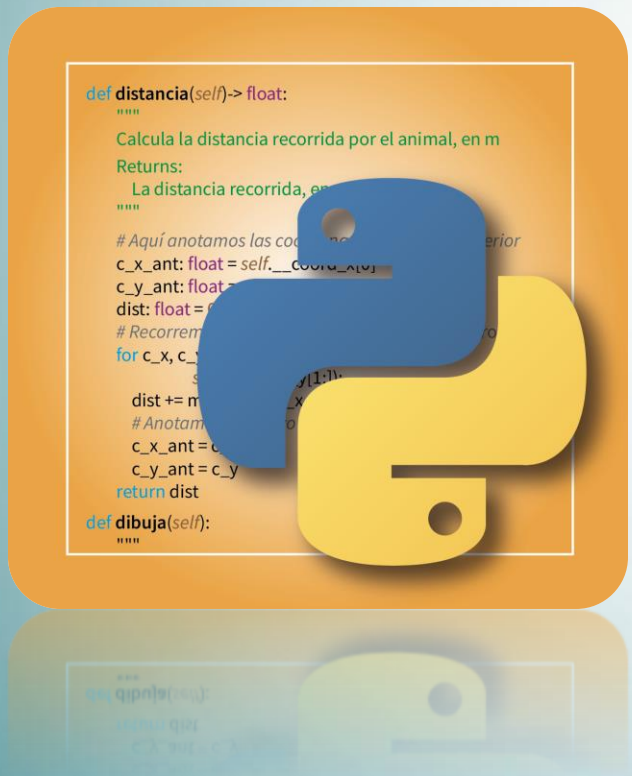
Programación

Práctica 10. Tratamiento de imágenes



Michael González Harbour

José Javier Gutiérrez García

José Carlos Palencia Gutiérrez

José Ignacio Espeso Martínez

Adolfo Garandal Martín

Departamento de Ingeniería Informática y Electrónica

Práctica 10: Tratamiento de imágenes

Objetivos: Utilizar el paquete `numpy` para realizar tratamiento de imágenes

Paso A: sobre una foto en color que elijas tú en formato `.jpg`, se desea hacer las siguientes operaciones:

1. Mostrar en pantalla la foto original
2. Obtener y mostrar la imagen en espejo
3. Obtener y mostrar un fragmento de la imagen en espejo
4. Obtener y mostrar el fragmento del paso anterior rotado 45º
5. Obtener y mostrar la imagen original en blanco y negro

En los apuntes del capítulo 5 (páginas 46 y 47) aparece un ejemplo del tratamiento de una imagen, que se puede usar como guía para realizar los pasos 1 a 3

Pistas para los pasos 2 y 3

Recuerda que cuando se usan rodajas de listas o arrays podemos usar los formatos:

```
arr[ini:fin] # elementos de ini a fin-1
arr[ini:fin:p] # de ini a fin-1 tomados cada p pasos
```

Si omitimos `ini`, vale `0`; si omitimos `fin`, vale la longitud del array

En los arrays bidimensionales de `numpy` se puede poner una tupla de "rodajas", con la primera rodaja para las filas y la segunda para las columnas

```
arr[i1:f1, i2:f2] # fila entre i1 y f1-1 y
                  # columna entre i2 y f2-1
arr[i1:f1:p1, i2:f2:p2] # filas cada p1 pasos y
                        # columnas cada p2 pasos
```

Pistas para los pasos 4 y 5

Buscar en Internet información sobre los siguientes módulos y operaciones:

- El módulo `ndimage` de `scipy` tiene una función `rotate` para rotar una imagen
- El módulo `skimage.color` tiene una función `rgb2gray` para convertir a blanco y negro
- El módulo `skimage.io` tiene operaciones `imshow` y `show` para mostrar imágenes en pantalla
  - usar esta operación para mostrar imágenes en blanco y negro

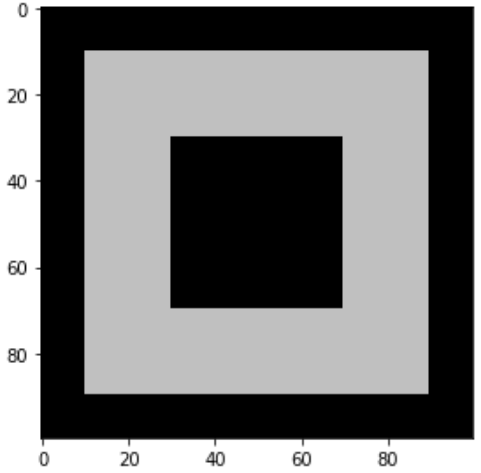
Paso B: Imagen creada por nosotros

Trabajaremos con una imagen creada por nosotros, en forma de array bidimensional de `numpy` de tamaño 100x100

