UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGON

"DESARROLLO DE UN SISTEMA DE SONIDO BIAURAL BASADO EN UNA COMPUTADORA PERSONAL"

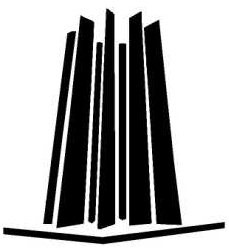
TESIS

PARA OBTENER EL TÍTULO DE:
INGENIERO MECÁNICO ELECTRICISTA
PRESENTA:

EDUARDO VILLEGAS ÁLVAREZ

DIRECTOR DE TESIS:
Dr. FELIPE ORDUÑA BUSTAMANTE

MÉXICO 2006



Agradecimientos,

A Dios, quien dio propósito a mi vida, y me enseñó que en su Nombre todo lo puedo. Me dio una familia que hasta hoy me ha apoyado, me dio unos amigos que en las buenas o en las malas están conmigo, una esposa que a pesar de que me conoce bien aún sigue a mi lado y mis hijos (as) quienes me motivan cada día a ser mejor persona y me inspiran aliento para darles lo mejor de mi.

A mis padres, Domingo Villegas Vargas. y Ma. Guadalupe Álvarez Valdez, que con todo sacrificio se han negado muchas veces así mismos para darme la oportunidad de llegar a estas alturas, me han enseñado valores y principios claves para el diario vivir. Quiero honrarlos con este trabajo de tesis aunque yo mismo sepa que merecen mucho más.

A mi hermana Elizabeth Villegas Álvarez, con quien crecí paralelamente disfrutando siempre buenos tiempos desde la niñez hasta ahora, que tenemos la responsabilidad de una familia.

A mi esposa, quien me dio el ánimo necesario en la última etapa de este trabajo y ha sido una persona de gran apoyo para mi, que junto con mis hijos es una fuente de inspiración para cada día superarme. Aclaro que por ahora solo tenemos a Edsa Janai Villegas Garrido y los demás a pesar de que aún no nacen, ya los quiero mucho y lucho por ellos.

A mi asesor de tesis el Dr. Felipe Orduña Bustamante, quien con toda paciencia y amistad me ha llevado de la mano durante el desarrollo de este trabajo, quiero mencionar que es todo un profesional y una persona muy valiosa y humilde. Mis respetos!!

A todos en general los quiero mucho y… ¡Muchas Gracias!!!

La paz os dejo, mi paz os doy; yo no os la doy como el mundo la da. No se turbe vuestro corazón, ni tenga miedo:

Jesucristo.
(Sn. Juan 14:27)

Índice general

Capítulo 1

Introducción.

1.1. Objetivo.

El objetivo de esta tesis es desarrollar un programa de filtrado rápido en tiempo real de señales de audio digital, que formará parte de un sistema de reproducción de sonido para dos altavoces basado en una computadora personal, desarrollado en el Laboratorio de Acústica y Vibraciones del CCADET-UNAM.

1.2. Justificación.

Existen equipos desarrollados para realizar reproducción de sonido biaural, sin embargo son equipos o aparatos de un muy alto costo, lo cual requiere altos recursos económicos para tenger acceso a este servicio. Lo que se pretende en esta tesis es desarrollar dicho sistema usando sólo una computadora personal y software de libre distribución, por lo que hace de este sistema, uno de facil acceso. El sistema desarrollado puede tener diferentes aplicaciones, pero la que queremos enfatizar es para uso médico, en pruebas de audiometría; dónde se evalúa la capacidad de escuchar sonidos.

1.3. Descripción general del tema de tesis.

La reproducción biaural intenta dar al oyente una sensación de ambientalización o sensación del espacio acústico. Es decir, que aunque el oyente no se encuentre físicamente en el lugar donde se realizan los eventos, tales como conciertos, conversaciones, etc. y mediante alguna técnica de grabación, pueda percibir los sonidos como si lo estuviera.

Se han propuesto varios métodos para satisfacer la necesidad de biauralidad, entre ellos estan los siguientes; grabación aplicando técnicas de microfoneo, reproducción usando un "muro de sonido", reproducción con sistemas de tres canales, sistemas cuadrafónicos matriciales, sistemas de sonido surround, etc. Estos métodos se explican adelante. En este trabajo se presenta un sistema de reproducción biaural usando elementos simples que en nuestra

actualidad son de fácil acceso, tales como una computadora personal, micrófonos, una fuente de sonido, altavoces, y software apropiado de libre distribución descrito posteriormente. El Sistema aquí descrito consiste en la creación de un programa en lenguaje C que recibe, procesa y reproduce audio en "tiempo real", usando una computadora personal.

Este trabajo de tesis es parte de un proyecto a largo plazo formado por diferentes etapas, el cual consiste en desarrollar un sistema de reproducción biaural y, en éste se desarrolló un componente del mencionado sitema. Dicho componente es la sección de *filtrado rápido*, el cual pretende entregar la salida de audio procesado a la mayor brevedad posible. El procesamiento de la señal de entrada se hace usando filtros digitales optimizados, el procedimiento de cálculo de estos filtros no es objeto de esta tesis, pero si requiere mayor información con respecto a este tema vea [1]. Estos filtros son usados para la demostración del filtrado rápido, eliminando el cross-talk.

En general, se documenta una técnica de procesamiento digital mediante el uso de una computadora personal, en la que se introducirá sonido proveniente de alguna fuente, tal como un radio FM o AM, reproductor de CD, etc. para que al procesarlo se obtenga un control óptimo de las señales en los oídos (eliminación de crosstalk).

Esta técnica, como se mencionó consiste en elaborar un programa en lenguaje C que hará el procesamiento sobre la plataforma Windows. Por lo que aquí se presenta el código fuente detallado en secciones, para una mejor explicación y comprensión. Las secciones o etapas son las siguientes: Manera en que recibe los datos el dispositivo de entrada que es la tarjeta de sonido, manera en que interpreta dichos datos el sistema operativo (Windows), método detallado de adecuación de la información para poder procesarla y así obtener la salida deseada, filtrado óptimo de la señal, método de filtrado en tiempo real (objetivo principal de ésta tesis), volviendo los datos procesados a su formato original, es decir adecuarlos en la manera necesaria para ser interpretados por el sistema operativo y después reproducirlos y enviarlos al dispositivo de salida de audio.

Una etapa importante que se realizó al final tiene que ver con mediciones del comportamiento del sistema, se anotaron los resultados y se compararon con los esperados mediante cálculo.

Este sistema fué probado en la cámara de transmisión ubicada en el Laboratorio de Acústica y Vibraciones del Centro de Ciencias Aplicadas y Desarrollo Tecnológico de la UNAM, bajo condiciones óptimas, obteniendo resultados satisfactorios.

1.4. Contenido de la tesis.

En el capitulo 1, se da una descripción general de lo que se encuentra documentado en esta tesis.

En el capitulo 2, se da una breve historia de los trabajos desarrollados por los pioneros de la reproducción biaural. Se describen los términos de convolución y correlación importantes para el filtrado digital de señales que se requiere, mostrando el algoritmo de filtrado rápido el cual se realiza mediante la transformada rápida de Fourier.

En el capitulo 3, se muestra la programación en lenguaje C de dicho algoritmo (filtrado rápido), dando detalles de la técnica de programación de audio en ambiente Windows con programas ejemplo, tales como `rec.c` y `sound.c` indicando también el software necesario para realizar el procesamiento que, como ya se mencionó, es de distribución libre. Los programas ejemplo servirán para ilustrar las dos fases principales del Sistema y con una adaptación de estos se creará el programa principal de ésta tesis `audio-filt.c`

En el capitulo 4, se hacen mediciones del Sistema comparando valores deseados, valores calculados y valores medidos, presentando las gráficas correspondientes.

En el capitulo 5, se ofrecen conclusiones, evaluando resultados y se hacen recomendaciones para investigaciones y/o desarrollos futuros.

Capítulo 2

Sistemas de reproducción biaural.

2.1. Sistemas de reproducción biaural.

2.1.1. Antecedentes.

A pesar de que muchos de los principios referentes al sonido estereofónico fueron desarrollados con gran entusiamo desde principios de 1930's, en nuestros días aún persiste una idea errónea de lo que significa la palabra "estéreo". Generalmente se asocia con la reproducción de sonido con dos altavoces, sin embargo viene del Griego ‘‘stereos’’, que significa sólido o tridimensional.

La asociación de sistemas de reproducción de sonido con sistemas de dos canales vino en los años 1950's, debido a que así lo imponían las limitaciones tecnológicas de las grabaciones fonográficas en disco de acetato, pues tenían sólo dos "paredes" en cada surco donde se codificaba la información y cada pared representaba un canal.

La estereofonía comenzó con el trabajo del Ingeniero Eléctrico Alan Dower Bumlein en el Reino Unido (1903-1942), ver Figura (2.1), quien se propuso a desarrollar lo que él llamó "Binaural Sound" o sonido biaural, esto después de ir al cine con su esposa y oír que el sonido sólo provenía de un lugar, no importando la ubicación de la "fuente" en la pantalla. Bumlein aseguraba que era posible ubicar el sonido dentro de un rango de ángulos usando una combinación apropiada de diferencias de retardo y nivel, asi que inició sus investigaciones.

Su trabajo se enfocó en la reproducción perfecta del campo sonoro a cada oído del sujeto

Figura 2.1: Alan D. Bumlein

y en el desarrollo de técnicas de "microfoneo"que permitieran la grabación de diferencias de amplitud y fase necesarias para una reproducción estéreo (tridimensional). Otros investigadores como Fletcher, Steinberg y Snow de los Laboratorios Bell en U.S.A tomaron otro camino en sus investigaciones. Ellos consideraron implementar un "muro de sonido" en el cual teóricamente, un número infinito de micrófonos era usado para reproducir todo un campo sonoro a través de un número infinito de altavoces. A pesar de que esto constituía un resultado teórico interesante, los investigadores de los Laboratorios Bell se dieron cuenta que para implementaciones caseras se requería un número más pequeño de canales.

Por lo tanto, mostraron después un sistema de tres canales constituido por canales izquierdo, derecho y central en un plano horizontal que podía representar la lateralización y profundidad del campo sonoro deseado con una exactitud aceptable.

El primer sistema estereofónico de tres canales como el descrito anteriormente fué utilizado como demostración en 1934 con la Orquesta de Filadelfia haciendo su presentación remota para un público presente en Washigton DC, a través de líneas telefónicas de banda ancha.

2.1.2. Sistema Cuadrafónico Matricial.

Mientras los métodos "estereofónicos" pueden ser herramientas poderosas en la reproducción de atributos de espacio de los campos sonoros, estos quedan cortos para la reproducción real en tres dimensiones. El sistema cuadrafónico intentó superar estas limitaciones capturando y transmitiendo información del campo sonoro directo y reverberante.

Para reproducir los cuatro canales requeridos en grabaciones cuadrafónicas con medios de almacenamiento y de transmisión de dos canales, era necesario desarrollar un esquema apropiado de codificación y decodificación. Fueron propuestos varios esquemas basados en matriz 4: 2: 4 cod/deco. Los sistemas cuadrafónicos fueron capaces de reproducir imparcialmente imágenes de sonido exactas en las áreas de enfrente y de atrás en el plano horizontal, pero mostraron serias limitaciones al intentar reproducir imágenes de sonido a los lados del sujeto.

Los experimentos mostraron que esto era una limitación asociada tanto con la síntesis del campo sonoro usando cuatro canales como mecanismos de psicoacústica humana. Estas limitaciones técnicas, más la presencia de dos formatos que se hacían competencia mutua en el mercado, contribuyeron a la desaparición de los sistemas cuadrafónicos.

2.1.3. Sonido envolvente Multicanal Surround.

A principios de los 1950's el primer formato de sonido multicanal fué desarrollado por 20th Century Fox. La combinación de formatos de pantalla-grande, el de 35mm en Cinema Scope así como el de 70 mm 6-tracks Todd-AO con sonido multicanal, fué la respuesta de la industria fílmica ante la creciente popularización de la televisión. El sonido fílmico estereofónico fué reproducido por tres altavoces frontales, pero estos nuevos formatos incluían un canal monofónico adicional que se reproducía en dos altavoces los cuales se ubicaban

detrás de la audiencia y se conocía como el canal de los efectos.

Este canal aumentaba la sensación de espacio para la audiencia, pero también tenía serias limitaciones tecnológicas. Ya que los oyentes que se sentaban cerca de línea central (on-center) con respecto a los altavoces, percibían una sensación de interiorización del sonido (inside-head), que da un efecto de localización similar a la imagen estéreo que reproducen los audífonos, como si el sonido se produjera del interior de la cabeza de la persona. Los oyentes que se sentaban lejos de la línea central (off-center) ubicaban la dirección en la que estaba el altavoz de efectos más cercano, cumpliéndose lo que dicta la ley de los primeros arrivos de principio de onda, de este modo destruía la sensación envolvente deseada. La solución para estos problemas fue encontrada al introducir un segundo canal reproducido sobre un arreglo de altavoces ubicados lateralmente a lo largo del teatro, para crear una mayor difusión del campo sonoro.

A mediados de los 1970's fué introducida una nueva tecnología de sonido por los Laboratorios Dolby, llamada Dolby Stereo. Estuvo basada en tecnología óptica que había sido usada en filmes de los 1930's y ésta superaba los problemas asociados con grabaciones magnéticas multitrack. Dolby desarrolló un método matricial para codificar cuatro canales (izquierdo, central, derecho y un monoaural sorround) dentro de dos canales usando una técnica derivada de métodos matriciales usados en los sistemas cuadrafónicos, pero también asegurando la compatibilidad retrospectiva mono y estéreo.

En 1992, se hicieron mejoras por Dolby Stereo Digital (SR-D). Este formato eliminó la codificación y decodificación basada en matrices y provió cinco canales separados (izquierdo, central, derecho y dos sorround independientes izquierdo y derecho) en una configuración conocida como stereo surround. Un sexto canal que mejoraba el sonido a bajas frecuencias (LFE, low-frequence-enhance) fué introducido al sistema para aumentar el rango dinámico (head-room) y prevenir la sobrecarga de las bocinas principales por frecuencias bajas. El ancho de banda del canal LFE está limitado entre 0 y 120 Hz, que es un régimen de frecuencia que está fuera del rango de localización humana para los oyentes en un cuarto reverberante, de esta manera se simplificaron los requerimentos de la ubicación del subwoofer usado para la reproducción del canal LFE.

Recientes avances en la compresión digital de audio y almacenamiento óptico (por ej: grabación de CD's,HD, etc.) han hecho posible reproducir seis canales separados de audio en un formato accesible centrado en el esquema de compresión Dolby AC-3. Con exitantes formatos nuevos como los discos de video digital (DVD) donde pronto el número de canales podría incrementar a diez o más. A pesar de que varios sistemas accesibles en el mercado son capaces de reproducir más de dos canales, la mayoría de los usuarios (particularmente aquellos con sistemas de cómputo de escritorio) encontrarían un uso impráctico a la utilización de múltiples altavoces.

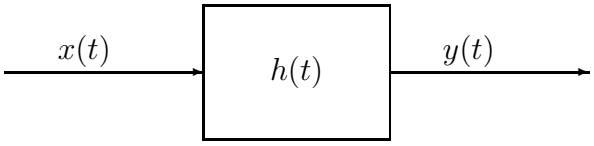


Figura 2.2: Relación entrada-salida en un sitema lineal.

2.2. Filtrado digital de señales.

2.2.1. Convolución y correlación en tiempo continuo.

Un sistema lineal de tiempo continuo transforma una señal de entrada $x(t)$ en la señal de salida $y(t)$ en forma lineal. En la Figura (2.2) se muestra un sistema lineal, donde se tiene una señal de entrada $x(t)$, la señal de salida $y(t)$ y la respuesta a impulso del sistema $h(t)$.

Convolución.

Definición: En matemáticas y en particular en análisis funcional, una convolución es un operador matemático que transforma dos funciones f y g en una tercera función que en cierto sentido representa la magnitud en la que se superponen, f y una versión trasladada e invertida de g. Una convolución es un tipo de cierto promedio en movimiento, como se puede observar si una de las funciones la tomamos como la función característica de un intervalo.

Volviendo a nuestro sistema lineal, matemáticamente, la señal de salida está dada por la siguiente ecuación integral, que define la operación de convolución entre las señales $h(t)$ y $x(t)$:

$$y(t) = \int_0^\infty h(\tau)x(t-\tau)d\tau \tag{2.1}$$

en donde la variable de integración τ se puede interpretar como un retardo de la señal de entrada $x(t)$. La señal de salida al tiempo t es la combinación (integral) de valores anteriores de la señal de entrada $x(t-\tau)$, ponderados por el valor de la respuesta a impulso $h(\tau)$, con retardos τ desde 0 hasta ∞. Esta operación se muestra esquemáticamente en la Figura (2.3).

La operación de convolución se denota tradicionalmente mediante el símbolo $*$, de manera que la señal de salida de un sistema lineal se puede expresar de la siguiente manera:

$$y(t) = (h * x)(t) = (x * h)(t); \tag{2.2}$$

es decir, que la salida $y(t)$ es la convolución de $h(t)$ con $x(t)$. La convolución tiene la propiedad de ser conmutativa.

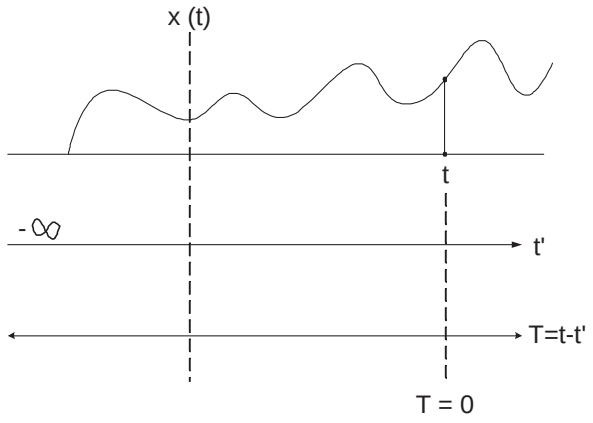


Figura 2.3: Esquema de la operación de convolución.

Autocorrelación.

Cabe mencionar primeramente que hay correlación entre dos variables cuando éstas cambian de tal modo que los valores que toma una de ellas son, hasta cierto punto predecibles a partir de los que toma la otra, y en el caso de la autocorrelación es una correlación con respecto a ella misma.

En forma similar a la convolución, se puede definir la integral de correlación de la señal de entrada, o la integral de autocorrelación, de la siguiente manera:

$$\phi_{xx}(\tau) \equiv \int_{-\infty}^{\infty} x(t)x(t+\tau)dt; \tag{2.3}$$

con el cambio de variable $t \rightarrow t - \tau$, esto es equivalente a

$$\phi_{xx}(\tau) = \int_{-\infty}^{\infty} x(t-\tau)x(t)dt. \tag{2.4}$$

La autocorrelación tiene la propiedad de simetría par

$$\phi_{xx}(\tau) = \phi_{xx}(-\tau) \tag{2.5}$$

La correlación definida aquí, es válida para señales no aleatorias de energía limitada (que tienen un principio y un fin en el tiempo), por ejemplo, para una señal transitoria. Las definiciones correspondientes para señales aleatorias, estacionarias, etc. no se expondrán en este trabajo.

Correlación.

Al definir la función de correlación entre dos señales diferentes por ejemplo, entre la entrada y la salida de un sistema lineal, se ve de la siguiente manera:

$$\phi_{xy}(\tau) \;\; = \;\; \int_{-\infty}^{\infty} x(t)y(t+\tau)dt, \tag{2.6}$$

$$= \int_{-\infty}^{\infty} x(t-\tau)y(t)dt, \tag{2.7}$$

que resulta no ser conmutativa, sino que tiene la propiedad de que al variar el orden entrada-salida o salida-entrada se invierte el signo del retardo, intercambiando x (entrada) y y (salida):

$$\phi_{yx}(\tau) = \phi_{xy}(-\tau) \tag{2.8}$$

En un sistema lineal se cumple el llamado *teorema de correlación entrada-salida*, que establece que la correlación entrada-salida es la respuesta a impulso $h(t)$ convolucionada con la función de autocorrelación de la señal de entrada ϕ_{xx}, como se muestra matemáticamente a continuación:

$$\phi_{xy}(\tau) = (h * \phi_{xx})(\tau) = (\phi_{xx} * h)(\tau); \tag{2.9}$$

este teorema se puede comprobar sustituyendo la ecuación (2.1) en la ecuación (2.6): agregando un retardo a la señal de salida:

$$\begin{aligned}
\phi_{xy}(\tau) &= \int_{-\infty}^{\infty} x(t) \left[\int_{0}^{\infty} h(\tau')x(t+\tau-\tau')d\tau' \right] dt, && (2.10)\\
&= \int_{-0}^{\infty} h(\tau') \left[\int_{-\infty}^{\infty} x(t)x(t+\tau-\tau')dt \right] d\tau', && (2.11)\\
&= \int_{-0}^{\infty} h(\tau')\phi_{xx}(\tau-\tau')d\tau', && (2.12)\\
&= (h * \phi_{xx})(\tau). && (2.13)
\end{aligned}$$

Relación entre convolución y correlación.

La correlación es una operación similar a la convolución con la diferencia de que en la correlación, no hay que "reflejar" (o invertir) una de las señales.

A continuación tenemos la integral de la correlación entrada-salida:

$$\phi_{xy}(\tau) = \int_{-\infty}^{\infty} x(t)y(t+\tau)dt; \tag{2.14}$$

si sustituimos en ésta t por $-t$ para obtener el reverso de $x[t]$, nos queda de la siguiente manera:

$$\phi_{xy}(\tau) = \int_{-\infty}^{\infty} x(-t)y(\tau-t)dt. \tag{2.15}$$

Ahora, definamos el reverso de la señal $x(t)$ como:

$$\tilde{x}(t) \equiv x(-t); \tag{2.16}$$

por lo tanto la correlación toma la siguiente forma matemática:

$$\begin{aligned}
\phi_{xy}[\tau] &= \int_{-\infty}^{\infty} \tilde{x}(t)y(\tau-t)dt, && (2.17)\\
&= (\tilde{x} * y)(\tau); && (2.18)
\end{aligned}$$

y concluimos que la correlación ϕ_{xy} es equivalente al reverso de x ($\tilde{x}$) convolucionado con y.

