Colab Python Vision Inspection Systems

Francesco Lupi

2024/2025

0.1 What is Python?

Python is a "high level" **programming language** (i.e., syntax) that makes it accessible and productive for programmers from any background or experience level. If you are curious check out the LINK.

You can:

download Python on your pc from the "*download*" button. To use it you need to install a "*code editor*" or better "*Integrated Development Environment*" (IDE). An example Pycharm

use web-based python notebook editors : Colab and Jupyter. Colab is hosted on a virtual machine which is essential another computer at Google running your code (NB: Colab has already installed many of the popular libraries you may need to run your code and can be accessed from anywhere).

NOTE: IDEs delivered as cloud-based Software-as-a-Service (**SaaS**) offer unique advantages over local development environments.

-Firslty there is no need to download software and configure local environments, developers can get on projects right away.

-Secondly, the a high level of standardization for team members' environments is provided, and the team can align the operations performed on their own computers.

-Thirdly, centralized development environment management also helps reduce potential security and intellectual property concerns because the code does not reside on individual developer computers.

-Lastly and obviously, the impact of processes on local computers changes.

0.2 Python core concepts

```
# Variables, type, and string concatenation (functions of the object)

# Defining all the variables of interest
string = "Hello, this year is "  # STRING
year = 2017  # INTEGER
today_temperature = 28.6  # FLOAT
hot = True  # BOOLEAN (be careful, Python is case-sensitive)

# Calling a function that uses all the variables defined above
print(string.upper() + str(year + 5) + ' and in November it will be ' + str(today_temperature) + ' degrees. Sad, but ' + str(hot))
type(string)

# Other conversions
# int(float)
# int(string)
# int(boolean)
# float(string)
# float(int) ------ be careful
# float(boolean) ------ be careful
# etc..

# Type conversion and rounding
a = int(round(today_temperature, 0))
a
```

0.2 Python core concepts

```
[ ]  #aritmetic operators

     #+
     #-
     #/ note that / return a float and // return an integer
     #*
     #others like exponentation etc.


     #comparison operators

     #>
     #>=
     #<
     #<=
     #==
     #!=

     #logical operators

     #AND (both are true)
     #OR (at least one is true)
     #NOT
```

0.2 Python core concepts

```
# Print numbers from 1 to 9 using While, For loops, and List

x = []  # Empty list to store the numbers

# While loop to print numbers from 1 to 9
i = 1
while i < 10:
    print(i)         # Print the current value of i
    x.append(i)      # Append the current value of i to the list x
    i = i + 1        # Increment i by 1
print('DONE while loop')

# For loop to iterate over the list x and print each element
for element in x:
    print(element)
print(x)  # Print the entire list
print('DONE for loop')

# Using range to print numbers from 0 to 8
for el in range(i):
    print(el)
```

0.2 Python core concepts

```python
# User-Defined Functions (UDFs)
# 1. What arguments (if any) it takes
# 2. What values (if any) it returns

# Declare the name of the function
def MIA_addizione(a, b):
    # Compute the sum of the two inputs and save in a variable
    c = a + b
    # Return the value
    return c

# Now call the function

# Calling the function with string arguments
print(MIA_addizione('ciao', ' come va'))  # String addition is concatenation!

# Calling the function with integer arguments
print(MIA_addizione(5, 3))  # Integer addition
```

1. Image Pre-Processing

1.1 Upload the packages needed for image processing

```
[1]  # NumPy is a Python library used for working with arrays
     # np = common abbreviation for numpy
     import numpy as np
```

```
[2]  # matplotlib is a collection of functions that make matplotlib work like MATLAB (for plots)
     from matplotlib import pyplot as plt
     %matplotlib inline
     #è possibile importare tutta la libreria ma è più onerosa
     ## import matplotlib as mplt
```

```
[3]  # OpenCV-Python is a library of Python bindings designed to solve computer vision problems
     import cv2
```

```
[4]  #packages provides a number of general image processing and analysis functions that
     #are designed to operate with arrays of arbitrary dimensionality.
     import scipy.ndimage as filt
```

```
[29] !pip install gdown  # Install gdown if you don't have it

     import gdown
```

1.2 Work with the image by making some tests and relative visualizations for visual confirmation

```
#LOAD IMAGE INTO THE WORKFOLDER
# 1. use !Wget function to load image from web or drive via LINK
# 2. load it manually drag and drop
# 3. upload it from local. Usually r is used before the path to make it raw.

#(example_from WEB: fractal plant image)
##!wget "https://digitalreflectionswamf14.files.wordpress.com/2014/09/cropped-ferns.jpg" -O FractalPlant.jpg

#(example_from web: peso image)
#!wget "http/" -O peso.jpg

#(example_from Gdrive: Coin image made with smartphone)
# Google Drive file ID
file_id = '1GR57ZcG_QuCZ1suB_7_8mhfrpT193y2E'

# Construct the download URL
url = f"https://drive.google.com/uc?id={file_id}"

# Download the file
gdown.download(url, 'peso.jpg', quiet=False)


#(example_from local: interactive)
##from google.colab import files
##files.upload()
```

1.2 Work with the image by making some tests and relative visualizations for visual confirmation

```
[36] #VARIABILE img: np matrix made by three channels, namely Red Green Blue (RGB)

     #When using cv2.imread remeber that store image in BGR so you need to convert
     ##img1 = cv2.imread('/content/FractalPlant.jpg')
     ##img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)

     img = plt.imread('/content/peso.jpg')

     type(img)
```

```
numpy.ndarray
```

```
#CHECK the ndarray

#number of dimensions of the matrix
print(np.ndim(img))

#matrix shape. N of rows, columns and the dimention of the matrix
print(np.shape(img))

#total product of elements (i.e., pixels) in row*columns*dimentions
print(np.size(img))
```

```
3
(1450, 1462, 3)
6359700
```

1.2 Work with the image by making some tests and relative visualizations for visual confirmation

```
#VISUALIZE the image as the superimposition of the three channels (or dimentions)
#in a plot with the row and columns dimentions

plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x790cfbda7cd0>
```

1.2 Work with the image by making some tests and relative visualizations for visual confirmation

```
#CHECK1 THE IMAGE MATRIX

print(img)  #image saved as a pixel matrix
```

```
[[[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]
```

