## Fundamentos de Sonido e Imagen.

### Grado en Ingeniería de Tecnologías de Telecomunicación
### Curso 2024-25

# Práctica I de imagen: colorimetría.

**Objetivos:** en esta práctica se pretende introducir las funciones de matlab que leen y manejan imágenes. También se trata de ver diferentes formatos de imagen y de relacionarlos con los conceptos de teoría del color del primer tema.

## 1 .- Introducción:

En esta práctica deberemos recordar los conocimientos básicos de la herramienta matlab obtenidos en otras asignaturas. Ahora se trata de aprender las funciones básicas de matlab para manejar imágenes. El material se encuentra en Moovi. Se recomienda crear un archivo .mlx para ordenar mejor los diferentes apartados de la práctica.

### 2 .- Leer Imágenes:

El programa matlab admite extensiones que reciben el nombre de toolboxes. Una toolbox (caja de herramientas) es un conjunto de funciones que nos ayudan a trabajar sobre un tema determinado. El matlab instalado en el laboratorio tiene instaladas las toolboxes de procesado de señal y de procesado de imagen. Para leer imágenes desde matlab se usa una función de la toolbox de imagen: **`imread`**. Por ejemplo:

```
im1 = imread('lenna.bmp');
% Leer la imagen del fichero lenna.bmp
```

Una vez que hemos leído una imagen podemos hacer otras operaciones con ella. Por ejemplo, verla con la función **`imshow`**. Para eso, debéis hacer:

```
imshow(im1);
```

A lo mejor, no hemos pensado en ello pero ahora **im1** es una variable de matlab que contiene una imagen digital (por supuesto, todas las imágenes de matlab son digitales). Fijaros en lo que ocurre si le preguntáis a matlab el tamaño de esa variable:

```
size(im1)
```

Resulta que matlab ve a la imagen como una matriz de números. Cada posición de la matriz está asociada a un punto luminoso de la imagen (pixel) y su valor representa el color de cada punto. Lo que acabamos de hacer con la función **size** es averiguar el tamaño de la imagen, el primer valor es el número de filas (alto de la imagen) y el segundo valor es el número de columnas (ancho de la imagen). Fijaros que esta es una imagen en escala de grises (blanco y negro) y el color de cada punto se puede representar con un único número (llamado luminancia o brillo, variable Y). La luminancia va entre el valor mínimo (0, para el negro) y el valor máximo (255, para el blanco).

Comprobad los valores máximo y mínimo de luminancia. Por ejemplo, el máximo se puede calcular así:

```
max(max(im1))
```

¿Qué hubiera pasado si hacemos **max(im1)**? ¿Por qué? ¿Y si hacemos **max(im1(:))**? ¿Qué es **im1(:)**?

## 3 .- Formatos de Imagen en Escala de Grises:

La imagen en escala de grises que acabamos de leer tiene el blanco en 255 (ojo, este valor no tiene por qué alcanzarse, ¿se alcanza en la imagen con la que estamos trabajando?). Eso es así porque matlab lee las imágenes, por defecto, en un formato de 8 bits llamado **uint8** (unsigned integer of 8 bits). Nótese que 8 bits, sin signo, permiten un rengo entre 0 y $2^8$-1=255.

**MUY IMPORTANTE:** al cargar archivos de imágenes se asignarán a variables de tipo uint8, que es un entero sin signo de 8 bits con valores entre 0 y 255, como acabamos de decir. Otro formato es el llamado **double** que representa los grises con un número real entre 0 y 1 (0 para el negro, 1 para el blanco). Es importante que en cada momento sepáis en que formato de datos estáis trabajando para evitar problemas de cálculo y representación.

Se puede convertir entre ambos formatos con las funciones **im2double** e **im2uint8**.

