. Asimismo también se va a dar un valor umbral, que se corresponde en cada caso con el valor de abscisas del punto en el que se produce el cruce de las dos curvas, y las probabilidades de pérdida y falsa alarma que se obtienen para dicho umbral.

Hay que tener en cuenta que los umbrales que se van a mostrar con cada figura no son los umbrales óptimos, dado que umbral óptimo será aquel que proporcione valores para ambas probabilidades cercanos al objetivo marcado. A priori, para este objetivo vamos a fijar un valor bastante estricto, de forma que sea prácticamente imposible tener una falsa detección o una pérdida de detección. El límite inferior de ambas probabilidades con el que trataremos de trabajar será de 1×10^{-6}, por lo tanto el umbral escogido para cada una de las parejas frecuencia-tiempo no debería ser el correspondiente al cruce de ambas distribuciones, sino que se debe jugar con varios valores hasta que nos aproximemos al objetivo buscado. Sin embargo este proceso sería muy engorroso debido al elevado número de experimentos, así que tomando el umbral de decisión como el punto en el que se cortan ambas curvas conseguiremos hacernos una idea muy aproximada de cuáles son la frecuencia de muestreo y el tiempo de captura óptimo, entonces será el momento de jugar con el umbral para encontrar los valores deseados.

Para realizar estos experimentos nos hemos valido de varias rutinas implementadas en Matlab, cuyo código se mostrará al final de este documento. En estos códigos se podrá comprobar que no se han realizado un experimento por cada pareja frecuencia-tiempo, ya que algunos de ellos no tienen ningún sentido debido a que es imposible trabajar en tiempo real. Así, para una frecuencia de muestreo de 44.1 KHz se han hecho algunos experimentos a modo de ilustración, pero resulta evidente que no va a ser esta nuestra frecuencia de muestreo óptima, ya que conlleva un elevadísimo número de muestras que hacen imposible trabajar en tiempo real.

Para la realización de los experimentos se han realizado 12 grabaciones correspondientes a cuatro anuncios distintos, de forma que a cada anuncio publicitario le corresponden tres grabaciones, dos de ellas son distintas emisiones del anuncio de la misma emisora y la tercera se ha grabado de una emisora diferente, de forma que las pruebas a realizar sean lo más objetivas posible.

Para cada pareja frecuencia-tiempo vamos a obtener 24 valores del máximo coeficiente de correlación y 1200 valores aleatorios de dicha correlación. Estos se obtendrán comparando una emisión de cada anuncio con cada una de las otras dos hasta tres veces, por lo tanto se realizarán seis comparaciones por cada anuncio, que multiplicadas por cuatro anuncios dan como resultado un total de 24 comparaciones para cada pareja frecuencia-tiempo.

Para cada comparación o correlación salvaremos los siguientes valores:

- Máximo de los coeficientes de correlación, útil para establecer un umbral que nos dé la probabilidad de detección, así como el valor de la abscisa para el que se obtiene.

- Cincuenta coeficientes tomados aleatoriamente de la función correlación resultante, exceptuando el máximo coeficiente de correlación y los valores que resultan a su alrededor en una longitud de Lh muestras, pues estos pueden falsear el ajuste de la función densidad de probabilidad.

Esto lo haremos para distintas frecuencias de muestreo y distintas duraciones de la señal tomada de la radio como ya hemos dicho, o sea se hará una simulación para cada una de las parejas resultantes de tomar una frecuencia y un tiempo de captura. Estos son los valores con los que vamos a trabajar:

Tiempos de captura
2 seg
1 "
1/2 "
...
1/128 "

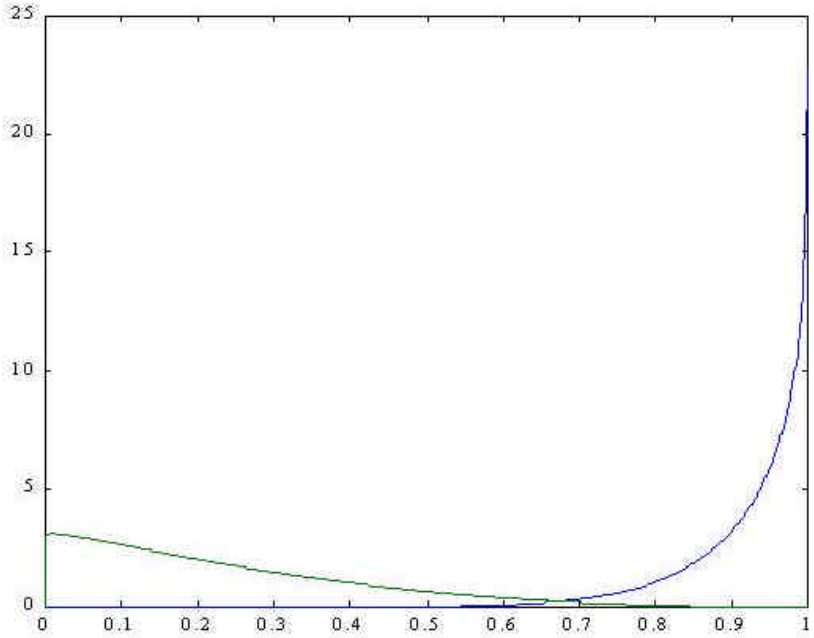
Frecuencias de muestreo
44.100 KHz
22.050 "
11.025 "
8.000 "
4.000 "

- **Resultados de los experimentos realizados:**

A continuación mostramos los resultados obtenidos para cada pareja frecuencia de muestreo-tiempo de captura.

