eman ta zabal zazu

UPV, Facultad de Informática

Dpto. de Arquitectura y Tecnología de Computadores

PROCESADO DIGITAL DE SEÑAL

PROYECTO ESPECÍFICO - LABORATORIO 3 (SIm)

SEÑALES – IMÁGENES

OBJETIVOS

Familiarizarse con el manejo de imágenes mediante MATLAB, en concreto:

- Utilización de los modos de almacenamiento más habituales: "color verdadero", paleta de color...
- Carga de imágenes desde fichero y visualización.
- Uso del histograma para realce de imágenes.

CONCEPTOS BÁSICOS

Antes de comenzar, vamos a revisar algunas ideas importantes para manejar imágenes:

- Una imagen es una matriz de puntos, cada uno de ellos de un color determinado.
- En imágenes monocromas se representa el color de cada punto mediante un número (normalmente de **8 bits**), que indica el brillo o nivel de gris de ese punto.
- En imágenes en color, para representar cada punto se necesitan las tres componentes de color que lo definen (R, G y B). Normalmente, se emplean 8 bits por componente (**24 bits** en total).
- Ahora bien, para almacenar la información del color de todos los puntos de la imagen tenemos dos opciones:
  a) Usar "**color verdadero**" (True-Color), es decir, guardar por cada punto de la matriz los 3 valores RGB. Normalmente, se utilizan 8 bits por color (24 bits), lo que supone unos 16 millones de colores diferentes.
  b) Usar una **paleta de color** (colormap), es decir, una tabla en la que se guardan los componentes RGB de los colores que aparecen en la imagen, y por cada punto de la imagen un índice que apunta a la posición de la paleta correspondiente al color del punto en cuestión. Normalmente en la paleta de color se guardan los colores usando **24 bits**, y el tamaño de la paleta suele ser de como máximo de **256 posiciones**, de forma que por cada punto de la imagen sólo necesitamos **8 bits.**

  La opción b) resulta incómoda si se quiere procesar la imagen, pero se utiliza para reducir el espacio utilizado, y supone una reducción en el número de colores distintos que existen en la imagen.
- Por otro lado, la información se puede guardar en un fichero con diferentes formatos (BMP, PCX, GIF, JPG, TIF ...). Salvo el BMP, los demás tratan de reducir el espacio ocupado mediante algún algoritmo de compresión. El formato GIF, por ejemplo, es bueno para comprimir dibujos y JPG y TIF para guardar fotografías.

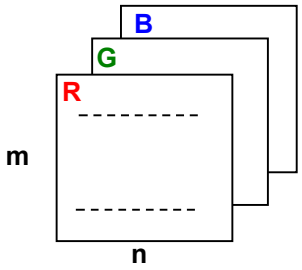
Imágenes en MATLAB

Las matrices, que son la estructura de datos básica de MATLAB, se prestan de forma natural al manejo de imágenes. Los elementos de estas matrices podrán ser de tipo `double`, o en caso de querer ahorrar espacio, `uint8` o `uint16`.

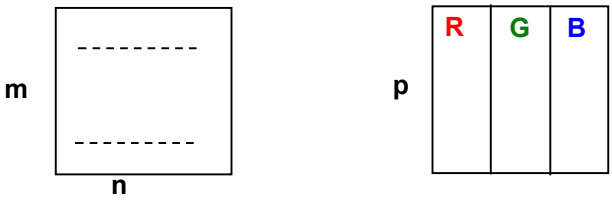
Las funciones del toolbox de procesado de imágenes distinguen 4 tipos básicos de imágenes, que en MATLAB se denominan:

- **Imagen RGB** (truecolor) corresponde a imágenes con color verdadero. Una imagen de mxn píxeles se guarda en **una matriz de 3 dimensiones mxnx3,** de forma que la componente roja (R), verde (G) y azul (B) de un pixel se guardan a lo largo de la tercera dimensión. Podemos pensar que la imagen está formada por tres planos, uno por cada componente de color. Los elementos de la matriz pueden ser:

- de tipo **double**, su valor estará, en este caso, dentro del rango [0, 1],
- o del tipo **uint8** (rango [0, 255]) o **uint16** (rango [0, 65535]).



