ANEXO I – ELEMENTOS DE LA SINTAXIS DE MPEG-2

Los detalles de las capas presentes en la jerarquía del flujo elemental de vídeo MPEG-2 se describen a continuación.

BLOQUE

El bloque es el único elemento de MPEG-2 que no tiene cabecera. Contiene de forma sucesiva los coeficientes cuantificados acabando con un código especial para informar al descodificador del final del bloque. Todos los coeficientes, a excepción del primero en imágenes de tipo I, se representan en el flujo de datos mediante un código de longitud variable (vlc) obtenido a través de unas tablas predefinidas en el estándar. Este código indicará su valor y el número de ceros que le preceden según la matriz de zig-zag.

El caso del coeficiente DC se trata de forma especial en imágenes de tipo I. Este se codifica diferencialmente ya que es muy probable que bloques próximos en la imagen mantengan niveles promedios de luminancia y crominancia parecidos.

MACROBLOQUE

El elemento de menor nivel que contiene cabecera es el macrobloque. Su cabecera indica la posición del macrobloque con respecto al anterior, recorriéndolos de izquierda a derecha y de arriba a abajo. Esto quiere decir que un macrobloque puede no estar presente en el flujo de datos (a excepción siempre del primero y último de cada *slice*). Este caso solamente ocurrirá al emplear predicción temporal y para su reconstrucción se utilizará el macrobloque del fotograma referencia si es de tipo P o el inmediatamente anterior en el flujo de datos si se trata de uno de tipo B.

Si el macrobloque alberga información temporal, en la cabecera aparecerán también los vectores de movimiento, codificados de manera diferencial para aprovechar que el movimiento de varios bloques próximos entre sí en una imagen suele ser parecido. MPEG-2 vídeo permite además codificar en el flujo de datos únicamente los bloques en los que el residuo sea distinto de cero. A la cabecera se le puede añadir también un parámetro de cuantificación que ajusta el nivel de compresión y que será efectivo únicamente en los bloques internos.

SLICE

La capa directamente superior al macrobloque es el *slice*. En su cabecera se define siempre un parámetro de cuantificación para regular el compromiso compresión-calidad en los macrobloques internos que no lo indiquen. Además, con la aparición de cada nuevo *slice* en el flujo de datos, el descodificador debe resetear tanto los predictores de los coeficientes de continua como los de los vectores de movimiento. Esto asegura una vez más la robustez frente a errores en la transmisión.

PICTURE

Su cabecera debe indicar el tipo de la imagen aparecerá a continuación. Además incluirá la referencia temporal y también información sobre el buffer necesario para la descodificación de la misma. Esta cabecera irá acompañada de una extensión de cabecera que albergará otros parámetros, por ejemplo los que definen la precisión de los coeficientes DC o la selección de las tablas de MPEG-1 o MPEG-2, entre otros.

GOP

La aparición de esta cabecera permite al descodificador realizar un acceso aleatorio de la secuencia de vídeo. En cambio, los parámetros internos de esta cabecera únicamente se aprovechan en editores de vídeo digital, y no influyen en ningún caso en la descodificación del flujo de datos.

La información sobre el tamaño del GOP esta intrínseca en dos parámetros definidos al codificar la secuencia de vídeo, N y M. El primero de ellos (N) corresponde al período con el que aparecen las imágenes de tipo I en la secuencia, que marcarán un comienzo de GOP. El segundo de ellos (M) indica la frecuencia de las imágenes de tipo P dentro del GOP, en cuyo caso el resto de imágenes interiores lo formarán imágenes tipo B. Una configuración habitualmente utilizada es N=12, M=3.

Dado que para descodificar una imagen de tipo B el descodificador necesita antes la imagen de una referencia temporal futura, la ordenación en el flujo de datos de las imágenes no se corresponde con su relación temporal. En cambio, estas aparecen ordenadas de tal forma que el descodificador tenga un acceso previo a las imágenes de referencia necesarias para descodificar el fotograma actual, tal y como muestra la Figura 22.

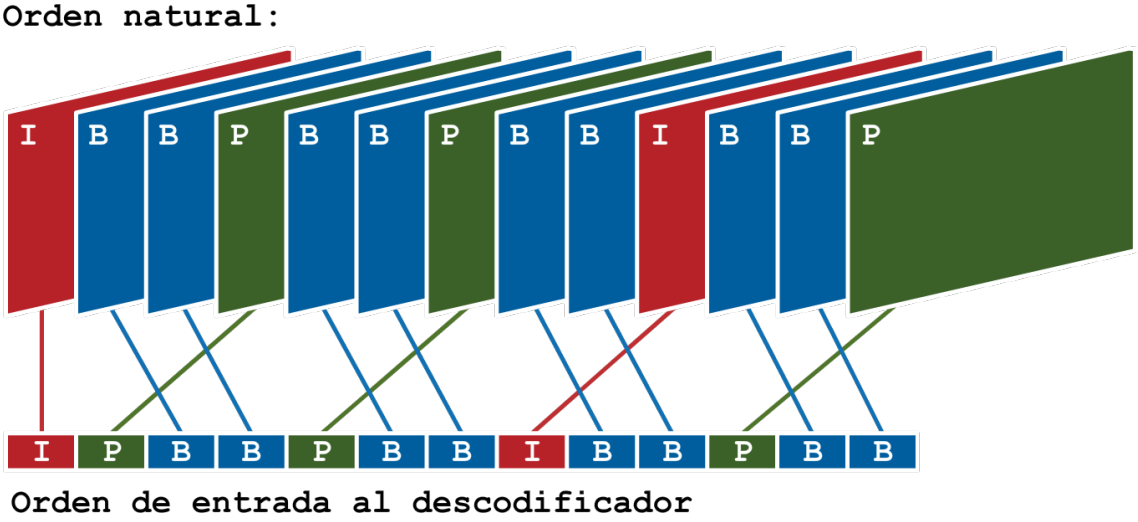


Figura 22 – Orden natural y orden en el flujo de datos de las imágenes en MPEG-2.

SEQUENCE

La secuencia es el elemento superior que engloba todos los demás elementos de una secuencia de vídeo en MPEG-2. Su cabecera define parámetros propios de la

secuencia que no podrán ser modificados, por lo que, si alguno de estos fuera necesario modificarlo, el codificador estará obligado a iniciar una nueva secuencia en el flujo de datos, empleando primero el código reservado *sequence_end_code* para terminar la primera.

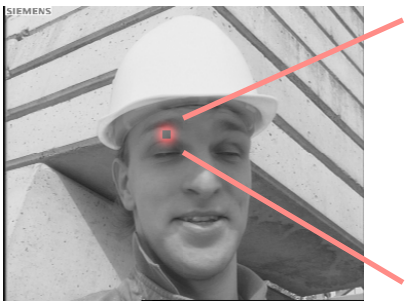
Alguno de los parámetros incluidos en la cabecera son el número de filas y columnas de las imágenes de la secuencia, la relación de aspecto de estas y su formato, la velocidad de fotogramas y el tamaño de buffer necesario para la descodificación de la secuencia. Además, si el codificador hubiera empleado alguna matriz de cuantificación personalizada estarían incluidas en este nivel para que el descodificador pudiera tener acceso a las mismas.

ANEXO II – CODIFICACIÓN DE UN BLOQUE EN MPEG-2

La codificación de un bloque contempla dos situaciones, según el tipo del que se trate. Para uno de tipo Intra el primer coeficiente será codificado diferencialmente, mientras que al resto de coeficientes se les tratará de un modo parecido al aplicado a los coeficientes de bloques de tipo no Intra (P o B).

CODIFICACIÓN DE UN BLOQUE DE TIPO INTRA

Se utilizará a modo de ejemplo un bloque de la componente de luminancia de una imagen de tipo I, mostrado en la Figura 23.



$$\begin{bmatrix} 114 & 115 & 120 & 120 & 120 & 118 & 115 & 118 \\ 118 & 120 & 122 & 122 & 121 & 119 & 117 & 118 \\ 121 & 122 & 124 & 125 & 122 & 122 & 121 & 119 \\ 120 & 122 & 123 & 122 & 118 & 119 & 117 & 115 \\ 119 & 121 & 121 & 120 & 118 & 118 & 118 & 114 \\ 118 & 118 & 118 & 118 & 117 & 115 & 115 & 113 \\ 118 & 118 & 117 & 116 & 117 & 116 & 113 & 112 \\ 114 & 115 & 115 & 115 & 114 & 113 & 112 & 109 \end{bmatrix}$$

Figura 23 – Niveles del bloque (16,20) de una imagen tipo I.