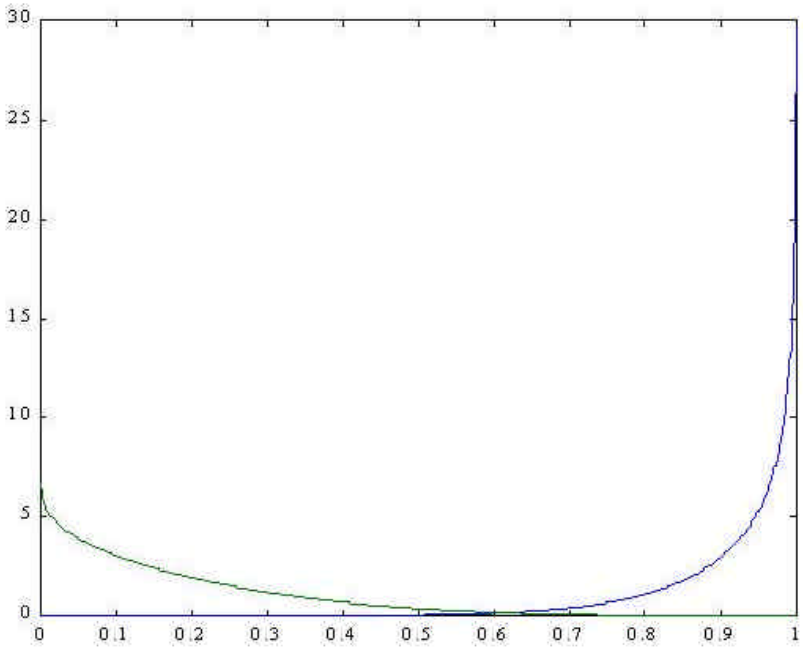
Fs = 44100; Tc = 1/128 seg

Umbral = 0.6745; Pérdida = 0.0163; Falarm =0.0219



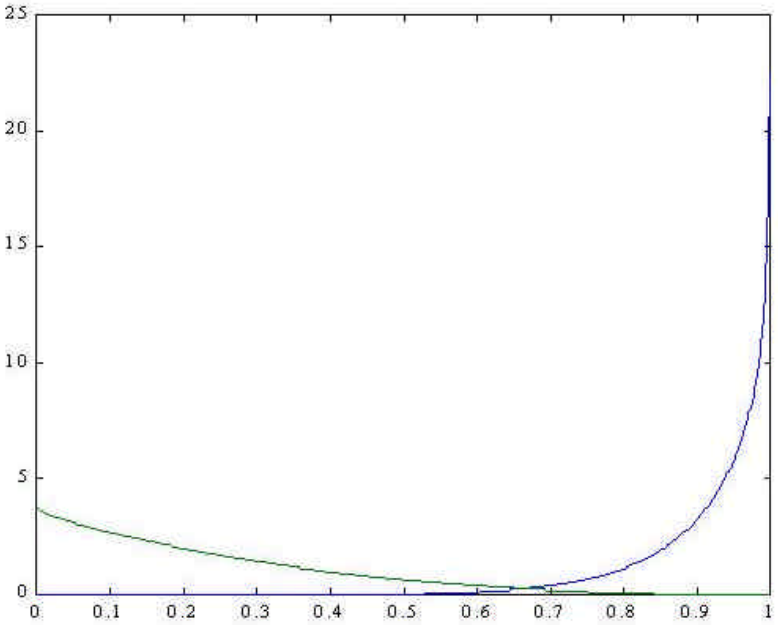
Fs = 44100; Tc = 1/64 seg

Umbral = 0.6126; Pérdida = 0.0125; Falarm =0.0142



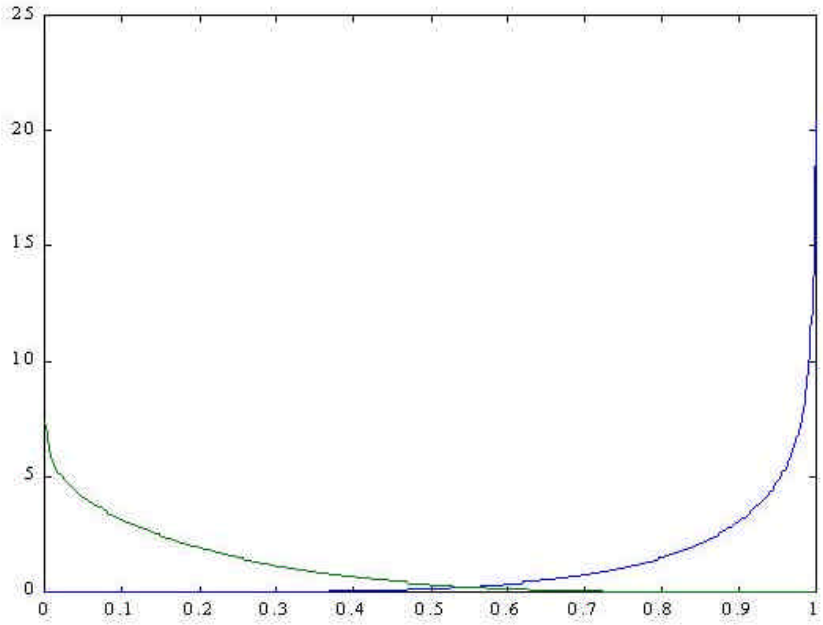
Fs = 22050; Tc = 1/128 seg

Umbral = 0.6641; Pérdida = 0.0172; Falarm =0.0228



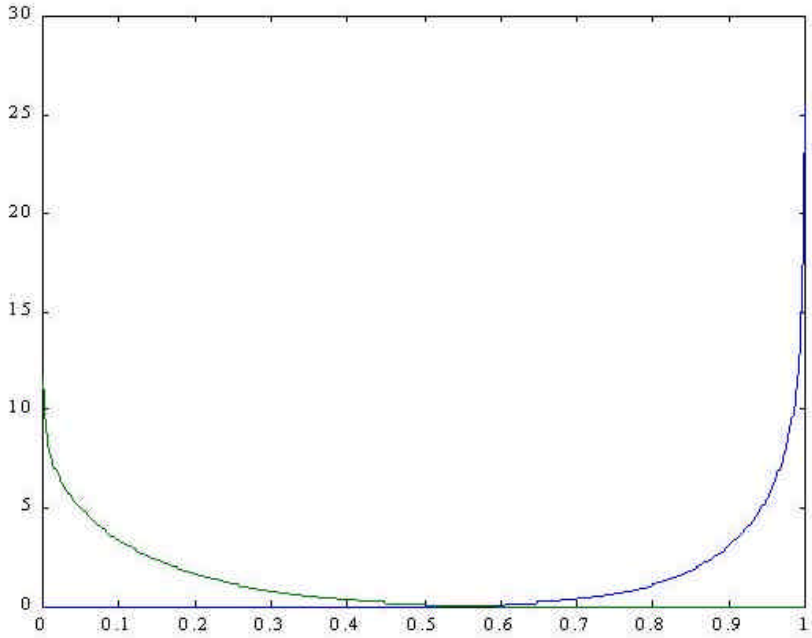
Fs = 22050; Tc = 1/64 seg

Umbral = 0.5484; Pérdida = 0.0220; Falarm =0.0219



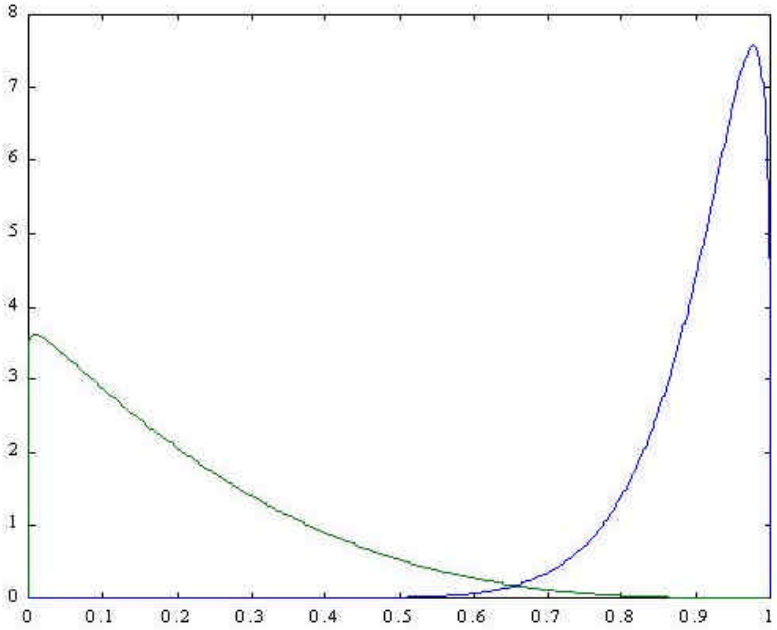
Fs = 22050 Hz; Tc = 1/32 seg

Umbral = 0.5572; Pérdida = 0.0047; Falarm =0.0050



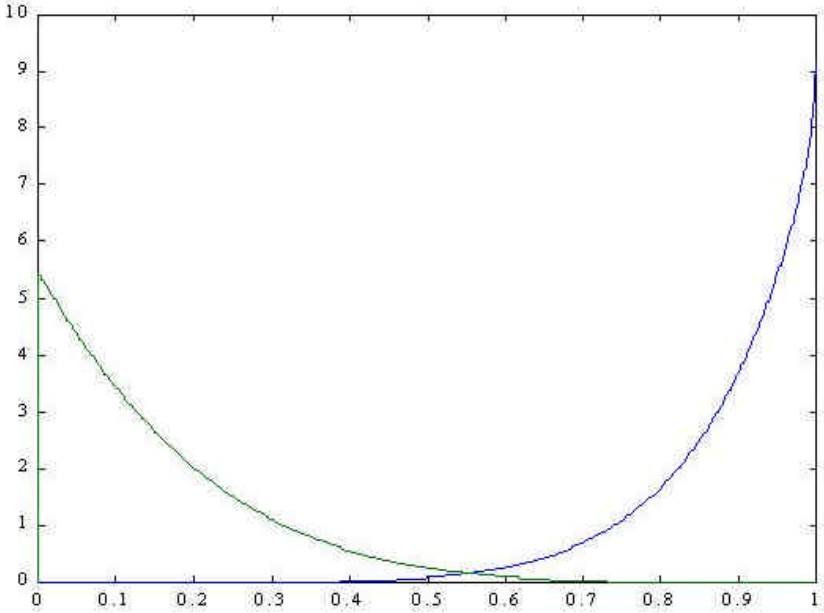
Fs = 11025 Hz; Tc = 1/128 seg

Umbral = 0.6565; Pérdida = 0.0098; Falarm =0.0154



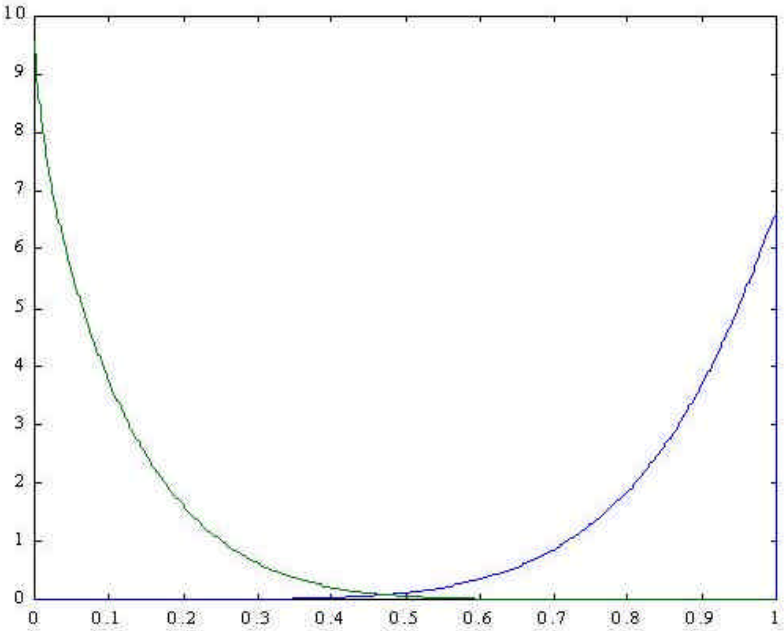
Fs = 11025 Hz; Tc = 1/64 seg

Umbral = 0.5534; Pérdida = 0.0108; Falarm =0.0119



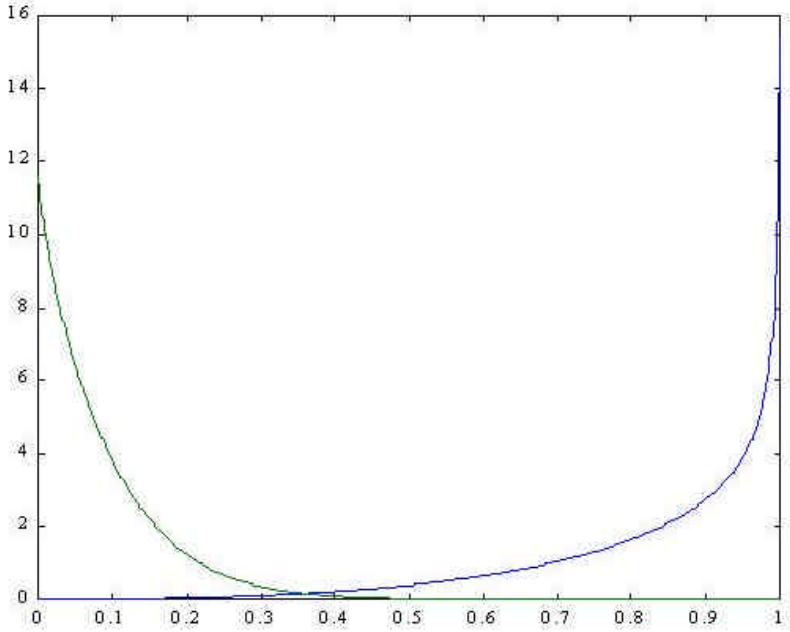
Fs = 11025 Hz; Tc = 1/32 seg

Umbral = 0.4729; Pérdida = 0.0061; Falarm =0.0059



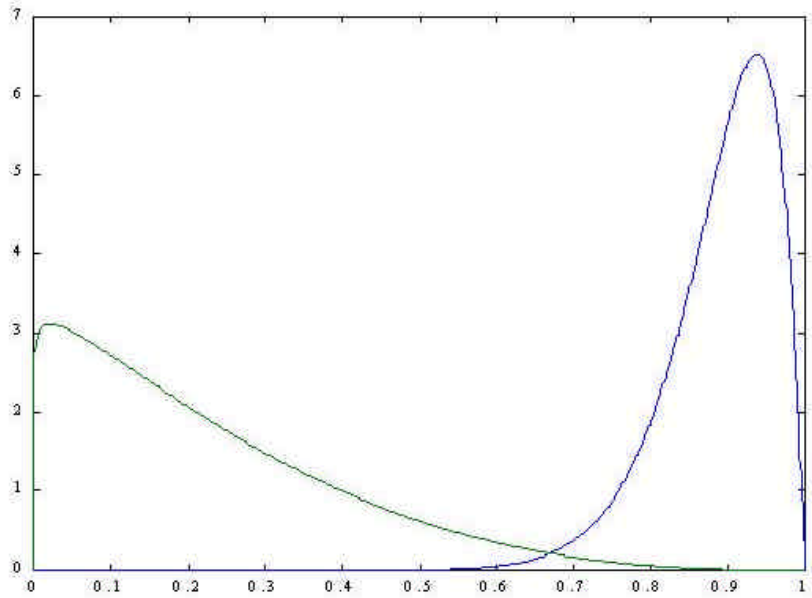
Fs = 11025 Hz; Tc = 1/16 seg

Umbral = 0.3579; Pérdida = 0.0156; Falarm =0.0097



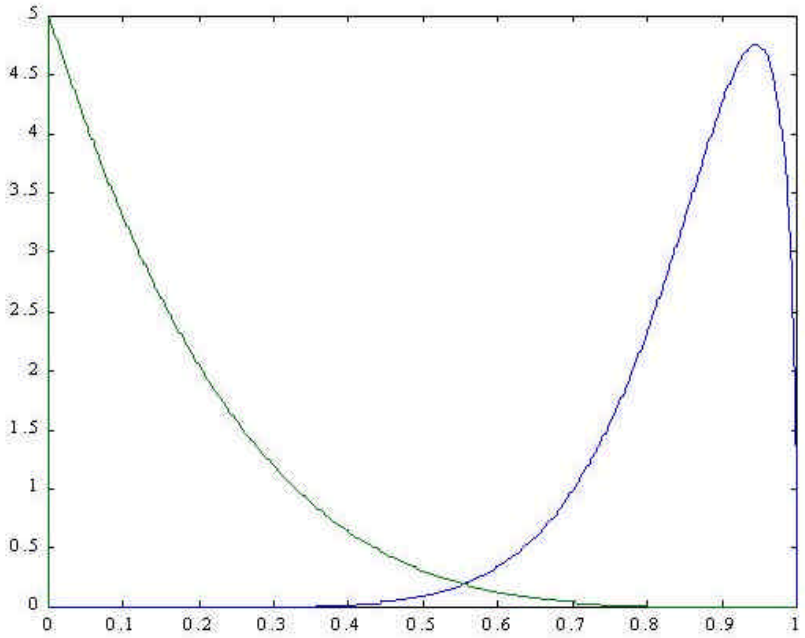
Fs = 8000 Hz; Tc = 1/128 seg

Umbral = 0.6703; Pérdida = 0.0095; Falarm =0.0186



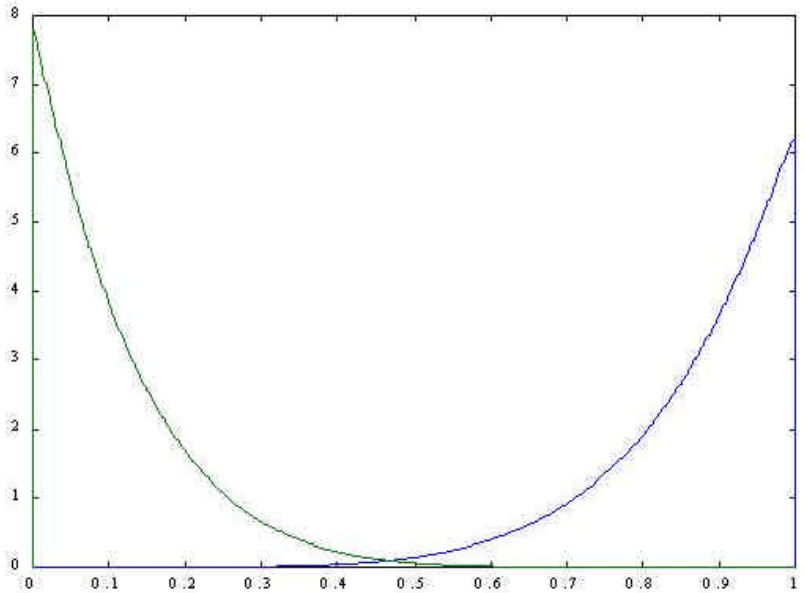
Fs = 8000 Hz; Tc = 1/64 seg

Umbral = 0.5548; Pérdida = 0.0127; Falarm =0.0170



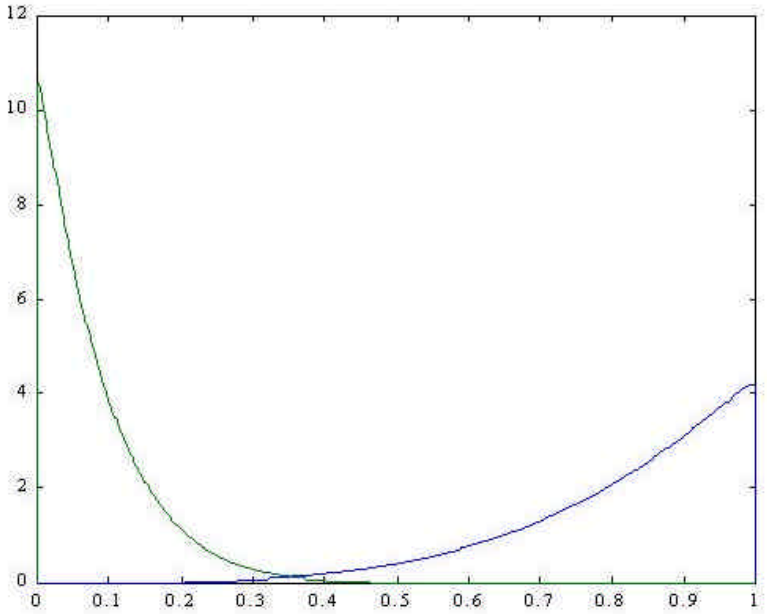
Fs = 8000 Hz; Tc = 1/32 seg

Umbral = 0.4670; Pérdida = 0.0069; Falarm =0.0064



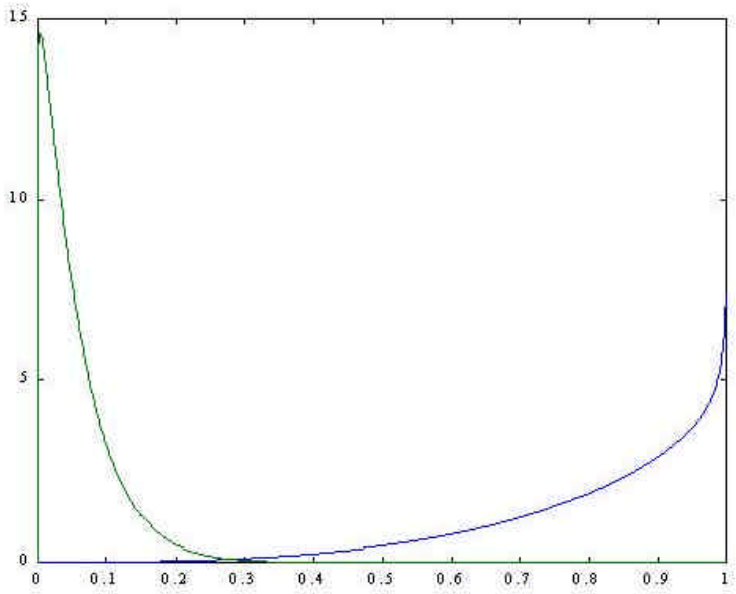
Fs = 8000 Hz; Tc = 1/16 seg

Umbral = 0.3544; Pérdida = 0.0095; Falarm =0.0068



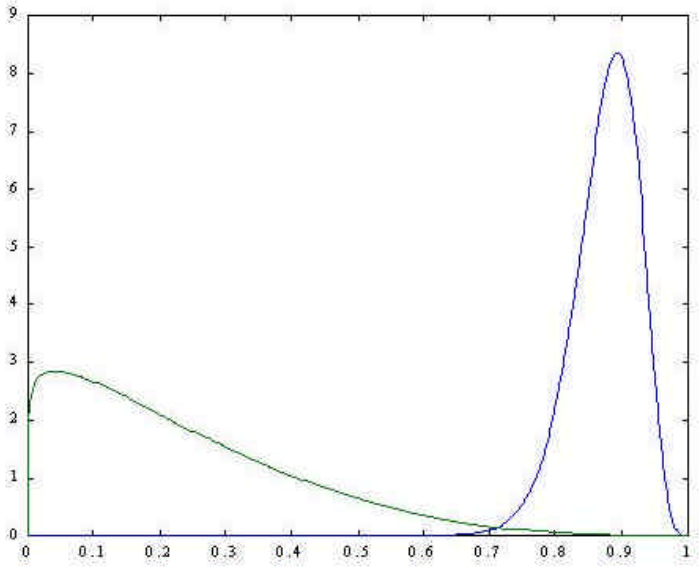
Fs = 8000 Hz; Tc = 1/8 seg

Umbral = 0.2809; Pérdida = 0.0059; Falarm =0.0033



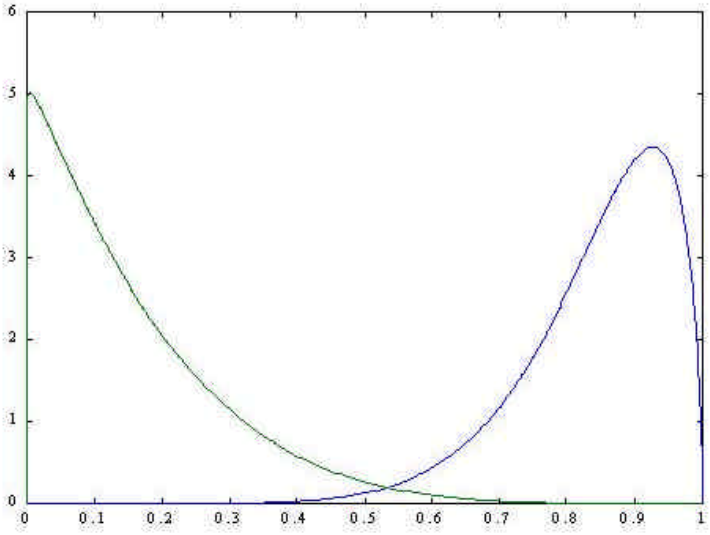
Fs = 4000 Hz; Tc = 1/128 seg

Umbral = 0.7121; Pérdida = 0.0039; Falarm =0.0119



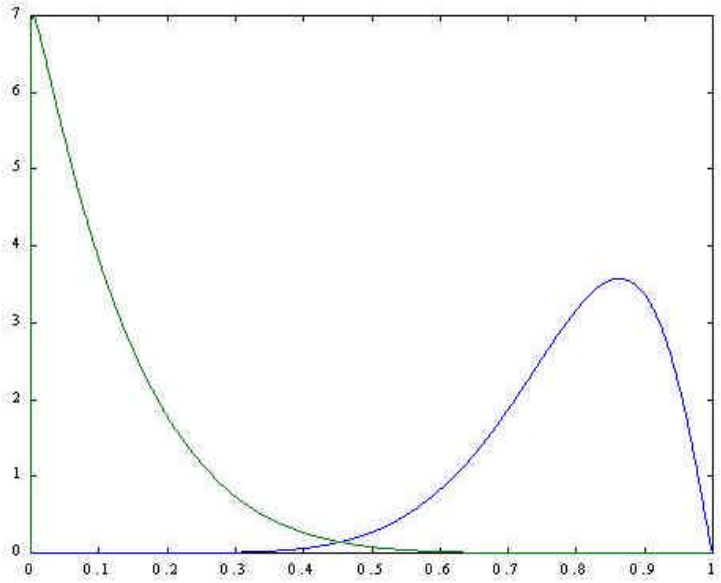
Fs = 4000 Hz; Tc = 1/64 seg

Umbral = 0.5339; Pérdida = 0.0124; Falarm =0.0163



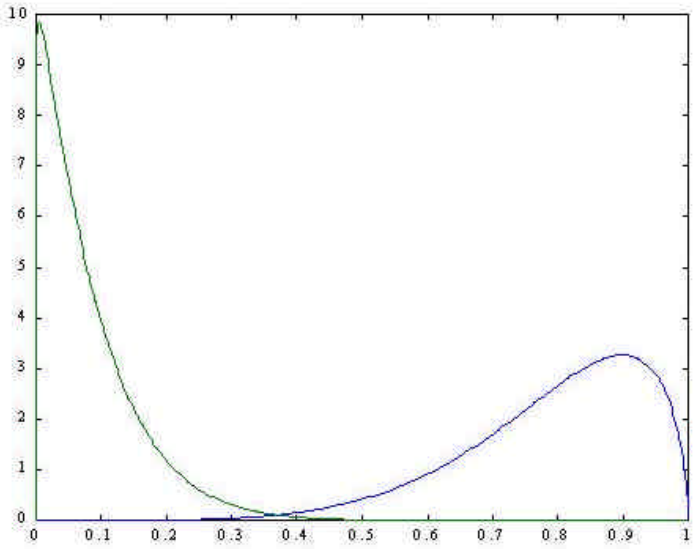
Fs = 4000 Hz; Tc = 1/32 seg

Umbral = 0.4538; Pérdida = 0.0084; Falarm =0.0103



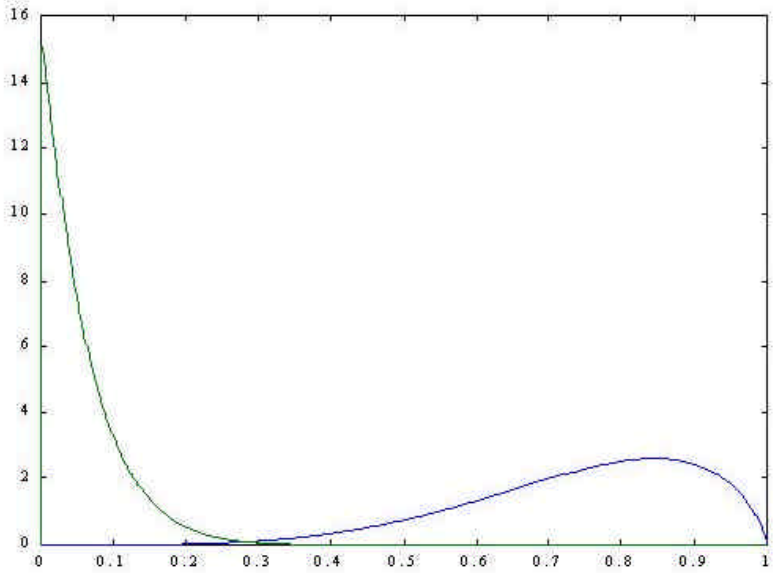
Fs = 4000 Hz; Tc = 1/16 seg

Umbral = 0.3702; Pérdida = 0.0061; Falarm =0.0053



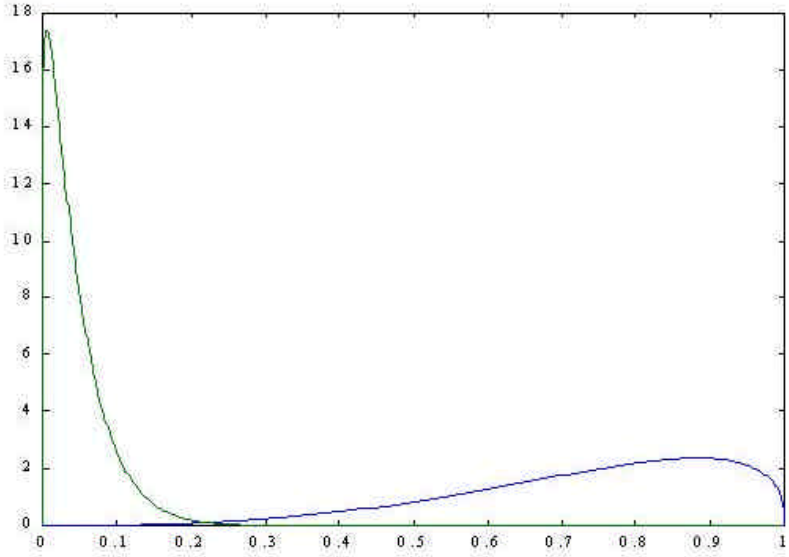
Fs = 4000 Hz; Tc = 1/8 seg

Umbral = 0.2839; Pérdida = 0.0056; Falarm =0.0042



Fs = 4000 Hz; Tc = 1/4 seg

Umbral = 0.2218; Pérdida = 0.0061; Falarm =0.0034



En la cabecera de cada gráfica se han incluido los datos correspondientes a cada una de ellas, incluyendo los valores de la pareja frecuencia de muestreo-tiempo de captura, así como el umbral de detección y las probabilidades de pérdida y falsa alarma. Se han resaltado en color rojo los mejores valores para cada valor de frecuencia de muestreo, de forma que se pueda hacer un rápido análisis con sólo echar un vistazo. En el siguiente apartado comentaremos los resultados obtenidos.

4.4.4) Resultados y conclusiones.

Como se acaba de decir, los mejores valores para cada frecuencia entre las pruebas realizadas se han resaltado en rojo, estos son:

Frecuencia	Tcaptura	Umbral	Ppérdida	Pfalarma
4 KHz	**¼ seg**	**0.2218**	**0.0061**	**0.0034**
4 KHz	1/8 seg	0.2839	0.0056	0.0034
8 KHz	1/8 seg	0.2809	0.0059	0.0033
11,025 KHz	1/32 seg	0.4729	0.0061	0.0059
22,050 KHz	1/32 seg	0.5572	0.0047	0.0050
44,1 KHz	1/64 seg	0.6126	0.0125	0.0142

De estos valores podemos descartar la mayoría, pues no debemos olvidar que intentamos trabajar en tiempo real. Por lo tanto descartamos todas las frecuencias, excepto 4 y 8 KHz, debido a que para una frecuencia de 11025 Hz o superior el número de muestras de cada anuncio es muy alto y el tiempo de procesado aumenta considerablemente, lo cual se ha podido comprobar durante la realización de las pruebas.

Nos quedamos por lo tanto con tres opciones, de éstas también se descarta la correspondiente a 4 KHz y tiempo de captura igual a un octavo de segundo, pues

si hacemos los cálculos el número de muestras a comparar es de 500 muestras, mientras que para las otras dos opciones es de 1000 muestras, resultando este último número de muestras mucho más adecuado para la calidad de la correlación.

De esta manera sólo nos quedan dos opciones entre las que elegir, en ambas el número de muestras a comparar es de 1000 muestras y los valores que se obtienen para las probabilidades son muy parecidos. Sin embargo trabajando con una frecuencia de muestreo de 4 KHz el tiempo empleado en la comparación es menor, dado que el anuncio que tenemos almacenado tiene la mitad de muestras totales que con 8 KHz. A priori este motivo no debería ser decisivo en nuestra elección, pues con ambas frecuencias se puede realizar más de una comparación en el tiempo que dura cualquier anuncio publicitario que se emita hoy en día, por lo tanto con ambas se puede trabajar en tiempo real.

Sin embargo, como se puede adivinar observando los valores obtenidos estamos muy lejos del objetivo marcado para la probabilidad de pérdida y falsa alarma, de forma que haciendo una comparación de un trozo de cada anuncio que se emita con el anuncio que tenemos almacenado, el número de errores que se obtiene es muy alto, o sea, nuestro algoritmo no funciona. Este hecho hace que tengamos que buscar alguna solución, y la que se ha adoptado es la de hacer más de una comparación por cada anuncio que se emita y plantear un algoritmo de decisión adecuado para que la probabilidad de error sea mínima. De esta forma, la probabilidad de error equivale al producto de las probabilidades de error de todas las comparaciones que se hagan por cada emisión de un anuncio, la probabilidad conjunta equivale al producto de las probabilidades marginales cuando los sucesos son independientes, y en este caso lo son.

Así, si se muestrea a 4 KHz se pueden realizar más comparaciones en el tiempo que dura un anuncio que si muestreamos a 8 KHz, por lo tanto la frecuencia escogida para implementar nuestra aplicación es 4 KHz, y el tiempo de captura es 0.25 segundos, lo cual equivale a decir que Lh = 1000 muestras. Con esta frecuencia de muestreo se estará introduciendo aliasing en el dominio de la frecuencia, pero ya hemos visto antes que no nos afecta para nuestro fin.

Tan sólo nos queda ya por definir el algoritmo de detección y el umbral óptimo, y esto es lo siguiente que se va a ver.

4.5) Condiciones del decisor y umbral de detección.

A continuación se va a analizar la rutina principal de este proyecto, estudiando los procesos a llevar a cabo para la detección adecuada de anuncios publicitarios. También se tratará con detenimiento el método utilizado para capturar los datos con los que se ha de trabajar desde Matlab en tiempo real.

4.5.1) Introducción.

Como ya hemos dicho nuestro objetivo es detectar anuncios en FM ó AM. Trabajaremos con una tasa de muestreo de 4 KHz y el trozo de señal que iremos capturando de la emisora de radio será de 1/4 seg (Lh=1000). Trataremos de detectar un anuncio solamente en una emisora en tiempo real.

