FACULTAD DE INGENIERÍA
UNIVERSIDAD DE BUENOS AIRES

67.61 - FUNDAMENTOS MATEMÁTICOS DE LA VISIÓN ROBÓTICA

Lo importante es el balance

CAMILA SERRA (97422)

MARIANO EZEQUIEL WILLINER (83469)

17 de mayo de 2021

Introducción

El objetivo de este trabajo es encontrar el umbral con búsqueda binaria según lo indicado y comparar el resultado con la binarización por el método de Otsu. Además, programar la implementación del método de binariazación local por Bernsen.

1. Implementar 1: Búsqueda manual del umbral

[1]:
```
%matplotlib inline

import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
```

[2]:
```
def get_sub_array(array, indexes):
    sub_array = []
    for index in indexes:
        sub_array.append(array[index])

    return sub_array
```

[3]:
```
def divide_image(image, threshold):
    image_as_array = image.ravel()
    lower_values = []
    upper_values = []

    for value in image_as_array:
        if (value < threshold):
            lower_values.append(value)
        else:
            upper_values.append(value)

    return lower_values,upper_values
```

[4]:
```
#Paso1: Definir umbral inicial (en general la media de la imagen)
def find_threshold(image, threshold = 128, delta = 1.0):

    #Paso2: Dividir la imagen en dos partes
    lower_values,upper_values = divide_image(image, threshold)

    #Paso3: Encontrar la media de cada parte
```

```
    lower_mean = np.mean(lower_values)
    upper_mean = np.mean(upper_values)

    #Paso4: Calcular el nuevo umbral (promedio entre media anterior y␣
 ↪actual)
    new_threshold = np.mean([lower_mean, upper_mean])

    #Paso5: Criterio de detención (o recalculo)
    if abs(new_threshold - threshold) < delta:
        return new_threshold
    else:
        return find_threshold(image, new_threshold, delta)
```

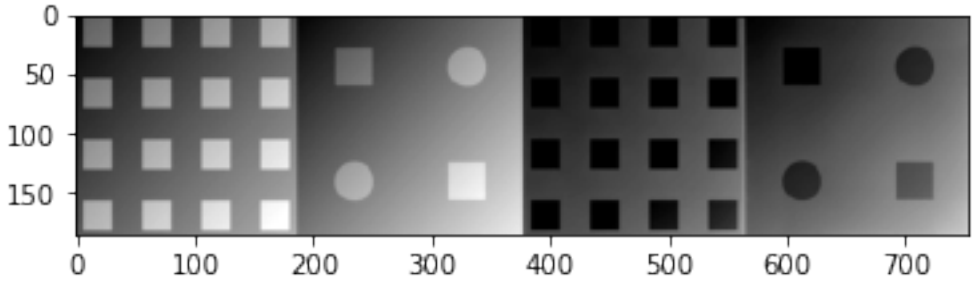
[5]:
```
original_image = cv.imread("Sombreado.png", cv.IMREAD_GRAYSCALE)

mean = np.mean(original_image)
manual_threshold = find_threshold(original_image, mean)
```

[6]:
```
fixed_binarized_image = cv.threshold(original_image,
                                     manual_threshold, 255,
                                     cv.THRESH_BINARY)[1]
```

[7]:
```
otsu_binarized_image = cv.threshold(original_image,
                                    manual_threshold, 255,
                                    cv.THRESH_BINARY + cv.THRESH_OTSU)[1]
```

[8]:
```
print("Original")
plt.imshow(original_image, cmap='gray', vmin=0, vmax=255)
plt.show()
```

