Visión por Computador

Grado en Informática Industrial y Robótica (GIIROB)

Práctica 1:

Adquisición y visualización de imágenes

Contenido:

Bibliografía:

- J. Howse & J. Minichino. “Learning OpenCV 4 Computer Vision with Python 3” (third Ed.) Packt Publishing Ltd. 2020.
- Sandigan Dei, “Python Image Processing. Cookbook”. Packt Publñishing Ltd. 2020.
- B.R. Hunt; R.L. Lipsman, J.M. Rosenberg. “A guide to MATLAB”. Cambridge University Press. 2001.
- Rafael Gonzalez, Richard Woods, Steven Eddins. ”Digital Image Processing Using Matlab”. Second Edition, McGraw Hill, 2010
- Official documentation for Python 3.10 https://docs.python.org/es/3/tutorial/index.html

En esta primera práctica se va a realizar una introducción del entorno de programación Thonny, para Python, con especial énfasis en las operaciones básicas para leer y escribir imágenes desde archivos, así como modificar las propiedades de dichas imágenes. Se asume un conocimiento previo del lenguaje y del entorno de programación. En el anexo 1 se tiene una descripción resumida del lenguaje Python.

Objetivos

- Recordar el entorno de programación Thonny para Python y las distintas bibliotecas involucradas, incluyendo OpenCV-Python, Matplotlib y Scikit-image.
- Entender el manejo de las operaciones de lectura y escritura de imágenes desde archivos.
- Describir las operaciones básicas de conversión y visualización de imágenes.

Material

El material a utilizar consistirá en los siguientes paquetes y entornos:

- Paquete de Python 3.10 (NO superior).
- Biliotecas de funciones OpenCV 4.0, Pillow, Matplotlib, Scikit-Image, Numpy y SciPy.
- IDE Tonny o similar.

Todos los paquetes y componentes indicados son libres y se pueden encontrar en internet.

1. Introducción

Python es un potente lenguaje de programación, de tipo interpretado, que combina estructuras de datos de alto nivel con un lenguaje simple y efectivo. También incorpora estructuras de programación orientadas a objetos (clases y métodos) y mecanismos de definición de variables no tipificadas.

La sintaxis simple y elegante de Python, junto con la proliferación de bibliotecas de funciones libres y módulos y herramientas de terceros, lo han llevado a ser el lenguaje más utilizado en la actualidad en entornos académicos y científicos. El intérprete de Python y su extensa librería estándar se encuentran disponibles libremente en código fuente y de forma binaria, para la mayoría de las plataformas desde la Web de Python, https://www.python.org/, y se pueden distribuir libremente. Se recomienda la utilización de la versión Python 3.10.x porque es actual y es la que permite en el entorno de desarrollo Thonny que se va a utilizar a lo largo del curso. También se puede usar cualquier otro IDE para Python, como PyChaarm, IDLE, Spyder, Atom, etc...

La referencia básica del lenguaje Python, donde se describe la sintaxis y la semántica de lenguaje, se puede encontrar en https://docs.python.org/es/3/reference/index.html#reference-index. Por otro lado, los tipos de objetos integrados no esenciales y las funciones integradas se pueden encontrar en https://docs.python.org/es/3/library/index.html#library-index. La biblioteca estándar del lenguaje (Python Standard Library) incluye un repertorio muy largo de funciones, algunas de las cuales están escritas en lenguaje C y codificados para el procesador, lo que permite acceder a los recursos y funcionalidades del sistema, como entrada-salida y archivos, que no serían accesibles directamente al programador. En cambio, la mayoría del resto de módulos están escritos en Python y permiten acceder a un montón de funciones estandarizadas para los problemas típicos de la programación. La idea es promover la portabilidad de estas funciones, abstrayendo las funcionalidades específicas de las plataformas y conseguir así interfaces (APIs) neutrales.

Además de las funciones estándar, hay una extensa colección de miles de componentes desarrollados por la comunidad de Python para todo tipo de aplicaciones. Esta colección se encuentra en el repositorio llamado Índice de Paquetes de Python (PyPI), y se puede encontrar en https://pypi.org/. Este es el repositorio de software para el lenguaje Python que permite encontrar e instalar paquetes, así como compartir el software propio con otras personas o comunidades.

Para la mejor comprensión de todos los tipos de datos, las estructuras de control y las funciones estándar del lenguaje, se recomienda visualizar el tutorial sobre Python que se puede encontrar en https://docs.python.org/es/3/tutorial/index.html.