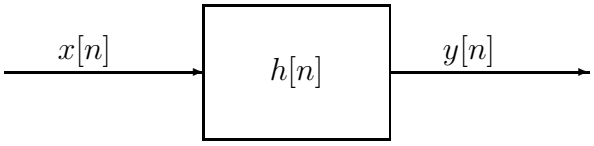


Figura 2.4: Relación entrada-salida en un sitema lineal de tiempo discreto.

2.2.2. Convolución y correlación en tiempo discreto.

En el caso de sistemas y señales de tiempo discreto se tienen relaciones análogas al caso de tiempo continuo.

La Figura (2.4) ilustra el caso de un sistema lineal de tiempo discreto, donde la señal de entrada $x(n)$ es transformada en la señal de salida $y(n)$ de manera lineal.

Convolución.

La señal de salida es representada matemáticamente por la siguiente suma que define la operación de convolución discreta entre las señales $h(n)$ y $x(n)$:

$$y[n] = \sum_{m=0}^{\infty} h[m]x[n-m] \equiv (h * x)[n], \tag{2.19}$$

$$y[n] = \sum_{m=-\infty}^{n} x[m]h[n-m] \equiv (x * h)[n]; \tag{2.20}$$

en donde m se considera como un retardo de la señal de entrada $x(n)$. Por lo que la señal de salida en el tiempo n es una combinación (suma) de valores anteriores de la señal de entrada $x(n-m)$, ponderados por el valor de la respuesta a impulso del sistema $h(n)$, con retardos m desde 0 hasta ∞.

La señal de salida $y[n]$ es la convolución de $h[n]$ y $x[n]$

$$y[n] = (h * x)[n] = (x * h)[n]; \tag{2.21}$$

o bien, debido a que la convolución tiene la propiedad de ser conmutativa, la señal de salida $y[n]$ es también la convolución de las señales $x[n]$ y $h[n]$.

Autocorrelación discreta.

Como en el caso continuo, aquí también de manera similar a la convolución, se puede definir la suma de correlación de la señal de entrada que también se conoce como autocorrelación:

$$\phi_{xx}[m] = \sum_{n=-\infty}^{\infty} x[n]x[n+m] \tag{2.22}$$

Como se mencionó, la correlación tiene la propiedad de simetría, es decir no importa el signo de m:

$$\phi_{xx}[m] = \phi_{xx}[-m]. \tag{2.23}$$

Correlación discreta.

Definimos la función de correlación discreta de la señal entrada-salida mediante la siguiente expresión:

$$\phi_{xy}[n] = \sum_{n=-\infty}^{\infty} x[n]y[n+m] \tag{2.24}$$

la correlación no es conmutativa, pero tiene la peculiaridad de que la autocorrelación de la señal de entrada es igual al inverso de la correlación salida-entrada.

$$\phi_{xx}[n] = \phi_{yx}[-n] \tag{2.25}$$

Sustituyendo la Ecuación (2.19) en la Ecuación (2.24) tenemos:

$$\phi_{xy}[n] = \sum_{n=-\infty}^{\infty} x[n] \sum_{m'=0}^{\infty} h[m'] \cdot x[n+m-m'], \tag{2.26}$$

$$\phi_{xy}[n] = \sum_{m'=0}^{\infty} h[m'] \sum_{n=-\infty}^{\infty} x[n] \cdot x[n+m-m']; \tag{2.27}$$

resultando la siguiente ecuación:

$$\phi_{xy}[m] = \sum_{m'=0}^{\infty} h[n']\phi_{xx}[m-m'] \tag{2.28}$$

para concluir con el *teorema de correlación entrada-salida*:

$$\phi_{xy}[m] = (h * \phi_{xx}[m]). \tag{2.29}$$

Este teorema de correlación entrada-salida para los sistemas lineales discretos, establece lo siguiente: La correlación discreta de entrada-salida es igual a la convolución de la respuesta impulso del sistema con la función de autocorrelación de la señal de entrada.

Relación entre convolución y correlación.

La suma de la correlación discreta entrada-salida:

$$\phi_{xy}[m] = \sum_{n=-\infty}^{\infty} x[n]y[n+m]; \tag{2.30}$$

también se puede efectuar invirtiendo el orden del índice n pues el orden de los sumandos no altera la suma; es decir, se puede realizar la sustitución $n \to -n$, quedando como sigue:

$$\phi_{xy}[m] = \sum_{n=-\infty}^{\infty} x[-n]y[m-n]. \tag{2.31}$$

Si definimos el reverso de la señal $x[n]$ como

$$\tilde{x}[n] \equiv x[-n]; \tag{2.32}$$

entonces la correlación toma la forma

$$\begin{aligned} \phi_{xy}[m] &= \sum_{n=-\infty}^{\infty} \tilde{x}[n]y[m-n], & (2.33) \\ &= (\tilde{x} * y)[m]. & (2.34) \end{aligned}$$

Por lo que la correlación ϕ_{xy} es equivalente a la convolución de $\tilde{x}$ (reverso de x) con y.

2.3. Diseño de filtros óptimos.

En esta sección se presenta el método de cálculo de filtros óptimos que puede aplicarse a la compensación (o ecualización) de sistemas de reproducción de sonido. A pesar de que este no es el tema de esta tesis, sí es pertinente mencionarlo, debido a que este es uno de los métodos que pueden aplicarse para calcular los filtros que en esta tesis se suponen especificados de antemano.

El cálculo de filtros óptimos se basa en una técnica matemática desarrollada por el investigador estadounidense Norbert Wiener, que a continuación se explicará de manera general. La técnica emplea un filtro que modifica la señal de entrada a un sistema, con el fin de que este produzca una señal deseada especificada de antemano. Se define una señal de error como la diferencia entre la señal deseada y la señal de salida del sistema. El método determina el filtro óptimo según el criterio de minimizar el valor cuadrático medio de la señal de error.

En el diagrama de la figura (2.5) se tiene un sistema físico, que se supone lineal e invariante en el tiempo, la respuesta deseada del sistema, y el filtro que prealimenta el sistema; en donde se identifican los siguientes elementos:

G: Sistema físico con repuesta a impulso $g[n]$, de longitud L.

H: Filtro no recursivo con repuesta a impulso $h[n]$, de longitud N.

$r[n]$: Señal de entrada (por ejemplo: una señal grabada).

$q[n]$: Señal de alimentación del sistema físico (entrada a la fuente).

$p[n]$: Señal de salida (reproducida en un micrófono).

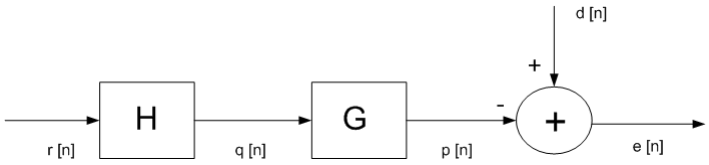


Figura 2.5: Diagrama diseño de filtros Óptimos.

$d[n]$: Señal deseada.

$e[n]$: Señal de error.

El objetivo es que, dadas la señal de entrada $r[n]$, la señal deseada $d[n]$ y la respuesta a impulso del sistema $g[n]$, se determine la respuesta a impulso del filtro $h[n]$, de manera que el valor cuadrático medio de la señal de error $e[n]$ sea mínimo.

Se define la siguiente función de costo:

$$J = \mathcal{E}\{e^2[n]\} \tag{2.35}$$

en donde $\mathcal{E}$ es el operador de valor esperado o valor medio, y la función de costo J depende de los coeficientes del filtro $h[n]$. A partir de la teoría del cálculo diferencial de varias variables, la función de costo J alcanza su valor mínimo cuando $h[n]$ satisface la siguiente ecuación:

$$\partial J/\partial h[n] = 0, \tag{2.36}$$

con $0 \leq m < N$. Esta condición determinará los coeficientes del filtro óptimo $h[n]$.

2.3.1. Relaciones entrada-salida.

La señal de salida $p[n]$ es la señal reproducida y se puede expresar como la convolución de la respuesta a impulso del sistema con la señal de alimentación, como sigue:

$$p[n] = g * q[n], \tag{2.37}$$

$$p[n] = \sum_{m=0}^{L-1} g[m] \cdot q[n-m]. \tag{2.38}$$

Por su parte, la señal de alimentación del sistema $q[n]$ es la convolución de la respuesta a impulso del filtro con la señal de entrada, de la siguiente manera:

$$q[n] = h * r[n], \tag{2.39}$$

$$q[n] = \sum_{m=0}^{N-1} h[m] \cdot r[n-m]. \tag{2.40}$$

Finalmente, la señal deseada $d[n]$ puede representarse como la convolución de una respuesta impulso deseada $a[n]$, de longitud M, con la señal de entrada:

$$d[n] = (a * r)[n]. \tag{2.41}$$

La señal de error se define como sigue:

$$\begin{aligned} e[n] &= d[n] - p[n], && (2.42)\\ e[n] &= d[n](-g * q)[n], && (2.43)\\ e[n] &= d[n](-g * h * q)[n], && (2.44)\\ e[n] &= d[n](-h * g * r)[n], && (2.45)\\ e[n] &= d[n](-h * f)[n]; && (2.46) \end{aligned}$$

en donde, luego de aplicar la propiedad conmutativa de la convolución, se introduce la señal $f[n] = g * r[n]$, que representa la señal de salida del sistema no compensado (no ecualizado). Finalmente, recordando que la convolución discreta de una señal con otra se representa como una sumatoria, la funcion de error queda de la siguiente manera:

$$e[n] = d[n] - \sum_{m=0}^{N-1} h[m] \cdot f[n-m]. \tag{2.47}$$

Por lo tanto, se puede sustituir esta expresión en la función de costo, como sigue:

$$\begin{aligned} J &= \mathcal{E}\{e^2[n]\}, && (2.48)\\ &= \mathcal{E}\{e[n] \cdot e[n]\}, && (2.49)\\ &= \mathcal{E}\{(d[n] - \sum_{m=0}^{N-1} h[m] \cdot f[n-m]) \cdot (d[n] - \sum_{l=0}^{N-1} h[l] \cdot f[n-l])\}, && (2.50)\\ &= \mathcal{E}\{d^2[n]\} - 2\sum_{m=0}^{N-1} h[m] \cdot \mathcal{E}\{f[n-m] \cdot d[n]\} \\ &\quad + \sum_{m=0}^{N-1}\sum_{l=0}^{N-1} h[m]h[l] \cdot \mathcal{E}\{f[n-m] \cdot d[n-l]\}. && (2.51) \end{aligned}$$

Ahora, recordamos que la función de correlación (estadística) entre las señales estacionarias $x[n]$ y $y[n]$ se define, de dos maneras equivalentes, como:

$$\begin{aligned} \phi_{xy}[m] &= \mathcal{E}\{x[n] \cdot y[n+m]\}, && (2.52)\\ &= \mathcal{E}\{x[n-m] \cdot y[n]\}. && (2.53) \end{aligned}$$

Utilizando la segunda forma en la ecuación (2.51) obtenemos:

$$J = \phi_{dd}[0] - 2\sum_{m=0}^{N-1} h[m]\phi_{fd}[m] + \sum_{m=0}^{N-1}\sum_{l=0}^{N-1} h[m]h[l]\phi_{ff}[m-l]; \tag{2.54}$$

calculando la derivada parcial

$$\partial J/\partial h[m] = -2\phi_{fd}[n] + 2\sum_{m=0}^{N-1} h[m]\phi_{ff}[m-n]; \tag{2.55}$$

igualando a cero y factorizando, encontramos que los coeficientes óptimos de $h[n]$ satisfacen la siguiente ecuación:

$$\sum_{m=0}^{N-1} h[m]\phi_{ff}[m-n] = \phi_{fd}[n], \tag{2.56}$$

con $0 \leq n < N-1$. Esto representa un sistema de ecuaciones lineales simultáneas, cuya solución proporciona los coeficientes del filtro óptimo $h[n]$.

En el caso especial en el que la señal de entrada $r[n]$ es ruido blanco, entonces la función de autocorrelación corresponde a la función delta

$$\phi_{rr}[m] = \delta[m]; \tag{2.57}$$

es decir, $\phi_{rr}[0] = 1$, $r[n]$ auto-correlaciona con corrimiento $m = 0$, pero $\phi_{rr}(m) = 0$ para $m \neq 0$. Recordando que $f[n] = g * r[n]$ con respuesta a impulso $g[n]$ de longitud L, y $d[n] = a * r[n]$ con respuesta a impulso deseada $a[n]$ de longitud M. Entonces en el caso en que $r[n]$ es ruido blanco, las funciones de auto-correlación $\phi_{ff}[m]$ y correlación $\phi_{fd}[m]$, de las señales $f[n]$ y $d[n]$, se convierten en sumas de correlación de las respuestas a impulso $g[n]$ y $a[n]$, de la siguiente manera:

$$\begin{aligned}
\phi_{ff}[m] &= \sum_{i=0}^{L-1} g[i]g[i+m], && (2.58)\\
&= \phi_{gg}[m], && (2.59)\\
\phi_{fd}[m] &= \sum_{i=0}^{L-1} g[i]a[i+m], && (2.60)\\
&= \phi_{ga}[m]. && (2.61)
\end{aligned}$$

El sistema de ecuaciones cambia a:

$$\sum_{m=0}^{N-1} h[m]\phi_{gg}[m-n] = \phi_{ga}[n]. \tag{2.62}$$

Esta se puede representar en notación matricial. Los elementos de $h[m]$ y $\phi_{ga}[n]$ se pueden poner en forma vectorial:

$$\mathbf{h} = \begin{bmatrix} h[0] \\ h[1] \\ h[2] \\ \vdots \\ h[N-1] \end{bmatrix}, \tag{2.63}$$

$$\mathbf{r_{ga}} = \begin{bmatrix} \phi_{ga}[0] \\ \phi_{ga}[1] \\ \phi_{ga}[2] \\ \vdots \\ \phi_{ga}[N-1] \end{bmatrix}, \tag{2.64}$$

y los elementos de $\phi_{gg}[m-n]$ se pueden poner en una misma matriz:

$$\mathbf{R_{gg}} = \begin{bmatrix} \phi_{gg}[0] & \phi_{gg}[1] & \phi_{gg}[2] & \cdots & \phi_{gg}[N-1] \\ \phi_{gg}[1] & \phi_{gg}[0] & \phi_{gg}[1] & \cdots & \phi_{gg}[N-2] \\ \phi_{gg}[2] & \phi_{gg}[1] & \phi_{gg}[0] & \cdots & \phi_{gg}[N-3] \\ \vdots & \vdots & \vdots & & \\ \phi_{gg}[N-1] & \phi_{gg}[N-2] & \phi_{gg}[N-3] & \cdots & \phi_{gg}[0] \end{bmatrix}. \tag{2.65}$$

Por lo tanto, los coeficientes del filtro se obtienen resolviendo el sistema de ecuaciones:

$$\mathbf{R_{gg}} \cdot \mathbf{h} = \mathbf{r_{ga}}. \tag{2.66}$$

La matriz $\mathbf{R_{gg}}$ tiene en cada diagonal los valores $\phi_{gg}[0]$, $\phi_{gg}[1]$, $\phi_{gg}[2]$, etc., a partir de la diagonal principal; esta estructura se conoce como matriz Toeplitz. Este tipo de matrices admite el uso de métodos numéricos eficientes para resolver el sistema de ecuaciones (algoritmo de Levinson-Durbin).

2.4. Representación matricial de la convolución discreta.

Como se vió con anterioridad, la convolución de señales de tiempo discreto se define como una suma de productos. Ahora, se mostrará que la convolución se puede representar también de forma matricial.

2.4.1. Convolución lineal (no periódica).

También se mencionó, que la operación de convolución tiene la propiedad de ser conmutativa y gracias a ésta podemos representar dicha operación con las siguientes expresiones.

$$y[n] = \sum_{m=0}^{\infty} h[m]x[n-m], \tag{2.67}$$

$$y[n] = \sum_{m=-\infty}^{n} x[m]h[n-m]. \tag{2.68}$$

para nuestros fines utilizamos la segunda ecuación y asignando valores de $n = 0$, $n = 1$, $n = 2$... etc.

$$y[0] = \sum_{m=-\infty}^{0} h_{0-m}x_m = \cdots + h_2x_{-2} + h_1x_{-1} + h_0x_0 + \cdots, \tag{2.69}$$

$$y[1] = \sum_{m=-\infty}^{1} h_{1-m}x_m = \cdots + h_2x_{-1} + h_1x_0 + h_0x_1 + \cdots, \tag{2.70}$$

$$y[2] = \sum_{m=-\infty}^{2} h_{2-m}x_m = \cdots + h_2x_0 + h_1x_1 + h_0x_2 + \cdots; \tag{2.71}$$

representando la ecuación anterior en forma matricial, suponiendo que la respuesta a impulso del filtro tiene una longitud de $L_h = 6$ coeficientes, se tiene la siguiente matriz, que será la matriz de convolución:

$$\begin{bmatrix}
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
\cdots & h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
\cdots & h_4 & h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
\cdots & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 & \cdots \\
\cdots & 0 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 & \cdots \\
\cdots & 0 & 0 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & 0 & 0 & \cdots \\
\cdots & 0 & 0 & 0 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & 0 & \cdots \\
\cdots & 0 & 0 & 0 & 0 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & \cdots \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots &
\end{bmatrix}
\begin{bmatrix} \vdots \\ x_{-3} \\ x_{-2} \\ x_{-1} \\ x_{-0} \\ x_1 \\ x_2 \\ x_3 \\ \vdots \end{bmatrix}
=
\begin{bmatrix} \vdots \\ y_{-3} \\ y_{-2} \\ y_{-1} \\ y_{-0} \\ y_1 \\ y_2 \\ y_3 \\ \vdots \end{bmatrix} \tag{2.72}$$

La matriz de convolución es una matriz Toeplitz,(diagonales iguales); si h es finita, se obtiene una matriz de banda.

Convolución de señales finitas.

Ejemplo de convolución lineal finita con señales discretas de longitud $L = 4$:

$$\mathbf{h} = [h_0h_1h_2h_3], \mathbf{x} = [x_0x_1x_2x_3] \tag{2.73}$$

$$\begin{bmatrix}
h_0 & 0 & 0 & 0 \\
h_1 & h_0 & 0 & 0 \\
h_2 & h_1 & h_0 & 0 \\
h_3 & h_2 & h_1 & h_0 \\
0 & h_3 & h_2 & h_1 \\
0 & 0 & h_3 & h_2 \\
0 & 0 & 0 & h_3
\end{bmatrix}
\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}
=
\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} \tag{2.74}$$

Este ejemplo ilustra el resultado general de que la convolución de dos secuencias de longitudes N_1 y N_2 es $N = N_1 + N_2 - 1$.

A continuación un ejemplo de convolución con una señal de entrada de longitud $L_x = 8$ muestras y la respuesta a impulso de un filtro de longitud $L_h = 4$ coeficientes:

$$\mathbf{h} = [h_0 h_1 h_2 h_3], \mathbf{x} = [x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7] \tag{2.75}$$

$$\begin{bmatrix} h_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_1 & h_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 & 0 \\ h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 \\ 0 & h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 \\ 0 & 0 & h_3 & h_2 & h_1 & h_0 & 0 & 0 \\ 0 & 0 & 0 & h_3 & h_2 & h_1 & h_0 & 0 \\ 0 & 0 & 0 & 0 & h_3 & h_2 & h_1 & h_0 \\ 0 & 0 & 0 & 0 & 0 & h_3 & h_2 & h_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_3 & h_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & h_3 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \end{bmatrix} \tag{2.76}$$

El vector resultante (señal de salida) tiene una longitud de $L_y = L_x + L_h - 1$ muestras.

2.5. Transformada de Fourier y convolución.

2.5.1. Distintos tipos de tranformaciones de Fourier.

Existen diferentes tipos de transformaciones de Fourier que a continuación se describen.

Tranformada integral de Fourier.

La transformada integral de Fourier se aplica en el caso de señales de tiempo continuo; es de dominio infinito y continua en los dominios del tiempo y frecuencia. Se define de la siguiente manera: ver figura (2.6)

$$G(f) = \int_{-\infty}^{\infty} g(t) e^{-j2\pi ft} dt, \text{ (frecuencia)} \tag{2.77}$$

Para el dominio del tiempo tenemos la siguiente ecuación, ver figura (2.7)

$$g(t) = \int_{-\infty}^{\infty} G(f) e^{-j2\pi ft} df; \text{ (tiempo)} \tag{2.78}$$

Series de Fourier.

Las series de Fourier describen señales periódicas como una combinación de señales armónicas (sinusoides). A continuación representamos una señal periódica en el dominio del tiempo y discreta en el dominio de la frecuencia.