- **Imagen indexada** (indexed image) corresponde a imágenes con paleta de colores. Una imagen de mxn píxeles se guarda en **una matriz de datos mxn,** y una **matriz paleta px3**, de forma que cada fila de la paleta contiene un color definido con sus componentes RGB y cada elemento de la matriz de datos contiene un **índice** que apunta a la paleta.
  - Los valores de la matriz de datos pueden ser **double** (donde el 1 apunta a la primera fila de la paleta, el 2 a la segunda, ...) o también, **uint8** o **uint16** (en estos dos últimos casos es el 0 el que apunta a la primera fila de la paleta).
  - Los valores de la paleta tienen que ser **double** y del rango [0, 1].



- **Imagen de intensidades** (intensity image) corresponde a imágenes monocromas. Una imagen de mxn píxeles se guarda en **una matriz de mxn,** de forma que los valores de la matriz representan el nivel de brillo o de gris de cada pixel. Estos valores pueden ser **double**, **uint8** o **uint16**. Puede considerarse un caso particular de la imagen RGB con los tres planos de color iguales (sólo guardamos uno). También, puede interpretarse como una imagen indexada con una paleta de colores (que no guardamos), que sólo contiene niveles de grises ordenados de negro a blanco.
- **Imagen binaria** (binary image) corresponde a imágenes con sólo dos colores, negro y blanco. Una imagen de mxn píxeles se guarda en **una matriz de mxn**, cuyos elementos sólo pueden tomar valor 0 o 1. Estos elementos pueden ser de tipo `double` o `uint8`, y deben tener el flag "logical" activado (se activa con la función **logical**).

CARGA DE IMÁGENES DESDE FICHERO EN MATLAB

MATLAB dispone de la función **imread** capaz de leer los formatos más comunes de ficheros de imagen. Para comprobarlo, vamos a leer algunos ficheros que se encuentran en la carpeta de la práctica. Para leerlos basta hacer:

```
[X1 pal1] = imread('girasol.jpg');  → truecolor
[X2 pal2] = imread('girasol.bmp');  → indexed
[X3 pal3] = imread('camion.bmp');   → indexed (paleta de grises)
[X4 pal4] = imread('moon.tif');     → monocroma sin paleta
[X5 pal5] = imread('circles.tif');  → binary
whos                                %para ver el tamaño de lo leído
```

Puede comprobarse que, en el primer caso, se trata de una imagen RGB, ya que se ha obtenido una matriz de 3 dimensiones (283 filas, 187 columnas y 3 componentes de color) y una paleta de color vacía. Es una imagen almacenada en formato “color verdadero”. Podíamos haber leído el fichero con el comando:

```
X1 = imread('girasol.jpg');
```

Pero, como a priori el nombre del fichero no nos permite saber de qué tipo de imagen se trata, lo más fácil es leer también la paleta, aunque luego resulte estar vacía. De todos modos, existe una función para obtener información sobre la imagen contenida en el fichero sin leerla. Se llama **`imfinfo`**.

En el segundo caso (se trata de la misma imagen, pero en modo indexado), el resultado es una matriz de 283 filas x 187 columnas, y una paleta de 256 colores, con lo que el espacio necesario para almacenar la imagen es menor. Puede sorprenderte que el fichero JPG ocupe menos que el BMP, pero esto es debido a que en JPG se comprime la información alterando la imagen original, aunque de forma casi inapreciable a la vista.

En el tercer caso, y aunque la imagen es monocroma, también obtenemos una imagen indexada. Lo que sucede es que la paleta de colores está formada sólo por niveles de gris, es decir, las componentes RGB de cada color son iguales.

El cuarto caso corresponde a una imagen monocroma sin paleta y el quinto a una imagen binaria.

Otro detalle importante se aprecia en el resultado del comando **`whos`**. Los elementos de las matrices imagen son de clase **`uint8`** para ahorrar espacio. El problema es que cuando una variable es de tipo **`uint8`**, MATLAB puede no admitir operaciones aritméticas sobre ella (depende de la versión utilizada). Para pasar de un tipo a otro se usan las funciones **`double(x)`**, o bien **`uint8(x)`**.

