laboratorio 1

August 28, 2019

1 Procesamiento y análisis de imágenes

1.1 Bienvenid@s al primer laboratorio

En este laboratorio se verá * Punto ciego * Obtener negativo de una imagen * Obtener imagen en escala de grises * Calcular histograma y acumulación de una imagen * Obtener una imagen binaria dado un umbral * Aplicar métodos para calcular el umbral

1.1.1 Requisitos

- Python 3.6+ https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-python.html
- scikit-image https://scikit-image.org/docs/dev/install.html

Ejercicio 1: Leer, modificar y guardar una imagen

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, data
```

[7]:
```
img = io.imread("Imagenes/astronaut.png")
io.imshow(img)
plt.axis('off')
plt.show()
```

Algunas operaciones básicas sobre matrices, que pueden ser útiles

[8]:
```
np.max(img)
```

[8]:
```
255
```

[9]:
```
img.shape
```

[9]:
```
(512, 512, 3)
```

Modificando la imagen original, agregando filas negras y columnas blancas

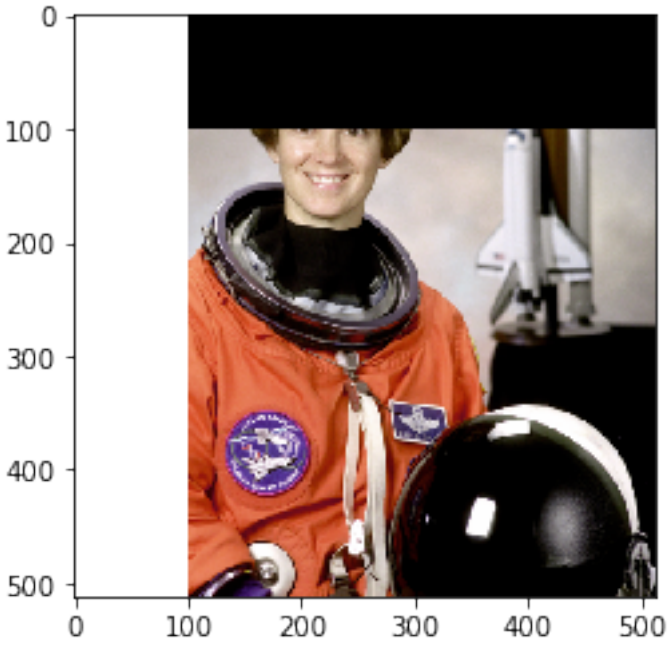
[10]:
```
np.zeros(3)
```

[10]:
```
array([0., 0., 0.])
```

[11]:
```
img[0:100]  = np.zeros(3)
```

[12]:
```
img[:,0:100] = 255* np.ones(3)
```

[13]:
```
plt.imshow(img)
plt.show()
```



[14]:
```
io.imsave("output/astronaut_changes.png", img)
```

Aquí tienen como tarea encapsular todo en funciones

Ejercicio 2: Leer una imagen a color, convertir a tonos de gris y guardar la nueva imagen
Según teoría tenemos la siguiente fórmula

[15]:
```
def rgb2gray(rgb):

