

Práctica 2. Codificación sin pérdidas

Resumen

En esta práctica programarás un codificador sin pérdidas para imágenes digitales que utiliza un código de longitud fija.

1. Codificador sin pérdidas de imágenes monocromáticas

En esta parte de la práctica consideraremos la codificación sin pérdidas de una imagen digital utilizando un código de longitud fija. Supón que \mathbf{x} es la matriz de valores *uint8* de una imagen digital monocromática. Queremos codificar \mathbf{x} sin pérdidas utilizando un código y almacenar la secuencia de bits generada en disco (figura 1). Para ello, un codificador debe generar un fichero que contenga la secuencia de bits que resulta de la codificación de \mathbf{x} . A partir de dicho fichero, un decodificador debe ser capaz de decodificar la secuencia de bits y visualizar correctamente la imagen obtenida \mathbf{x} .

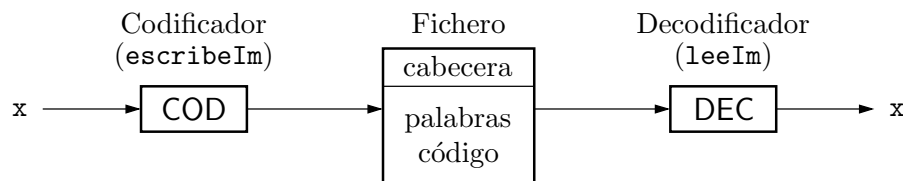


Figura 1. Codificación y decodificación sin pérdidas de \mathbf{x} .

Como cada píxel de \mathbf{x} puede tomar 256 valores, podemos codificar \mathbf{x} con un código de longitud fija de 8 bits ($\log_2 256 = 8$). Aparte de la palabra código de cada píxel de \mathbf{x} , el codificador debe incluir en el fichero toda la información que sea necesaria para una correcta decodificación y visualización de \mathbf{x} . Esta información adicional se incluye en la *cabecera* del fichero (figura 1).

La función `escribeIm` codifica \mathbf{x} con un código de longitud fija de 8 bpp y genera un fichero cuyo nombre es el especificado en la variable `fi`. El código de esta función es:


```

function escribeIm(x,nombre)
    [f c] = size(x);           % Obtiene el número de filas y de columnas
    fid = fopen(nombre,'w');    % Abre el fichero en modo escritura
    fwrite(fid,f,'uint16');     % Escribe el número de filas en el fichero
    fwrite(fid,c,'uint16');     % Escribe el número de columnas en el fichero
    fwrite(fid,x,'uint8');      % Escribe las palabras código en el fichero
    fclose(fid);               % Cierra el fichero
end

```


En la función `escribeIm` tenga en cuenta que:

- Recuerda que `uint8` y `uint16` son las clases de MATLAB para enteros sin signo de 8 y 16 bits, respectivamente.
- La información incluida en la cabecera del fichero generado es el número de filas (`f`) y de columnas (`c`) de `x`. Cada una de estas cantidades se codifica con 16 bits.
- La apertura del fichero (con `fopen`) proporciona un *identificador de fichero* (`fid`). Cualquier operación posterior sobre el fichero abierto, como por ejemplo su escritura (con `fwrite`) o su clausura (con `fclose`), debe especificar su identificador como argumento.
- Cuando la variable `x` a escribir en un fichero es un vector, la función `fwrite` escribe los elementos de `x` secuencialmente. Cuando `x` es una matriz, como por ejemplo en la última instrucción `fwrite` de `escribeIm`, la función `fwrite` escribe los valores de `x` en el fichero por columnas: primero escribe la primera columna de `x`, a continuación escribe la segunda columna de `x`, y así sucesivamente hasta la última columna.
- La codificación de la imagen es trivial. Como la clase de `x` (`uint8`) representa los los valores de `x` con un código de longitud fija de 8 bits, que es el tipo de código con el queremos codificar la imagen, la codificación se realiza con una instrucción `fwrite`.

 Al codificar con la función `escribeIm`,

¿Cuál es el número máximo de filas que puede tener la imagen a codificar?

¿Y el máximo número de columnas?

 Si `x` es la luminancia de la imagen que contiene dicho fichero `i1.png`, ¿cuál debería ser el tamaño del fichero generado? bytes

Usando `escribeIm`, genera un fichero de nombre `li1` que contenga la **luminancia** de `i1` codificada. **Utilizando el explorador de archivos de tu sistema operativo**, obtén el tamaño del fichero que has generado y comprueba que coincide con el que has calculado.

Ten en cuenta que el fichero que has generado no es un fichero de texto. Por tanto, no tiene sentido abrir dicho fichero con un procesador de textos. Para obtener la información que contiene es necesario utilizar un decodificador específico.