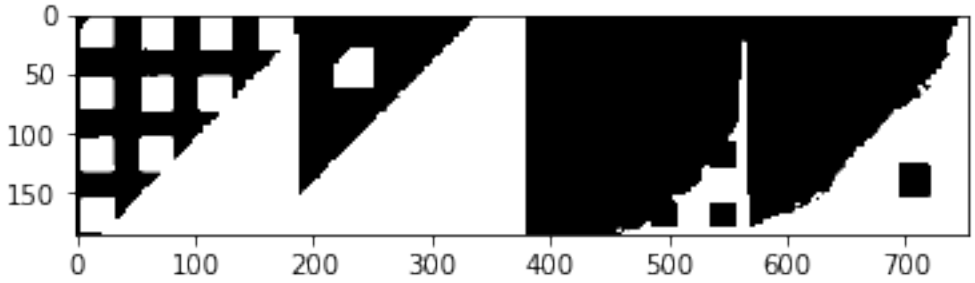
```
Original
```



[9]:
```
print("Binarized - Fixed")
plt.imshow(fixed_binarized_image, cmap='gray', vmin=0, vmax=1)
```

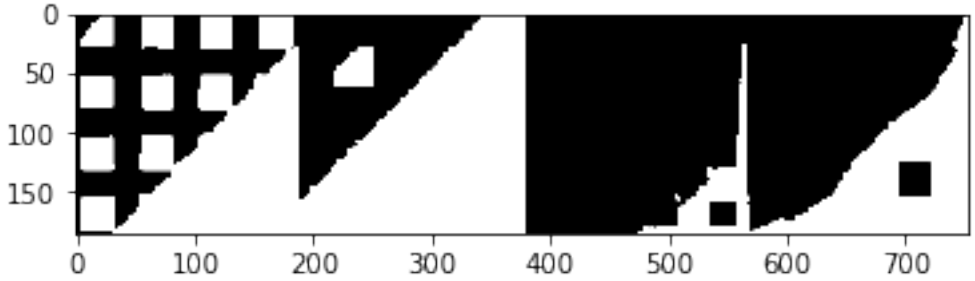
```
plt.show()
```

```
Binarized - Fixed
```



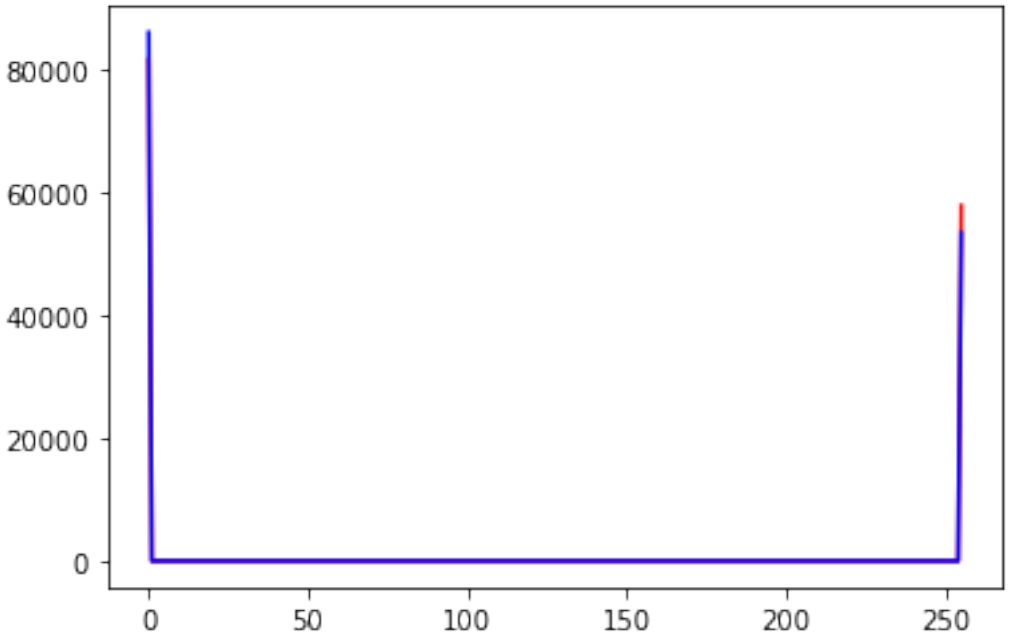
[10]:
```
print("Binarized - Otsu")
plt.imshow(otsu_binarized_image, cmap='gray', vmin=0, vmax=1)
plt.show()
```

```
Binarized - Otsu
```



