# PRÁCTICA 6:

# PROCESAMIENTO DIGITAL DE IMÁGENES (PYTHON)

## 1. Representación y formatos de imagen

Las imágenes son señales con dos variables independientes (espaciales) a diferencia de las señales monodimensionales con una sóla variable independiente (temporal) que hemos manejado hasta ahora.

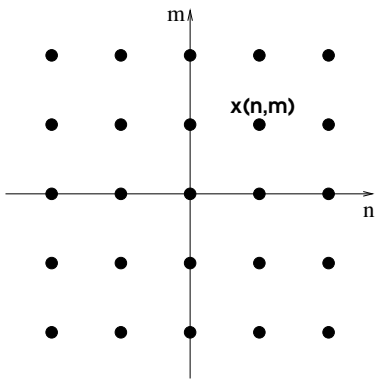


Figura 1: *Representación de una señal bidimensional.*

En la figura 1 se representa una señal bidimensional $x(n,m)$. Como podemos ver, se trata de una matriz en la que cada elemento (o *pixel*) queda posicionado mediante las variables espaciales $n$ y $m$, siendo $x(n,m)$ un valor de intensidad luminosa. El caso más simple corresponde a las imágenes en escala de grises en las que el valor del pixel representa un nivel de gris (desde un valor mínimo/negro al máximo/blanco) en un rango discreto o continuo. Rangos típicos: 0-negro a 1-blanco en imágenes tipo float, o 0-negro a 255-blanco en imágenes tipo entero 8-bits.
NOTA: Obsérvese que al representar la imagen como una matriz, los ejes $n$ y $m$ aparecen rotados $90^o$ en el sentido de las agujas del reloj.

Las imágenes en color se obtienen mediante la superposición de 3 imágenes que representan los niveles de intensidad de rojo (matriz R), verde (matriz G) y azul (matriz B) de cada pixel. Este tipo de representacion de imágenes en color se denomina *RGB*. Alternativamente, pueden usarse otras representaciones del color. La más extendida es la que usa una luminancia (componente de intensidad) y dos crominancias (componentes de color), aunque existen diversas formas de definirlas.

### 1.1. Manejo de formatos en Python

Para la manipulación de los distintos formatos de imagen se requerirá tener instalado el módulo *scikit-image* en nuestro entorno de Anaconda, e importaremos los siguientes módulos:

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, color, util
import scipy.ndimage as ndi
from scipy import signal
```

Para almacenar y transmitir imágenes existen una serie de formatos estándar. El formato más sencillo, que no aplica ningún tipo de compresión, es el Bitmap (.bmp). Por ejemplo, podemos cargar y visualizar la siguiente imagen:

```
I=io.imread('lena.bmp')
plt.imshow(I)
```

donde $I$ es un tensor conteniendo 3 imágenes (componentes R, G y B). También podemos convertir a escala de grises directamente durante la lectura mediante:

```
I=io.imread('lena.bmp',as_grey=True)
plt.imshow(I)
```

de forma que $I$ es ahora una matriz simple. En las realizaciones prácticas que siguen nos restringiremos a imágenes en escala de grises.

Tambien es posible leer y escribir imágenes en otros formatos estándar como Graphics Interchange Format (.gif), Joint Photographic Experts Group (.jpeg), Portable Network Graphics (.png), o Tagged Image File Format (.tiff). El comando `imread` detecta automáticamente el tipo de fichero según la extensión del mismo.

## 2. Realce de Imágenes: Equalización de Histogramas

Mediante el realce de imágenes se pretende resaltar o clarificar determinados aspectos de una imagen. Una de las técnicas más extendidas para el realce de imágenes es la de *ecualización de histogramas* (HEQ), cuyo objetivo es básicamente aumentar el contraste de la imagen. Para ello, la técnica HEQ trata de redistribuir las intensidades de los pixels de forma que el histograma cambie a otro predeterminado, usualmente plano.

El histograma de una imágen es un gráfico en el que se muestra la distribución de la intensidad luminosa de una imagen. El histograma se obtiene dividiendo el intervalo de intensidades $[0,255]$ en un determinado número de subintervalos `Nbins`. Para cada subintervalo se representa el número de pixels cuya intensidad cae dentro del mismo. Como ejemplo de cómputo de histograma, podemos usar el siguiente código sobre una imagen PCM de 8 bits/pixel:

```
I=io.imread('tire.tif')           # Lectura de imagen tiff
plt.figure(1)                     # Mostrar imagen original
plt.imshow(I,cmap='gray')
Nbins=256                         # Número de subintervalos
bins=np.arange(-0.5,Nbins)        # Definir limites de los subintervalos
Hist,bins=np.histogram(I,bins)    # Cómputo del histograma
```