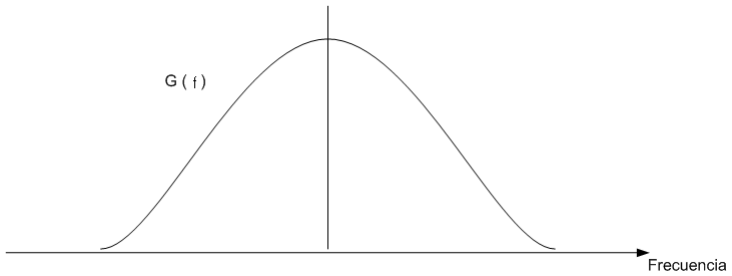


Figura 2.6: Transformada integral de Fourier en frecuencia

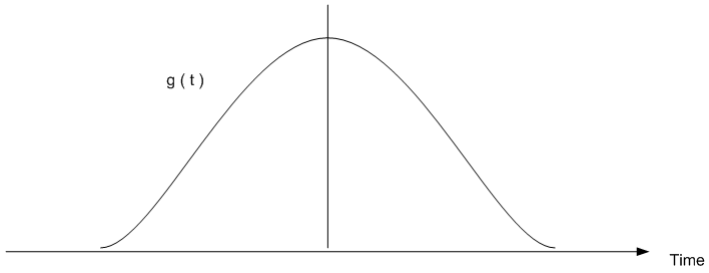


Figura 2.7: Transformada integral de Fourier en el tiempo

$$G(f_k) = \frac{1}{T}\int_{-T/2}^{T/2} g(t)e^{-j2\pi f_k t}dt, \quad \text{(frecuencia)} \tag{2.79}$$

ver figura (2.8)

$$g(t) = \sum_{k=-\infty}^{\infty} G(f_k)e^{j2\pi f_k t}; \qquad \text{(tiempo)} \tag{2.80}$$

ver figura (2.9)

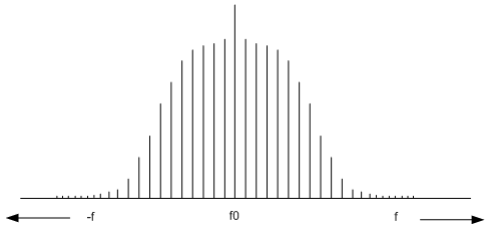


Figura 2.8: Serie de Fourier en frecuencia

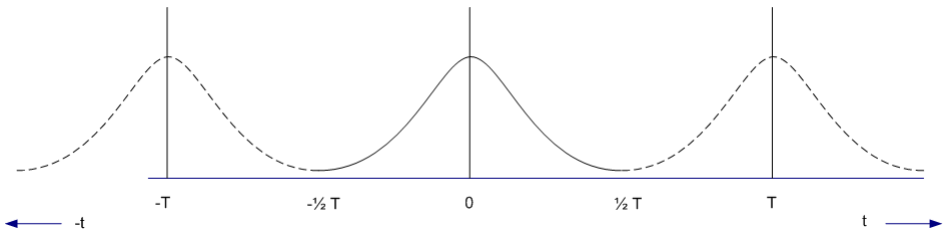


Figura 2.9: Serie de Fourier en el tiempo

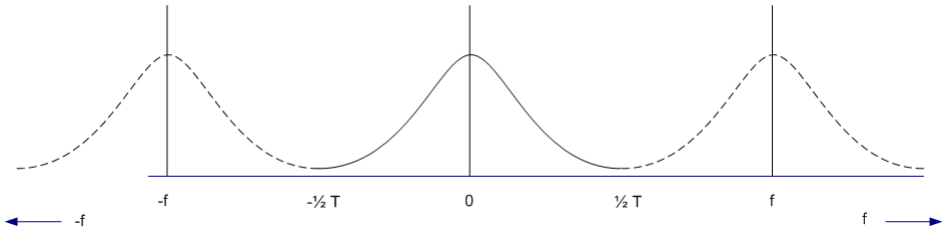


Figura 2.10: Función muestreada en el tiempo, representada en el dominio de la frecuencia

Función muestreada.

En este caso la transformación es discreta en el dominio del tiempo y periódica en el dominio de la frecuencia.

$$G(f) = \sum_{n=-\infty}^{\infty} g(t_n) e^{j2\pi f_n}, \tag{2.81}$$

ver figura (2.10)

$$g(t_n) = \frac{1}{f_s} \int_{-f_s/2}^{f_s/2} G(f) e^{j2\pi f t_n} \tag{2.82}$$

ver figura (2.11)

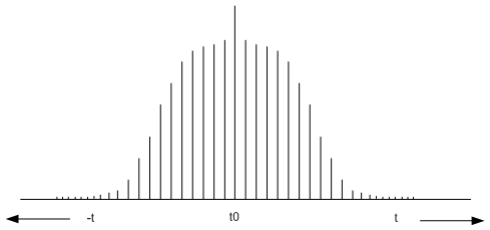


Figura 2.11: Función muestreada en el tiempo

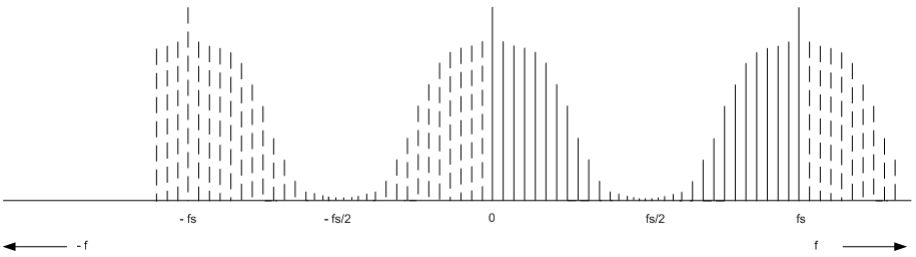


Figura 2.12: Transformada discreta en el dominio de la frecuencia

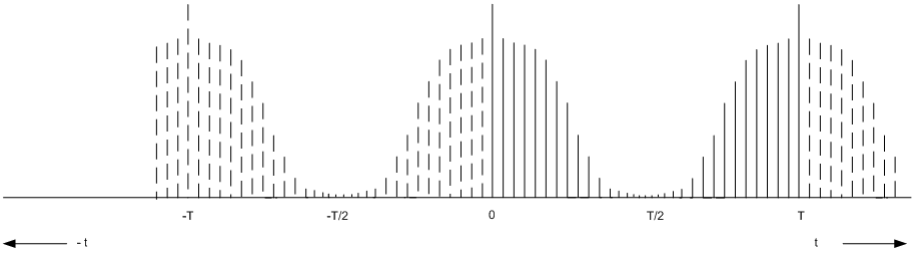


Figura 2.13: Transformada discreta en el dominio del tiempo

Transformada discreta de Fourier.

Discreta y periódica en ambos dominios (tiempo y frecuencia).

$$G(f_k) = \frac{1}{N} \sum_{n=0}^{N-1} g(t_n)\, e^{-j\frac{2\pi nk}{N}} \tag{2.83}$$

ver figura (2.12)

$$g(t_n) = \sum_{k=0}^{N-1} G(f_k) e^{j\frac{2\pi nk}{N}}. \tag{2.84}$$

ver figura (2.13)

2.5.2. Transformada de Fourier de la convolución y de la correlación.

Teorema de convolución.

Sean $X(\omega)$, $Y(\omega)$ y $H(\omega)$ las transformadas de Fourier respectivas de las funciones $x(t)$, $y(t)$ y $h(t)$

$$X(\omega) \;=\; \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt, \tag{2.85}$$

$$Y(\omega) = \int_{-\infty}^{\infty} y(t)e^{-j\omega t}dt, \tag{2.86}$$

$$H(\omega) = \int_{-\infty}^{\infty} h(t)e^{-j\omega t}dt \tag{2.87}$$

Si $y(t)$ es la señal de salida de un sistema lineal con señal de entrada $x(t)$ y función de respuesta a impulso $h(t)$, se tiene que

$$y(t) = (h * x)(t), \tag{2.88}$$

$$y(t) = \int_{0}^{\infty} h(\tau)x(t-\tau)d\tau. \tag{2.89}$$

El teorema de convolución establece que

$$Y(\omega) = H(\omega) \cdot X(\omega); \tag{2.90}$$

es decir, en el dominio de la frecuencia la operación de convolución equivale a una multiplicación.

Transformada de Fourier de la Correlación.

Recordando que la relación entre convolución y correlación (es decir, que la correlación de $x(t)$ con $y(t)$ es equivalente a la convolución de la señal invertida (reverso) $\tilde{x}(t) = x(-t)$ con $y(t)$), podemos concluir que la transformada de Fourier de la correlación es

$$\Phi(\omega) = \tilde{X}(\omega)Y(\omega); \tag{2.91}$$

en donde $\tilde{X}(\omega)$ es la transformada de Fourier de $\tilde{x}(t)$.

Si $x(t)$ es una señal que toma valores reales, entonces se puede simplificar el resultado, ya que

$$\tilde{X}(\omega) = \int_{-\infty}^{\infty} x(-t)e^{-j\omega t}dt; \tag{2.92}$$

haciendo el cambio de variable $t \to -t$ tenemos

$$\tilde{X}(\omega) = \int_{-\infty}^{\infty} x(t)e^{+j\omega t}dt, \tag{2.93}$$

$$\tilde{X}(\omega) = \left[\int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt\right]^* \tag{2.94}$$

por lo que

$$\tilde{X}(\omega) = X^*(\omega). \tag{2.95}$$

Por lo tanto, si $x(t)$ es real, entonces

$$\Phi_{xy}(\omega) = X^*(\omega) \cdot Y(\omega); \tag{2.96}$$

es decir, la transformada de Fourier de la correlación de $x(t)$ con $y(t)$, es equivalente al producto de las transformadas de Fourier $X^*(\omega)$ ($X(\omega)$ conjugada) con $Y(\omega)$.

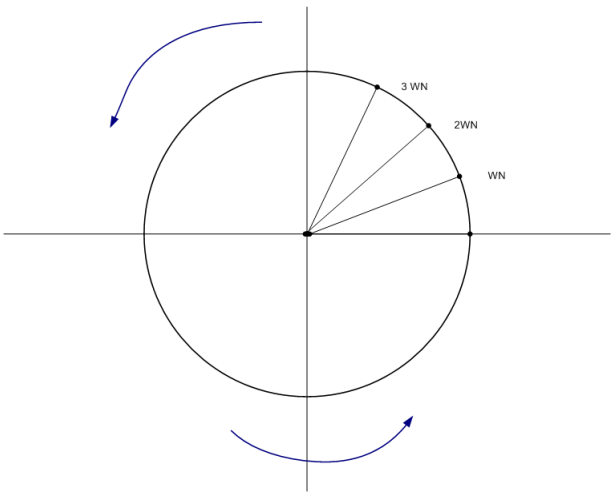


Figura 2.14: Plano Complejo

2.5.3. Transformada discreta de Fourier.

La transformada discreta de Fourier (o DFT, por sus siglas en inglés) está definida por

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{-kn}, \tag{2.97}$$

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k W_N^{(+nk)} \tag{2.98}$$

en donde

$$W_N \equiv e^{+2\pi j/N}, \tag{2.99}$$

$$W_N^{-kn} = e^{-2\pi jkn/N}, \tag{2.100}$$

$$W_N^{+nk} = e^{+2\pi jnk/N}. \tag{2.101}$$

ver figura (2.14)

Teorema de convolución discreta y periódica.

Ahora en el caso discreto tenemos que $X[k]$, $Y[k]$ y $H[k]$ son transformadas discretas respectivas de las señales $x[n]$, $y[n]$ y $h[n]$

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-2\pi ink/N}, \tag{2.102}$$

$$Y[k] = \sum_{n=0}^{N-1} y[n] e^{-2\pi ink/N}, \tag{2.103}$$

$$H[k] = \sum_{n=0}^{N-1} h[n]e^{-2\pi ink/N} \tag{2.104}$$

Si $y[n]$ es la señal de salida de un sistema lineal con señal de entrada $x[n]$ y función de respuesta a impulso $h[m]$, se tiene que

$$y[n] = \sum_{n=0}^{N-1} h[m]x[n-m] \tag{2.105}$$

$$y[n] = (h * x)[n] \tag{2.106}$$

concluyendo que la sumatoria en el dominio del tiempo, equivale a una multiplicación de las misma funciones en el dominio de la frecuencia, si y sólo si las señales son discretas y de periodo N.

$$Y[k] = H[k] \cdot X[k] \tag{2.107}$$

Teorema de correlación discreta y periódica.

Igual que en el caso continuo, la transformada discreta de Fourier de una señal periódica invertida (en reversa) es

$$\tilde{X}[k] = X^*[k]; \tag{2.108}$$

con $x[n]$ real.

Por lo tanto también se cumple el mismo teorema para la correlación, que establece que la correlación de $x[n]$ con $y[n]$ es equivalente a la convolución de la señal invertida $\tilde{x} = x[-n]$. Nótese que dicho teorema cumple sólo para señales discretas de periodo N.

$$\phi_{xy}[m] = \sum_{n=0}^{N-1} x[n]y[n+m], \tag{2.109}$$

$$\phi_{xy}[m] = (\tilde{X} * Y)[m], \tag{2.110}$$

$$\Phi_{xy}[k] = X^*[k] \cdot Y[k]; \tag{2.111}$$

con $x[n]$ real (que en nuestro caso simpre será así la señal de entrada).

2.5.4. Representación matricial de la DFT.

Una vez habiendo explicado la DFT con ecuaciones, ahora nos es necesario para el procesamiento digital de señales representarla de manera matricial.

Definimos primero los vectores $\mathbf{x}$ y $\mathbf{X}$ con elementos

$$(\mathbf{x})_n \equiv x[n], \tag{2.112}$$

$$(\mathbf{X})_k \equiv X[k]. \tag{2.113}$$

Nota: Se escribe con negritas cuando se habla de vectores o matrices.

También definimos la matriz $\mathbf{W}$ con elementos

$$(\mathbf{W})_{kn} \equiv W_N^{-nk}. \tag{2.114}$$

La definición de DFT es como sigue, sin embargo considerando lo anterior, es decir considerando que se habla de matrices y vectores tenemos que la representación matricial de la DFT se escribe de la siguiente manera

$$\begin{aligned} (\mathbf{X})_k &= \sum_{n=0}^{N-1} (\mathbf{W})_{kn} (\mathbf{x})_n, & (2.115)\\ \mathbf{X} &= \mathbf{W} \cdot \mathbf{x}. & (2.116) \end{aligned}$$

Similarmente, la IDFT (Transformada Discreta de Fourier Inversa) se puede escribir

$$\begin{aligned} (\mathbf{x})_n &= \sum_{k=0}^{N-1} (\mathbf{W}^{-1})_{nk} (\mathbf{X})_k, & (2.117)\\ \mathbf{x} &= \mathbf{W}^{-1} \cdot \mathbf{X}; & (2.118) \end{aligned}$$

en donde la matriz $\mathbf{W}^{-1}$ tiene elementos

$$(\mathbf{W}^{-1})_{kn} \equiv W_N^{+kn}. \tag{2.119}$$

Nótese que debido a las propiedades de la DFT, la matriz $\mathbf{W}^{-1}$ es en efecto la inversa de la matriz $\mathbf{W}$.

Veamos un ejemplo en donde $N = 4$ y $W_4 = e^{2\pi j/4}$. Cabe señalar que el primer índice de W numera los renglones y el segundo numera las columnas.

$$\mathbf{W} = \begin{bmatrix} W_4^{-0\cdot 0} & W_4^{-0\cdot 1} & W_4^{-0\cdot 2} & W_4^{-0\cdot 3} \\ W_4^{-1\cdot 0} & W_4^{-1\cdot 1} & W_4^{-1\cdot 2} & W_4^{-1\cdot 3} \\ W_4^{-2\cdot 0} & W_4^{-2\cdot 1} & W_4^{-2\cdot 2} & W_4^{-2\cdot 3} \\ W_4^{-3\cdot 0} & W_4^{-3\cdot 1} & W_4^{-3\cdot 2} & W_4^{-3\cdot 3} \end{bmatrix} \tag{2.120}$$

si $W_4 = e^{2\pi j/4}$ entonces $W_4 = j$ ver figura (2.15)

$$\begin{aligned} \mathbf{W} &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}, & (2.121)\\ \mathbf{W}^{-1} &= \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix}. & (2.122) \end{aligned}$$

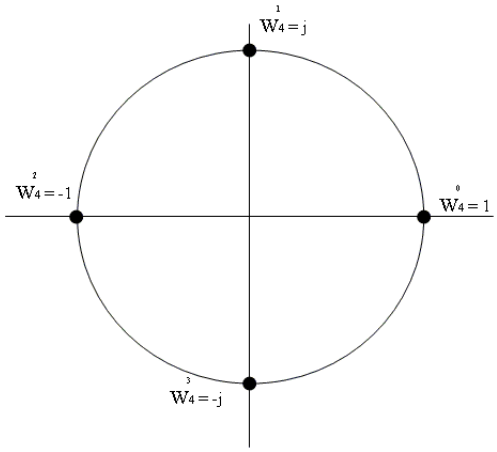


Figura 2.15: Círculo unitario

Representando la transformada directa tenemos la siguiente ecuación de matrices:

$$\mathbf{Wx} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, \tag{2.123}$$

$$= \begin{bmatrix} X_0 + X_1 + X_2 + X_3 \\ X_0 - jX_1 - X_2 + jX_3 \\ X_0 - X_1 + X_2 - X_3 \\ X_0 + jX_1 - X_2 - jX_3 \end{bmatrix} \equiv \mathbf{X} \tag{2.124}$$

Obteniendo la transformada inversa, que es aquella que multiplicada por su matriz original es igual a la matriz identidad, se tiene lo siguiente:

$$\mathbf{W^{-1}X} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix}, \tag{2.125}$$

$$= \frac{1}{4} \begin{bmatrix} X_0 + X_1 + X_2 + X_3 \\ X_0 + jX_1 - X_2 - jX_3 \\ X_0 - X_1 + X_2 - X_3 \\ X_0 - jX_1 - X_2 + jX_3 \end{bmatrix} = \mathbf{x}. \tag{2.126}$$

2.6. Algoritmo de filtrado rápido.

Una vez entendida la representación matricial de las transformaciones discretas de Fourier, pasemos a la representacion matricial del filtrado rápido.

2.6.1. Representación matricial de la convolución circular.

En el caso de señales discretas de periodo N, la convolución circular (periódica o cíclica) se define por la siguiente ecuación:

$$y_n = \sum_{m=0}^{N-1} h_m x_{(n-m \bmod N)}, \tag{2.127}$$

$$y_n = \sum_{m=0}^{N-1} h_{(n-m \bmod N)} x_m; \tag{2.128}$$

donde y_n es la señal de salida. Por ejemplo, si tomamos la segunda de las ecuaciones anteriores con $N = 4$:

$$y_0 = h_0x_0 + h_3x_1 + h_2x_2 + h_1x_3, \tag{2.129}$$

$$y_1 = h_1x_0 + h_0x_1 + h_3x_2 + h_2x_3, \tag{2.130}$$

$$y_2 = h_2x_0 + h_1x_1 + h_0x_2 + h_3x_3, \tag{2.131}$$

$$y_3 = h_3x_0 + h_2x_1 + h_1x_2 + h_0x_3 \tag{2.132}$$

representada en forma matricial

$$\begin{bmatrix} h_0 & h_3 & h_2 & h_1 \\ h_1 & h_0 & h_3 & h_2 \\ h_2 & h_1 & h_0 & h_3 \\ h_3 & h_2 & h_1 & h_0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{2.133}$$

La matriz de convolución circular es una matriz *circulante* (hacia abajo y hacia la derecha), lo que también le da una estructura tipo *Toeplitz*. Aquí se supone que todas las señales: x, y, h, son de periodo N.

2.6.2. Teorema de convolución cíclica (discreta).

Si X_k, Y_k y M_k son las transformadas discretas de Fourier de las secuencias periódicas x_n, y_n y h_n entonces

$$y_n = h_n * x_n, \text{ (convolución)} \tag{2.134}$$

$$Y_k = M_k \cdot X_k, \text{ (multiplicación)}; \tag{2.135}$$

es decir, la transformada discreta de la convolución en el dominio del tiempo es equivalente a una multiplicación en el dominio de la frecuencia. Este resultado es la base para varios algoritmos rápidos para calcular convolución, correlación y filtrado rápido.

2.6.3. Algoritmo rápido de convolución lineal finita.

Tomamos como ejemplo las secuencias h_n, x_n de tamaño $N = 4$. En caso de que las longitudes sean distintas, N es el tamaño de la más grande. Consideremos la convolución cíclica extendida con ceros al doble de tamaño $2N = 8$:

$$\begin{bmatrix} h_0 & 0 & 0 & 0 & 0 & h_3 & h_2 & h_1 \\ h_1 & h_0 & 0 & 0 & 0 & 0 & h_3 & h_2 \\ h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 & h_3 \\ h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 \\ 0 & h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 \\ 0 & 0 & h_3 & h_2 & h_1 & h_0 & 0 & 0 \\ 0 & 0 & 0 & h_3 & h_2 & h_1 & h_0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ 0 \end{bmatrix} \tag{2.136}$$

En este caso, la convolución se encuentra en las primeras 7 muestras del resultado. Esto conduce al siguiente algoritmo.

2.6.4. Algoritmo de convolución finita.

Para calcular la convolución de dos secuencias finitas $y_n = h_n * x_n$:

1. Extender h_n y x_n con ceros hasta un tamaño mayor o igual que $L_h + L_x - 1$ (suma de las longitudes menos uno).

2. Calcular

$$H_k = \text{DFT}(h_n), \tag{2.137}$$
$$Y_k = \text{DFT}(X_n). \tag{2.138}$$

3. Calcular

$$Y_n = \text{IDFT}(H_k \cdot X_k) \tag{2.139}$$

4. El resultado se encuentra almacenado en los primeros $L_h + L_x - 1$ elementos de Y_n.

2.6.5. Correlación finita.

De manera similar, se tiene el siguiente resultado para la correlación:

$$\phi_{xy}(n) = \text{corr}(X_n, y_n) \tag{2.140}$$
$$\phi_{xy}(n) = \text{IDFT}(X_k^* \cdot Y_k) \tag{2.141}$$

El algoritmo rápido procede sobre los mismos pasos que el algoritmo de convolución lineal finita.