Para ello se ha implementado un programa principal, cuyo código se verá posteriormente, desde el cual se hacen llamadas a cada una de las subrutinas que hemos visto, especialmente a la de comparación de dos tramas de audio, ya que, como hemos adelantado, se tratará de hacer todas las comparaciones posibles en el tiempo de duración del anuncio.

El proceso que lleva a cabo este programa o rutina principal es el siguiente:

1) Inicialización de todos las constantes y variables que son necesarias. Las constantes *Ex* (energía de las muestras del anuncio a detectar, el cual tenemos grabado) y *M* (tamaño de las secciones en que se divide el anuncio para implementar el algoritmo *Overlapp & Add*) se han calculado previamente y se pasan como parámetros de entrada.

2) Se captura un trozo de señal de una emisora de radio, la que se tenga sintonizada para la detección del anuncio. Este trozo de señal tendrá 1000 muestras, ya que la captura se realiza a una frecuencia de 4000 muestras/seg. durante 0.25 segundos.

3) Se compara la señal capturada con el anuncio que tenemos grabado a través el proceso de comparación que ya hemos visto, la correlación cruzada.

4) Si la comparación es positiva, o sea, hay un fragmento del anuncio que tenemos grabado que coincide con la señal capturada, entonces se entra en el bucle de detección positiva. Si la comparación es negativa, se vuelve al punto 2).

5) Si se entra en el bucle de detección positiva se hace práctimente lo mismo, se seguirá capturando señal de la emisora de radio y se comparará con el anuncio, así hasta que el anuncio finalice. Una vez en este bucle se evaluarán ciertos parámetros de cada una de las comparaciones que se van sucediendo y en función de dichos parámetros se decidirá si realmente es el anuncio a detectar o, por el contrario, es una falsa alarma. En éste caso se abandonará el bucle de detección positiva inmediatamente.

6) Una vez que acaba el anuncio, si resultó ser el anuncio que buscamos se da un mensaje de detección positiva y se vuelve al punto 2).

A continuación vemos cada uno de estos puntos con más detalle.

4.5.2) Interfaz Sound Blaster-Matlab.

Uno de los principales problemas con que nos hemos encontrado para llevar a cabo este proyecto ha sido con el de la adquisición de datos desde la tarjeta de TV/FM. Si se intentara capturar datos directamente desde dicha tarjeta al Matlab el

proceso sería realmente complejo y difícil de solucionar, pero gracias a que la tarjeta de audio Sound Blaster funciona como interfaz se pueden capturar datos sin mayor dificultad, aunque con algunas limitaciones. La tarjeta Soundblaster funciona con varios formatos de audio ya conocidos, muchos de los cuales hemos visto con anterioridad. Uno de ellos es el formato "*wav*", por lo tanto no hay que hacer ninguna conversión de datos para trabajar en Matlab.

Antes se ha utilizado la función "*waveload.m*" para poder trabajar en Matlab con anuncios ya grabados y almacenados en memoria. Vimos que esta función sustituye a la función "*wavread.m*" de Matlab, la cual no funcionaba por una pequeña variación en la cabecera de las señales "*.wav*" que utilizamos.

Sin embargo ahora no se trata de pasar señales de audio ya grabadas al entorno de trabajo de Matlab, sino de obtener las señales en dicho entorno de trabajo en tiempo real, sin cabeceras, sino sólo datos. O sea, se trata de adquirir o capturar datos de una emisora de radio o televisión en tiempo real.

El Matlab tiene una librería que está dedicada precisamente a la captura de datos en tiempo real desde múltiples dispositivos, se llama "*Data Acquisition Toolbox*", DAQ, sin embargo dicha librería es costosa y no disponemos de ella.

No obstante hemos encontrado una versión limitada de una de las funciones de esta librería que es ideal para nuestro propósito, ésta es la función "*recordsnd.m*", la cual se describe a continuación:

> *RECORDSOUND graba sonido (audio) directamente en una matriz de Matlab.*
>
> *Y=RECORDSOUND(SEGUNDOS) graba una señal audio mono, durante un tiempo igual al parámetro de entrada SEGUNDOS y a una tasa de muestreo de 8192 Hz y con muestras de 8 bits, en el vector Y, con rango -1 <= Y <= 1.*

Y=RECORDSOUND(SEGUNDOS,FS) hace exactamente lo mismo pero a la frecuencia de muestreo indicada en FS.

Y=RECORDSOUND(SECONDS,FS,NUMCANALES), donde el parámetro de entrada NUMCANALES indica si la señal audio a grabar es mono (=1) o estéreo (=2).

Y=RECORDSOUND(SECONDS,FS,NUMCANALES, BITS), donde el parámetro de entrada BITS sirve para indicar el número de bits por muestra en la grabación, 8 ó 16.

Como se puede observar esta función es válida y adecuada para nuestros propósitos, pero presenta una limitación por ser una versión de libre distribución. Esta limitación consiste en que sólo se puede grabar durante un tiempo menor o igual a 1 segundo cada vez que se utiliza. Nosotros trabajamos grabando la señal audio de la emisora de radio o televisión durante 1/4 de segundo, por lo tanto no tendremos ningún problema a priori y hemos resuelto la captura de datos en tiempo real de una forma sencilla y rápida.

No obstante no sería válida si se dispusiera de los medios suficientes para trabajar en tiempo real con una base de datos que contenga una gran cantidad de anuncios, y un algoritmo de conmutación que fuera seleccionando las distintas emisoras para monitorizarlas todas simultáneamente. Por ello nuestra aplicación podrá detectar un anuncio conocido, cualquiera de los que se emiten tanto por radio como por televisión, en una emisora o cadena, la que esté sintonizada en el momento de la detección.

4.5.3) Método de decisión.

Tan sólo nos queda por analizar una cuestión, que es el método y los parámetros a utilizar para decidir si la detección es positiva o no. El estudio del método de decisión es de muy importante, y de él dependerá en gran parte el

funcionamiento y éxito de la rutina principal. Por lo tanto a continuación se va a examinar la implementación del decisor y de todos sus parámetros. Se estudiará con especial cuidado el más importante de estos parámetros, que es el umbral de detección, y las condiciones impuestas para considerar una detección positiva o negativa.

- **Parámetros del decisor.**

Ya hemos visto el proceso que sigue la rutina principal para comprobar si el anuncio que se está emitiendo es el que estamos buscando. Durante dicho proceso nos ayudaremos de los siguientes parámetros:

- ***Umbral*:** se trata del umbral de detección, en cada comparación que se haga del fragmento de señal que se captura con el anuncio grabado se comparará el máximo coeficiente de correlación obtenido con el umbral de detección. Si el coeficiente de correlación es mayor o igual al umbral la detección se considerará positiva. Estudiaremos su valor , que es constante, con posterioridad.