```
centers = 0.5*(bins[1:]+bins[:-1]) # Calcular centros de los histogramas
plt.figure(2)                      # Dibujar histograma original
plt.stem(centers,Hist)
plt.title('Histograma Original')
```

Ya que la imagen original está cuantizada a 8 bits, utilizamos los 256 niveles de gris para definir los subintervalos para el histograma.

**Ejercicios**:

1. Obtener y representar la transformación puntual de la imagen anterior que ecualiza su histograma a otro plano. Explicar el resultado. NOTA: se considerarán todos los niveles de gris posibles ($Nbins = 256$), aunque sería posible fijar un número inferior.
2. Aplicar la transformación obtenida a la imagen. Comparar la imagen e histograma resultante con los originales y comentar el resultado.

## 3. Transformada de Fourier 2D

También se puede definir la Tranformada de Fourier (FT) para señales bidimensionales. La forma más eficiente de implementarla es aplicándo una transformada 1D a las filas de la imagen seguida de otra transformada 1D a las columnas (o viceversa). En Python esto puede realizarse aplicando dos veces el comando de FFT monodimensional `fft` de numpy, o directamente con el comando `fft2`. La FT transforma los 2 ejes espaciales en dos ejes de frecuencia (horizontal $\omega_1$ y vertical $\omega_2$). Podemos observar el resultado de aplicar la FT a una imagen mediante el siguiente programa:

```
# Transformada de Fourier y espectro
# Cargar imagen y convertir a niveles de gris
I=io.imread('mandrill.tif',as_grey=True)
plt.figure(1)
plt.subplot(1,2,1), plt.imshow(I,cmap='gray')
# Se selecciona una subimagen con una textura clara
B=I[130:130+100,150:150+200]
plt.subplot(1,2,2), plt.imshow(B,cmap='gray')
N,M=B.shape
imsize=N*M
plt.figure(2)
plt.imshow(B,cmap='gray')
# Computar FFT2
F=np.fft.fft2(B)
# Usar escala logaritmica para correcta visualizacion
plt.imshow(10*np.log10(np.abs(F)**2/imsize),cmap='hot')
plt.colorbar()
# Cuadrantes reordenados (frec. (0,0) en el centro)
F=np.fft.fftshift(F)
plt.figure(3)
plt.imshow(10*np.log10(np.abs(F)**2/imsize),cmap='hot')
plt.colorbar()
```

**Ejercicio**: explicar el código y comentar los resultados del programa anterior.

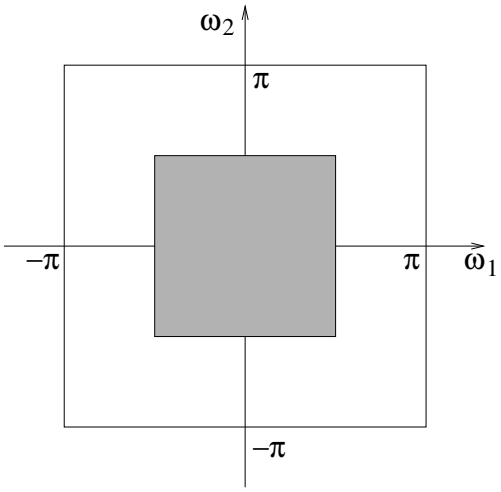


Figura 2: *Respuesta en frecuencia de un filtro 2D pasabaja cuadrado.*

# 4. Restauración de Imágenes

A diferencia del realce, el objetivo de la *restauración* de imágenes es obtener a partir de una imagen degradada por algun tipo de distorsión otra imagen que represente lo más fielmente posible a la original. En esta práctica, generaremos la señal degradada como:

```
I=io.imread('lena.bmp',as_grey=True) # Cargar imagen
J=util.random_noise(I, mode='s&p')   # Contaminar con ruido sal y pimienta
```

donde el segundo argumento de `random_noise` representa el tipo de ruido aplicado. En los siguientes apartados veremos dos técnicas de restauración basadas en filtrado lineal y no lineal.

## 4.1. Filtrado Lineal

Una primera técnica de restauración puede ser la aplicación de un **filtrado lineal pasabaja**. Este filtrado suaviza las variaciones bruscas en la señal introducidas por el ruido. En la figura 2 se representa la respuesta en frecuencia de un filtro pasabaja 2D del tipo que se va a implementar. Aunque la respuesta representada tiene una banda pasante cuadrada, es posible utilizar otras formas como circular, elíptica, etc.

Diseñaremos un filtro FIR pasabaja cuadrado, separable, de tamaño $11x11$, y con frecuencia de corte $f_c = 0{,}18$ ($f = 1$ corresponde a la frecuencia de muestreo), mediante el método de muestreo en frecuencia. Definiremos la respuesta en frecuencia (muestreada) deseada de la siguiente forma (usando separabilidad):

