Digital Image Processing - Lab Session 4

Sofia Noemi Crobeddu
student number: 032410554

International Master of Biometrics and Intelligent Vision
Université Paris-Est Cretéil (UPEC)

December 3, 2024

Contents

Introduction

In this project we analyze some morphological operations in image processing: erosion, dilation, opening, closing and top-hat transform.

1 Morphology

In the field of image processing, mathematical morphology is the study of the geometric structure of the image. It is a useful tool for the representation and description of the shape of a region, and its approach is through set operations. The aim is to distinguish meaningful information about the form from irrelevant information.

In this first step, we focus on defining and applying **structuring elements** for two morphological operations: dilation and erosion. In these operations structuring elements are used to probe or explore the input image, obtaining the result based on the connectivity of pixels. As we saw in Assignment 3, we have two types of connectivity:

- **4-connectivity** or **V4**, where a pixel is connected with its neighbors that lie horizontally or vertically, i.e. it has 4 possible neighbors (up, down, left, right).
- **8-connectivity** or **V8**, where a pixel is connected with all its neighbors, including the diagonal ones, i.e. it has 8 possible neighbors (4 horizontal or vertical and 4 diagonal).

A structuring element is a matrix that defines the neighborhood used to process each pixel in the image. The central pixel of the structuring element, i.e. the origin, corresponds to the pixel in the image being processed. We build them based on V4 and V8. There re two methods to find them: with the library *skimage.morphology* and with *OpenCV*.

Below you can see both the options to construct them. For the next points we will consider the ones found with OpenCV just for coherence (we will use the same library), but the result is the same.

```python
from skimage.morphology import square, diamond

#V4 Structuring Element (4-connectivity)
struct_el_v4 = diamond(1)  #Generates a diamond-shaped structuring element for 4-connectivity
print("V4 Structuring Element:\n", struct_el_v4)

#V8 Structuring Element (8-connectivity)
struct_el_v8 = square(3)  #Generates a square structuring element for 8-connectivity
print("V8 Structuring Element:\n", struct_el_v8)
```

```python
import cv2

#Option 2 with cv2
struct_el2_v4 = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))
print("V4 Structuring Element:\n", struct_el2_v4)

# V8 Structuring Element (8-connectivity)
struct_el2_v8 = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
print("V8 Structuring Element:\n", struct_el2_v8)
```

Here there are the outputs:

- **V4 Structuring Element:**

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- **V8 Structuring Element:**

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

In this second step we use the structural elements obtained before, to implement the operations of erosion and dilation. Having a look at them more deeply:

- **erosion** removes pixels on the boundaries of the objects in an image, and it is implemented by comparing the structural element with the neighborhood of each pixel. If the structuring element fits completely within the pixel's neighborhood, this pixel remains; otherwise, it is removed and set to 0.

- **dilation** is basically the opposite of erosion. It adds pixels to the boundaries of the objects in an image, and it is implemented as erosion, by comparing the structural element with the neighborhood of each pixel. If the structuring element fits partially o completely within the neighborhood, the pixel is set to 1 and becomes part of the object.