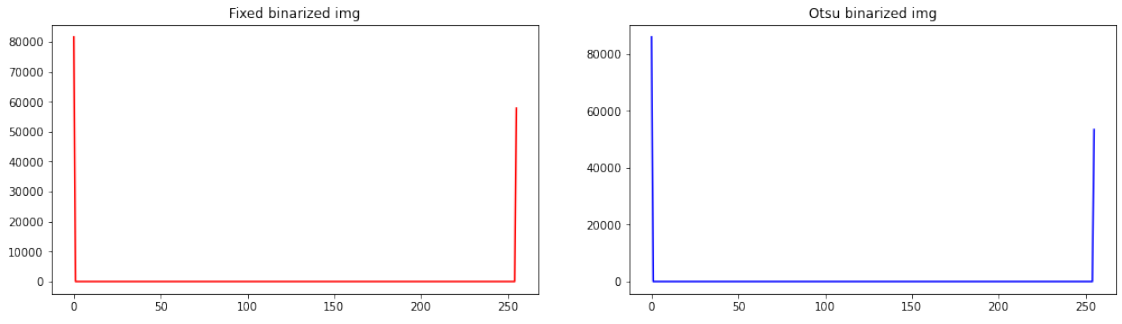
[11]:
```
fig = plt.figure()

hist1,bins1 = np.histogram(fixed_binarized_image.ravel(),256,[0,256])
plt.plot(hist1, 'r')

hist2,bins2 = np.histogram(otsu_binarized_image.ravel(),256,[0,256])
plt.plot(hist2, 'b')
```

[11]:
```
[<matplotlib.lines.Line2D at 0x7f4eae0e13d0>]
```



[12]:
```
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_size_inches(15, 5)
fig.tight_layout(pad=5.0)

ax1.set_title('Fixed binarized img')
ax1.plot(hist1, 'r')

ax2.set_title('Otsu binarized img')
ax2.plot(hist2, 'b')
```

[12]: [<matplotlib.lines.Line2D at 0x7f4eae012700>]



El resultador de la binarización fija fue similar al método de Otsu, y como se puede ver, este tipo de binarización no es adecuado para este tipo de imágenes.

2. Implementar 2: Binarización local, método Bernsen

Implementar para una ventana de 3x3, 5x5 o 7x7.

Contraste: Máximo-Mínimo

gris_medio: Media

[1]:
```
%matplotlib inline

import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
```

[2]:
```
def divide_in_tiles(image, tile_size):
    tiles = []
    image_height, image_width = image.shape
    rows = np.ceil(image_height / tile_size).astype(int)
    cols = np.ceil(image_width/ tile_size).astype(int)

    for i in range(rows):
        top = i * tile_size
        bottom = min(top + tile_size - 1, image_height - 1)
        for j in range(cols):
            left = j * tile_size
            right = min(left + tile_size - 1, image_width - 1)
            tiles.append(image[top:bottom, left:right])

    return tiles, rows, cols
```

[3]:
```
def generate_from_tiles(tiles, rows, cols):
    image_rows = []
    for i in range(rows):
        image_row = tiles[i * cols]
        for j in range(1,cols):
            image_row = np.hstack((image_row, tiles[(i * cols) + j]))
        image_rows.append(image_row)

    return np.vstack(image_rows)
```

