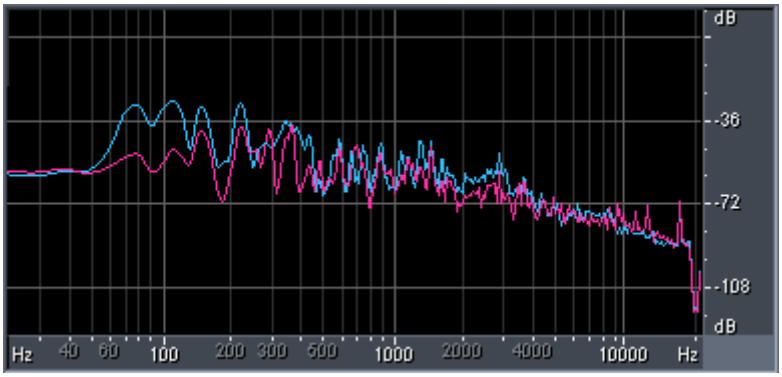
TFC

NORMALIZACIÓN DEL VOLUMEN DE ARCHIVOS DE SONIDO EN FORMATO MP3



Consultor: Antonio Bonafonte Cávez
Alumno: Gonzalo Poch Lacalle

ÍNDICE

Pág.

1.- INTRODUCCIÓN

El presente trabajo consiste en calcular la ganancia de volumen que hay que aplicar a un archivo de sonido digital para conseguir la normalización del volumen de ese archivo. Para ello, es necesario calcular la potencia de la señal original y compararla con un valor de referencia. Esto nos permitirá saber si debemos aumentar o disminuir el volumen del archivo.

La potencia de la señal de todo un archivo se calcula analizando tramos pequeños del fichero de audio y considerando los tramos de mayor potencia, en concreto, el 10% más elevado.

El oído humano no percibe los sonidos con la misma intensidad con que se producen en la realidad. Es más sensible a determinadas frecuencias y menos a otras, y siempre dentro de un rango de frecuencias que van desde los 20 Hz a los 20 KHz, siendo, además, insensible a frecuencias que se encuentren fuera de este rango. Por ello, es necesario ponderar la potencia de la señal según la frecuencia de que se trate. Esto nos dará, más que la potencia real del archivo, la potencia perceptual, de modo que el proceso de normalización podrá efectuar los ajustes de manera más precisa.

El programa de cálculo de la ganancia por canal se ha desarrollado en Visual C++ 2.0, pero sin utilizar librerías propias del entorno Windows, ya que se ha tenido presente la portabilidad a otras plataformas.

Ha formado parte de este trabajo, también, el estudio de la codificación MP3, así como de otros formatos de codificación de audio, tales como wav PCM o AU, por lo que se incluirá una breve referencia.

El presente TFC está enmarcado dentro de un proyecto más amplio, realizado por otros compañeros, que incluye la lectura de archivos MP3, su modificación para implementar la ganancia calculada según lo que aquí se refleja, y el desarrollo de una interfaz de usuario que permita aplicar la normalización a ficheros MP3.

2.- EL SONIDO: CONCEPTOS BÁSICOS

El sonido consiste en variaciones de la presión del aire, que pueden ser convertidas a una señal eléctrica mediante un micrófono. Esta señal eléctrica es una señal analógica (continua a lo largo del tiempo). En un instante determinado, el valor de dicha señal analógica es la **amplitud instantánea** (presión del sonido en ese instante). Si representamos esta señal en un gráfico en el que el eje de coordenadas sea el tiempo y el de ordenadas la amplitud de la señal, entonces hablamos de una **representación en el tiempo**.

Para representar digitalmente (numéricamente) una señal analógica, debemos convertir la amplitud en cada instante en un número que represente su valor. Cada valor constituye una muestra. La **frecuencia de muestreo** es el número de veces que hemos tomado una muestra por segundo y se mide en **Hertz** (Hz).

Un **ancho de banda** es un rango de frecuencias. Las frecuencias que las personas pueden oir están en el rango de los 20 a 20.000 Hz y se asume que este ancho de banda es el máximo rango audible. En realidad, cada persona tiene rangos de audición diferentes, pero siempre dentro del máximo audible indicado.

El **teorema de Nyquist** nos dice que el ancho de banda que puede ser representado utilizando una determinada frecuencia de muestreo es la mitad de dicha frecuencia. Esto significa que la frecuencia de muestreo debe ser mayor que dos veces el ancho de banda que queremos digitalizar. Antes hemos dicho que el oído humano percibe frecuencias únicamente en un rango de 20 a 20.000 Hz, por eso los CDs de música utilizados actualmente usan una frecuencia de muestreo de 44.100 muestras por segundo. Esto nos da un ancho de banda de 22.050 Hz, que cubre perfectamente la banda de frecuencias audibles.

Los formatos de archivo digital, entre ellos, el MP3, almacenan la información digitalizada así obtenida.

La potencia del sonido se mide en **decibelios** (dB). Un decibelio representa un ratio logarítmico entre dos valores. El decibelio se define como 10*log10(v1/v2). Un valor lineal de 2 corresponde a 3 dB y un valor lineal de 10 corresponde a 10 dB.

3.- EL FORMATO MP3

3.1.- ORIGEN DEL FORMATO

El formato MP3 fue creado por un grupo de expertos, que se autodenominó Moving Pictures Experts Group (MPEG), dedicado a desarrollar estándares genéricos para la representación codificada de vídeo, audio y combinación de ambos.

El objetivo originario era la codificación de audio y vídeo conjuntamente para su almacenamiento en soporte digital, pero pronto se encontraron muchas aplicaciones tales como la emisión en audio digital, la transmisión vía RDSI, el sonido para televisión digital, difusión vía Internet (Microsoft Netshow, Apple Quicktime), dispositivos de audio portátiles (Sony Mpman) y almacenamiento e intercambio de ficheros de música.