2.6.6. Algoritmo de convolución (lineal) rápida, no finita.

Aquí se supone que la señal x_n es de duración infinita pero el filtro h_n es de longitud finita. Tomamos nuevamente el caso en que h_n es de longitud $N = 4$. Consideremos la convolución cíclica de h_n extendida con ceros hasta tamaño $2N = 8$, con un bloque de muestras de la señal x_n del mismo tamaño (sin extender con ceros):

$$\left[\begin{array}{cccc:cccc} h_0 & 0 & 0 & 0 & 0 & h_3 & h_2 & h_1 \\ h_1 & h_0 & 0 & 0 & 0 & 0 & h_3 & h_2 \\ h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 & h_3 \\ h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 \\ 0 & 0 & h_3 & h_2 & h_1 & h_0 & 0 & 0 \\ 0 & 0 & 0 & h_3 & h_2 & h_1 & h_0 & 0 \end{array}\right] \left[\begin{array}{c} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{array}\right] = \left[\begin{array}{c} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \cdots \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{array}\right] \tag{2.142}$$

En este caso, la mitad superior (primeras N muestras) del vector resultante está contaminada por los efectos de la convolución circular (excepto por la muestra y_3), pero la mitad inferior (últimas N muestras) sí corresponden de manera correcta con las muestras de la convolución lineal de h_n con x_n. Esto conduce al siguiente algoritmo de filtrado rápido.

2.6.7. Método de "guardar y traslapar".

Para calcular bloques consecutivos de muestras de la convolución $y_n = h_n * x_n$ se realiza lo siguiente:

1. Tomar N de tamaño mayor o igual que L_h (lonitud de h_n) y extender h_n con ceros hasta tamaño $2N$.

2. Calcular la transformada discreta de Fourier $H_k = \mathrm{DFT}(h_n)$ (de tamaño $2N$).

3. Obtener un bloque de N muestras nuevas de la señal de entrada $X_n^{(\mathrm{nuevas})}$ y formar un vector extendido (de tamaño $2N$):

$$X_n = \left(\begin{array}{c} X_n^{(\mathrm{antiguas})} \\ X_n^{(\mathrm{nuevas})} \end{array} \right). \tag{2.143}$$

4. Calcular

$$X_k = DFT(X_n). \tag{2.144}$$

5. Calcular

$$Y_n^{(ext)} = IDFT(H_k \cdot X_k) \tag{2.145}$$

6. Distinguir en el resultado las muestras correctas e incorrectas (contaminadas por convolución circular):

$$Y_n = \begin{pmatrix} Y_n^{(incorrectas)} \\ Y_n^{(correctas)} \end{pmatrix}. \tag{2.146}$$

7. Actualizar los bloques de señal de entrada:

$$X_n^{(\text{antiguas})} = X_n^{(\text{nuevas})}. \tag{2.147}$$

8. Ir al paso 3.

De esta manera, se puede continuar indefinidamente, de manera que, por cada bloque de N muestras "nuevas" de la señal de entrada, se obtiene un bloque de N muestras "correctas" de la señal de salida. En el siguiente capítulo se desarrolla un programa para realizar este algoritmo, operando en tiempo real, con señales de audio recibidas y enviadas por medio de una tarjeta de sonido en una computadora personal PC.

Capítulo 3

Programación del algoritmo de filtrado rápido.

3.1. Programación de audio en ambiente Windows.

En esta sección se describen las técnicas de programación de audio en el sistema operativo Windows, primero se explica qué es una tarjeta de sonido, sus características principales, formatos de datos que soporta, cómo se configura y cómo se programa. También se explica qué es la Interfaz de Programación de Aplicaciones (API) de Windows; particularmente, las funciones más importantes para acceder a la tarjeta de sonido. Para dejar claro este proceso se presentan como ejemplo dos programas en lenguaje `C`, uno llamado `rec.c` que tiene como objetivo ilustrar la manera en que se puede programar la tarjeta de sonido de la PC para grabar sonido (como señal de audio en forma de onda, o *waveform audio*), y otro llamado `snd.c` que ilustra la forma en que se reproduce sonido (como señal de audio).

3.1.1. Tarjeta de sonido.

Iniciemos mencionando la relación original de las computadoras personales (PC) con el audio, y es que las PC's no fueron diseñadas en un principio para emitir o procesar sonido de alta calidad, la excepción eran algunos pitidos emitidos con un pequeño altavoz interno, por ejemplo al iniciar el ordenador, y en general, para llamar la atención del usuario, al ocurrir algún error, etc. Sin embargo, debido a la importancia del sonido para ciertas aplicaciones, se introdujeron dispositivos dedicados, como *tarjetas de sonido*, cuyo uso se generalizó con el advenimiento de los sistemas multimedios.

La tarjeta de sonido es un dispositivo que opera principalmente con información de dos tipos: audio digital y datos musicales. Originalmente se usó comúnmente para reproducir sonido; sin embargo, con el transcurso del tiempo se mejoró su desempeño, habilitándolas también para capturar audio. Una tarjeta de sonido típica cuenta con conectores de salida para altavoces y audífonos; algunas tarjetas más sofisticadas tienen salidas de audio (salida analógica, o *Line Out*), salida digital y MIDI. Sus conectores de entrada son: micrófono (mo-

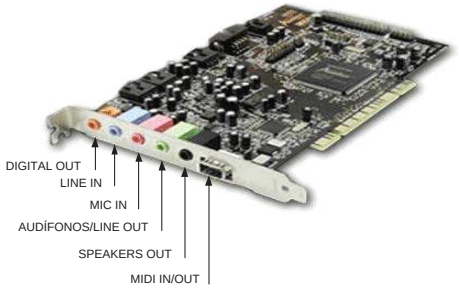


Figura 3.1: Tarjeta de sonido típica de una computadora personal.

nofónico), entrada analógica (o *Line In*), señal de CD (conectada internamente), entradas digitales y MIDI. Véase la Figura (3.1).

Las principales funciones de una tarjeta de sonido son:

1. Reproducción de sonido (mono/estéreo)
2. Grabación de sonido(mono/estéreo).
3. Entrada y salida de información MIDI.

Puede reproducir audio a partir de una sola fuente, o de la mezcla de varias fuentes, tales como: señal del micrófono (*Mic*), entrada de línea analógica (*Line In*), señal de CD, señal digital interna (*WAV*) y sintetizador MIDI interno. En el proceso de grabación, una tarjeta típica normalmente sólo puede operar, alternativamente, con una de las siguientes fuentes de audio: señal de micrófono (*Mic*), entrada de línea analógica (*Line In*), señal de CD, señal digital (*WAV*) y entrada MIDI.

3.1.2. Interfaz de Programación de Aplicaciones (API) de Windows.

El desarrollo de programas de aplicación para el sistema operativo Windows está soportado por una gran cantidad de herramientas y recursos informáticos. Un papel central lo desempeña la Interfaz de Programación de Aplicaciones, o brevemente: API (*Application Programming Interface*) de Windows [9]. La API de Windows es una enorme colección de funciones que residen en el propio sistema operativo Windows, listas para ser usadas por cualquier programa de aplicación. Estas funciones están almacenadas en varias bibiotecas de enlace dinámico (*dynamic-link libraries*, o DLL). Estas bibliotecas contienen funciones de la API que tienen que ver con el manejo de memoria, manejo de procesos, control de aspectos de la interfaz de usuario, puertos de comunicación, control de operaciones gráficas, de sonido, etc.

Los programas de Windows se enlazan dinámicamente a estas DLLs; es decir, las funciones de la API no están incluidas explícitamente en el archivo ejecutable de cada programa de Windows, sino que cada programa invoca las funciones de la API para que sean ejecutadas de manera centralizada por el propio sistema operativo.

Cuando un programa de Windows es cargado en la memoria, Windows lee la información almacenada en el programa. Esa información incluye el nombre de las funciones que el programa usa, y las DLLs donde residen esas funciones. Cuando Windows encuentra esa información en el programa, cargará las DLLs y localizará las funciones indicadas en el programa, de manera que las llamadas a estas transfieran el control a la función correcta.

Debido a esto, para programar la tarjeta de sonido es necesario consultar la documentación de la Interfaz de Programación de Aplicaciones (API) de Windows, que está disponible en los discos compactos distribuidos con Microsoft Visual Studio, o bien en Internet en la siguiente dirección:

```
http://msdn.microsoft.com/library/default.asp?url=
/library/en-us/multimed/htm/_win32_waveform_audio_reference.asp
```

Esta página también se puede visitar en la dirección `http://msdn.microsoft.com` y siguiendo los enlaces: *Library*, *Win32 and COM Development*, *Platform SDK*, *Platform SDK Documentation Contents*, *Windows Multimedia*, *SDK Documentation*, *Windows Multimedia*, *Multimedia Audio*, *Waveform Audio*. *Waveform Audio Reference*.

Por otro lado es necesario mencionar que Windows soporta varios dispositivos de entrada que pueden ser de audio, de video, de texto, etc. La parte de audio se opera con tarjetas de sonido, a través de las siguientes interfaces de programación del API de Windows: MIDI (*Musical Instrument Digital Interface*), MCI (*Media Control Interface*) y *Waveform Audio* (audio en forma de onda). Los programas `rec.c` y `snd.c`, que se presentan más adelante, usan la interfaz de programación de forma de onda (*Waveform Audio API*, o *Wave-API*) para ilustrar las técnicas de grabación y reproducción de sonido. Por lo tanto, toda vez que se mencione el dispositivo de entrada o salida de audio, estaremos refiriéndonos, en lo sucesivo, a la tarjeta de sonido en relación con su entrada o salida de audio en forma de onda.

3.1.3. Funciones y código fuente usado en los programas.

El código fuente de los programas que se presentan más adelante en este capítulo, hace referencia a elementos de programación (estructuras de datos, funciones, constantes, etc.) que, por un lado, provienen directamente del Wave-API de Windows, y por el otro, representa código desarrollado en forma propia. En este último caso, se trata de aportaciones de un equipo de trabajo encabezado por el asesor de esta tesis: Dr. Felipe Orduña Bustamante, y que incluye como participantes al autor de esta tesis: Eduardo Villegas Álvarez, y a los estudiantes de la Maestría en Ingeniería Eléctrica (opción Instrumentación) de la UNAM: José Antonio Arredondo Garza, Juan Ignacio Cervantes Cruz, Javier de Jesús Fonseca Madrigal, y Hugo Enrique Lazcano Hernández. La mayor parte del código propio fue desarrollado durante la impartición del curso: Procesamiento Digital de Audio, en el semestre 2003-2 de

dicha maestría, bajo la titularidad del Dr. Felipe Orduña. Todos los elementos de código propio se presentan y se detallan íntegros más adelante. Los elementos de programación que provienen del Wave-API de Windows se explican parcialmente, según sea necesario o importante, y el resto pueden identificarse por el contexto en el que se usan.

3.1.4. Herramientas de programación en lenguaje `C`.

Ahora bien, se pueden escribir dichos programas en diferentes lenguajes. Nosotros utilizamos el lenguaje `C` por ser de uso más o menos general y por tener los recursos necesarios (desempeño en tiempo de ejecución etc.) para la elaboración de estos. De igual manera, podemos recurrir a diferentes compiladores de lenguaje `C`, como Visual C, Borland C, etc. Sin embargo, en nuestro caso haremos uso del compilador GNU `gcc` del proyecto *Cygwin* [10], que es un compilador de software libre.

`gcc` es un compilador para programas escritos en lenguaje `C`; sus siglas significan GNU Compiler Collection. El proyecto GNU (acrónimo recursivo para "GNU No es Unix") tiene el objetivo de desarrollar un sistema operativo tipo Unix completo, con la peculiaridad de ser software libre. GNU comenzó en 1984 y se financía por la Free Software Foundation. [11]

3.1.5. Configuración de la tarjeta de sonido.

Para utilizar una determinada tarjeta de audio de la PC, es necesario primeramente habilitarla como dispositivo de entrada de audio del sistema. Para esto se debe configurar en el ícono de *Multimedia* o *Dispositivos de sonido y audio* ubicado en el *Panel de Control* de la PC, donde se tiene acceso a la configuración de los dispositivos de sonido, video, MIDI, reproductor de CD y otros dispositivos multimedia. De igual forma, se debe habilitar esa misma tarjeta, u otra en su caso, como el dispositivo de salida.

Una vez especificados los dipositivos de entrada y salida de sonido que serán utilizados por el sistema, es necesario que el dispositivo de entrada se configure para especificar el origen (entrada) y el destino (salida) de las señales de audio a procesar. Esto se hace desde la consola de audio de Windows, véase la Figura (3.2), seleccionando la entrada de audio analógica *Line In*. Se procede de manera similar con el dispositivo de salida, Figura (3.3), dejando activa la salida de onda y ajustándole una ganancia adecuada.

En las siguientes secciones de este cápitulo se describen dos técnicas para manipular audio en forma de onda, la primera es para grabación (señal de entrada) y la segunda para reproducción (señal de salida) de sonido, a través del dispositivo de audio de la computadora. Ambas técnicas se ilustran por medio de dos programas en lenguaje `C`: 1) `snd.c` y 2) `rec.c`. Más adelante, se presenta el programa `audio-filt.c`, que es el objeto principal de esta tesis; ahí se detalla la programación del filtrado rápido. Estos programas utilizan algunas de las funciones del API de Windows para manipular las señales de audio que entran y salen de la tarjeta de sonido.

Consola de Windows para
Grabación de Audio.

Figura 3.2: Consola para control de grabación de audio en Windows.

Consola de Audio de Windows para
Reproducción.

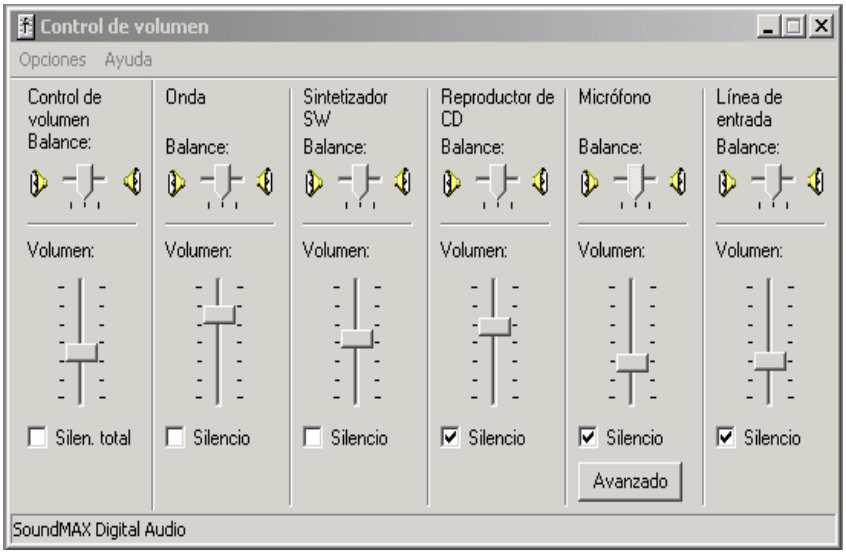


Figura 3.3: Consola para control de reproducción de audio en Windows.

3.2. Programación del proceso de grabación de sonido.

Para realizar la grabación pensemos que lo que se va a grabar en la memoria del ordenador son dígitos entregados por la tarjeta de sonido que representan la señal de audio analógico, por lo tanto los pasos que se deben seguir son los siguientes:

1. Reservar memoria para acumular los bloques de audio recibidos.
2. Grabarlos en un formato con parámetros que se explicarán adelante.
3. Guardarlos en un archivo.

La secuencia de un programa que graba audio obedece a la siguiente rutina: Primeramente recibe bloques de audio, los procesa y después los envia al lugar indicado en memoria. Para ejemplificar lo anterior, presentamos un programa que graba y promedia diez bloques consecutivos de entrada de audio.

3.2.1. Programa de grabación `rec.c`.

Windows permite indicar el formato de audio a través de los miembros de una estructura de datos determinada de antemano. A continuación tenemos una porción de programa en lenguaje `C` donde se muestra el formato que se le asignará al bloque de audio a través de variables globales:

```
unsigned int sampling_rate = 44100; /* muestras por segundo */
unsigned int channels = 1;          /* señal monofónica */
unsigned int bytes_per_sample = 2;  /* audio de 16-bits */
unsigned int samples = 65536;       /* muestras por bloque */
volatile int recording = 0;         /* estado de grabación: 1 o 0 */
unsigned int num_blocks = 10;       /* bloques por grabar */
unsigned int blocks_recorded = 0;   /* bloques grabados */
```

En el fragmento anterior de código del programa `rec.c` se declaran las variables que se utilizarán para dar formato a cada bloque de audio grabado y para controlar el número de bloques grabados. Los valores asignados indican que la grabación se hará con un ritmo de muestreo de 44100 muestras por segundo, el formato será monofónico, que el tamaño de cada muestra es de 2 bytes (16 bits), se tomarán bloques de 65536 muestras, la duración de la grabación es de 10 bloques; también se utiliza un indicador del estado de grabación (encendido o apagado) y un contador de bloques grabados.

3.2.2. Función `record_wave_data`.

La función `record_wave_data` engloba el procedimiento de grabación, desde especificar el formato de audio, abrir el dispositivo de entrada de audio, preparar estructuras para alojar los datos de audio, hasta reponer y cerrar el dispositivo de audio.

En el siguiente fragmento de código se verifica si el formato de audio puede ser soportado por el dipositivo de entrada del sistema. También hace el llenado del formato requerido para el bloque de datos de la estructura nativa de Windows, asignando los valores de número de canales, frecuencia de muestreo y algunos parámetros requeridos como el porcentaje de bytes por segundo, alineación de bloques y el número de bytes por muestra, que son necesarios para la administración de bloques que realiza internamente el sistema operativo.

```
void
record_wave_data (void)
{
  HWAVEIN wave_in; // Referencia al dispositivo de entrada de forma de onda.
  PCMWAVEFORMAT wave_fmt; // Datos del formato de forma de onda.

  wave_fmt.wf.wFormatTag = WAVE_FORMAT_PCM;
  wave_fmt.wf.nChannels = channels;
  wave_fmt.wf.nSamplesPerSec = sampling_rate;
  wave_fmt.wf.nAvgBytesPerSec = channels * bytes_per_sample * sampling_rate;
  wave_fmt.wf.nBlockAlign = channels * bytes_per_sample;
  wave_fmt.wBitsPerSample = 8 * bytes_per_sample;
```

Una vez establecidas las variables de formato, se hace la configuración y gestión de acceso al dispositivo de entrada de audio. Para ello se utiliza la función `WaveInOpen` que abre el dispositivo de entrada de audio para grabar.

```
  if (waveInOpen (&wave_in,
          WAVE_MAPPER,
          (LPWAVEFORMATEX) &wave_fmt,
          (DWORD) wave_in_func,
          (DWORD) NULL,
          CALLBACK_FUNCTION) != MMSYSERR_NOERROR)
    {
      fputs ("waveInOpen: Error.\n", stderr);
      exit (EXIT_FAILURE);
    }
```

Resumiendo los primeros dos pasos del programa `rec.c`: primero se llenó una estructura llamada `wave_fmt` con el formato adecuado para el archivo de audio en cada uno de sus elementos (o miembros); en seguida se mandó llamar la función `waveInOpen`, donde a su vez se especifica la función `CALLBACK` que se explicará mas adelante.

No habiendo ningún error al abrir el dispositivo de entrada de audio, se manda llamar la función `add_new_wave_in_hdr(wave_in, wave_data)` que crea estructuras o templetes en blanco donde se puedan alojar los datos de audio asignándoles su encabezado correspondiente. Obsérvese el siguiente fragmento de código donde se visualiza el proceso de añadir dos estructuras para ser llenadas por la función `record_wave_data`:

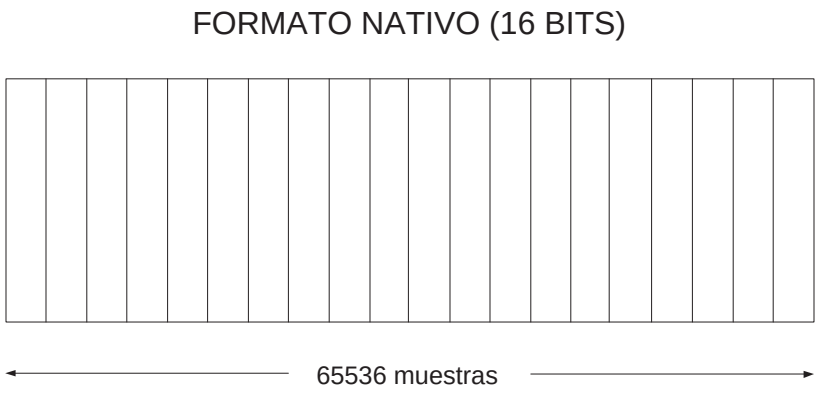


Figura 3.4: Bloque de audio en formato nativo.

```
add_new_wave_in_hdr (wave_in, wave_data);
add_new_wave_in_hdr (wave_in, wave_data);
```

Se manda a llamar la función `add_new_wave_in_hdr(wave_in wave_data)` dos veces para iniciar la grabación con dos bloques listos, y el proceso de incrementar el número de bloques hasta diez se delega en la función `CALLBACK` que se verá en detalle más adelante. De esta manera se tiene un bloque en blanco adicional para solventar alguna contingencia en el tiempo de ejecución (El código fuente de la función `add_new_wave_in_hdr` y otras funciones auxiliares se muestran mas adelante).

Es importante mencionar que las estructuras que se generan con la función `add_new_wave_in_hdr`, alojan los bloques de audio en formato nativo, es decir de 16 bits. En la Figura 3.4 se ilustra la estructura de un bloque de audio manejado internamente por el API de Windows.

En seguida, se utiliza la función `waveInStart` para encender el dispositivo de audio e iniciar la grabación:

```
if (waveInStart (wave_in) != MMSYSERR_NOERROR)
  {
    fputs ("waveInStart: Error.\n", stderr);
    exit (EXIT_FAILURE);
  }
```

En caso de haber alguna falla al encender el dispositivo de entrada de audio el programa se termina con un anuncio de falla ‘‘`waveInStart: Error.`’’ y en caso contrario se sigue ejecutando el programa.