El primer paso es consiste en dejar el valor central del rango de los coeficientes a cero, para lo que es necesario restar 128 a cada uno de ellos. Al bloque resultado se le realizará la transformada DCT bidimensional empleando la Fórmula 1, para la que MPEG-2 representa los valores obtenidos en números enteros con 11 bits.

$$EPY(:,:,16,20) = \begin{bmatrix} -81 & 10 & -9 & -2 & -2 & 1 & 0 & 0 \\ 15 & -5 & -3 & -4 & 1 & 0 & 2 & 0 \\ -11 & -4 & -1 & 0 & 2 & -2 & 1 & 1 \\ -5 & -1 & 0 & 0 & 1 & -1 & 0 & 1 \\ -5 & 0 & -1 & -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & -2 & -1 & 0 & 1 & 1 & 0 & 0 \\ 3 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A partir de este punto MPEG-2 trata manera especial el primer coeficiente, ya que suele contener la mayor parte de la energía del bloque. Por esto separaremos la codificación del bloque en dos apartados diferentes.

COEFICIENTE DC

En la cabecera de imagen se incluye la variable *intra_dc_precision*, que indica la precisión en bits con la que se representará el coeficiente DC de las imágenes tipo I. Esta precisión vendrá interpuesta por un factor *K* que reduce el rango de representación de los coeficientes, según sea el valor de dicha variable.

intra_dc_precision	Precisión (bits)	*K*	Valor de reinicio
0 (00)	8	8	128
1 (01)	9	4	256
2 (10)	10	2	512
3 (11)	11	1	1024

Tabla 3 – Posibles valores de la variable *intra_dc_precision*.

De esta forma, el valor cuantificado del coeficiente DC quedaría de la siguiente forma:

$$EPscY(1,1,16,20) = round(\frac{round(EPY(1,1,16,20)}{K})$$

Si en nuestro ejemplo eligiéramos *intra_dc_precision = "00"*, el valor del coeficiente de continua cuantificado que obtendríamos sería:

$$EPscY(1,1,16,20) = round({}^{-81}/_{8}) = -10$$

Esta información es la que almacenará la estructura de datos. A partir de aquí será el codificador binario el que se encargue de codificarlo diferencialmente. Ya que la imagen la generan tres componentes, MPEG-2 define un predictor para cada una de ellas. Será este predictor el que se codificará y aparezca en el flujo de datos. Estos predictores se reiniciarán al comienzo de cada *slice* a los valores indicados en la Tabla 3, pero como en nuestro caso se ha dividido el proceso en dos partes, al realizar la codificación diferencial de los coeficientes estos ya se encuentran cuantificados de tal forma que el valor central es cero, es decir, el valor de reinicio en este caso será siempre 0.

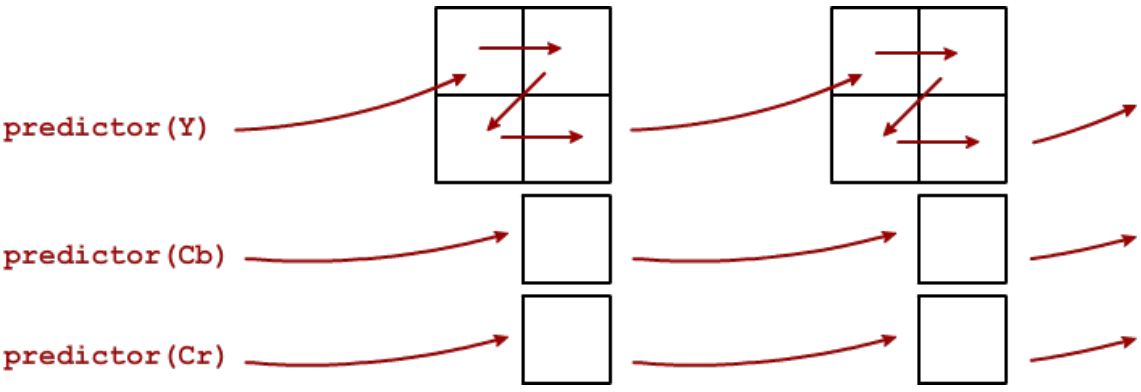


Figura 24 – Codificación diferencial de los coeficientes DC.

La codificación de esos predictores en el flujo de datos viene determinada a través de dos valores: tamaño en bits necesario para representarlo (*dct_dc_size_luminance)* y valor del predictor (*dct_dc_differential)*. El primero de ellos se trata de un código de longitud variable (*vlc*) que indica el número de bits en los que a continuación estará representado el predictor. Este código se obtiene de la Tabla 4.

dct_dc_size_luminance	Código vlc
0	100
1	00
2	01
3	101
4	110
5	1110
6	11110
7	111110
8	1111110
9	11111110
10	111111110
11	111111111

Tabla 4 – Codificación binaria de la variable *dct_dc_size_luminance*.

Suponiendo en nuestro ejemplo que se trata del primer bloque del *slice*, el valor a codificar del predictor de luminancia sería la diferencia con el valor central del rango de los coeficientes:

$$predictor[1] = -10 - 0 = -10$$

Como en nuestro caso para representar el valor absoluto de -10 necesitamos 4 bits:

$$dct\_dc\_size\_luminance = "110"$$

Para representar el predictor en nuestro ejemplo, al ser negativo, es necesario representarlo en valor absoluto para después invertir todos los bits:

$$10 = "1010" \Rightarrow dct\_dc\_differential = "0101"$$

Por lo que la representación en el flujo de datos del coeficiente DC del bloque en este caso sería:

$$dct\_dc\_size\_luminance + dct\_dc\_differential = "1100101"$$

COEFICIENTES AC

A continuación en el flujo de datos aparecen de manera consecutiva el resto de coeficientes del bloque. Para codificarlos es necesario previamente cuantificar su valor transformado. Esta cuantificación ser realiza en dos partes. En la primera, todos los valores son reducidos en un factor escala (*quantiser_scale*) codificado o bien en la cabecera de macrobloque o bien en la cabecera del *slice*, y que en nuestro ejemplo supondremos un valor final de 4. En la segunda parte se aplica la matriz de pesos

definida para las imágenes tipo I (*MatQuantIY*), donde para nuestro caso se elegirá la propia del estándar (Figura 9).

$$EPscY = round\left(\frac{fix(32 \cdot round(EPY))}{MatQuantIY \cdot 2 \cdot quantiser\_scale}\right)$$

$$EPscY(:,:,16,20) = \begin{bmatrix} -10 & 3 & -2 & 0 & 0 & 0 & 0 & 0 \\ 4 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\ -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

El siguiente paso es recorrer todos los coeficientes cuantificados, excepto el primero, según la matriz de ordenación definida en la Figura 9.

$$3, 4, -2, -1, -2, 0, -1, -1, -1, -1, 0, 0, -1, 0, 0 \ \ldots \ 0, 0$$

Estos datos serán representados mediante lo que MPEG-2 llama el par *run-level*. Se trata del número de ceros consecutivos que preceden a cada coeficiente junto con el nivel de dicho coeficiente. Esto se realiza para todos los valores distintos de cero. En nuestro caso quedaría así:

RUN	***LEVEL***
0	3
0	4
0	-2
0	-1
0	-2
1	-1
0	-1
0	-1
0	-1
2	-1

Tabla 5 – Valores *run-level* de los coeficientes AC.

La codificación de estos valores en el flujo de datos se realiza mediante un código de longitud variable obtenido de la Tabla B-14 en [1]. Existe otra tabla en el estándar que cumple el mismo propósito, pero se utiliza únicamente en el caso de requerir una compatibilidad con MPEG-1, por lo que en este ejemplo no se utilizará.

RUN	*LEVEL*	*VLC*
0	3	001010
0	4	00001100
0	-2	01001
0	-1	111
0	-2	01001
1	-1	0111
0	-1	111
0	-1	111
0	-1	111
2	-1	01011

Tabla 6 – Códigos de longitud variable para los valores *run-level* del bloque.