Durante la primera fase de trabajo del MPEG, en 1988, se concibió la especificación MPEG-1, que, en su versión de audio, consiste en tres modos de operación llamados “layers” o capas. Cada capa tiene un nivel mayor de complejidad y rendimiento, siendo la capa más compleja la “layer-3” que proporciona la mejor calidad de sonido utilizando pocos bits. MPEG-1 Layer-3 es lo que conocemos como MP3.

Entre las especificaciones evolucionadas de MPEG-1, destacan:

- MPEG-2: incorpora la codificación de la configuración de canales 5.1, utilizado hoy en día en los sistemas “cine en casa” (home cinema).

- MPEG-2 AAC (Advanced Audio Coding): soporta señales multicanal y un rango más amplio de frecuencias de muestreo.

- MPEG-4: introduce nuevas ventajas para los servicios interactivos, como videoconferencia por Internet.

3.2.- CODIFICACIÓN MP3

A continuación describiremos los principales aspectos de la codificación MP3.

3.2.1.- MODOS DE OPERACIÓN

- Mono canal: un solo canal de sonido, para transmisiones monoaurales.

- Canal dual: dos canales, pero independientes. Utilizado para tener dos versiones de audio diferentes, por ejemplo, en dos idiomas diferentes.

- Estéreo: dos canales estereofónicos.

- Joint estéreo: la información de los dos canales estéreo se combina a fin de reducir el número de bits para representar, para lo cual existen dos técnicas:

    - Intensity Stereo Coding: se combinan los dos canales estéreo y se codifica el canal resultante junto con dos coeficientes que definen cómo debe presentarse este canal para cada uno de los canales estéreo de la salida.

    - Matrix Stereo Coding: en lugar de codificar los dos canales independientemente, se codifican la suma y la diferencia de ambos canales. La suma es la suma de los canales dividida por √2, y la diferencia, la resta de canales dividida por √2. La ventaja principal de este sistema es aprovechar la similitud entre las dos señales y reducir el número de bits necesarios para representar la señal sin pérdida de datos.

3.2.2.- FRECUENCIAS DE MUESTREO

MPEG-1 define las siguientes frecuencias de muestreo: 32.000, 48.000 y 44.100 Hz. MPEG-2 utiliza 22.050, 24.000 y 16.000 Hz. Finalmente, MPEG-2.5, que es la última extensión de MPEG-2, utiliza 11.025, 12.000 y 8000 Hz.

3.2.3.- TASA DE BITS

MP3 soporta diferentes tasas de bits en un rango que va desde los 8 Kbits/s a los 320 Kbits/s. Además, se permite que la tasa de bits varíe a lo largo de los diferentes bloques (audio frames) de los que se compone un archivo MP3.

3.2.4.- ESTRUCTURA DE LA CABECERA DE UN FRAME MPEG AUDIO

Un fichero MP3 se compone de varios bloques denominados frames. Cada frame contiene una cabecera y a continuación la información de audio codificada. En el caso de los layer o capas 1 y 2, los frames son totalmente independientes, de manera que podemos cortar un fichero MPEG y reproducirlo sin problemas. Sin embargo, en el caso de la capa 3, los frames no son totalmente independientes, ya que se utiliza una técnica de “byte reservoir”, que es una especie de buffer intermedio. En el peor de los casos se necesitan nueve frames para descodificar uno.

La cabecera de frame tiene una longitud de 4 bytes. Los primeros 11 bits siempre están a 1 y se denominan sincronismo de frame. Opcionalmente, detrás de la cabecera hay una suma CRC de 16 bits. A continuación describimos los campos de dicha cabecera.

CUADRO RESUMEN DEL FORMATO DE LA CABECERA:

AAAAAAAA AAABBCCD EEEEFFGH IIJJKLMM

Letra	Longitud	Posición	Descripción
A	11	31-21	Sincronismo de frame
B	2	20,19	Versión MPEG Audio 01 – reservada 10 – MPEG Versión 2 11 – MPEG Versión 1
C	2	18,17	Capa 00 – reservada 01 – layer 3 10 – layer 2 11 – layer 1
D	1	16	Bit de protección 0 – protegido por CRC 1 – no protegido
E	4	15,12	Tasa de bits
F	2	11,10	Frecuencia de muestreo 00 – 44.100Hz 01 – 48.000 Hz 10 – 32.000 Hz 11 – reservada
G	1	9	Bit de relleno 0 – el frame no está rellenado 1 – el frame está rellenado con un slot extra
H	1	8	Bit privado (sólo informativo)
I	2	7,6	Modo canal 00 – estéreo 01 – joint estéreo 10 – canal dual 11 – mono
J	2	5,4	Extensión de modo (sólo joint stereo)
K	1	3	Bit de copyright 0 – audio sin copyright 1 – audio sin copyright
L	1	2	Bit de original 0 – copia del original 1 – original
M	2	1,0	Bit de énfasis 00 – ninguno 01 – 50/15 ms 10 – reservado 11 – CCIT J.17

3.2.5.- TÉCNICAS DE CODIFICACIÓN DE AUDIO

Los algoritmos de codificación de audio actuales, entre ellos el MP3, utilizan diferentes técnicas de codificación, entre las que podemos destacar las siguientes:

- Enmascaramiento auditivo: dado que una señal débil puede hacerse inaudible (enmascarada) por una señal más fuerte, si esta última está cercana en frecuencia a la primera, la técnica del enmascaramiento auditivo consiste en no registrar las señales enmascaradas. Este fenómeno descrito para el ámbito frecuencial, también se verifica en el ámbito temporal, es decir, una señal fuerte enmascara señales más débiles inmediatamente antes y después de producirse la señal fuerte.

