Práctica 1: Imágenes en Matlab			
Apellidos, nombre	**Iván Domínguez**	**Fecha**	
Apellidos, nombre	**Lucas Miranda**		

El objetivo de esta práctica es presentar al alumno las herramientas que ofrece MATLAB para la representación y manejo de imágenes.

Desarrolle cada ejercicio en un fichero de comandos 'P1_ejercicioX.m' separado y anote las observaciones que se le pidan como comentarios. Para conocer el funcionamiento preciso de los comandos que se introducen en este guion, utilice la ayuda de MATLAB. Para evitar posibles interferencias con otras variables o ventanas recuerde incluir siempre las instrucciones

```
clear all;
close all;
clc;
```

al principio de cada fichero de comandos.

Al finalizar la práctica, comprima el documento con las observaciones y los ficheros '.m' generados en un único fichero con el nombre 'TSV_P1_ApellidosNombre.zip', conéctese al sistema de entrega de prácticas de Moodle y entréguelo.

1. Lectura, visualización, transformación y grabación de imágenes

1.1. Ejercicio 1: lectura de imágenes a partir de un archivo.

Para leer una imagen de un archivo utilice la función **imread()**:

```
[ima, map]=imread(<filename>);
```

Si la imagen almacenada es una **imagen indexada**, ***ima*** será una **matriz (*array*) de dos dimensiones** (correspondiente a las **filas x columnas**) de tipo **uint8** o **uint16**, y ***map*** será su tabla de colores (VLT-*Video Lookup Table*), una **matriz** de **tres columnas** cuyas **filas** (igual al **número de colores** de la imagen) indican las tres componentes R, G, B del color que representan (si las tres componentes fueran iguales, representan un nivel de gris).

Si la imagen grabada no es indexada, se trata de una **imagen true-color**, es decir, una imagen en la que el valor de cada píxel indica directamente su color; no es un índice de una VLT por lo que en estas imágenes ***map* está vacío**. En una **imagen true-color, *ima*** será **una matriz de dos dimensiones o tres matrices de dos dimensiones** (un tensor), correspondientes a una o tres bandas (canales), de tipo **uint8**. Si se trata de una matriz (array de dos dimensiones) de dos dimensiones (es decir, que a cada elemento de la imagen le corresponde un valor), el valor de los píxeles es directamente un nivel de luminancia o nivel de gris que va de 0 (negro) a 255 (blanco); por lo tanto, no incluye información de color (estas imágenes podemos denominarlas ***true-bw***). Si se trata de tres matrices de dos dimensiones (array de tres dimensiones), a cada elemento de la imagen le corresponden tres valores enteros que indican directamente las componentes roja, verde y azul de cada píxel, cada una de ellas variable entre 0 (componente inexistente) y 255 (máxima saturación en esa componente). Dentro de la categoría true-color, las imágenes binarias son un tipo especial que solamente presenta dos valores; estas imágenes están formadas por una sola banda (es decir, un

matriz/array de dos dimensiones) de tipo **uint8** (con valores 0 o 255), o **logical** (con valores 0 o 1).

Lea las siguientes imágenes de la librería de MatLab

```
[ima1, map1] = imread("peppers.png");
whos ima1 map1

[ima2, map2] = imread("cameraman.tif");
whos ima2 map2

[ima3, map3] = imread("corn.tif");
whos ima3 map3
```

Comente de qué tipo es cada imagen y su tamaño.

La imagen 3 es una imagen indexada pues es la única con una tabla de colores disponible en map. Tiene tamaño 415x312 de tipo unit8 y tres canales, en el map, cada uno de 256 bits y tipo double.

La imagen 1, por su parte, es una imagen true-color, pues tiene una tercera matriz especificando los canales. Sus dimensiones son 384x512x3 de tipo unit8.

La imagen 2 también es una true-color, una true-bw concretamente, pues no dispone de una matriz para los canales. Sus dimensiones son 256x256 de tipo unit8.

1.2. Ejercicio 2: visualización de imágenes

Partiendo el ejercicio 1, visualice en una figura las diferentes imágenes haciendo uso de los comandos **figure**, **subplot** e **imshow.**

```
figure;
subplot(3,1,1);
imshow(ima, [min_ima max_ima]);
```

Para representar una matriz de valores en forma de imagen se utiliza la función **imshow**, indicando la función a representar y, para imágenes ***true-bw***, (para *true-color* se puede poner rango, pero no afecta; para *indexadas* no se puede poner rango) los valores de la función a que corresponden el negro (habitualmente el mínimo de la función) y el blanco (habitualmente el máximo de la función).