Quedará, por tanto, la parte correspondiente a los coeficientes AC del bloque representada con la siguiente cadena binaria:

"00101000001100010011110100101111111111101011"

FIN DE BLOQUE

Como en la codificación del bloque aparecen de forma consecutiva todos los coeficientes cuantificados, el descodificador no sabe exactamente dónde termina cada bloque. Por ello en MPEG-2 se debe insertar un código especial de fin de bloque que indicará al descodificador que no existen más coeficientes codificados para dicho bloque.

$$end\_of\_block = \text{"10"}$$

Por tanto, la codificación completa del bloque será la concatenación de todas las cadenas anteriores:

"11001010010100000110001001111010010111111111110101110"

La codificación resultante ocupa 54 bits en el flujo de datos. El mismo bloque con una codificación sin compresión de 8 bits por píxel nos resultaría en una cadena de 512 bits de longitud. Con lo cual, el bloque comprimido en MPEG-2 tendría un tamaño del 10,54% del original, algo acorde con los resultados presentados en el apartado 3.6.

CODIFICACIÓN DE UN BLOQUE DE TIPO NO INTRA

Las imágenes de tipo P pueden incluir macrobloques que se aprovechen de la compensación de movimiento. En este caso la codificación de los coeficientes de los bloques sigue el mismo proceso que los coeficientes AC del caso anterior, salvo unas

ligeras modificaciones. Ya que el desarrollo de la codificación de estos bloques se asemeja al presentado en el apartado anterior, se enumeran a continuación únicamente las diferencias con el proceso anterior:

- El primer coeficiente no se codifica diferencialmente, sino que se incluye a la codificación *run-level* del resto de coeficientes.
- La matriz de cuantificación definida en MPEG-2 tiene ahora el valor 16 en todas sus posiciones.
- La fórmula para la cuantificación en este caso es la siguiente:

$$EPscY = fix\left(\frac{round(2 \cdot EPY) + sign(EPY)}{2 \cdot quantiser\_scale}\right)$$

ANEXO III – ESTRUCTURA DE DATOS IMPLEMENTADA

A continuación se presentan los detalles de la estructura de datos con la que trabaja la herramienta, y que sirve para almacenar toda la información generada por el compresor híbrido que necesita el codificador binario para generar el flujo elemental de vídeo.

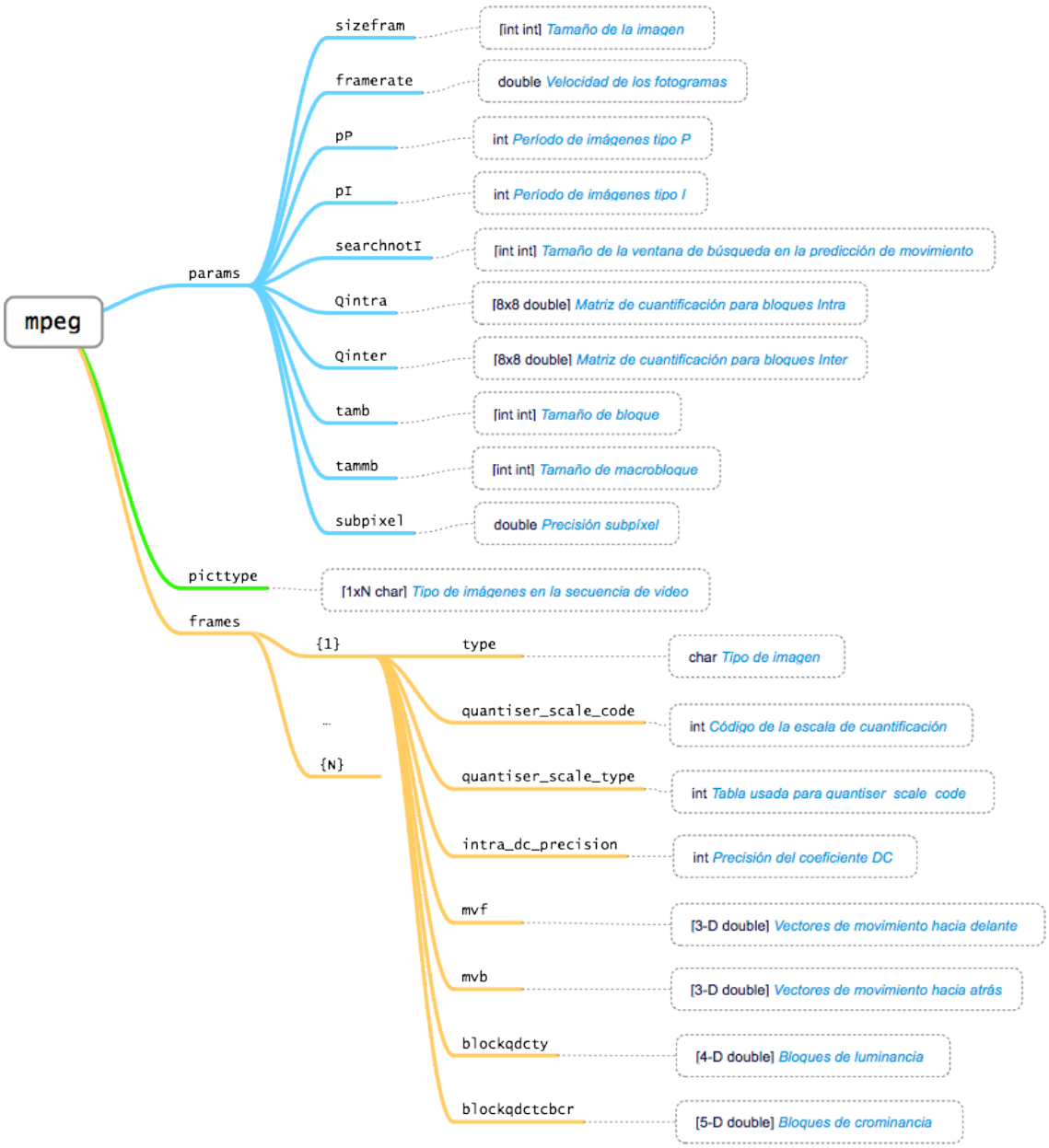


Figura 25 – Detalles sobre la información almacenada en la estructura de datos.

ANEXO IV – EJEMPLOS DE CÓDIGO DE USO EN PRÁCTICAS

En este anexo se presenta un ejemplo de un posible guión a seguir en una práctica de compresión de vídeo. Se incluyen tanto el código a seguir en cada apartado de la práctica como los ficheros fuente de las principales funciones de la herramienta implementada, todos ellos disponibles en las siguientes páginas.

analiza_estructura.m	Donde se detalla la estructura generada por el compresor híbrido y se muestran ejemplos gráficos de la información almacenada.
compromiso_rate_distortion.m	Ofrece al alumno la comprobación del compromiso compresión-calidad.
depuracion_proceso_completo.m	Permite al alumno profundizar en el proceso de compresión de vídeo y ejecutar la herramienta para modificar libremente los parámetros de entrada y observar su resultado
vídeo2mpeg.m	Código fuente del compresor híbrido MPEG-2 implementado.
mpeg2ES.m	Código fuente del codificador binario MPEG-2 implementado