De todos los paquetes y módulos reportados, los que nos interesan en particular son las herramientas dedicadas al trabajo con imágenes, entre las que se incluyen:

- NumPy, SciPy
- OpenCV 4.0
- Pillow
- Matplotlib
- Scikit-Image y Scikit-learn

NumPy es el paquete fundamental de Python para calculo científico. Incluye el objeto 'ndarray' que es una matriz n-dimensional que describe una colección de ítems del mismo tipo. Además, incluye una extensa colección de funciones al algebra lineal, de Transformada de Fourier y de números aleatorios. **SciPy** es una potente biblioteca de funciones de cálculo científico construida como extensión de NumPy. Incluye módulos para estadística, optimización, integración, algebra lineal, transformada de Fourier, proceso de señal e imagen, etc… También incluye estructuras de datos de alto nivel para manipular y visualizar imágenes. La documentación de NumPy y SciPy se puede encontrar en https://docs.scipy.org/doc/.

OpenCV (Open Source Computer Vision Library: http://opencv.org) es una extensa biblioteca de funciones, de código abierto y uso libre (Open-source BSD-licensed), que incluye cientos de funciones y de algoritmos de visión por computador. OpenCV tiene una estructura modular, con bibliotecas estáticas o compartidas, que incluye: funcionalidades básicas, proceso de imagen, análisis de vídeo, calibración de cámaras, reconstrucción 3D, detección de objetos, GUI de alto nivel, entrada/salida de vídeo, etc... La documentación de openCV se puede encontrar en https://docs.opencv.org/4.0.0/d1/dfb/intro.html.

Pillow es la versión sencilla y amigable de la biblioteca **PIL** (Python Image Library). La biblioteca PIL añade capacidades para la E/S y procesamiento de imágenes en Python. La documentación sobre Pillow se puede encontrar en https://pillow.readthedocs.io/en/latest/handbook/index.html.

Matplotlib es una biblioteca amplia para crear visualizaciones estáticas, animadas o interactivas en Python. Define un conjunto de funciones con una interfaz similar a la de MATLAB, denominada '*matplotlib.pyplot*', y también un amplio repertorio de métodos para trabajar con imágenes. La información sobre Matplotlib se puede encontrar en https://matplotlib.org/stable/tutorials/introductory/index.html.

Scikit-image es una colección de algoritmos para proceso de imagen en Python que trabaja con matrices numpy como imágenes. Es completamente libre y la información se puede encontrar en https://scikit-image.org/docs/stable/user_guide.html. Por su parte, Scikit-learn es otra biblioteca para 'Machine Learning' en Python, que incluye métodos para preprocesamiento de imágenes y extracción de características, así como para aprendizaje supervisado y no supervisado, clasificación, regresión, agrupamiento, PCA y optimización de modelos. La guía de usuario se puede encontrar en https://scikit-learn.org/stable/user_guide.html.

Por último, mencionar que para construir aplicaciones que requieran interfaces gráficas de usuario (Graphic User Interface: GUI) podemos necesitar algún módulo Python adicional. Existen bastantes de estos módulos, pero la instalación estándar de Python incorpora de serie el **Tkinter** (Tk Interface). Tkinter es una colección de funciones que permite introducir elementos gráficos (llamados widgets) para construir: ventanas, marcos, botones, desplegables, etc... Con estos elementos podremos hacer aplicaciones sencillas que permitan interactuar con el usuario. https://docs.python.org/es/3/library/tk.html

De todas las herramientas mencionadas, utilizaremos preferentemente OpenCV y Matplotlib para el procesamiento y manipulación de las imágenes en esta práctica. Así mismo, utilizaremos la aplicación Thonny como entorno de desarrollo.

2. El entorno de desarrollo (IDE)

Para trabajar con Python solo hace falta un editor de texto, editar los programas en texto plano (*.py) y luego ejecutar el Python.exe que crea un monitor para ejecutar estos programas. Pero siempre es más fácil y amigable emplear entornos de desarrollo (Integrated Development Environment: IDE) que son programas que integran todos los elementos para editar, analizar, ejecutar y depurar los programas en Python como en otros múltiples lenguajes.

Para Python existen muy diversos IDEs. Algunos son de pago, como PyCharm, Kdevelop, VisualStudio o SlicEdit. Otros son libres, como Thonny, PyDev o Spider. IDLE es un entorno sencillo que viene incluido con el propio software de Python. Finalmente, otros son simples editores de código como Sublime, Atom o Notepad++, pero que no permiten la ejecución/depuración. Los entornos IDE multiplataforma y multilenguaje, la mayoría de pago, son los más completos y permiten mucha funcionalidad. Sin embargo, son complejos de aprender y manejar.

