Introducción

El objetivo de este trabajo práctico es:

- Encontrar el umbral con búsqueda binaria según lo indicado en clase y comparar el resultado con la binarización por el método de Otsu.

- Programar el método de binarización local por Bernsen.

Desarrollo

Búsqueda binaria

Supuestos:

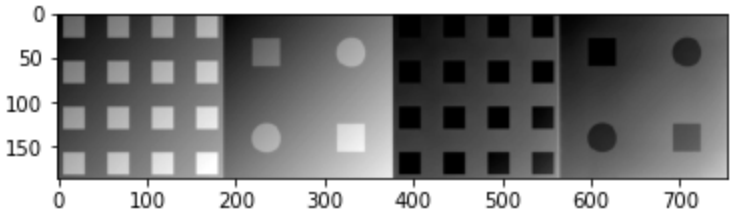
- *Se utiliza el valor de delta por defecto indicado en clase.*

```
def get_sub_array(array, indexes):
    sub_array = []

    for index in indexes:
        sub_array.append(array[indexes])

    return sub_array

def calculate_mean(values):
    return np.mean(values)

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
```

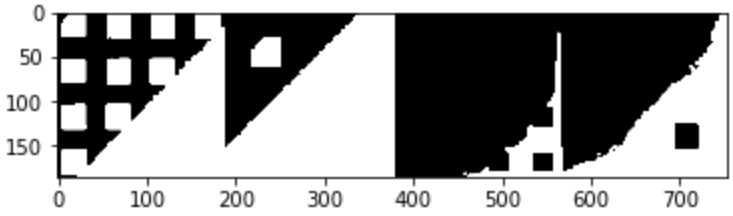
```python
def find_threshold(image, threshold = 128, delta = 1.0):
    lower_values,upper_values = divide_image(image, threshold)

    lower_mean = calculate_mean(lower_values)

    upper_mean = calculate_mean(upper_values)

    new_threshold = calculate_mean([lower_mean, upper_mean])

    if abs(new_threshold - threshold) < delta:
        print("DONE\n")
        return new_threshold
    else:
        print("processing...")
        return find_threshold(image, new_threshold, delta)
```

```python
original_image = cv.imread("shading.png",cv.IMREAD_GRAYSCALE)
mean = calculate_mean(original_image.ravel())
manual_threshold = find_threshold(original_image, mean)
fixed_binarized_image = cv.threshold(original_image,
                                     manual_threshold, 255,
                                     cv.THRESH_BINARY)[1]
otsu_binarized_image = cv.threshold(original_image,
                                    manual_threshold, 255,
                                    cv.THRESH_BINARY + cv.THRESH_OTSU)[1]

print("Original")
plt.imshow(original_image, cmap='gray', vmin=0, vmax=255)
plt.show()

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

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

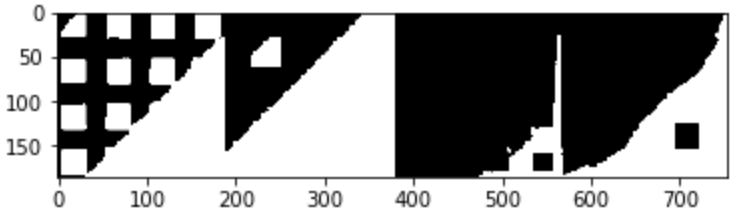
Original



Binarized - Fixed



Binarized - Otsu



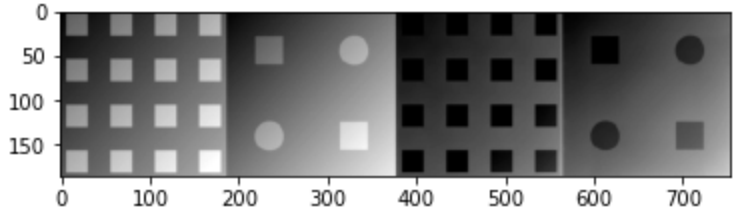
Se verifica que el resultado de la binarización fija es similar al método de Otsu y se concluye que este tipo de binarización no es adecuado para este tipo de imágenes.

Bernsen

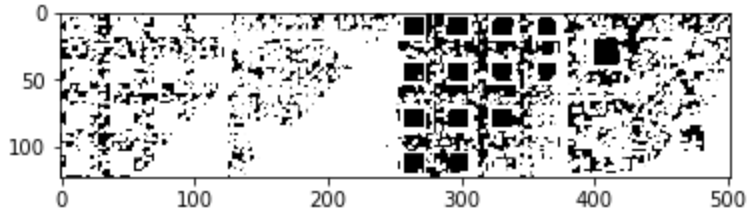
```
def divide_in_tiles(image, tile_size):
    tiles = []
    image_height, image_width = image.shape
    rows = np.ceil(image_height / tile_size).astype(int)
    cols = np.ceil(image_width/ tile_size).astype(int)

    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

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)

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
```

```
original_image = cv.imread("shading.png", cv.IMREAD_GRAYSCALE)
window_sizes = (3, 5, 7)
contrast = 2

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()
```

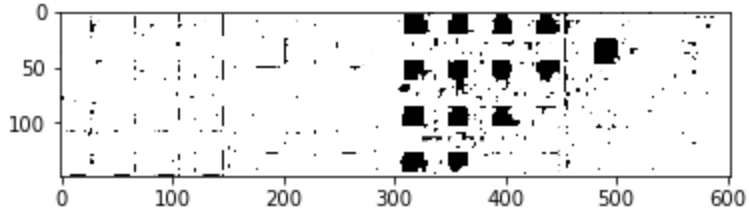
Original



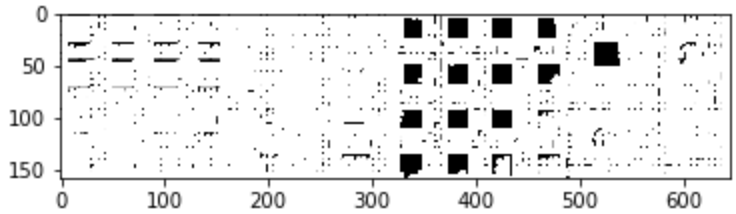
Binarized - Bernsen 3X3



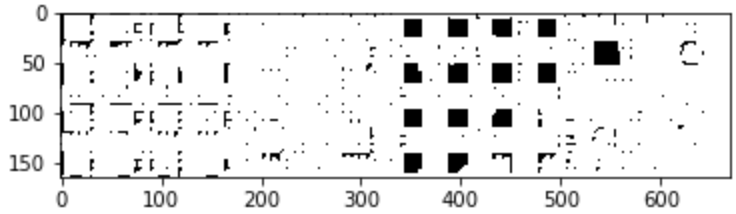
Binarized - Bernsen 5X5



Binarized - Bernsen 7X7



Binarized - Bernsen 9X9



Se verifica que 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 obtenido es mejor que en el punto anterior, no parece ser aceptable para reconocer las formas de la imagen.