    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.299 * r + 0.5870 * g + 0.1140 * b
    return gray
```

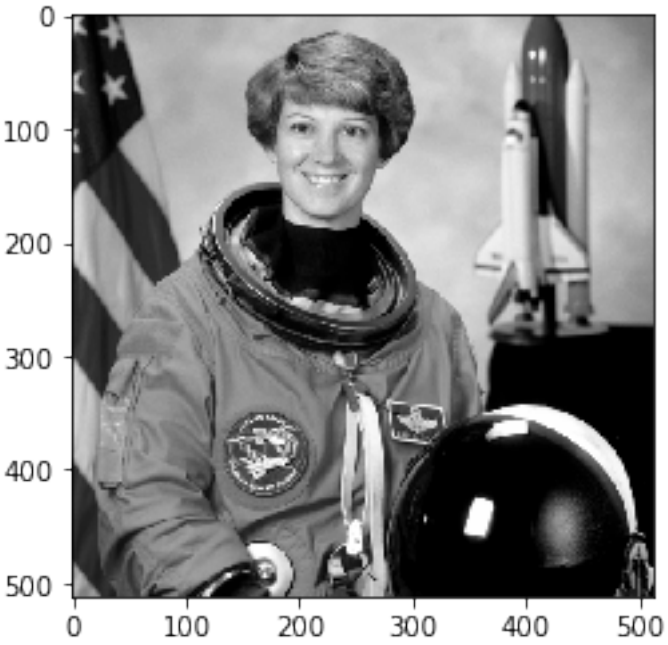
[16]:
```
img = io.imread("Imagenes/astronaut.png")
gray = rgb2gray(img)
np.max(gray)
```

[16]:
```
255.0
```

[17]:
```
gray.shape
```

[17]:
```
(512, 512)
```

[18]:
```
plt.imshow(gray, cmap='gray')
plt.show()
```



[19]:
```
gray = np.array(gray, dtype=np.uint8)
```

[20]:
```
io.imsave("output/astronaut_gray.png", gray)
```

Resumiendo en funciones:

[21]:
```
def get_grayscale(name_image, extension):
    img = io.imread("Imagenes/{}.{}".format(name_image, extension))
    gray = rgb2gray(img)
    gray = np.array(gray, dtype=np.uint8)
    io.imsave("output/{}_grayscale.{}".format(name_image, extension), gray)
```

[22]:
```
get_grayscale("astronaut", "png")
```

Ejercicio 3: Leer una imagen en escala de gris, convertir al negativo y guardar la nueva imagen

[23]:
```
img = io.imread("Imagenes/autopista.tif")
```

[24]:
```
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.show()
```

[25]:
```
negative = 255 - img
plt.imshow(negative, cmap='gray')
plt.axis('off')
plt.show()
```

[26]:
```
def get_negative(name_image, extension):
    img = io.imread("Imagenes/{}.{}".format(name_image, extension))
    M = 1
```