To implement them, we first convert the images into binary ones, applying the Otsu's thresholding method seen in previous assignment. In both cases, we use the structuring elements with 4-connectivity and 8-connectivity, and we apply them to *shape* and *shape2* images. The functions used are *erode* and *dilate* from *OpenCv*.
The process followed is similar for both images and in both images the number of iterations used is 2. Since in the first original image *shape* the objects are thinner, we can notice that for example in erosion with a structuring element V-8, we can see just some small points for the second image.
In general, by comparing the results of erosion and dilation on both images, it was possible to visually distinguish between diagonal and circular shapes, except for erosion V8 (since we put 2 iterations). regarding the ability to count clearly the objects, in these cases it is possible. However, it may depend on the specific arrangement and connectivity of shapes in the image.
The codes are shown below and the resulting images obtained are in Figures 1 and 2.

```python
from PIL import Image
import numpy as np
from skimage.filters import threshold_otsu

#Load the image shapes
image_shapes = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 4 - IP\\IP4_v2 1\\IP4_v2\\shapes.png")
#To array
image_array_shapes = np.array(image_shapes)

#Function for Otsu thresholding
def otsu_method(img):
    otsu_threshold = threshold_otsu(img)
    #Binarization
    binary_image = (img > otsu_threshold).astype(np.uint8)
    return binary_image, otsu_threshold

#Binarization
binary_image_shapes, otsu_threshold_shapes = otsu_method(image_array_shapes)
```

```python
import matplotlib.pyplot as plt
from matplotlib.colors import NoNorm

#Erosion and Dilation for image shapes through the kernels found in point 1.1 with cv2.
#Through V4 structuring element
eroded_v4_shapes = cv2.erode(binary_image_shapes, struct_el2_v4, iterations=2)
dilated_v4_shapes = cv2.dilate(binary_image_shapes, struct_el2_v4, iterations=2)

#Through V8 structuring element
eroded_v8_shapes = cv2.erode(binary_image_shapes, struct_el2_v8, iterations=2)
dilated_v8_shapes = cv2.dilate(binary_image_shapes, struct_el2_v8, iterations=2)

#Plot results
fig, ax = plt.subplots(3, 2, figsize=(8, 10))
ax[0, 0].imshow(image_shapes, cmap='gray', norm=NoNorm())
ax[0, 0].set_title("Original shapes image.")
ax[0, 0].axis('off')

ax[0, 1].imshow(binary_image_shapes, cmap='gray')
ax[0, 1].set_title("Binary shapes image.")
ax[0, 1].axis('off')

ax[1, 0].imshow(eroded_v4_shapes, cmap='gray')
ax[1, 0].set_title("Erosion (V4) for shapes image.")
ax[1, 0].axis('off')

ax[1, 1].imshow(dilated_v4_shapes, cmap='gray')
ax[1, 1].set_title("Dilation (V4) for shapes image.")
ax[1, 1].axis('off')

ax[2, 0].imshow(eroded_v8_shapes, cmap='gray')
ax[2, 0].set_title("Erosion (V8) for shapes image.")
ax[2, 0].axis('off')
```

```python

ax[2, 1].imshow(dilated_v8_shapes, cmap='gray')
ax[2, 1].set_title("Dilation (V8) for shapes image.")
ax[2, 1].axis('off')

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

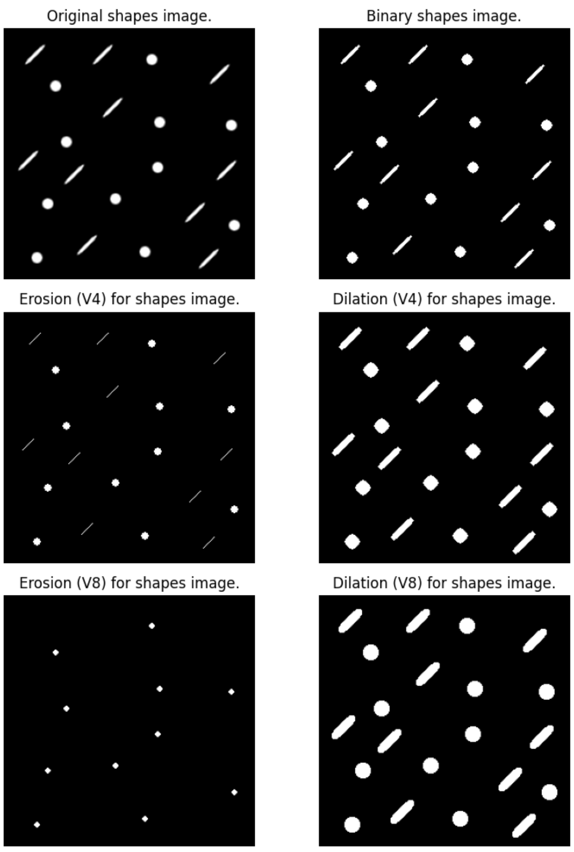
Original shapes image.
Binary shapes image.
Erosion (V4) for shapes image.
Dilation (V4) for shapes image.
Erosion (V8) for shapes image.
Dilation (V8) for shapes image.

Fig. 1: Erosion and Dilation for *shapes* image.