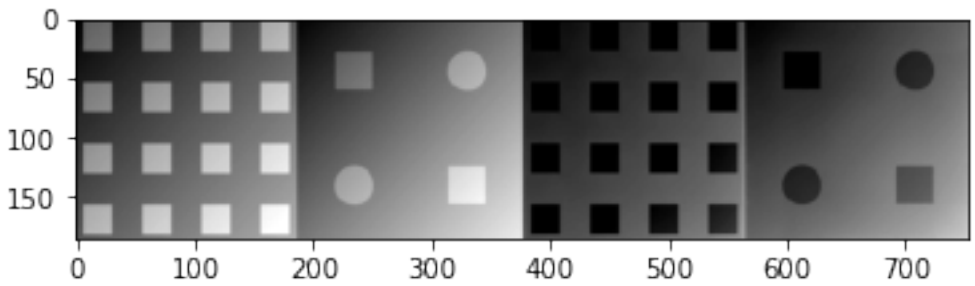
[4]:
```
#if(contraste_local < contraste_referencia)
#  pixel=(gris_medio >= 128)? objeto:background
#else
#  pixel=(pixel >= gris_medio)? objeto:background

def bernsen(tiles, contrast):
    for tile in tiles:
```

```
        if tile.size != 0:
            local_contrast = np.max(tile) - np.min(tile)
            mean = np.mean(tile)
            for i in range(tile.shape[0]):
                for j in range(tile.shape[1]):
                    if local_contrast < contrast:
                        if mean >= 128:
                            tile[i, j] = 255
                        else:
                            tile[i, j] = 0
                    else:
                        if tile[i, j] >= mean:
                            tile[i, j] = 255
                        else:
                            tile[i, j] = 0
```

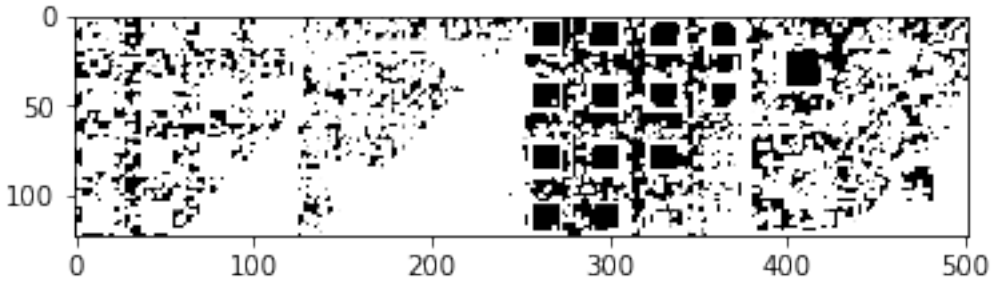
[5]:
```
original_image = cv.imread("Sombreado.png", cv.IMREAD_GRAYSCALE)
window_sizes = (3, 5, 7)
contrast = 2
```

[6]:
```
print("Original")
plt.imshow(original_image, cmap='gray', vmin=0, vmax=255)
plt.show()

for window_size in window_sizes:
    tiles, rows, cols = divide_in_tiles(original_image.copy(),␣
 ↪window_size)
    bernsen(tiles, contrast)
    binarized_image = generate_from_tiles(tiles, rows, cols)
    print("Binarized - Bernsen {0}X{0}".format(window_size))
    plt.figure(window_size)
    plt.imshow(binarized_image, cmap='gray', vmin=0, vmax=1)
    plt.show()
```

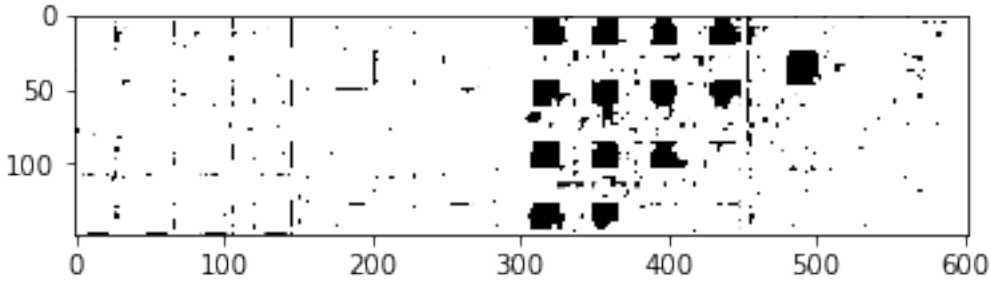
```
Original
```



Binarized - Bernsen 3X3



Binarized - Bernsen 5X5



Binarized - Bernsen 7X7