VISUALIZACIÓN DE IMÁGENES EN MATLAB

Para visualizar imágenes en MATLAB existen las funciones `image` e `imagesc`, pero el toolbox de procesado de imágenes nos ofrece la función **`imshow`**, más flexible y cómoda, que es capaz de manejar los 4 tipos de imagen descritos antes, distinguiendo de forma automática el tipo de almacenamiento:

- Si la imagen esta almacenada en una matriz **mxnx3** utiliza color verdadero, entonces interpreta los valores de los componentes de color según el tipo de datos. Si es `uint8` el rango va de 0 a 255 y si es `double` va de 0 a 1.

```
imshow(X1)
```

- Si es una matriz **mxn**, dependiendo de si le pasamos una paleta de colores como argumento la interpretará como indexada o como monocroma.

```
imshow(X2, pal2)
imshow(X3, pal3)
imshow(X4)
```

- Si es una matriz **mxn** de tipo logical, la interpretará como binaria.

```
imshow(X5)
```

En el caso de las imágenes monocromas, la función admite argumentos que permiten especificar rangos no habituales de valores de brillos. Esto puede verse en la ayuda de la función.

Para ver la paleta de colores podemos hacer;

```
colorbar
```

Para ver un fragmento ampliado puede utilizarse el comando `axis` o el zoom del menú de la figura.

Tras lo visto en este apartado y en el anterior, prueba a cargar y visualizar las imágenes que encuentres en la carpeta de la práctica.

CONVERSIÓN ENTRE TIPOS DE IMÁGENES EN MATLAB

En algunos casos nos interesará cambiar el modo de almacenamiento de una imagen, o convertirla en otra reduciendo la profundidad de color. El toolbox de procesado de imagen incluye funciones para ello (**ind2rgb**, **rgb2gray**, ...), pero puede ser instructivo realizar algunas conversiones sin utilizarlas.

Ejemplo 1:

Para hacer procesado de imágenes en color, en muchos casos, conviene utilizar color verdadero. Si no es así (por ejemplo, la imagen indexada X2), una posible manera de hacer el cambio es la siguiente:

```
Rpal = pal2(:,1);                  %columna de pal2 asociada al rojo
Gpal = pal2(:,2);                  %columna de pal2 asociada al verde
Bpal = pal2(:,3);                  %columna de pal2 asociada al azul
XX2 = X2 + 1;                      %matriz de índices (ha de comenzar en 1)
Y2 = cat(3,Rpal(XX2), Gpal(XX2), Bpal(XX2));
figure,imshow(Y2)
```

El resultado es una matriz mxnx3 con toda la información RGB y sin necesidad de paleta.

Ejemplo 2:

Otra posible conversión, sería el paso de una imagen RGB a una imagen monocroma. Como ejemplo, vamos ahora a convertir la imagen en color `X1` en una imagen monocroma y visualizarla:

```
R = (X1(:,:,1));          %matriz asociada al rojo
G = (X1(:,:,2));          %matriz asociada al verde
B = (X1(:,:,3));          %matriz asociada al azul
X1bn = (0.30*R + 0.59*G + 0.11*B);
figure, imshow(X1bn)
```

Como puedes ver, en `X1bn` hemos calculado el brillo o nivel de gris de cada punto mediante una media ponderada de los colores básicos R, G y B. Los coeficientes empleados se deben a que la sensibilidad del ojo a los diferentes colores no es uniforme (mayor para el verde, menor para el azul). De todos modos, los valores utilizados pueden cambiarse para resaltar colores. Por ejemplo, si subimos el coeficiente correspondiente al azul (bajando los otros para que sumen 1) los puntos con más azul aparecerán más luminosos en la imagen en blanco y negro. El efecto es el mismo que el que consigue un fotógrafo utilizando filtros de color al fotografiar en blanco y negro.

HISTOGRAMA Y REALCE DE IMÁGENES

Se denomina **histograma** de una imagen monocroma a un gráfico que muestra la cantidad de puntos que existen en la imagen para cada nivel de gris. En MATLAB se obtiene mediante la función **`histogram`**. Por ejemplo, para la imagen `X1bn` del apartado anterior, lo podemos calcular y visualizar haciendo lo siguiente:

```
H1 = histogram(X1bn(:),0:255);  % dibujar el histograma
H2 = histcounts(X1bn(:),0:256); % almacenar en un vector los valores
```

`X1bn(:)` es la matriz `X1bn` convertida en un vector para poder utilizarla con la función `histogram`.

