Trabajo Práctico 2: Análisis Armónico

1. Speaker Recognition

En este trabajo intentaremos un simple identificador personal a partir del habla. Nos basaremos en el trabajo [2]. En dicho trabajo, se analizan muestras de voz de distintas personas utilizando análisis cepstral de mel-frecuencias y luego se utiliza cuantificación vectorial (ver [3]) para la caracterización de cada individuo.

1. Consiga muestras de habla de varios individuos. Intente crear al menos 3 fragmentos de 1-2 segundos de audio provenientes de al menos 10 individuos distintos. Etiquete cada fragmento con algo identifique al individuo para control. Notas:

   a) Guarde los archivos en un formato sin compresión (wav, por ej.).

   b) Utilice un solo canal de audio (mono).

   c) Use una frecuencia de muestreo de 8 kHz.

2. Separe los fragmentos de audio en dos partes:

   a) Una parte para entrenamiento que contenga un solo fragmento de cada individuo.

   b) Una parte para prueba que contenga el resto de los fragmentos.

3. Procese cada fragmento de entrenamiento de la siguiente manera:

   a) Obtenga los coeficientes mel-cepstrales estáticos y dinámicos de la forma explicada en la Sección II de [1]. Para un entendimiento intuitivo del análisis mel-cepstral puede consultar distintas fuentes, por ej., [4].

   b) Reduzca la cantidad de datos resultante a sólo 16 vectores-código utilizando un algoritmo de cuantización vectorial donde la distorsión está dada por la suma de los cuadrados de los componentes de los vectores. Ver [3]. Un algoritmo poco eficiente para este paso está dado por el código de Octave `vq(data,nvector)` que aparece más abajo.

   El resultado de estos pasos va a ser un conjunto de 16 vectores código de 26 componentes cada uno para cada individuo: cada persona se encontrará identificada/representada por estos 16 vectores.

4. Procese cada fragmento de prueba de la siguiente manera:

   a) Obtenga los coeficientes mel-cepstrales estáticos y dinámicos de la forma explicada en la Sección II de [1].

   b) Determine a qué individuo corresponde buscando qué conjunto de vectores código minimiza la distorsión media. Un algoritmo poco eficiente para este paso está dado por el código de Octave `meandist(data,code)` que aparece más abajo.

5. A partir de los resultados en el paso anterior, determine la efectividad del algoritmo como la tasa de identificaciones acertadas.

6. Sea libre y creativo: cambie el procedimiento seguido de la manera que se le ocurra, proponga variaciones, innovaciones, mejoras, etc.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Parámetros de entrada:
%   * data = dataset original
%   * nvector = número de code vectors requeridos
%
%Parámetros de salida:
%   * code = code vectors resultantes

function code = vq(data,nvector)

%distorsion epsilon->determina cuánto itero
deps = 1e-10;

[dim,n] = size(data);

%randomizo
data = data(:,randperm(n));

%code vectors
code = zeros(dim,nvector);

%clusters
clus = zeros(nvector,n+1);

%inicialmente separo el clusters al azar y
%calculo los centroides correspondientes
m = floor(n/nvector);
for k = 1:nvector-1
    clus(k,n+1) = m;
    clus(k,1:m) = (k-1)*m+1:k*m;
    code(:,k) = mean(data(:,clus(k,1:m)).').';
end
clus(nvector,n+1) = length((nvector-1)*m+1:n);
clus(nvector,1:clus(nvector,n+1)) = (nvector-1)*m+1:n;
code(:,nvector) = mean(data(:,clus(nvector,1:clus(nvector,n+1))).').';

%En cada paso reagrupo los clusters y calculo la nueva
%distorsión media. Si el cambio de distorsión no es significativo
%paro
olddist = 0;
newdist = +inf;
while abs(newdist-olddist) > deps
    olddist = newdist;
    clus = zeros(nvector,n+1);
    dist = 0;
    for l = 1:n
        distances = norm(repmat(data(:,l),1,nvector)-code,2,'columns');
        [m,i] = min(distances);
        dist = dist+m;
        i = min(i);
        clus(i,n+1) = clus(i,n+1)+1;
        clus(i,clus(i,n+1)) = l;
    end
    newdist = dist/n;
```