- Codificación perceptual: consiste en eliminar componentes redundantes de la señal original, teniendo en cuenta la correlación entre las muestras. Además, se eliminan los componentes que son perceptualmente irrelevantes al oído humano.

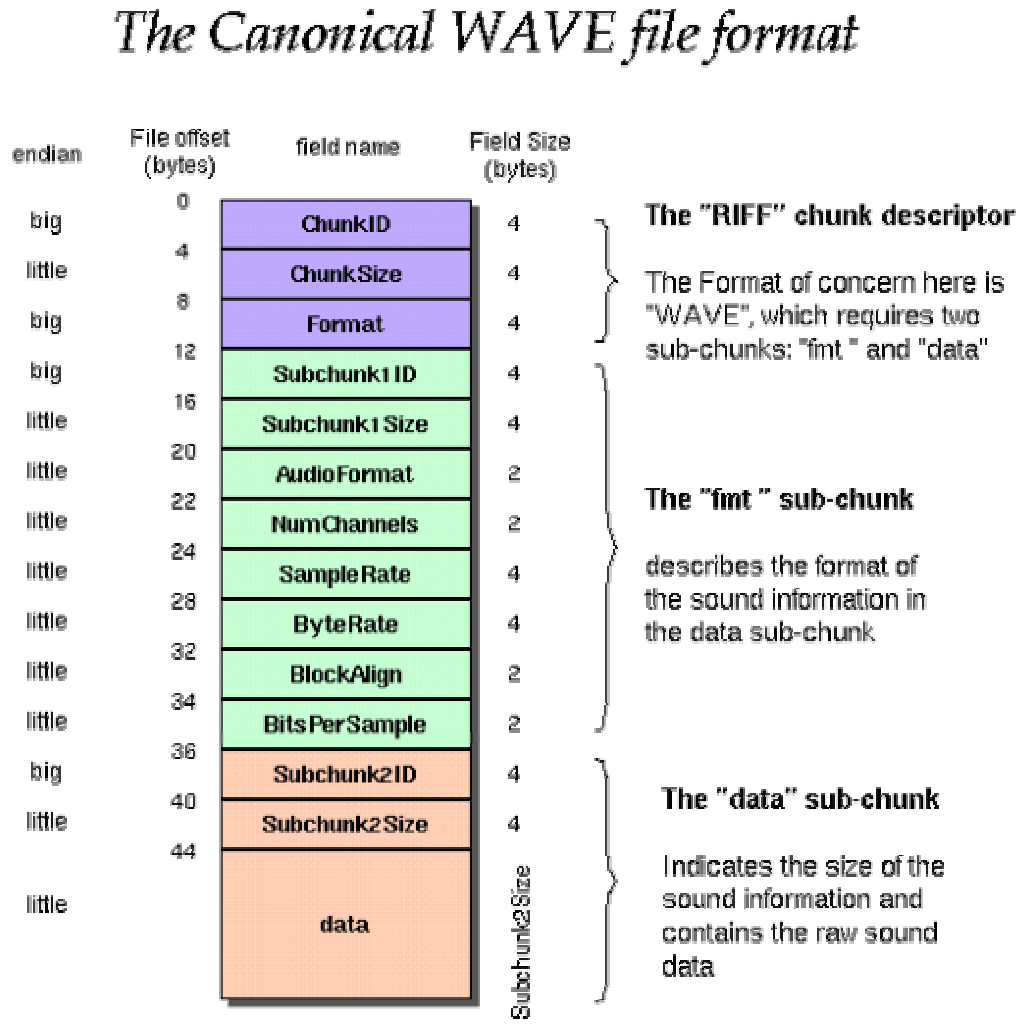
- Window switching: cuando en la señal origen hay períodos de silencio seguidos de un sonido de percusión, puede aparecer el fenómeno de pre-ecos, que consiste en la distribución de un error de cuantificación en todo el bloque codificado. Mediante esta técnica, se utilizan bloques de menor tamaño para reducir los efectos de la propagación de errores de cuantificación.

4.- EL FORMATO WAV

El archivo .wav, consiste en una cabecera y, a continuación, los datos de sonido, que son simplemente valores proporcionales a la señal eléctrica que se aplica a los altavoces. No utiliza, por tanto, ninguna técnica de compresión. Una de la codificaciones que más se utiliza en los archivos .wav es PCM (Pulse Code Module), si bien WAV permite otras codificaciones como ley A, ley Mu, codificación diferencial e incluso GSM.

Para realizar las pruebas del programa de cálculo de la ganancia por canal, se han utilizado archivos en este formato.

La cabecera de un fichero wav se compone de tres bloques, el bloque RIFF, el bloque fmt y el bloque data, como se aprecia en el siguiente esquema:



A continuación presentamos una tabla que contiene la descripción de los campos de cada bloque:

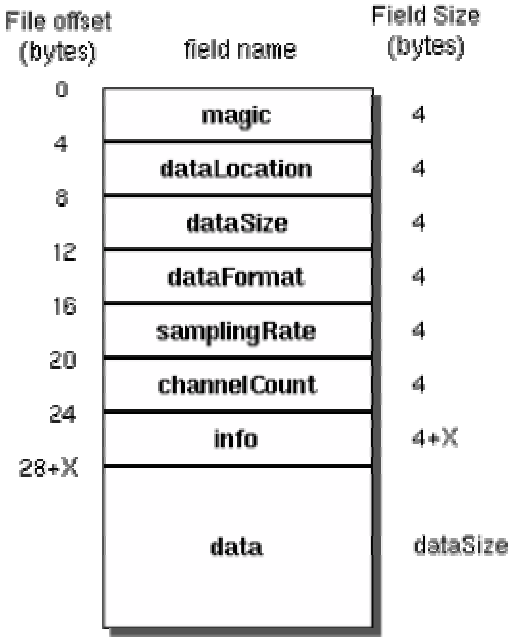
Desplazamiento	Tamaño	Nombre	Descripción
0	4	ChunkID	Contiene las letras “RIFF”
4	4	ChunkSize	4+(8+SubChunk1Size)+(8+SubChunk2Size)
8	4	Format	Contiene las letras “WAVE”
12	4	SubChunk1ID	Contiene las letras “fmt “
16	4	SubChunk1Size	Valor 16
20	2	AudioFormat	Valor 1
22	2	NumChannels	Mono = 1, Estéreo = 2
24	4	SampleRate	Frecuencia de muestreo
28	4	ByteRate	SampleRate*NumChannels*BitsPerSample/8
32	2	BlockAlign	NumChannels*BitsPerSample/8
34	2	BitsPerSample	Número de bits por muestra
36	4	SubChunk2ID	Contiene las letras “data”
40	4	SubChunk2Size	NumSamples*NumChannels*BitsPerSample/8
44	*	Data	Datos de sonido

Dentro de los datos de sonido, en el caso de que se registren dos canales, la primera muestra corresponde al canal izquierdo y la segunda, al canal derecho. Las muestras de 8 bits se almacenan como bytes sin signo. Las muestras de 16 bits son enteros con signo almacenados en complemento a 2.

5.- EL FORMATO AU

Es un formato utilizado por las plataformas UNIX y se corresponde con los archivos cuya extensión es .au o .snd. El formato de los datos varía en función del campo dataFormat. Si el formato es linear-16, sigue el mismo formato que la del wav PCM.

La cabecera del archivo tiene la disposición que se refleja en el siguiente esquema:



Los campos de la cabecera se reflejan en la siguiente tabla:

Desplazamiento	Tamaño	Nombre	Descripción
0	4	magic	Contiene las letras “.snd”
4	4	dataLocation	Desplazamiento desde el inicio del fichero a los datos de audio
8	4	dataSize	Número de bytes en los datos de audio
12	4	dataFormat	Formato de los datos
16	4	sampligRate	Frecuencia de muestreo
20	4	channelCount	Número de canales en el fichero de audio
24	4+X	info	Comentarios relacionados con el audio
28+X	*	data	Datos de audio

6.- CÁLCULO DE LA GANANCIA DE UN ARCHIVO AUDIO DIGITAL

El objetivo del desarrollo del programa es calcular la ganancia que hay que aplicar a un archivo de audio digital para que éste tenga una potencia media de referencia, es decir, para normalizar el volumen del archivo.

Hay que tener en cuenta que, como no se dispone de la funcionalidad de lectura de un archivo MP3, se parte de un archivo .wav y se aplica, a cada parte leída, la función FFT (Fast Fourier Transform) para poder trabajar en el ámbito frecuencial, que es como trabajará el API proveniente del otro TFC.

6.1.- LECTURA DE UN ARCHIVO .WAV

Mediante esta funcionalidad, se lee el archivo .wav pasado como parámetro del programa. Este módulo se encarga de verificar que el archivo esté en el formato correcto (PCM) y de tratarlo por bloques. Cada uno de estos bloques será analizado por el resto de módulos del programa con el fin de obtener la potencia media del fichero.

6.2.- TRANSFORMACIÓN DE LA SEÑAL AL ÁMBITO FRECUENCIAL

Para cada uno de los bloques a tratar, se aplica la función FFT de manera independiente para cada uno de los canales. Como implementación de la función FFT se ha escogido la publicada por Don Cross en su página web correspondiente a las transformadas de Fourier.

Esta implementación exige una entrada de un número de puntos que sea una potencia de 2. La salida es un array de números complejos desglosando los coeficientes reales e imaginarios.

Como, por una parte, se está trabajando con archivos con una frecuencia de muestreo de 44.100 Hz y, por otra, se está utilizando una FFT con 4.096 puntos, y al ser la entrada números reales, tenemos que:

a) El primer elemento del array de salida corresponde a la frecuencia 44.100*1/4.096, o sea, 10,76 Hz. El segundo elemento corresponde a la frecuencia 44.100*2/4.096 = 21,53 Hz. En general, el elemento i corresponde a la frecuencia 44.100*i/4.096 = 10,76 i Hz.

b) Al llegar al índice i = 2.048, se obtiene la frecuencia de Nyquist, que siempre es la mitad de la frecuencia de muestreo. Por tanto, ignoramos los elementos desde el 2.049 al 4.095.

c) La potencia para cada frecuencia se calcula de la siguiente manera:

   Pot[i] = ((RealOut[i]/n)*(RealOut[i]/n))+((ImagOut[i]/n)*(ImagOut[i]/n))

donde:

Pot[i] es la potencia de la señal para la frecuencia 10,76 i,
n es el número de muestras,
RealOut[i] es el coeficiente real del número complejo,
ImagOut[i] es el coeficiente imaginario del número complejo.

Mediante este procedimiento, Pot[i] corresponde a la potencia de la señal original. Podemos ponderar esta potencia, en las diferentes frecuencias, para compensar la sensibilidad del oído humano, en la forma que luego se explicará.

Para calcular la potencia del tramo, se suman las Pot[i] y para obtener el valor en decibelios, utilizamos la siguiente fórmula:

Potencia en dB = 10.0*log10(2*suma(Pot[i]))

Se ha comprobado que el valor es similar al obtenido en el ámbito temporal, calculado como sigue:

Potencia en dB = 10.0*log10(suma(Pot[i])/n)

donde:

Pot[i] = $(muestra_i)^2$,
n = número de muestras del tramo.