- ***Detec:*** esta variable se inicializa como un vector de 15 ceros. Cuando hay una comparación positiva se pone el primer elemento del vector a 1, y se entra en el bucle de detección. Se vuelve a hacer otra comparación con otro fragmento de señal capturado unos segundos después, y si de nuevo la comparación es positiva se pondrá el segundo elemento del vector a 1, en cambio si es negativa se pondrá a cero. Este proceso se repetirá hasta que se salga del bucle de detección, si salimos del bucle por que se ha incumplido alguna de las condiciones, o sea, porque no se trata del anuncio que buscamos, el vector ***Detec*** se pone a cero y se comienza de nuevo. Sin embargo si se sale del bucle porque ha transcurrido el tiempo que dura el anuncio este vector será el que nos informe del número de comparaciones positivas y negativas que se han realizado durante la emisión de un anuncio.

- ***Tdetec:*** esta variable tiene un funcionamiento parecido al de ***Detec***, ya que ambas se inicializan y actualizan de la misma manera y en los mismos instantes. En ***Tdetec*** se irán almacenando los tiempos relativos a la duración total del anuncio en los que se realiza cada comparación positiva. Así, la primera comparación positiva que se haga pondrá el primer elemento de ***Tdetec*** a 0, mientras que el segundo elemento reflejará la ubicación del fragmento de señal capturado dentro del anuncio en unidades temporales. Una vez dentro del bucle de detección los elementos de este vector se irán actualizando tras cada comparación.

  Tanto ***Detec*** como ***Tdetec*** son vectores de 15 elementos, se ha tomado así porque es probable que se vayan a hacer más de 15 comparaciones en el caso de una detección positiva (muchas menos si no se trata del anuncio que buscamos). Pero si el anuncio a buscar es de muy larga duración habrá que aumentar el número de elementos de ambos vectores.

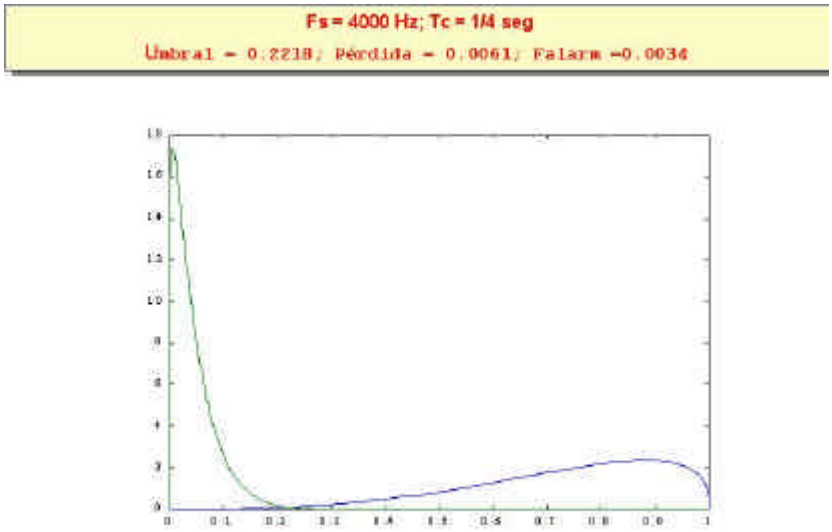
- ***Tdur:*** es el tiempo que dura el anuncio que tenemos grabado. Es muy fácil de calcular, ya que el anuncio es un parámetro de entrada y sabemos calcular su número de muestras (Longitud = length(anuncio)); entonces:

  tdur **=** Longitud (muestras) / 4000 (muestras/seg)

  Esta constante, junto con ***Tdetec*** será la que nos diga cuándo se ha de salir del bucle de detección.

- **Umbral de detección:**

Éste es el principal de los parámetros, ya que se considerará cada una de las comparaciones positiva o negativa en función de su valor. Para su estudio se hace necesario volver al análisis estadístico, recordemos la gráfica y valores obtenidos para una frecuencia de 4000 Hz y un tiempo de captura de 0.25 segundos:



Asímismo, se muestra a continuación una tabla con los valores teóricos obtenidos para distintos umbrales de decisión:

Umbral	Ppérdida	Pfalarma
0.10	3.1569e-004	0.1001
0.15	0.0014	0.0267
0.2	0.0042	0.0065
0.2218	**0.0061**	**0.0034**
0.25	0.0095	0.0014
0.30	0.0187	2.8210e-004
0.35	0.0327	4.9154e-005
0.40	0.0531	7.4206e-006
0.45	0.0810	9.4744e-007
0.50	0.1177	9.9148e-008
0.55	0.1645	8.1589e-009
0.60	0.2222	4.9904e-010

En la tabla se puede observar que hay un valor central (umbral = 0.2218) para el cual las probabilidades de pérdida y falsa alarma son similares. Sin embargo las probabilidades obtenidas son muy altas para los objetivos marcados ($1x10^{-6}$).

Además hemos de tener en cuenta que realizaremos varias comparaciones durante la emisión del anuncio, y que jugaremos con los resultados de cada una de ellas imponiendo ciertas condiciones que nos dirán si la detección es positiva o no, con lo cual estas probabilidades sólo son orientativas.

A continuación mostramos los valores obtenidos tras realizar una prueba con un umbral de decisión de 0.5, en la primera gráfica tenemos el máximo coeficiente de correlación obtenido en cada una de las comparaciones negativas, aparece en negrita el mayor de todos ellos.

Columns 1 through 7

0.2259 0.3635 0.3411 0.3439 0.2632 0.3006 0.3596

Columns 8 through 14

0.4626 0.4037 0.3257 0.3577 0.3143 0.3705 0.3203

Columns 15 through 21

0.3707 0.4338 0.3781 0.4768 0.4492 0.4360 0.3557

Columns 22 through 28

0.4447 0.3596 0.2682 0.4860 0.4356 0.4514 0.4881

Columns 29 through 35

0.3881 0.3325 0.4005 **0.4899** 0.2303 0.3121 0.1839

Columns 36 through 42

0.3068 0.4245 0.3176 0.3101 0.3022 0.2290 0.3460

En esta otra gráfica podemos observar el máximo coeficiente de correlación obtenido en cada una de las comparaciones positivas. También se han incluído los valores de correlación que han provocado una falsa alarma en alguna comparación: aparece en azul el valor del coeficiente de correlación correspondiente a la comparación que provoca la entrada en el bucle de detección, y en rojo el valor del coeficiente que provoca la salida de dicho bucle por no tratarse del anuncio que buscamos. Asímismo se ha marcado en negrita al menor

de los coeficientes de correlación de todas las detecciones positivas, que como se puede comprobar son dos.

```
Columns 1 through 7

  0.5006   0.2447   0.9345   0.9290   0.9330   0.9146   0.9637

Columns 8 through 14

  0.9510   0.9718   0.5514   0.4047   0.5311   0.4203   0.9741

Columns 15 through 20

  0.9719   0.9829   0.9794   0.9874   0.9608   0.9482
```

Aquí se puede observar que hay una correlación que provoca que se entre en el bucle de detección (0.5006), pero la siguiente comparación confirma que es una falsa alarma (0.2447). Seguidamente se observan los valores de una detección positiva (0.9345 0.9290 0.9330 **0.9146** 0.9637 0.9510 0.9718), dos nuevas comparaciones que dan falsa alarma (0.5514 0.4047 0.5311 0.4203), y de nuevo otra detección positiva (0.9741 0.9719 0.9829 0.9794 0.9874 0.9608 0.9482).

Recordamos que al hablar de falsa alarma, hay que distinguir entre falsa alarma en una comparación y falsa alarma en una detección, una detección va a estar compuesta por varias comparaciones, por lo tanto una falsa alarma en una comparación no significa que se haya cometido un error. De hecho en las falsas alarmas que comentamos en estas gráficas se entra en el bucle de detección pero se sale inmediatamente, de forma que no se reconoce el anuncio que se está emitiendo.

En el algoritmo que hemos planteado hemos actuado de la siguiente manera: tras hacer estas pruebas se detectó que con un umbral demasiado bajo (como es el valor 0.2218) teníamos demasiados errores por falsa alarma, además se puede comprobar que para las comparaciones positivas el máximo coeficiente de correlación es próximo a la unidad. Por lo tanto parece mucho más adecuado aumentar el umbral de decisión, de ahí que el umbral de decisión escogido sea de

0.5. Teóricamente este umbral nos da un valor muy bueno para la probabilidad de falsa alarma (9.9148e-008), pero se obtiene un valor poco aceptable para la probabilidad de pérdida (0.1177). Sin embargo tras observar los coeficientes obtenidos cuando hay una detección positiva, el menor de ellos tiene un valor de 0.9146, vemos que en la práctica se tiene una probabilidad de pérdida muy pequeña.

Si además tenemos en cuenta que vamos a trabajar con varias comparaciones por cada emisión de un anuncio y que tenemos libertad para imponer las condiciones de detección positiva, gracias a los parámetros que se obtienen, nos damos cuenta que es necesario tomar un umbral de decisión cercano al valor promedio del dominio en el que trabajamos.

- **Condiciones para la detección:**

Por lo general en los algoritmos de decisión nos encontraremos con una sóla condición que vendrá impuesta por el valor del umbral de decisión: si el valor obtenido es mayor al del umbral de decisión se ejecutarán una serie de secuencias y si es menor se ejecutarán otras sentencias distintas.

Sin embargo aquí tenemos un caso especial, ya que no sólo hacemos una comparación, sino que hacemos todas las posibles mientras dura el anuncio con el propósito de minimizar la probabilidad de error. Por lo tanto hemos de trabajar con los resultados obtenidos de hacer cada una de estas comparaciones. Ello nos da la posibilidad de imponer numerosas condiciones en función de las cuales consideraremos la detección positiva o negativa.

Si estas condiciones que se imponen son demasiado restrictivas, por ejemplo, que todas las comparaciones realizadas durante la emisión de un anuncio sean positivas, entonces tendremos una probabilidad de falsa alarma despreciable y la probabilidad de pérdida será relativamente alta. Si las condiciones son muy relajadas ocurrirá el efecto contrario, difícilmente tendremos una pérdida pero habrá multitud de falsas alarmas.

Al poder jugar con tantos parámetros se pueden imponer muchísimas condiciones y de hecho habrá muchas combinaciones de ellas que sean adecuadas. Tras hacer numerosas pruebas y obtener buenos resultados éstas son las que hemos impuesto:

1) En cuanto haya una comparación positiva se entrará en el bucle de detección, si una vez dentro de este bucle se realizan dos comparaciones negativas se saldrá automáticamente del bucle, considerando que no es el anuncio buscado.

2) Una vez dentro del bucle de detección el valor de los elementos del vector ***Tdetec*** debe ser creciente, si no es así se saldrá automáticamente del bucle de detección.

3) Una vez se sale del bucle de detección se comprueba cuántas comparaciones positivas ha habido, para considerar que la detección es positiva deben realizarse al menos el sesenta por ciento de las comparaciones positivas.

Conjugando estas tres condiciones se obtiene un detector muy robusto, de forma que el número de errores, ya sea por falsa alarma o por pérdida, es prácticamente despreciable.

4.5.4) Pruebas realizadas. Eficacia del decisor.

Para realizar estas pruebas se ha tomado una emisión de FM, durante la cual se repite un mismo anuncio dos veces. Por otra parte hemos conseguido grabar este anuncio de otra emisora de radio diferente. Tras ejecutar la rutina de detección en múltiples ocasiones no se ha observado ninguna falsa alarma ni ninguna pérdida.

A continuación mostramos el mensaje de detección que aparece en pantalla cada vez que se reconoce o detecta el anuncio:

```
ACTIVE LA EMISORA Y PULSE CUALQUIER TECLA PARA
CONTINUAR
EL ANUNCIO HA SIDO EMITIDO A LAS
20:35
DEL
16-Sep-2001
¿DESEA SEGUIR CON LA DETECCIÓN? NO=0, SI=1

EL ANUNCIO HA SIDO EMITIDO A LAS
20:38
DEL
16-Sep-2001
¿DESEA SEGUIR CON LA DETECCIÓN? NO=0, SI=1
```

La rutina principal devuelve también una tabla, llamada *"Tabla_detecciones"*, en la que se indican todas las emisiones detectadas:

```
» tabla_detecciones

tabla_detecciones =

    '20:35'    '16-Sep-2001'
    '20:38'    '16-Sep-2001'
```

Además de realizar este experimento se han realizado muchos otros, con emisiones tanto de radio como de televisión, y anuncios de todo tipo y duración. En todos ellos el resultado ha sido satisfactorio. Además este resultado puede incluso ser más fiable si se graba el anuncio de la misma emisora en la que lo vamos a buscar.

5) CONCLUSIONES Y LINEAS FUTURAS DE TRABAJO:

Para finalizar vamos a hacer un análisis del detector, así como de las posibles mejoras que se podrían introducir para convertirlo en una herramienta más potente y versátil.

Como ya hemos visto con el detector de anuncios publicitarios que se ha implementado se puede monitorizar un anuncio en concreto y en una única emisora de radio o televisión. Tras las pruebas realizadas hemos comprobado como esta aplicación funciona correctamente con los mínimos recursos: un PC, tarjeta Sound Blaster, tarjeta de radio y televisión y el entorno de trabajo de Matlab. Sin embargo se puede incrementar la capacidad de detección de la aplicación si se dispone de más recursos. A continuación se muestran algunas de las ideas o posibilidades de lo que pueden ser las líneas futuras de trabajo, aunque será necesario un análisis mucho más exhaustivo para estudiar la posibilidad de llevar a cabo cada una de ellas:

- Reconocimiento de anuncios publicitarios utilizando una base de datos en la cual se almacenen dichos anuncios, de esta forma no se trabajaría tan sólo con un anunciante, sino que se podrían monitorizar varios anuncios simultáneamente. El número de estos anuncios dependerá en gran parte de la capacidad de procesado de que se disponga.

- Realización de detecciones monitorizando varias emisoras simultáneamente. Para ello sería necesario implementar un algoritmo de conmutación entre emisoras o canales de radio y televisión, de forma que se haga un barrido de las emisoras que se pretendan controlar. Este barrido se podría realizar programando la tarjeta de radio y televisión, de forma que se capture secuencialmente un trozo de señal en cada una de las emisoras. A la hora de procesar las señales capturadas haría falta una mayor capacidad de procesado, así como una sincronización entre el algoritmo detector y el algoritmo conmutador. También se haría necesario un identificador para cada emisora.

- Monitorización de anuncios publicitarios con varios anunciantes y varias emisoras de radio o televisión. Esta sería la aplicación más completa, ya que

abarca a las dos anteriores, no obstante es de una complejidad muy alta y se necesitarían recursos tal vez muy costosos.

- Reconocimiento de anuncios publicitarios en TV a través de la comparación de tramas de audio y de video. Se tendrían dos procesos en paralelo, el primero de ellos (reconocimiento de tramas de audio) sería el que ya hemos visto; el segundo (reconocimiento de tramas de vídeo) supondría la captura de imágenes más un procesado adicional de éstas. Es sabido que este procesado adicional sería mucho más complejo, ya que habría que trabajar con matrices de datos (el audio se puede almacenar en vectores unidimensionales, pero no así las imágenes).

Es evidente que para implementar alguna de estas aplicaciones es necesario disponer tanto de recursos tecnológicos como humanos, dado que probablemente no sea suficiente un PC, sino que habría que programar una placa DSP de alta capacidad, la cual proporcionaría una velocidad de procesado muy superior. También sería conveniente disponer del *Data Acquisition Toolbox* de Matlab, el cual nos permite trabajar con varios canales en cada uno de los cuales se podría monitorizar una emisora.

Juntos, MATLAB y el *Data Acquisition Toolbox* ofrecen un entorno integrado único para soportar todo el proceso de adquisición y análisis de datos. Se pueden analizar y visualizar fácilmente datos al vuelo, guardarlos para un procesado posterior y hacer actualizaciones iterativas a su *setup* de ensayo basándose en los resultados de su análisis. Las principales características de esta librería son:

- Interfaz con dispositivos de adquisición de datos de estándar en la industria.

- Acceso directamente desde MATLAB a datos medidos en directo.

- Entorno integrado único para adquisición, análisis y visualización de datos.
- Entrada analógica, salida analógica y E/S digital.

- Interfaz directa para características de circuito soportadas, tales como:

  - Adquisiciones de canal único y múltiple.

  - E/S "single-point" y "suffered".

- Triggers de hardware y software.

- Kit adaptador para crear interfaces personalizadas para hardware no soportado.