```python
#Load the image shapes2
image_shapes2 = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 4 - IP\\IP4_v2 1\\IP4_v2\\shapes2.png")
#To array
image_array_shapes2 = np.array(image_shapes2)

#Binarization
binary_image_shapes2, otsu_threshold_shapes2 = otsu_method(image_array_shapes2)
```

```python
#Erosion and Dilation for image shapes through the kernels found in point 1.1 with cv2.
#Through V4 structuring element
eroded_v4_shapes2 = cv2.erode(binary_image_shapes2, struct_el2_v4, iterations=2)
dilated_v4_shapes2 = cv2.dilate(binary_image_shapes2, struct_el2_v4, iterations=2)

#Through V8 structuring element
eroded_v8_shapes2 = cv2.erode(binary_image_shapes2, struct_el2_v8, iterations=2)
dilated_v8_shapes2 = cv2.dilate(binary_image_shapes2, struct_el2_v8, iterations=2)

#Plot results
fig, ax = plt.subplots(3, 2, figsize=(8, 10))
ax[0, 0].imshow(image_shapes2, cmap='gray', norm=NoNorm())
ax[0, 0].set_title("Original shapes2 image.")
ax[0, 0].axis('off')

ax[0, 1].imshow(binary_image_shapes2, cmap='gray')
```

```
ax[0, 1].set_title("Binary shapes2 image.")
ax[0, 1].axis('off')

ax[1, 0].imshow(eroded_v4_shapes2, cmap='gray')
ax[1, 0].set_title("Erosion (V4) for shapes2 image.")
ax[1, 0].axis('off')

ax[1, 1].imshow(dilated_v4_shapes2, cmap='gray')
ax[1, 1].set_title("Dilation (V4) for shapes2 image.")
ax[1, 1].axis('off')

ax[2, 0].imshow(eroded_v8_shapes2, cmap='gray')
ax[2, 0].set_title("Erosion (V8) for shapes2 image.")
ax[2, 0].axis('off')

ax[2, 1].imshow(dilated_v8_shapes2, cmap='gray')
ax[2, 1].set_title("Dilation (V8) for shapes2 image.")
ax[2, 1].axis('off')

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

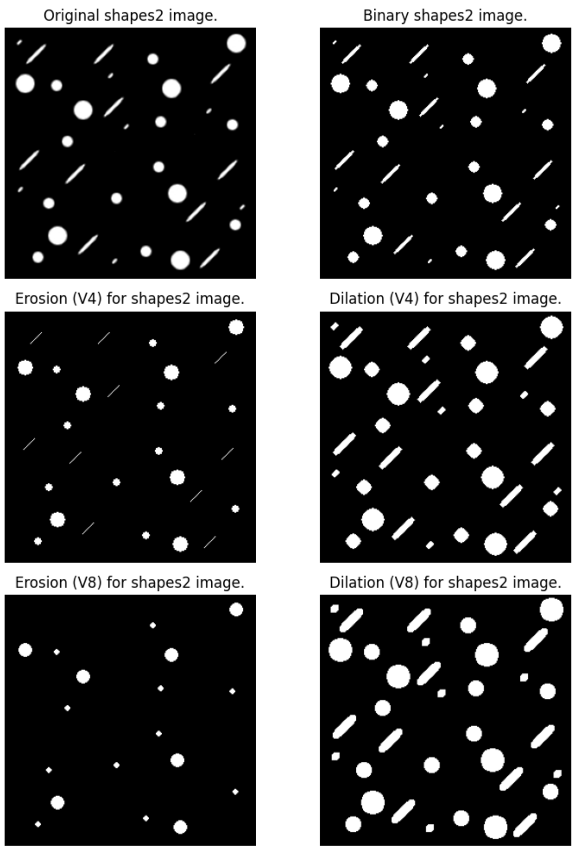
Original shapes2 image.
Binary shapes2 image.
Erosion (V4) for shapes2 image.
Dilation (V4) for shapes2 image.
Erosion (V8) for shapes2 image.
Dilation (V8) for shapes2 image.

Fig. 2: Erosion and Dilation for *shapes2* image.

Now we go to the next step, where we analyze the operations of opening and closing on *eng* and *eng2* images. Having a look at them more deeply:

- **opening** is equal to erosion followed by dilation. It is used to remove small objects, i.e. noise, from the image, while preserving the shape and size of the larger objects.
- **closing** is instead the opposite process, i.e. dilation followed by erosion. It is efficient for closing small gaps within the objects in the image.

