5. Referencias

[1]. *"Audio Fingerprinting: Concepts And Applications"*, Pedro Cano, Eloi Batlle, Emilia Gomez, Leandro de C.T.Gomes, and Madeleine Bonnet ; Studies in Computational Intelligence (SCI) 2, 233–245, 2005.

[2]. *"Mixed watermarking fingerprinting approach for integrity verification of audio recordings"*. Gómez E, Cano P, Gomes L de C T, Batlle E, Bonnet M (2002) Proc. of the Int. Telecommunications Symposium, Natal, Brazil.

[3]. *"A perceptual audio hashing algorithm: a tool for robust audio identification and information hiding",* M. Mihak y R. Venkatesan, in 4th Int. Information Hiding Workshop, Pittsburg, PA, April 2001.

[4]. "*Robust audio hashing for content identification"* Haitsma J, Kalker T, Oostveen J. Proc. of the Content-Based Multimedia Indexing, Firenze, Italy (2001)

[5]. *"Audio Watermarking and Fingerprinting: For Which Applications?",* Pedro Cano, Eloi Batlle, Emilia Gomez, Leandro de C.T.Gomes, and Madeleine Bonnet, Journal of New Music Research 32(1) pps. 65–82, 2003.

[6]. *"Digital watermarks for audio signals"* Boney, L., Tewfik, A., & Hamdy, K. (1996)*. IEEE Proceedings Multimedia*.

[7]. *"Techniques for data hiding".* Bender, W., Gruhl, D., Morimoto, N., & Lu, A., (1996)*.* IBM System Journal vol. 35, pp. 313-336.

[8]. *"Digital watermarking of audio signals using a psychoacoustic auditory model and spread-spectrum theory".* Garcia, R. A. (1999). 107th AES Convention.

[9]. *"A review of audio fingerprinting",* Cano, P., E. Batlle, T. Kalker, and J. Haitsma Journal of VLSI Signal Processing 41, 271–284, 2005

[10]. *"A Highly Robust Audio Fingerprinting System",* Jaap Haitsma, Ton Kalker, in *Proceedings of the International Symposium on Music Information Retrieval*, Paris, France, 2002.

[11]. *"Detection and Logging Advertisements Using its Sound",* J. Lourens, in Proc. of the COMSIG, Johannesburg, 1990.

[12]. *"Identification of highly distorted audio material for querying large scale databases",* F. Kurth, A. Ribbrock, and M. Clausen, in *Proc. AES 112th Int. Conv.*, Munich, Germany, May 2002.

[13]. *"Short-Term Sound Stream Characterisation for Reliable, Real-Time Occurrence Monitoring of Given Sound-Prints",* G. Richly, L. Varga, F. Kovàs, and G. Hosszú, in Proc.10th Mediterranean Electrotechnical Conference, MEleCon, 2000.

[14]. *"Content-based identification of audio material using mpeg-7 low level description",* E. Allamanche, J. Herre, O. Helmuth, B. Fröba, T. Kasten, and M. Cremer in Proc. of the Int. Symp. Of Music Information Retrieval, Indiana, USA, Oct. 2001.

[15]. *"A new approach to the automatic recognition of musical recordings",* C. Papaodysseus, G. Roussopoulos, D. Fragoulis, T. Panagopoulos, and C. Alexiou, *J. Audio Eng. Soc.*, vol. 49, no. 1/2, 2001, pp. 23–35.

[16]. *"Very Quick Audio Searching: Introducing Global Pruning to the Time-Series Active Search",* A. Kimura, K. Kashino, T. Kurozumi, and H. Murase, in Proc. of Int. Conf. on Computational Intelligence and Multimedia Applications, Salt Lake City, Utah, May 2001.

[17]. *"Modulation frequency features for audio fingerprinting*", S. Sukittanon and L. Atlas in Proc. of the ICASSP, May 2002.

[18]. *"Extracting Noise-Robust Features from Audio Data,"* C. Burges, J. Platt, and S. Jana, in Proc. of the ICASSP, Florida, USA, May 2002.

[19]. *"Robust Sound Modelling for Song Detection in Broadcast Audio*", P. Cano, E. Batlle, H. Mayer, and H. Neuschmied, in Proc. AES 112th Int. Conv., Munich, Germany, May 2002.

[20]. *"Method and article of manufacture for content-based analysis, storage, retrieval and segmentation of audio information",* T. Blum, D. Keislar, J. Wheaton, and E. Wold, U.S. Patent 5,918,223, June 1999.

[21]. *"Automatic Song Identification in Noisy Broadcast Audio,"* E. Batlle, J. Masip, and E. Guaus, in *Proc. of the SIP*, Aug. 2002.

[22]. *"Fast Subsequence Matching in Time-Series Databases",* C. Faloutsos, M. Ranganathan, and Y. Manolopoulos in Proc. On the ACM SIGMOD, Minneapolis, MN, 1994, pp. 419–429.