```
    if (np.max(img) > 1):
        M = 255
    negative = M - img
    negative = np.array(negative, dtype=np.uint8)
    io.imsave("output/{}_negative.{}".format(name_image, extension), negative)
```

[27]:
```
get_negative("autopista", "tif")
```

Ejercicio 4: Dada una imagen en tonos de gris, calcular el histograma y la acumulación

[35]:
```
img = data.camera()
```

[36]:
```
plt.imshow(img, cmap="gray")
plt.axis('off')
plt.show()
```

[37]:
```
img.shape
```

[37]:
```
(512, 512)
```

[38]:
```
histo = np.zeros((256,2))
for i in range(0, 256):
    histo[i,0] = i
    histo[i,1] = np.count_nonzero(img == i)
np.sum(histo[:,1])
```

[38]:
```
262144.0
```

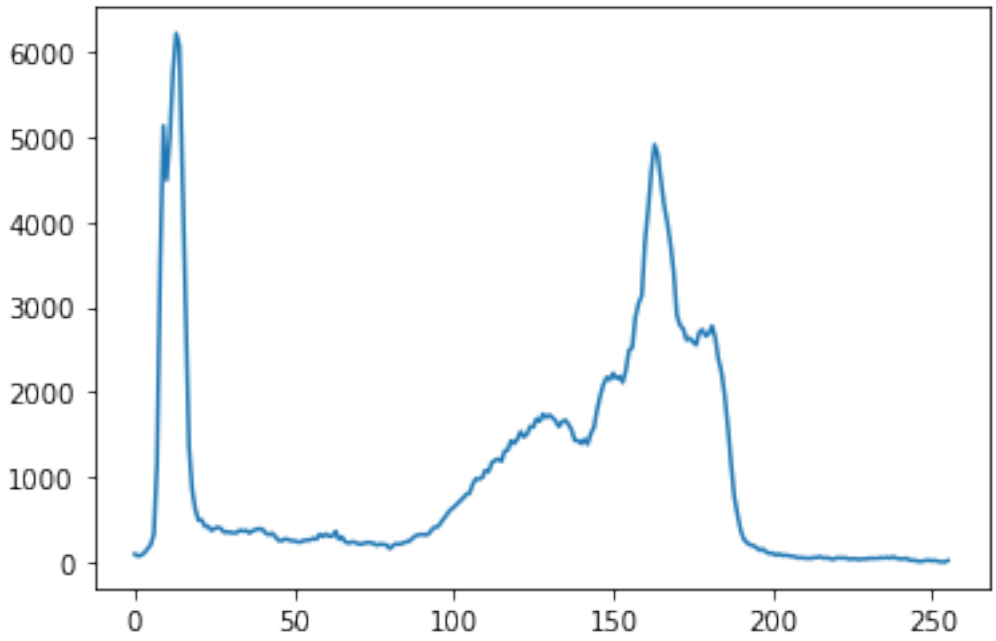
[39]:
```
512*512
```

[39]:
```
262144
```

[40]:
```
img.shape
```

[40]:
```
(512, 512)
```

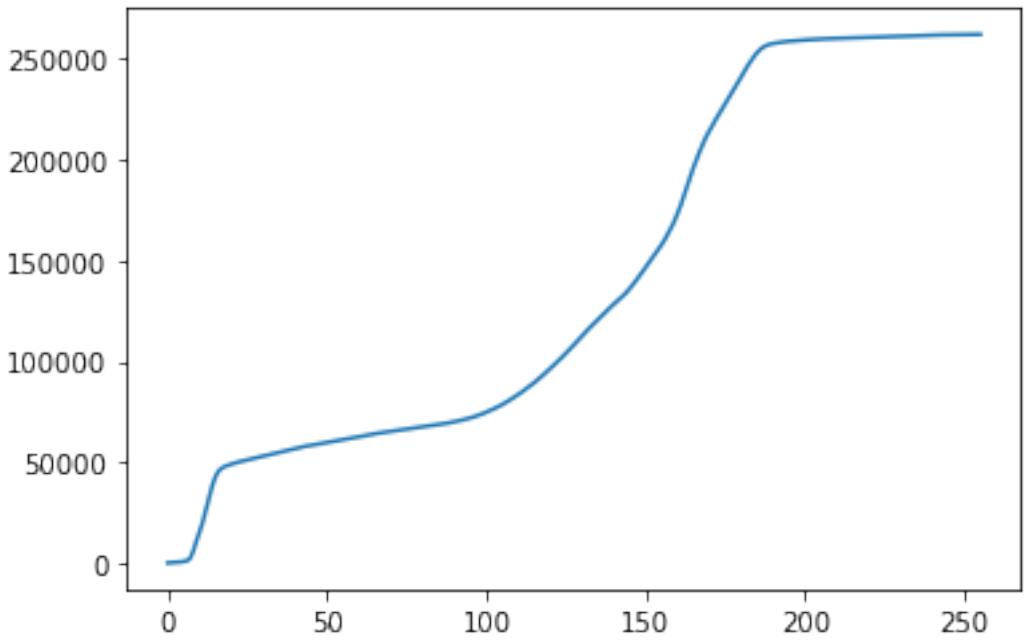
[41]:
```
plt.plot(histo[:,0], histo[:,1])
plt.show()
```



[42]:
```
acc = np.zeros((256,2))
ain = 0
for i in range(0, 256):
    acc[i,0] = i
    acc[i,1] = ain + np.count_nonzero(img == i)
    ain = acc[i,1]
acc[255,1]
```

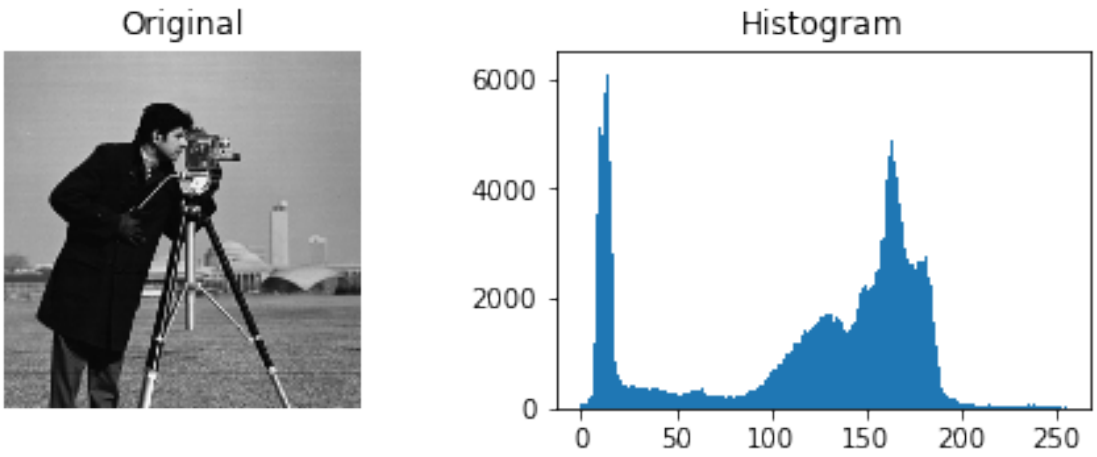
[42]:
```
262144.0
```

[43]:
```
plt.plot(acc[:,0], acc[:,1])
plt.show()
```



[48]:
```
def plot_histogram(image):
    fig, axes = plt.subplots(1, 2, figsize=(8, 2.5))
    ax = axes.ravel()