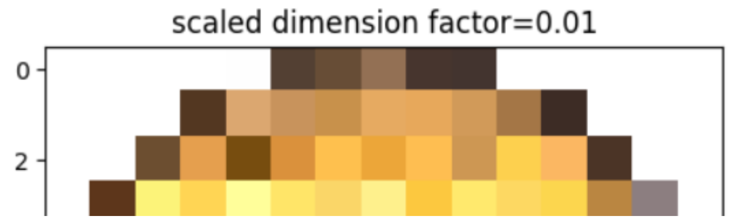
1.2 Work with the image by making some tests and relative visualizations for visual confirmation

```
[41] #CHECK2 THE IMAGE MATRIX

     print(img[200,:,1]) #print the line 200 of the image on the GREEN channel (R=0, G=1, B=2)

     [255 255 255 ... 255 255 255]
```

```
#CHECK3 THE IMAGE MATRIX

print(img[200,300:310,1]) #print the values from 300 to 310 of image row 200 on the GREEN channel (R=0, G=1, B=2)

[61 60 58 55 63 65 63 58 60 68]
```

1.2 Work with the image by making some tests and relative visualizations for visual confirmation

```
#SCALE THE SIZE OF THE IMAGE

#Scaling Factor or Scale Factor is usually a number that scales or multiplies some quantity,
#in our case the width and height (i.e., rows and columns) of the image.
#It helps keep the aspect ratio intact and preserves the display quality. So the
#image does not appear distorted, while you are upscaling or downscaling it.

#del scale_down
scale_down = 0.01
#scale_down = '0.6'

scaled_f_down = cv2.resize(img, None, fx= scale_down, fy= scale_down, interpolation= cv2.INTER_LINEAR)

plt.imshow(scaled_f_down)
plt.title('scaled dimension factor='+str(scale_down))
plt.show()
plt.imshow(img)
plt.title('original dimension 660x660')

#more info about resizing here: https://learnopencv.com/image-resizing-with-opencv/
```



1.3 Esploring the 3 channels: Red,Green, Blue (RGB)

```
#IMAGE SINGLE BAND PLOT (RGB)

#INFO about Color space: https://learnopencv.com/color-spaces-in-opencv-cpp-python/
#Difference between additive primaries (RGB - emitted spectrum) and subtractive primaries (CMY - white light incident on pigment, absorbed spectrum)

provaBanda = img.copy()  # This will create a shallow copy by initializing a whole different instance rather than referencing it (you reference it by using the '=' operator in numpy).
#More info here https://numpy.org/doc/stable/reference/generated/numpy.copy.html
#NOTE1: provaBanda = img[:,:,:] acts the same as .copy(). When you want to copy all the components, use .copy() or select all the components [:,:,....]
#NOTE2: x = img[:,:,1] acts as a shallow copy as well. When you want to copy a component and assign it to a new vector, there's no need to use .copy()

# The band that I do not set to 0 is the one chosen (R=0; G=1; B=2)
provaBanda[:,:,0] = 0  # Set R to zero
provaBanda[:,:,2] = 0  # Set B to zero

plt.imshow(provaBanda)
plt.title('3 matrices: R, B set to zero and G from 0 to 255')
plt.show()
plt.imshow(img[:,:,1], cmap='gray')  # Without cmap, the one-dimensional plot uses a standard cmap that highlights differences
plt.title('Single matrix: G from 0 to 255')
plt.show()
plt.imshow(img)
plt.title('Original 3 matrices and related 3 channels')
```

1.3 Esploring the 3 channels: Red,Green, Blue (RGB)

```
[ ]  #COLOR SPACES RGB, HSV, HLS

     #HSV (Hue, Saturation, Value)
     img3 = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)

     #HLS (Hue, Lightness, Saturation)
     ##img2 = cv2.cvtColor(img, cv2.COLOR_RGB2HLV)
```

1.4 Grayscale: 256 values in the gray shadows from white to black

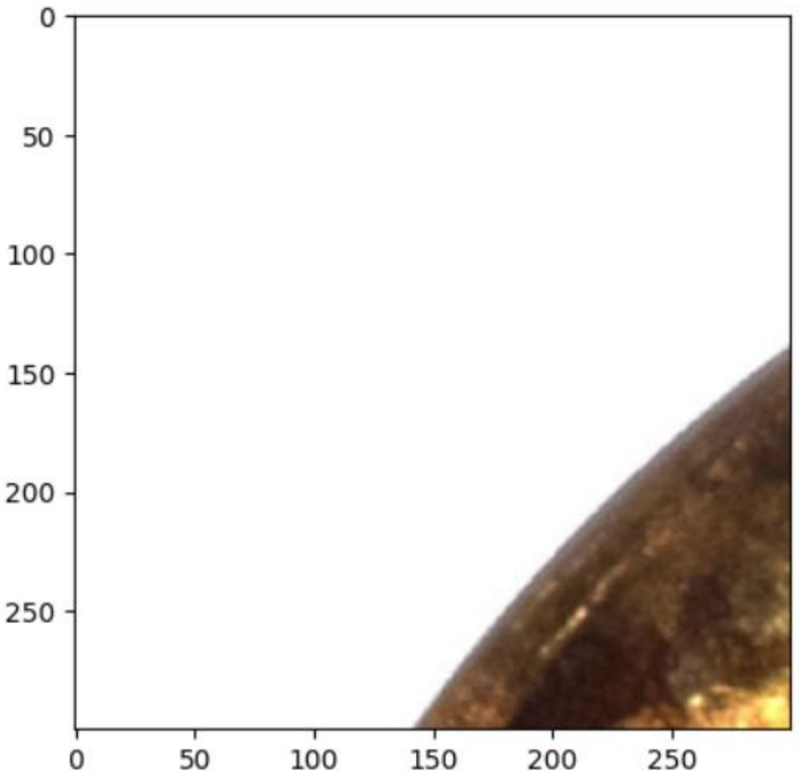
```python
# Usually, VALUE (luminosity) is used to convert an image to grayscale
# (I could use R, G, or B indifferently for grayscale, which one to use? For this, I convert to HSV and usually use V.
# Note that for specific applications, a channel like G might be used if it highlights a better feature.)

img2 = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)

lum_img = img2[:, :, 2]  # Extracted and copied (in this case, no need to use .copy()) the component V = value (H=0; S=1; V=2)

# Plot V using a colormap to only plot in grayscale [0,255] using a total of 256 values on a mono-dimensional matrix (NOT 256 ON the 3 RGB channels)
# BE CAREFUL when using the function imshow without cmap='gray' like this "plt.imshow(lum_img)"
# In this case, it considers all the 3 channels superimposed (256 red, 256 green, 256 blue).
plt.imshow(lum_img, cmap='gray')

# Add colorbar and title
plt.colorbar()
plt.title('Plot VALUE in grayscale')
```