[23]. *Modern Information Retrieval*, R. Baeza-Yates and B. Ribeiro-Neto Addison Wesley, 1999.

[24]. *"Audio Fingerprinting: Nearest Neighbour Search in High Dimensional Binary Spaces,"* M. Miller, M. Rodriguez, and I. Cox in 5th IEEE Int. Workshop on Multimedia Signal Processing: Special session on Media Recognition, US Virgin Islands, USA, Dec. 2002.

[25]. *"System and Methods for Recognizing Sound and Music Signals in High Noise and Distortion",* A.L.-C.Wang and J. Smith II, U.S. Patent Application Publication US 2002/0083060 A1, 2002.

[26]. *"System Analysis and Performance Tuning for Broadcast Audio Fingerprinting",* E. Battle, J. Masip, P. Cano, Proc. of the 6th Int. Conference on Digital Audio Effects, London, 6-11 September 2003

[27]. *"Distortion Discriminant Analysis for Audio Fingerprinting",* Christopher J.C. Burges, John C. Platt and Soumya Jana, IEEE Transactions on Speech and Audio Processing, Vol. XX, NO. Y, Month ZZ

[28]. *"A Modulated Complex Lapped Transform and its Applications to Audio Processing"* H. Malvar IEEE Transactions on Speech and Audio Processing 1999 pag. 1421-1424

[29]. *"The Modulated Lapped Transform, Its Time-Varying Forms and Its Applications to Audio Coding Standards"* Seymour Shlien IEEE Transactions on Speech and Audio Processing Vol.5 No4 July 1997

[30]. *"Auditory masking in audio compression*," H. Malvar, in *Audio Anecdotes*, K. Greenebaum, Ed. A. K. Peters Ltd., 2001.

Anexo 1: Funciones usadas para el algoritmo de Philips

Función principal para el cálculo de la huella

```
%    archivo: fingerprint_princ.m
%
%    Función que, dado un archivo de audio, calcula su huella.
%



function F=fingerprint(audio)

Fs=44100/8;        %frecuencia de muestreo de la señal de audio
N=2048;          %numero de muestras por frame
overlap=31/32;  %factor de overlap
desplaz=floor(N-N*overlap); %desplazamiento de la ventana para cumplir
con el factor de overlap
num_band=33;    %numero de bandas en que dividimos cada frame
f_inferior=300; %frecuencia inferior para la division en bandas
f_superior=2000; %frecuencia superior para la division en bandas

ventana=hanning(N);



%Para solventar el problema de que el numero de muestras no es
%múltiplo de N, nos quedarnos con el numero entero de muestras de la
%señal mas alto posible que sea multiplo de 'desplaz'

numero_fingers=floor((length(audio)-N)/desplaz); %numero de huellas
sin contar la primera
audio=audio(1:(N+numero_fingers*desplaz));

Eperanterior=zeros(num_band,1);    %energía de las bandas del frame
anterior
m=1;            %índices para guardar los bits de los fingerprints

%Calculo del vector para la división en 33 bandas separadas
%logarítmicamente entre 300 Hz y 2000 Hz

indice=logspace(log10(f_inferior),log10(f_superior),num_band+1);
indice=round(indice*N/Fs);

%Bucle que divide la señal en tramas y las procesa

for k=0:desplaz:length(audio)-N
    frame=audio(k+1:k+N,1);
    frame=ventana.*frame;
    FRAME=abs(fft(frame));

    % Se divide la trama en bandas

    bandaSup=FRAME(indice(num_band):indice(num_band+1),1);
```

```
    for k=num_band:-1:2
        banda=FRAME(indice(k-1):indice(k),1);

        % Se llama a la función que calcula la energía de las bandas

        Eactual=calc_energia(banda);
        Esuperior=calc_energia(bandaSup);

        %Se obtiene el bit F(m,k-1) de la huella

        F(m,k-1)=bit_derivation(Eactual,Esuperior,Eperanterior(k-
1,1),Eperanterior(k,1));


        %actualizamos variables

        bandaSup=banda;
        Eperanterior(k,1)=Esuperior;

         %Para evitar coger un valor equivocado al llamar a
"bit_derivation"

        if k==2
            Eperanterior(k-1,1)=Eactual;
        end

    end

    m=m+1;

end
```

Función que calcula la energía de las bandas

```
% archivo: calc_energia.m
%
% Función que calcula la energía de una banda,
% empleada para el calculo del fingerprints

function E=calc_energ(x)

[M,N]=size(x);

if M>=N
    E=x'*x;
else
    E=x*x';
end
```

Función que calcula cada bit de la huella

```
% Archivo: bit_derivation.m
%
% Función que calcula un bit de la fingerprint
%
% Parámetros:
%   e: energía de la banda
%   e_banterior: energía de la banda anterior
%   e_tanterior: energía de la banda en el periodo anterior
%   e_tb: energía de la banda posterior en el periodo anterior

function F=bit_derivation(e,e_banterior,e_tanterior,e_tb)

ED=e-e_banterior-(e_tanterior-e_tb);

if ED>0
    F=1;
else
    F=0;
end
```