APÉNDICE A: ESTRUCTURA DEL FORMATO WAVE.

APENDICE A: ESTRUCTURA DEL FORMATO WAVE

Waveform Audio File Format (WAVE)

This section describes the Waveform format, which is used to represent digitized sound.

The WAVE form is defined as follows. Programs must expect(and ignore) any unknown chunks encountered, as with all RIFF forms. However, *<fmt-ck>* must always occur before *<wave-data>*, and both of these chunks are mandatory in a WAVE file.<

```
WAVE-form> ->
      RIFF( 'WAVE'
      <fmt-ck>                // Format
      [<fact-ck>]             // Fact chunk
      [<cue-ck>]              // Cue points
      [<playlist-ck>]         // Playlist
      [<assoc-data-list>]     // Associated data list
      <wave-data>   )         // Wave data
```

WAVE chunks are described in the following sections.

WAVE Format Chunk

The WAVE format chunk ***<fmt-ck>*** specifies the format of the *<wave-data>*. The *<fmt-ck>* is defined as follows:

```
<fmt-ck> ->   fmt( <common-fields> <format-specific-fields> )

<common-fields> ->
      struct
      {
            WORD wFormatTag;          // Format category
            WORD wChannels;           // Number of channels
            DWORDdwSamplesPerSec;     // Sampling rate
            DWORDdwAvgBytesPerSec;    // For buffer estimation
            WORD wBlockAlign;         // Data block size
      }
```

Common Fields Chunk

The fields in the ***<common-fields>*** chunk are as follows:

Field	Description
wFormatTag	A number indicating the WAVE format category of the file. The content of the *<format-specific-fields>* portion of the `fmt' chunk, and the interpretation of the waveform data,on this value. must register any new WAVE format categories. See ``Registering Multimedia Formats'' in Chapter 1, ``Overview of Multimedia,'' for information on registering WAVE format categories. ``Wave Format Categories,'' following this section, lists the currently defined WAVE format categories.
wChannels	The number of channels represented in the waveform data, such as 1 for mono or 2 for stereo.
dwSamplesPerSec	The sampling rate (in samples per second)

at which each channel should be played.

dwAvgBytesPerSec The average number of bytes per second at which the waveform data should be transferred. Playback software can estimate the buffer size using this value.

wBlockAlign The block alignment (in bytes) of the waveform data. Playback software needs to process a multiple of wBlockAlign bytes of data at a time, so the value of wBlockAlign can be used for buffer alignment.

Format Specific Fields Chunk

The ***<format-specific-fields>*** consists of zero or more bytes of parameters. Which parameters occur depends on the WAVE format category-see the following section for details. Playback software should be written to allow for (and ignore) any unknown *<format-specific-fields>* parameters that occur at the end of this field.

WAVE Format Categories

The format category of a WAVE file is specified by the value of the wFormatTag field of the `**fmt'** chunk. The representation of data in <wave-data>, and the content of the <format-specific-fields> of the `fmt' chunk, depend on the format category.

The currently defined open non-proprietary WAVE format categories are as follows:

wFormatTag Value		Format Category
WAVE_FORMAT_PCM	(0x0001)	Microsoft Pulse Code Modulation (PCM)

The following are the registered proprietary WAVE format categories:

wFormatTag Value		Format Category
FORMAT_MULAW	(0x0101)	IBM mu-law format
IBM_FORMAT_ALAW	(0x0102)	IBM a-law format
IBM_FORMAT_ADPCM	(0x0103)	IBM AVC Adaptive Differential PCM format

Microsoft WAVE_FORMAT_PCM format

The following sections describe the Microsoft WAVE_FORMAT_PCM format. If the *wFormatTag* field of the *<fmt-ck>* is set to WAVE_FORMAT_PCM, then the waveform data consists of samples represented in pulse code modulation (PCM) format. For PCM waveform data, the *<format-specific-fields>* is defined as follows:

```
<PCM-format-specific> ->
      struct
      {
            WORD wBitsPerSample;    // Sample size
      }
```

The ***wBitsPerSample*** field specifies the number of bits of data used to represent each sample of each channel. If there are multiple channels, the sample size is the same for each channel.

For PCM data, the ***wAvgBytesPerSec*** field of the `fmt' chunk should be equal to the following formula rounded up to the next whole number:

$$\text{wChannels} \times \text{wBitsPerSecond} \times \frac{\text{wBitsPerSample}}{8}$$

The ***wBlockAlign*** field should be equal to the following formula, rounded to the next whole number:

$$\text{wChannels} \times \frac{\text{wBitsPerSample}}{8}$$

Data Packing for PCM WAVE Files

In a single-channel WAVE file, samples are stored consecutively. For stereo WAVE files, channel 0 represents the left channel, and channel 1 represents the right channel. The speaker position mapping for more than two channels is currently undefined. In multiple-channel WAVE files, samples are interleaved.

The following diagrams show the data packing for a **8-bit** mono and stereo WAVE files:

Data Packing for 8-Bit Mono PCM:

Sample 1	Sample 2	Sample 3	Sample 4
Channel 0	Channel 0	Channel 0	Channel 0

Data Packing for 8-Bit Stereo PCM:

Sample 1		Sample 2	
Channel 0 (left)	Channel 1 (right)	Channel 0 (left)	Channel 0 (right)

The following diagrams show the data packing for **16-bit** mono and stereo WAVE files:

Data Packing for 16-Bit Mono PCM:

Sample 1		Sample 2	
Channel 0 low-order byte	Channel 0 high-order byte	Channel 0 low-order byte	Channel 0 high-order byte

Data Packing for 16-Bit Stereo PCM:

Sample 1			
Channel 0 (left) low-order byte	Channel 0 (left) high-order byte	Channel 1 (right) low-order byte	Channel 1 (right) high-order byte

Data Format of the Samples

Each sample is contained in an integer i. The size of i is the smallest number of bytes required to contain the specified sample size. The least significant byte is stored first. The bits that represent the sample amplitude are stored in the most significant bits of i, and the remaining bits are set to zero.

For example, if the sample size (recorded in nBitsPerSample) is 12 bits, then each sample is stored in a two-byte integer. The least significant four bits of the first (least significant) byte is set to zero.

The data format and maximum and minimums values for PCM waveform samples of various sizes are as follows:

SampleSize	DataFormat	Max.Value	MinimumValue
One to eight bits	Unsigned integer	255 (0xFF)	0
Nine or more bits	Signed integer i	Largest positive value of i	Most negative value of i

For example, the maximum, minimum, and midpoint values for 8-bit and 16-bit PCM waveform data are as follows:

Format	Max.Value	Min.Value	MidpointValue
8-bit PCM	255 (0xFF)	0	128 (0x80)
16-bit PCM	32767 (0x7FFF)	-32768 (-0x8000)	0

Examples of PCM WAVE Files

Example of a PCM WAVE file with 11.025 kHz sampling rate, mono, 8 bits per sample:

```
RIFF( 'WAVE'      fmt(1, 1, 11025, 11025, 1, 8)
                  data( <wave-data> ) )
```

Example of a PCM WAVE file with 22.05 kHz sampling rate, stereo, 8 bits per sample:

```
RIFF( 'WAVE'      fmt(1, 2, 22050, 44100, 2, 8)
                  data( <wave-data> ) )
```

Example of a PCM WAVE file with 44.1 kHz sampling rate, mono, 20 bits per sample:

```
RIFF( 'WAVE'     INFO(INAM("O Canada"Z))
                  fmt(1, 1, 44100, 132300, 3, 20)
                  data( <wave-data> ) )
```

Storage of WAVE Data

The *<wave-data>* contains the waveform data. It is defined as follows:

```
<wave-data> ->   { <data-ck> : <data-list> }
<data-ck>   ->   data( <wave-data> )
<wave-list> ->   LIST( 'wavl' { <data-ck> :     // Wave samples
                       <silence-ck> }... )     // Silence
<silence-ck> ->  slnt( <dwSamples:DWORD> )     // Count of
                                               // silent samples
```

Note: The `slnt' chunk represents silence, not necessarily a repeated zero volume or baseline sample. In 16-bit PCM data, if the last sample value played before the silence section is a 10000, then if data is still output to the D to A converter, it must maintain the 10000 value. If a zero value is used, a click may be heard at the start and end of the silence section. If play begins at a silence section, then a zero value might be used since no other information is available. A click might be created if the data following the silent section starts with a nonzero value.

FACT Chunk

The <fact-ck> fact chunk stores important information about the contents of the WAVE file. This chunk is defined as follows:

```
<fact-ck> -> fact( <dwFileSize:DWORD> )  // Number of samples
```

The `fact'' chunk is required if the waveform data is contained in a **`wavl''** LIST chunk and for all compressed audio formats. The chunk is not required for PCM files using the **`data''** chunk format.

The "fact" chunk will be expanded to include any other information required by future WAVE formats. Added fields will appear following the *<dwFileSize>* field. Applications can use the chunk size field to determine which fields are present.

Cue-Points Chunk

The *<cue-ck>* cue-points chunk identifies a series of positions in the waveform data stream. The <cue-ck> is defined as follows:

```
<cue-ck> ->   cue( <dwCuePoints:DWORD>   // Count of cue points
                  <cue-point>... )       // Cue-point table
<cue-point> ->    struct
                  {
                        DWORD  dwName;
                        DWORD  dwPosition;
                        FOURCC fccChunk;
                        DWORD  dwChunkStart;
                        DWORD  dwBlockStart;
                        DWORD  dwSampleOffset;
                  }
```

The *<cue-point>* fields are as follows:

Field	Description
dwName	Specifies the cue point name. Each *<cue-point>* record must have a unique dwName field.
dwPosition	Specifies the sample position of the cue point.This is the sequential sample number within the play order. See ``Playlist Chunk,'' later in this document, for a discussion of the play order.
fccChunk	Specifies the name or chunk ID of thechunk containing the cue point.
dwChunkStart	Specifies the file position of the start of the chunk containing the cue point. This is a byte offset relative to the start of the data section of the `wavl' LIST chunk.
dwBlockStart	Specifies the file position of the start of

	the block containing the position. This is a byte offset relative to the start of the data section of the `wavl' LIST chunk.
dwSampleOffset	Specifies the sample offset of the cuepoint relative to the start of the block.

Examples of File Position Values

The following table describes the <cue-point> field values for a WAVE file containing multiple `data' and `slnt' chunks enclosed in a `wavl' LIST chunk:

CuePointLoc.	Field	Value
a `slnt'	fccChunk	FOURCC value `slnt'.
	dwChunkStart	File position of the`slnt' chunk relative to the start of the data section in the `wavl' LIST chunk.
	dwBlockStart	File position of the datasection of the `slnt' chunk relative to the start of the data section of the `wavl' LIST chunk.
	dwSampleOffset	Sample position of the cuepoint relative to the start of the `slnt' chunk.
In a PCM `data' chunk	fccChunk	FOURCC value `data'.
	dwChunkStart	File position of the`data' chunk relative to the start of the data section in the `wavl' LIST chunk.
	dwBlockStart	File position of the cuepoint relative to the start of the data section of the `wavl' LIST chunk.
	dwSampleOffset	Zero value.
In a compressed `data' chunk	fccChunk	FOURCC value `data'.
	dwChunkStart	File position of the startof the `data' chunk relative to the start of the data section of the `wavl' LIST chunk.
	dwBlockStart	File position of theenclosing block relative to the start of the data section of the `wavl' LIST chunk. The software can begin the decompression at this point.
	dwSampleOffset	Sample position of the cuepoint relative to the start of the block.

The following table describes the <cue-point> field values for a WAVE file containing a single `data' chunk:

CuePointLoc.	Field	Value
Within PCM data	fccChunk	FOURCC value `data'.
	dwChunkStart	Zero value.
	dwBlockStart	Zero value.
	dwSampleOffset	Sample position of the cuepoint relative to the start of the `data' chunk.
In a compressed `data' chunk	fccChunk	FOURCC value `data'.
	dwChunkStart	Zero value.
	dwBlockStart	File position of theenclosing block relative to the start of the `data' chunk. The software can begin the decompression at this point.
	dwSampleOffset	Sample position of the cuepoint relative to the start of the block.

Playlist Chunk

The *<playlist-ck>* playlist chunk specifies a play order for a series of cue points. The <playlist-ck> is defined as follows:

```
<playlist-ck> -> plst(  <dwSegments:DWORD>   // Count of play segments
                        <play-segment>... )  // Play-segment table

<play-segment> ->  struct {
                              DWORD dwName;
                              DWORD dwLength;
                              DWORD dwLoops;
                            }
```

The *<play-segment>* fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the <cue-ck> cue-point table.
dwLength	Specifies the length of the section in samples.
dwLoops	Specifies the number of times to play the section.

Associated Data Chunk

The *<assoc-data-list>* associated data list provides the ability to attach information like labels to sections of the waveform data stream. The *<assoc-data-list>* is defined as follows:

```
<assoc-data-list> ->  LIST('adtl'
```

```
                              <labl-ck>         // Label
                              <note-ck>         // Note
                              <ltxt-ck>         // Text with data length
                              <file-ck> )       // Media file

<labl-ck> ->      labl( <dwName:DWORD> <data:ZSTR> )

<note-ck> ->      note( <dwName:DWORD> <data:ZSTR> )

<ltxt-ck> ->      ltxt( <dwName:DWORD>
                        <dwSampleLength:DWORD>
                        <dwPurpose:DWORD>
                        <wCountry:WORD>
                        <wLanguage:WORD>
                        <wDialect:WORD>
                        <wCodePage:WORD>
                        <data:BYTE>... )

<file-ck> ->      file( <dwName:DWORD>
                        <dwMedType:DWORD>
                        <fileData:BYTE>...)
```

Label and Note Information

The `labl' and `note' chunks have similar fields. The `labl' chunk contains a label, or title, to associate with a cue point. The `note' chunk contains comment text for a cue point. The fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the *<cue-ck>* cue-point table.
data	Specifies a NULL-terminated string containing a text label (for the *`labl'* chunk) or comment text (for the *`note'* chunk).

Text with Data Length Information

The `ltxt'' chunk contains text that is associated with a data segment of specific length. The chunk fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the *<cue-ck>* cue-point table.
dwSampleLength	Specifies the number of samples in the segment of waveform data.
dwPurpose	Specifies the type or purpose of the text. For example, dwPurpose can specify a FOURCC code like *`scrp'* for script text or *`capt'* for close-caption text.
wCountry	Specifies the country code for the text. See *``Country Codes''* in Chapter 2, *``Resource Interchange File*

	Format,'' for a current list of country codes.
wLanguage, wDialect	Specify the language and dialect codes for the text. See ``Language and Dialect Codes'' in Chapter 2, ``Resource Interchange File Format,'' for a current list of language and dialect codes.
wCodePage	Specifies the code page for the text.