PRACTICA - ANÁLISIS DE LA ESTRUCTURA

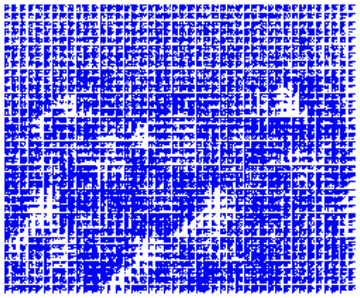
```
% La estructura generada (mpeg) almacena toda la información necesaria para poder generar una secuencia de video reproducible.
% Ejecuta las siguientes líneas:

clear all, close all

load estructura.mat;
mpeg

% ¿Qué piensas que almacena cada campo? Ejecuta las siguientes líneas una a
% una e identifica los parámetros

mpeg.params
mpeg.frames{1}

% Como puedes observar, el primer frame es de tipo I. ¿Qué tipo de frame
% será el que contenga mpeg.frames{2}?

mpeg.picttype
mpeg.frames{2}

% Como puedes comprobar, los frames están almacenados de tal forma que el
% descodificador no tenga problemas al realizar la compensación de
% movimiento, ya que los frames de tipo B requieren de un fotograma
% posterior al tener predicción bidireccional.

% ¿Cómo crees que están almacenados los coeficientes asociados a la
% luminancia? ¿Y los vectores de movimiento? Ejecuta las siguientes líneas
% para comprobarlo

mpeg.frames{2}.blockqdcty(:,:,12,12)
mpeg.frames{2}.mvf(6,6,:)


% A continuación se puede comprobar cómo intervienen los vectores de
% movimiento en la compensación de movimiento

rgb1=ycbcr2rgb(cat(3,yuvdecint.Y(:,:,1),imresize(yuvdecint.Ch(:,:,1,1),2),imresize(yuvdecint.Ch(:,:,1,2),2)));
rgb2=ycbcr2rgb(cat(3,yuvdecint.Y(:,:,4),imresize(yuvdecint.Ch(:,:,4,1),2),imresize(yuvdecint.Ch(:,:,4,2),2)));

figure('Position', [100, 100, 1000, 600]);

subplot(2,2,1)
subimage(rgb1);
title('Imagen de referencia');
axis off;

[xx,yy]=meshgrid(1:16:mpeg.params.sizefram(2),1:16:mpeg.params.sizefram(1));
%hi=imshow( [yuvdecint.Y(:,:,4) ; yuvdecint.Y(:,:,1)]); % Solo luminancia
% hi=imshow( [rgb2 ; rgb1] );
subplot(2,2,3)
subimage(rgb2)
hold on;
title('Frame P con los vectores de movimiento.');
hq=ht;
mv1=mpeg.frames{2}.mvf(:,:,1);
mv2=mpeg.frames{2}.mvf(:,:,2);
hq=quiver(xx+mv2+8,yy+mv1+8,mv2,mv1,0);
set(hq,'Color','w');
axis off;


% A continuación puedes comprobar las diferencias entre los coeficientes
% retenidos de una imagen I y una imagen P

subplot(2,2,2);
spy(reshape(permute(mpeg.frames{1}.blockqdcty,[1 3 2 4]),mpeg.params.sizefram),0.5);
title('Coeficientes retenidos imagen I');
axis off;

subplot(2,2,4);
spy(reshape(permute(mpeg.frames{2}.blockqdcty,[1 3 2 4]),mpeg.params.sizefram),0.5);
title('Coeficientes retenidos imagen P');
axis off;
```

```
mpeg =

      params: [1x1 struct]
    picttype: 'IBBPBBPBBPIBP'
      frames: {1x13 cell}


ans =

      sizefram: [288 352]
     framerate: 30
```

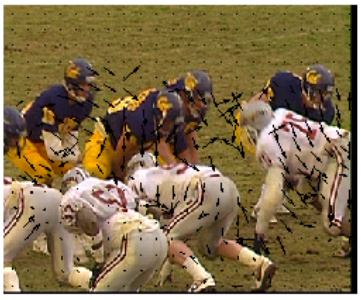
```
            pP: 3
            pI: 10
    searchnotI: [16 16]
        Qintra: [8x8 double]
        Qinter: [8x8 double]
          tamb: [8 8]
         tammb: [16 16]
      subpixel: 0.5000


ans =

                  type: 'I'
    quantiser_scale_code: 2
    quantiser_scale_type: 1
      intra_dc_precision: 0
              blockqdcty: [4-D double]
           blockqdctcbcr: [5-D double]


ans =

IBBPBBPBBPIBP


ans =

                  type: 'P'
    quantiser_scale_code: 4
    quantiser_scale_type: 1
      intra_dc_precision: 0
                   mvf: [18x22x2 double]
              blockqdcty: [4-D double]
           blockqdctcbcr: [5-D double]


ans =

    -2    -5    -3    -3     0    -1    -2     0
     4    -1     0     2     3     2    -3    -1
     1     1     5     0     0     0    -2     0
     1     1     2     0    -1     0     0     1
     0     0     1     0     0     0     0     0
    -1    -1     0     0     0     0     0     0
     0     0    -1     0     0     0     0     0
     0     0     0     0     0     0     0     0


ans(:,:,1) =

     7


ans(:,:,2) =

    11
```

Imagen de referencia

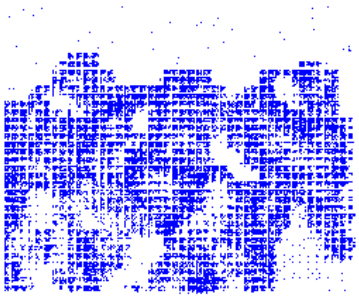
Coeficientes retenidos imagen I



Frame P con los vectores de movimiento.