1.4 Grayscale: 256 values in the gray shadows from white to black

```
# GRAYSCALE IMAGE CONVERSION USING prebuilt function. In some cases, it works better.

imgGray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)  # Use prebuilt function to convert RGB (660,660,3) to GRAY SCALE (660,660)

plt.imshow(imgGray, cmap='gray')  # REMINDER: To plot MONO-DIMENSIONAL matrices, use cmap 'gray'
plt.colorbar()
plt.title('Grayscale plot using prebuilt function')
plt.show()
plt.imshow(img)
plt.title('Original image with 3 channels')
```

1.4 Grayscale: 256 values in the gray shadows from white to black

```
#REGION OF INTEREST (ROI) or CROP
imgplot=plt.imshow(img[0:300,0:300])
```

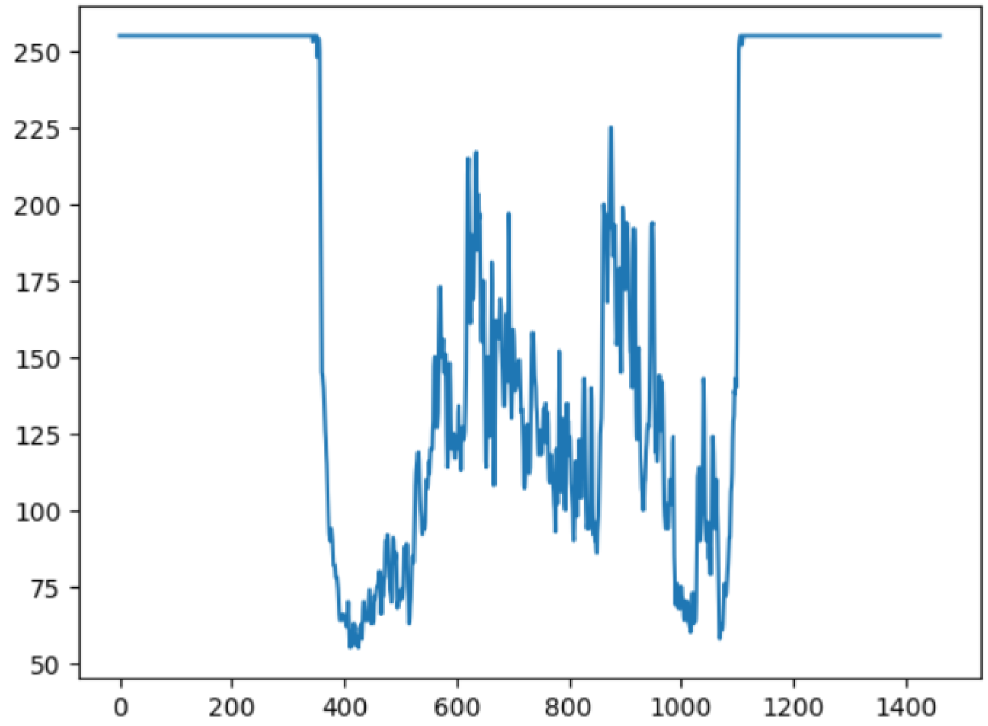


2. Histograms and Binarization: Black and White

2.0 Light profile

```
#Plot one row in the LIGHT channel
img3 = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
plt.plot(img3[100,:,1] )
```

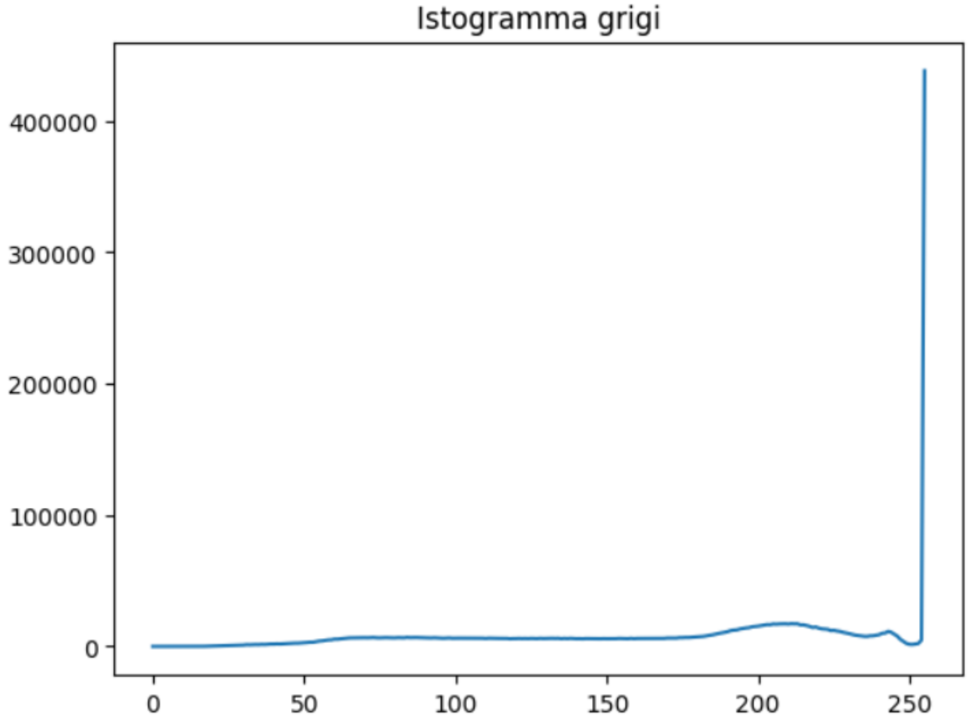
```
[<matplotlib.lines.Line2D at 0x790cf9ef05e0>]
```



2.1 Gray histogram

```
#GRAY historgram (USE THIS ONE)
plt.plot(filt.histogram(imgGray[:,:], 0, 255, 256 )) #immagine, min,max, n classi
plt.title('Istogramma grigi')
```

```
Text(0.5, 1.0, 'Istogramma grigi')
```