Script del análisis de falso positivo

```
% archivo: falsopositivo.m
%
% Script con todo el proceso del análisis de falso positivo
%

clear all;

%Proceso completo del análisis de falso positivo

Fs=44100/8;         %frecuencia de muestreo
alpha=0.35;         %umbral

% cargamos el bloque de 3 segundos y obtenemos su huella

audio1=readwav('pictures5512_3sg.wav');
f1=fingerprint_princ(audio1);
[m,n]=size(f1);

% cargamos el archivo con el que vamos a comparar, entero

audio2=readwav('1123075521_5512_20min.wav');
f2=fingerprint_princ(audio2);

%se comparan ambas huellas

d=busqueda(f1,f2);
d=d./(m*n);             %BER de cada comparación

% sacamos la media y la desviación estándar y llamamos a la
herramienta
% dfittool

media=mean(d)
desv=std(d)

dfittool
```

Función para leer archivos de audio

Esta función debió ser utilizada por una incompatibilidad entre los archivos tratados con Adobe Audition y la orden 'wavread' de Matlab. La presente función hace lo mismo que 'wavread'.

```
function [y,fs,wmode,fidx]=readwav(filename,mode,nmax,nskip)
%READWAV  Read a .WAV format sound file
[Y,FS,WMODE,FIDX]=(FILENAME,MODE,NMAX,NSKIP)
%
% Input Parameters:
%
%   FILENAME gives the name of the file (with optional .WAV extension)
or alternatively
%                can be the FIDX output from a previous call to
READWAV
%   MODE        specifies the following (*=default):
%
%    Scaling: 's'    Auto scale to make data peak = +-1
%             'r'    Raw unscaled data (integer values)
%             'q'    Scaled to make 0dBm0 be unity mean square
%             'p' *  Scaled to make +-1 equal full scale
%             'o'    Scale to bin centre rather than bin edge (e.g.
127 rather than 127.5 for 8 bit values)
%                     (can be combined with n+p,r,s modes)
%             'n'    Scale to negative peak rather than positive peak
(e.g. 128.5 rather than 127.5 for 8 bit values)
%                     (can be combined with o+p,r,s modes)
%     Offset: 'y' *  Correct for offset in <=8 bit PCM data
%             'z'    No offset correction
%   File I/O: 'f'    Do not close file on exit
%             'd'    Look in data directory: voicebox('dir_data')
%
%   NMAX     maximum number of samples to read (or -1 for unlimited
[default])
%   NSKIP    number of samples to skip from start of file
%              (or -1 to continue from previous read when FIDX is
given instead of FILENAME [default])
%
% Output Parameters:
%
%   Y        data matrix of dimension (samples,channels)
%   FS       sample frequency in Hz
%   WMODE    mode string needed for WRITEWAV to recreate the data file
%   FIDX     Information row vector containing the element listed
below.
%
%           (1)  file id
%           (2)  current position in file
%           (3)  dataoff    byte offset in file to start of data
%           (4)  nsamp  number of samples
%           (5)  nchan  number of channels
%           (6)  nbyte  bytes per data value
%           (7)  bits   number of bits of precision
%           (8)  code   Data format: 1=PCM, 2=ADPCM, 6=A-law, 7=Mu-law
%           (9)  fs     sample frequency
%
%   If no output parameters are specified, header information will be
printed.
%
```

```
%   For stereo data, y(:,1) is the left channel and y(:,2) the right
%
%   See also WRITEWAV.

%   *** Note on scaling ***
%   If we want to scale signal values in the range +-1 to an integer
in the
%   range [-128,127] then we have four plausible choices corresponding
to
%   scale factors of (a) 127, (b) 127.5, (c) 128 or (d) 128.5 but each
choice
%   has disadvantages.
%   For forward scaling: (c) and (d) cause clipping on inputs of +1.
%   For reverse scaling: (a) and (b) can generate output values < -1.
%   Any of these scalings can be selected via the mode input: (a) 'o',
(b) default, (c) 'on', (d) 'n'

%      Copyright (C) Mike Brookes 1998-2003
%      Version: $Id: readwav.m,v 1.5 2006/11/06 08:16:02 dmb Exp $
%
%   VOICEBOX is a MATLAB toolbox for speech processing.
%   Home page:
http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
%   This program is free software; you can redistribute it and/or
modify
%   it under the terms of the GNU General Public License as published
by
%   the Free Software Foundation; either version 2 of the License, or
%   (at your option) any later version.
%
%   This program is distributed in the hope that it will be useful,
%   but WITHOUT ANY WARRANTY; without even the implied warranty of
%   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%   GNU General Public License for more details.
%
%   You can obtain a copy of the GNU General Public License from
%   ftp://prep.ai.mit.edu/pub/gnu/COPYING-2.0 or by writing to
%   Free Software Foundation, Inc.,675 Mass Ave, Cambridge, MA 02139,