    ax[0] = plt.subplot(1, 2, 1)
    ax[1] = plt.subplot(1, 2, 2)

    ax[0].imshow(image, cmap=plt.cm.gray)
    ax[0].set_title("Original")
    ax[0].axis('off')

    ax[1].hist(image.ravel(), bins=256)
    ax[1].set_title('Histogram')
    plt.show()
```

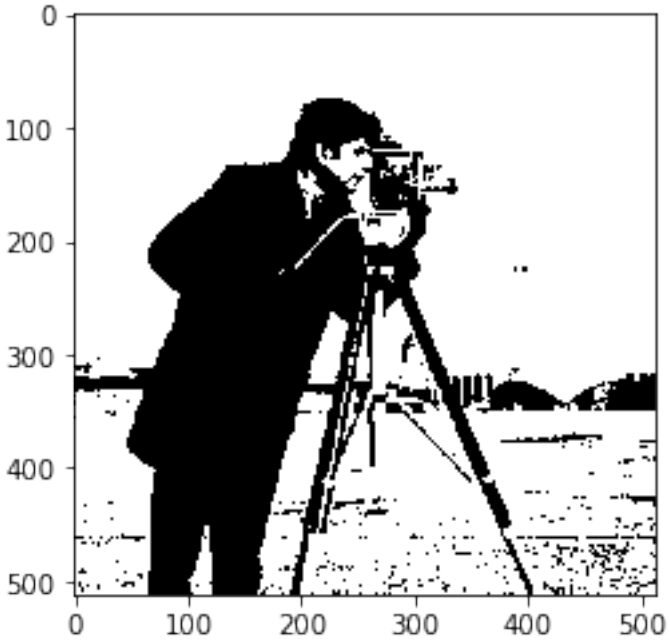
[49]:
```
plot_histogram(img)
```



Ejercicio 5: Dada una imagen en tonos de gris y un valor umbral, devolver una imagen binaria

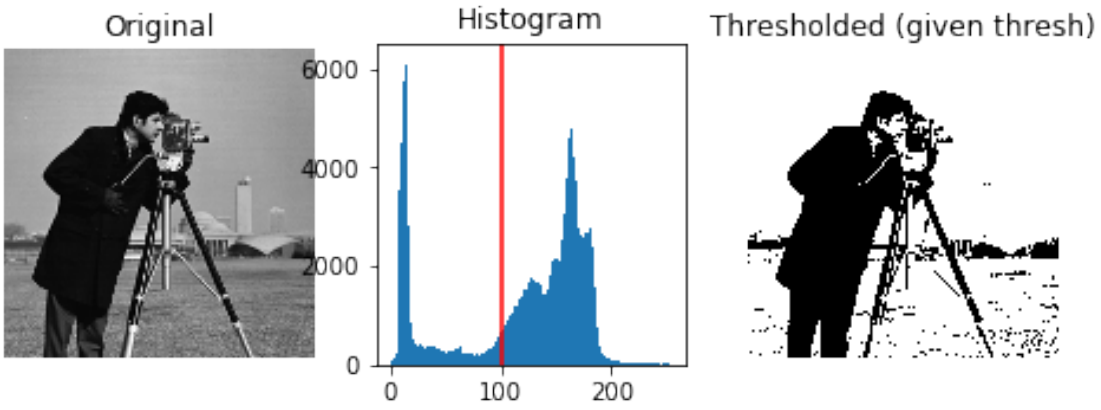
[50]:
```
img = data.camera()
```

[51]:
```
thresh = 100
binary = img > thresh
plt.imshow(binary, cmap=plt.cm.gray)
plt.show()
```

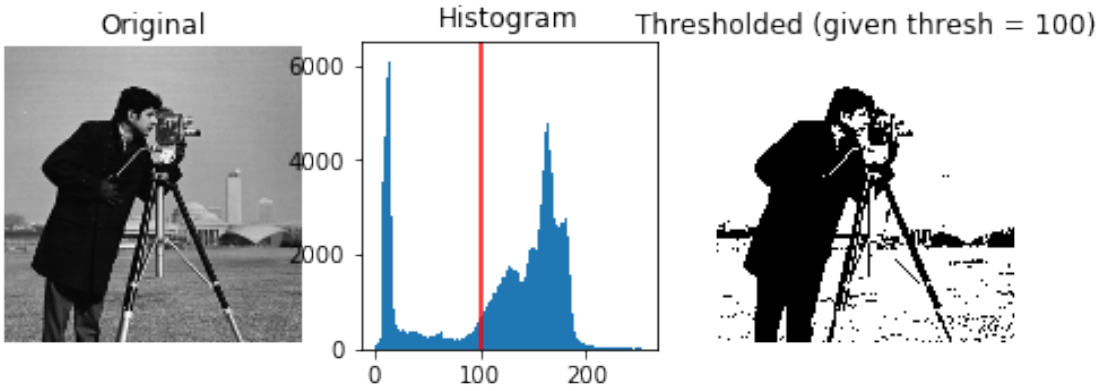


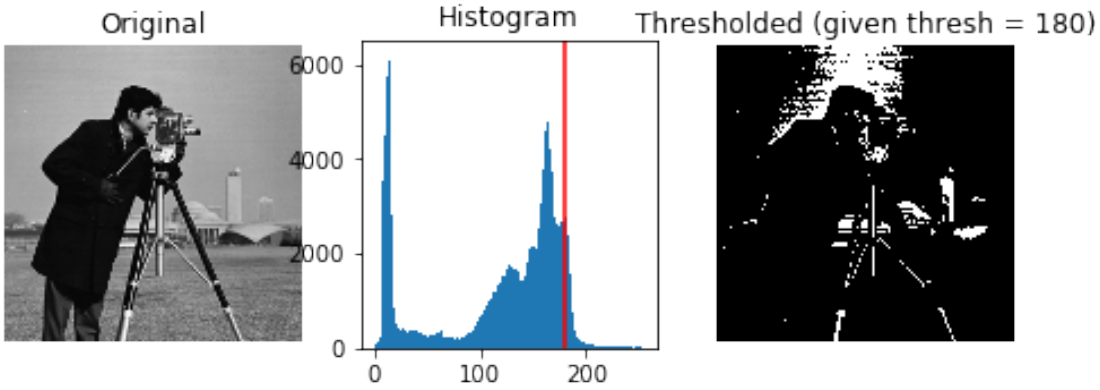
[66]:
```
def plot_histogram(im1, im2, thresh, name):
    fig, axes = plt.subplots(1, 3, figsize=(8, 2.5))
    ax = axes.ravel()