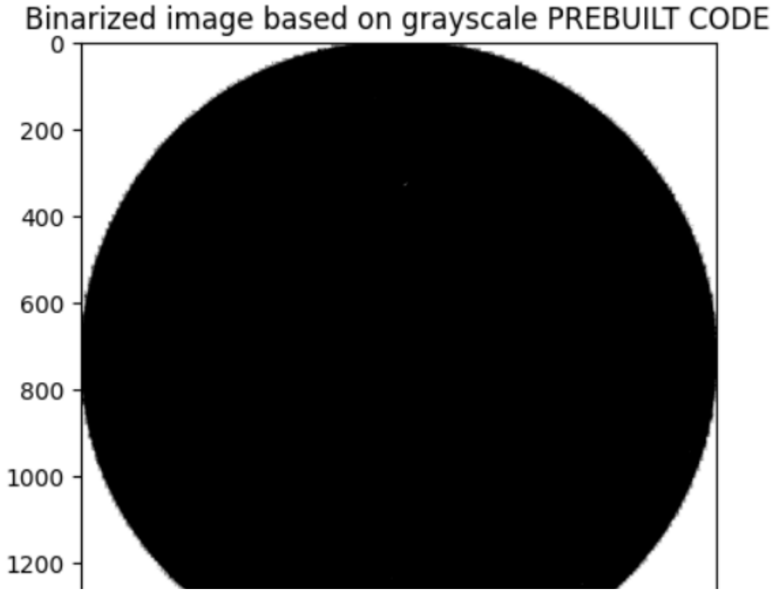
2.2 Global Binarization (Manual)

```
# GLOBAL MANUAL BINARIZATION: FOR LOOP

# MONOCHANNEL VERSION
thresh = 254  # Set the threshold for binarization based on the bi-modal histogram (in this case, for the first example, let's consider the green channel)
imgBIN = np.zeros((rishor, risver))  # Create a matrix of zeros (black) with the image's row and column dimensions
# Loop for binarization where I assign a value of 255 (white) for all pixels above the threshold
for riga in range(rishor):
    for col in range(risver):
        if img[riga, col, 1] >= thresh:  # CHOSEN MONOCHANNEL
            imgBIN[riga, col] = 255

plt.imshow(imgBIN, cmap='gray')  # Use cmap 'gray' to plot black and white (using the 0,255 value scale)
plt.title('Binarized image based on the green channel MANUAL CODE')
plt.show()

# GRAYSCALE VERSION (USE THIS PREFERABLY WHEN YOU WANT TO BINARIZE AND WORK ON GRAYSCALE INPUT IMAGE)
thresh = 254  # Set the threshold for binarization based on the bi-modal histogram (usually grayscale histogram)
imgBIN = np.zeros((rishor, risver))  # Create a matrix of zeros (black) with the image's row and column dimensions
# Loop for binarization where I assign a value of 255 (white) for all pixels above the threshold
for riga in range(rishor):
    for col in range(risver):
        if imgGray[riga, col] >= thresh:  # GRAYSCALE
            imgBIN[riga, col] = 255

plt.imshow(imgBIN, cmap='gray')  # Use cmap 'gray' to plot black and white (using the 0,255 value scale)
plt.title('Binarized image based on grayscale MANUAL CODE')
```

2.3 Global Binarization (Automated)

```python
# GLOBAL AUTOMATED BINARIZATION: SOLUTION 1

threshold = 254  # Create threshold based on the grayscale histogram
imgbin = ((imgGray > threshold))

plt.imshow(imgbin, cmap='gray')
plt.title('Binarized image based on grayscale PREBUILT CODE')
```

```
Text(0.5, 1.0, 'Binarized image based on grayscale PREBUILT CODE')
```



2.3 Global Binarization (Automated)

```python
# GLOBAL AUTOMATED BINARIZATION: SOLUTION 2 - OpenCV function cv2.threshold

# INFO1: About cv2.threshold here https://www.pyimagesearch.com/2021/04/28/opencv-thresholding-cv2-threshold/#:~:text=We%20use%20the%20cv2.,T%2C%20is%20the%20threshold%20value.)
# INFO2: About OpenCV here https://learnopencv.com/opencv-threshold-python-cpp/

# NOTE1: Working on a grayscale image for binarization usually gives better results (imgGray).
# It's also possible to binarize by considering a channel of the image (e.g., the green channel img[:,:,1]) in specific cases, as mentioned earlier, but it should generally be avoidec

# NOTE2: The cv2.threshold function returns a tuple of 2 values: the first, T, is the threshold value. In the case of simple thresholding, this value is trivial since we manually supp
# But in the case of Otsu's thresholding, where T is dynamically computed for us, it's nice to have that value. The second returned value is the thresholded image itself.

T1, thresh1 = cv2.threshold(imgGray, 254, 255, cv2.THRESH_BINARY)  # Image input, threshold T, output value for pixels above the threshold
# Thresholding method chosen: BINARY, in this case, ANY pixel intensity p that is greater than T is set to the output value, and any p that is less than T is set to 0

# Let's see other methods
T, thresh2 = cv2.threshold(imgGray, 250, 255, cv2.THRESH_BINARY_INV)  # Inverse of BINARY function (sets pixels above threshold to 0 and those below to the output value)
T, thresh3 = cv2.threshold(imgGray, 250, 255, cv2.THRESH_TRUNC)  # The destination pixel is set to the threshold if the source pixel value is greater than the threshold. Otherwise, it
T, thresh4 = cv2.threshold(imgGray, 250, 255, cv2.THRESH_TOZERO)  # The destination pixel value is set to the pixel value of the corresponding source if the source pixel value is grea
T, thresh5 = cv2.threshold(imgGray, 250, 255, cv2.THRESH_TOZERO_INV)  # Inverse of TOZERO function (The destination pixel value is set to zero if the source pixel value is lower than

titles = ['Original Image', 'BINARY', 'BINARY_INV', 'TRUNC', 'TOZERO', 'TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(6):
    plt.subplot(2, 3, i + 1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
```

2.3 Global Binarization (Automated)

```
# GLOBAL ADVANCED AUTOMATIC THRESHOLD (OTSU): Using OpenCV

(T, threshOTSU) = cv2.threshold(imgGray, 0, 255, cv2.THRESH_OTSU)  # 0 in this case means 'I don't care' about the threshold.
# In this way, the algorithm chooses the optimal value for T by itself. It is based on the fact that the grayscale histogram is bimodal
# (Otsu's method assumes that our image contains two classes of pixels: the background and the foreground) and tries to find the optimal T val
plt.imshow(threshOTSU, 'gray')

print('According to OTSU, the optimal threshold is T=', T)  # Let's see the optimal threshold that OTSU found. It's not that optimal.
# This is because the histogram is not bimodal, so it cannot find the optimal T in an optimized way.
```

```
According to OTSU, the optimal threshold is T= 163.0
```

