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)

8 de junio 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)
```

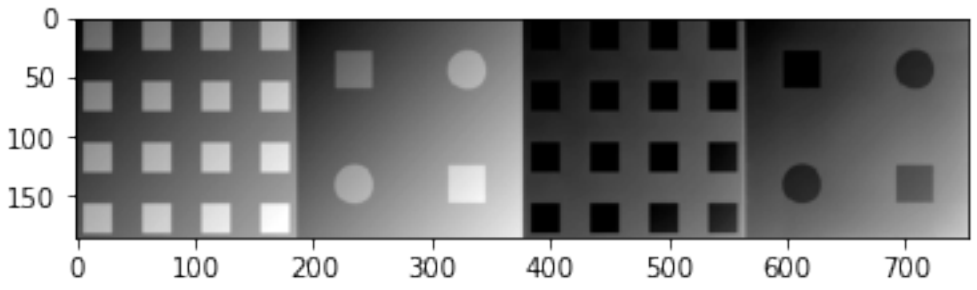
[5]:
```
original_image = cv.imread("Sombreado.png", cv.IMREAD_GRAYSCALE)

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]
```

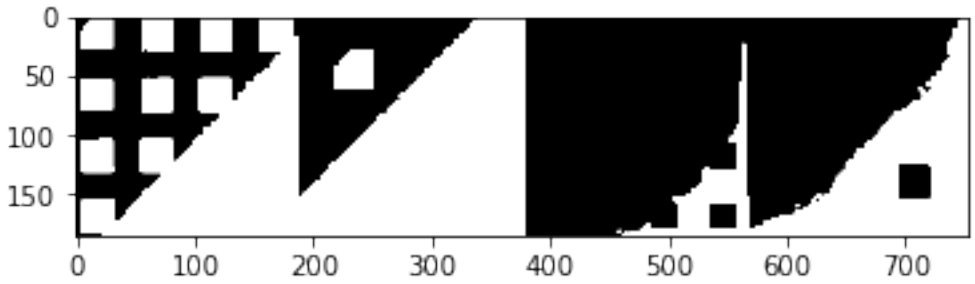
[8]:
```
print("Original")
plt.imshow(original_image, cmap='gray', vmin=0, vmax=255)
plt.show()
```

```
Original
```



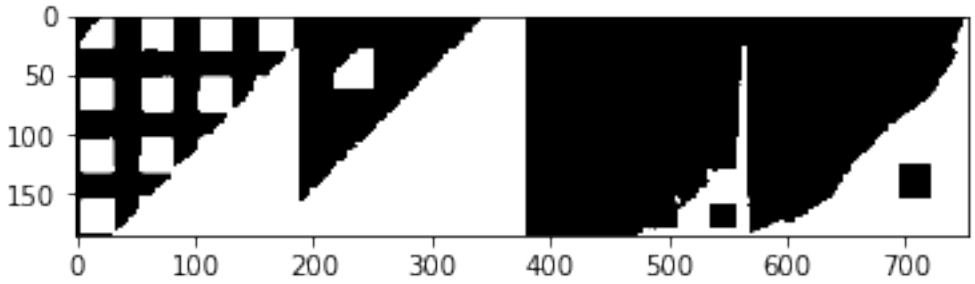
[9]:
```
print("Binarized - Fixed")
plt.imshow(fixed_binarized_image, cmap='gray', vmin=0, vmax=1)
plt.show()
```