Por este motivo, hemos seleccionado un entorno sencillo, especial para principiantes, como es el **Thonny**. Fue desarrollado en el Instituto de Ciencias de la Computación, de la Universidad de Tartu, Estonia, e incorpora el Python 3.10.x. La interfaz de usuario es muy básica, como se aprecia en la figura 1 siguiente.

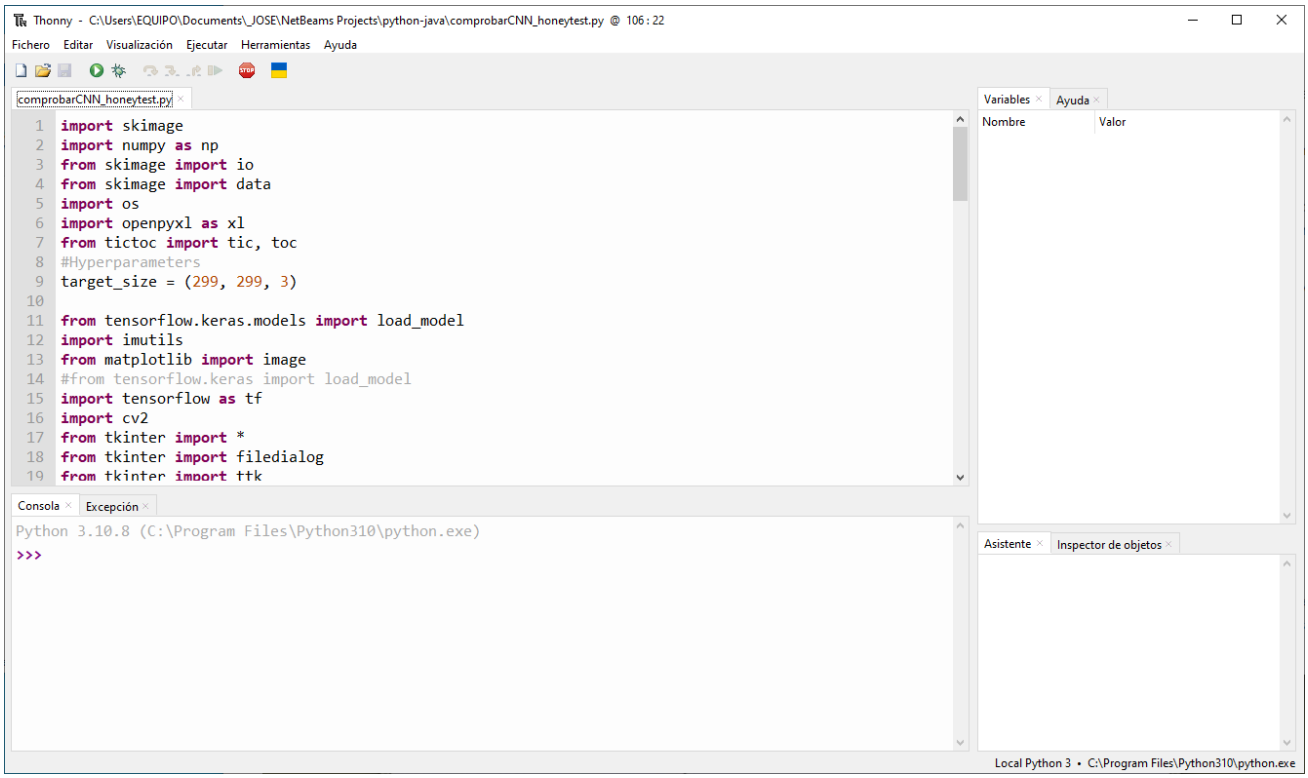


Figura 1 – La ventana principal del **Thonny**.

Utiliza un esquema de colores para marcar el código y representar las funciones. El editor permite ejecutar los programas, paso a paso y con puntos de parada para la depuración. También detecta los errores de sintaxis,

como paréntesis y comillas sin cerrar, y alerta de la repetición de nombres. También permite incorporar fácilmente módulos y paquetes de terceros obtenidos desde PyPI.

La descarga e instalación de Thonny se puede obtener en https://thonny.org/, cuidando que la instalación sea para todos los usuarios.

La figura 1 muestra la ventana principal del entorno **Thonny**. Dentro de ésta aparecen múltiple sub-ventanas, siendo la más importante la ventana de edición, donde mostramos el código Python que queremos ejecutar y que podemos editar, leer o guardar en archivos *.py.