Coeficientes retenidos imagen P

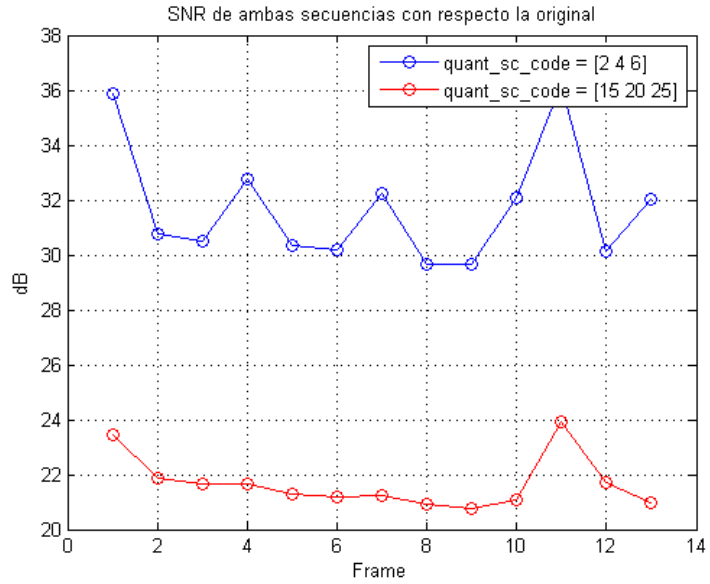


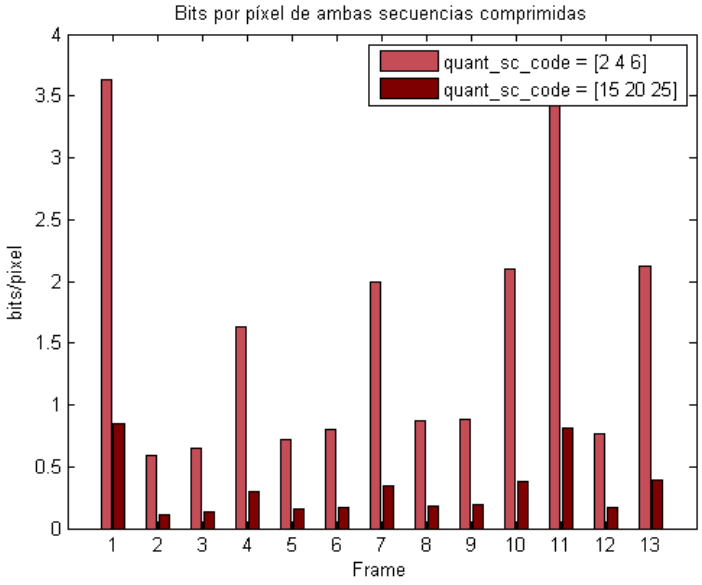
Published with MATLAB® R2014a

PRACTICA - COMPROMISO RATE-DISTORTION

```
% La escala de cuantificación (quant_sc_code) es uno de los parámetros que regula la compresión de la secuencia de video. Este influye
% directamente sobre la cuantificación de forma que a valores altos más coeficientes se acercarán a cero, aumentando la compresión de
% la secuencia. Ejecuta las siguientes líneas para comprobar el efecto que tiene sobre la secuencia final:

clear all
close all force
load('params.mat');

addpath('fuentesdevideo');
addpath('output');
filename='football_cif.y4m';
[Y,U,V]=readY4M(filename,Nframes); Y=double(Y);
mplay([Y [U ; V]],framerate);
yuvvid.Y=double(Y);
yuvvid.Ch=double(cat(4,U,V));

quant_sc_code = [2 4 6];
[mpeg,yuvdecint]=video2mpeg(yuvvid,framerate,pP,pI,quant_sc_code,scale_type,searchreg,i_dc_prec,subpixel);
Ydecint=yuvdecint.Y;    Udecint=yuvdecint.Ch(:,:,:,1);  Vdecint=yuvdecint.Ch(:,:,:,2);

quant_sc_code = [15 20 25];
[mpeg2,yuvdecint2]=video2mpeg(yuvvid,framerate,pP,pI,quant_sc_code,scale_type,searchreg,i_dc_prec,subpixel);
Ydecint2=yuvdecint2.Y;    Udecint2=yuvdecint2.Ch(:,:,:,1);  Vdecint2=yuvdecint2.Ch(:,:,:,2);

hnm(2)=mplay([Y [U ; V]  Ydecint [Udecint ; Vdecint]  Ydecint2 [Udecint2 ; Vdecint2]],framerate);
%set(hnm(2),'Name','Original (izqda)  ------  quant_sc_code = [2 4 6] (centro)  ------  quant_sc_code = [15 20 25] (drcha)');


figure
SNRYrec=10*log10(squeeze(mean(mean(yuvvid.Y.^2,1),2)./...
mean(mean((yuvvid.Y-double(Ydecint)).^2,1),2)));
plot(1:Nframes,SNRYrec,'bo-');
title('SNR reconstrucción para luminancia')
hold on
SNRYrec=10*log10(squeeze(mean(mean(yuvvid.Y.^2,1),2)./...
mean(mean((yuvvid.Y-double(Ydecint2)).^2,1),2)));
plot(1:Nframes,SNRYrec,'ro-');
grid on; xlabel('Frame'); ylabel('dB');
xlim([0 Nframes+1])
legend('quant\_sc\_code = [2 4 6]','quant\_sc\_code = [15 20 25]')
title('SNR de ambas secuencias con respecto la original')



% Como puedes ver la cantidad de compresión afecta a la calidad de video
% final. Para analizar mejor cuánto se comprime en cada uno de los casos
% procedemos a generar el flujo elemental de video acorde a la norma MPEG-2
% y ver así los bits por píxel reales obtenidos. Ten en cuenta que la
% secuencia original necesita 24 bits por cada píxel.

global debug;
debug=0;
[bits bsize]=mpeg2ES(mpeg,'output/pruebas1.m2v',scan_type,i_vlc_format,MB_skipped_f);
[bits2 bsize2]=mpeg2ES(mpeg2,'output/pruebas2.m2v',scan_type,i_vlc_format,MB_skipped_f);
fprintf('\n\n');
disp(['Tamaño con quant_sc_code = [2 4 6]: ',num2str(size(bits,2)),' bits (',num2str(size(bits,2)/8192),' KB)']);
disp(['Tamaño con quant_sc_code = [15 20 25]: ',num2str(size(bits2,2)),' bits (',num2str(size(bits2,2)/8192),' KB)']);

[picttype,index]=shufflestream(pP,pI,Nframes); [natural iindex]=sort(index);
figure
bar_handle=bar([bsize(iindex)/(size(yuvvid.Y,1) * size(yuvvid.Y,2)) bsize2(iindex)/(size(yuvvid.Y,1) * size(yuvvid.Y,2))]);
set(bar_handle(1),'FaceColor',[196,77,88]/255)
ylabel('bits/pixel')
xlabel('Frame')
xlim([0 Nframes+1]);
legend('quant\_sc\_code = [2 4 6]','quant\_sc\_code = [15 20 25]')
title('Bits por píxel de ambas secuencias comprimidas')
```

```
Tamaño con quant_sc_code = [2 4 6]: 2048200 bits (250.0244 KB)
Tamaño con quant_sc_code = [15 20 25]: 425608 bits (51.9541 KB)
```

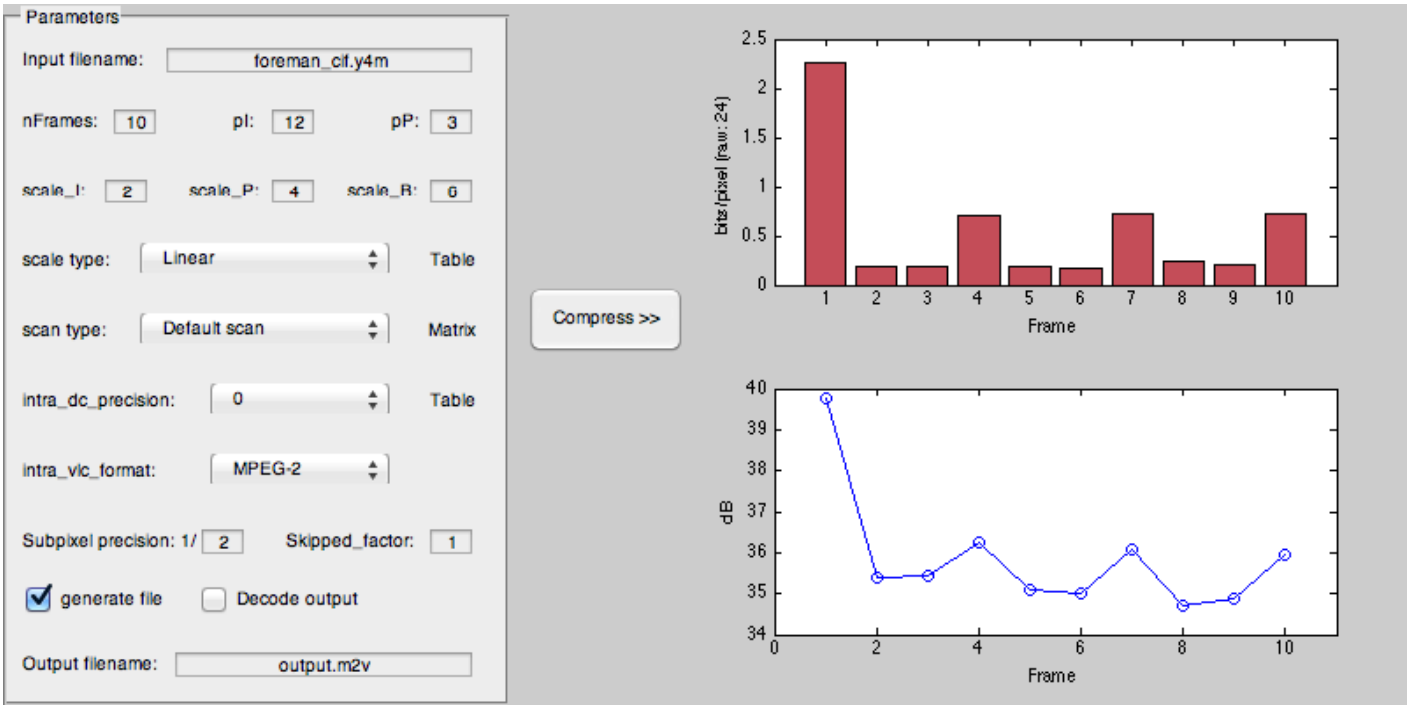




Published with MATLAB® R2014a

PRACTICA - DEPURACIÓN DEL PROCESO COMPLETO

```
% Ahora se analizarán tanto la compresión de la señal como la generación
% del flujo elemental de video MEPG-2. Para ello edita las siguientes
% funciones y crea un punto de ruptura en su bucle principal.

clear all
close all force
addpath('fuentesdevideo');
addpath('output');
edit video2mpeg
edit mpeg2ES

% Ejecuta la herramienta y comprime con los valores por defecto para
% depurar los procesos

interfaz;


% Los parámetros de configuración de la secuencia de video comprimida
% aparecen en las cabeceras de cada capa del flujo elemental de video. Uno
% de estos parámetros es el aspect_ratio. Modifica el aspect_ratio en la
% siguiente función (línea 5) para que escriba el valor 3 (0011).
% ¿Cómo afecta esto al flujo elemental generado?

edit PutSequenceHeader
interfaz;


% Modifica el resto de parámetros que ofrece la herramienta para comprobar
% cómo afectan al resultado final mediante el archivo generado y las
% gráficas presentadas.

interfaz;
```