Los valores de min y max se puede fijar según se quiera realizar la visualización. Para mostrar el rango completo, se debe calcular el mínimo y máximo de la imagen;

```
min_ima=min(ima(:))
max_ima= max(ima(:))
```

en otros casos se fija al rango posible (e.g., [0 255]). Si no se utiliza este rango ajustado a la imagen, los valores de la imagen inferiores a **min** se presentarán como negros y los superiores a **max** se presentarán como blancos, por lo que se perderá resolución en la representación de los tonos de gris.

*Nota: El parámetro de **imshow** 'InitialMagnification' fijado al valor 100 fuerza a que la imagen se presente a tamaño real (cada píxel de la imagen corresponde con un punto del monitor), siempre que el tamaño del monitor lo permita. Este es el único modo de garantizar que la imagen se presenta con la máxima precisión.*

Para representar varias imágenes en una misma ventana utilice el comando **subplot** en combinación con el comando **imshow**. Observe que en este caso no es posible controlar que la imagen se presente a tamaño real (por lo que el parámetro '*InitialMagnification*' no es de utilidad).

Para cada una de las imágenes represente en una figura la imagen original, la imagen original con el rango especificado máximo y mínimo, y con un rango comprimido (e.g., min_ima*2 y max_ima/ 2).

Comente por qué motivos las imágenes de cada figura se ven iguales o distintas, para ello quizás le resulte de ayuda el incluir el comando **colorbar** tras la representación de cada imagen.

La primera imagen se ve prácticamente idéntica. Podemos notar que al “achatar” el espectro de colores, lo que obtenemos es un espectro más discreto. Esto quiere decir que entre cada par de colores, puede haber algún color intermedio faltante. Pero vemos que esto apenas es perceptible por el ojo humano.

La segunda imagen, por su parte, si que parece cambiar bastante. Si bien al tener tres canales, como antes, un cambio en la granularidad de los colores apenas es perceptible, sí que lo es cuando solo tenemos un canal en blanco y negro. Algunos grises pasan a ser más claros y otros más oscuros, cambiando bastante el significado de la imagen a simple vista.

En cuanto a la tercera, ocurre un poco lo mismo que con la segunda. Es claro como algunos grises pasan a ser más claros. Y otros más oscuros, resultando en una pérdida de información en cuanto al color.

1.3. Ejercicio 3: transformación de clase

Partiendo el ejercicio 2, genere para cada una de las tres imágenes una versión indexada, una true-color (3 canales) y una true-bw (1 canal), visualice las tres versiones de cada imagen en una figura diferente.

Para ello, utilice las funciones **gray2ind()**, **rgb2ind()**, **ind2rgb()** e **ind2gray()**. Para convertir una imagen en color en otra con niveles de gris se recomienda utilizar **rgb2gray()**; deduzca de la ayuda de esta función cómo convertir las imágenes dadas.

Para convertir una imagen ***true-bw*** a una imagen ***true-color*** (seguirá siendo "blanco y negro", pero compatible con las true-color para operar), se debe generar una imagen ***true-color*** (tres canales) y poner los mismos valores en los mismos, esto es, copiar la imagen ***true-bw*** en cada una de las bandas.

```
ima_rgb (:,:,1)= ima_gray;
ima_rgb (:,:,2)= ima_gray;
ima_rgb (:,:,3)= ima_gray;
```

Genere versiones de las imágenes indexadas con 256, 128 y 64 niveles. Para ello, parta de la imagen ***true-color*** o ***true-bw*** y haga uso de **gray2ind()** o **rgb2ind()** con el argumento ***c***.

Comente las similitudes y diferencias entre las diferentes versiones.

Cuanto mayor es la granularidad, mayor número de colores podemos observar en la imagen 1. Esto es lógico, puesto que al pasar al formato indexado lo normal será hacerlo perdiendo colores. Cuando lo pasamos a true-bw desde true-color, el resultado es el esperado. Sin embargo, al hacerlo desde su version indexada, en la que habíamos perdido color, la diferencia entre otra versión bw y su versión actual no es tan grande como en true-color, puesto que en realidad estamos perdiendo 3 veces menos colores (salvo que en otra escala).

Cuando en lugar de 64 niveles, le pasamos 128, la diferencia es más que mínima. Todavía se puede apreciar en el true-color, pero apenas en el true-bw.

En el caso de poner 256 niveles, se sigue apreciando una ligera pérdida de calidad, pero realmente la imagen apenas pierde información perceptible por el ojo humano como tal en el true-color. Por su parte, en el true-bw soy incapaz de notar la diferencia.

1.4. Ejercicio 4: grabación de imágenes a un archivo.

MatLab permite trabajar y visualizar imágenes de tipo **double**, con valores tanto positivos como negativos. Sin embargo, los formatos de almacenamiento de imágenes más extendidos exigen que éstas se encuentren indexadas (de tipo **uint8** o **uint16**) o que sean true-color con componentes de tipo **uint8**. De ahí que para almacenar o intercambiar imágenes sea recomendable que tengan este tipo antes de grabarlas, junto con su mapa de colores si son indexadas.