```
# Generar respuesta en frecuencia deseada Hd
Lfir=11 # Orden FIR 1D (impar)
fc=0.18 # Frecuencia de corte 1D (f=1 --> frecuencia de muestreo)
Lcut=int(np.round(fc*Lfir)) # bin de frecuencia para fc
lcent=int((Lfir-1)/2) # punto central (freq. cero)
Hd_1D=np.zeros(Lfir)   # filtro Horizontal
Hd_1D[lcent-Lcut:lcent+Lcut+1]=np.ones(2*Lcut+1)
Hd=np.outer(Hd_1D,Hd_1D) # filtro 2D separable cuadrado (filtros H y V identicos)
```

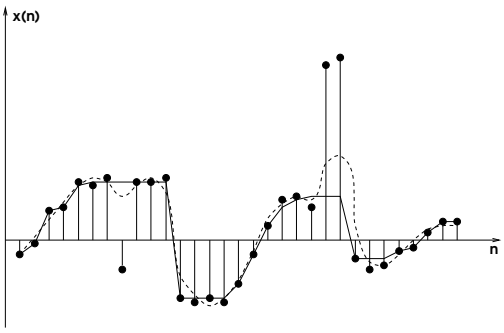


Figura 3: *Efectos del filtrados lineal ($\cdots$) y de mediana ($-$) sobre muestras de una señal monodimensional corrompidas por ruido impulsivo.*

```
plt.figure(1)
plt.imshow(Hd)
plt.colorbar()
plt.title('Respuesta en frecuencia deseada')
```

**Ejercicios**:

1. Obtener la respuesta impulsiva del filtro. Visualizar gráficamente esta respuesta y su correspondiente respuesta en frecuencia. Aspectos importantes a tener en cuenta en el diseño:

   *a*) La transformada de Fourier inversa necesaria para el diseño se computa mediante el comando `ifft2`. Antes de computarla debemos modificar la respuesta en frecuencia deseada de la siguiente manera: 1) reorganizar los cuadrantes de frecuencia mediante `ifftshift`, y b) introducir una fase lineal para que la respuesta impulsiva quede centrada en el punto $(lcent, lcent)$ y no en $(0,0)$. Para ello, se genera una exponencial con argumento $e^{-j\omega l_{cent}}$ (donde $\omega$ toma valores en los puntos de frecuencia considerados para la `ifft2`) que se aplica a las componentes horizontal y vertical. Ambas componentes se combinan mediante el comando de producto `outer`.

   *b*) La transformada de Fourier necesaria para representar la respuesta en frecuencia del filtro diseñado se computa mediante el comando `fft2`. Después de computarla debemos reorganizar los cuadrantes de la transformada mediante `fftshift` para una correcta visualización.

2. Aplicar el filtro diseñado (haciendo uso de la función `convolve` de *scipy.ndimage*) a la señal degradada y comparar (comentar) las imágenes original, degradada y restaurada.

## 4.2. Filtrado No Lineal

El filtrado lineal anterior presenta dos inconvenientes para su uso en restauración de imágenes. El primero es que genera una imagen cuyos contornos aparecen desdibujados debido a la característica pasabaja del filtro. Además, no es capaz de eliminar suficientemente muestras de tipo impulsivo claramente erróneas. Esto podemos observarlo en la figura 3, donde se represen-

tan las muestras de una señal monodimensional contaminada y el resultado de aplicar un filtro lineal pasabaja en línea discontinua.

Un **filtro de mediana** es un filtro no lineal que para cada muestra de señal de entrada $x(n)$ obtiene una salida que es la mediana de las muestras contenidas en un entorno de radio $N$ con centro en la muestra $n$ actual,

$$y(n) = median(x(n-N), x(n+N))$$

La figura 3 muestra en línea continua el resultado de aplicar el filtro de mediana. Se observa que este filtro es capaz de eliminar muestras ruidosas de tipo impulsivo a la vez que mantiene los bordes nítidos, cualidades muy interesantes para imágenes. Para la aplicación de los filtros de mediana a imágenes en necesario considerar entornos bidimensionales de tamaño $N \times M$.

**Ejercicios**: aplicar el filtro de mediana `median_filter` de *scipy.ndimage* de tamaño 3x3 a la señal degradada y comparar con las imágenes original, degradada y restaurada con filtro lineal. ¿Qué ocurre si se aumenta el tamaño del filtro de mediana a 5x5 o 7x7?