La ventana que Consola muestra el marcador del entorno Python (>>>) donde podemos crear variables de todo tipo de manera manual e insertar y ejecutar órdenes de Python. También permite ejecutar los programas y mostrar los resultados de Entrada/Salida estos programas. Además, hay varias ventanas auxiliares para mostrar las variables en la depuración, el inspector de objetos o el asistente, que nos permitirán comprobar el estado del programa. Los menús de la aplicación incorporan todas las herramientas necesarias para el trabajo simple con Python.

El trabajo habitual en el entorno consiste en introducir las órdenes o funciones pertinentes en la ventana de Consola y visualizar los resultados, bien en la propia ventana o bien en una o varias ventanas auxiliares, creadas por las mismas funciones o por nosotros, denominadas *figuras*. Sin embargo, es MUY CONVENIENTE realizar este trabajo mediante pequeños programas o 'scripts' que podemos crear con el editor incluido en el entorno. Este editor nos permitirá ejecutar las secuencias de órdenes, total o parcialmente, hacer ejecución paso-a-paso y visualizar variables o resultados intermedios.

3. Tipos de imágenes

Los tipos de imágenes que podemos manejar en Python son los siguientes:

- BINARY:
  - Matriz de booleanos (0s y 1s) que representan una imagen blanco/negro. También se llama *bilevel image*.
- INDEXED:
  - Matriz de clase bool, uint8, uint16 o float cuyos valores son índices de un mapa de color (colormap). Un mapa de color es una matriz mx3 de las clases uint8 o float, donde cada elemento 'i' representa las proporciones de R, G y B para cada pixel de dicha imagen con valor 'i'. También se llama *pseudocolor image*.
- INTENSITY:
  - Matriz de clase uint8, uint16 o float cuyos valores representan la intensidad de los pixeles. También se llama *grayscale image* o *graylevel image*.
- TRUECOLOR:
  - Matriz de m x n x 3 valores de clase uint8, uint16 o float que representan las componentes de rojo (R), verde (G) y azul (B) de la imagen en color. También se llama *RGB image* o *BGR image,* según el orden de los canales.

Estos tipos de imágenes se almacenan en archivos mediante distintos formatos. En Python se utilizan, entre otros, los siguientes: BMP, PNG, JPEGG y TIFF.

4. La imagen en Python

En cualquier lenguaje, una imagen digital se representa mediante una matriz de números (booleano, enteros o de coma flotante). El tamaño de esta matriz (MxNxC) representa el número de filas (M) por el número de columnas (N) y el número de canales (C). Sin embargo, en Python se suele utilizar un tipo estándar para este propósito, el ndarray definido en la biblioteca NumPy.

Un ndarray es una matriz N-dimensional, que describe una colección de '*items*' del mismo tipo. Los *items* pueden ser indexados usando coordenadas enteras. Todos los ndarrays son homogéneos, pues todos sus elementos son tratados de la misma forma, ocupan el mismo espacio de memoria y son todos del mismo tipo. Este tipo puede ser un tipo escalar básicos (int, float,..) o pueden ser tipos compuestos.

La clase numpy.ndarray permite crear y manipular los ndarrays de múltiples formas. Para crear un elemento de esa clase usaremos el constructor de más bajo nivel:

- ndarray(shape[, dtype=float, buffer=None, offset=0, strides = None, order=None...])

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
... offset=np.int_().itemsize,
... dtype=int)                    # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

También podemos emplear otras formas de más alto novel, como array, zeros, empty o dtype, para matrices de tipos o datos concretos.

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]], np.int32)
>>> type(x)
<class 'numpy.ndarray'>
>>> x.shape
(2, 3)
>>> x.dtype
  dtype('int32')
```

Un array se puede acceder mediante indexación con los índices entre corchetes [] y separados por comas:

```
>>> x[1, 2]
6
```

Y también se pueden extraer sub-matrices de un array:

```
>>> y = x[:,1]
>>> y
array([2, 5], dtype=int32)
>>> y[0] = 9 # this also changes the corresponding element in x
>>> y
array([9, 5], dtype=int32)
>>> x
array([[1, 9, 3],
[4, 5, 6]], dtype=int32)
```

Existen innumerables métodos y atributos de los ndarray, que podemos encontrar en el archivo *numpy-ref.pdf*. Este archivo está disponible en PoliformaT.

4.1- Operaciones con imágenes en Pillow

En este apartado citaremos las funciones básicas para manejar las imágenes, el uso de paletas de colores y las operaciones de conversión entre tipos de imágenes, para la biblioteca **Pillow:**

En **Pillow** las imágenes se obtienen mediante la clase image:

```
>>> from PIL import Image
>>> im = Image.open("hopper.ppm")
>>> print(im.format, im.size, im.mode)
PPM (512, 512) RGB
>>> im.show()
```

También se pueden crear imágenes mediante diversos constructores:

- Image.new(mode, size, color=0)
- Image.fromarray(obj)
- Image.frombytes(mode, size, data, decode_name='raw', *args)
- Image.frombuffer(mode, size, data, decoder='raw', *args)