Tenemos interés en realizar la medida en frecuencia por dos motivos. Primero, porque los ficheros MP3 que es el motivo de este proyecto, ofrecen una representación en frecuencia. El trabajar directamente en frecuencia ahorrará tener que descodificar y normalizar en el tiempo, según el proceso antes explicado. Y, en segundo lugar, al tener la representación frecuencial, podremos ponderar a voluntad las componentes frecuenciales con gran facilidad.

6.3.- HISTOGRAMA DE POTENCIAS

Cada una de las potencias calculadas por tramo y canal, se contabiliza en un histograma de potencias en decibelios. El histograma es un array de dos dimensiones, una dimensión corresponde a las diferentes potencias y la otra, a cada uno de los canales.

El histograma tiene una dimensión de 900 posiciones por canal, es decir, la primera posición de este array corresponde al número de veces en que se ha contabilizado un tramo con una potencia entre 0 y 0,1 dB. El segundo elemento es el número de veces en que se ha contabilizado un tramo con una potencia entre 0,1 y 0,2 dB, y así sucesivamente hasta llegar a los 90 dB.

6.4.- CÁLCULO DE LA POTENCIA POR CANAL

Una vez se ha completado el histograma de potencias, el cálculo de la potencia por canal considera únicamente aquellas potencias que se encuentran en el 10% más frecuente, de acuerdo con el histograma detallado en el punto anterior. La potencia final se obtiene de la media ponderada de dichas potencias más frecuentes.

6.5.- CÁLCULO DE LA GANANCIA POR CANAL

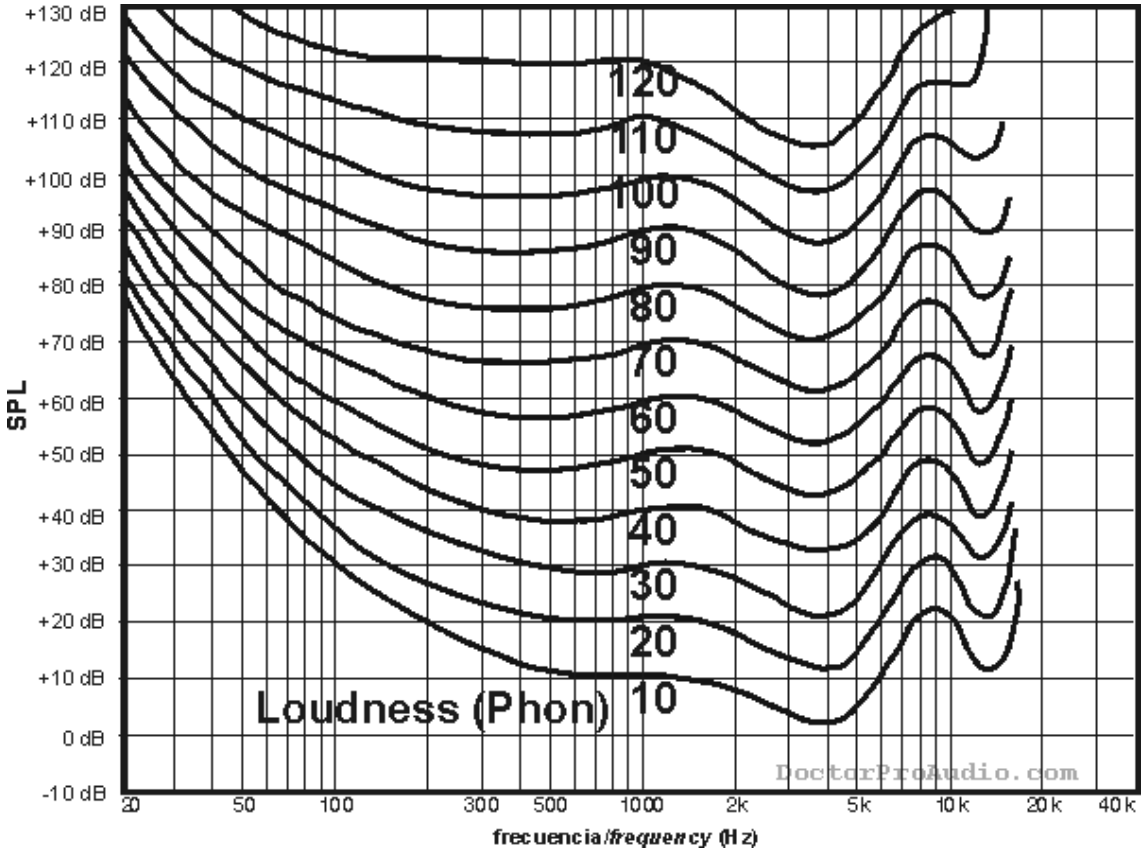
Teniendo en cuenta el valor de la potencia de referencia y la potencia calculada, obtenemos la diferencia en dB. Para convertir la ganancia en dB a ganancia lineal, deberíamos utilizar la siguiente fórmula:

$$\text{Ganancia lineal} = 10^{(\text{Ganancia en dB}/20)}$$

7.- PONDERACIÓN DE LA POTENCIA CALCULADA SEGÚN LA FRECUENCIA

Las personas no oímos las frecuencias de sonido con el mismo nivel. Es decir, nuestros oídos son más sensibles a algunas frecuencias y menos a otras. Las curvas de Fletcher-Munson, o curvas de igual sonoridad, representan el nivel que deberán tener las diferentes frecuencias para que el oído humano las perciba como si tuvieran el mismo nivel.

A continuación mostramos un gráfico con diferentes curvas isofónicas:



Como puede verse, no existe una única curva, sino diferentes curvas según el volumen de la señal y, a niveles altos de volumen, las curvas tienden a aplanarse.