El histograma nos da, de un vistazo, una idea de la distribución de brillos en la imagen, y es muy útil cuando se quiere realzar la imagen mejorando el contraste entre puntos. Como ejemplo, haz lo siguiente con la imagen **`camion.bmp`**:

- Cárgala en las variables **`camion`** y **`paleta`** y visualízala. Es una imagen monocroma, pero está almacenada con paleta de colores. Además, los niveles de gris de la paleta están desordenados.

```
[camion, paleta]= imread('camion.bmp');
```

- Para obtener una matriz con el nivel de gris de cada punto:

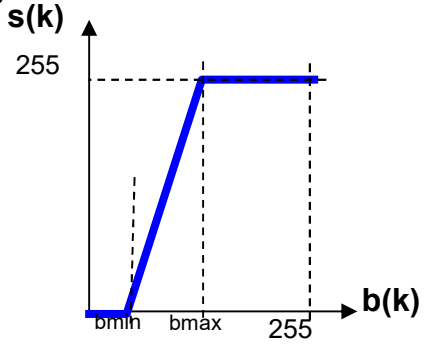
```
camion = camion + 1;
Rpal = paleta(:,1);      %paleta contiene 3 columnas iguales: niveles de gris
camion = uint8(Rpal(camion)*255); %imagen monocroma, niveles de gris entre 0 y 255.
```

- Calcula el histograma y dibújalo.

Modificación de niveles de brillo:

a) Método 1:

Habrás observado que la imagen obtenida tiene poco contraste, debido a las condiciones de luz con que ha sido tomada. En el histograma se aprecia que la mayoría de los puntos tienen un brillo comprendido entre 40 y 100 (aproximadamente), los puntos con brillo superior corresponden al cielo. A la vista del histograma, se nos puede ocurrir cómo cambiar los niveles de gris de los puntos de la imagen para que se aprecien mejor los detalles de la zona oscura. Por ejemplo, podemos cambiar los valores de brillo de todos los puntos de la imagen según una función de transformación como la de la figura:



b(k): brillo original (valores entre 0-255)
s(k): función de transformación para obtener nuevo brillo

Esto es, si los puntos que queremos realzar son los comprendidos entre *bmin* y *bmax*, se procede de la siguiente forma:

- a los puntos de brillo menor que *bmin* se les asigna brillo 0 (negro),
- a los puntos de brillo mayor que *bmax* se les asigna brillo 255 (blanco),
- a los puntos del intervalo [*bmin*, *bmax*] se les aplica una transformación lineal: multiplicar por factor de intensificación *fint*= 255 / (*bmax-bmin*).

En el caso particular de la imagen 'camion' se desea realzar los puntos de brillo comprendido entre 40 (*bmin*) y 125 (*bmax*, en vez de 100 para simplificar el cálculo). Para ello, en MATLAB, basta ejecutar las siguientes instrucciones:

```
Ftrans = [zeros(1,bmin)  (1:(bmax-bmin))*fint    255*ones(1,(256-bmax))];

camion2 = uint8(Ftrans(camion + 1));
```

Así conseguimos que los puntos de brillo menor que 40 (*bmin*), pasen a tener brillo 0 (negro) pero como hay pocos no importa. Los puntos con brillo mayor a 125 (*bmax*, cielo) pasan a tener brillo 255 (blanco) pero en este caso no nos importa perder detalle en esa zona. Los intermedios (de brillo entre 40 y 125), se distribuyen linealmente en un rango más amplio que antes, con lo que al visualizarlos se diferenciarán mejor (factor de intensificación *fint* = 3).

Visualiza la nueva imagen y verás que esta simple transformación lineal mejora sensiblemente la imagen.

b) Método 2:

La transformación de brillos no tiene por qué ser lineal, y puede utilizarse cualquier curva para obtener mejores resultados. En algunos casos, se busca igualar al máximo el histograma.

A continuación, puedes ver una forma sencilla de obtener una función de transformación que reparte los valores de brillo tratando de que el histograma sea lo más plano posible:

$$Ftrans(i) = \frac{255}{N}\sum_{k=0}^{i} h(k)$$

siendo N el nº de puntos de la imagen, y $h(k)$ el valor k del histograma, es decir el nº de veces que aparece el brillo k en la imagen original.