2.4 Adaptive threshold (Automated)

```
# AUTOMATIC DINAMIC (aka ADAPTIVE) THRESHOLD

#INFO: https://www.pyimagesearch.com/2021/05/12/adaptive-thresholding-with-opencv-cv2-adaptivethreshold/

#when the lighting conditions are non-uniform — such as when different parts of the image are illuminated more than others,
#we can run into some serious problem. And when that is the case, we will need to rely on ADAPTIVE thresholding.
#The general assumption that underlies all adaptive and local thresholding methods is that smaller regions of an image are more likely to have
#thus a specif threshold is set for specific areas. Choosing the size of the pixel neighborhood for local thresholding is therefore crucial. To


threshDINAMICmean = cv2.adaptiveThreshold(imgGray, 255,cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 659,1) #image in input, #output value
#cv2.ADAPTIVE_THRESH_MEAN_C indicate that we are using the arithmetic mean of the local pixel neighborhood to compute our threshold value of T.
#The fourth value to cv2.adaptiveThreshold is the threshold method, again just like the simple thresholding and Otsu thresholding methods we pa
#The fifth parameter is pixel neighborhood size (aka Kernel). DEVE ESSERE DISPARI in questo algoritmo. Computing the mean grayscale pixel inten
#Constant C subtracted from the mean or weighted mean (see the details below). Normally, it is positive but may be zero or negative as well.

#usiamo un altro metodo cv2.ADAPTIVE_THRESH_GAUSSIAN_C
threshDINAMICgauss = cv2.adaptiveThreshold(imgGray, 255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 151,1)


plt.imshow(threshDINAMICmean, 'gray')
plt.title('mean')
plt.show()
plt.title('Gaussian average')
plt.imshow(threshDINAMICgauss, 'gray')
```

3. Image filtering

3.0 Kernel

```
#example of 3x3 IDENTITY kernel.
#NOTE: the sum of the element must be 1 because is a weighted mean. Each element of the kernel is multiplied for the elment of the matrix below

kernel1 = np.array([[0, 0, 0],
                    [0, 1, 0],
                    [0, 0, 0]])
```

3.0 Kernel

```
#EXAMPLE 1: filter2D() function from OpenCV

Identity = cv2.filter2D(img,-1, kernel1)
#The first argument is the source image
#The second argument is ddepth, which indicates the depth of the resulting image. A value of -1 indicates that the final image will also have t
#The final input argument is the kernel, which we apply to the source image

plt.imshow(Identity)
plt.title('Identity filter')
plt.show()
plt.imshow(img)
plt.title('Original Image')
```

3.1 Blurring

```
# Blurring - smoothes the image out.

blur = cv2.blur(img,(11, 11)) #uniform average. The function automatically created the kernel which elements sum is 1 (for aritmetic operation

gblur = cv2.GaussianBlur(img,(5,5),0) #weighted average. Gaussian blur weights pixel values, based on their distance from the center
#of the kernel. Pixels further from the center have less influence on the weighted average. Applying blurring helps remove some of the high
#frequency edges in the image that we are not concerned with and allow us to obtain a more "clean" segmentation.
#kernel is required as input.

#facciamolo anche sull'imagine in scale di grigio
gblurGray= cv2.GaussianBlur(imgGray,(151,151),0)


titles = ['Original Image','Blurred','Gaussian Blur', 'Gaussian blur Gray']
images = [img, blur, gblur, gblurGray]

for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
```

3.1 Blurring

```
#other algorithm blurring
plt.imshow(img, interpolation="bicubic")
```

```
<matplotlib.image.AxesImage at 0x790cf597eb90>
```

3.1 Blurring

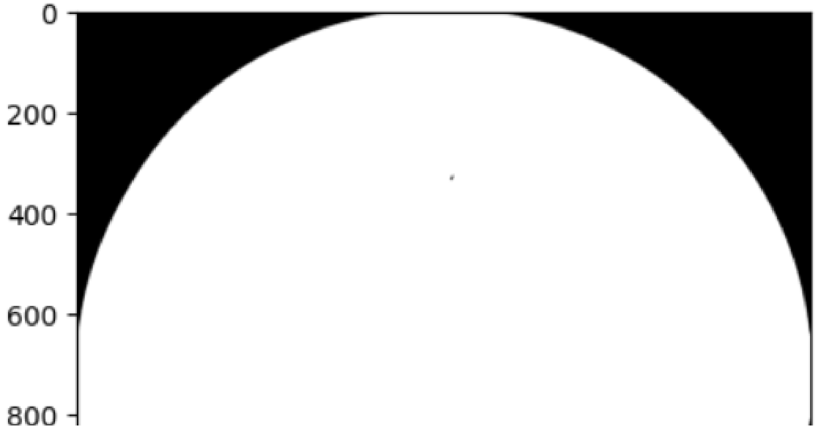
```
#other algorithm for blurring

median = cv2.medianBlur(img, 15) #In median blurring, each pixel in the source image is replaced by the median value of the image pixels in the

plt.imshow(median)
```

```
<matplotlib.image.AxesImage at 0x790cf5a486a0>
```

3.1 Blurring

```
# IMPORTANT NOTE: USUALLY THE BEST BINARIZED IMAGE IS OBTAINED BY FIRST CONVERTING TO GRAYSCALE AND APPLYING BLURRING BEFORE BINARIZATION

# BLURRING GRAYSCALE IMAGE. USE BLURRING MODERATELY, CHECK RESULTS VISUALLY, 11X11 SEEMS GOOD (EXPERIMENTALLY)
gblurGray = cv2.GaussianBlur(imgGray, (11, 11), 0)

T, imgBINgrayBlur = cv2.threshold(gblurGray, 250, 255, cv2.THRESH_BINARY_INV)
plt.imshow(imgBINgrayBlur, 'gray')
```

```
<matplotlib.image.AxesImage at 0x790cfbd25510>
```



3.1 Blurring

```
#Algorithm for MASKING

imgMasked = cv2.bitwise_and(img, img, mask=imgBINgrayBlur)
plt.imshow(imgMasked)
```