```
Binarized - Fixed
```



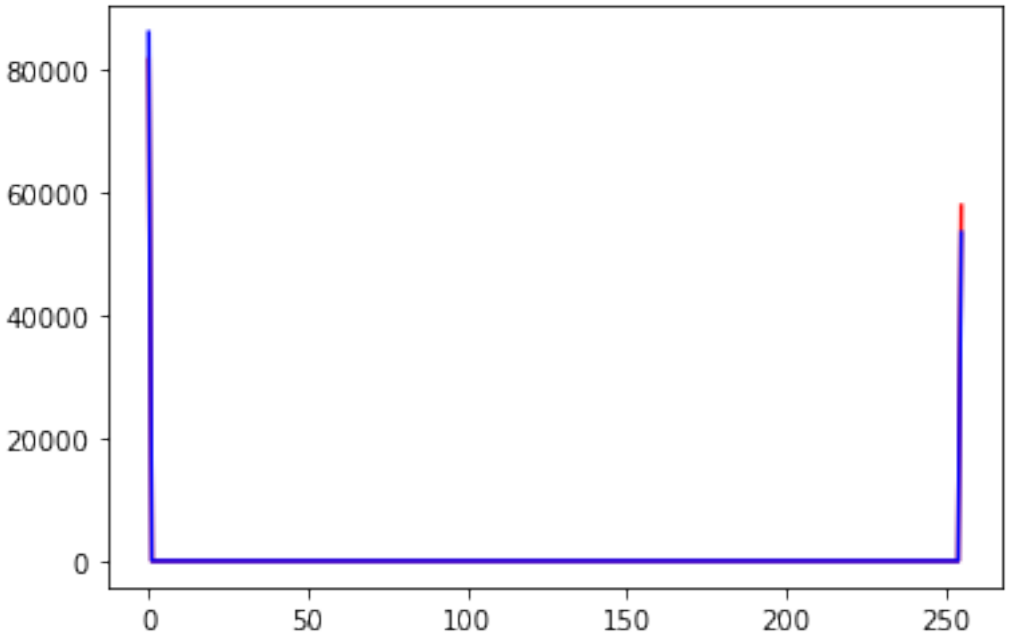
[10]:
```
print("Binarized - Otsu")
plt.imshow(otsu_binarized_image, cmap='gray', vmin=0, vmax=1)
plt.show()
```

```
Binarized - Otsu
```



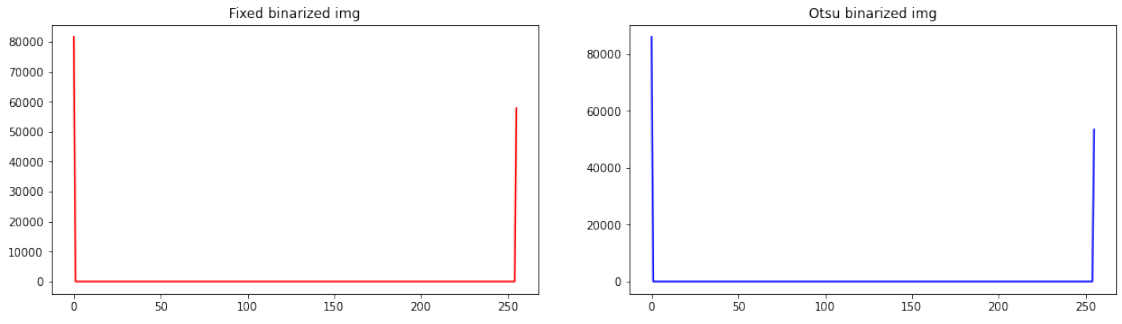
[11]:
```
fig = plt.figure()

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 resultado 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 get_window(image, center_x, center_y, size):
    image_height, image_width = image.shape
    window_half_size = np.floor(size/2).astype(int)

    top = max(center_y - window_half_size, 0)
    bottom = min(center_y + window_half_size, image_height - 1)
    left = max(center_x - window_half_size, 0)
    right = min(center_x + window_half_size, image_width - 1)

    return image[top:bottom, left:right]
```

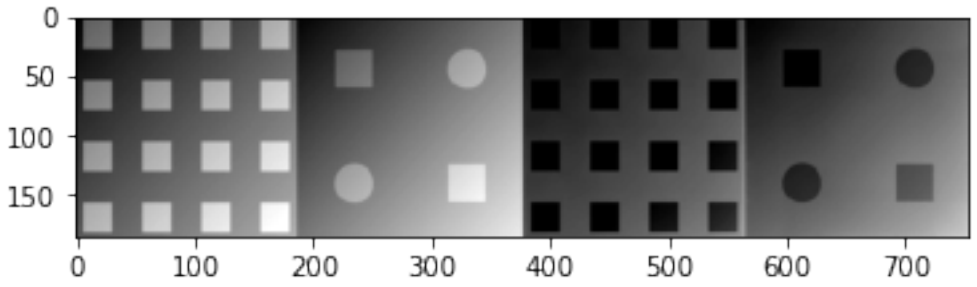
[3]:
```
#if(contraste_local < contraste_referencia)
#  pixel=(gris_medio >= 128)? objeto:background
#else
#  pixel=(pixel >= gris_medio)? objeto:background

def bernsen(image, contrast, window_size):
    rows = image.shape[0]
    columns = image.shape[1]
    output = image.copy()

    for i in range(rows):
        for j in range(columns):
            window = get_window(image, j, i, window_size)
            local_contrast = np.max(window) - np.min(window)
            mean = np.mean(window)
            if local_contrast < contrast:
                if mean >= 128:
                    output[i, j] = 255
                else:
                    output[i, j] = 0
            else:
                if output[i, j] >= mean:
```