Embedded File Information

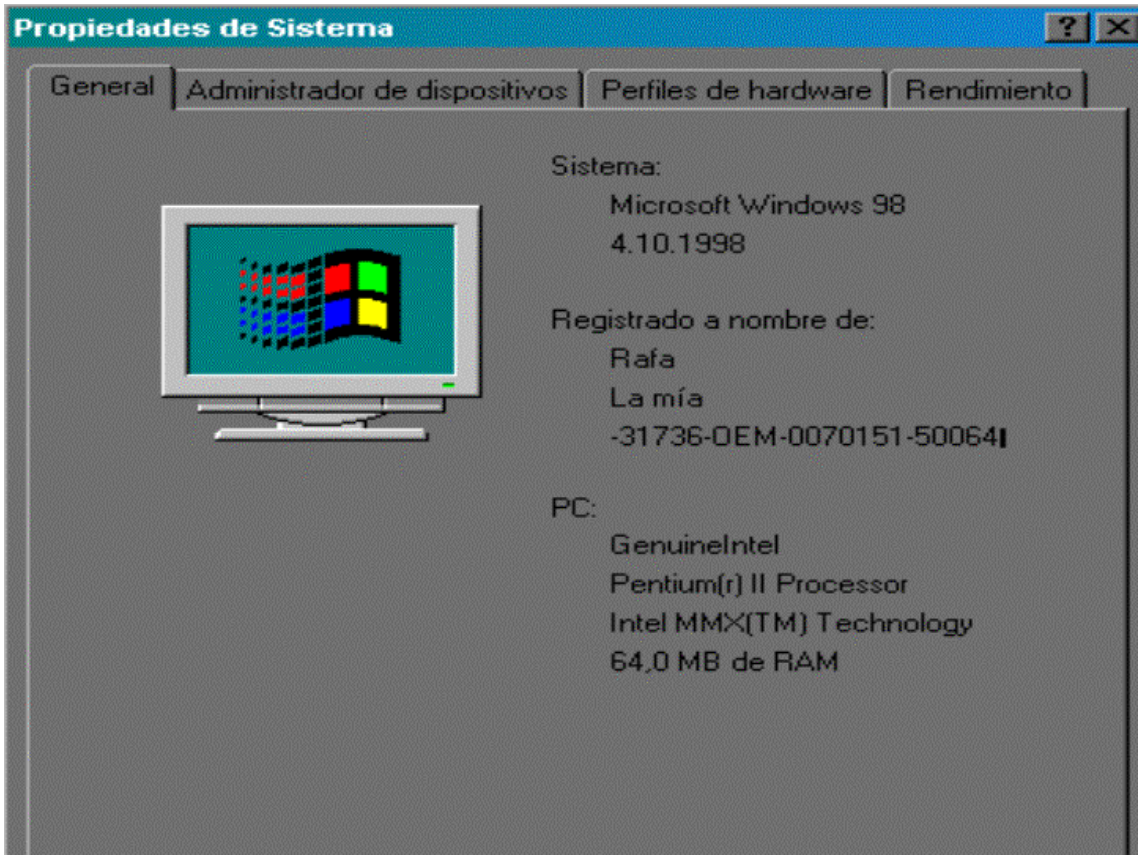
The `file' chunk contains information described in other file formats (for example, an `RDIB' file or an ASCII text file). The chunk fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the *<cue-ck>* cue-point table.
dwMedType	Specifies the file type contained in the fileData field. If the fileData section contains a RIFF form, the *dwMedType* field is the same as the RIFF form type for the file. This field can contain a zero value.
fileData	Contains the media file.

III. APÉNDICE B: CARACTERÍSTICAS TÉCNICAS DEL HARDWARE EMPLEADO EN LA REALIZACIÓN DEL PROYECTO.

APENDICE B: CARACTERISTICAS TECNICAS DEL HARDWARE EMPLEADO EN LA REALIZACION DEL PROYECTO

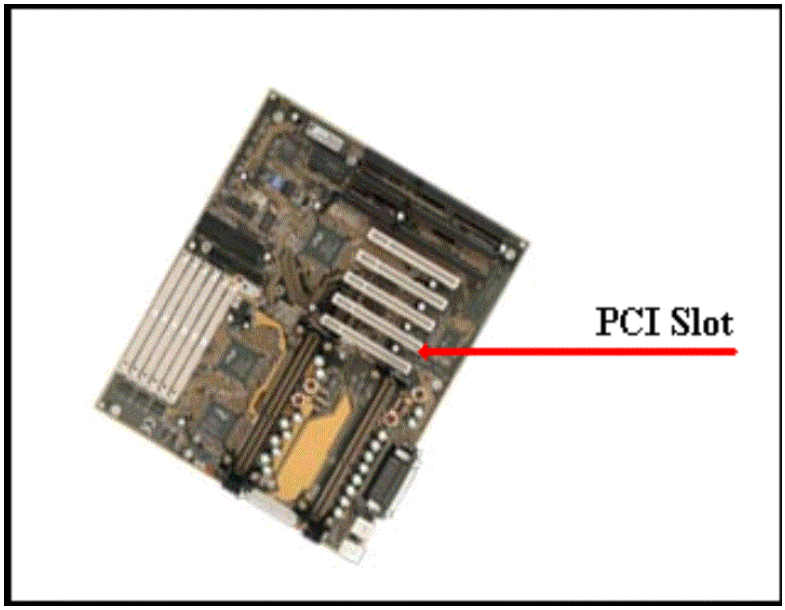
1) Ordenador personal (PC), Pentium II-450 MHz, 64 Mbytes SDRAM:



2) Tarjeta con dispositivo de captura de TV/FM:

- Requerimientos del sistema:

1. IBM PC compatible, Pentium processor or higher.
2. VGA card with their chipset specific Direct Draw 5.0 driver.
3. One free 32-bit PCI V2.1 slot.
4. Runs under Microsoft Windows 95/98.



- Características técnicas:

The WinView video capture cards come with following features:

1. PCI 2.1 compliance with Plug & Play compatibility.
2. No feature connector, VGA loop-back cable needed.

3. Video/Audio interleave data stream conforms to Microsoft AVI file format.

4. Continuous AVI file capturing with up to 30 frames per second.

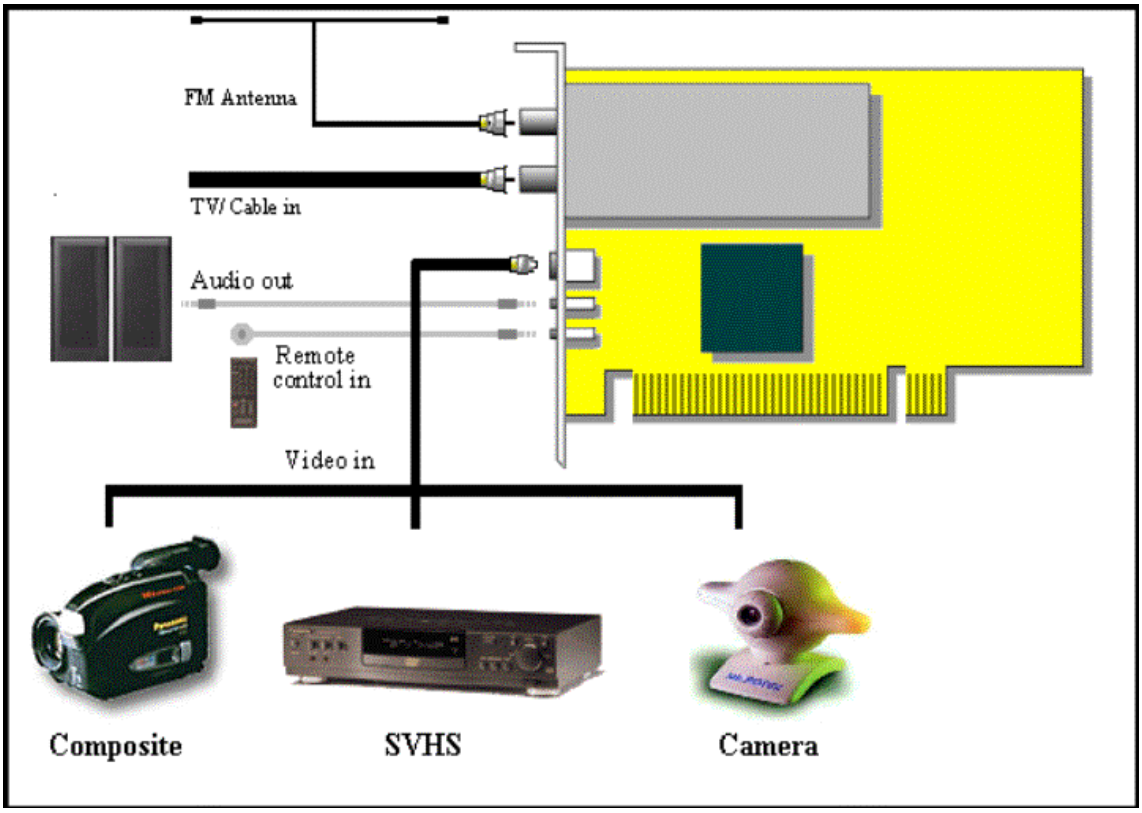
5. Microsoft Video for Windows driver interface for user application.

6. Support major VGA card with their chipset specific Direct Draw 5.0 driver.

7. TV/FM tuner module ready.

8. Optional CCD camera, video conference kit.

9. External input sources include one composite AV via RCA jack, and one super VHS via 4-pin mini-din connector.



The features described above are general in WinView series products. Please refer to their respective product catalog, manual or packing box for detail product features and specifications.

3) Tarjeta de sonido Sound Blaster AudioPCI 128:

- Prestaciones principales de Sound Blaster AudioPCI 128:

Además del software de instalación, las utilidades de configuración, los controladores de dispositivo y las utilidades de inicialización, Sound Blaster AudioPCI 128:

1. Sintetizador:

Proporciona hasta 128 voces de síntesis de tabla de ondas.

Incluye lotes de sonidos General MIDI/ Roland MT-32 de 2MB y 4MB así como emulación de sintetizador AdLib/ SoundBlaster 16 (con OPL2) FM.

Debido a la alta frecuencia de transferencia de datos que es posible conseguir a través del bus PCI, Sound Blaster AudioPCI 128 puede utilizar un método flexible para almacenar y acceder a los sonidos de tabla de ondas. Todos los sonidos de los instrumentos de 128 General MIDI están almacenados en el disco duro del ordenador para acceder a ello utilizando la RAM del ordenador. Esta tecnología permite lotes de expansión de sonido.

2. Audio Digital:

Permite la grabación de sonido en estéreo con calidad de CD (hasta 16 bits, frecuencia de muestreo de 48.0 Khz) desde entradas de Aux, CD y Línea así como desde el sintetizador interno.

Permite la grabación de sonido en mono desde la entrada de Micro con 20 Dbs de mejora.

Permite la grabación de sonido en mono desde la entrada/salida de TAD.

Reproduce muestras de ondas de estéreo de sonido de 8 o 16 bits.

Activa la reproducción de archivos de onda estándar de ordenador (.voc, .wav, etc.).

3. MIDI:

Proporciona una interfaz MPU-401 compatible con MIDI, incluida una entrada de MIDI y una salida MIDI (requiere un cable externo que no se suministra).

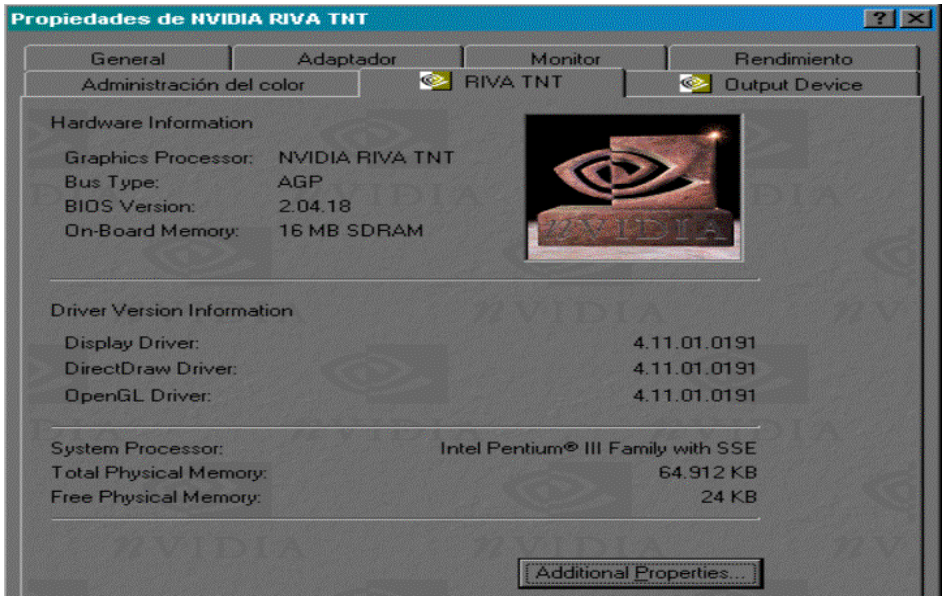
Permite la grabación de eventos MIDI desde un controlador de teclado MIDI.

Activa la reproducción de archivos MIDI estándar de ordenador (p.ej. .MID, .ROL, etc.) tanto en el sintetizador interno como en uno externo.

4. Mezclador:

El mezclador de software permite mezclar las salidas del sintetizador y de audio digital con señales que procedan de las entradas de CD, Aux, Mic y TAD de la tarjeta.

4) Tarjeta gráfica:



IV. RUTINAS EMPLEADAS. CÓDIGOS.

FUNCION WAVELOAD.M

```
function [y,Fs,Format]=waveload(wavefile)

if nargin~=1
   error('Waveload tiene un argumento, que es el nombre del fichero
         .WAV');
end

if isempty(findstr(wavefile,'.')),
   wavefile=[wavefile,'.wav'];
end

fid=fopen(wavefile,'rb');    % 'r' es sólo lectura y 'b' indica 'big-
                             % endian byte ordering'
                             % para fread.

if fid ~= -1                 % Si fopen devuelve un -1 no se puede
                             % abrir el fichero.
   % read riff chunk
   header=fread(fid,4,'uchar');
   header=fread(fid,1,'uint32');
   header=fread(fid,4,'uchar');

   % read format sub-chunk
   header=fread(fid,4,'uchar');
   header=fread(fid,1,'uint32');

   Format(1)=fread(fid,1,'ushort');    % Valor 1, entonces formato PCM
   Format(2)=fread(fid,1,'ushort');    % Número de canales (1 ó 2).
   Fs=fread(fid,1,'uint32');           % Tasa de muestreo.
   Format(3)=Fs;
   Format(4)=fread(fid,1,'uint32');    % Promedio de bytes por segundo
   Format(5)=fread(fid,1,'ushort');    % Alineamiento de bloque
   Format(6)=fread(fid,1,'ushort');    % Bits por muestra

   y=[];
   DataNotFound=1;                     % Búsqueda del data chunk

   while DataNotFound,
```

```
                                        % Busca de dos en dos bytes
     header=fread(fid,2,'uchar');       % hasta que encuentra 'da',
     if header-['da']' == 0,            % de esta forma ignoramos los
         DataNotFound=0;                % espacios en blanco.
     end
   end

   header=fread(fid,2,'uchar');                % Búsqueda de 'ta'
   nsamples=fread(fid,1,'uint32')/Format(5);   % Número de muestras.

   if Format(6)==8,
      y=fread(fid,nsamples,'uchar');   % Las muestras son de 8 bits.
      y=y-128;                         % Quita el offset (mean(y)
      y=y./128;                        % devuelve la media de y).
                                       % y normaliza las muestras.
   elseif Format(6)==16,
      y=fread(fid,nsamples,'short');   % Las muestras son de 16 bits.
      y=y-0;                           % Cuando se codifica con 16
      y=y./32768;                      % bits ya se ha eliminado
   end                                 % el offset, sólo hay que
                                       % normalizar.
   fclose(fid);
 end

 if fid == -1
    error('No se puede abrir el fichero .WAV debido al fichero de
entrada!');
 end;
```

FUNCION OVERADD.M

```
function [s]=overadd(x,h,Lh,M,Ex)

% Función que realiza el procedimiento "overlap and add".
% h corresponde a la señal más corta,o sea, al trozo del anuncio que
% queremos detectar. En este caso x corresponde a la señal que nos
% llega de la radio.
% M es la longitud de las secciones. Dividiremos "x" en L secciones de
% longitud M, suponiendo siempre que la longitud de "x", Lx, es
% múltiplo de M.
Lx=length(x);
L=Lx-Lh+1;           % En realidad esta es la longitud de la señal de
                     % correlación que realmente nos es útil.
N=M+Lh-1;            % En primer lugar determinamos N, que va a ser la
                     % longitud de la FFT
                     % con la que vamos a trabajar.
N=2^ceil(log2(N));   % Para que el cálculo de la DFT sea más eficiente

L=ceil(Lx/M);     % L=nº de secciones de longitud M en que se divide x
h1=flipud(h);     % Invertimos una de las señales para obtener la
                  % correlación.
DFTH=fft(h1,N);

for i=1:L
   DFTX=fft(x(M*(i-1)+1:min(M*i,Lx)),N);
   Y=DFTX.*DFTH;
   MAT(i,M*(i-1)+1:M*(i-1)+N)=ifft(Y,N)';
end

s=abs(sum(MAT));   % Tomamos sólo las muestras que nos interesan, pues
                   % las colas de la correlación no nos sirven para
s=s(Lh:Lx);        % nuestro estudio.
Eh=sum(h.^2);
% Normalizamos cada uno de los coeficientes de correlación por la energía
if Eh>0            % de las señales utilizadas para su cálculo. El vector
s=s./sqrt(Ex*Eh);  % Ex se corresponde con los parámetros de normalización
end                % calculados con la función energía.m. Ex se calcula
                   % de antemano debido al elevado número de cálculos.
```