Para ponderar las potencias de la señal original, se ha obtenido manualmente la tabla que se muestra a continuación, que asocia frecuencias con un factor divisor que compensa la diferente sensibilidad del oído humano a las frecuencias. Esta tabla parte de la curva isofónica en el nivel de 80 fones, que corresponde aproximadamente al volumen medio de una canción de un CD master.

Frecuencia	SPL	Diferencia en dB	Factor divisor
20	113,750	33,750	48,70
23	110,000	30,000	31,62
25	107,500	27,500	23,71
30	102,500	22,500	13,34
35	100,000	20,000	10,00
40	96,250	16,250	6,49
45	93,750	13,750	4,87
50	92,500	12,500	4,22
60	91,250	11,250	3,65
70	88,750	8,750	2,74
80	86,250	6,250	2,05
90	85,000	5,000	1,78
100	83,750	3,750	1,54
150	80,000	0,000	1,00
200	77,500	-2,500	0,75
300	76,250	-3,750	0,65
400	76,000	-4,000	0,63
500	76,100	-3,900	0,64
600	76,250	-3,750	0,65
700	77,500	-2,500	0,75
800	78,500	-1,500	0,84
900	79,000	-1,000	0,89
1.000	80,000	0,000	1,00
1.400	81,000	1,000	1,12
1.800	77,500	-2,500	0,75
2.000	76,250	-3,750	0,65
2.500	72,500	-7,500	0,42
3.000	71,000	-9,000	0,35
3.400	70,000	-10,000	0,32
4.000	71,250	-8,750	0,37
4.500	72,500	-7,500	0,42
5.000	73,750	-6,250	0,49
6.000	76,250	-3,750	0,65
7.000	83,125	3,125	1,43
8.000	87,500	7,500	2,37
9.000	87,500	7,500	2,37
10.000	85,375	5,375	1,86
13.000	76,250	-3,750	0,65
16.000	80,000	0,000	1,00
17.000	85,375	5,375	1,86
20.000	110,000	30,000	31,62

De esta manera, la potencia perceptual la podemos calcular como sigue:

PotPerceptual[i] = Pot[i]/Filtro[i]

donde Pot[i] es la potencia de la señal original y Filtro[i] es el factor por el que hay que dividir la potencia original para compensar la sensibilidad del oído humano a las frecuencias del sonido de acuerdo con la tabla anterior.

8.- POTENCIA DE REFERENCIA

La potencia de referencia la debe proporcionar el usuario. Actualmente no existe un estándar en la potencia de sonido en la industria de la música. Al contrario, hay informes (www.digido.com/integrated.com) que muestran variaciones de 10 a 12 dB. Donde sí se ha llegado a un cierto grado de consenso es en la industria del cine, porque se ha estandarizado un valor en torno a los 83 dB. Este valor fue propuesto en los años 70 por Ioan Allen, de los laboratorios Dolby. Así pues, la potencia de referencia para la normalización que se utilizará en caso de que el usuario no haya especificado ninguna será de 83 dB.

9.- CONCLUSIONES

Una de las partes más importantes dentro del proceso de normalización del volumen de un archivo MP3, es el cálculo de la ganancia. Multiplicando los valores de la señal original por esta ganancia obtenemos el archivo normalizado, teniendo en cuenta que no debemos sobrepasar el límite máximo representable (clipping).

Trabajar con archivos MP3 nos permite obtener los valores de la señal en frecuencia, por lo que podemos tomar en consideración, además de la potencia original de la señal, el comportamiento del oído humano a diferentes frecuencias.

Durante las pruebas realizadas con el programa de cálculo de la ganancia, confirmamos que la potencia media utilizada en grabaciones de CDs comerciales no es estándar. Además, se observa que existen diferencias considerables entre los CDs actuales (con tendencia a ser registrados con volumen alto) y los de hace algunos años.

Finalmente, podemos decir que el proceso de normalización no garantiza un volumen constante a lo largo de una canción, sino que únicamente se trata de un valor medio. Un ejemplo extremo en el que el proceso de normalización puede no darnos resultados satisfactorios es la música clásica, en que se combinan fases altas y bajas de volumen, silencios y subidas extremas.

10.- BIBLIOGRAFÍA

- Xarxes de Computadors I, Jordi Iñigo Griera, Universitat Oberta de Catalunya, Ed. Febrero 1999.

- An Introduction to MPEG Layer-3, EBU TECHICAL REVIEW, Junio 2000.

- Audio Coding: 3-dimensional stereo and presence, David Robinson, Universidad de Essex, Enero 2002.

- MPEG Digital Audio Coding, Peter Noll, IEEE Signal Processing Magazine, Septiembre 1997.

- MP3 and AAC Explained, Karlheinz Brandenburg.

- Sun .au sound file format, Paul Bourke, Noviembre 2001, http://astronomy.swin.edu.au/~pbourke/dataformats/au

- DSP Theory Introduced, José Mª Catena, 2000.

- MIXFFT FAQ page, Jens J. Nielsens, http://hjem.get2net.dk/jjn/fftfaq.htm

- Fast Fourier Transforms, Don Cross, Febrero 2000.

- Fletcher-Munson Equal Loudness Curves, ACS Articles, Diciembre 2001, http://www.allchurchsound.com/acs/edart/fmelc.html

- Acoustic Noise Suppresion for Speech Signals Using Auditory Masking Effects, Joachim Thiemann.