En el siguiente fragmento se programa un ciclo indefinido durante el cual se atienden los mensajes que envía la función `CALLBACK` explicada más adelante:

```
while (recording)
  {
```

```
    ;
  }
```

Este ciclo termina cuando la función `CALLBACK` pone la variable `recording` en cero.

Por último, es conveniente y necesario concluir el procesamiento reponiendo y cerrando el dispositivo de entrada de audio; evitando así, que el acceso al dispositivo de audio quede bloqueado, y permitiendo que el sistema continúe funcionando correctamente al terminar el programa. Esto se realiza con las siguientes instrucciones:

```
  waveInReset (wave_in);
  waveInClose (wave_in);
}
```

El último corchete termina la función `record_wave_data`.

3.2.3. Rutina `CALLBACK` del programa `rec.c`.

Como se mencionaba anteriormente, al inicir el procesamiento de grabación, Windows manda llamar en varias ocasiones la función `waveInProc` de tipo `CALLBACK`, definido por Windows. El nombre de esta función es arbitrario (a gusto del programador) y básicamente sirve como un formato o templete, en el que se atienden mensajes originados durante la operación del dispositivo de audio: apertura del dispositivo, recepción de un nuevo bloque de audio, cierre del dispositivo. En resumen, la función `CALLBACK` es llamada internamente (por el sistema operativo) con mensajes que indican el estado del dispositivo.

La función `CALLBACK` y los mensajes que el sistema operativo envía a través de ella para atender a los dispositivos de entrada de audio, se describen de la siguiente manera en la documentación del Wave-API.

`CALLBACK`: Es una función de respuesta que Windows llama al usarse el dispositivo de entrada de audio en forma de onda. Necesita cinco parámetros: `HANDLE hwave` indica al dispositivo de forma de onda, `UINT message` contiene el mensaje de control del dispositivo de forma de onda según el estado en el que se encuentra; es decir, si está listo para recibir datos, si los está recibiendo y/o procesando, o si se debe cerrar debido a que ya no hay datos de entrada (trabajo realizado). Los mensajes que se reciben pueden ser:

- `WIM_OPEN`: El mensaje `WIM_OPEN` es enviado a la función `CALLBACK` cuando el dispositivo de forma de onda (entrada) es abierto con la función `waveInOpen`.
- `WIM_DATA`: Este mensaje es enviado cuando el manejador del dispositivo de entrada terminó con un bloque de datos enviados por la función `waveInAddBuffer`.
- `WIM_CLOSE`: Es enviado cuando el dispositivo de entrada es cerrado por la función `waveInClose`.

`DWORD dwInstance`: Es el tercer parámetro de la función `CALLBACK` que hace uso del tipo de datos especificados con `waveInOpen`.

`DWORD dwParam1, DWORD dwParam2:` Son parámetros reservados de Windows los cuales deben ser cero.

Nota: La función de `CALLBACK` no produce ningún valor.

En nuestro caso, la función `CALLBACK` se llama `wave_in_func` y se programó de la siguiente manera:

```
void CALLBACK
wave_in_func (HANDLE hwave, UINT message, DWORD dwInstance,
              DWORD dwParam1, DWORD dwParam2)
{
  switch (message)
    {
    case WIM_OPEN:
      {
        recording = 1;
        fputs ("Recording", stderr);
        break;
      }
    case WIM_DATA:
      {
        LPWAVEHDR wave_hdr = (LPWAVEHDR) dwParam1;

        if ((blocks_recorded + 2) < num_blocks)
          {
            add_new_wave_in_hdr (hwave, wave_data);
          }
        if (blocks_recorded < num_blocks)
          {
            blocks_recorded++;
            accumulate_wave_hdr_to_wave_data (wave_hdr, wave_data);
          }
        if (blocks_recorded == num_blocks)
          {
            recording = 0;
          }
        delete_wave_in_hdr (hwave, wave_hdr);
        fputc ('.', stderr);
        break;
      }
    case WIM_CLOSE:
      {
        fputc ('\n', stderr);
```

```
        break;
      }
    }
}
```

Como se puede observar, dentro del programa `rec.c` y en la rutina `CALLBACK`, existen diferentes funciones auxiliares. A continuación se muestra el código fuente de estas funciones, con una breve explicación de su funcionamiento.

3.2.4. Funciones auxiliares del programa `rec.c`.

A continuación se presenta el código fuente de las funciones utilizadas en el programa `rec.c` y en la rutina `CALLBACK`.

Función: `add_new_wave_in_hdr`.

La función `add_new_wave_in_hdr` agrega al dispositivo de audio indicado por `wave_dev`, un nuevo bloque de datos para grabar audio en formato nativo (16 bits), compatible en número de canales y número de muestras, con la estructura `wave_data`, que especifica un bloque de audio en formato de punto flotante (apto para procesamiento numérico).

```
void
add_new_wave_in_hdr (HWAVEIN wave_dev, WAVE_DATA * wave_data)
{
  LPWAVEHDR wave_hdr;
  size_t size;

  size = BYTES_PER_SAMPLE * wave_data->channels * wave_data->samples;
  wave_hdr = create_wave_hdr (size);
  waveInPrepareHeader (wave_dev, wave_hdr, sizeof (WAVEHDR));
  wave_hdr->dwUser = (DWORD) wave_data;
  waveInAddBuffer (wave_dev, wave_hdr, sizeof (WAVEHDR));
}
```

Función: `create_wave_hdr`.

La función `create_wave_hdr` crea un nuevo bloque de audio en formato nativo. Primeramente, se reserva memoria para la estructura `wave_hdr`, asegurándose que esté en blanco llenándola de ceros; en ella se escribirá el encabezado del bloque. Enseguida se reserva memoria para las muestras del bloque de audio `block` que también se inicializa con ceros.

```
LPWAVEHDR
create_wave_hdr (size_t size)
{
```

```
  LPWAVEHDR wave_hdr;
  void * block;

  wave_hdr = safe_malloc (sizeof (WAVEHDR));
  memset (wave_hdr, 0, sizeof (WAVEHDR));

  block = safe_malloc (size);
  memset (block, 0, size);

  wave_hdr->lpData = block;
  wave_hdr->dwBufferLength = size;
  wave_hdr->dwFlags = 0;

  return wave_hdr;
}
```

Función: `safe_malloc`.

La función `safe_malloc` se utiliza para reservar memoria de una manera segura; en caso de no haber suficiente espacio, el programa da el aviso ‘‘`Out of memory`’’ y termina inmediatamente.

```
static void *
safe_malloc (size_t size)
{
  void * mem;
  mem = malloc (size);
  if (mem == NULL)
    {
      fprintf (stderr, "Out of memory\n");
      exit (EXIT_FAILURE);
    }
  return mem;
}
```

Función: `accumulate_wave_hdr_to_wave_data`.

Esta función acumula (suma) el bloque de audio de la estructura `wave_hdr` (formato nativo) a la estructura `wave_data` (formato de punto flotante). Las estructuras `wave_data` se verán mas adelante.

```
void
accumulate_wave_hdr_to_wave_data (LPWAVEHDR wave_hdr,
                  WAVE_DATA * wave_data)
```

```
{
  unsigned int n, m, step;
  short int * src;
  double * dest;

  step = wave_data->channels;
  for (n = 0; n < wave_data->channels; n++)
    {
      src = (short int *) wave_hdr->lpData;
      src += n;
      dest = wave_data->channel[n];
      for (m = 0; m < wave_data->samples; m++)
    {
      (*dest++) += (*src);
      src += step;
    }
    }
}
```

Función: `delete_wave_in_hdr`.

La función `delete_wave_in_hdr` libera la memoria reservada para la estructura `wave_in_hdr`.

```
void
delete_wave_in_hdr (HWAVEIN wave_dev, LPWAVEHDR wave_hdr)
{
  if (wave_hdr != NULL)
    {
      if ((wave_hdr->dwFlags & WHDR_DONE) == 0)
    {
      fputs ("delete_wave_in_hdr (): "
         "refusing to delete an active buffer\n",
         stderr);
      return;
    }
      waveInUnprepareHeader (wave_dev, wave_hdr, sizeof (WAVEHDR));
      free (wave_hdr->lpData);
      free (wave_hdr);
    }
}
```

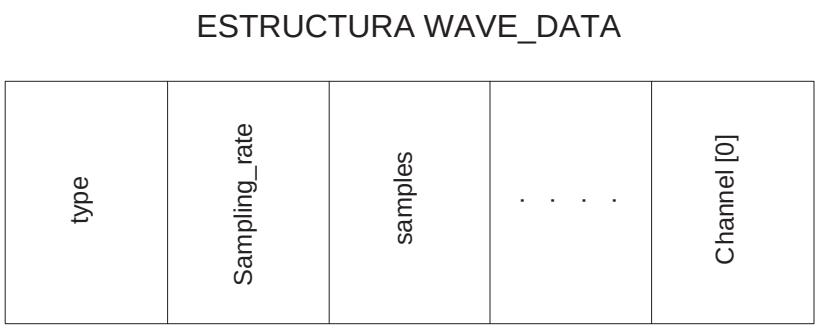


Figura 3.5: Estructura `WAVE_DATA`.

3.2.5. Estructura `WAVE_DATA`.

Como se mencionó anteriormente, con el API de Windows se procesan bloques de datos en el formato nativo del dispositivo de audio (por ejemplo en 16 bits). Sin embargo, por razones de precisión en los cálculos numéricos, este tipo de datos no es conveniente para realizar el procesamiento digital de las señales. En su forma nativa, el bloque de audio se almacena como números enteros de 16 bits, que en términos de lenguaje `C` se puede representar, por ejemplo, en variables de tipo `short int` (dependiendo del compilador). Paralelamente a la estructura para bloques de audio en formato nativo, crearemos una estructura semejante, pero con datos de tipo `double float` o *punto flotante de doble precisión*. En este formato, es posible procesar los datos de manera más confiable y conveniente al utilizar procedimientos de cálculo numérico, como la transformada rápida de Fourier (FFT), entre otras.

La estructura `WAVE_DATA` se define de la siguiente manera:

```
typedef struct {
  int type;
  unsigned int sampling_rate;
  unsigned int channels;
  unsigned int samples;
  double * channel[2];
} WAVE_DATA;
```

Esta estructura contiene diferentes campos como se ilustra en la Figura (3.5).

1. La variable `type` se utiliza para indicar con un (0) si los datos contenidos representan un bloque de audio en función del tiempo, o con un (1) si los datos están en función de la frecuencia.

2. La variable `sampling_rate` indica la frecuencia de muestreo en hertz.

3. La variable `channels` el número de canales.

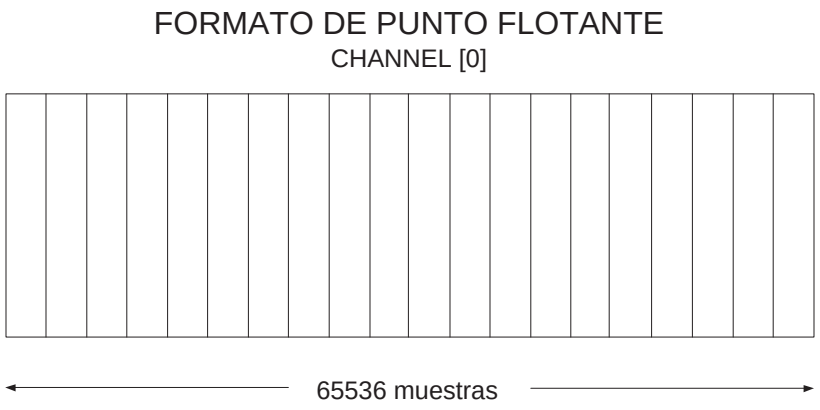


Figura 3.6: Bloque de audio con formato de punto flotante.

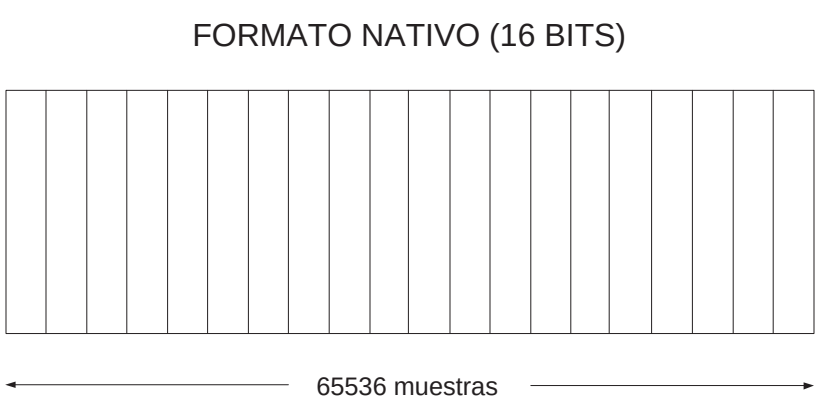


Figura 3.7: Bloque de audio en formato nativo.

4. La variable `samples` es el número de muestras por canal.

Los bloques de audio se acumulan en un arreglo de dos índices `double * channel [2]`, en el que se pueden representar señales de audio monofónicas (de un canal) o estereofónicas (de dos canales). En el caso monofónico, el bloque de audio está representado por el elemento `channel[0]`. En el caso estereofónico, el bloque de audio del canal izquierdo está representado por `channel[0]` y el canal derecho por `channel[1]`.

La estructura `WAVE_DATA` contiene la información de los bloques de audio en forma paralela (equivalente) al formato nativo del dispositivo de audio, en un formato más adecuado para procesarlos numéricamente. Comparar Figura (3.6) vs. Figura (3.7).

3.3. Programación del proceso de reproducción de sonido.

A continuación se presenta brevemente la contraparte de `rec.c`: un programa que reproduce audio. Está escrito igualmente en lenguaje `C` y su nombre es `snd.c`, haciendo referencia al verbo "to sound" en inglés que significa "sonar".

Para la reproducción del audio es necesario:

1. Leer los bloques de audio de un archivo previamente grabado.
2. Reproducirlos con un formato específico (wave_header).
3. Liberar espacio en memoria.

3.3.1. Programa de reproducción `snd.c`.

No hay mucha diferencia en la estructura general de los programas `snd.c` y `rec.c`, sólo se sustituyen algunas funciones que nos permitirán hacer la reproducción. Para no redundar en los puntos comunes, mostraremos sólo el nombre de las funciones que se utilizan en la reproducción de audio. Posteriormente, se muestra el código fuente de la rutina `CALLBACK` para `snd.c`, así como el de las funciones auxiliares no presentadas con anterioridad,

3.3.2. Función `play_wave_data`.

Esta función es muy similar a `record_wave_data`) del programa `rec.c`:

- Inicia de igual manera que en `rec.c`, indicando el formato de audio.
- Manda llamar la función `waveOutOpen` para hacer la gestión de acceso al dispositivo de salida y termina si hay algún error.
- Llama dos veces la función `add_new_wave_out_hdr` para preparar dos bloques o templetes de salida en blanco de reserva en la reproducción (véase la función `add_new_wave_in_hdr` en `rec.c`).
- Llama la función `waveOutReset (wave_out);` para reiniciar el dispositivo de salida.
- Llama la función `waveOutClose (wave_out);` para cerrar el dispositivo de salida.

3.3.3. Rutina `CALLBACK` del programa `snd.c`.

La función `waveOutProc` de tipo `CALLBACK` se llama `wave_out_func` y atenderá los siguientes mensajes emitidos por el manejador del dispositivo de salida de audio:

`WOM_OPEN:` El mensaje `WOM_OPEN` es enviado a la función `CALLBACK` cuando el dispositivo de salida de audio en forma de onda se abre con la función `waveOutOpen`.

`WOM_DONE`: Este mensaje es enviado internamente cuando se ha terminado de reproducir un bloque de audio.

`WOM_CLOSE`: Es enviado cuando el dispositivo de salida se cierra con la función `waveOutClose`.

A continuación se presenta el código fuente de la función `CALLBACK` para el dispositivo de salida.

```
void CALLBACK
wave_out_func (HANDLE hwave, UINT message, DWORD dwInstance,
               DWORD dwParam1, DWORD dwParam2)
{
  switch (message)
    {
    case WOM_OPEN:
      {
        playing = 1;
        fputs ("Playing", stderr);
        break;
      }
    case WOM_DONE:
      {
        LPWAVEHDR wave_hdr = (LPWAVEHDR) dwParam1;

        if ((blocks_played + 2) < num_blocks)
          {
            add_new_wave_out_hdr (hwave, wave_data, 0, 0);
          }
        if (blocks_played < num_blocks)
          {
            blocks_played++;
          }
        if (blocks_played == num_blocks)
          {
            playing = 0;
          }
        delete_wave_out_hdr (hwave, wave_hdr);
        fputc ('.', stderr);
        break;
      }
    case WOM_CLOSE:
      {
        fputc ('\n', stderr);
```

```
        break;
      }
    }
}
```

3.3.4. Funciones auxiliares del programa `snd.c`.

Terminamos este apartado concluyendo que la programación de los programas `rec.c` y `snd.c` tienen mucho en común, sólo algunos cambios marcan la diferencia entre grabar y reproducir audio. A continuación tenemos las funciones, de alguna manera diferentes, pues en la mayoría (como se vió anteriormente) sólo cambia el dispositivo de entrada por el de salida.

Función: `add_new_wave_out_hdr`.

La función `add_new_wave_out_hdr` agrega al dispositivo de audio indicado por `wave_dev`, un nuevo bloque de salida de audio en formato nativo, que es compatible en número de canales y número de muestras, con el bloque de audio indicado por `wave_data` en formato de punto flotante.

```
void
add_new_wave_out_hdr (HWAVEOUT wave_dev, WAVE_DATA * wave_data,
                      DWORD dwFlags, DWORD dwLoops)
{
  LPWAVEHDR wave_hdr;
  size_t size;

  size = BYTES_PER_SAMPLE * wave_data->channels * wave_data->samples;
  wave_hdr = create_wave_hdr (size);
  copy_wave_data_to_wave_hdr (wave_hdr, wave_data);
  waveOutPrepareHeader (wave_dev, wave_hdr, sizeof (WAVEHDR));
  wave_hdr->dwUser = (DWORD) wave_data;
  wave_hdr->dwFlags |= dwFlags;
  wave_hdr->dwLoops = dwLoops;
  waveOutWrite (wave_dev, wave_hdr, sizeof (WAVEHDR));
}
```

Función: `delete_wave_out_hdr`.

La función `delete_wave_out_hdr` libera la memoria reservada para la estructura `wave_hdr` (en forma similar, aunque no idéntica, a la función `delete_wave_in_hdr`).

```
void
delete_wave_out_hdr (HWAVEOUT wave_dev, LPWAVEHDR wave_hdr)
```

```
{
  if (wave_hdr != NULL)
    {
      if ((wave_hdr->dwFlags & WHDR_DONE) == 0)
    {
      fputs ("delete_wave_out_hdr (): "
         "refusing to delete an active buffer\n",
         stderr);
      return;
    }
      waveOutUnprepareHeader (wave_dev, wave_hdr, sizeof (WAVEHDR));
      free (wave_hdr->lpData);
      free (wave_hdr);
    }
}
```

3.4. Bibliotecas para calcular la FFT.

Cualquier señal expresada en el dominio del tiempo puede expresarse como una suma, discreta o continua según el caso, de sinusoides de distintas frecuencias. De este modo, la señal queda determinada como un conjunto, discreto o continuo, de amplitudes y fases (una para cada frecuencia) que forma el espectro en frecuencia de la señal; quedando así representada en el dominio de la frecuencia. El espectro en frecuencia se obtiene mediante la transformada de Fourier de la señal. Para efectos de procesamiento digital, el caso más común es disponer de una señal representada como un conjunto finito de muestras. Por ello, se suele usar la Transformada Discreta de Fourier (DFT) para pasar al dominio de la frecuencia.

El cálculo de la DFT, partiendo directamente de su definición matemática más común (ver Capítulo 2), es computacionalmente lento para la mayoría de las aplicaciones; sobre todo cuando se quiere hacer procesamiento digital de la señal en tiempo real. Por ello, es importante el uso de algoritmos de transformación rápida (FFT).

Aunque existen varias posibilidades para ello, nosotros usaremos la colección de algoritmos FFT llamado `FFTW` [12]. `FFTW` es una colección libre de rutinas rápidas en C para calcular Transformadas Discretas de Fourier en una o más dimensiones de tamaño arbitrario en la entrada, de datos reales y complejos. Dentro de las características que ofrece, está su buen desempeño en una variedad de plataformas. Incluye transformadas complejas, reales, simétricas y paralelas, inclusive maneja arreglos de diferentes dimensiones. Las siglas `FFTW` significan, en un tono jocoso: *"Fastest Fourier Transform in the West"*; es decir: "la transformada de Fourier más rápida del Oeste". Otra característica a considerar, es que dicha librería de rutinas es de libre distribución o software libre, se puede obtener de la siguiente dirección en la Internet: *http://www.fftw.org* y se puede acoplar al sistema de compilación de *cygwin* anteriormente descrito.

En este trabajo se utilizó la biblioteca de funciones `FFTW` para calcular la Transformada

Discreta de Fourier (DFT) de bloques de audio recibidos de la tarjeta de sonido de la PC y así lograr realizar su procesamiento en "tiempo real".

3.5. Programación del algoritmo de filtrado rápido.

Tomando en cuenta las secciones anteriores, se tienen las bases suficientes para detallar la programación del algoritmo de filtrado rápido. En esta sección se hará uso de las dos principales aplicaciones de la tarjeta de sonido: grabación (entrada) y reprodución (salida) de audio. Sólo que en esta parte se realizan de manera simultánea. Para ello se hizo el programa `audio-filt.c` en lenguaje C, con el detalle interesante de que tales procesamientos se harán con un ahorro considerable de tiempo. Esto gracias al algoritmo de filtrado rápido, y las bondades que ofrece la biblioteca FFTW, "Fastest Fourier Transform in the West". También se incluye la función `CALLBACK` y funciones auxiliares llamadas para efectuar el filtrado rápido.