USA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%

if nargin<1 error('Usage:
[y,fs,wmode,fidx]=READWAV(filename,mode,nmax,nskip)'); end
if nargin<2 mode='p';
else mode = [mode(:).' 'p'];
end
k=find((mode>='p') & (mode<='s'));
mno=all(mode~='o');                     % scale to input limits not
output limits
sc=mode(k(1));
z=128*all(mode~='z');



info=zeros(1,9);
if ischar(filename)
    if any(mode=='d')
```

```
        filename=fullfile(voicebox('dir_data'),filename);
    end
   fid=fopen(filename,'rb','l');
   if fid == -1
      fn=[filename,'.wav'];
      fid=fopen(fn,'rb','l');
      if fid ~= -1 filename=fn; end
   end
   if fid == -1 error(sprintf('Can't open %s for input',filename));
end
   info(1)=fid;
else
   info=filename;
   fid=info(1);
end

if ~info(3)
   fseek(fid,8,-1);                        % read riff chunk
   header=fread(fid,4,'uchar');
   if header' ~= 'WAVE' fclose(fid); error(sprintf('File does not
begin with a WAVE chunck')); end

   fmt=0;
   data=0;
   while ~data                          % loop until FMT and DATA chuncks
both found
      header=fread(fid,4,'char');
      len=fread(fid,1,'ulong');
      if header' == 'fmt '                     % ******* found FMT chunk
*********
         fmt=1;
         info(8)=fread(fid,1,'ushort');         % format: 1=PCM, 6=A-
law, 7-Mu-law
         info(5)=fread(fid,1,'ushort');         % number of channels
         fs=fread(fid,1,'ulong');               % sample rate in Hz
         info(9)=fs;                            % sample rate in Hz
         rate=fread(fid,1,'ulong');             % average bytes per
second (ignore)
         align=fread(fid,1,'ushort');           % block alignment in
bytes (ignore)
         info(7)=fread(fid,1,'ushort');         % bits per sample
         fseek(fid,len-16,0);                   % skip to end of
header
         if any([1 6 7]==info(8)) info(6)=ceil(info(7)/8);
         else info(6)=1; sc='r';
         end
      elseif header' == 'data'               % ******* found DATA chunk
*********
         if ~fmt fclose(fid); error(sprintf('File %s does not contain
a FMT chunck',filename)); end
         info(4) = fix(len/(info(6)*info(5)));
         info(3)=ftell(fid);
         data=1;
      else                              % ******* found unwanted chunk
*********
         fseek(fid,len,0);
      end
   end
else
   fs=info(9);
end
```

```
if nargin<4 nskip=info(2);
elseif nskip<0 nskip=info(2);
end

ksamples=info(4)-nskip;
if nargin>2
   if nmax>=0
      ksamples=min(nmax,ksamples);
   end
elseif ~nargout
   ksamples=min(5,ksamples);
end
if ksamples>0
   info(2)=nskip+ksamples;
   pk=pow2(0.5,8*info(6))*(1+(mno/2-
all(mode~='n'))/pow2(0.5,info(7)));  % use modes o and n to determine
effective peak
   fseek(fid,info(3)+info(6)*info(5)*nskip,-1);
   nsamples=info(5)*ksamples;
   if info(6)<3
      if info(6)<2
         y=fread(fid,nsamples,'uchar');
         if info(8)==1 y=y-z;
         elseif info(8)==6
            y=pcma2lin(y,213,1);
            pk=4032+mno*64;
            pkp=pk;
         elseif info(8)==7
            y=pcmu2lin(y,1);
            pk=8031+mno*128;
            pkp=pk;
         end
      else
         y=fread(fid,nsamples,'short');
      end
   else                                        % 3 or 4 byte values
      if info(6)<4
         y=fread(fid,3*nsamples,'uchar');
         y=reshape(y,3,nsamples);
         y=([1 256 65536]*y-pow2(fix(pow2(y(3,:),-7)),24)).';
      else
         y=fread(fid,nsamples,'long');
      end
   end
   if sc ~= 'r'
      if sc=='s'
          sf=1/max(max(abs(y)),1);
      elseif sc=='p'
          sf=1/pk;
      else
         if info(8)==7
            sf=2.03761563/pk;
         else
            sf=2.03033976/pk;
         end
      end
      y=sf*y;
   else                                % mode = 'r' - output raw values
      if info(8)==1 y=y*pow2(1,info(7)-8*info(6)); end  % shift to get
```

```
the bits correct
   end

   if info(5)>1 y = reshape(y,info(5),ksamples).'; end
else
   y=[];
end

if all(mode~='f') fclose(fid); end

if nargout>2
   wmode=setstr([sc 'z'-z/128]);
   if info(8)==1                                        % PCM modes
      if ~mno wmode=[wmode 'o']; end
      if any(mode=='n') wmode=[wmode 'n']; end
      wmode=[wmode num2str(info(7))];
   elseif info(8)==6
      wmode = [wmode 'a'];
   elseif info(8)==7
      wmode = [wmode 'u'];
   end
   fidx=info;
elseif ~nargout
   codes=' '*ones(9,6); codes(1+[1 2 6 7],:)=['PCM   ';'ADPCM ';'A-law
';'Mu-law'];
   fprintf(1,'\n%d Hz sample rate\n%d channel x %d samples = %.3g
seconds\ndata type %d: %d bit %s\n',info([9 5 4]),info(4)/info(9),
info([8 7]),char(codes(1+max(0,min(8,info(8))),:)));
end
```