- con todos los valores representando gamas de gris entre cero (negro) y uno (blanco)

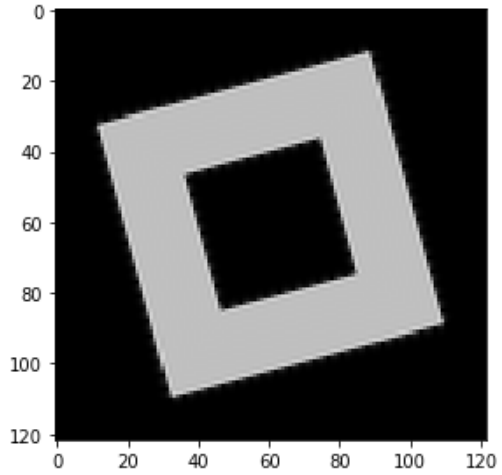
1. Empezamos con un array de todo ceros (números reales)
2. Cambiamos algunos puntos a gris claro, formando un cuadrado con bordes de anchura = 20 píxeles (px)
   - Podemos hacerlo por bandas. Por ejemplo esta instrucción cambia una banda de anchura 20px a color gris:

```
img[10:90, 10:30]=0.75
```

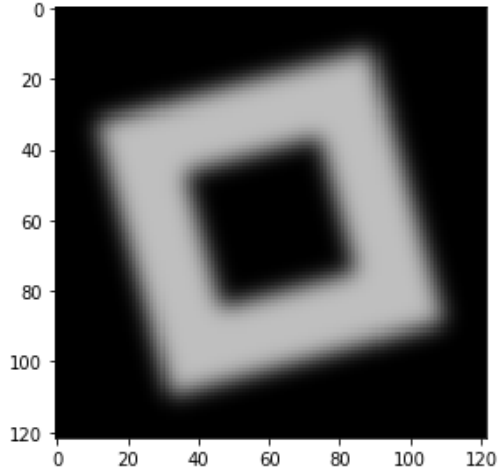


Paso B (cont.)

3. Rotamos la imagen 15 grados
   - La rotación cambia algunos valores a negativo, lo que hace que la imagen se represente en falso color
   - lo solucionaremos calculando el valor absoluto de cada píxel con la función `numpy.absolute`

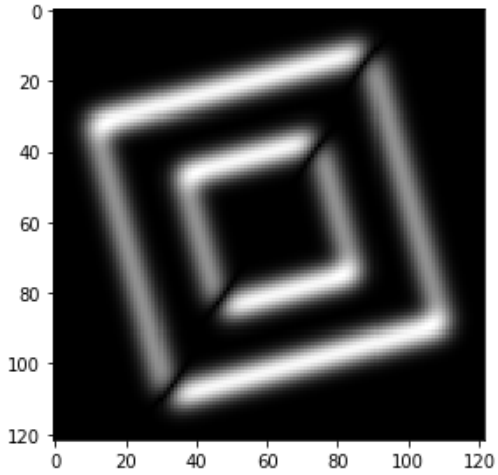


4. Suavizamos la imagen con un filtro gausiano. Usamos la función `gaussian_filter` de `ndimage` con valor de `sigma`=3



Paso B (cont.)

5. Finalmente, hacemos una extracción de bordes con estos pasos:
   - Sobre la imagen suavizada aplicamos un filtro *sobel* al eje `X`, usando la función `sobel` de `ndimage` con un parámetro `axis=0`
   - También sobre la imagen suavizada aplicamos un filtro *sobel* al eje `Y`, con `axis=1`
   - Combinamos los resultados de ambas operaciones como `|0.5*sobel_x + 0.5*sobel_y|`
   - Tras este paso pueden quedar valores mayores que `1` o demasiado bajos. Para solucionarlo dividimos todos los puntos de la imagen por el valor máximo, que se puede obtener con la función `max` de `numpy`



Nota: para manejar los arrays en esta práctica no es necesario ningún bucle

Estructura del código:

Meter la parte A en un módulo llamado `foto.py`, con un programa `main`()

Meter la parte B en un módulo llamado `imagen_creada.py`, con un programa `main`()