```
>>> from PIL import Image
>>> import numpy as np
>>> a = np.full((1, 1), 300)
>>> im = Image.fromarray(a, mode="L")
>>> im.getpixel((0, 0))
 # 44
>>> im = Image.fromarray(a, mode="RGB")
>>> im.getpixel((0, 0))
 # (44, 1, 0)
```

La lectura/Escritura de imágenes a/desde archivo se realiza mediante las funciones que se muestran a continuación para **Pillow**:

- Lectura de imágenes digitales desde archivos
  - Image A = Image.open(filename, mode= 'r', format= None)
    - format: identifica el formato por el contenido del archivo
- Escritura de imágenes en archivos
  - Bool res = Image.save(filename, format= None, **params)
    - format: BLP; BMP; DDS, DIB, EPS, GIF, ICNS, ICO, JPEG, JPEG2000, PCX, PNG, PPM, TIFF,..
    - Params: depende de cada formato (ver la documentación)

Las funciones para manejar las imágenes son:

Image.apply_transparency ()	Image.putalpha(alpha)
Image.copy()	Image.paste(im, box=None, mask=None)
Image.crop(box= None)	Image.merge(mode, bands)
Image.getbox()	Image.putdata(data, scale=1.0, offset=0.0)
Image.getchannel()	Image.putpalette(data, rawmode='RGB')
Image.getdata(band= None)	Image.putpixel(xy, value)
Image.getextrema()	Image.split()
Image.getpalette(rawmode='RGB')	Image.point(lut, mode=None)
Image.getpixel(xy)	

Image.convert(mode=None, matrix=None, dither=None, palette=Palette.WEB, colors=256)

- mode: modo al que convertir.
- dither: si se convierte de 'RGB' a 'P' , o de 'RGB' o 'L' a '1'.
- palette: si se convierte de 'RGB' a 'P'. Las paletas disponibles son Palette.WEB y Palette.ADAPTIVE.
- colors: Número de colores a usar en la paleta adaptativa. Por defecto son 256 colores.

Modos:

- '1': 1 bit/pixel
- 'L': 8 bits/pixel
- 'P': 8 bits/pixel con paleta de colores
- 'RGB': 3x8 bits/pixel
- 'RGBA': 4x8 bits/pixel con transparencia
- 'CMYK': 4x8 lits/pixel, separación de color
- 'YCbYr': 3x8 bits/pixel, formato vídeo
- 'LAB': 3x8 bits/pixel, L*a*b
- 'HSV': 3x8 bits/pixel, Hue, Saturation, Value
- 'I': 32 bits signed integer pixels
- 'F': 32 bits floating point pixels

Atributos:

- Image.filename: str
- Image.format: Optional[str]
- Image.mode: str
- Image.size: tuple[int]
- Image.with: int
- Image.height: int
- Image.palette: Optional[PIL.ImagePalette]
- Image.info: dict
- Image.is_animated: bool
- Image.n_frames: int

Un mapa o paleta de colores (colormap) es una matriz de m x 3 números. Cada fila de la matriz representa un vector de 3 coordenadas de color RGB. La fila k-esima representa las coordenadas de color, en el rango [0,255], asociadas al índice de color 'k' en una imagen de tipo INDICES

- ImagePalete.ImagePalette(mode= 'RGB', palette=None, size=0)
  - mode: el modo a usar con la paleta. Tipo 'RGB' por defecto.
  - pallete: Paleta opcional a usar. Matriz o lista de enteros unit8. La lista debe consistir en colores seguidos de todos los canales, por ejemplo: RGBRGBRGB…
- Image.remap_palette(dest_map, source_palette=None)
  Re-escribe la imagen con la paleta reordenada.
  - dest_map: lista de índice reordenados. Por ejemplo (0,1,2,5,7,4,3, 8...). list(range(256)) será una lista con la ordenación original.
  - source_pallete: Byte o None

Las imágenes se visualizan en ventanas – figures – mediante:

- Image.show(title=None)

Las funciones para transformar las imágenes son:

Image.tobitmap(name='image')
Image.tobytes(encoder_name='raw', *args)
Image.transform(size, method, data,..)
Image.traspose(method)
Image.verify()
Image.rotate(angle, resample,..)
Image.getextrema()
Image.getpalette(rawmode='RGB')
Image.getpixel(xy)
Image.resize(size, reasmple=…)
Image.reduce(factor, box=None)
Image.quantize(colors=256, method=…)
Image.getproiection()
Image.putpalette(data, rawmode='RGB')
Image.putpixel(xy, value)
Image.split()
Image.point(lut, mode=None)

Además de la clase Image, Pillow contiene muchas clases auxiliares como Image.Color; Image.File; Image.Filter; Image.Font; Image.Math; etc… Muchas de ellas son llamadas desde la clase principal Image. Otras, en cambio, puede ser usadas directamente. Para una descripción detallada de las clases, funciones y parámetros consulten las ayudas o tutoriales correspondientes:

https://pillow.readthedocs.io/en/latest/reference/Image.html#PIL.Image.open

4.2 - Operaciones con imágenes en **scikit-image**

En este apartado citaremos las funciones básicas para manejar las imágenes, el uso de paletas de colores y las operaciones de conversión entre tipos de imágenes, para la biblioteca **scikit-image.** El paquete principal se importa con el nombre **skimage.:**

En **scikit-image** las imágenes son matrices declaradas como *numpy arrays* (ndarrays), de forma que se puede usar todo el repertorio de funciones NumpPy para manipular estas matrices. Sin embargo, el paquete skimage incorpora varios sub-paquetes para importar y exportar las imágenes con los tipos que acepta la biblioteca, que son uint8, uint16, uint32, float, int8, int16 e int32.