```
                    output[i, j] = 255
                else:
                    output[i, j] = 0
    return output
```

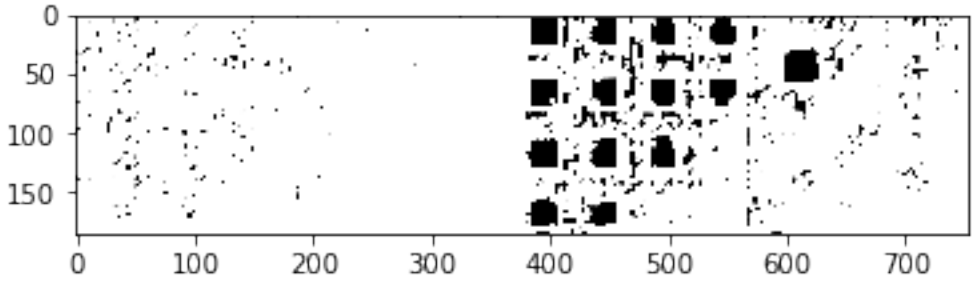
[4]:
```
original_image = cv.imread("Sombreado.png", cv.IMREAD_GRAYSCALE)
window_sizes = (3, 5, 7, 9, 11, 13, 15)
contrast = 2
```

[5]:
```
print("Original")
plt.imshow(original_image, cmap='gray', vmin=0, vmax=255)
plt.show()

for window_size in window_sizes:
    print("Processing Bernsen with window size {0}X{0}...\n".
 ↪format(window_size))
    binarized_image = bernsen(original_image, contrast, window_size)
    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
```

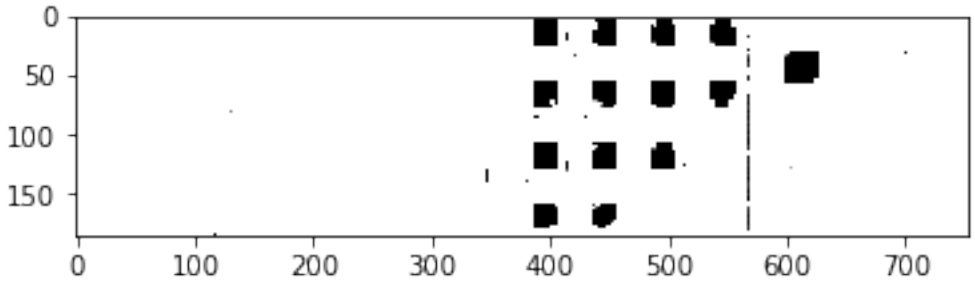



```
Processing Bernsen with window size 3X3...