Función que calcula el error entre dos trozos de audio

```
% archivo: busqueda.m
%
% Función que va comparando dos huellas y obteniendo el error
% En las matrices entrada van los fingerprints correspondientes a los
3
% segundos de detección de la señal de audio (matriz de dimensión
256x32) y los fingerprints del anuncio
% (matriz de dimensión ¿x32)

function d=busqueda(detectado,anuncio)

[m1,n1]=size(detectado);
[m2,n2]=size(anuncio);

for k=1:m2-m1
    d(k)=sum(sum(abs(anuncio(k:k+m1-1,:)-detectado)));
end
```

Script del análisis de robustez

```
% Archivo: robustez.m
%
% Script para calcular la BER entre dos trozos de audio
%

clear all;
Fs=44100/8;        %frecuencia de muestreo


% cargamos el bloque de 3 segundos original y obtenemos su huella

audio1=readwav('jimmy_3sg_5512.wav');
f1=fingerprint_princ(audio1);
[m1,n1]=size(f1);

% cargamos el bloque modificado con el que vamos a comparar

audio2=readwav('jimmy_3sg_comp_5512.wav');
f2=fingerprint_princ(audio2);
[m2,n2]=size(f2);

% para tener en cuenta el hecho de que por la distorsión
% puede que los dos trozos sean de distinta duración

if m1<m2
    d=busqueda(f1,f2);
    d=d./(m1*n1);
    BER=min(d)

else if m1==m2
    BER=sum(sum(abs(f2-f1)))/(m1*n1)

    else
        d=busqueda(f2,f1);
        d=d./(m2*n2);
        BER=min(d)
    end
end
```

Anexo 2: Funciones usadas para el preprocesado de filtrado

Script del análisis de falso positivo

```
% Archivo: falsopositivo_filt.m
%
% Script que realiza el análisis de falso positivo, se diferencia del
% anterior en la inclusión de la orden filter, que filtra el audio
% antes de calcular la huella

clear all;

%Proceso completo del análisis de falso positivo

Fs=44100/8;        %frecuencia de muestreo
alpha=0.35;        %umbral

% cargamos el bloque de 3 segundos y obtenemos su huella

audio1=readwav('pictures5512_3sg.wav');
audio1filt=filter([0.99 -0.99],[1 -0.98],audio1);
f1=fingerprint_princ(audio1filt);
[m,n]=size(f1);

% cargamos el archivo con el que vamos a comparar, entero

audio2=readwav('1123075521_20min.wav');
audio2filt=filter([0.99 -0.99],[1 -0.98],audio2);
f2=fingerprint_princ(audio2filt)

%se comparan ambas huellas

d=busqueda(f1,f2);
d=d./(m*n);             %BER de cada comparación

% sacamos la media y la desviación estándar

media=mean(d)
desv=std(d)

dfittool
```

Script del análisis de robustez

```
% Archivo: robustez_filt.m
%
% Script para calcular la BER entre dos trozos de audio, también se
% diferencia del anterior en la inclusión de la orden filter
%

clear all;
Fs=44100/8;        %frecuencia de muestreo


% cargamos el bloque de 3 segundos original y obtenemos su huella

audio1=readwav('vangelis_3sg_5512.wav');
audio1filt=filter([0.99; -0.99],[1; -0.98],audio1);
f1=fingerprint_princ(audio1filt);
[m1,n1]=size(f1);

% cargamos el bloque modificado con el que vamos a comparar

audio2=readwav('vangelis_3sg_lsc-1_5512.wav');
audio2filt=filter([0.99; -0.99],[1; -0.98],audio2);
f2=fingerprint_princ(audio2filt);
[m2,n2]=size(f2);

if m1<m2
    d=busqueda(f1,f2);
    d=d./(m1*n1);
    BER=min(d)

else if m1==m2
    BER=sum(sum(abs(f2-f1)))/(m1*n1)

    else
        d=busqueda(f2,f1);
        d=d./(m2*n2);
        BER=min(d)
    end
end
```