```
>>> from skimage.util import img_as_ubyte
>>> image = np.array([0, 0.5, 1], dtype=float)
>>> img_as_ubyte(image)
array([  0, 128, 255], dtype=uint8)
>>> from skimage.util import img_as_uint
>>> out = img_as_uint(sobel(image))
>>> plt.imshow(out)
```

skimage incorpora múltiples funciones y métodos para trabajar con las imágenes, como data, io, exposure, filters, transform, utils, etc…

```
>>> from skimage import data
>>> camera = data.camera()
>>> type(camera)
<type 'numpy.ndarray'>
>>> camera.shape
(512, 512)
>>> camera.size
262144
```

También se puede trabajar con OpenCV, pero teniendo en cuenta que OpenCV utiliza imágenes en color BGR y **scikit** usa RGB. En OpenCV las imágenes indican 'ancho', 'alto' y 'color', mientras que **scikit** los denomina 'rows', 'columns' y 'channel'. La siguiente instrucción invertirá el canal de color dejando las dimensiones de la imagen como estaban:

```
>>> img = img[:, :, ::-1]
```

Para trabajar con imágenes de OpenCV podemos hacer:

```
>>> from skimage.util import img_as_float

>>> image = img_as_float(any_opencv_image)
```

Para trabajar con imágenes de **scikit-image** podemos hacer:

```
from skimage.util import img_as_ubyte

>>> cv_image = img_as_ubyte(any_skimage_image)
```

Las funciones de conversión entre espacios de color es muy variada y se encuentra en el módulo **skimage.color**:

skimage.color.gray2rgb(image,..)	skimage.color.lab2rgb(image,..)
skimage.color.gray2rgba(image,..)	skimage.color.rgb2lab(image,..)
skimage.color.rgb2gray(image,..)	skimage.color.label2rgb(label,..)
skimage.color.hsv2rgb(image,..)	skimage.color.rgba2rgb(image,..)
skimage.color.rgb2hsv(image,..)	…

Para una información más detallada de estas funciones y sus parámetros consulte la guía de usuario disponible en:
https://scikit-image.org/docs/stable/user_guide.html

La visualización de datos se emplea habitualmente otros paquetes más potentes como **matplotlib**.

4.3 - Operaciones con imágenes en **OpenCV**

En este apartado citaremos las funciones básicas para manejar las imágenes, el uso de paletas de colores y las operaciones de conversión entre tipos de imágenes, para la biblioteca **OpenCV.**

En **OpenCV** las imágenes se tratan mediante la clase Mat, que permite construir ndarrays pero con métodos y atributos propios de las imágenes.

- cv.Mat(int ros, int cols, int type)

Donde type puede ser: CV_<bit depth>{U|S|F}C(<number of channels>)

  - bit_depth: 8,16,32,..
  - U,S,F: Unsigned, Signed, Float
  - Ejemplo: cv.Mat(64,64,CV_8UC3) para imagen de 3 canales de valores de 8 bits sin signo, tipica BGR (No RGB).