3.5.1. Programa `audio-filt.c`.

El programa `audio-filt.c` tiene ciertas similitudes con los programas `rec.c` y `snd.c`, por lo que se explicarán en detalle sólo aquellas partes que difieren de los anteriores.

De la misma manera que se programaron `rec.c` y `snd.c`, `audio-filt.c` se inicia declarando variables globales donde se indica el formato de audio a través de los miembros de una estructura de datos. Como se muestra en el siguiente bloque de código fuente, se declaran también estructuras del tipo `WAVE_DATA` con apuntadores a diferentes vectores, estos alojarán el audio de entrada, audio procesado, filtros, etc. Esto se explicará en detalle más adelante.

```
/* Estructuras WAVE_DATA. */
WAVE_DATA * wHH[2]; // Filtro vs. frecuencia, tamaño 2N.
WAVE_DATA * wx;     // Señal de entrada vs. tiempo, tamaño N.
WAVE_DATA * wxx;    // Señal de entrada vs. tiempo, tamaño 2N.
WAVE_DATA * wXX;    // Señal de entrada vs. frecuencia, tamaño 2N.
WAVE_DATA * wy;     // Señal de salida vs. tiempo, tamaño N.
WAVE_DATA * wyy;    // Señal de salida vs. tiempo, tamaño 2N.
WAVE_DATA * wYY;    // Señal de salida vs. vs. frecuencia, tamaño 2N.
WAVE_DATA * waux;   // Señal de salida parcial vs. frecuencia, tamaño 2N.
```

Recordemos la forma en que se define la estructura `WAVE_DATA`.

```
/* Definición de estructura WAVE_DATA. */
typedef struct {
  int type; // 0:respuesta en tiempo, 1:respuesta en frecuencia
  unsigned int sampling_rate; // frecuencia de muestreo
  unsigned int channels; // número de canales
  unsigned int samples; // número de muestras por canal
```

```
  double * channel[2];
} WAVE_DATA;
```

Para utilizar la FFTW con eficacia, es necesario entender un concepto básico de la estructura interna de la FFTW, ya que no utiliza un algoritmo fijo para computar la transformada, pues puede adaptar el algoritmo a los detalles del hardware y así alcanzar un mejor funcionamiento. Para ello cuenta con un planificador (*planner*) que determina la manera más rápida de realizar la transformada para cada arquitectura de cómputo. El "planner" produce una estructura de datos llamada *plan* y esta contiene dicha información. Posteriormente, el *plan* se pasa al ejecutor de FFTW junto con los datos de entrada, y el ejecutor realiza la transformada como el *plan* se lo dicta. El plan se puede reutilizar tantas veces como se necesite.

Con las siguientes líneas se declaran las variables `rfftw_plan fft;` para realizar la transformada de Fourier directa (FFT) y `rfftw_plan ifft;` para realizar la transformada de Fourier inversa (IFFT):

```
/* Planes para FFT/IFFT. */
rfftw_plan fft;
rfftw_plan ifft;

// Dispositivos de entrada y salida de audio.
HWAVEIN wave_in;
HWAVEOUT wave_out;

// Indicador de actividad del proceso de filtrado.
volatile int filtering = 0;
```

En esta parte también se declaran las variables con las que se hará referencia a los dispositivos de entrada y salida de audio, y un indicador de que el proceso de filtrado está activo o inactivo.

3.5.2. Función: `audio_filt_init`.

En esta función se lee la especificación del filtro desde un archivo de datos. El filtro puede ser monofónico (un canal de entrada y uno de salida) o estereofónico (dos canales de entrada y dos de salida). También se reserva memoria para varias estructuras `WAVE_DATA` que se utilizarán durante la ejecución del programa para almacenar bloques de audio de las señales de entrada y salida, además de otras que servirán de auxiliares.

La función `audio_filt_init` inicia abriendo el archivo que contiene el filtro, para leer los datos y alojarlos en `wx` que es un vector de entrada de tamaño `N` que equivale al número de muestras. En el código que sigue, el vector `wx`, creado con la función `read_wave_data` se "pedirá prestado" (usándolo temporalmente) sólo para leer el filtro, ya que su función fundamental en el programa es otra, como se explicará más adelante.

```
void
audio_filt_init (void)
{
  FILE * file;
  int unsigned m;

  file = safe_fopen (filename, "r");
  wx = read_wave_data (file);
  wave_data_scale (wx, scale);
  sampling_rate = wx->sampling_rate;
  channels = wx->channels;
  N = samples = wx->samples;
  NN = 2 * N;
  if (channels != 1 && channels != 2)
    {
      fprintf (stderr, "Sólo para uno o dos canales!\n");
      exit (EXIT_FAILURE);
    }
```

Una vez leído el filtro, se aplica un factor de escala (opcional) para controlar la ganancia global del filtro), se asignan los parámetros de formato en la estructura. En el caso de que la señal de audio no sea monoaural o estéreo, el programa terminará con el siguiente aviso: ‘‘Sólo para uno o dos canales!’’.

Continuando con la función `audio_filt_init`, se crean las estructuras de tipo `WAVE_DATA` con las cuales se realiza el procesamiento de filtrado rápido.

```
  wy = create_wave_data (0, sampling_rate, channels, N);
  wxx = create_wave_data (0, sampling_rate, channels, NN);
  wyy = create_wave_data (0, sampling_rate, channels, NN);
  wXX = create_wave_data (1, sampling_rate, channels, NN);
  wYY = create_wave_data (1, sampling_rate, channels, NN);
```

La función `create_wave_data` se explicó en `rec.c`. El primer parámetro indica el dominio de los datos, el valor `0` indica el dominio del tiempo y `1` el de la frecuencia, `sampling_rate` indica la frecuencia de muestreo, `channels` el número de canales y `N`, `NN=2*N`, indican el tamaño de las estructuras (número de muestras).

En el siguiente ciclo (`for`), se crean las estructuras `WAVE_DATA` para los elementos de la matriz de filtros `wHH` (en correspondencia con el número de canales). En ellas se alojará el filtro, representado en el dominio de la frecuencia, en bloques de doble tamaño `NN`. Al terminar el ciclo se crea otra estructura llamada `waux` de iguales características.

```
  for(m=0; m<channels; m++)
```

```
    {
      wHH[m] = create_wave_data (1, sampling_rate, channels, NN);
    }
  waux = create_wave_data (1, sampling_rate, channels, NN);
```

A continuación, se crean los planes para la FFTW (anteriormente explicados). Tanto para la transformada directa (FFT) como para la transformada inversa (IFFT).

```
  fft = rfftw_create_plan (NN, FFTW_REAL_TO_COMPLEX, FFTW_ESTIMATE);
  ifft = rfftw_create_plan (NN, FFTW_COMPLEX_TO_REAL, FFTW_ESTIMATE);
```

Como se recordará, la estructura `wx` (que pedimos "prestada") contiene los datos leídos del archivo del filtro. Con la función `wave_data_to_time` se convierten al dominio del tiempo (aplicando la transformada inversa IFFT en caso necesario) en `wy` (que también se usa aquí de manera sólo temporal).

```
  wave_data_to_time (wy, wx, NULL);
```

Teniendo los datos del filtro en el dominio del tiempo, se copia `wy` de tamaño `N`, a la primera mitad de `wyy` de tamaño doble `NN` (usada también de forma temporal), esto se hace para cada canal, controlado por un ciclo `for` como se puede observar en el código que sigue.

```
  for (m=0; m<channels; m++)
    {
      memcpy (wyy->channel[m], wy->channel[m], N * sizeof (double));
      memset (wy->channel[m], 0, N * sizeof (double));
    }
```

`memcpy` se utiliza para copiar `wy` a `wyy` y `memset` para llenar con ceros el área de datos de `wy`.

El filtro en su nueva ubicación `wyy` tiene que convertirse al dominio de la frecuencia y almacenarse en `wHH` (destino final), quedando así preparado para aplicar el algoritmo de filtrado rápido en el dominio de la frecuencia.. La transformación al dominio de la frecuencia se hace utilizando la función `wave_data_to_freq`. Primero se transforma el primer elemento del vector `wHH`. Si el filtrado es monofónico (un sólo canal), entonces eso concluye la transformación y se cierra el archivo.

```
  wave_data_to_freq (wHH[0], wyy, fft);

  if (channels == 1)
    {
      fclose (file);
    }
```

Si se trata de dos canales, entonces se libera la memoria reservada para `wx` y se lee, a través de esa misma variable, la segunda columna de la matriz de filtros (que luego se almacenará en el segundo elemento de `wHH`). Después se cierra el archivo del filtro y se repite el mismo procedimiento anterior. Es decir, se escalan los coeficientes del filtro, se transforman al dominio del tiempo en `wy`, se copian a `wyy` (de tamaño `NN`), se almacenan ceros en `wy`, y se convierten los filtros al dominio de la frecuencia.

```
  else
    {
      delete_wave_data (wx);
      wx = read_wave_data (file);
      fclose (file);
      wave_data_scale (wx, scale);
      wave_data_to_time (wy, wx, NULL);
      for(m=0; m<channels; m++)
        {
          // Copiar filtro en wy a primera mitad de wyy.
          memcpy (wyy->channel[m], wy->channel[m], N * sizeof (double));
          // Poner wy en cero.
          memset (wy->channel[m], 0, N * sizeof (double));
        }
      wave_data_to_freq (wHH[1], wyy, fft);
    }
```

Ahora se especifica que `wx` está en el dominio del tiempo (en caso de que los elementos del filtro se hubieran leido de una especificación en el dominio de la frecuencia), con la asignación:

```
  wx->type = 0;
}
```

Esto deja preparados los datos del filtro en la forma y estructura deseada. El último corchete concluye la función `audio_filt_init`.

3.5.3. Función: `audio_filt`.

En este programa, se sustituyen las funciones `record_wave_data` y `play_wave_data` (de los programas anteriores) por `audio_filt`, que en cierta forma es una mezcla de ambas. Para mayor referencia revisar en `rec.c` la sección 3.2.2 y en `snd.c` la sección 3.3.2.

```
void
audio_filt (void)
{
```

```
PCMWAVEFORMAT wave_fmt;

wave_fmt.wf.wFormatTag = WAVE_FORMAT_PCM;
wave_fmt.wf.nChannels = channels;
wave_fmt.wf.nSamplesPerSec = sampling_rate;
wave_fmt.wf.nAvgBytesPerSec = channels * bytes_per_sample * sampling_rate;
wave_fmt.wf.nBlockAlign = channels * bytes_per_sample;
wave_fmt.wBitsPerSample = 8 * bytes_per_sample;

if (waveInOpen (&wave_in,
                WAVE_MAPPER,
                (LPWAVEFORMATEX) &wave_fmt,
                (DWORD) wave_in_func,
                (DWORD) NULL,
                CALLBACK_FUNCTION) != MMSYSERR_NOERROR)
  {
    fputs ("waveInOpen: Error.\n", stderr);
    exit (EXIT_FAILURE);
  }
if (waveOutOpen (&wave_out,
                WAVE_MAPPER,
                (LPWAVEFORMATEX) &wave_fmt,
                (DWORD) wave_out_func,
                (DWORD) NULL,
                CALLBACK_FUNCTION) != MMSYSERR_NOERROR)
  {
    fputs ("waveOutOpen: Error.\n", stderr);
    exit (EXIT_FAILURE);
  }
add_new_wave_in_hdr (wave_in, wx);
if (waveInStart (wave_in) != MMSYSERR_NOERROR)
  {
    fputs ("waveInStart: Error.\n", stderr);
    exit (EXIT_FAILURE);
  }
```

A partir de este momento, quedan habilitados ambos dispositivos de audio (entrada y salida), realizándose el procedimiento de filtrado rápido que se explica más adelante. A continuacion se muestra el código en donde se indica que en caso de que el usuario oprima "Enter", la variable `filtering` se pone en cero, lo que produce que se detengan, se reinicialicen y se cierren los dispositivos de entrada y salida de audio. Esto terminaría el proceso de filtrado rápido.

```
  getchar (); // Procesar audio, y esperar a que el usuario oprima Enter.
  filtering = 0; // Marcar fin del filtrado.

  /* Close audio wave devices.
     Order is important here: */
  waveInStop (wave_in);
  waveInReset (wave_in);
  waveOutReset (wave_out);
  waveInClose (wave_in);
  waveOutClose (wave_out);
}
```

3.5.4. Rutinas CALLBACK del programa audio-filt.c.

Como se mencionó anteriormente en la sección 3.2.3, la función `CALLBACK` es una función de respuesta que Windows llama al usarse el dispositivo de audio, administrando mensajes de control del dispositivo de audio (tarjeta de sonido) según el estado en que se encuentre o la operación que esté realizando. En este apartado se explicará brevemente dicha función apoyándose en las rutinas `CALLBACK` de `rec.c` y `snd.c`.

Entrada de audio.

Empezaremos diciendo que al abrir el dispositivo de entrada de audio la variable que indica que el programa se está ejecutando es `filtering`, a diferencia de la variable `recording` usada en `rec.c`. Así mismo, se indica la tecla con la cual se puede suspender la ejecución del programa; en este caso se determinó utilizar la tecla `Enter`.

```
void CALLBACK
wave_in_func (HANDLE hwave, UINT message, DWORD dwInstance,
              DWORD dwParam1, DWORD dwParam2)
{
  switch (message)
    {
    case WIM_OPEN:
      {
        filtering = 1;
        fputs ("audio-filt: Terminar con Enter...", stderr);
        fflush (stderr);
        break;
      }
```

Para `audio-filt.c` no se especifica un número de bloques a grabar, pues la duración de la ejecución del programa la determina el usuario, iniciando o deteniendo dicho proceso. Por

lo tanto WIM_DATA que lleva el registro de los bloques terminados, se ejecuta bajo la condición de que filtering esté activo para agregar más bloques si es necesario.

```
    case WIM_DATA:
      {
        LPWAVEHDR wave_hdr = (LPWAVEHDR) dwParam1;

        if (filtering)
          {
            add_new_wave_in_hdr (hwave, wx);
            filter_wave_data (wave_hdr);
          }
        delete_wave_in_hdr (hwave, wave_hdr);
        break;
      }
    case WIM_CLOSE:
      {
        break;
      }
    }
}
```

El caso WIM_CLOSE se ejecuta cuando se cierra el dispositivo de audio.

Salida de audio.

Ahora, para explicar esta segunda parte de la rutina CALLBACK de audio_filt.c nos apoyaremos en la misma rutina para snd.c de la sección 3.3.3, donde los mensajes enviados son para el dispositivo de salida de audio.

```
void CALLBACK
wave_out_func (HANDLE hwave, UINT message, DWORD dwInstance,
               DWORD dwParam1, DWORD dwParam2)
{
  switch (message)
    {
    case WOM_OPEN:
      {
        break;
      }
    case WOM_DONE:
```

```
      {
        LPWAVEHDR wave_hdr = (LPWAVEHDR) dwParam1;

        delete_wave_out_hdr (hwave, wave_hdr);
        break;
      }
    case WOM_CLOSE:
      {
         break;
      }
    }
}
```

En esta parte no existe la variable `playing` como en `rec.c` para sólo sonar o reproducir audio, sino que se maneja por la variable `filtering` antes mencionada, ya que el programa `audio_filt.c` une las dos aplicaciones, grabar y reproducir. Atendiendo a mensajes de `WOM_OPEN` para abrir el dispositivo de salida de audio, `WOM_DONE` que es enviado cuando un bloque se terminó de reproducir, y `WOM_CLOSE` que es enviado mensaje cuando se ha cerrado dicho dispositivo.

3.5.5. Función: `filter_wave_data`.

La función `filter_wave_data` realiza el algoritmo de filtrado rápido, recibiendo bloques de señal del dispositivo de entrada de audio, aplicando el filtro, y produciendo bloques de señal para el dispositivo de salida de audio. Es importante notar que esta `filter_wave_data` se ejecuta desde la función `CALLBACK` del dispositivo de *entrada de audio*; específicamente, en atención del mensaje `WIM_DATA`, cada vez que se recibe un nuevo bloque de entrada de audio.

La función `filter_wave_data`, igual que las funciones anteriores, inicia declarando variables.

```
void
filter_wave_data (LPWAVEHDR wave_hdr)
{
  unsigned int n;
  unsigned int k;

  // Copiar bloques del dispositivo de entrada de audio en wx.
  copy_wave_hdr_to_wave_data (wave_hdr, wx);
```

La función `copy_wave_hdr_to_wave_data` copia los bloques de entrada de audio que se reciben de la tarjeta de sonido, de la estructura `wave_hdr` de tipo `WAVE_HDR`, a la estructura `wx` de tipo `WAVE_DATA`. La estructura `wx`, utilizada anteriormente de manera temporal, desempeña

ahora su papel principal. Recordemos que `wx` se usó antes temporalmente para leer los datos del filtro. Ahora, su función es recibir los bloques de entrada de audio. Lo mismo ocurre más adelante con las estructuras `wy`, `wyy`, `wYY`, que fueron usadas antes de manera temporal, pero que son usadas en lo que sigue en su papel principal (alojar bloques de la señal de salida).

La estructura `wxx`, es de tamaño doble `NN=2*N`, mientras que `wx` es de tamaño `N`. Por lo que dividiremos para esta explicación a `wxx` en dos partes, primera y segunda mitad. En la primera mitad se alojará el bloque de "muestras antiguas" y en la segunda mitad el bloque de "muestras nuevas".

Como en la segunda mitad de `wxx` se alojará un "bloque nuevo", es necesario antes mover el bloque que ya está ahí para que cuando se guarde "el más nuevo", es decir el siguiente bloque de datos, no sobreescriba al anterior. Lo que se hace es copiar este bloque, ahora "antiguo", a la primera mitad de `wxx`. En seguida se copian los datos de `wx` (entrada de audio), a la segunda mitad de `wxx` (bloques nuevos), y se prepara `wYY` llenándola con ceros, para ser utilizada más adelante. Estas operaciones se realizan para cada uno de los canales de audio, controlado por un ciclo `for`.

```
for (k=0; k<channels; k++)
  {
    // Copiar segunda mitad de wxx a primera mitad de wxx
    memcpy (wxx->channel[k], wxx->channel[k] + N, N * sizeof (double));

    // Copiar wx a segunda mitad de wxx
    memcpy (wxx->channel[k] + N, wx->channel[k], N * sizeof (double));

    // Almacenar ceros en wYY
    memset (wYY->channel[k], 0, NN * sizeof (double));
  }
```

La estructura `wxx` contiene hasta ahora una mitad con datos antiguos y la segunda mitad con datos nuevos. Su transformación al dominio de la frecuencia quedará almacenada en `wXX`.

La operación que necesitamos realizar para fines del procesamiento de la señal es la convolución, explicada en la sección 2.2.1. Una operación un tanto complicada; sin embargo tiene una equivalente si se cambian los mismos datos al dominio de la frecuencia, realizando sólo una multiplicación para cada frecuencia.

En el programa, la operación de filtrado (multiplicación) en el dominio de la frecuencia se realiza transformándola de la siguiente manera:

$$\begin{bmatrix} Y_0(\omega_k) \\ Y_1(\omega_k) \end{bmatrix} = \begin{bmatrix} H_{00}(\omega_k) & H_{01}(\omega_k) \\ H_{10}(\omega_k) & H_{11}(\omega_k) \end{bmatrix} \begin{bmatrix} X_0(\omega_k) \\ X_1(\omega_k) \end{bmatrix}, \tag{3.1}$$

$$= \begin{bmatrix} H_{00}(\omega_k) \cdot X_0(\omega_k) \\ H_{10}(\omega_k) \cdot X_0(\omega_k) \end{bmatrix} + \begin{bmatrix} H_{01}(\omega_k) \cdot X_1(\omega_k) \\ H_{11}(\omega_k) \cdot X_1(\omega_k) \end{bmatrix}; \tag{3.2}$$

en donde ω_k indica la dependencia en función de la frecuencia discreta con índice k. Las variables que representan la función de respuesta del filtro y las señales de entrada y salida

en el dominio de la frecuencia, están representadas en el programa por estructuras `WAVE_DATA` (que pueden contener dos bloques de datos), de la siguiente manera:

$$\texttt{wHH[0]} = \begin{bmatrix} H_{00}(\omega_k) \\ H_{10}(\omega_k) \end{bmatrix}, \tag{3.3}$$

$$\texttt{wHH[1]} = \begin{bmatrix} H_{01}(\omega_k) \\ H_{11}(\omega_k) \end{bmatrix}, \tag{3.4}$$

$$\texttt{wXX} = \begin{bmatrix} X_0(\omega_k) \\ X_1(\omega_k) \end{bmatrix}, \tag{3.5}$$

$$\texttt{wYY} = \begin{bmatrix} Y_0(\omega_k) \\ Y_1(\omega_k) \end{bmatrix}; \tag{3.6}$$

en donde `wHH[0]` y `wHH[1]` contienen la respuesta del filtro (primera y segunda columna de la matriz de filtros respectivamente), `wXX` contiene dos bloques de audio (canal izquierdo X_0 y canal derecho X_1), y `wYY` contiene la señal de salida (Y_0 y Y_1).