A veces igualar el histograma de esta manera, produce imágenes de aspecto poco real, pero permite distinguir detalles ocultos.

c) Aplicación a un caso particular:

Como aplicación de todo lo dicho, escribe una función que implemente estos dos métodos para realzar imágenes. Como punto de partida dispones del fichero **`ajuhist.m`** con la estructura y la explicación de los parámetros a utilizar.

Una vez escrita pruébala con las imágenes **`camion.bmp`**, **`cameraman.tif`**, **`nina.tif`**, **`liftingbody.png`** y **`monedas.tif`**. Además, añade una imagen, la que tú quieras, que te parezca interesante para procesarla mediante esta técnica.

P3 1 ajuhist.m

FILTRADO DE IMÁGENES

Dentro del Image Processing Toolbox podemos encontrar algoritmos para el procesamiento de imágenes (2D y 3D) que nos permiten, entre otras cosas, segmentar imágenes, reducir ruido, transformaciones geométricas, generar histogramas, etc.

Dentro de los algoritmos de filtrado y mejora de imágenes, a modo de ejemplo, vamos a experimentar con los filtros de detección de bordes. Estos filtros se pueden utilizar para múltiples aplicaciones: reconocimiento de objetos, determinación de fallos en cadenas productivas, detección de anomalías, etc.

Antes de ver algunas de las posibilidades que nos proporciona Matlab, vamos a implementar un filtro de detección de bordes muy utilizado denominado sobel y lo vamos a aplicar a una imagen ejemplo (`kursaal1.jpg`). Para ello, solo tenemos que definir una máscara con unos valores determinados y realizar una operación de correlación de la imagen con dicha máscara. En concreto, los valores de la máscara (3x3) para el caso de un filtro sobel son los siguientes:

```
[I, pal] = imread('fotos\kursaal1.jpg');
I=rgb2gray(I);                          % transformamos en tonos de grises
SOBEL = [ 1  2  1;   0  0  0 ;  -1 -2 -1 ];  % mascara filtro sobel
IH=imfilter(I,SOBEL);                   % bordes horizontales (H)
IV=imfilter(I,SOBEL');                  % bordes verticales (V)
IS=imadd(IH,IV);                        % bordes sobel (H+V)
figure,
subplot(2,2,1),subimage(I),title('Imagen Original');
subplot(2,2,2),subimage(IH),title('Sobel Horizontal');
subplot(2,2,3),subimage(IV),title('Sobel Vertical');
subplot(2,2,4),subimage(IS),title('Sobel');
```

De este modo, podríamos definir diferentes filtros partiendo de matrices con diferentes valores. Por ejemplo, si la matriz tuviera los siguientes valores: `[ 1  1  1;   0  0  0 ;  -1 -1 -1 ]` ; estaríamos definiendo un filtro de tipo prewitt (también sirve para la detección de bordes). Con otros valores, podéis probar, se pueden generar filtros que sirvan para otro tipo de operaciones sobre la imagen: erosionado, relleno, etc.

Matlab nos proporciona, ya implementados, los filtros más utilizados en el procesamiento de imagen. Para ello, podemos utilizar la función `fspecial` que permite construir filtros como los que hemos mencionado (‘`sobel`’, ‘`prewitt`’) y otros filtros que sirven para, por ejemplo, simular el movimiento lineal de una cámara al extraer una fotografía (‘`motion`’). Tras definir la máscara con la que queremos procesar la imagen, utilizaremos la función imfilter para realizar la operación de correlación (por defecto) o de convolución (‘`conv`’ como tercer parámetro) entre la máscara y la imagen.

Por ejemplo, si queremos generar un filtro de tipo sobel como el que hemos implementado anteriormente utilizando las funciones de Matlab, podríamos hacerlo de la siguiente manera:

```
[I, pal] = imread('fotos\kursaal1.jpg');
I=rgb2gray(I);                          % transformamos en tonos de grises
SOBEL=double(fspecial('sobel'));
IH=imfilter(I,SOBEL);                   % bordes horizontales (H)
IV=imfilter(I,SOBEL');                  % bordes verticales (V)
IS=imadd(IH,IV);                        % bordes Sobel (H+V)
figure,
subplot(2,2,1),subimage(I),title('Imagen Original');
subplot(2,2,2),subimage(IH),title('Sobel Horizontal');
subplot(2,2,3),subimage(IV),title('Sobel Vertical');
subplot(2,2,4),subimage(IS),title('Sobel');
```

A modo de ejemplo, aplica este tipo de filtros a algunas imágenes ejemplo que hemos incluido en el material del laboratorio.