<matplotlib.image.AxesImage at 0x790cf5cfb160>

3.2 Sharpening

```
# NOTE: Sum of kernel elements = 1
kernel3 = np.array([[0, -1,  0],
                    [-1,  5, -1],
                    [0, -1,  0]])

# Working on an ROI: Region of Interest to better visualize the output of the operation

imgROI = img[0:300, 0:300]  # ROI

sharp_img = cv2.filter2D(imgROI, ddepth=-1, kernel=kernel3)

plt.imshow(sharp_img)
plt.title('sharp')
plt.show()
plt.imshow(imgROI)
plt.title('original')
```

4.
Morphological Operations

4.0 Other image example

```
#import the fractal plant image and binarize it quickly.
#This particular image can be useful to understand some morphological operations.

!wget "https://digitalreflectionswamf14.files.wordpress.com/2014/09/cropped-ferns.jpg" -O FractalPlant.jpg

imgFRAC = cv2.imread('/content/FractalPlant.jpg')

imgFRACgray = cv2.cvtColor(imgFRAC, cv2.COLOR_BGR2GRAY)  # Convert to grayscale

T, imgFRACgrayBIN = cv2.threshold(imgFRACgray, 110, 255, cv2.THRESH_BINARY)  # Binarize

plt.imshow(imgFRACgrayBIN, 'gray')  # Plot the binarized image
```

4.1 Kernel definition

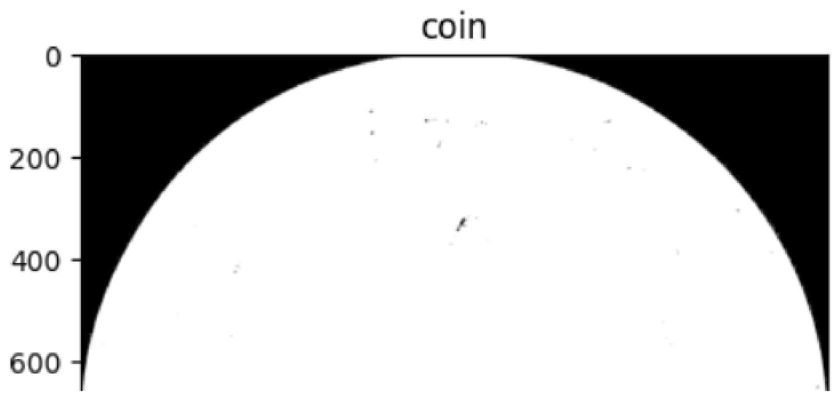
[71]
```
#manually creation of structuring elements with help of Numpy. Usare elementi dispari
kernel = np.ones((5,5),np.uint8)
print(kernel)
```

```
[[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]]
```

```
#In some cases, you may need elliptical/circular shaped kernels. So for this purpose, OpenCV has a function,
#cv2.getStructuringElement(). You just pass the shape and size of the kernel, you get the desired kernel.
K1=cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
K2=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
K3=cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))

print(K1)
print()
print(K2)
print()
print(K3)
```

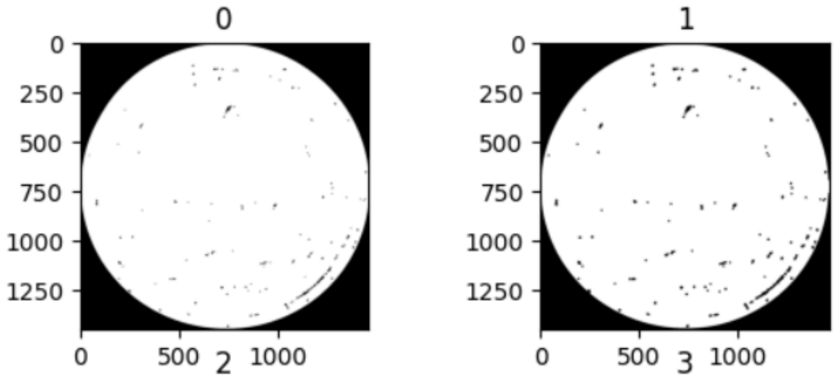
4.2 Erosion

```
#The kernel slides through the image (same as in 2D convolution). A pixel in the original image (either 1 or 0)
#will be considered 1 only if all the pixels overlapped by the kernel is 1, otherwise it is eroded (made to zero).
erosion = cv2.erode(thresh2,kernel,iterations = 0)
plt.imshow(erosion, cmap='gray')
plt.title('coin')
plt.show()

erosion = cv2.erode(imgFRACgrayBIN,kernel,iterations = 3)
plt.imshow(erosion, cmap='gray')
plt.title('fractal plant')
```



4.2 Erosion

```python
# apply LOOP (iterative) erosions
for i in range(4):
    eroded = cv2.erode(thresh2.copy(), kernel, iterations=i + 1)
    plt.subplot(2,2,i+1),plt.imshow(eroded,'gray')
    plt.title(i)

plt.show()

for i in range(4):
    eroded = cv2.erode(imgFRACgrayBIN.copy(), kernel, iterations=i + 1)
    plt.subplot(2,2,i+1),plt.imshow(eroded,'gray')
    plt.title(i)
```

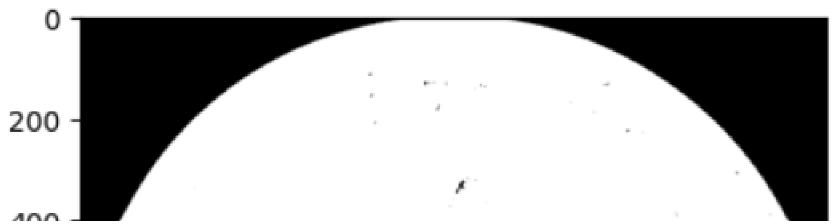


4.3 Dilatation

```
#It is just opposite of erosion. Here, a pixel element (0 or 1) is turned to ‘1’ if at least one pixel under the kernel
# is ‘1’. So it increases the white region in the image or size of foreground object increases
dilation = cv2.dilate(thresh2,kernel,iterations = 1)
plt.imshow(dilation , cmap='gray')
plt.show()

dilation = cv2.dilate(imgFRACgrayBIN,kernel,iterations = 1)
plt.imshow(dilation , cmap='gray')
```

4.4 Opening (erosion followed by dilation)

```
#Normally, in cases like NOISE removal, erosion is followed by dilation. Erosion removes
#white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won’t
#come back, but our object area increases.


opening = cv2.morphologyEx(thresh2, cv2.MORPH_OPEN, kernel)
plt.imshow(opening , cmap='gray')

plt.show()

opening = cv2.morphologyEx(imgFRACgrayBIN, cv2.MORPH_OPEN, kernel)
plt.imshow(opening , cmap='gray')
```