El objetivo de este ejercicio es grabar las imágenes obtenidas en el ejercicio 3. Cuando la imagen sea de tipo **double**, utilice primero la función **im2uint8()** para convertirla a una imagen true-color con componentes de tipo **uint8** (este paso no es estrictamente necesario, ya que la función **imwrite** lo realiza automáticamente para imágenes de tipo **double**). Recuerde que esta función (y, por lo tanto, la función **imwrite**) sólo arroja el resultado esperado si la imagen de tipo **double** toma valores en el rango [0,1]; si no, debe realizar la conversión usted mismo, mediante la función **rescale()** o siguiendo el procedimiento que se verá en otras prácticas.

Utilice la función **imwrite()** parta almacenarlas en ficheros con formato TIFF. Luego intente visualizar la imagen grabada con un programa cualquiera (que no sea MatLab). Si no lo consigue, es porque el ajuste de rango de la imagen obtenida no es correcto.

Por último, utilice la función **imfinfo(filename)** para verificar las propiedades de las imágenes guardadas y compárelas con las originales. Utilice la ayuda de MATLAB para obtener más información acerca de esta función.

2. Operaciones con imágenes

2.1. Ejercicio 5: operaciones básicas con imágenes.

Este ejercicio propone realizar las operaciones necesarias para calcular la diferencia (restar) a cada imagen original las versiones generadas anteriormente.

Para poder operar en MatLab con una sola imagen (ponderarla, escalarla, etc.) sin perder rango de representación en el resultado ésta siempre ha de ser de tipo **double** y siempre ha de ser true-color. Si la imagen es indexada no suele tener sentido operar con ella, ya que las variaciones de valor de sus píxeles suponen cambios de índices en una tabla, operación cuyo resultado es en general incierto. Si lo que se desea es operar con dos imágenes (sumarlas, restarlas, etc.), ambas deben ser de tipo **double** y de tipo true-color, y ambas deben tener las mismas dimensiones (filas, columnas y número de bandas o valores por píxel).

Las conversiones necesarias para conseguir esta situación se resumen en:

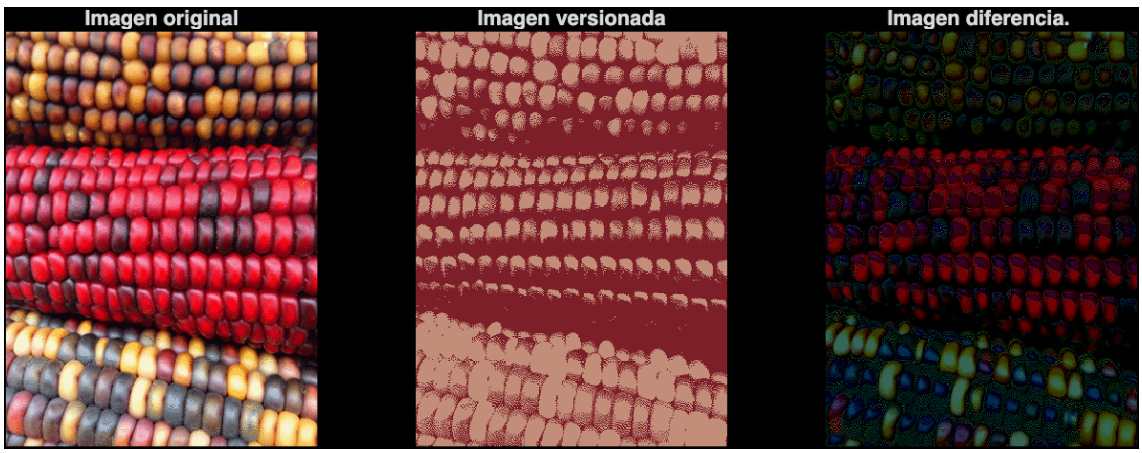
- Para convertir una imagen indexada en una imagen *true-color* utilice la función `ind2rgb(ima)`. Esta función devuelve una imagen *true-color* que además es de tipo `double` y definida en el rango $[0,1]$.
- Para convertir una imagen (indexada o *true-color*) de tipo `uint8` o `uint16` a una de tipo `double` utilice `im2double(ima)`. Esta función devuelve una imagen *double* definida en el rango $[0,1]$, y de la misma clase (indexada o *true-color*) que la original.
- Para convertir una imagen *true-color* de tres componentes (rojo, verde, azul) en una imagen *true-color* de una componente (nivel de gris), utilice `rgb2gray(ima)`. En este caso, se perderá la información de color.
- Ver Ejercicio 3 para convertir una imagen *true-color* de una componente (nivel de gris) en una imagen *true-color* de tres componentes (rojo, verde, azul), ya que MatLab no aporta una función específica para esa transformación.