Anexo 3: Funciones usadas en el preprocesado de DDA

Script del análisis de falso positivo

```
% Archivo: falsopositivo_dda.m
%
% Script para el análisis de falso positivo. Se diferencia de
% los anteriores en que se llama a la función preprocdda, que es la
que
% hace el preprocesado en sí mismo y en que se trabaja a 44100 Hz

clear all;

%Proceso completo del análisis de falso positivo

Fs=44100;        %frecuencia de muestreo
alpha=0.35;        %umbral

% cargamos el bloque de 3 segundos y obtenemos su huella

audio11=readwav('pictures44100_3sg.wav');
preproc1=preprocdda(audio11);

audio1=decimate(preproc1,2);        %de 11025 Hz a 5512,5
f1=fingerprint_princ(audio1);
[m,n]=size(f1);

% cargamos el archivo con el que vamos a comparar, entero

audio21=readwav('1123075521_5512_20min.wav');

preproc2=preprocdda(audio21.');

audio2=decimate(preproc2,2);        % de 11025 Hz a 5512,5
f2=fingerprint_princ(audio2);

%se comparan ambas huellas

d=busqueda(f1,f2);
d=d./(m*n);             %BER de cada comparación

% sacamos la media y la desviación estándar

media=mean(d)
desv=std(d)
```

Script del análisis de robustez

```
% Archivo: robustez_dda.m
%
% Script para calcular la BER entre dos trozos de audio con
preprocesado
% DDA. Se diferencia de los anteriores en que se cargan los archivos a
% 44100 Hz
%

clear all;


% cargamos el bloque de 3 segundos original y obtenemos su huella

audio11=readwav('vangelis44100_3sg.wav');
preproc1=preprocdda(audio11);

audio1=decimate(preproc1,2);     %se pasa de 11025 Hz a 5512 Hz

f1=fingerprint_princ(audio1);
[m1,n1]=size(f1);

% cargamos el bloque modificado con el que vamos a comparar

audio21=readwav('vangelis44100_3sg_lsc-4.wav');
preproc2=preprocdda(audio21);

audio2=decimate(preproc2,2);     %se pasa de 11025 Hz a 5512 Hz

f2=fingerprint_princ(audio2);
[m2,n2]=size(f2);

if m1<m2
    d=busqueda(f1,f2);
    d=d./(m1*n1);
    BER=min(d)

else if m1==m2
    BER=sum(sum(abs(f2-f1)))/(m1*n1)

    else
        d=busqueda(f2,f1);
        d=d./(m2*n2);
        BER=min(d)
    end
end
```

Función que hace el primer paso de preprocesado

```
% Archivo: preprocdda.m
%
% Función que realiza el primer paso de preprocesado del algoritmo DDA
%

function x=preprocdda(audio)

Fs=11025;                   %frecuencia de muestreo
N=4096;                     %numero de muestras por frame
overlap=1/2;                %factor de solapamiento
desplaz=floor(N-N*overlap); %desplazamiento para comenzar cada trama

%antes de nada se submuestrea el audio a Fs

audio11025=decimate(audio,4);

% para calcular el numero de tramas

numtramas=floor((length(audio11025)-N)/desplaz); %numero de tramas sin
contar la primera
audio11025=audio11025(1:(N+numtramas*desplaz));

%Generamos el espectro logarítmico

transf=mclt(audio11025,desplaz);

spect=10*log10(abs(transf));

%ya tenemos el espectro logarítmico, ahora hay que realizar el primer
paso
%propiamente dicho
%Primero hay que realizar la DCT

for k=0:desplaz:length(spect)-desplaz

    transdct(k+1:k+desplaz)=dct(spect(k+1:k+desplaz));

    %se filtra paso-bajo, cogiendo solo los componentes principales

    ramp=[1 0.8 0.6 0.4 0.2 zeros(1,desplaz-5)];
    transdct(k+1:k+desplaz)=transdct(k+1:k+desplaz).*ramp;

    %y se hace la transformada inversa

    A(k+1:k+desplaz)=idct(transdct(k+1:k+desplaz));

    for i=1:desplaz
        A(k+i)=A(k+i)-6;           %se bajan 6 dB
        if A(k+i)<=-70
            A(k+i)=-70;
        end
        if spect(k+i)-A(k+i)>0
            xt(k+i)=spect(k+i)-A(k+i);     %xt todavía en el dominio
mclt, ahora hay que antitransformar
```

```
        else
            xt(k+i)=0;
        end
    end
end

% se vuelve a pasar a dominio del tiempo

x=imclt(xt,desplaz);
```