- Correlating Sound Measurements and the Human Hearing, www.norsonic.com/web_pages/correlation.html

- How to Make Better Recordings in the 21st Century, Bob Katz, Septiembre 2000, www.digido.com/integrated.html

ANEXO: PROGRAMA CÁLCULO GANANCIA

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "fourier.h"

/* Nro. de milisegundos que tiene cada trozo analizado */
#define MSEG_TRAMO 40

#define TROZO 32768  // Trozo de música que leemos cada vez

#define N_FFT 4096 // Nro. de puntos de la FFT (debe ser una potencia
de 2)

// Definiciones estándar

typedef short int WORD;
typedef int DWORD;

/*    Información    sobre    el    formato    PCM:    http://ccrma-
www.stanford.edu/courses/422/projects/WaveFormat */

typedef struct {
      char ChunkID[4];
      DWORD Chunksize;
      char Format[4];
} RIFFChunk;

typedef struct {
      char Subchunk1ID[4];
      DWORD Subchunk1Size;
      WORD AudioFormat;
      WORD NumChannels;
      DWORD SampleRate;
      DWORD ByteRate;
      WORD BlockAlign;
      WORD BitsPerSample;
} FmtChunk;

FmtChunk fmt;  // Formato del PCM

typedef struct {
      char Subchunk2ID[4];
      DWORD Subchunk2Size;
} DataHeaderChunk;

/* Conversión de 2 bytes a un entero */
union byte_int {
      char byte[2];
      WORD  ent;
};

/* Histograma de potencias por canal */
long hist[1300][2];

/* Valor máximo y mínimo del fichero .wav */
int max=0,min=32000;
```

```
// Tabla de la curva de igual sonoridad para 80 fones

struct tabla_igual_sonoridad {
      int frec;
      float factor;
} tis[43]= {
     0, 48.7,
      20,48.7,
      23,31.62,
      25,23.71,
      30,13.34,
      35,10,
      40,6.49,
      45,4.87,
      50,4.22,
      60,3.65,
      70,2.74,
      80,2.05,
      90,1.78,
      100,1.54,
      150,1,
      200,0.75,
      300,0.65,
      400,0.63,
      500,0.64,
      600,0.65,
      700,0.75,
      800,0.84,
      900,0.89,
      1000,1,
      1400,1.12,
      1800,0.75,
      2000,0.65,
      2500,0.42,
      3000,0.35,
      3400,0.32,
      4000,0.37,
      4500,0.42,
      5000,0.49,
      6000,0.65,
      7000,1.43,
      8000,2.37,
      9000,2.37,
      10000,1.86,
      13000,0.65,
      16000,1,
      17000,1.86,
      20000,31.62,
      100000,31.62};

/*
filtro: Calcula el factor ponderado de la potencia para una frecuencia
determinada
*/

float filtro(float frecuencia) {
      int i;
      float d;
      for(i=43; tis[i].frec>frecuencia; i--);
```

```
      if (i==42)
         d=31.62;
      else
         if (i==0)
                  d=48.7;
         else
                  d=tis[i].factor+((tis[i+1].factor-
tis[i].factor)*(frecuencia-tis[i].frec))/(tis[i+1].frec-tis[i].frec);

      return d;
}


/*
trataTrozo: Calcula la potencia media de un tramo de n bytes y
actualiza el
                  histograma.
*/
void trataTrozo(char *musica, size_t n) {
   size_t i;
   union byte_int t;
   int j;
   float v;
   float db;
   float pot;
   float RealIn[N_FFT], ImagIn[N_FFT], RealOut[N_FFT], ImagOut[N_FFT];

   // Canal izquierdo
   j=0;
   for(i=0; i<n; i+=fmt.BitsPerSample/4) {
      // Valor de una muestra
      if (fmt.BitsPerSample==16) {
            t.byte[0]=musica[i];
            t.byte[1]=musica[i+1];
            RealIn[j]=(float) t.ent;
        }
        if (fmt.BitsPerSample==8)
           RealIn[j]=(float) musica[i];
        ImagIn[j++]=0.0;

      if (t.ent>max) max=t.ent; /* Cálculo del valor máximo */
      if (t.ent<min) min=t.ent; /* Valor mínimo */
   }
   // Rellenar con 0's el resto del array de puntos de la FFT si no
hay suficientes
   for(i=j; j<N_FFT; j++) {
      RealIn[j]=0.0;
        ImagIn[j]=0.0;
   }


   fft_float(j,0,RealIn,ImagIn,RealOut,ImagOut);

   pot=0.0;
   for(i=1; i<=(j/2); i++)
       pot+=filtro((float)
(i*fmt.SampleRate)/j)/((RealOut[i]/j)*(RealOut[i]/j))+((ImagOut[i]/j)*
(ImagOut[i]/j));
   i--;

   if(pot==0.0)
```

```
     db=0.0;
   else
     db=10.0*log10(2*pot);
   if(db<0) db=0.0;

   if (db<130.0) {
      i=db*10;
      hist[i][0]++;
      }
   else {
      printf("\nError: el promedio es de %f dB.",db);
      exit(1);
   }

   if (fmt.NumChannels==1) return;  // Si es un archivo mono, ya hemos
acabado

   // Canal derecho
   j=0;
   for(i=2; i<n; i+=fmt.BitsPerSample/4) {
      t.byte[0]=musica[i];
      t.byte[1]=musica[i+1];
      RealIn[j]=(float) t.ent;   // Valor de una muestra
        ImagIn[j++]=0.0;
      if (t.ent>max) max=t.ent; // Cálculo del valor máximo
      if (t.ent<min) min=t.ent; // Valor mínimo
   }
   for(i=j; j<N_FFT; j++) {
      RealIn[j]=0.0;
        ImagIn[j]=0.0;
   }


   fft_float(j,0,RealIn,ImagIn,RealOut,ImagOut);

   pot=0.0;
   for(i=1; i<=(j/2); i++)
       pot+=filtro((float)
(i*fmt.SampleRate)/j)/((RealOut[i]/j)*(RealOut[i]/j))+((ImagOut[i]/j)*
(ImagOut[i]/j));
   i--;

   if(pot==0.0)
     db=0.0;
   else
     db=10.0*log10(2*pot);
   if(db<0) db=0.0;

   if (db<130.0) {
      i=db*10;
      hist[i][1]++;
      }
   else {
      printf("\nError: el promedio es de %f dB.",db);
      exit(1);
   }