    ax[0] = plt.subplot(1, 3, 1)
    ax[1] = plt.subplot(1, 3, 2)
    ax[2] = plt.subplot(1, 3, 3, sharex=ax[0], sharey=ax[0])

    ax[0].imshow(im1, cmap=plt.cm.gray)
    ax[0].set_title("Original")
    ax[0].axis('off')

    ax[1].hist(im1.ravel(), bins=256)
    ax[1].set_title('Histogram')
    ax[1].axvline(thresh, color='r')

    ax[2].imshow(im2, cmap=plt.cm.gray)
    ax[2].set_title('Thresholded (' + name + ')')
    ax[2].axis('off')
    plt.show()
```

[67]:
```
thresh = 100
binary = img > thresh
plot_histogram(img, binary, thresh, "given thresh")
```



[68]:
```
def get_binary(image, thresh):
    binary = image > thresh
    plot_histogram(image, binary, thresh, "given thresh = {}".format(thresh))
get_binary(img, 100)
get_binary(img, 180)
```





Ejercicio 6: Dada una imagen en tonos de gris, aplicar método Otsu para calcular umbral y binarizar

[69]:
```
img = data.camera()
```

[70]:
```
np.histogram(img, np.arange(0,257))
```

[70]:
```
(array([ 102,   76,   89,  114,  159,  209,  335, 1173, 3523, 5129, 4490,
        4980, 5762, 6212, 6067, 4480, 2805, 1375,  860,  625,  498,  503,
         426,  430,  379,  398,  418,  402,  358,  362,  354,  349,  355,
         381,  371,  379,  348,  375,  390,  398,  394,  349,  327,  345,
         311,  263,  256,  279,  274,  258,  258,  245,  242,  261,  265,
         268,  285,  276,  334,  308,  337,  309,  313,  370,  276,  301,
         245,  230,  244,  242,  222,  217,  230,  238,  235,  224,  205,
         220,  213,  208,  164,  203,  222,  216,  230,  243,  251,  281,
         315,  328,  334,  327,  333,  381,  417,  420,  473,  525,  573,
         620,  650,  687,  727,  760,  803,  816,  924,  986,  981, 1002,
        1085, 1070, 1174, 1202, 1219, 1190, 1314, 1321, 1432, 1399, 1453,
        1539, 1477, 1509, 1598, 1592, 1691, 1659, 1745, 1705, 1734, 1702,
```

```
        1663, 1598, 1655, 1673, 1626, 1556, 1435, 1435, 1398, 1449, 1391,
        1514, 1590, 1800, 1964, 2096, 2176, 2147, 2225, 2163, 2185, 2117,
        2258, 2494, 2524, 2885, 3045, 3134, 3767, 4147, 4596, 4906, 4790,
        4497, 4195, 3993, 3737, 3415, 2906, 2785, 2746, 2615, 2635, 2596,
        2555, 2702, 2737, 2662, 2686, 2778, 2646, 2392, 2236, 1961, 1595,
        1162,  776,  565,  381,  275,  235,  209,  200,  176,  146,  160,
         126,  109,  113,   86,  103,   84,   90,   81,   75,   72,   56,
          60,   54,   53,   51,   60,   62,   70,   53,   58,   51,   38,
          57,   58,   55,   58,   40,   48,   46,   37,   45,   42,   51,
          52,   51,   52,   60,   55,   63,   53,   69,   49,   48,   44,
          54,   28,   28,   24,   14,   17,   27,   29,   22,   26,   16,
          10,   10,   26]),
 array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
         13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
         26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
         39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
         52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
         65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
         78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
         91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
        104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
        117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
        130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
        143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
        156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,
        169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181,
        182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194,
        195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
        208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
        221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233,
        234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246,
        247, 248, 249, 250, 251, 252, 253, 254, 255, 256]))
```

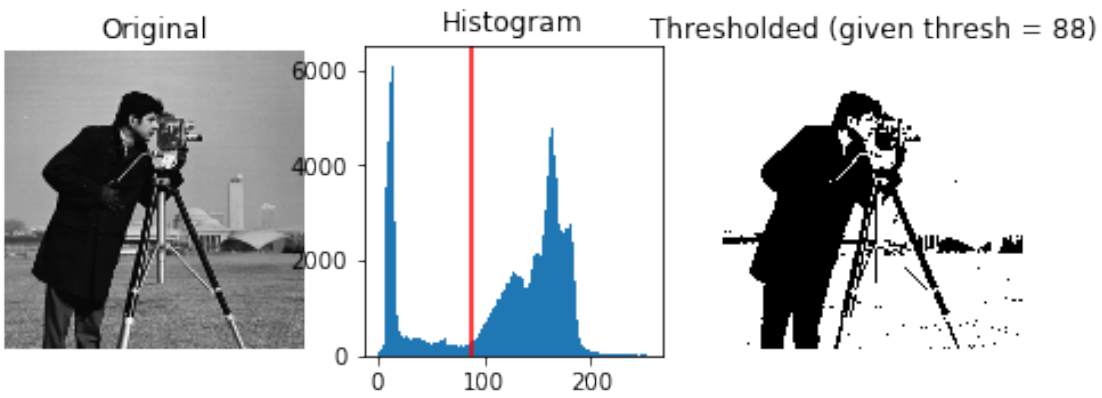
[71]:
```python
def otsu(gray):
    pixel_number = gray.shape[0] * gray.shape[1]
    mean_weigth = 1.0/pixel_number
    his, bins = np.histogram(gray, np.arange(0,257))
    final_thresh = -1
    final_value = -1
    intensity_arr = np.arange(256)
    mu = np.sum(intensity_arr * his)
    # print(mu)
    # print(mean_weigth)
    for t in bins[1:-1]: # This goes from 1 to 254 uint8 range (Pretty sure␣
↪wont be those values)
        W0 = np.sum(his[:t])* mean_weigth
        W1 = 1 - W0
```