```
>>> # crear una imagen de ceros o diagonal
>>> import cv2 as cv
>>> Mat img = cv.zeros(5,5,CV_32F)
>>> Mat img = cv.eye(5,5,CV_32F)
```

```
Eye:
1,  0,  0,  0,  0
0,  1,  0,  0,  0
0,  0,  1,  0,  0
0,  0,  0,  1,  0
0,  0,  0,  0,  1

Ones:
1,  1,  1,  1,  1
1,  1,  1,  1,  1
1,  1,  1,  1,  1
1,  1,  1,  1,  1
1,  1,  1,  1,  1

Zeros:
0,  0,  0,  0,  0
0,  0,  0,  0,  0
0,  0,  0,  0,  0
0,  0,  0,  0,  0
0,  0,  0,  0,  0
```

```
>>> import numpy as np
>>> import cv2 as cv
>>> img = cv.imread('messi5.jpg')
>>> px = img[100,100]
>>> print( px )
[157 166 200]
# accessing only blue pixel
>>> blue = img[100,100,0]
>>> print( blue )
157
>>> img[100,100] = [255,255,255]
>>> print( img[100,100] )
[255 255 255]
```

La clase Mat es equivalente a la clase ndarray, por lo que se pueden intercambiar. Para acceder a los elementos de la imagen se puede utilizar la indexación propia de los ndarrays, que viene con la funcionalidad de NumPy, pero este método se considera lento, sobre todo para trabajar con imágenes de gran tamaño. Por eso se prefiere usar los métodos array.item() y array.itemset():

```
# accessing RED value
>>> img.item(10,10,2)
59
# modifying RED value
```

```
>>> img.itemset((10,10,2),100)
>>> img.item(10,10,2)
100
```

La lectura/Escritura de imágenes a/desde archivo se realiza mediante las funciones que se muestran a continuación:

- Lectura de imágenes desde archivos
  - Mat A = cv.imread(filename,flags = IMREAD_COLOR)
    - flags: cv.IMREAD_COLOR; cv.IMREAD_GRAYSCALE; cv.IMREAD_UNCHANGED,…
- Escritura de imágenes en archivos
  - Bool res = cv.imwrite(filename,img [,params])
    - Img: es un tipo Mat o equivalente
    - Params: cv.IMWRITE_JPEG_QUALITY; cv.IMWRITE_PNG_STRATEGY; …

```
>>> cv_image = img_as_ubyte(any_skimage_image)
```

Las funciones de conversión entre espacios de color es muy variada y se encuentra en el módulo cv:

cv.cvColor(image, flag)
cv.cvtColotTwoPlane()
cv.demosaicing()
flag:
cv.COLOR_BGR2BGRA, cv.COLOR_RGB2RGBA, cv.COLOR_BGRA2BGR, cv.COLOR_RGBA2RGB
cv.COLOR_GRAY2RGB, cv.COLOR_RGB3GRAY, ….(más de 100 códigos de conversión de color)

Las funciones para transformar las imágenes son muy variadas, entre las que se puede destacar:

cv.convertMaps(..)
cv.getAffineTransform()
cv.getPerspectiveTransform()
cv.getRectSubPix(…)
cv.getRotationMatrix2D()
cv.invertAffineTransform()
……..
cv.resize(…)
cv.remap(….)
……..

La visualización de datos se emplea habitualmente otros paquetes más potentes como **matplotlib**.

Para una descripción detallada de las funciones y sus parámetros consulten las ayudas o tutoriales correspondientes.
https://docs.opencv.org/4.0.0/dc/d2e/tutorial_py_image_display.html

4.4 - Operaciones con imágenes en **Matplotlib**

Para **Matplotlib** se puede emplear el tipo ndarray, pero las imágenes de pueden leer o guardar en Pillow:

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> from PIL import Image
>>> ii = Image.open(filename)    # ii tipo Image
>>> img = np.asarray(ii)         # img tipo ndarray
>>> imgplot = plt.imshow(img)
```

El módulo matplotlib.pyplot es una colección de funciones que hacen que **matplotlib** trabaje como MATLAB. Cada función de pyplot crea y cambia las figuras, crea ejes donde situar los gráficos y dibujos, decora todos los elementos, etc... siempre con la misma terminología que MATLAB.

