Práctica 2:
Compresión de imágenes JPEG. Coeficientes DC

Resumen

En esta práctica y la siguiente se pretende que el alumno realice un codificador de imagen JPEG con Matlab cuyo resultado sea interpretado correctamente por cualquier aplicación estándard que acepte ficheros de imagen.

1. Introducción

El objetivo de las dos prácticas es realizar un codificador de imagen JPEG en Matlab. Se pretende con ello profundizar y consolidar los conocimientos adquiridos en clase de teoría. La validación del trabajo desarrollado se hará visualizando la imagen con cualquier programa de imágenes, por ejemplo el Paint, Photoshop, PowerPoint, etc, o bien, desde el mismo Matlab con la función `imread`. Dado que la tarea es relativamente compleja, se propone partir de un caso simple e ir ampliando la función a realizar en Matlab hasta conseguir implementar todo el proceso JPEG Baseline completo.

Uno de los objetivos de la práctica es que el alumno aprenda a buscar por sí mismo la información que necesita para realizar la tarea, por lo que el guión de la práctica no es exhaustivo. No obstante, para que la tarea no tenga una dificultad inicial grande, se proponen varios pasos. En los primeros, parte del trabajo necesario para obtener un fichero jpeg que sea estándard, en concreto la cabecera, se da ya hecha en forma de fichero binario que debe ser leído por la función que desarrolla el alumno e incorporado al fichero de salida que se debe generar como resultado.

El primer paso será generar en Matlab una imagen sencilla de tres bloques de 8x8 con tres niveles de gris distintos pero de valor constante en cada bloque. Se proporciona la cabecera binaria con las tablas de Huffman y de cuantificación.

El segundo paso consistirá en manejar distintos tamaños de imagen y el tercero considerar la componente continua de la crominancia. En la siguiente práctica, el resto de pasos tienen como objetivo realizar una función totalmente operativa en la que, dada una imagen cualquiera, se genere el fichero jpeg correspondiente, para lo cual, además del trabajo realizado en los pasos anteriores, se deberá generar la cabecera adecuada, realizar el zigzag de los coeficientes de alterna, codificarlos, etc.

Matlab ofrece un entorno de programación más sencillo que lenguaje C pero mucho más lento a la hora de procesar, por lo que será interesante trabajar con imágenes pequeñas para que los tiempos de ejecución sean aceptables. Para no tener que cambiar de cabecera, aunque en algunos momentos no se trabaje con color, siempre codificaremos tomando como imagen de partida una tipo RGB.

Los resultados de la prácticas serán las funciones mijpeg_pasoN y se deberán enviar por email al profesor de laboratorio justo antes de finalizar cada sesión.

2. Paso 1. Imagen de 3 bloques constantes

Con el objeto de tener un primer paso sencillo y que el resultado sea un fichero pequeño no muy complicado de comprobar, se propone realizar una función que genere el fichero jpeg de una imagen de 3 bloques, RGB, pero con las tres componentes iguales(imagen de grises) para que las de crominancia sean nulas, donde en cada bloque de 8x8 todos los pixels sean iguales de modo que sólo tengamos que ocuparnos, de momento, de la componente DC. Asímismo, para que este paso resulte lo más sencillo posible no será necesario ocuparse de generar los distintos campos de la cabecera ni de las tablas de Huffman ya que toda la información necesaria, para esta imagen en concreto que se propone, se os da confeccionada en un fichero de cabecera binario y las tablas de Huffman estándard a utilizar están en la función `tab_huf_std.m`.

En el fichero `demo_paso1.m`, teneis creada la imagen mediante la siguiente instrucción

```
im1=[zeros(8,8), 255*ones(8,8), 128*ones(8,8)];
imC = ones(8,24,3);
imC(:,:,1)=im1;
imC(:,:,2)=im1;
imC(:,:,3)=im1;
```

Figura 1: Imagen tres bloques grises

y a continuación se encuentra la llamada a la función `mijpeg_paso1`

La función `mijpeg_paso1` la debeis realizar vosotros y debe tomar la variable imC como entrada e, implementando el proceso de codificación, escribir un fichero de salida de nombre, por ejemplo, out_paso1.jpg, `mijpeg_paso1(imC,'out_paso1.jpg')`. Se os proporciona un esqueleto de la función junto con algunos ficheros necesarios. Entre ellos está la cabecera, que incluye la descripción binaria de las tablas de Huffman que se propone utilizar, así como las tablas de cuantificación y los datos relativos a la imagen, tamaño, número de capas, qué tabla utiliza cada componente, el número de bits, etc. El fichero que contiene la cabecera se llama `cabecera_paso1.bin`. A partir del esqueleto se debe crear la función `mijpeg_paso1` completando las tareas necesarias indicadas con *XXXX REALIZAR*. Lo primero es la división del resultado de la DCT por la tabla de cuantificación. Lo segundo es redondear. Depués hay que proceder al cálculo del coeficiente DC diferencial, todo ello sólo para la luminancia. La codificación de los coeficientes DC de las componentes Cb y Cr, así como los coeficientes AC de las tres componentes Y, Cb, Cr están incluidos en el esqueleto de la función y se les da el valor de EOB (End Of Block), que, en nuestro caso, implica nivel 0 en todos ellos. La funcion `code_add_bits(DC)` genera los bits addicionales que corresponden en la codificacion del coeficiente DC. Las tablas utilizadas son las indicadas en el anexo K de la norma ISO/IEC

10918-1 (ver documento suministrado norma_itu-t81.pdf, pag 143). Nótese también que, antes de comenzar a codificar, se ha realizado la conversión de RGB a YCbCr.