4.5 Closing (dilation followed by erosion)

```python
#It is useful in closing small holes inside the foreground objects, or small black points on the object
closing = cv2.morphologyEx(thresh2, cv2.MORPH_CLOSE, kernel)
plt.imshow(closing , cmap='gray')
plt.show()

closing = cv2.morphologyEx(imgFRACgrayBIN, cv2.MORPH_CLOSE, kernel)
plt.imshow(closing , cmap='gray')
```

4.6 Top-hat

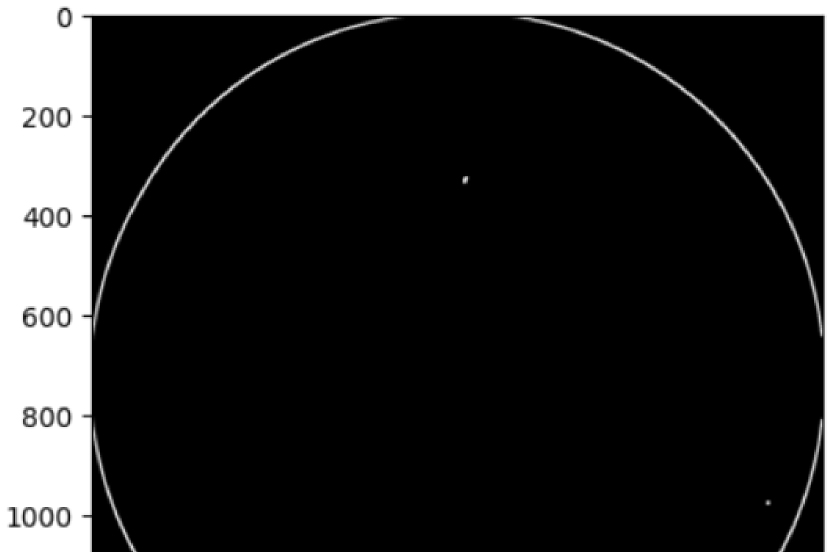
```
#It is the difference between input image and Opening of the image
tophat = cv2.morphologyEx(thresh2, cv2.MORPH_TOPHAT, kernel)
plt.imshow(tophat , cmap='gray')
plt.show()


tophat = cv2.morphologyEx(imgFRACgrayBIN, cv2.MORPH_TOPHAT, kernel)
plt.imshow(tophat , cmap='gray')
```

4.7 Bottom-hat (Black hat in Python)

```
#It is the difference between Closing of the image and the input image
bothat = cv2.morphologyEx(thresh2, cv2.MORPH_BLACKHAT, kernel)
plt.imshow(bothat , cmap='gray')
plt.show()

bothat = cv2.morphologyEx(imgFRACgrayBIN, cv2.MORPH_BLACKHAT, kernel)
plt.imshow(bothat , cmap='gray')
```

4.8 Gradient and convolution

```
#It is the difference between dilation and erosion of an image. The result will
#look like the outline of the object.
gradient = cv2.morphologyEx(imgBINgrayBlur, cv2.MORPH_GRADIENT, kernel)
plt.imshow(gradient, cmap='gray')
plt.show()

gradient = cv2.morphologyEx(imgFRACgrayBIN, cv2.MORPH_GRADIENT, kernel)
plt.imshow(gradient , cmap='gray')
```



4.9 Skeletonizing or Medial Axis Transform

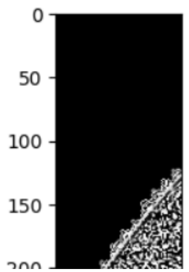
```
#Skeletonization is a process for reducing foreground regions in a binary image to a skeletal
#that largely preserves the extent and connectivity of the original region while throwing away most of
#the original foreground pixels. A way to think about the skeleton is as the loci of centers of bi-tangent circles
#that fit entirely within the foreground region being considered.

#plotto ROI dell'originale
plt.imshow(imgFRACgrayBIN[200:300,400:600], cmap=plt.cm.gray),plt.title('Original')
plt.show()

#method 1
from skimage.morphology import medial_axis, skeletonize #importo pacchetto

# Compute the medial axis (skeleton)
skel1,distance = medial_axis(imgFRACgrayBIN, return_distance=True)
# Distance to the background for pixels of the skeleton
dist_on_skel = distance * skel1
#confrontiamo i risultati rispetto ad una ROI
plt.imshow(dist_on_skel[200:300,400:600], cmap='gray'),plt.title('Method 1: notare che è in scala di grigio per rappresentare lo spessore ')
plt.show()
```

4.9 Skeletonizing or Medial Axis Transform

```
#method 2

#I NEED AN IMAGE WITH ONLY 0 AND 1, NOT 0 AND 255

righe = len(imgFRAC[:, 0, 1])  # number of rows
colonne = len(imgFRAC[0, :, 1])  # number of columns

img01 = np.zeros((righe, colonne))  # create a matrix of zeros
for riga in range(righe):
    for col in range(colonne):
        if imgFRACgrayBIN[riga, col] > 0:
            img01[riga, col] = 1  # set to 1 if the pixel is greater than 0

from skimage.morphology import skeletonize as sk  # import package

# Compute the skeleton
skel2 = sk(img01)

# Plot the original image (cropped part)
plt.imshow(imgFRACgrayBIN[200:300, 400:600], 'gray'), plt.title('Original')
plt.show()

# Plot the skeleton (cropped part)
plt.imshow(skel2[200:300, 400:600], cmap=plt.cm.gray), plt.title('Method 2: note that it is binary')
```

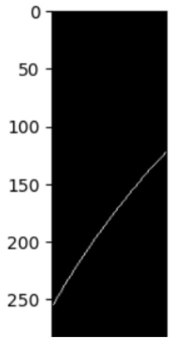
4.10 Others

```
plt.imshow(filt.laplace(imgGray[100:500,100:200]), cmap='gray') #edges grigi
plt.show()
```



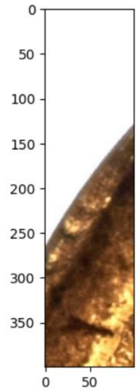

```
plt.imshow(filt.laplace(imgBINgrayBlur[100:500,100:200]), cmap='gray') #edges binary
```

<matplotlib.image.AxesImage at 0x790cf4d34be0>




```
plt.imshow(img[100:500,100:200])
```