Published with MATLAB® R2014a

video2mpeg.m

```
 function [mpeg,yuvdec]=video2mpeg(yuvvid,framerate,pP,pI,quantiser_scale_code,quantiser_scale_type,searchnotI,intra_dc_precision,subpixel)
% [mpeg,yuvdec]=video2mpeg(yuvvid,framerate,pP,pI,quantiser_scale_code,quantiser_scale_type,searchreg,intra_dc_precision,subpixel)
% Implementa codificador híbrido en matlab.
% Los argumentos de entrada son todos necesarios son:
% yuvvid: La secuencia de vídeo a comprimir en formate 4:2:0. yuvvid.I - array filasxcolumnasxframes de valores uint8 de luminancia.
% yuvvic.Ch - array (filas/2)x(columnas/2)xframesx2 de valores uint8 de crominancia.
% framerate: frame rate de la secuencia de vídeo.
% pP: periodicidad de los frames de tipo P (entre ellos, se introducen frames I).
% pI: periodicidad de los frames de tipo I (entre ellos, se introducen frames P).
% quantiser_scale_code: Punteros en el rango 1-31 (de poca a mucha
% cuantificación) a los factores de escala de cuantificación de las tablas
% de la norma MPEG-2. 2 opciones: 1. si se da un vector de tamaño 3, los 3 valores
% están asociados, respectivamente, a los frames de tipo I, a los de tipo P
% y a los de tipo B. 2. si se da un vector de tamaño nº de frames del
% vídeo, el factor de escala se le aplica a cada frame.
% quantiser_scale_type: Uso de tabla de factores de escala lineal (valor
% 1, como en MPEG-1) o logarítmica (MPEG-2)
% searchreg: vector de dimensión 2 asociado a las regiones de búsqueda en
% predicción inter: +/-searchnotI(1) para búsqueda en filas y
% +/-searchnotI(2) para búsqueda en columnas.
% dctprec: Precisión del coeficiente DC de macrobloques intra. Varía en n
% rango 0,1,2,3 (8, 9, 10 u 11 bits respectivamente)
% subpixel: Valor para estimación/compensación de movimiento con precición
% subpixel. Valores: 1 (no uso subpixel), 1/2 ó 1/4.
%
% Los argumentos de salida no son todos necesarios. Si se llama a la
% función con un número menos que el dado, se van devolviendo de izquierda
% a derecha. Son:   mpeg,yuvdec
% mpeg: estructura asociada al video comprimido.
% yuvdec: estructura similar a yuvvid. Devuelve la secuencia descodificada
% por el descodificador interno al codificador.
% Diagrama de bloques implementado:
%                                                .---------------------------------.
%                                                |                                 |
%   .-------.                                    v                                 |
%   |  YUV  |     x          eP  .-------.  EP  .----------.  EPsc  .---------.     .--------.
%   |4:2:0  |-------->( - )----->|  DCT  |----->| QUANTIZE |------->| HUFFMAN |---->| BUFFER |----> OUTPUT STREAM
%   | input | |         ^        '-------'      '----------'   |    | CODING  |     '--------'
%   '-------' |         |                                      |    '---------'
%             |         |                                 .----------.
%             |         |                                 | INVERSE  |
%             |         |                                 | QUANTIZE |
%             |         |                                 '----------'
%             |         |                                      | EPQ
%             |         |                                      v
%             |         |                                 .---------.
%             |         |                                 | INVERSE |
%             |         |                                 |   DCT   |
%             |         |                                 '---------'
%             |         |                                      | ePQ
%             |         |  MC-Prediction                       v
%             |         |------------------------------------>( + )
%             |         |                                      |
%             |         |                                      | y
%             |         |                                  .-------.
%             |         |                                  | FRAME |
%             |       p |                                  | STORE |
%             |         |                                  '-------'
%             |  .------------.                                |
%             |  |   motion   |                             ref   |
%             |  |compensation|<-------------------------------|
%             |  '------------'                                |
%             |         ^                                      | Previous
%             |         | MV data                              | reconstructed
%             |  .------------.                                | frame
%             |  |   motion   |                                |
%             '->|compensation|<-------------------------------'
%                '------------'

scale_tab=[2*(1:31) ; (8*(2.^floor((1:31)/8)-1)+mod(1:31,8).*2.^floor((1:31)/8))]'; %load scale_tab; % load quantiser_scale table
yuvdec=yuvvid;  yuvdec.Y(:)=0;  yuvdec.Ch(:)=0;
sizefram=[size(yuvdec.Y,1) size(yuvdec.Y,2)];
Nframes=size(yuvdec.Y,3);

load Qintra
Qinter=16*ones(8);
tamb=[8 8];
tammb=[16 16];
[params.sizefram,params.framerate,params.pP,params.pI,params.searchnotI,params.Qintra,params.Qinter,...
    params.tamb,params.tammb,params.subpixel]=deal(sizefram,framerate,pP,pI,searchnotI,Qintra,Qinter,tamb,tammb,subpixel);
mpeg.params=params;

% Filtros y kernels para interpolación acorde a 7.6.4 del estándar
% ('Forming predictions').
hY1=1-subpixel*abs(-(1/subpixel-1):(1/subpixel-1));
hY1=hY1'*hY1;   hY2=eye(1/subpixel,1)*eye(1,1/subpixel);
hCh1=[1/2 1 1/2];   hCh1=hCh1'*hCh1;    hCh2=ones(1/subpixel);

[ind_mc,ind_me,funos]=variables_para_indexacion_motionestimcompens(mpeg.params);
[indblocky,indblockcbcr,MatQuantIY,MatQuantICh]=variables_para_tratamiento_por_bloques_coder(mpeg.params);
```

```
[picttype,index]=shufflestream(pP,pI,Nframes);
if length(quantiser_scale_code)==3,
    aux=strrep(strrep(strrep(picttype,'I','1'),'P','2'),'B','3');
    quantiser_scale_code=quantiser_scale_code(str2num(aux')');
end
mpeg.picttype=picttype;

[xY,pY,ePY,ePQY]=deal(zeros(sizefram));
[xCh,pCh,ePCh,ePQCh]=deal(zeros([sizefram/2 2]));

[refRecentY,refOldY,xYp,pfYsp,pbYsp]=deal(zeros(sizefram/subpixel));
[refRecentCh,refOldCh,pfChsp,pbChsp]=deal(zeros([sizefram/subpixel 2]));

[ePQChtmp]=deal(zeros([sizefram 2].*[0.5 0.5 1]));
[EPY,EPscY,EPcompY,EPQscY,EPQY]=deal(zeros(size(indblocky)));
ePChtmp=zeros(size(indblockcbcr));
[EPCh,EPscH,EPscCh,EPcompCh,EPQscCh,EPQCh]=deal(zeros(size(indblockcbcr).*[1 1 1 1 1]));
szch=size(EPCh);



% BUCLE PRINCIPAL COMPRESIÓN HÍBRIDA VÍDEO
for frame=1:Nframes,
    mpeg.frames{frame}.type=picttype(index(frame));
    flag_intra=0;
    xY(:,:)=double(yuvvid.Y(:,:,index(frame)));
    xCh(:,:,:)=double(squeeze(yuvvid.Ch(:,:,index(frame),:)));