El siguiente fragmento de código realiza la operación descrita:

```
// Transformar wxx al dominio de la frecuencia en wXX.
wave_data_to_freq (wXX, wxx, fft);

for (n=0; n<channels; n++)
  {
    // Replicar canal n de wXX en todos los canales de waux.
    for (k=0; k<channels; k++)
      {
        memcpy (waux->channel[k], wXX->channel[n], NN * sizeof (double));
      }
    // Multiplicar waux por columna n de wHH, resultado en waux.
    wave_data_cmult (waux, wHH[n]);

    // Acumular (sumar) waux en wYY.
    wave_data_add (wYY, waux);
  }
```

La función `wave_data_to_freq (wXX, wxx, fft)` transforma los bloques de audio al dominio de la frecuencia, usando las funciones de la biblioteca FFTW. Luego se ejecuta un ciclo `for` sobre el índice `n`, en el que se copia el canal `n` de `wXX`, en cada uno de los canales de `waux` (por medio de otro ciclo `for` sobre el índice `k`). De esta manera, para cada `n`, la estructura auxiliar `waux`, contiene el mismo bloque de señal, X_0 o X_1, *duplicado* en sus dos elementos:

$$\texttt{waux} = \begin{bmatrix} X_0(\omega_k) \\ X_0(\omega_k) \end{bmatrix}, \ (\texttt{n} = 0), \tag{3.7}$$

$$\texttt{waux} \;=\; \begin{bmatrix} X_1(\omega_k) \\ X_1(\omega_k) \end{bmatrix},\ (\texttt{n} = 1). \tag{3.8}$$

Como se comentó, la convolución en el dominio del tiempo equivale a una multiplicación en el dominio de la frecuencia. Esta operación se realiza con la función `wave_data_cmult (waux, wHH[n])`, que multiplica sus operandos elemento a elemento, y deja el resultado en el primer operando: `waux`. La función `wave_data_add (wYY, waux)` acumula el resultado en `wYY`, que contiene los elementos de la señal de salida en el dominio de la frecuencia.

Finalmente, la tarjeta de sonido deberá reproducir la señal filtrada en el dominio del tiempo, por lo que se implementó tambien la función `wave_data_to_time (wyy, wYY, ifft)` que vuelve los datos del dominio de la frecuencia al dominio del tiempo. En seguida, se copia la segunda mitad de `wyy` a `wy`, pues la primera no es utilizable (contaminada por los efectos de convoluación circular). Recordemos que `wyy` es de tamaño `NN` y `wy` de tamaño `N`, también recordemos que las estructuras de tamaño `N`, nos sirven para interactuar con la tarjeta de sonido, y las tamaño doble `NN` para realizar el algoritmo de filtrado rápido.

```
  // Transformar wYY al dominio del tiempo en wyy.
  wave_data_to_time (wyy, wYY, ifft);

  // Copiar segunda mitad de wyy en wy.
  for (k=0; k<channels; k++)
    {
      memcpy (wy->channel[k], wyy->channel[k] + N, N * sizeof (double));
    }

  // Pasar wy al bloque de salida del dispositivo de audio.
  add_new_wave_out_hdr (wave_out, wy, 0, 0);
}
```

La estructura de datos ya puede ser enviada al dispositivo de salida de audio, por lo tanto se pasan los datos al bloque de salida `wave_hdr`, compatible con el formato de datos nativo del Wave-API de Windows.

Esto concluye la función `filter_wave_data`.

3.5.6. Función: `audio_filt_term`.

Esta función se llama al terminar el programa. Aquí se libera la memoria asignada a todas las estructuras `WAVE_DATA` utilizadas.

```
void
audio_filt_term (void)
{
  unsigned int m;
```

```
  delete_wave_data (wx);
  delete_wave_data (wy);
  delete_wave_data (wxx);
  delete_wave_data (wyy);
  delete_wave_data (wXX);
  delete_wave_data (wYY);
  for (m=0; m<channels; m++)
    {
      delete_wave_data (wHH[m]);
    }
  delete_wave_data (waux);
```

También se libera la memoria asignada para los planes de FFT/IFFT:

```
  rfftw_destroy_plan (fft);
  rfftw_destroy_plan (ifft);
}
```

3.5.7. Función principal: `main`.

En la función principal se procesan los argumentos de la línea de comandos y se ejecutan en secuencia las funciones `audio_filt_init`, `audio_filt`, `audio_filt_term`.

```
int
main (int argc, char * argv[])
{
  char * progname = argv[0];

  /* Procesa argumentos de la línea de comandos. */
  for (;;)
    {
      char usage[] =
        "Uso: %s [OPCIONES]\n"
        "Opciones (y valores normales):\n"
        "  -i audio-filt.dat  Archivo de entrada.\n"
        "  -s 1.0             Factor de escala para el filtro.\n"
        "  -h                 Muestra esta ayuda y termina.\n"
        "";
      char bad_usage[] =
        "%s: Error en la línea de comando.  Intentar '%s -h'.\n";
      char optlist[] = "i:s:h";
      extern int optind;
```

```
      extern int opterr;
      extern char * optarg;
      extern int getopt ();
      int c;

      opterr = 0;
      c = getopt (argc, argv, optlist);
      if (c == EOF) /* No más opciones. */
        {
          if (argc > optind) /* Extra args? */
            {
              fprintf (stderr, bad_usage, progname, progname);
              exit (EXIT_FAILURE);
            }
          break;
        }
      switch (c)
        {
        case 'i':
          filename = optarg;
          break;
        case 's':
          scale = atof (optarg);
          break;
        case 'h':
          printf (usage, progname);
          exit (EXIT_SUCCESS);
        default:
          fprintf (stderr, bad_usage, progname, progname);
          exit (EXIT_FAILURE);
        }
    }

  audio_filt_init ();
  audio_filt ();
  audio_filt_term ();
  return EXIT_SUCCESS;
}
```

No encontrándose ningún error, se devuelve al sistema operativo un código indicando éxito (conclusión normal) en la ejecución del programa.

Capítulo 4

Aplicación a un sistema de reproducción biaural.

En este capítulo [13] se resume la realizacion de un sistema de reproducción de sonido biaural basado en los dispositivos de audio que están comúnmente disponibles en las computadoras personales actuales. El sistema opera en tres etapas:

1. Una etapa de identificacion de las respuestas a impulso del sistema electroacústico, entre los dos altavoces del sistema de sonido y dos micrófonos biaurales instalados en un maniquí acústico con dimensiones antropométricas promedio.

2. Una etapa de cálculo eficiente de filtros digitales óptimos para el control prealimentado del sistema de sonido.

3. Una etapa de reprodución biaural basada en algoritmos de filtrado rápido en tiempo real de las señales de sonido que desean reproducirse.

El sistema logra el objetivo de reproducir las señales de sonido en los oídos de manera totalmente controlada e independiente. Una de las aplicaciones previstas es la realización de pruebas auditivas evitando el uso de audífonos y algunos de los problemas asociados a estos. El algoritmo de filtrado rápido en tiempo real, que es el objeto de esta tesis, se inscribe precisamente en la tercera etapa de operación del sistema.

4.1. Sistema de reproducción.

Los sistemas de reproducción de sonido de dos canales suponen una distribución simétrica de fuentes como se muestra en la Figura (4.1). La técnica estereofónica convencional alimenta señales diferentes a las fuentes izquierda y derecha, previamente preparadas (mezcladas) para dar al oyente una sensación de espacio sonoro[1]. Este objetivo se cumple en forma limitada debido principalmente al cruce de canales que ocurre naturalmente de la fuente derecha al oído izquierdo y viceversa[2].

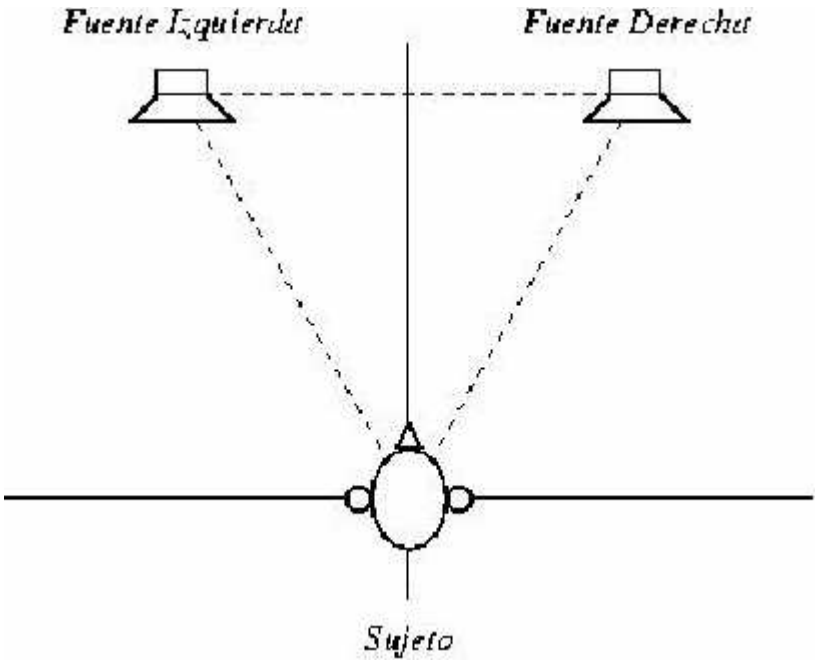


Figura 4.1: Sistema de reproducción de sonido de dos canales.

Si designamos con los símbolos $Q_1(z), Q_2(z)$ las transformadas z de las señales que alimentan a los altavoces (izquierdo y derecho, respectivamente), y con $P_1(z), P_2(z)$ las correspondientes a las señales reproducidas en los oídos del oyente; entonces la relación entre estas se da a través de las combinaciones lineales siguientes:

$$P_1(z) = G_{11}(z)Q_1(z) + G_{12}(z)Q_2(z), \quad (4.1)$$

$$P_2(z) = G_{21}(z)Q_1(z) + G_{22}(z)Q_2(z); \quad (4.2)$$

en donde $G_{11}(z)$ representa la función de respuesta del altavoz izquierdo al oído izquierdo, $G_{12}(z)$ la del altavoz derecho al oído izquierdo, etc. A cada una de estas funciones de respuesta le corresponde una respuesta a impulso en el dominio del tiempo.

El caso de la reproducción estereofónica convencional corresponde a poner $Q_1(z) = L(z)$ y $Q_2(z) = R(z)$, en donde $L(z)$ y $R(z)$ son las señales de canal izquierdo y derecho, respectivamente. La forma de las ecuaciones anteriores manifiesta claramente el cruce de canales que se produce en este caso.

La técnica de reproducción de sonido biaural, basada en la cancelación de cruce de canales (*acoustic crosstalk cancellation*), procesa las señales destinadas a los oídos izquierdo y derecho, en la etapa previa a la alimentación del los altavoces[3], como se muestra en la Figura (4.2). El procesamiento de las señales puede diseñarse de manera que las señales reproducidas en los oídos izquierdo y derecho del oyente correspondan a las señales deseadas [4,5]. Esta correspondencia puede ser sólo aproximada, especialmente a bajas frecuencias, en las que la separación entre los oídos es mucho menor que la longitud de onda acústica.

Por otra parte, en altas frecuencias, el efecto de cancelación de cruce se logra sólo en zonas muy pequeñas, localizadas en la posición original de los oídos del sujeto, limitando la libertad de movimiento. Sin embargo, a frecuencias medias y en situaciones fijas, la técnica biaural resulta muy efectiva. Las aplicaciones de interés incluyen: reproducción de audio para computadoras de escritorio, automóviles, y realización de pruebas audiométricas sin audífonos.

La Figura (4.2). muestra un sistema de cancelación de cruce para un sistema simétrico de dos canales. En el caso más general, las señales que se alimentan a los altavoces se expresan de la siguiente manera:

$$Q_1(z) = H_{11}(z)L(z) + H_{12}(z)R(z), \tag{4.3}$$
$$Q_2(z) = H_{21}(z)L(z) + H_{22}(z)R(z), \tag{4.4}$$

en donde $H_{11}(z)$ representa la función de respuesta del filtro biaural que alimenta el canal izquierdo al altavoz izquierdo, $H_{12}(z)$ la del canal derecho al altavoz izquierdo, etc. Estos filtros se diseñan para minimizar el valor cuadrático medio de las señales de error definidas de la siguiente manera:

$$E_1(z) = L(z) - P_1(z), \tag{4.5}$$
$$E_2(z) = R(z) - P_2(z), \tag{4.6}$$

este criterio conduce, de manera aproximada, a la condición deseada en la que se reproduce la señal izquierda en el oído izquierdo y la señal derecha en el oído derecho, minimizando el cruce de canales.

La realización práctica de un sistema de reproducción biaural se desarrolló, en este trabajo, con base en la plataforma de audio de Microsoft Windows(tm), común en las computadoras personales actuales. Este sistema está dividido en tres etapas (las primeras dos desarrolladas de manera independiente a esta tesis):

Una etapa de identificación del sistema de reproducción de sonido, en la que se alimentan señales de prueba (secuencias pseudo-aleatorias de máxima longitud [6], consecutivamente a los altavoces izquierdo y derecho, para obtener las respuestas a impulso correspondientes a las funciones de respuesta $G_{11}(z)$, $G_{12}(z)$, etc.

Una etapa de cálculo de los filtros biaurales $H_{11}(z)$, $H_{12}(z)$, etc., basado en el algoritmo de deconvolución rápida[7]. Este cálculo se efectuó en Matlab(tm).

Una etapa de filtrado rápido en tiempo real (objeto de esta tesis), basado en el algoritmo de convolución rápida [8], en la que se filtran las señales izquierda y derecha, a tasas de muestreo que pueden llegar a los 48 kHz, y se alimentan los altavoces para lograr la cancelación de cruce de canales en la posición del oyente.

Las siguientes secciones muestran resultados típicos de la operación de este sistema.

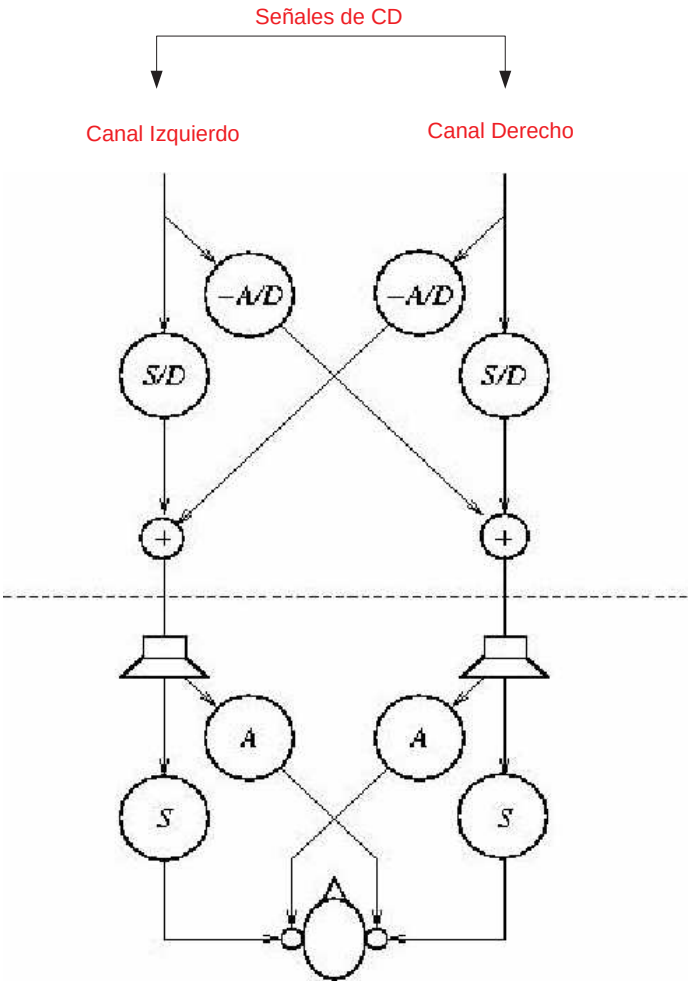


Figura 4.2: Diagrama de señales en un sistema de reproducción de sonido biaural.

4.2. Identificación del sistema de reproducción de sonido.

La Figura (4.3) muestra las respuestas a impulso correspondientes a un sistema de reproducción de sonido de dos canales. La Figura (4.4) muestra el nivel de cruce de canales que ocurre naturalmente. Se observa que el nivel de cruce es de alrededor de -10 dB para frecuencias entre 500 Hz y 8000 Hz.

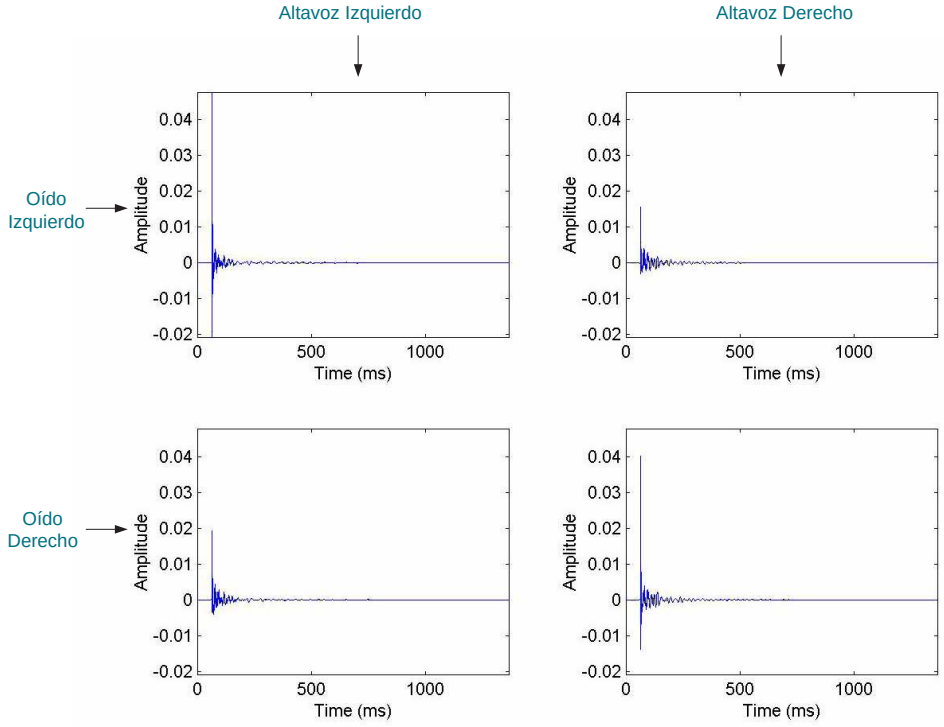


Figura 4.3: Respuestas a impulso del sistema de reproducción de sonido. Las dos gráficas del primer renglón corresponden a las señales reproducidas en el oído izquierdo provenientes del altavoz izquierdo y derecho, respectivamente. Las dos gráficas del segundo renglón corresponden a las señales reproducidas en el oído derecho.

4.3. Respuesta deseada y cálculo de filtros digitales.

La Figura (4.5) muestra las respuestas a impulso deseadas, entre los canales de entrada izquierdo y derecho y los oídos, en un sistema de reproducción biaural con cancelación de cruce de canales. La Figura (4.6) muestra las respuestas a impulso de los filtros digitales

de prealimentación que aproximan la respuesta del sistema de reproducción a la respuesta deseada.

4.4. Reproducción de señales biaurales.

La Figura (4.7) muestra una estimación de las respuestas del sistema de reproducción biaural incluyendo los filtros digitales de prealimentación. La Figura (4.8) muestra el nivel de cruce de canales correspondiente.

La Figura (4.9) muestra las respuestas a impulso medidas en el sistema de reproducción biaural. La Figura (4.10) muestra el nivel de cruce de canales correspondiente.

Resumiendo, en este capítulo1 se mostraron ejemplos de las etapas funcionales del sistema de reproducción biaural. Las respuestas a impulso medidas, corresponden con buena aproximación a las respuestas deseadas. El nivel de cruce de canales que se obtiene queda por debajo de -20 dB en un rango de frecuencias entre 500 Hz y 7000 Hz aproximadamente. El sistema se encuentra actualmente en una etapa de evaluación y depuración, con miras a mejorar su desempeño. En primera instancia, el algoritmo de filtrado rápido en tiempo real, cuya implementación en el sistema está descrita en esta tesis, mostró un funcionamiento correcto.

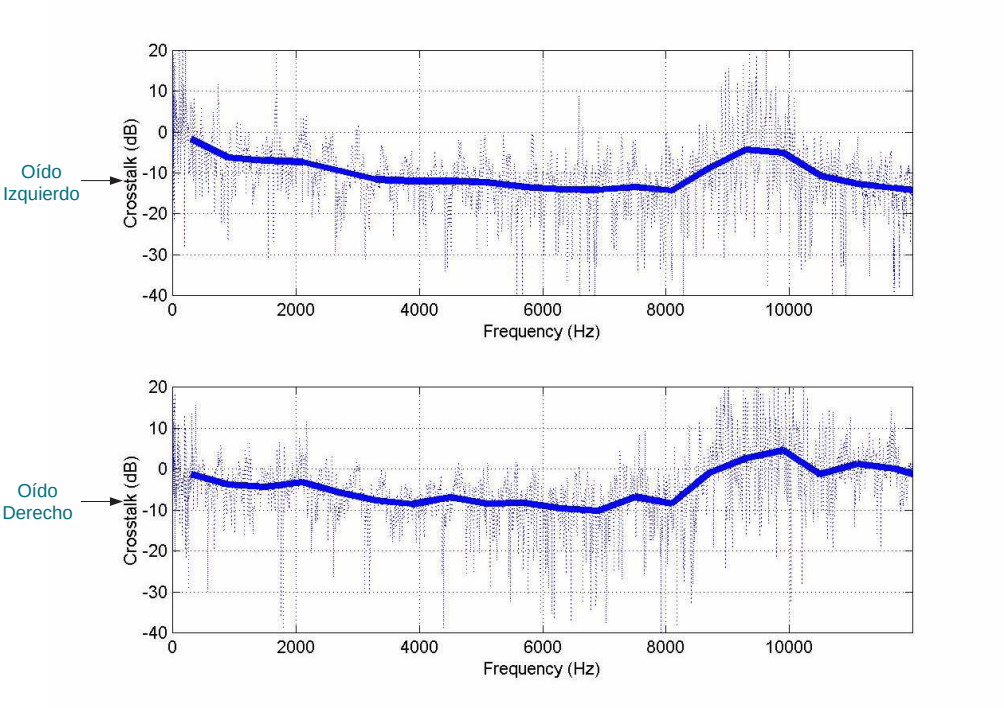


Figura 4.4: Nivel de cruce de canales del sistema de reproducción de sonido, debido principalmente al efecto acústico de la cabeza humana. La gráfica superior indica el nivel de cruce del canal derecho en la señal reproducida en el oído izquierdo. La gráfica inferior indica el nivel de cruce del canal izquierdo en el oído derecho.