<matplotlib.image.AxesImage at 0x790cf4eacf10>



4.10 Others

```
#ALTERNATIVE CONTOUR OpenCV function

#the findContours() function has three required arguments

#image: The binary input image obtained in the previous step.
#mode: This is the contour-retrieval mode. e.g. RETR_TREE means the algorithm will retrieve all possible contours from the binary image. More co
#method: This defines the contour-approximation method. In this example, we will use CHAIN_APPROX_NONE.Though slightly slower than CHAIN_APPROX_

#More info about countorning here: https://learnopencv.com/contour-detection-using-opencv-python-c/

#get contours
contours, hierarchy = cv2.findContours(imgBINgrayBlur,cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

#plot contours SUPERIMPOSED on the original
image_copy = img.copy()
cv2.drawContours(image=image_copy, contours=contours, contourIdx=-1, color=(0, 255, 0), thickness=2, lineType=cv2.LINE_AA)

plt.imshow(image_copy)
```

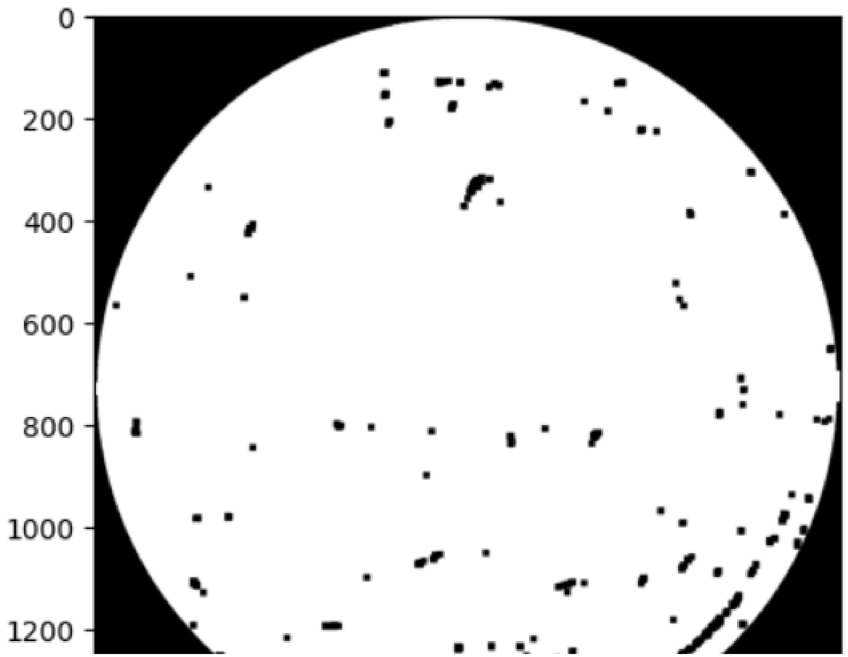
```
<matplotlib.image.AxesImage at 0x790cf5215e10>
```

5.
Blob Operations

5.1 Example 1

```
# I create a series of blobs inside the coin by performing erosion

eroded = cv2.erode(thresh2.copy(), kernel, iterations=3)

plt.imshow(eroded, 'gray')
```

```
<matplotlib.image.AxesImage at 0x790cf4459cc0>
```



5.1 Example 1

```python
# Set up the SimpleBlobDetector with default parameters.
params = cv2.SimpleBlobDetector_Params()

# Filter by Area.
params.filterByArea = 1
params.minArea = 100
params.maxArea = 300  # Specify max area because the default is not infinite

# Filter by Circularity
params.filterByCircularity = 0
params.minCircularity = 0.9
params.maxCircularity = 1

# Filter by Convexity
params.filterByConvexity = 0
params.minConvexity = 0.1  # Set a small positive value
params.maxConvexity = 1

# Filter by Inertia
params.filterByInertia = 0
params.minInertiaRatio = 0.9
params.maxInertiaRatio = 1

# Create the detector
detector = cv2.SimpleBlobDetector_create(params)
```

5.1 Example 1

```
#Detect
keypoints = detector.detect(eroded)

#Plot
keypoints
```

```
(< cv2.KeyPoint 0x790cf4424ba0>,
 < cv2.KeyPoint 0x790cf459abe0>,
 < cv2.KeyPoint 0x790cf459ae50>,
 < cv2.KeyPoint 0x790cf4f22b20>,
 < cv2.KeyPoint 0x790cf5060b40>,
 < cv2.KeyPoint 0x790cf55dc0f0>,
 < cv2.KeyPoint 0x790cf55dd260>,
 < cv2.KeyPoint 0x790cf55dcc30>,
```

5.1 Example 1

```
im_with_keypoints = cv2.drawKeypoints(eroded, keypoints, np.array([]), (255,0,0), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.figure(figsize = (15,15))
plt.imshow(im_with_keypoints)

plt.title('Blob with an area between 10 and 100 pixels highlighted. Note that the larger ones were not highlighted')
```

5.2 Example 2

```
# Construct the correct download URL
url = "https://drive.google.com/uc?id=1ZSEsD3Nz4C-vE3zVXsnhk5ZZg99Td9ez"

# Download the file
gdown.download(url, 'prova.png', quiet=False)

# Read the image
immagine = plt.imread('/content/prova.png')

# Apply threshold
T, BW = cv2.threshold(immagine, 0, 255, cv2.THRESH_BINARY)
I = BW.astype(np.uint8)

# Show the result
plt.imshow(I, cmap='gray')
plt.title('Sample Image: Geometric Shapes')
plt.show()
```

5.2 Example 2

```python
import cv2

# Set up the SimpleBlobdetector with default parameters.
params = cv2.SimpleBlobDetector_Params()

# on/off for the following params, 1 means I activate the control and need to define minimum and maximum parameters, 0 means I turn it off.

# Filter by Area.
params.filterByArea = 1
params.minArea = 5000
params.maxArea = 10000  # specify the max, because by default it's not infinite

# Filter by Circularity
params.filterByCircularity = 0
params.minCircularity = 0.9
params.maxCircularity = 1

# Filter by Convexity
params.filterByConvexity = 1
params.minConvexity = 0.9
params.maxConvexity = 1

# Filter by Inertia
params.filterByInertia = 0
params.minInertiaRatio = 0.9
params.maxInertiaRatio = 1

# Create the detector with the parameters set above
detector = cv2.SimpleBlobDetector_create(params)
```

Interactive Colab file

- https://colab.research.google.com/drive/12xXpXw49qFtrtnkwr1F7a-FxEMghVjyj#scrollTo=Lz8pGt1SRj8R