Una vez generadas las imágenes diferencias, visualícelas en una figura para cada imagen original que tenga a la izquierda la original, en el centro la versión y a la derecha la imagen diferencia.

Calcule la energía de cada imagen mediante la siguiente función

```
function E = calcularEnergia(ima)
ima=double(ima);
E = sum(ima(:).^2);
end
```

El resultado de las 3 imágenes sería el siguiente:



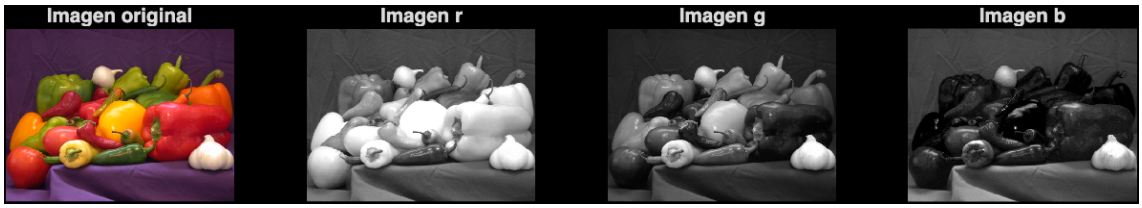
Las energías de cada imagen diferencia sería: 1.5129e+04, 3.0684e+03, 1.1691e+04 respectivamente.

2.2. Ejercicio 6: Planos RGB

Este ejercicio propone intercambiar los planos de color (RGB) de una imagen para entender la percepción del color del SVH.

Lea la imagen **peppers** y genere 3 imágenes "true-bw" para cada uno de los planos de color. Represente en una figura con cuatro columnas la imagen original y cada uno de los planos representados en escala de grises. Añada una segunda fila, donde cada banda se represente en su "color primario".

Finalmente, genere tres imágenes con las bandas intercambiadas, esto es: BRG, RBG y BGR. Represente una figura con la primera fila con la imagen original y las tres bandas RGB en "color primario", y las otras tres con las imágenes con las bandas intercambiadas.



2.3. Ejercicio 7: Procesamiento a nivel de bloque

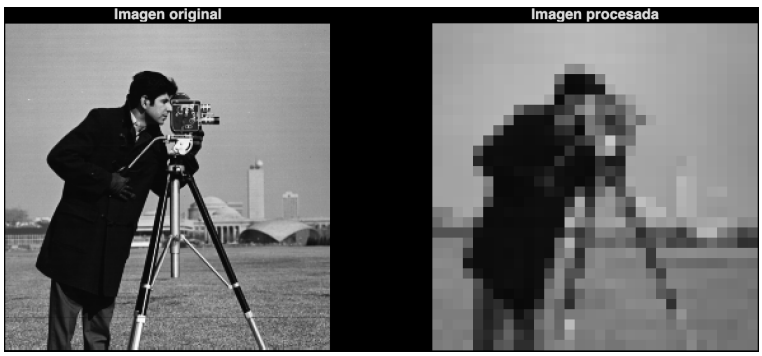
En el ejercicio 6, se ha descrito cómo realizar operaciones para modificar imágenes a nivel de píxel. En cambio, en algunos tipos de operaciones es conveniente dividir la imagen en bloques y aplicar una función a cada bloque de manera individual. Para ello podemos utilizar la función **blkproc()**

```
[ima_procesada]= blkproc(ima,[m n],fun)
```

Donde **m** y **n** indican las dimensiones del bloque a procesar y **fun** es un enlace a una función que acepta como entrada una matriz **x** (de dimensiones **m** y **n**) y devuelve una matriz, vector o escalar **y**, donde: **y = fun(x)**. Los enlaces a funciones se definen mediante el uso del comando **'@'.** Por ejemplo, para la función **mean()** sería**:**

```
func = @(x) mean(mean(x));
```

Para ilustrar el uso de esta función, lea la imagen **cameraman** y utilice la función **blkproc()** para calcular el valor medio de bloques de tamaño 8x8 (se recomienda consultar la ayuda MATLAB). Represente la imagen original y la procesada en una figura y comente su relación con una operación de escalado de una imagen.



En realidad esta operación no hace más que transformar la imagen en una de tamaño 8 veces más pequeña. Coge bloques de píxeles 8x8 y los recalcula como su promedio, consiguiendo transformar tal bloque en un solo píxel.

2.4. Recomendación final

Consulte la ayuda de Matlab para conocer y aplicar los diferentes parámetros de las funciones **figure** e **imshow** de cara a generar figuras con el máximo de información.

Por ejemplo:

```
figure('Name', 'Umbraliza Global');
subplot(2,3,1),
imshow(ima_original), colorbar,
title(['imagen original', sprintf('E=%g',calcularEnergia(ima_original))]);
```