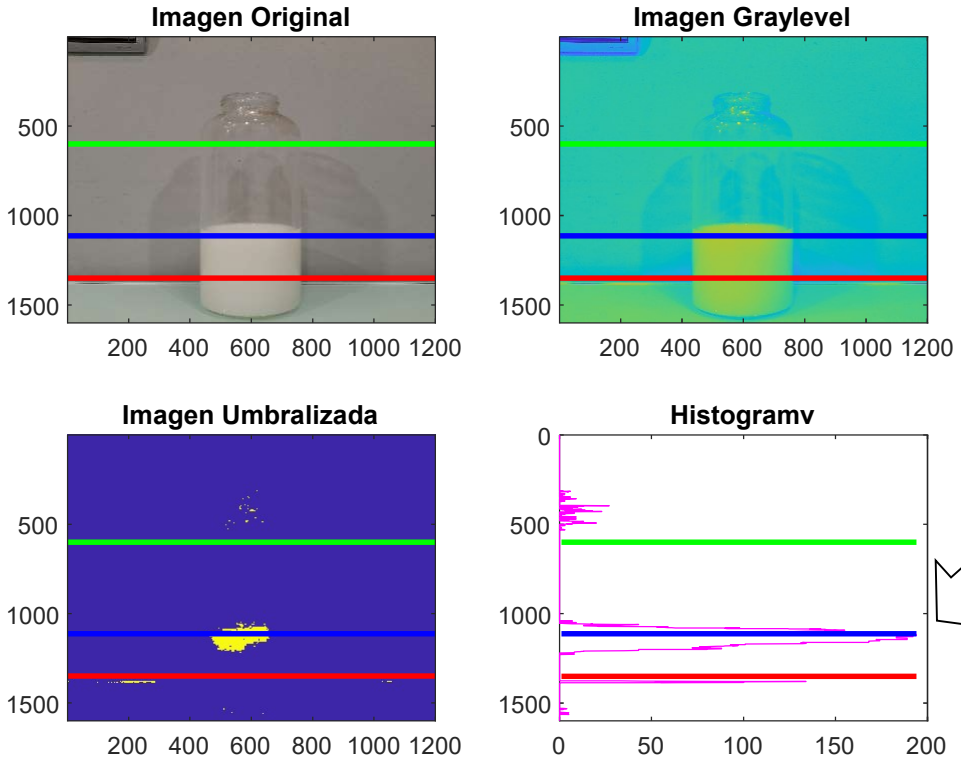
Por un lado, tienes 3 ejemplos relativos a imágenes de **carreteras** con visibilidad reducida, por ejemplo, como ayuda a un sistema de conducción autónoma o asistida: carretera00-01-02.jpg. Aplícalo e incluye las figuras correspondientes en el informe técnico.

Por otro lado, vamos a aplicar este filtro en un contexto de **control de llenado de botellas** en una cadena productiva. En este caso vamos a umbralizar la imagen para binarizarla y utilizar el histograma horizontal para detectar si el nivel de llenado de la botella se encuentra entre dos límites predefinidos o no. En el fichero BotellaLeche.m, tienes el código aplicado al ejemplo de una botella con leche (`bot_leche.jpeg`). A continuación, la salida final de dicho programa.

Se trata que apliques esta técnica en otra empresa que se encarga de llenar botellas con bebida de cola (`bot11-14.jpeg`). Tendrás que hacer ciertas modificaciones al código utilizado para la botella de leche. Para este caso, utiliza los siguientes límites: 700 para high y 1000 para low.

Seguro que se puede hacer mejor, así que podéis probar otras alternativas, además de la propuesta. Incluso podéis incluir en la entrega la aplicación de este tipo de técnicas sobre **otros ejemplos similares** o **en otro contexto** que se os ocurra.

P3_2_filtrado.m



Linea azul: valor máximo del histograma dentro de los límites rojo (low) y verde (high)