Función que hace la transformada MCLT

```
% Archivo: mclt.m
%
% Función que calcula la transformada MCLT de las distintas tramas
% de una señal de audio, de longitud M

function trans=mclt(x,M)

N=2*M;
j=sqrt(-1);
n=0;
k=0;
m=0;

%Ventana de análisis

ha=zeros(2*M,1);

pa=zeros(2*M,M);
mclt=zeros(length(x),1);

%Ahora se computa la transformada

n=1:2*M;
ha=-sin(((n-1)+1/2)*pi/(2*M));

% creamos la matriz de análisis, pa(n,k)

for k=1:M
    for n=1:2*M
         pa(n,k)=ha(n)*sqrt(2/M)*cos(((n-1)+(M+1)/2)*((k-
1)+1/2)*pi/M)-j*ha(n)*sqrt(2/M)*sin(((n-1)+(M+1)/2)*((k-1)+1/2)*pi/M);
    end
end

%dividimos la señal en tramas de longitud 2M y multiplicamos por
%la matriz pa, dando lugar a transformadas de longitud M

for m=0:M:length(x)-N
    frame=x(m+1:m+N);
    trans(m+1:m+M)=pa.'*frame.';

    if m==length(x)-N                       % para mandar la última
```

```
trama
        frame=[x(m+M+1:m+N) x(1:M)];
        trans(m+M+1:m+N)=pa.'*frame.';

    end
end
```

Función que realiza la transformada MCLT inversa

```
% Archivo: imclt.m
%
% Función que realiza la transformada MCLT inversa, con tramas de
% longitud M

function y=imclt(x,M)

j=sqrt(-1);
n=0;
k=0;
m=0;
y=zeros(length(x),1);
yp=zeros(2*M,1);            % transformadas parciales
ypant=zeros(M,1);
hs=zeros(2*M,1);            %Ventana de análisis

ps=zeros(2*M,M);


%Ahora se computa la transformada

n=1:2*M;
hs=-sin(((n-1)+1/2)*pi/(2*M));

% se genera la matriz de síntesis, ps(n,k)

for k=1:M
    for n=1:2*M
         ps(n,k)=1/2*hs(n)*sqrt(2/M)*[cos(((n-1)+(M+1)/2)*((k-
1)+1/2)*pi/M)+j*sin(((n-1)+(M+1)/2)*((k-1)+1/2)*pi/M)];
    end
end

% En el algoritmo, para sacar la primera trama hace falta la última
% transformada parcial

ypant=ps*x(length(x)-M+1:length(x)).';

% se van sacando todas las tramas a partir de la transformada parcial
de
% dicha trama y de la anterior
```

```
for m=0:M:length(x)-M
    yp=ps*x(m+1:m+M).';   %resultado parcial
    y(m+1:m+M)=real(yp(1:M))+real(ypant(M+1:2*M));
    ypant=yp;
end
```

Función que hace el segundo paso de preprocesado

```
% Archivo: preprocdda2.m
%
% Función que realiza el segundo paso de preprocesado del algoritmo
DDA
%

function x=preprocdda2(audio)

Fs=11025;                    %frecuencia de muestreo
N=4096;                      %numero de muestras por frame
overlap=1/2;                 %factor de solapamiento
desplaz=floor(N-N*overlap); %desplazamiento para comenzar cada trama

%antes de nada se submuestrea el audio a Fs

audio11025=decimate(audio,4);

% para calcular el numero de tramas

numtramas=floor((length(audio11025)-N)/desplaz); %numero de tramas sin
contar la primera
audio11025=audio11025(1:(N+numtramas*desplaz));

%Generamos el espectro logarítmico

transf=mclt(audio11025,desplaz);

spect=10*log10(abs(transf));

%segundo paso

spectexp=10.^(spect/10);         % se eleva el espectro logarítmico

for k=0:desplaz:length(spect)-desplaz

    Ht(k+1:k+desplaz)=hthres(spectexp(k+1:k+desplaz),Fs);      %umbral
auditivo en u.n.
    HtdB(k+1:k+desplaz)=10*log10(Ht(k+1:k+desplaz));

    for i=1:desplaz
        if spect(k+i)-HtdB(k+i)>=0
            xt(k+i)=spect(k+i)-HtdB(k+i);

        else
            xt(k+i)=0;

        end
    end
```

```
end

% se vuelve a pasar a dominio del tiempo

x=imclt(xt,desplaz);
```