    %figure, imshow(uint8([xY [xCh(:,:,1) ; xCh(:,:,2)]]))

    mpeg.frames{frame}.quantiser_scale_code=quantiser_scale_code(index(frame));
    mpeg.frames{frame}.quantiser_scale_type=quantiser_scale_type;
    quantiser_scale = scale_tab(quantiser_scale_code(index(frame)),quantiser_scale_type);
    mpeg.frames{frame}.intra_dc_precision = intra_dc_precision; % Could be different each frame (not covered)
    switch mpeg.frames{frame}.type
        case 'I'
               flag_intra=1;
               [pY(:,:),pCh(:,:,:)]=deal(128,128);
        case 'P'
            % Interpolación acorde a 7.6.4 del standard ('Forming
            % predictions')
            xYp(:,:)=round(convn(kron(xY,hY2),hY1,'same'));
            % Las siguientes funciones obtienen los vectores de movimiento
            % y la imagen predicha.
            meas=getmeas4tempmatch(refRecentY,xYp,ind_me,funos,mpeg.params);
            [mpeg.frames{frame}.mvf,pfYsp(:,:),pfChsp(:,:,:)]=...
                getmvfrommeas(meas,mpeg.params,refRecentY,refRecentCh,ind_mc);
            pY(:,:)=pfYsp(1:1/subpixel:end,1:1/subpixel:end);
            pCh(:,:,:)=pfChsp(1:2/subpixel:end,1:2/subpixel:end,:);

        case 'B'
            xYp(:,:)=round(convn(kron(xY,hY2),hY1,'same'));
            % Se comienzan obteniendo vectores de movimiento y predicciones
            % para las las imágenes de referencia (f: forward, b: bachward)
            measf=getmeas4tempmatch(refOldY,xYp,ind_me,funos,mpeg.params);
            [mvf,pfYsp(:,:),pfChsp(:,:,:),measfmin]=getmvfrommeas(measf,mpeg.params,refOldY,refOldCh,ind_mc);
            measb=getmeas4tempmatch(refRecentY,xYp,ind_me,funos,mpeg.params);
            [mvb,pbYsp(:,:),pbChsp(:,:,:),measbmin]=getmvfrommeas(measb,mpeg.params,refRecentY,refRecentCh,ind_mc);
            % Usando la situación que da lugar a mejor distancia euclídea,
            % se decide y codifica qué macrobloques usarán sólo predicción
            % forward, sólo predicción backward o predicción interpolada
            % forward y backward.
            measdmind2=0.5*squeeze(sum(sum(pfYsp(ind_mc.ind).*pbYsp(ind_mc.ind),1),2))+...
                0.25*measfmin.xTx+0.25*measbmin.xTx-measfmin.xTt-measbmin.xTt+measfmin.tTt;
            d2min=min(measfmin.d2,min(measbmin.d2,measdmind2));
            bbb=double(d2min==measbmin.d2); %bbb: binaria con unos para MB backward
            fff=(1-bbb).*double(d2min==measfmin.d2);    %fff: binaria con unos para MB forward
            both=(1-bbb).*(1-fff);  %both: binaria con unos para MB interpolados
            % Se obtiene consecuentemente a lo anterior la imagen predicha:
            pYsp(:,:)=0.5*kron(both+2*fff,ones(tammb/subpixel)).*pfYsp+0.5*kron(both+2*bbb,ones(tammb/subpixel)).*pbYsp;
            pChsp(:,:,:)=repmat(0.5*kron(both+2*fff,ones(tammb/subpixel)),[1 1 2]).*pfChsp+repmat(0.5*kron(both+2*bbb,ones(tammb/subpixel)),...
                [1 1 2]).*pbChsp;
            mpeg.frames{frame}.mvf=mvf;
            mpeg.frames{frame}.mvf(find(repmat(bbb,[1 1 2])))=NaN;
            mpeg.frames{frame}.mvb=mvb;
            mpeg.frames{frame}.mvb(find(repmat(fff,[1 1 2])))=NaN;
            pY=round(pYsp(1:1/subpixel:end,1:1/subpixel:end));
            pCh=round(pChsp(1:2/subpixel:end,1:2/subpixel:end,:));
    end

    ePY(:,:)=xY-pY;
    EPY(:,:,:,:)=dctdim(dctdim(ePY(indblocky),1),2);
    ePCh=xCh-pCh;
    EPCh(:,:,:,:,:)=dctdim(dctdim(ePCh(indblockcbcr),1),2);

        % CUANTIFICACIÓN: no proporcionada por el estándar. Tomada de
        % 'Techniques & Standards of Image & Video Coding' (Rao, Wang),
        % 11.2.3 (páginas 302-303), realizando modificaciones de lo que
        % parecen erratas.
    if  flag_intra,
        % AC de EPscY/Ch: cambiado orden round y fix respecto a la referencia. Si no
        % no parece implementarse quantificador aproximadamente uniforme.
        EPscY(:,:,:,:)=round(fix(32*round(EPY)./MatQuantIY)/(2*quantiser_scale));
        EPscY(1,1,:,:)=round(round(EPY(1,1,:,:))/2^(3-mpeg.frames{frame}.intra_dc_precision));
        EPscCh(:,:,:,:,:)=round(fix(32*round(EPCh)./MatQuantICh)/(2*quantiser_scale));
        EPscCh(1,1,:,:,:)=round(round(EPCh(1,1,:,:,:))/2^(3-mpeg.frames{frame}.intra_dc_precision));
```

```
    else
        EPscY(:,:,:,:)=fix((round(32*round(EPY)/16)+sign(EPY))/(2*quantiser_scale));
        EPscCh(:,:,:,:,:)=fix((round(32*round(EPCh)/16)+sign(EPCh))/(2*quantiser_scale));
    end


    % Save quantised coeffs:
    mpeg.frames{frame}.blockqdcty=EPscY;
    mpeg.frames{frame}.blockqdctcbcr=EPscCh;

    EPQscY(:,:,:,:)=mpeg.frames{frame}.blockqdcty;
    EPQscCh(:,:,:,:,:)=mpeg.frames{frame}.blockqdctcbcr;

    % CUANTIFICACIÓN INVERSA: Acorde a 7.4.2.3 de la norma (Reconstruction
    % Formulae)
    if flag_intra,
        EPQY(:,:,:,:)=fix(2*EPQscY.*MatQuantIY*quantiser_scale/32);
        EPQY(1,1,:,:)=EPQscY(1,1,:,:)*2^(3-mpeg.frames{frame}.intra_dc_precision);
        EPQCh(:,:,:,:,:)=fix(2*EPQscCh.*MatQuantICh*quantiser_scale/32);
        EPQCh(1,1,:,:,:)=EPQscCh(1,1,:,:,:)*2^(3-mpeg.frames{frame}.intra_dc_precision);
    else
        EPQY(:,:,:,:)=fix((2*EPQscY+sign(EPQscY)).*16*quantiser_scale/32);
        EPQCh(:,:,:,:,:)=fix((2*EPQscCh+sign(EPQscCh)).*16*quantiser_scale/32);
    end
    EPQY(find(EPQscY==0))=0;       EPQCh(find(EPQscCh==0))=0;
    % 7.4.3 norma: saturación
    EPQY=min(max(EPQY,-2048),2047);     EPQCh=min(max(EPQCh,-2048),2047);
    % observacaracteristicacuantificacion;

    % Mismatch Control in decoding process (7.4.4 norma):
    EPQY(end,end,:,:)=EPQY(end,end,:,:)+(1-mod(sum(sum(EPQY,1),2),2)).*(1-2*mod(EPQY(end,end,:,:),2));
    EPQCh(end,end,:,:,:)=EPQCh(end,end,:,:,:)+(1-mod(sum(sum(EPQCh,1),2),2)).*(1-2*mod(EPQCh(end,end,:,:,:),2));

    ePQY(:,:)=min(max(reshape(permute(round(idctdim(idctdim(EPQY,1),2)),[1 3 2 4]),sizefram),-256),255);
    ePQCh(:,:,:)=min(max(reshape(permute(round(idctdim(idctdim(EPQCh,1),2)),[1 3 2 4 5]),[0.5*sizefram 2]),-256),255);
    yuvdec.Y(:,:,index(frame))=min(max(round(pY+ePQY),0),255);
    yuvdec.Ch(:,:,index(frame),:)=min(max(round(pCh+ePQCh),0),255);
%     figure, imshow(uint8([yuvdec.Y(:,:,index(frame)) ...
%         [yuvdec.Ch(:,:,index(frame),1) ; yuvdec.Ch(:,:,index(frame),2)]]))

    % Store reference pictures
    if  (picttype(index(frame))=='I')||(picttype(index(frame))=='P')
        [refOldY(:,:),refOldCh(:,:,:)]=deal(refRecentY,refRecentCh);
        refRecentY(:,:)=round(convn(kron(double(yuvdec.Y(:,:,index(frame))),hY2),hY1,'same'));
        refRecentCh(:,:,1)=round(kron(convn(kron(double(yuvdec.Ch(:,:,index(frame),1)),...
            [1 0 ; 0 0]),hCh1,'same'),hCh2));
        refRecentCh(:,:,2)=round(kron(convn(kron(double(yuvdec.Ch(:,:,index(frame),2)),...
            [1 0 ; 0 0]),hCh1,'same'),hCh2));
    end
end

yuvdec.Y=uint8(yuvdec.Y);   yuvdec.Ch=uint8(yuvdec.Ch);
```