2.1. Comprabación del funcionamiento

Con el objetivo de ayudar a que la función se implemente correctamente, a continuación se detalla el trabajo que ésta debe ir realizando.

Primer bloque, luminancia: Como todos los pixels tienen nivel cero en RGB, al pasar a YCbCr y ubicarlos en el margen -128, +127, la Y de todos los pixels del primer bloque es -128. En el segundo bloque es +127, y en el tercero cero.

El coeficiente DC de la DCT2D del primer bloque debe dar -1024. Al dividirlo por el coeficiente de cuantificación (16) y redondear debe dar -64. Al restar el anterior, como es el primero, el resultado es -64, por tanto se codifica con el par((nbits=7, codigo 11110) , (bits adicionales 011 1111)). A continuación los bits correspondientes al código EOB, en este caso 1010, ya que todos los coeficientes de alterna de la luma del primer bloque son cero. Primer bloque, crominancia Cb: Todo es cero. Los bits que genera el codificador para el coeficiente DC cero es 00 y luego como todos los coeficientes de alterna son cero se pone el código EOB que también resulta ser 00. Primer bloque, crominancia Cr: Ocurre lo mismo que con Cb, por tanto 0000. **Segundo bloque, luminancia:** Como se ha dicho antes, la Y de todos los pixels es +127. El coeficiente DC de la DCT2D es 1016. Al dividirlo por 16 y redondear da 63. Al restar éste menos el coeficiente DC anterior da 63-(-64)=127, que se codifica con el par((nbits=7, codigo 11110), (bits adicionales 111 1111)). Lo que sigue para los coeficientes de alterna es igual que en el primer bloque 1010. Segundo bloque, crominancia Cb: Igual que en el primer bloque 0000. Segundo bloque, crominancia Cr: Igual que en el primer bloque 0000.

Tercer bloque, luminancia: La Y es cero. El coeficiente DC de la DCT2D, dividido y redondeado también resulta cero. Al restar el anterior tenemos 0-63, que se codifica con el par ((nbits=6, codigo 1110), (bits adicionales 00 0000)). Los coeficientes de alterna, igual que en el primer bloque, se codifican con 1010. Tercer bloque, crominancia Cb: Igual que en el primer bloque 0000. Tercer bloque, crominancia Cr: Igual que en el primer bloque 0000. Como lo cantidad de bits no es multiplo de 8 y faltan 2 bits para que lo sean, al final, se añaden dos unos.

Bloque	Y dc n	Y dc add	Y ac	Cb dc	Cb ac	Cr dc	Cr ac
1	1 1110	011 1111	1010 (EOB)	00	00 (EOB)	00	00 (EOB)
2	1 1110	111 1111	1010 (EOB)	00	00 (EOB)	00	00 (EOB)
3	1110	00 0000	1010 (EOB)	00	00 (EOB)	00	00 (EOB)

Cuadro 1: Bits resultantes de la codificicación de la imagen del paso 1.

3. Paso 2. Imagen de cualquier tamaño, bloques constantes

Ahora se utilizará el fichero `demo_paso2.m` para la generación de la imagen imC que tiene 8 bloques de 8x8 cada uno, con sus pixels iguales y correspondientes a los siguientes colores: blanco, amarillo, cian, verde,magenta, rojo, azul y negro.

Figura 2: Carta de colores, 8 bloques constantes. El primero es blanco.

Realizar una copia de la función `mijpeg_paso1` con el nombre `mijpeg_paso2` y ampliadla para que modifique en la cabecera el tamaño de la imagen, acorde con las dimensiones de la imagen de entrada (para no complicar, que sea siempre múltiplo entero de 8). En nuestro caso, el byte más significativo del número de líneas se debe escribir como un `uint8` en la posición 626 (271 hex) de la variable `cabecera` y el menos significativo en la 627 (272 hex). El número de columnas se debe escribir del mismo modo en las dos posiciones siguientes.

En la ejecución de `demo_paso2.m` se prueba la codificación sobre la imagen imC y también sobre `lena_256_color.tif`. Se puede comprobar que, en ambos casos, la imagen decodificada sólo contiene el coeficiente DC de la luminancia de cada bloque de 8x8 puesto que la crominancia todavía no ha sido codificada (se hace en el siguiente paso).

4. Paso 3. Coeficiente DC de la crominancia

Ahora se utilizará el fichero `demo_paso3.m` , que es casi idéntico a `demo_paso2.m`. Realizar una copia de la función `mijpeg_paso2` con el nombre `mijpeg_paso3` y ampliadla para que realice la codificación de los coeficientes DC de la crominancia. Comprobad el funcionamiento mediante `demo_paso3.m`. Observad que se sigue decodificando únicamente el coeficiente DC de cada bloque de 8x8, pero ya aparece la crominancia.