Funciones para obtener el umbral de audición

```
% HTHRES   Hearing threshold
%
% H. S. Malvar - Nov'00, (c) H. S. Malvar
%
% Syntax:  Ht = hthres(X,fs)
%
%    X = vector of frequency magnitude components
%   fs = sampling frequency, in Hz
%   Ht = vector of thresholds, in rms

function  Ht = hthres(X,fs)

Dabs  = 60;  % in dB, how much to bring down Fletcher-Munson curves;
             % it depends on the assumed playback level

Thmin = -60; % in dB, how far down can any threshold go
Thmax = 60;  % in dB, how far up can any threshold go
Rfac  = 8;   % Power reduction factor in dB for masking within the
             % same bark frequency band

 % Bark subbands upper limits

Bh = [100 200 300 400 510 630 770 920 1080 1270 1480 1720 2000 2320
2700 3150 3700 4400 5300 6400 7700 9500 12000 15500 22050];


P = X.*X; % power spectrum
TdB = 0*P; % Threshold in dB

Nbands = length(X);

f=[0:Nbands-1]'*fs/(2*(Nbands-1)); % subband center frequencies, in Hz

% Loop to generate Bark power spectrum and Bark center thresholds

dindx = Nbands*2/fs; % # of coefficients per Hz

stop = 0;

i1 = 1;

il=[]; iu=[];

for i = 1:25   % scan all 25 Bark bands
  % i-th Bark subband covers original subbands i1 to i2
```

```
   i2 = round(Bh(i)*dindx);
   if i2 > Nbands
      i2 = Nbands;

      stop = 1; % stop loop if signal bandwidth is reached

   end

   il = [il;i1];
   iu = [iu;i2];

   % Average RMS signal amplitude over Bark band [i1,i2]
   Arms = sqrt(mean(P(i1:i2)));
   % average signal amplitude over Bark band [i1,i2], in dB
   Adb = 20*log10(Arms+eps);
   % Tr = relative threshold given power level within i-th band
   Tr = Adb - Rfac; % center threshold

   Sbu(i) = Tr;

   i1 = i2+1;

   if stop break; end;

end

Lsb = i; % number of Barks covered for the given fs


% Cross-Bark spreading

Sbu = Sbu';
Sb = Sbu;

pl = -100;
pr = -100;
pli = Sbu(1);

for i = 1:Lsb

   if i < Lsb; pr = Sbu(i+1)-25; else pr = -100; end

   if pli > pl; pl = pli; end

   pl = pl - 10;

   if pl > Sb(i); Sb(i) = pl; end

   if pr > Sb(i); Sb(i) = pr; end

   pli = Sbu(i);

end

% Adjust thresholds considering absolute masking
```

```
for i = 1:Lsb
   i1 = il(i);
   i2 = iu(i);

   Tr = Sb(i);

   % Ta = Fletcher-Munson absolute threshold

   Ta = mean(thrabs(f(i1:i2)))-Dabs;

   if Tr < Ta; Tr = Ta; end

   % Clip at minimum and maximum levels

   if Tr < Thmin; Tr = Thmin; end

   if Tr > Thmax; Tr = Thmax; end

   Sb(i) = Tr; % save dB thresholds in vector Sb

   TdB(i1:i2) = Tr; % same threshold for all bands within Bark band

end

Ht = 10.^(TdB/20); % convert from power to rms levels
```

```
%THRABS   Absolute hearing threshold (Fletcher-Munson)
%
% H. S. Malvar - Nov'00, (c) H. S. Malvar
%
% Syntax:  T = thrabs(f)
%
%  f  = frequency, in Hz (or a vector of frequencies)
%  T  = corresponding absolute threshold of hearing, in dB SPL

function T=thrabs(f)


fk = f/1000;

% Approximation to the F&M curves

T = 3.64*(fk.^(-.8))-6.5*exp(-.6*(fk-3.3).^2)+1e-3*fk.^4;
```

Agradecimientos

Antes de dar por acabada esta memoria, es el momento de dedicar algunas palabras a todos aquellos que han formado parte del (largo) proceso que culmina con el presente proyecto.

A veces parece que estos años se han pasado volando, que no ha sido tanto el tiempo transcurrido, pero han sido muchas las mañanas, muchas las tardes, que se han hecho llevaderas gracias a la buena compañía y a los buenos amigos con los que he tenido la suerte de coincidir. Algunos podrán venir a la presentación, otros ya son personas de provecho y tendrán que trabajar, pero sea como sea se merecen todo mi agradecimiento por todos estos años. Y a todos los que, fuera de la escuela, se interesan por lo que hago, por como me va y confían en mí, aquí incluyo a los amigos de toda la vida (desde el instituto juntos, ¡ya son años!) y en general a todos los que han estado pendientes desde la distancia, incluso desde fuera de nuestras fronteras.

También me gustaría agradecer al tutor Dr. José Ramón Cerquides por su ayuda para sacar adelante el proyecto en momentos en los que parecía que se iba a atascar por carecer de algún material necesario para su realización (buen momento para agradecer también al señor Malvar de Microsoft, gran detalle el suyo) y a los miembros del tribunal.

Debo agradecer también su ayuda a mi compañero Alejandro Alvárez, que ha estado trabajando en un proyecto similar al mío y ha compartido conmigo su trabajo.

Y, por último y más importante a la familia, toda entera. Empezando por mi hermano, siempre a mi lado y ahora marcándome el camino a seguir, por supuesto mis padres, que me han aportado todo el apoyo y la confianza necesarios para poder afrontar esto tranquilamente, sin otras preocupaciones, sin presión y apoyándome también en todo lo que han considerado mejor para mí aunque ello les cueste tenerme lejos. Y a todos los demás, por estar siempre atentos y, aunque muchas veces no sepan exactamente en qué estoy trabajando, interesarse por saber como me va.