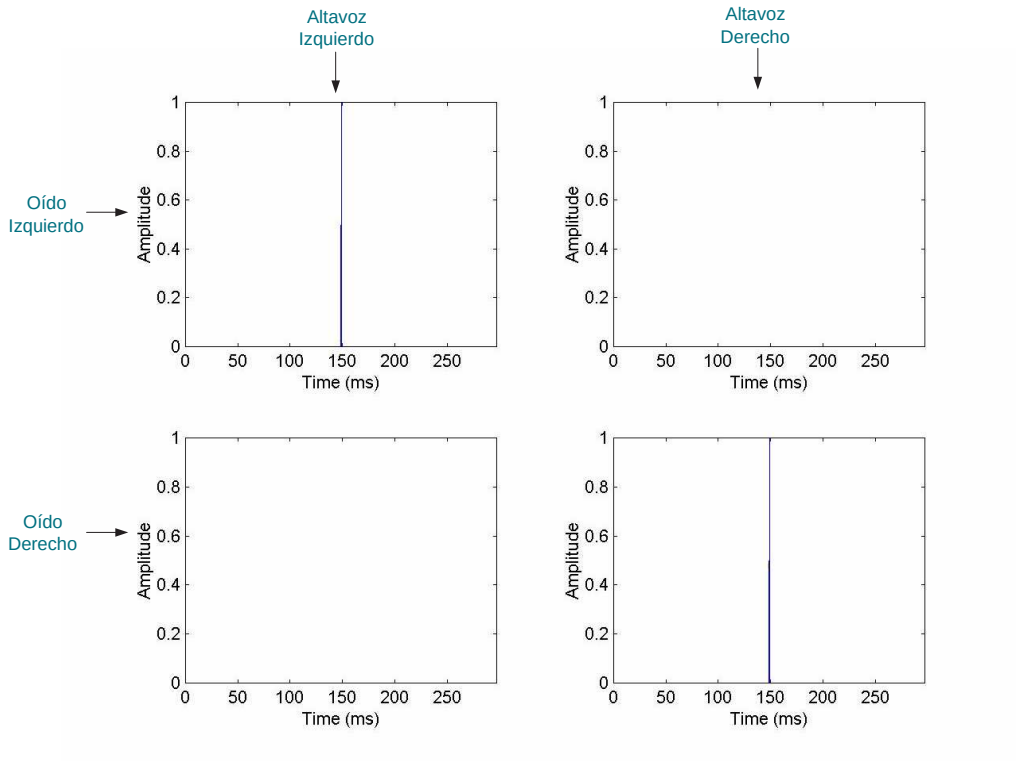


Figura 4.5: Respuesta deseada en un sistema de reproducción biaural. El objetivo es eliminar el cruce de canales (del canal derecho del oído izquierdo y viceversa), y, al mismo tiempo, mejorar la fidelidad de la reproducción de las señales (posiblemente independientes) destinadas a cada oído.

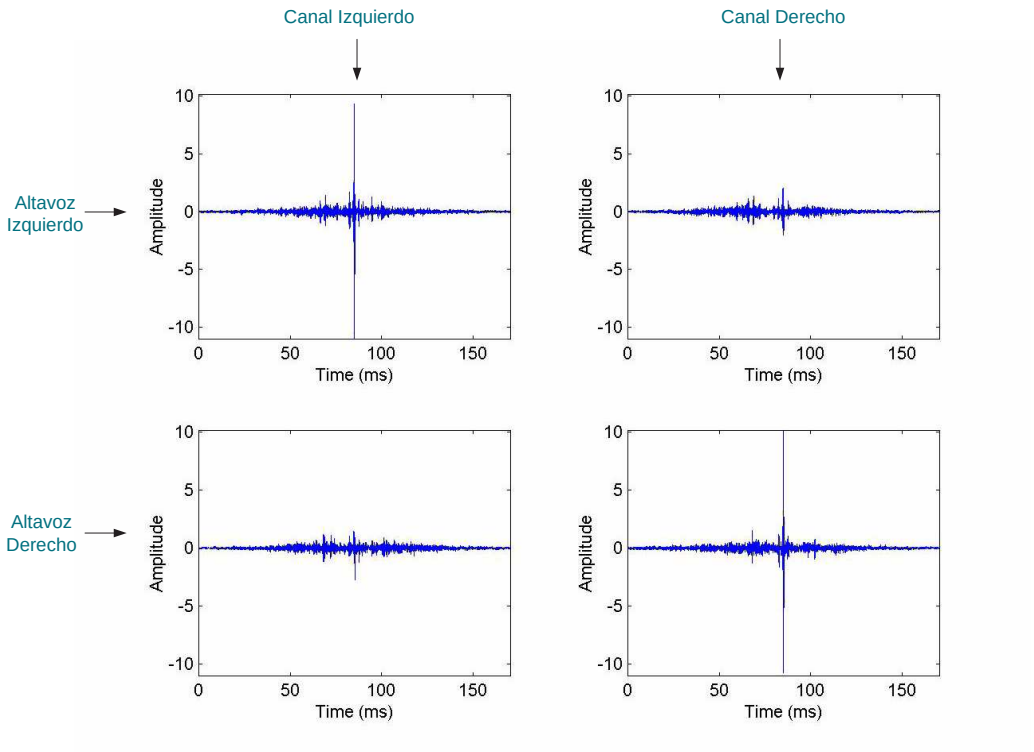


Figura 4.6: Respuestas a impulso de los filtros de reproducción biaural.

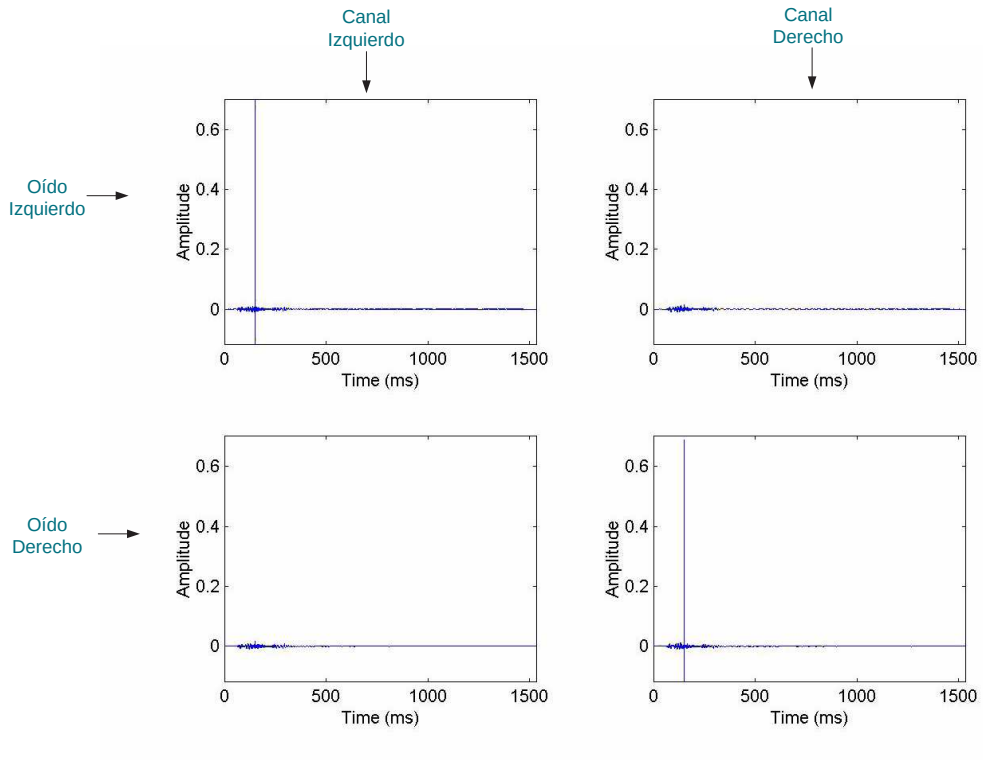


Figura 4.7: Respuestas a impulso del sistema de reproducción biaural, estimadas a partir de la respuesta del sistema de reproducción y de la respuesta de los filtros biaurales.

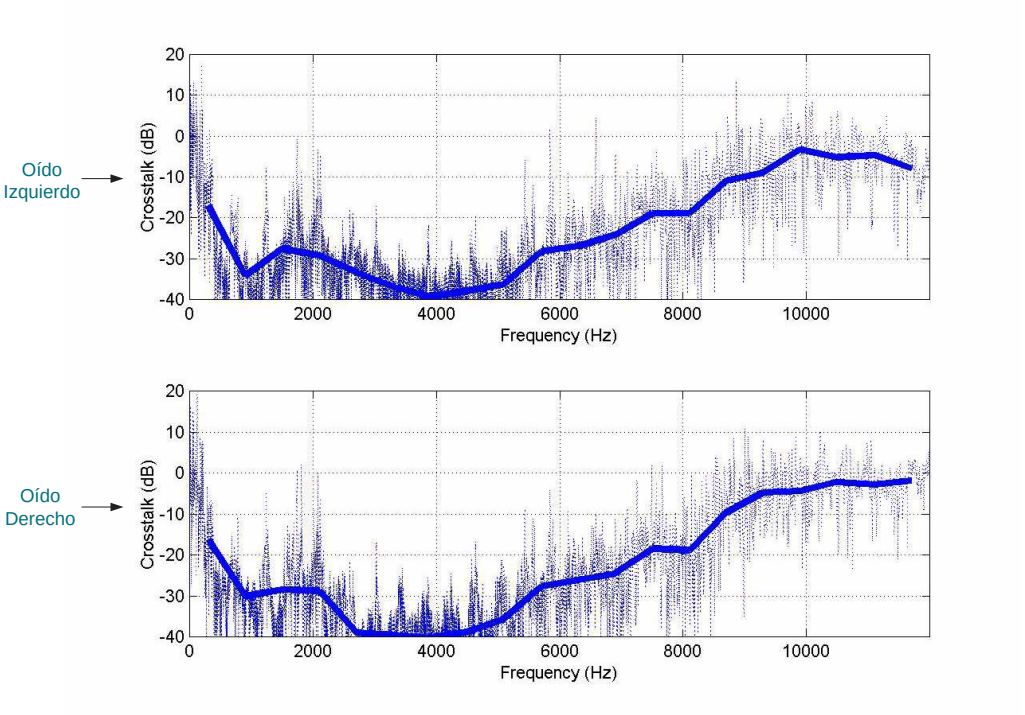


Figura 4.8: Nivel de cruce de canales correspondiente a la respuesta estimada del sistema de reproducción biaural.

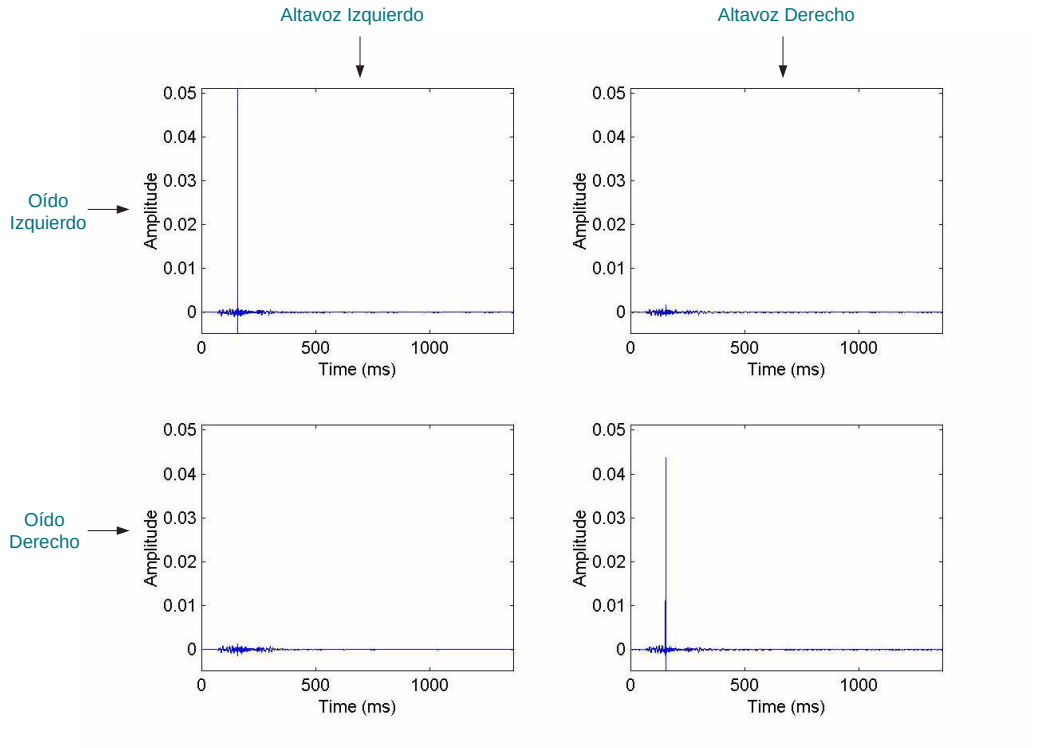


Figura 4.9: Respuestas a impulso medidas en el sistema de reproducción biaural.

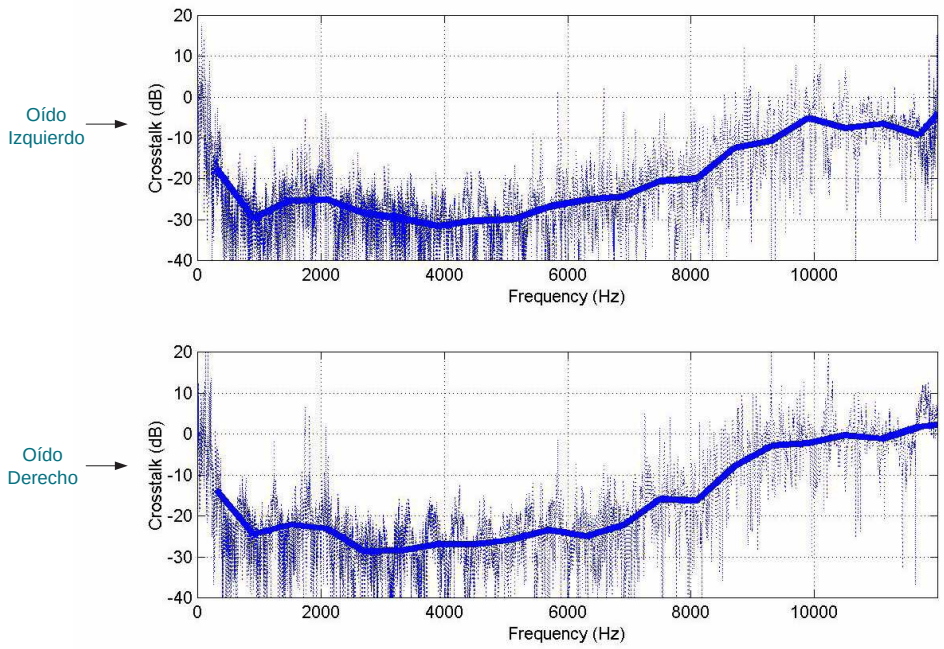


Figura 4.10: Nivel de cruce de canales medido en el sistema de reproducción biaural.

Capítulo 5

Conclusiones.

5.1. Aportaciones de esta tesis.

Para concluir, se presenta un resumen de las aportaciones de esta tesis y recomendaciones para desarrollos futuros.

Se inició con una descripción general del problema de la reproducción estereofónica, multicanal, biaural, ofreciendo de manera general una solución.

Se dió como referencia o punto de partida para este desarrollo, una breve historia de los trabajos realizados por los pioneros de la reproducción biaural tales como sistemas de reproducción matricial, técnicas de grabación multicanal, sonido envolvente surround, etc.

Se ofrecieron descripciones básicas de relación entrada-salida en sistemas lineales, convolución y correlación y su relación entre sí, transformadas de Fourier (muy importantes para el desarrollo de este trabajo), se hizo también referencia a temas como; diseño de filtros óptimos, también se muestra la manera en la que se pueden representar la Transformada de Fourier Discreta y la convolución discreta con matrices.

La parte principal de esta tesis documenta y explica a detalle la técnica de filtrado rápido basada en el uso de la transformada rápida de Fourier (FFT) y representaciones matriciales de las señales. Se mencionan los teoremas de convolución y correlación.

Se presentó también una realización práctica del algoritmo en lenguaje C, mediante el uso de bibliotecas de programación de audio en Windows (API de Windows) y bibliotecas de código abierto para calcular la FFT, por lo que se explican algunas funciones de estas librerías. Una finalidad de esto es mostrar de manera detallada, cómo se programa la tarjeta de sonido de un PC, instalada sobre la plataforma Windows. Se da también una explicación de los componentes de una tarjeta de sonido.

Una característica importante de este trabajo consiste en explicar detalladamente el código fuente del programa que realiza el procesamiento de señales, ofreciendo con esto una gran ayuda para los usuarios y/o posibles modificadores del código fuente para que puedan adaptarlo a sus necesidades o bien continuar con el desarrollo de este.

Para lograr una mejor comprensión, se ofrecieron por separado dos programas de prueba, uno llamado `rec.c` el cual graba y `snd.c` que reproduce sonido, mostrando la técnica en la

que se manipulan los datos de y para la tarjeta de sonido.

Se explicó a detalle la programación del filtrado rápido.

Se presentaron las mediciones efectuadas para verificar el buen funcionamiento del programa y apreciar su desempeño en condiciones reales. Se mostró en la parte de mediciones, una comparación entre el resultado esperado y el resultado obtenido.

En resumen, esta tesis ofrece un basto conocimiento en la implementación de programas en donde se requiera manipular señales de sonido, con herramientas de bajo costo, de buen desempeño y al alcance de cualquier usuario.

Puede ser aprovechado en usos médicos como es el caso de mediciones audiométricas sin audífonos, en desarrollo de tecnología, simulando nuevos equipos, en investigación, etc.

5.2. Recomendaciones para trabajo futuro.

Si bien este trabajo presenta un alto grado de eficacia en su desempeño, aún se le pueden hacer mejoras y aqui presentamos algunas recomendaciones para desarrollo futuro.

Este programa funciona para reproducir sonido en dos canales sin embargo como se sabe, existen sistemas de sonido que reproducen a más canales, entre estos están los sistemas 5.1 y 7.1, por lo que es conveniente se continue desarrollando y aumentando el número de canales.

Otro punto para considerar es el siguiente:

Como se mencionó anteriormente hay una ligera latencia o retardo entre la señal de entrada y la señal de salida, que una vez iniciada la reproducción no es notable al escucha, pero si se quiere hacer filtrado en vivo, es importante continuar con el desarrollo de este programa reduciendo o anulando la latencia entrada-salida.

Debido a su implementación "rústica", es decir sin la intención de demandarle demasiados recursos al PC, se corre y detiene el progama por medio de comandos en el shell de cygwin, sin embargo como todos sabemos y conocemos la tecnología de la computación avanza a pasos agigantados, aumentando considerablemente la capacidad de almacenamiento, capacidad de memoria, velocidad de procesamiento, etc. Por lo tanto esta situación ofrece una oportunidad para que programadores puedan desarrollar un front-end o interfaz gráfica de usuario, haciendo de éste un programa amigable para cualquier usuario.

Como primera versión de esta técnica de filtrado rápido, este trabajo logró su objetivo al poder ser ejecutado en ambiente Windows. Por lo que ahora se recomienda realizar una versión multiplataforma (Windows, Macintosh, Linux), basada en bibliotecas de audio de código abierto (portaudio).

Bibliografía

[1] Juan Ignacio Cervantes Cruz Tesis de Maestría en Ingeniería Eléctrica, (opc. Instrumentación UNAM.) *Cálculo de filtros óptimos para sistemas de reprodución de sonido biaural.*

[2] Chris Kiriakakis, Panagiotis Tsakalides, and Tomlinson Holman *Acquisition and Rendering Methods for Inmersive Audio.* "Surrounded by sound. January 1999"IEEE signal processing magazine.

[3] Chris Kiriakakis *Fundamental and Technological limitations of Inmersive Audio Systems.* "Proceedings of the IEEE. vol.86, No.5 May 1998"

[4] Brian W. Kernighan, Dennis M. Ritchie Ed. Prentice Hall. *El Lenguaje de Programación C.*

[5] S. Stearns, D. R. Hush, Ed. Prentice Hall (1990), *Digital sygnal análisis.*

[6] C. S. Burrus, T. W. Parks, Ed. Wiley interscience. *DFT/FFT and Convolution Algorithms, Theory and Implementation.*

[7] Charles Petzold, 5^a edición, Ed. Microsoft Press (1998), *Programing Windows.*

[8] Tim Kientzle, Adison - Wesley (1998), *A programer's guide to sound.*

[9] `http://msdn.microsoft.com`

[10] Emulador de Linux en Windows, entre otras aplicaciones contiene el compilador *gcc* `www.cygwin.com`

[11] Proyecto GNU que tiene la finalidad de desarrollar software de libre distribución y código abierto. `www.gnu.org`

[12] Collección de subrutinas o funciones en lenguaje C, util para calcular transformadas discretas de Fourier, con un tiempo mínimo de cálculo. `www.fftw.org`

[13] "SOMI XVIII Congreso de Instrumentación Acústica y Vibraciones **FOB18184**" *Realización de un Sistema de reproducción biaural en una computadora personal*