Meter la parte avanzada en el mismo módulo `imagen_creada.py`, en una función llamada `main_color`()

Nota: Las anotaciones de tipo para los arrays se pueden hacer usando el tipo `numpy.ndarray` (o `np.ndarray` en su caso)

Parte Avanzada

Hacer un proceso similar al de la parte B, pero trabajando con una imagen en color

Para la imagen en color, en lugar de representar cada píxel con un número real hacerlo con una tupla de tres valores enteros entre 0 y 255, representando cada uno la intensidad de rojo, verde y azul

- Ello implica que el array que almacena la imagen tiene tres dimensiones: la fila, la columna, y el color

Usar colores diferentes para las franjas que representan el cuadrado y el interior del cuadrado

Deben usarse números enteros, no reales

- la función `zeros()` de `numpy` produce números reales
- la función `full()` crea el array con los datos que queramos

Aspectos a tener en cuenta al trabajar con colores

El suavizado no se puede hacer con el filtro gausiano. Usar en su lugar `ndimage.uniform_filter` con `size=(5, 5, 1)`, por ejemplo. El tamaño 1 para la última dimensión del array es muy importante para manejar bien los colores

La detección de bordes solo se puede hacer con una imagen monocolor

- Para ello nos quedaremos con el color verde (índice 1 para la 3ª dimensión)
- Convertiremos la imagen verde, que es de tipo entero con signo, a entero sin signo con `skimage.util.img_as_ubyte`
- Luego convertiremos a tipo float con `skimage.util.img_as_float`
- Finalmente podemos usar el filtro sobel como en la figura en B y N

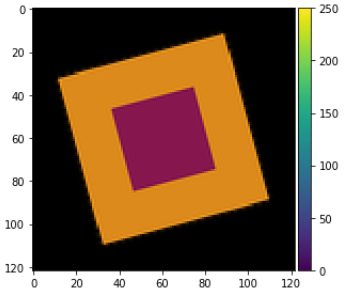
Aspectos a tener en cuenta (cont.)

Para mejorar el contraste el resultado de los bordes se puede elevar al cubo y luego normalizar dividiendo entre el máximo valor

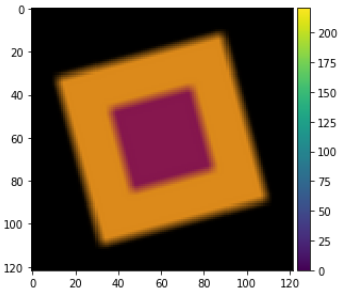
Parte opcional:

- puede resultar vistoso sobreimponer el resultado sobre el color verde de la figura rotada
  - Para ello será necesario reconvertir la figura con los bordes de tipo float a entero con signo con `skimage.util.img_as_int`
  - Posteriormente sumar los bordes al canal verde de la figura rotada
  - Finalmente, normalizar a valores entre 0 y 255 con la función `numpy.clip`

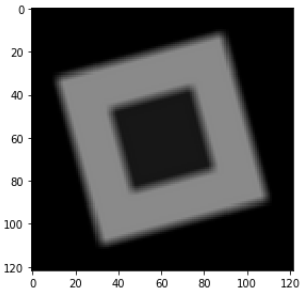
Ejemplo de resultados de la parte avanzada



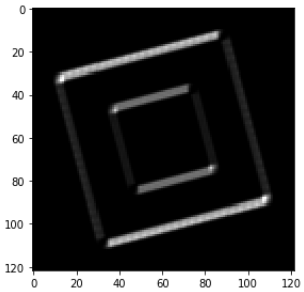
rotada



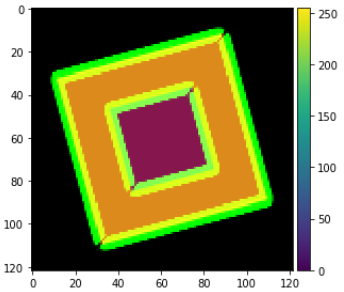
suavizada



color verde



bordes



bordes superpuestos al color verde (opcional)

Entregar 4 archivos:

- El código del módulo `foto.py`
- La imagen en formato `.jpg` utilizada para probar el módulo `foto.py`
- El código del módulo `imagen_creada.py`
- Un informe en `pdf` con:
  - capturas de pantalla del resultado de la parte A
  - capturas de pantalla del resultado de la parte B
  - capturas de pantalla del resultado de la parte avanzada, si se ha hecho