Binarized - Bernsen 3X3
```



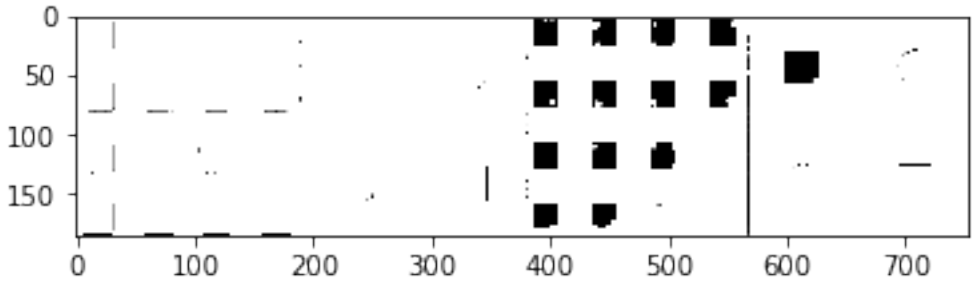
Processing Bernsen with window size 5X5...

Binarized - Bernsen 5X5



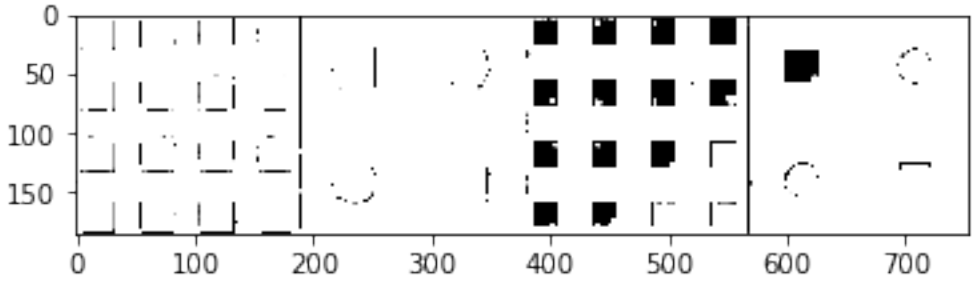
Processing Bernsen with window size 7X7...

Binarized - Bernsen 7X7



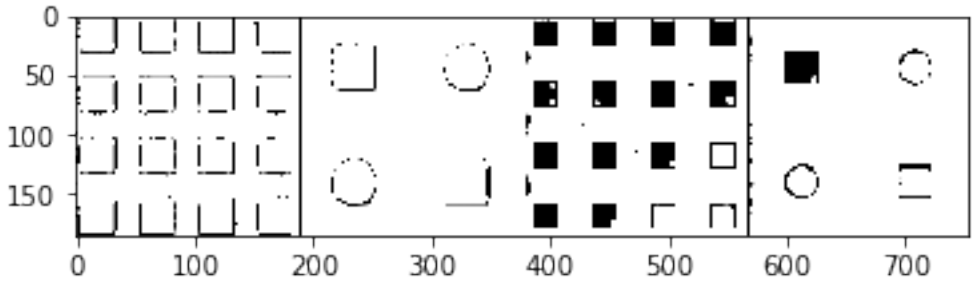
Processing Bernsen with window size 9X9...

Binarized - Bernsen 9X9



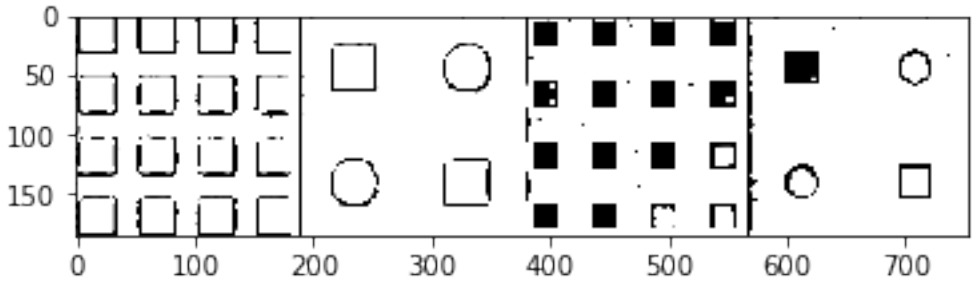
Processing Bernsen with window size 11X11...

Binarized - Bernsen 11X11



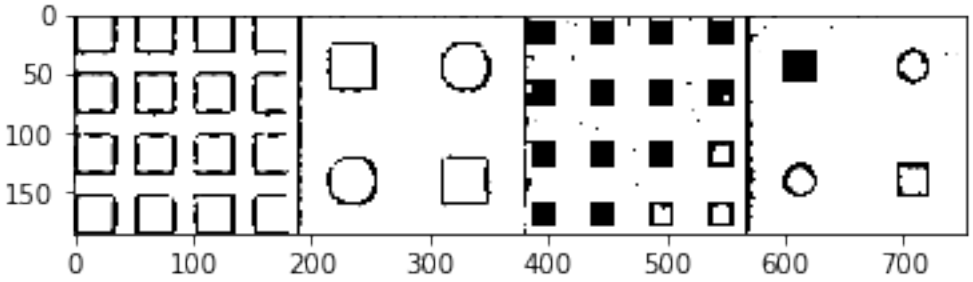
Processing Bernsen with window size 13X13...

Binarized - Bernsen 13X13



Processing Bernsen with window size 15X15...

Binarized - Bernsen 15X15



[ ]:

El mejor resultado se obtuvo con un tamaño de ventana de 7x7 y un contraste inicial bajo distinto de 0 (menor a 5. Si bien el resultado es mejor que los obtenidos por binarización global, no parece ser aceptable para binarizar esta imagen. De cualquier manera, si seguimos aumentando la cantidad de ventanas, podemos observar como va mejorando la binarización.