FUNCION OVERADD1.M

```
function [s,noper]=overadd1(Lh,x,y,M,Ex)

% Función que realiza el procedimiento "overlap and add".
% h corresponde a la señal más corta,o sea, al trozo del anuncio que
% queremos detectar en este caso.
% x corresponde a la señal que nos llega de la radio.
% M es la longitud de las secciones. Dividiremos "x" en L secciones de
% longitud M, supo niendo siempre que la longitud de "x", Lx, es
% múltiplo de M.

flops(0);
tic;
i=1;
Lx=length(x);
L=Lx-Lh+1;       % En realidad esta es la longitud de la
                 % señal de correlación que realmente nos es útil.

alea=floor(rand(1,1)*L);   % Tomamos aleatoriamente un trozo del
                           % anuncio, que va a hacer las funciones
h=y(alea:alea+Lh-1);       % de la señal que recibimos de la radio,
                           % h(n).

N=M+Lh-1;            % En primer lugar determinamos N, que va a ser la
                     % longitud de la FFT con la que vamos a trabajar.
N=2^ceil(log2(N));   % Para que el cálculo de la DFT sea más eficiente

L=ceil(Lx/M);     % L=nº de secciones de longitud M en que se divide x
h1=flipud(h);     % Invertimos una de las señales para obtener la
                  % correlación.
DFTH=fft(h1,N);
MAT=zeros(L,M*(L-1)+N);
s=zeros(1,M*(L-1)+N);

for i=1:L
   DFTX=fft(x(M*(i-1)+1:min(M*i,Lx)),N);
   Y=DFTX.*DFTH;
   MAT(i,M*(i-1)+1:M*(i-1)+N)=ifft(Y,N)';
end
```

```
s=abs(sum(MAT));
s=s(Lh:Lx);         % Tomamos sólo las muestras que nos interesan, pues
                    % las colas de la co-
Eh=sum(h.^2);       % rrelación no nos sirven para nuestro estudio.
s=s./sqrt(Ex*Eh);   % Normalizamos cada uno de los coeficientes de
                    % correlación por la energía de las señales
                    % utilizadas para su cálculo. El vector Ex se
toc                 % corresponde con los parámetros de normalización
                    % calculados con la función energía. Ex se calcula
noper=flops;        % de antemano debido al elevado número de cálculos.

[ymax,maxindex]=max(s)
plot([1:length(s)],s);
```

FUNCION ENERGIA.M

```
function [Ex]=energia(Lh,x,M)

% Función que calcula la energía del anuncio que se le pasa como
% entrada. Este valor de la energía se utilizará en la función
% overadd.m para normalizar el vector resultante de la correlación.
% El método utilizado para calcular la energía es el de correlar las
% muestras del anuncio con un vector de 1's que tiene la misma
% longitud que la otra señal a correlar, Lh.

unos=ones(Lh,1);
Lx=length(x);

N=M+Lh-1;
N=2^ceil(log2(N));

L=ceil(Lx/M);
DFTH=fft(unos,N);
MAT=zeros(L,M*(L-1)+N);

x=x.^2;

for i=1:L

   DFTX=fft(x(M*(i-1)+1:min(M*i,Lx)),N);
   Y=DFTX.*DFTH;
   MAT(i,M*(i-1)+1:M*(i-1)+N)=ifft(Y,N)';

end

Ex=abs(sum(MAT));

Ex=Ex(Lh:Lx);
```

FUNCION CUATRO.M

```
% Con esta función lo que se hace es diezmar el anuncio que tenemos
% grabado, ya que con los dispositivos de los que disponemos la mínima
% tasa de muestreo permitida es de 8000 Hz. Para trabajar con 4000 Hz
% hay que diezmar las muestras, tomando una de cada dos muestras.


function [salida]=cuatro(entrada)

L=length(entrada);
salida=entrada(1:2:L);
```

FUNCION ESTADISTICAS.M

```
function [ro,muestras]=estadisticas(Lh,x,y,z)

% Esta función nos va a servir para ajustar los valores de la
% correlación a una función de distribución, de forma que
% posteriormente se tenga un proceso monitorizado del que se puedan
% obtener las probabilidades de detección correcta, detección
% incorrecta, fallo,...
% Para ajustar dicha f. de dist. haremos múltiples pruebas con varios
% anuncios. Así se tomarán diferentes emisiones de una misma emisora,
% y también de distintas emisoras, de un mismo anuncio, de manera que
% nos acerquemos lo más posible al caso real.

% Para cada comparación o correlación salvaremos los siguientes
% valores:

   % Máximo de los coeficientes de correlación, útil para
   % establecer un umbral que nos dé la probabilidad de detección, así
   % como el valor de la abscisa para el que se obtiene-.

   % Cincuenta coeficientes tomados aleatoriamente de la correlación,
   % exceptuando el máximo coef. de correlación y los valores que
   % resultan a su alrededor en una distancia Lh, pues estos
   % pueden falsear el ajuste de la func. dens. de probabilidad. Estos
   % cincuenta valores nos servirán para calcular la probabilidad de
   % falsa alarma (probabilidad de detección errónea).


% La función dens. de probabilidad que vamos a utilizar es la función
% beta, que se ajusta muy bien a nuestras necesidades, ya que
% trabajaremos con el módulo de los coefic. de correlación, que varía
% entre 0 y 1.

% Esto lo haremos para distintas frecuencias de muestreo y distintas
% duraciones de la señal tomada de la radio, o sea se hará una
% simulación para cada una de las parejas resultantes de tomar una
% frecuencia y un tiempo de conmutación:
```

```
              % Tiempos                                  Frecuencias
                % 2     seg                              44.100 KHz
                % 1      "                               22.050 "
                % 1/2    "                               11.025 "
                %  ...                                    8.000 "
                % 1/128  "

ro=zeros(1,6);
aleat=zeros(1,50);
muestras=zeros(1,300);          % La longitud de las otras versiones
                                % del anuncio es la misma.
Lx=length(x);                   % M es la longitud de las secciones
                                % para el overlapp&add (sp. buscamos
if Lx>1572000, M=(2^19)-Lh+1;   % al menos tres secciones).
elseif 1572000>Lx & Lx>786000, M=(2^18)-Lh+1;
elseif 786000>Lx & Lx>393000, M=(2^17)-Lh+1;
elseif 393000>Lx & Lx>196000, M=(2^16)-Lh+1;
elseif 196000>Lx & Lx>98000, M=(2^15)-Lh+1;
elseif Lx<98000, M=(2^14)-Lh+1;
end
[Ex]=energia(Lh,x,M);


s(1,:)=overadd1(Lh,x,y,M,Ex);     % Se hacen cinco pruebas para cada
                                % versión del anuncio. La ver-
s(2,:)=overadd1(Lh,x,z,M,Ex);     % sión "x" es la original, la "y" y
                                % la "z" son versiones de la
s(3,:)=overadd1(Lh,x,y,M,Ex);     % misma o distinta emisora.

s(4,:)=overadd1(Lh,x,z,M,Ex);

s(5,:)=overadd1(Lh,x,y,M,Ex);

s(6,:)=overadd1(Lh,x,z,M,Ex);

L=Lx-Lh+1;    % L es la longitud de cada vector de correlación.
              % Tomamos los valores que posteriormente vamos a
for i=1:6     % utilizar para caracterizar el proceso estadísticamente
   r=s(i,:);
   [ro(i),posic(i)]=max(r);
   for j=1:50
```

```
      aleat(j)=floor(rand(1,1)*L);
      while (posic(i)-Lh)<aleat(j) & aleat(j)<(posic(i)+Lh)
         aleat(j)=floor(rand(1,1)*L);
      end
   end
   muestras(50*(i-1)+1:50*i)=r(aleat);
end
```

FUNCION ESTADISTICAS2.M

```
function [ro,muestras]=estadisticas2(Lh,x1,y1,z1,x2,y2,z2,x3,y3,z3,
x4,y4,z4)

[ro1,muestras1]=estadisticas(Lh,x1,y1,z1);
save provisional.mat ro1 muestras1

[ro2,muestras2]=estadisticas(Lh,x2,y2,z2);
ro=[ro1 ro2];
muestras=[muestras1 muestras2];
save provisional.mat ro muestras

[ro3,muestras3]=estadisticas(Lh,x3,y3,z3);
ro=[ro1 ro2 ro3];
muestras=[muestras1 muestras2 muestras3];
save provisional.mat ro muestras

[ro4,muestras4]=estadisticas(Lh,x4,y4,z4);
ro=[ro1 ro2 ro3 ro4];
muestras=[muestras1 muestras2 muestras3 muestras4];
save provisional.mat ro muestras

ro=[ro1 ro2 ro3 ro4];
muestras=[muestras1 muestras2 muestras3 muestras4];


% Vemos que para cada pareja frecuencia-tiempo vamos obteniendo 24
% valores del máximo coef. de correlación y 1200 valores aleatorios de
% dicha correlación, almacenados en las matrices ro y muestras
% respectivamente.
```

FUNCION ESTAD44.M

```
function [ro44,muestras44]=estad44(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4 ,z4)

% Con esta función,y apoyándonos en las ya vistas, "estadísticas2.m" y %
"estadísticas.m", se calculan los valores buscados para una
% determinada frecuencia, en este caso 44.1 KHz. Así la frecuencia es
% una constante, y la variable será la duración del trozo de anuncio:
% 2s, 1s, 1/2s, %..., 1/128s. Dependiendo de estos dos parámetros
% evitaremos hacer los cálculos correspondientes a alguna de las
% parejas frecuencia-tiempo, pues se sabe a priori que no va a ser la
% pareja óptima por el alto número de operaciones a realizar:

%Descartamos las parejas: 44.1 KHz-2 seg (88200 muestras)
%                         44.1 KHz-1 seg (44100 muestras)
%                         44.1 KHz-1/2 seg (22050 muestras)
%                         44.1 KHz-1/4 seg (11025 muestras)
%                         44.1 KHz-1/8 seg (5512 muestras)
%                         44.1 KHz-1/16 seg (2756 muestras)
%                         44.1 KHz-1/32seg (1378 muestras)


%[ro44200,muestras44200]=estadisticas2(88200,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro44001,muestras44001]=estadisticas2(44100,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro44002,muestras44002]=estadisticas2(22050,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro44004,muestras44004]=estadisticas2(11025,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro44008,muestras44008]=estadisticas2(5512,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro44016,muestras44016]=estadisticas2(2756,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);
```

```
%[ro44032,muestras44032]=estadisticas2(1378,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

[ro44064,muestras44064]=estadisticas2(689,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro44128,muestras44128]=estadisticas2(344,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

ro44=[ro44064;ro44128];
muestras44=[muestras44064;muestras44128];
```

FUNCION ESTAD22.M

```
function [ro22,muestras22]=estad22(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4 ,z4)

% Con esta función,y apoyándonos en las ya vistas, "estadísticas2.m" y
% "estadísticas.m", se calculan los valores buscados para una
% determinada frecuencia, en este caso 22.05 KHz. Así la frecuencia es
% una constante, y la variable será la duración del trozo de anuncio:
% 2s, 1s, 1/2s,..., 1/128s. Dependiendo de estos dos parámetros
% evitaremos hacer los cálculos correspondientes a alguna de las
% parejas frecuencia-tiempo, pues se sabe a priori que no va a ser la
% pareja óptima por el alto número de operaciones a realizar:


% Descartamos las parejas: 22.050 KHz-2 seg (44100 muestras)
%                          22.050 KHz-1 seg (22050 muestras)
%                          22.050 KHz-1/2 seg (11025 muestras)
%                          22.050 KHz-1/4 seg (5512 muestras)
%                          22.050 KHz-1/8 seg (2756 muestras)
%                          22.050 KHz-1/16 seg (1378 muestras)


%[ro22200,muestras22200]=estadisticas2(44100,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro22001,muestras22001]=estadisticas2(22050,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro22002,muestras22002]=estadisticas2(11025,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro22004,muestras22004]=estadisticas2(5512,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro22008,muestras22008]=estadisticas2(2756,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro22016,muestras22016]=estadisticas2(1378,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);
```

```
[ro22032,muestras22032]=estadisticas2(689,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro22064,muestras22064]=estadisticas2(344,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro22128,muestras22128]=estadisticas2(172,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

ro22=[ro22032;ro22064;ro22128];
muestras22=[muestras22032;muestras22064;muestras22128];
```

FUNCION ESTAD11.M

```
function [ro11,muestras11]=estad11(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4 ,z4)

% Con esta función,y apoyándonos en las ya vistas, "estadísticas2.m" y
% "estadísticas.m", se calculan los valores buscados para una
% determinada frecuencia, en este caso 11.025 KHz. Así la frecuencia
% es una constante, y la variable será la duración del trozo de
% anuncio: 2s, 1s, 1/2s,..., 1/128s.Dependiendo de estos dos
% parámetros evitaremos hacer los cálculos correspondientes
% a alguna de las parejas frecuencia-tiempo, pues se sabe a priori que
% no va a ser la pareja óptima por el alto número de operaciones a
% realizar:


% Descartamos las parejas: 11.025 KHz-2 seg (22050 muestras)
%                          11.025 KHz-1 seg (11025 muestras)
%                          11.025 KHz-1/2 seg (5512 muestras)
%                          11.025 KHz-1/4 seg (2576 muestras)
%                          11.025 KHz-1/8 seg (1378 muestras)

%[ro11200,muestras11200]=estadisticas2(22050,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro11001,muestras11001]=estadisticas2(11025,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro11002,muestras11002]=estadisticas2(5512,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro11004,muestras11004]=estadisticas2(2756,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

%[ro11008,muestras11008]=estadisticas2(1378,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,
y4,z4);

[ro11016,muestras11016]=estadisticas2(689,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);
```

```
[ro11032,muestras11032]=estadisticas2(344,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro11064,muestras11064]=estadisticas2(172,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro11128,muestras11128]=estadisticas2(86,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

ro11=[ro11016;ro11032;ro11064;ro11128];
muestras11=[muestras11016;muestras11032;muestras11064;muestras11128];
```

FUNCION ESTAD8.M

```
function [ro8,muestras8]=estad8(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4)

% Con esta función,y apoyándonos en las ya vistas, "estadísticas2.m" y %
"estadísticas.m", se calculan los valores buscados para una % %
% determinada frecuencia, en este caso 8 KHz. Así la frecuencia es una
% constante, y la variable será la duración del trozo de anuncio: 2s,
% 1s, 1/2s,..., 1/128s. Dependiendo de estos dos parámetros evitaremos
% hacer los cálculos correspondientes a alguna de las parejas
% frecuencia-tiempo, pues se sabe a priori que no va a ser la pareja
% óptima por el alto número de operaciones a realizar:

% Descartamos las parejas: 8 KHz-2 seg (16000 muestras)
%                          8 KHz-1 seg (8000 muestras)
%                          8 KHz-1/2 seg (4000 muestras)
%                          8 KHz-1/4 seg (2000 muestras)


%[ro8200,muestras8200]=estadisticas2(16000,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4
,z4);

%[ro8001,muestras8001]=estadisticas2(8000,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

%[ro8002,muestras8002]=estadisticas2(4000,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

%[ro8004,muestras8004]=estadisticas2(2000,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro8008,muestras8008]=estadisticas2(1000,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro8016,muestras8016]=estadisticas2(500,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro8032,muestras8032]=estadisticas2(250,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);
```

```
[ro8064,muestras8064]=estadisticas2(125,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro8128,muestras8128]=estadisticas2(62,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

ro8=[ro8008;ro8016;ro8032;ro8064;ro8128];
muestras8=[muestras8008;muestras8016;muestras8032;muestras8064;
muestras8128];
```