Por ejemplo:

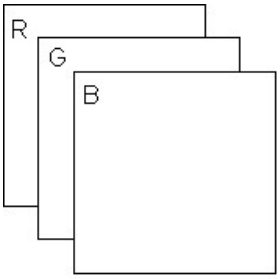
```
im2 = im2double(im1); % Ahora la imagen es double
imshow(im2); % "imshow" funciona igual
imshow(im2/2); % Intentad esto con im1
% La última instrucción nos enseña la imagen con
% un 50% menos de contraste, ¿qué significa esto?
```

## 4 .- Imágenes en Color:

Las imágenes en color se leen aparentemente de la misma manera:

```
im3 = imread('mandril.bmp');
```

Si hacemos **size(im3)** nos daremos cuenta de la diferencia. Esta vez la función **size** devuelve 3 números y el tercero vale 3. Eso se debe a que im3 es una matriz con tres dimensiones. Los dos primeros valores de **size** tienen el mismo significado de antes. Tres es el número de componentes necesarias para representar un color, por eso la tercera dimensión es 3. Se trata de que tenemos como tres imágenes en paralelo, la primera es la componente roja, la segunda la verde y la tercera la azul.




Por ejemplo, el pixel (10,10) de esta imagen (el que dista 10 pixeles en horizontal de la esquina superior izquierda y otro tanto en vertical) tiene por componentes RGB: **`im3(10,10,:)`**. Donde la componente roja es **`im3(10,10,1)`**, la verde es **`im3(10,10,2)`** y la azul es **`im3(10,10,3)`**.

Si queremos visualizar una imagen en color, se hace igual que en el caso anterior:

```
imshow(im3);
```

Una operación que vamos a necesitar es la separación de una imagen en sus tres matrices componentes, esto se hace así:

```
R = im3(:,:,1);
G = im3(:,:,2);
B = im3(:,:,3);
```

Cada una de estas componentes la podemos visualizar como si fuera una imagen blanco y negro individual. Esto puede parecer que no tiene mucho sentido pero nos permite saber cuando una de las componentes es grande o pequeña[1]. Ejemplo:

```
figure;imshow(R);title('Componente roja.');
figure;imshow(G);title('Componente verde.');
figure;imshow(B);title('Componente azul.');
```

Las imágenes en color pueden ser del tipo **`uint8`** o **`double`** igual que las anteriores. Cuando las leemos de fichero, por defecto, estarán en formato **`uint8`**. Podemos comprobar eso fácilmente viendo los máximos y mínimos de cada componente, por ejemplo para la componente roja:

```
max(max(im3(:,:,1)))
```

¿Cómo se calcula el mínimo? ¿Cómo se haría para otras componentes?

Para convertir la imagen **`uint8`** a double se usa la misma función im2double que se puede aplicar a toda la imagen o a componentes separadas:

```
im4 = im2double(im3); % Toda la imagen
r = im2double(R);
g = im2double(G);
b = im2double(B); % Componente a componente
```

Probar: **`imshow(im4/2)`**.

[1] Hay otro método mucho mejor para hacer esto. Ver apartado 5, ejercicio A.

## 5 .- Ejercicios:

A) Visualización de Componentes RGB:

* La siguiente función visualiza una imagen en color (**imc**) y sus componentes R,G,B en sus colores originales (rojo, verde y azul). Tecleadla, probadla y modificadla para que en vez de R, G y B genere tres componentes: una con la parte azul a cero (componente amarilla), otra con la parte verde a cero (componente magenta) y otra con el rojo a cero (componente cyan).

```
function img_comp =  VisualizaComponentes(imc)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                               %
% Funcion que visualiza componentes (R,G,B)     %
% en su color original.                         %
%                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
imR = imc;
imG = imc;
imB = imc; % Inicio
imR(:,:,[2 3]) = 0; % Parte roja
imG(:,:,[1 3]) = 0; % Parte verde
imB(:,:,[1 2]) = 0; % Parte azul
img_comp = [imc imR imG imB]; % Concatenar imagenes
imshow(img_comp); % Display
```