```
    for k = 1:nvector
        m = clus(k,n+1);
        code(:,k) = mean(data(:,clus(k,1:m)).').';
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Parámetros de entrada:
%   * data = dataset
%   * code = code vectors de un individuo
%
%Parámetros de salida:
%   * dist = distorsión media

function dist = meandist(data,code)

[dim,n] = size(data);
[dim,nvector] = size(code);

dist = 0;

for l = 1:n
    %busco la mínima distancia/distorsión de cada columna
    distances = norm(repmat(data(:,l),1,nvector)-code,2,'columns');
    [m,i] = min(distances);
    dist = dist+m;
end
dist = dist/n;
```

2. Speech Compression

En este trabajo intentaremos un simple compresor del habla. Nos basaremos en [5], donde se propone un esquema muy simple de compresión basado en tres pasos: i) transformación, ii) cuantificación, iii) codificación Huffman.

1. Consiga al menos 30 muestras de habla de varios individuos. Cada muestra debe consistir en por lo menos una palabra. Notas:
   a) Guarde los archivos en un formato sin compresión (wav, por ej.).
   b) Utilice un solo canal de audio (mono).
   c) Use una frecuencia de muestreo de 8 kHz.
2. Procese cada fragmento de la siguiente manera:
   a) Obtenga la FFT del fragmento.
   b) Guarde la mitad (+1) de los coeficientes (los restantes son iguales).
   c) Haga cero aquellos coeficientes cuyo módulo sea menor a ε.
   d) Cuantifique la parte real y la parte imaginaria de cada coeficiente con L bits.
   e) Determine la longitud en bits del fragmento comprimido usando codificación Huffman. Calcule el porcentaje de compresión. Como guía, pueden ver la función de Octave `complen()` que se encuentra más abajo.
   f) A partir de los coeficientes cuantificados, intente recuperar la señal original usando la IFFT. Tome la parte real de su resultado. Determine la distorsión como la distancia cuadrática media (promedio del cuadrado de la diferencia de las muestras). ¿Cómo se oye la señal recuperada?
3. Repita el paso anterior variando ε y L. Para $L = 4$ fijo, grafique la distorsión y la razón de compresión promedio en función de ε, $0 \leq \varepsilon 10$. Para $\varepsilon = 0{,}1$, grafique la distorsión y la razón de compresión promedio en función de L, con L de 1 a 8.
4. Sea libre y creativo: cambie el procedimiento seguido de la manera que se le ocurra, proponga variaciones, innovaciones, mejoras, etc.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Argumentos de entrada
%   *X= parte de la transformada de Fourier cuantizada y truncada que se comprimirá
%   *L= número de bits de la cuantización
%%% Argumento de salida
%   *lencomp= estimación de la longitud luego de la compresión
function lencomp = complen(X,L)

n = length(X);
rX = real(X);
iX = imag(X);

c = [rX;iX];
```

```
symbols = unique(c);
freq = hist(c,symbols);
p = freq/sum(freq);
%MATLAB:
%[dict,lencomp] = huffmandict(symbols,p);
%OCTAVE:
dict = huffmandict(symbols,p);

lencomp = 0;

%le tengo que sumar el diccionario
lencomp = lencomp+L*length(symbols);
for k = 1:length(symbols)
    lencomp = lencomp + length(dict{k});
end

%le tengo que sumar la codificación del mínimo y el máximo
%-> + 2 puntos flotantes
lencomp = lencomp + 2*64;

%le tengo que sumar la codificación de la longitud original de x
%-> + 1 entero de 32 bits
lencomp = lencomp + 32;

%le tengo que sumar la codificación de L
%-> + 1 entero de 8 bits
lencomp = lencomp + 8;

%le tengo que sumar el archivo comprimido
for k = 1:length(symbols)
    lencomp = lencomp + freq(k)*length(dict{k});
end
```

3. Integer FFT and Image Compression

En este trabajo intentaremos un simple compresor de imágenes sin pérdida de calidad. Nos basaremos en [6], donde se propone un esquema muy simple de compresión basado en tres pasos: i) transformación invertible que mapea enteros a enteros, ii) codificación aritmética (en nuestro caso, usaremos codificación Huffman).