```
        mut = np.sum(intensity_arr[:t]*his[:t])
        mu0 = mut / W0
        mu1 = (mu - mut) /W1
        #print mub, muf
        value = W0 * W1 * (mu0 - mu1) ** 2
        # print("W0", W0, "W1", W1)
        # print("t", t, "value", value)

        if value > final_value:
            final_thresh = t
            final_value = value
    return final_thresh
```

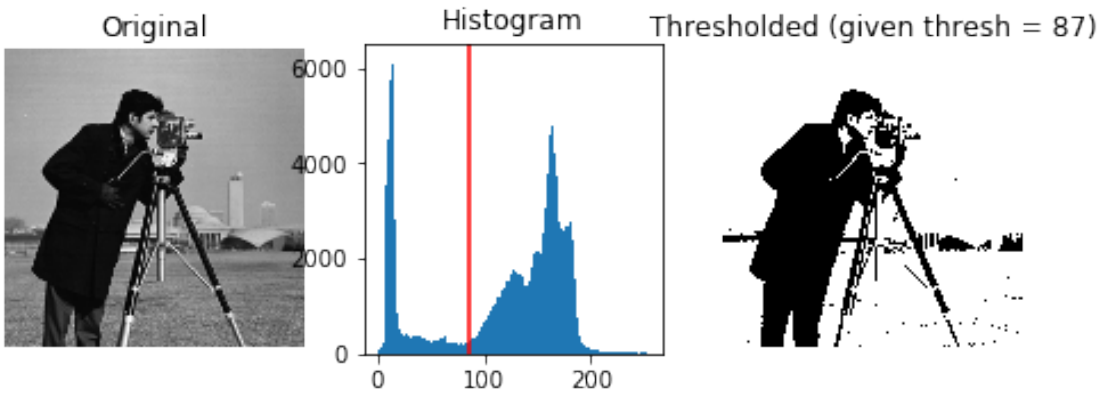
[72]:
```
otsu(img)
```

[72]: 88

[73]:
```
get_binary(img, 88)
```



[74]:
```
from skimage.filters import threshold_otsu
thresh = threshold_otsu(img)
get_binary(img, thresh)
```



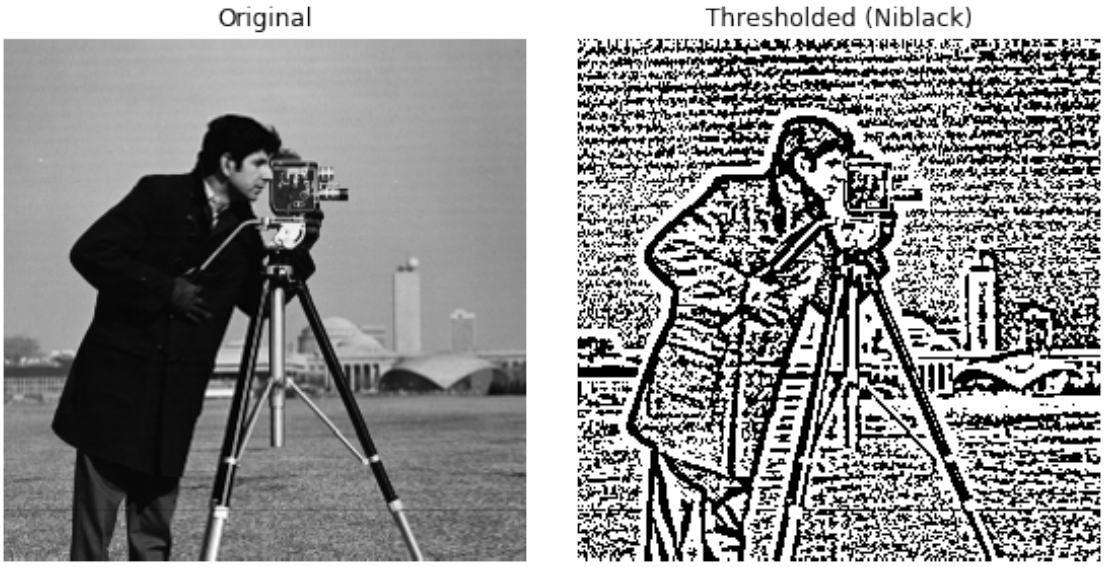
Ejercicio 7: Dada una imagen en tonos de gris, aplicar métodos basado en entropía, Isodata, local basado en promedio, Niblack, Sauvola para calcular umbral y binarizar

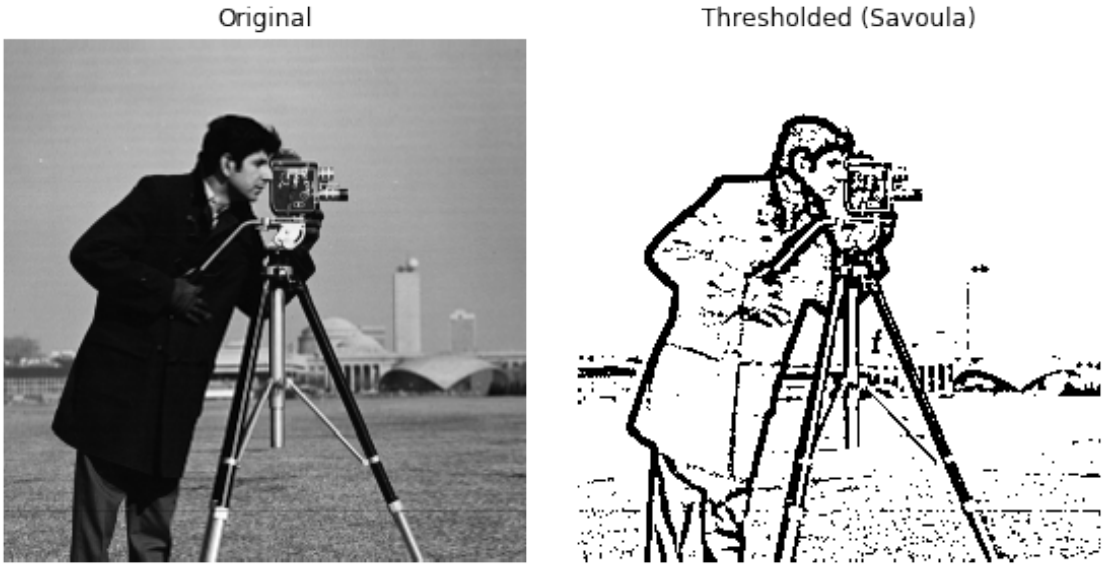
[75]:
```
from skimage.filters import threshold_otsu, try_all_threshold,␣
↪threshold_niblack, threshold_sauvola
```

[76]:
```
def get_treshold_adaptative(image, function, name):
    thresh = function(image)
    binary = image > thresh
    plot_2_images(image, binary, "Original", 'Thresholded (' + name + ')')
```

[77]:
```
def plot_2_images(im1, im2, name1, name2):
    fig, axes = plt.subplots(1, 2, figsize=(8, 4))
    ax = axes.ravel()

    ax[0].imshow(im1, cmap=plt.cm.gray)
    ax[0].set_title(name1)
    ax[0].axis('off')
    ax[1].imshow(im2, cmap=plt.cm.gray)
    ax[1].set_title(name2)
    ax[1].axis('off')

    fig.tight_layout()
    plt.show()
```

[78]:
```
get_treshold_adaptative(img, threshold_niblack, "Niblack")
get_treshold_adaptative(img, threshold_sauvola, "Savoula")
```





[79]:
```
try_all_threshold(img,  figsize=(10, 10), verbose=False)
plt.show()
```

Original

Isodata

Li

Mean

Minimum

Otsu

Triangle

Yen

Hay mucho que mejorar en estos códigos, la idea es que comiencen a jugar con las imágenes y así tener más experiencia, cualquier cosa al correo natalia.perez.g@usach.cl. ąNos vemos la próxima clase!