B) Calcular Componentes R, G y B de un Punto[2]:

* Crear una función matlab que devuelva las tres componentes R, G y B del punto **(x,y)** de una imagen en color. Las coordenadas se miden desde la esquina superior izquierda (hacia la derecha y hacia abajo) y deben empezar en el valor 0. La cabecera debería ser:

```
function [R,G,B] = ComponentesPunto(imc,x,y)
```

C) Convertir Imagen RGB en Escala de Grises:

* Conociendo la fórmula que relaciona la luminancia con las componentes de color:

$$Y = 0.3*R+0.59*G+0.11*B$$

Crear una función que convierta una imagen RGB en la imagen equivalente en escala de grises. La cabecera debería ser:

```
function imbn = ImagenGrises(imc)
```

Comparar esta función con la función de matlab **rgb2gray**.

[2] Se recomienda que empecéis este ejercicio (y los siguientes) desde cero, intentar modificar el código de la función anterior puede no ser lo más fácil.

D) Probar el Efecto Visual del Ruido:

* Estudiar la ayuda de la función **imnoise** (teclear "**help imnoise**") y probarla con las imágenes **festiva.bmp** y **t_toons.bmp**. El resultado debería ser similar a la figura 4.13 de los apuntes de imagen Probar varios tipos de ruido aunque el más significativo es el gaussiano.

E) Hacer el Negativo de una Imagen RGB:

* Hacer una función que calcule el negativo de una imagen en escala de grises. Si tenemos una imagen en escala de grises (una matriz de luminancia, I) su negativo es:

$$I' = max\_Y - I$$

Donde max_Y es el máximo valor de luminancia para el formato de I (255 si I es **uint8** y 1 si I es **double**). La cabecera podría ser así:

```
function imneg = NegativoGrises(im)
```

Comparar esta función con la función de matlab **imcomplement**. Repetir el ejercicio para imágenes en color (RGB). ¿Qué pasa si complementamos una imagen dos veces?

F) Intercambiar dos Planos de una Imagen RGB:

* Hacer una función con cabecera:

```
function imc2 = SwapRB(imc)
```

Donde la imagen de salida tiene:

- Por componente R, la componente B de la original.
- Por componente B, la componente R de la original.
- La componente G igual que en la original.

G) Aumentar una Imagen hasta el máximo Contraste:

* Se trata de aumentar al maximo el contraste de una imagen en escala de grises. Primero, hay que calcular el valor mínimo de luminancia (llamadle **m**) y también el máximo (**M**). Queremos que el mínimo pase a ser min_Y (0 para imágenes **uint8** y **double**) y que el máximo pase a ser max_Y (255 para imágenes **uint8** y 1 para **double**). La fórmula a aplicar sería:

$$\mathbf{I'} = \frac{\mathbf{I} - m}{M - m} max\_Y + min\_Y$$

H) Reducir el Número de Niveles de Gris de una Imagen:

* Teclear la siguiente función:

```
function img2 = ReduceGrises(img,N)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                %
% Funcion que reduce a N niveles de gris.  %
%                                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
imaux = im2double(img); % Pasar a real
if (N>=256)
    M = 255;
elseif (N<=2)
    M = 1;
else
    M = N-1;
end % El maximo de la imagen es el numero de niveles menos 1.
imaux2 = uint8(imaux*M);
      % Imagen con N niveles de gris (enteros de 0 a M=N-1).
img2 = double(imaux2)/M; % Imagen final
```

Esta función produce una imagen con N niveles de gris y la vamos a usar para comprobar cuál es el número de niveles significativos para el ojo humano. Leer en una variable la imagen "lenna.bmp" (si no la tenéis ya cargada), visualizar esa imagen (con ayuda de esta función) con los siguientes números de niveles: 256, 200, 128, 64, 40, 30, 20, 10 y 2. ¿A partir de qué valor se empieza a notar pérdida de calidad.