2. Decodificador

En esta parte de la práctica escribiremos una función para decodificar los ficheros generados con `escribeIm`. La primera línea de la función debe ser

```
function y = leeIm(nombre_sb)
```

donde

- `nombre_sb` es el nombre del fichero con la secuencia de bits a decodificar
- `y` es la matriz que contiene la imagen decodificada

La función debe decodificar la imagen y visualizarla en escala de grises. Para ello debe utilizar, entre otras, las funciones `fopen` (para abrir el fichero), `fread` (para leer el contenido del fichero) y `fclose` (para cerrar el fichero).

La instrucción `fread` tiene los siguientes argumentos:

```
a = fread(fid,size,precision)
```


donde

- `fid` es el identificador del fichero del que leemos los datos
- `size` especifica el número de datos que leemos
- `precision` es la clase de datos que leemos
- `a` es el array con los datos leídos

El argumento `size` puede ser de dos formas:

- `N` (`a` es un vector columna de `N` elementos)
- `[M,N]` (`a` es una matriz `M×N` que se rellena por columnas)

La función `leeIm` debe leer la información del fichero en el mismo orden con el que la función `escribeIm` la escribió en él. Por tanto, `leeIm` lee primero el número de filas, después lee el número de columnas y finalmente lee la matriz con los valores de la imagen.

 Ejecuta `y = leeIm('li1')` y comprueba que la imagen decodificada se visualiza correctamente. ¿La imagen decodificada coincide con la imagen original `x`?

3. Codificador y decodificador de imágenes RGB

En esta sección, modificaremos las funciones `leeIm` y `escribeIm` para que también puedan codificar y decodificar sin pérdidas imágenes RGB de 24 bpp.

La primera línea de la función del codificador debe ser:

```
function escribeRGB(nombre_im,nombre_sb,color)
```

donde

- `nombre_im` es el nombre del fichero que contiene la imagen a codificar
- `nombre_sb` es el nombre del fichero que contiene la secuencia de bits generada
- `color` indica si queremos codificar las componentes R, G y B de la imagen o únicamente la componente Y. En el primer caso, `color` debe tomar el valor 1 y en caso contrario debe tomar el valor 0.

Cuando la imagen es RGB, la función `escribeRGB` debe almacenar primero los valores de componente R, después los de la componente G y finalmente los de la componente B.


La primera línea de la función del decodificador debe ser

```
function leeRGB(nombre_sb)
```

donde `nombre_sb` es el nombre de la secuencia de bits a decodificar. Una vez decodificada la imagen, la función `leeRGB` debe mostrarla en pantalla usando la instrucción `imshow` (asegurate de que el argumento de `imshow` es `uint8`). La imagen se mostrará en gama de grises si se codificó con la opción `color=0`.

Codifica y decodifica la imagen `i1.png` de las dos formas posibles (`color=1` y `color=0`) y verifica que funciona correctamente.

Un *formato de fichero de imagen* es un estándar de codificación que permite almacenar imágenes digitales en archivos. Cada formato especifica el tipo de imágenes que pueden codificarse, la información que forma parte de la cabecera, la sintaxis de la secuencia de bits y las técnicas que pueden utilizarse para codificar las componentes de la imagen. Consulta la entrada *Image file formats* de Wikipedia para ver una lista de los formatos más utilizados.

 Al crear `escribeRGB` y `leeRGB` has definido un formato de fichero de imagen muy simple. En dicho formato:

¿En qué espacio de color tiene que estar la imagen a codificar?

¿Cuál es el número máximo de filas que puede tener la imagen de entrada?

¿Y el máximo número de columnas?

¿Qué información forma parte de la cabecera?

¿Cuál es la sintaxis de la secuencia de bits?

¿Con qué técnica se codifican las componentes de la imagen?


¿Cuál es la tasa binaria en bpp?

Las imágenes utilizadas en las prácticas de TDSC están codificadas en el formato PNG (*Portable Network Graphics*). Utilizando el **explorador de archivos**, obtén el tamaño del fichero `i1.png` y calcula la tasa binaria expresada en bpp.

 Tamaño del fichero: bytes Tasa binaria: bpp

La tasa binaria en PNG es significativamente inferior a la que proporciona el codificador implementado en esta práctica. Nuestro codificador es ineficiente en tasa binaria porque:

- La técnica de codificación utilizada es un código. Esta técnica no explota la dependencia que hay entre los píxeles de la imagen puesto que cada símbolo se codifica de manera independiente de los demás.
- El código usado es de longitud fija. Este tipo de códigos no tiene en cuenta las probabilidades de los símbolos a codificar.

 El codificador que hemos implementado es ineficiente en tasa binaria si lo comparamos con PNG u otras técnicas más sofisticadas, pero tiene algunas ventajas. ¿Cuáles son?