FUNCION ESTAD4.M

```
function [ro4,muestras4]=estad4(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4)

% Con esta función,y apoyándonos en las ya vistas, "estadísticas2.m" y
% "estadísticas.m", se calculan los valores buscados para una
% determinada frecuencia, en este caso 4 KHz. Así la frecuencia es una
% constante, y la variable será la duración del trozo de anuncio: 2s,
% 1s, 1/2s,..., 1/128s.Dependiendo de estos dos parámetros evitaremos
% hacer los cálculos correspondientes a alguna de las parejas
% frecuencia-tiempo, pues se sabe a priori que no va a ser la pareja
% óptima por el alto número de operaciones a realizar:

%Descartamos las parejas: 4 KHz-2 seg (8000 muestras)
%                         4 KHz-1 seg (4000 muestras)
%                         4 KHz-1/2 seg (2000 muestras)


%[ro4200,muestras4200]=estadisticas2(8000,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

%[ro4001,muestras4001]=estadisticas2(4000,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

%[ro4002,muestras4002]=estadisticas2(2000,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro4004,muestras4004]=estadisticas2(1000,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro4008,muestras4008]=estadisticas2(500,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro4016,muestras4016]=estadisticas2(250,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro4032,muestras4032]=estadisticas2(125,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);
```

```
[ro4064,muestras4064]=estadisticas2(62,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

[ro4128,muestras4128]=estadisticas2(31,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,
z4);

ro4=[ro4004;ro4008;ro4016;ro4032;ro4064;ro4128];
muestras4=[muestras4004;muestras4008;muestras4016;muestras4032;
muestras4064;muestras4128];
```

FUNCION GRAFICO.M

```
function [fdistro,fdistmu,perdida,falarm]=grafico(ro,muestras)

% Función que calcula la función estadística beta para las muestras
% correspondientes a una determinada pareja frecuencia-tiempo.También
% calcula la probabilidad de pérdida y de falsa alarma de dicha
% distribución.

t=[0:0.001:1];

[phat1,pci1]=betafit(ro);
[phat2,pci2]=betafit(muestras);

y1=betapdf(t,phat1(1),phat1(2));
y2=betapdf(t,phat2(1),phat2(2));

fdistro=y1;
fdistmu=y2;

plot(t,y1,t,y2);
zoom
umbral=input('Introducir el umbral:');

perdida=betacdf(umbral,phat1(1),phat1(2))
falarm=1-betacdf(umbral,phat2(1),phat2(2))
```

FUNCION DECISOR.M

```
function [numdetec,tabla_detecciones]=decisor(anuncio)

% La función de este programa es la de detectar anuncios en FM.
% Trabajaremos con una tasa de muestreo de 4 KHz y el trozo de señal
% que iremos cogiendo de la emisora de radio será de 1/4 seg
% (Lh=1000). Trataremos de detectar un solo anuncio en una sola
% emisora en tiempo real.
% detec es un vector que contabiliza el número de detecciones en una
% prueba.
% tdetec es un vector que mide el tiempo del anuncio transcurrido
% después de cada detección positiva.
% numdetec es un entero que mide el número de detecciones totales
% positivas.
% tdur es la duración del anuncio en cuestión.

clc
numdetec=0;
detec=zeros(1,15);
tdetec=zeros(1,15);
umbral=0.5;
x=anuncio;
Lx=length(x);
tdur=Lx/4000;                % Tiempo que dura el anuncio.
Lh=1000;

if Lx>1572000, M=(2^19)-Lh+1;
elseif 1572000>Lx & Lx>786000, M=(2^18)-Lh+1;
elseif 786000>Lx & Lx>393000, M=(2^17)-Lh+1;
elseif 393000>Lx & Lx>196000, M=(2^16)-Lh+1;
elseif 196000>Lx & Lx>98000, M=(2^15)-Lh+1;
elseif Lx<98000, M=(2^14)-Lh+1;
end

Ex=energia(Lh,x,M);       % NOTA: también lo podemos poner como parám.
                          % de entrada (+ rápido).
disp('ACTIVE LA EMISORA Y PULSE CUALQUIER TECLA PARA CONTINUAR');
pause
```

```
while detec(1)==0
   h=recordsnd(0.25,4000);
   corr=overadd(x,h,Lh,M,Ex);
   [valormax,index]=max(corr);

   if valormax>=umbral
      T0=datestr(now,15);    % T0 es la hora a la que comienza la
      tdetec(1)=0;           % detección.
      tdetec(2)=index/4000;
      detec(1)=1;
      i=1;
      while tdetec(i)<tdetec(i+1)<tdur
         i=i+1;
         h=recordsnd(0.25,4000);
         corr=overadd(x,h,Lh,M,Ex);
         [valormax,index]=max(corr);
         if valormax>=umbral
            detec(i)=1;
            tdetec(i+1)=index/4000;
         else                          % Si se dejara tdetec(i+1) a
            tdetec(i+1)=tdetec(i)+1e-6; % cero se saldría del bucle.
                                       % Si hay dos detecciones
            if detec(i-1)==0           % negativas después de una
               break          % positiva suponemos que ha sido una
            end               % falsa alarma y empezamos una nueva
         end                  % detección.
      end

      if sum(detec)>=ceil(i*0.6);
         dia=datestr(now,1);
         disp('EL ANUNCIO HA SIDO EMITIDO A LAS '),disp(T0),
         disp('DEL'),disp(dia)
         numdetec=numdetec+1;
         resultado=struct('hora',T0,'dia',dia);
         tabla_detecciones(numdetec,:)=struct2cell(resultado);
         respuesta=input('¿DESEA SEGUIR CON LA DETECCIÓN? NO=0,
                          SI=1   ');
         if respuesta==0
            return
         end
      end
```

```
      detec=zeros(1,15);
      tdetec=zeros(1,15);
   end
   if sum(h)==0
      return
   end
end
```

FUNCION WAVREAD.M

```
function [y,Fs,bits]=wavread(file,ext)

%WAVREAD Read Microsoft WAVE (".wav") sound file.

%   Y=WAVREAD(FILE) reads a WAVE file specified by the string FILE,
%   returning the sampled data in Y. The ".wav" extension is appended
%   if no extension is given.  Amplitude values are in the range
%   [-1,+1].

%   [Y,FS,BITS]=WAVREAD(FILE) returns the sample rate (FS) in Hertz
%   and the number of bits per sample (BITS) used to encode the
%   data in the file.

%   [...]=WAVREAD(FILE,N) returns only the first N samples from each
%       channel in the file.
%   [...]=WAVREAD(FILE,[N1 N2]) returns only samples N1 through N2 %
%   from each channel in the file.
%   SIZ=WAVREAD(FILE,'size') returns the size of the audio data
%   contained in the file in place of the actual audio data, returning
%   the vector SIZ=[samples channels].

%   Supports multi-channel data, with up to 16 bits per sample.


%   Copyright (c) 1984-96 by The MathWorks, Inc.
%   $Revision: 5.5 $  $Date: 1996/04/10 15:30:41 $

%   D. Orofino, 11/95

if nargin>2,
  error('Too many input arguments.');
end
% Append .wav extension if it's missing:
if isempty(findstr(file,.')),
  file=[file '.wav'];
end
fid=fopen(file,'rb','l');   % Little-endian
if fid == -1,
```

```
  error('Can''t open WAVE file for input.');
end

% Find the first RIFF chunk:
riffck=find_cktype(fid,'RIFF');
if ~isstruct(riffck), error(riffck); end

% The subchunk better be WAVE:
waveck=find_cktype(fid,'WAVE',1);
if ~isstruct(waveck), error(waveck); end

% Find <fmt-ck> chunk:
fmtck=find_cktype(fid,'fmt');
if ~isstruct(fmtck), error(fmtck); end

% Read <wave-format>:
wavefmt = read_wavefmt(fid,fmtck);

% Find <data-ck> chunk:
datack=find_cktype(fid,'data');
if ~isstruct(datack), error(datack); end

% Parse structure info for return to user:
Fs = wavefmt.nSamplesPerSec;
bits = wavefmt.nBitsPerSample;

% Determine if caller wants data:
if nargin<2, ext=[]; end    % Default - read all samples
exts=prod(size(ext));
if strncmp(lower(ext),'size',exts),
  % Caller doesn't want data - just data size:
  samples = read_wavedat(datack,wavefmt,-1);
  fclose(fid);
  if isstr(samples), error(samples); end
  y = [samples wavefmt.nChannels];
  return;
elseif exts>2,
  error('Index range must be specified as a scalar or 2-element vector.');
elseif (exts==1),
  ext=[1 ext];  % Prepend start sample index
end
```

```
% Read <wave-data>:
datack = read_wavedat(datack,wavefmt,ext);
fclose(fid);
if ~isstruct(datack), error(datack); end

% Return audio data to user:
y = datack.Data;

% end of wavread()


% ------------------------------------------------------------------------
% Private functions:
% ------------------------------------------------------------------------

% READ_CKINFO: Reads next RIFF chunk, but not the chunk data.

%   If optional sflg is set to nonzero, reads SUBchunk info instead.
%   Expects an open FID pointing to first byte of chunk header.
%   Returns a new chunk structure.

function ck=read_ckinfo(fid,sflg)
if nargin<2, sflg=0; end
ck.fid = fid;
ck.Data = [];
[s,cnt] = fread(fid,4,'char');
if cnt~=4, ck='Error reading file header.'; return; end
ck.ID = deblank(setstr(s'));
if ~sflg,
  % Read chunk size (skip if subchunk):
  [sz,cnt] = fread(fid,1,'ulong');
  if cnt~=1, ck='Error reading file header.'; return; end
  ck.Size = sz;
end
return;

% FIND_CKTYPE: Finds a chunk with appropriate type.

%   Searches from current file position specified by fid.
%   Leaves file positions to data of desired chunk.
```

```
%   If optional sflg is set to nonzero, finds a SUBchunk instead.

function ck = find_cktype(fid,type,sflg)
if nargin<3, sflg=0; end
while 1,
  ck=read_ckinfo(fid,sflg);
  if ~isstruct(ck), return; end
  if strcmp(ck.ID,type), return; end
  % Return error flag if this is not the desired subchunk type:
  if sflg, ck='Error reading file header.'; return; end
  % Skip over data in chunk:
  if(fseek(fid,ck.Size,0)==-1),
    ck='Error reading file header.'; return;
  end
end
return;

% READ_WAVEFMT: Read WAVE format chunk.

%   Assumes fid points to the wave-format subchunk.
%   Requires chunk structure to be passed, indicating
%   the length of the chunk in case we don't recognize
%   the format tag.

function fmt=read_wavefmt(fid,ck)
total_bytes=ck.Size; % # bytes in subchunk

% Read standard <wave-format> data:
fmt.wFormatTag      = fread(fid,1,'ushort'); % Data encoding format
fmt.nChannels       = fread(fid,1,'ushort'); % Number of channels
fmt.nSamplesPerSec  = fread(fid,1,'ulong');  % Samples per second
fmt.nAvgBytesPerSec = fread(fid,1,'ulong');  % Avg transfer rate
fmt.nBlockAlign     = fread(fid,1,'ushort'); % Block alignment
nbytes=14;  % # of bytes read so far

% Read format-specific info:
if fmt.wFormatTag==1,
  % Read standard <PCM-format-specific> info:
  [bits,cnt]=fread(fid,1,'ushort');
  nbytes=nbytes+2;
  if (cnt~=1)|(nbytes~=total_bytes),
```

```
    fmt='Error reading file format.'; return;
  end
  fmt.nBitsPerSample=bits;
else
  % Skip over any remaining bytes for unknown formats:
  if nbytes>total_bytes,
    fmt='Error reading file format.'; return;
  elseif nbytes<total_bytes,
    if(fseek(fid,total_bytes-nbytes,0)==-1),
      fmt='Error reading file format.'; return;
    end
  end
end
return;

% READ_WAVEDAT: Read WAVE data chunk

%   Assumes fid points to the wave-data chunk
%   Requires <data-ck> and <wave-format> structures to be passed.
%   Requires extraction range to be specified.
%   Setting ext=[] forces ALL samples to be read.  Otherwise,
%       ext should be a 2-element vector specifying the first
%       and last samples (per channel) to be extracted.
%   Setting ext=-1 returns the number of samples per channel,
%       skipping over the sample data.

function dat=read_wavedat(datack,wavefmt,ext)
total_bytes=datack.Size; % # bytes in this chunk

if wavefmt.wFormatTag==1,
  % PCM Format:
  % Determine # bytes/sample - format requires rounding
  %  to next integer number of bytes:
  BytesPerSample = ceil(wavefmt.nBitsPerSample/8);
  if     BytesPerSample==1, dtype='uchar'; % unsigned 8-bit
  elseif BytesPerSample==2, dtype='short'; % signed 16-bit
  else
    dat='Cannot read WAVE files with more than 16 bits/sample.'; return;
  end
  total_samples = datack.Size/BytesPerSample;
  SamplesPerChannel = total_samples/wavefmt.nChannels;
```

```
if ~isempty(ext) & ext==-1,
  % Just return the samples per channel, and fseek past data:
  dat = SamplesPerChannel;
  if(fseek(datack.fid,total_samples,0)==-1),
    dat='Error reading file.';
  end
  return;
end
% Determine sample range to read:
if isempty(ext),
  ext = [1 SamplesPerChannel];    % Return all samples
else
  if prod(size(ext))~=2,
    dat='Sample limit vector must have 2 elements.'; return;
  end
  if ext(1)<1 | ext(2)>SamplesPerChannel,
    dat='Sample limits out of range.'; return;
  end
  if ext(1)>ext(2),
    dat='Sample limits must be given in ascending order.'; return;
  end
end
% Skip over leading samples:
if ext(1)>1,
  % Skip over leading samples, if specified:
  if(fseek(datack.fid, ...
        BytesPerSample*(ext(1)-1)*wavefmt.nChannels,0)==-1),
    dat='Error reading file.'; return;
  end
end
% Read desired data:
nSPCext = ext(2)-ext(1)+1; % # samples per channel in extraction range
dat = datack;  % Copy input structure to output
extSamples = wavefmt.nChannels*nSPCext;
dat.Data = fread(datack.fid, [wavefmt.nChannels nSPCext], dtype);
% if cnt~=extSamples, dat='Error reading file.'; return; end
% Skip over trailing samples:
if(fseek(datack.fid, BytesPerSample * ...
         (SamplesPerChannel-ext(2))*wavefmt.nChannels, 0)==-1),
  dat='Error reading file.'; return;
end
```

```
  % Determine if a pad-byte is appended to data chunk,
  %   skipping over it if present:
  if rem(datack.Size,2), fseek(datack.fid,1,0); end
  % Rearrange data into a matrix with one channel per column:
  dat.Data = dat.Data';
  % Normalize data range: min will hit -1, max will not quite hit +1.
  if BytesPerSample==1,
    dat.Data = (dat.Data-128)/128;  % [-1,1)
  else
    dat.Data = dat.Data/32768;  % [-1,1)
  end
else
  % Unknown wave-format for data.
  dat='Unrecognized file format.';
end

return;

% end of wavread.m
```

V. BIBLIOGRAFÍA.

REFERENCIAS

[1] Freund, J. E., Walpole, R. E., *Estadística Matemática con Aplicaciones.*

[2] Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B., *Bayesian Data Analisis.*

[3] Haykin, S. S., *Communication Systems*, John Wiley and Sons.

[4] Johnson, D. H., and Dudgeon, D. E, *Array Signal Processing (Concepts and Techniques)*, Prentice-Hall Signal Processing Series.

[5] Oppenheim, A. V., and Schafer, R. W., *Digital Signal Processing*, Prentice-Hall.

[6] Oppenheim, A. V., and Schafer, R. W., *Discrete-Time Signal Processing*, Prentice-Hall.

[7] Oppenheim, A. V., *Applications of Digital Signal Processing*, Prentice-Hall.

[8] Proakis, J. G., and Manolakis, D. G., *Tratamiento Digital de Señales*, Prentice-Hall.

[9] Schaeaffer and McClave, *Probabilidad y Estadística para Ingeniería*, Grupo Editorial Iberoamérica.

[10] Sánchez, J. M., y Binefa, X., *Reconocimiento Automático de Anuncios de TV.* Artículo Internet.

[11] Zhu Liu, *Adaptative and Multimodal Approach To Multimedia Content Analysis.* Proyecto Internet.

[12] *Manual del Data Acquisition Toolbox 1.0,* Manual Internet (www.mathworks.com/products).