}

/*
```

```
calculaHistograma: partir los bytes de música leídos en trozos adecuados para
                           generar N_FFT puntos (función FFT)
*/
void calculaHistograma(char *musica, size_t leidos) {
   size_t i,faltan;
   size_t bloque=N_FFT*(fmt.BitsPerSample/8)*fmt.NumChannels;

   faltan=leidos;
   i=0;
   while (faltan>=bloque) {
      trataTrozo(musica+(bloque*i),bloque);
      i++;
      faltan-=bloque;
   }
   if (faltan>0)
      trataTrozo(musica+(bloque*i),faltan);

}

/*
calcPotencia: calcular la potencia a partir de los histogramas por canal
*/
float calcPotencia(int num_canal) {
   long tope,aux_tope;
   int i;
   float promedio;

   tope=0;
   for(i=0; i<1300; i++)
     tope+=hist[i][num_canal];
   tope/=10;  // Coger el 10% más frecuente
   aux_tope=tope;

   promedio=0.0;
   i=1299;
   while (tope>0) {
     if (hist[i][num_canal]>0) {
            if (tope>hist[i][num_canal]) {
                  tope-=hist[i][num_canal];
                  promedio+=hist[i][num_canal]*(i/10.0);
            }
            else {
                  promedio+=tope*(i/10.0);
                  tope=0;
            }
     }
     i--;
   }
   promedio/=aux_tope;

   return promedio;
}

// Programa principal
void main(int argc,char *argv[]) {
      FILE *f;
      size_t leidos,i;
      int fin=0;
      RIFFChunk r;
```

```
      DataHeaderChunk DataHeader;
      char musica[TROZO];
      int pot_referencia;
      float pot;

      printf("\nTFC Normalización.\n");
      printf("\nCálculo de la ganancia por canal.");
      printf("\nAutor: Gonzalo Poch Lacalle\n");
      if (argc<2) {
         printf("\nUso: Normaliz fichero_entrada [potencia_en_db]");
         exit(1);
      }

      // Abrir el fichero .wav a calcular la ganancia
      if ((f=fopen(argv[1],"rb"))==NULL) {
         printf("\nError al abrir el fichero de entrada %s",argv[1]);
         exit(1);
      }

      // Obtener la potencia que quiere el usuario
      if (argc<3)
            pot_referencia=0;
      else
            pot_referencia=atoi(argv[2]);
      if(pot_referencia==0)
                  pot_referencia=83;
      if ((pot_referencia<0) || (pot_referencia>90)) {
         printf("\nLa potencia debe ser un valor entre 0 y 90 dB.");
         exit(1);
      }

      /* Inicializar array histograma de potencias */
      for(i=0; i<1300; i++) {
         hist[i][0]=0L;
         hist[i][1]=0L;
         }

    // Leer los datos de cabecera
      fread(&r,sizeof(RIFFChunk),1,f);
      fread(&fmt,sizeof(FmtChunk),1,f);
      fread(&DataHeader,sizeof(DataHeaderChunk),1,f);

      // Verificar cabecera leída
      if (memcmp(r.ChunkID,"RIFF",4)) {
         printf("\nNo es un fichero RIFF.");
         exit(1);
      }
      if (memcmp(r.Format,"WAVE",4)) {
         printf("\nNo es un fichero WAVE.");
         exit(1);
      }
      if (memcmp(fmt.Subchunk1ID,"fmt ",4)) {
         printf("\nSubchunk1ID:    %s    No    es    un    fichero    PCM
estándar.",fmt.Subchunk1ID);
         exit(1);
      }
      if (memcmp(DataHeader.Subchunk2ID,"data",4)) {
         printf("\nSubchunk2ID:    %s    No    es    un    fichero    PCM
estándar.",DataHeader.Subchunk2ID);
         exit(1);
      }
```

```
      /* Mostrar datos de la cabecera y parámetros de entrada */
      printf("\nFichero: %s",argv[1]);
      printf("\nPotencia de referencia para la normalización: %d
dB.\n",pot_referencia);
      printf("\nTamaño: %ld",fmt.Subchunk1Size);
      printf("\nAudioformat: %d",fmt.AudioFormat);
      printf("\nNum. Channels: %d",fmt.NumChannels);
      printf("\nSample Rate: %ld",fmt.SampleRate);
      printf("\nByte Rate: %ld",fmt.ByteRate);
      printf("\nBlock Align: %d",fmt.BlockAlign);
      printf("\nBits per sample: %d",fmt.BitsPerSample);
      printf("\nSubchunk2size: %ld",DataHeader.Subchunk2Size);

      // Leer el fichero de entrada
      while (!fin){
         leidos=fread(musica,1,TROZO,f);

         calculaHistograma(musica,leidos);
         if (leidos<TROZO)
            fin=1;
      }
      fclose(f);

    // Mostrar la potencia del fichero y ganancia
      if (fmt.NumChannels==2) {
          pot=calcPotencia(0);
            printf("\n\nCanal izquierdo -> Potencia: %f Ganancia en dB:
%f ",pot,(float) pot_referencia-pot);
          pot=calcPotencia(1);
            printf("\nCanal derecho -> Potencia: %f Ganancia en dB: %f
",pot,(float) pot_referencia-pot);
      }
      else {
          pot=calcPotencia(0);
            printf("\n\nPotencia: %f Ganancia en dB: %f ",pot,(float)
pot_referencia-pot);
      }
      // Mostrar los valores máximo y mínimo
      printf("\nValor maximo: %d Valor minimo: %d",max,min);


}
```