Published with MATLAB® R2014a

mpeg2ES.m

```
function [bits,bsize]=mpeg2ES(mpeg,filename,zigzag,intra_vlc_format,MBsf)
%[bits]=generatefile(mpeg,'prueba.m2v',2,1);
% mpeg:     estructura con la información
% filename: nombre de archivo de salida
% zigzag:   scan seleccionado [1:default_scan, 2:alternate_scan]
% intra_vlc_format: tablas mpeg1 o mpeg2 [0:mpeg2, 1:mpeg1]
% MBsf: Macroblock skipped factor (nnz(MB) para considerarlo codificado)

global dc_dct_pred;
dc_dct_pred = [0 0 0];
global PMV;
PMV(1,1,:) = [0 0]; % PMV Forward
PMV(1,2,:) = [0 0]; % PMV Backward
f_out = fopen(filename,'w');

hwb = waitbar(0,'Coding Bitstream');

global DClumtab
load DClumtab.mat;
global DCchromtab
load DCchromtab.mat;
global dct_code_tab
load dct_code_tab.mat; % Table B-15 (mpeg1)
global dct_code_tab_14
load dct_code_tab_14.mat; % Table B-14 (mpeg2)
global scan;
load scan.mat;
global cbp_VLC;
load cbp_VLC.mat; % coded_block_pattern_VLC
global mai_VLC;
load mai_VLC.mat; % macroblock_address_increment_VLC
global alternate_scan;
alternate_scan = zigzag-1;
global f_code;
f_code_h=ceil(log2(mpeg.params.searchnotI(2)));
f_code_v=ceil(log2(mpeg.params.searchnotI(1)));
f_code = max(f_code_h,f_code_v); %  pag 94



global debug;
if (debug)
    display('. - New slice');
    display('o -  Macroblock with only motion vectors (no residue coded)');
    display('x -  Macroblock with only residue (no motion vectors coded)');
    display('* -  Skipped macroblock');
    display('+ -  Would be skipped if not the first/last mb in the slice');
end

bits = [];
bsize = [];
BitStream = [];
BitStream = Alignbits(BitStream);
BitStream = PutSequenceHeader(BitStream,mpeg.params.sizefram(2),mpeg.params.sizefram(1),mpeg.params.framerate);
BitStream = Alignbits(BitStream);
BitStream = PutSequenceExtension(BitStream);
BitStream = Alignbits(BitStream);
%BitStream = PutSequenceDisplayExtension(BitStream); % Not needed in mpeg-2
%BitStream = Alignbits(BitStream);

temporal_reference = 0;
for frame=1:1:size(mpeg.frames,2)
    macroblock_address_increment=1;
    if (mpeg.frames{frame}.type == 'I') % A new GOP starts
        BitStream = PutGOPHeader(BitStream);
        BitStream = Alignbits(BitStream);
        temporal_reference = 0; % reset to zero at start of each GOP
    end
    fprintf('Frame %d (%s) .',frame,mpeg.frames{frame}.type);
    BitStream = PutPictureHeader(BitStream,mpeg.frames{frame}.type,temporal_reference);
    temporal_reference = mod(temporal_reference+1,1024);
    BitStream = Alignbits(BitStream);
    if (mpeg.frames{frame}.type ~= 'I') mpeg.frames{frame}.intra_dc_precision=0; end
    BitStream = PutPictureCodingExtn(BitStream,mpeg.frames{frame}.type,mpeg.frames{frame}.intra_dc_precision,...
        mpeg.frames{frame}.quantiser_scale_type,intra_vlc_format);

    ind_slice = 0;
    for ind1 = 1:2:size(mpeg.frames{1,1}.blockqdcty,3) % #blocks in rows
        for ind2 = 1:2:size(mpeg.frames{1,1}.blockqdcty,4) % #blocks  in cols
            canskip=1;
            if(ind2 == 1) % First macroblock of a slice
                BitStream = Alignbits(BitStream);
                canskip=0;
                if ( mpeg.frames{frame}.type == 'B' )
                    rMBY=[]; rMBCb=[]; rMBCr=[]; rmvf=[]; rmvb=[]; % erase references
                end
                BitStream = PutSliceHeader(ind_slice, BitStream,mpeg.frames{frame}.quantiser_scale_code,mpeg.frames{frame}.type); % 1 row 1 slice
                ind_slice = mod(ind_slice + 1,175);
                dc_dct_pred=[0,0,0]; % Reset always to 0 (we are working with quantized coeffs with zero in the middle of the range - pag 78)
                PMV(1,1,:) = [0 0]; % Reset PMV Forward
                PMV(1,2,:) = [0 0]; % Reset PMV Backward
            end
```

```
            if( ind2 == size(mpeg.frames{1,1}.blockqdcty,4)-1 ) % Last macroblock of a slice
                canskip=0;
            end

            MBY = [mpeg.frames{frame}.blockqdcty(:,:,ind1,ind2) mpeg.frames{frame}.blockqdcty(:,:,ind1,ind2+1); ...
                mpeg.frames{frame}.blockqdcty(:,:,ind1+1,ind2) mpeg.frames{frame}.blockqdcty(:,:,ind1+1,ind2+1)];
            MBCb = mpeg.frames{frame}.blockqdctcbcr(:,:,round(ind1/2), round(ind2/2),1);
            MBCr = mpeg.frames{frame}.blockqdctcbcr(:,:,round(ind1/2), round(ind2/2),2);

            switch mpeg.frames{frame}.type % Macroblock_type (page 154)
                case {'I'}
                    [BitStream macroblock_address_increment]= PutMacroBlock(BitStream,MBY,MBCb,MBCr,MBsf,intra_vlc_format,...
                        mpeg.frames{frame}.intra_dc_precision,mpeg.frames{frame}.quantiser_scale_code,...
                        mpeg.frames{frame}.type,macroblock_address_increment,'','',canskip,'','','','','');

                case {'P'}
                    mvf=squeeze(mpeg.frames{frame}.mvf(round(ind1/2),round(ind2/2),:));
                    [BitStream macroblock_address_increment] = PutMacroBlock(BitStream,MBY,MBCb,MBCr,MBsf,'',...
                        mpeg.frames{frame}.intra_dc_precision,mpeg.frames{frame}.quantiser_scale_code,...
                        mpeg.frames{frame}.type,macroblock_address_increment,mvf,'',canskip,'','','','','');

                case {'B'}
                    mvf=squeeze(mpeg.frames{frame}.mvf(round(ind1/2),round(ind2/2),:));
                    mvb=squeeze(mpeg.frames{frame}.mvb(round(ind1/2),round(ind2/2),:));
                    [BitStream macroblock_address_increment] = PutMacroBlock(BitStream,MBY,MBCb,MBCr,MBsf,'',...
                        mpeg.frames{frame}.intra_dc_precision,mpeg.frames{frame}.quantiser_scale_code,...
                        mpeg.frames{frame}.type,macroblock_address_increment,mvf,mvb,canskip,rMBY,rMBCb,rMBCr,rmvf,rmvb);
                    rMBY=MBY; rMBCb=MBCb; rMBCr=MBCr; rmvf=mvf; rmvb=mvb; % save references
            end
        end
        fprintf('.');
    end

    BitStream = Alignbits(BitStream);
    out_cod = reshape(BitStream,8,length(BitStream)/8)';
    out_cod = bin2dec(out_cod);
    fwrite(f_out,out_cod,'uint8');
    bits=[bits BitStream];
    bsize = [bsize ; length(BitStream)];
    BitStream=[];
    disp(' Done');
    waitbar(frame/size(mpeg.frames,2),hwb,'Coding Bitstream');
end

% Put Sequence End code
BitStream = Alignbits(BitStream);
BitStream = [BitStream  dec2bin(439,32)];
bits=[bits BitStream];
%bsize = [bsize ; length(BitStream)]; %These bits don't appear in graph

out_cod = reshape(BitStream,8,length(BitStream)/8)';
out_cod = bin2dec(out_cod)';
fwrite(f_out,out_cod,'uint8');
fclose(f_out);

close(hwb)
```

Published with MATLAB® R2014a