1. Consiga al menos 10 imágenes en escala de grises sin comprimir. Recomendamos que no se utilicen más de 256 niveles de grises (un byte por pixel) y que no sean demasiado grandes.

2. Programación:

   a) Implemente la transformada discreta coseno bidimensional (TDC2) que mapea matrices de enteros a matrices de enteros propuesta por Yusong *et al.* [6]. Asuma que trabaja sólo con matrices de 8×8.

   b) Implemente la inversa de la TDC2.

   c) Pruebe el correcto funcionamiento de sus programas con algunos ejemplos.

3. Procese cada imagen elegida de la siguiente manera:

   a) Determine la longitud en bits de la imagen comprimida usando codificación Huffman. Como guía, pueden ver la función de Octave `complen()` que se encuentra más abajo.

   b) Obtenga la imagen transformada que consiste en aplicar la TDC2 a cada bloque de la imagen de 8×8 píxeles.

   c) Determine la longitud en bits de la transformada comprimida usando codificación Huffman.

   d) Cuantifique la ganancia (o pérdida) de relación de compresión debida a la transformación.

4. Sea libre y creativo: cambie el procedimiento seguido de la manera que se le ocurra, proponga variaciones, innovaciones, mejoras, etc.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Argumento de entrada
%   *x= matriz a comprimir
%%% Argumento de salida
%   *lencomp= estimación de la longitud luego de la compresión
function lencomp = complen(x)

[m,n] = size(x);
x = reshape(x,m*n,1);
x = x-min(x);

symbols = unique(x);
freq = hist(x,symbols);
p = freq/sum(freq);
dict = huffmandict(symbols,p);
```

```
lencomp = 0;

%le tengo que sumar el diccionario
lencomp = lencomp+nextpow2(max(x))*length(symbols);
for k = 1:length(symbols)
    lencomp = lencomp + length(dict{k});
end

%le tengo que sumar la codificación del tamaño original de x
%-> + 2 entero de 32 bits
lencomp = lencomp + 2*32;

%le tengo que sumar la codificación del mínimo
%-> asumo que con un entero de 32 bits basta
lencomp = lencomp + 32;

%le tengo que sumar el archivo comprimido
for k = 1:length(symbols)
    lencomp = lencomp + freq(k)*length(dict{k});
end
```

Referencias

[1] Wei Han, Cheong-Fat Chan, Chiu-Sing Choy, and Kong-Pang Pun. An efficient mfcc extraction method in speech recognition. In *Proceedings IEEE International Symposium on Circuits and Systems, 2006. ISCAS 2006.*, pages 4 pp.–, May 2006.

[2] Md Rashidul Hasan, Mustafa Jamil, Md Golam Rabbani, and Md Saifur Rahman. Speaker identification using mel frequency cepstral coefficients. In *3rd International Conference on Electrical & Computer Engineering ICECE*, volume 2004, 2004.

[3] Y. Linde, A Buzo, and R.M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, Jan 1980.

[4] James Lyons. Mel frequency cepstral coefficient (mfcc) tutorial. http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/. Accesado en Septiembre 2014.

[5] G. Rajesh, A Kumar, and K. Ranjeet. Speech compression using different transform techniques. In *2nd International Conference on Computer and Communication Technology (ICCCT)*, pages 146–151, Sept 2011.

[6] Yan Yusong, Wang Chunmei, Su Guangda, and Shi Qingyun. Invertible integer dct applied on progressive until lossless image compression. In *Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis, 2003. ISPA 2003.*, volume 2, pages 1018–1023 Vol.2, Sept 2003.