Para trabajar con ventanas e introducir todo tipo de gráficos se emplean las funciones estándar:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
```

Fig = plt.figure()	plt.xlabel(str)
fig, ax = plt.subplots(..)	plt.ylabel(str)
plt.plot(….)	plt.title(str)
ax.set_xlabel(…)	plt.legend()
ax.set_ylabel(…)	plt.gcf()
ax.set_title(…)	plt.gca()
ax.legend(…)	plt.sca()
ax.grid(bool)	
ax.annotate(….)	……..
ax.set_xscale(…)	
ax.set_yscale(…)	
…….	

Para trabajar con imágenes, no con gráficos, se utiliza las funciones definidas en matplotlib.pyplot, con:

- plt.imshow (I), plt.imshow (I,[low high]), plt.imshow (I,map)
- imgplot = plt.imshow (….)

- Las paletas de colores se definen en los módulos matplotlib.colorbar y matplotlib.cm. Para mostrar la paleta de colores de forma gráfica podemos usar la función, más sencilla, del módulo pyplot:
  - plt.colorbar (….)
- Además, existen docenas de módulos dentro de matplotlib para innumerables operaciones, como matplotlib.units, matplotlib.scale, matplotlib.projections, matplotlib.image, matplotlib.colors, matplotlib.container, matplotlib.contour, matplotlib.figure, ….

Para obtener información de todos estos módulos de matplotlib consultar:
https://matplotlib.org/stable/tutorials/index.html
https://matplotlib.org/stable/api/matplotlib_configuration_api.html

5. Ejercicios y entrega de resultados

► Ejercicio 1 – Trabajar con imágenes

1. Descargue algunas imágenes de la carpeta <practicas\IMAGES> en PoliformaT o busque en internet imágenes de cualquier tipo, como paisajes, flores, galaxias, etc..; con formatos diversos como: jpg, tif, gif o png. Como ejemplo busque alguna imagen de galaxias o nebulosas en :
   https://www.reddit.com/r/astrophotography/
   Ponga los archivos de imagen en la carpeta de trabajo actual y muéstrelas una ventana.

2. Lea alguno de los archivos de color de tipo tif descargado muéstrelo en una ventana mediante Pillow. Si no se visualiza bien intente poner el mapa de colores o paleta que se corresponda. Muestre también la barra de colores al lado de la imagen. ¿Cuántos colores tiene la imagen?

3. Reduzca el número de colores de la imagen anterior a 16 y muéstrela. Compare ambas imágenes visualmente.

4. Repita las operaciones anteriores con una imagen de tipo jpg. Conviértala primero a tipo 'P' con paleta y repita las operaciones anteriores.

► Ejercicio 2 – Manipulación de la imagen

1. Utilizando cualquiera de las imágenes introducidas en el ejercicio anterior, aplique distintas transformaciones y compruebe el resultado.

2. Convierta cualquier imagen en color en una imagen de niveles de gris y visualice ambas.

3. Tomar una imagen RGB y muestre los tres componentes R (rojo), G (verde) y B (azul) como imágenes de gris por separado. Pruébelo con la imagen 'aloel.jpg'.

► Ejercicio 3 – Aritmética con imágenes

1. Lea las imágenes '*cameraman.tif*' y "*moon.tif*", que se encuentras en la instalación de MATLAB.

2. Redimensiónelas para que tengan el mismo tamaño, por ejemplo 256 x 256.

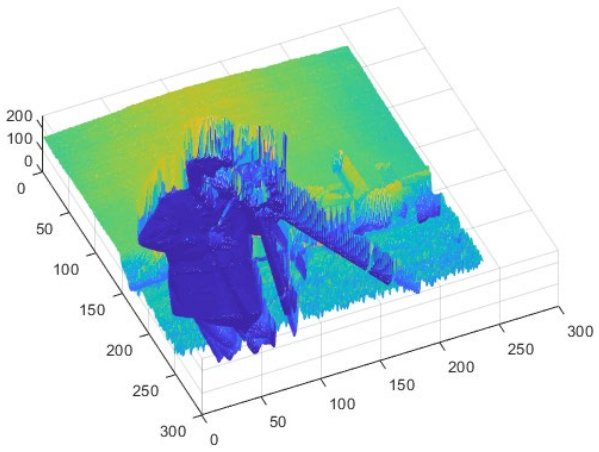
3. Realice algunas operaciones aritméticas entre ellas y visualice el resultado.

4. Realice la combinación lineal siguiente y visualice:

$$S = CAM * 1.8 – MOON * 1.2 + 128$$

► Ejercicio 4 – Visualización 3D

1. Tome alguna de las imágenes anteriores, de tipo grayscale, y represéntela como una superficie empleando las funciones plt.contourf y numpy.meshgrid.



Para entregar los resultados de la práctica, haga un documento (Word o similar) con los distintos ejercicios incluyendo, en cada uno, lo siguiente:

- Enunciado del ejercicio
- Script de Python (copy – paste)
- Imágenes de los resultados (menú ***Edit->Copy figure*** en la ventana y paste en el Word)
- Comentarios personales sobre el ejercicio y los resultados (si procede)

Incluya, en la página inicial, el título de la práctica y el nombre del/los alumnos. Convierta el archivo a pdf y súbalo a PoliformaT al espacio compartido de cada alumno implicado.