In both operations we use the structuring elements that define the shape and the size of the neighborhood around each pixel to consider during the operation. As for the previous points we use str. el. with 4-connectivity and 8-connectivity.
Another comparison that we do is the application of these operations on images modified by the function *binary_fill_holes*: it fills the small holes within the objects. This function is important since sometimes, after applying this type of operations, gaps remain within the object of interest. With this function we fill them, in order to improve the visual outcome. The application of the function can be done before or after opening/closing. It is a personal choice and in this case, after some trials, it was better to apply it before.
The process was the same for both the images, and the code is shown below. As we can see from Figures 3 and 4, it is possible to count the external ”teeth” in all cases, but the visualization of them is better after the application of *binary_fill_holes*, since the center in black is not present. The difference between the application of opening and closing with a structural element V4 or V8 is not so evident here, since we didn't increment the number of iterations. However the slightly difference show us that 4-connectivity is more suitable for these images.

```python
#Load the image eng
image_eng = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\assignment
     4 - IP\\IP4_v2 1\\IP4_v2\\eng.png").convert("L")
#We put .convert("L") to take just 2 dimensions. This is due to the fact that Image.open in
    reading the image, interprets it as 3 channels.

#To array
image_array_eng = np.array(image_eng)

#Binarization
binary_image_eng, otsu_threshold_eng = otsu_method(image_array_eng)
```

```python
from scipy.ndimage import binary_fill_holes
from matplotlib.gridspec import GridSpec

#Opening: removes noise.
#This operation is the same as Erosion + Dilation.
opened4_eng = cv2.morphologyEx(binary_image_eng, cv2.MORPH_OPEN, struct_el2_v4) #(V4)
opened8_eng = cv2.morphologyEx(binary_image_eng, cv2.MORPH_OPEN, struct_el2_v8) #(V8)

#Closing: closes the gaps.
#This operation is the same as Dilation + Erosion.
closed4_eng = cv2.morphologyEx(opened4_eng, cv2.MORPH_CLOSE, struct_el2_v4)
closed8_eng = cv2.morphologyEx(opened8_eng, cv2.MORPH_CLOSE, struct_el2_v8)

#Filling the gaps
opened4_eng_post = binary_fill_holes(opened4_eng).astype(np.uint8) * 255
opened8_eng_post = binary_fill_holes(opened8_eng).astype(np.uint8) * 255
closed4_eng_post = binary_fill_holes(closed4_eng).astype(np.uint8) * 255
closed8_eng_post = binary_fill_holes(closed4_eng).astype(np.uint8) * 255

#Creating the grid for the graphs
fig = plt.figure(figsize=(20, 15))
gs = GridSpec(3, 4, figure=fig)

#First row with two columns
ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[0, 1])

#Second row with four columns
ax3 = fig.add_subplot(gs[1, 0])
ax4 = fig.add_subplot(gs[1, 1])
ax5 = fig.add_subplot(gs[1, 2])
ax6 = fig.add_subplot(gs[1, 3])

#Third row with four columns
ax7 = fig.add_subplot(gs[2, 0])
ax8 = fig.add_subplot(gs[2, 1])
ax9 = fig.add_subplot(gs[2, 2])
ax10 = fig.add_subplot(gs[2, 3])

#Plots
ax1.imshow(image_eng, cmap='gray', norm=NoNorm())
ax1.set_title("Image eng.")
ax1.axis('off')

ax2.imshow(binary_image_eng, cmap='gray')
ax2.set_title("Binary eng image.")
ax2.axis('off')
```

```

ax3.imshow(opened4_eng, cmap='gray')
ax3.set_title("Opening (V4) for eng image.")
ax3.axis('off')

ax4.imshow(opened4_eng_post, cmap='gray')
ax4.set_title("Opening (V4) for eng image after binary_fill_holes.")
ax4.axis('off')

ax5.imshow(opened8_eng, cmap='gray')
ax5.set_title("Opening (V8) for eng image.")
ax5.axis('off')

ax6.imshow(opened8_eng_post, cmap='gray')
ax6.set_title("Opening (V8) for eng image after binary_fill_holes.")
ax6.axis('off')

ax7.imshow(closed4_eng, cmap='gray')
ax7.set_title("Closing (V4) for eng image.")
ax7.axis('off')

ax8.imshow(closed4_eng_post, cmap='gray')
ax8.set_title("Closing (V4) for eng image after binary_fill_holes.")
ax8.axis('off')

ax9.imshow(closed8_eng, cmap='gray')
ax9.set_title("Closing (V8) for eng image.")
ax9.axis('off')

ax10.imshow(closed8_eng_post, cmap='gray')
ax10.set_title("Closing (V8) for eng image after binary_fill_holes.")
ax10.axis('off')

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

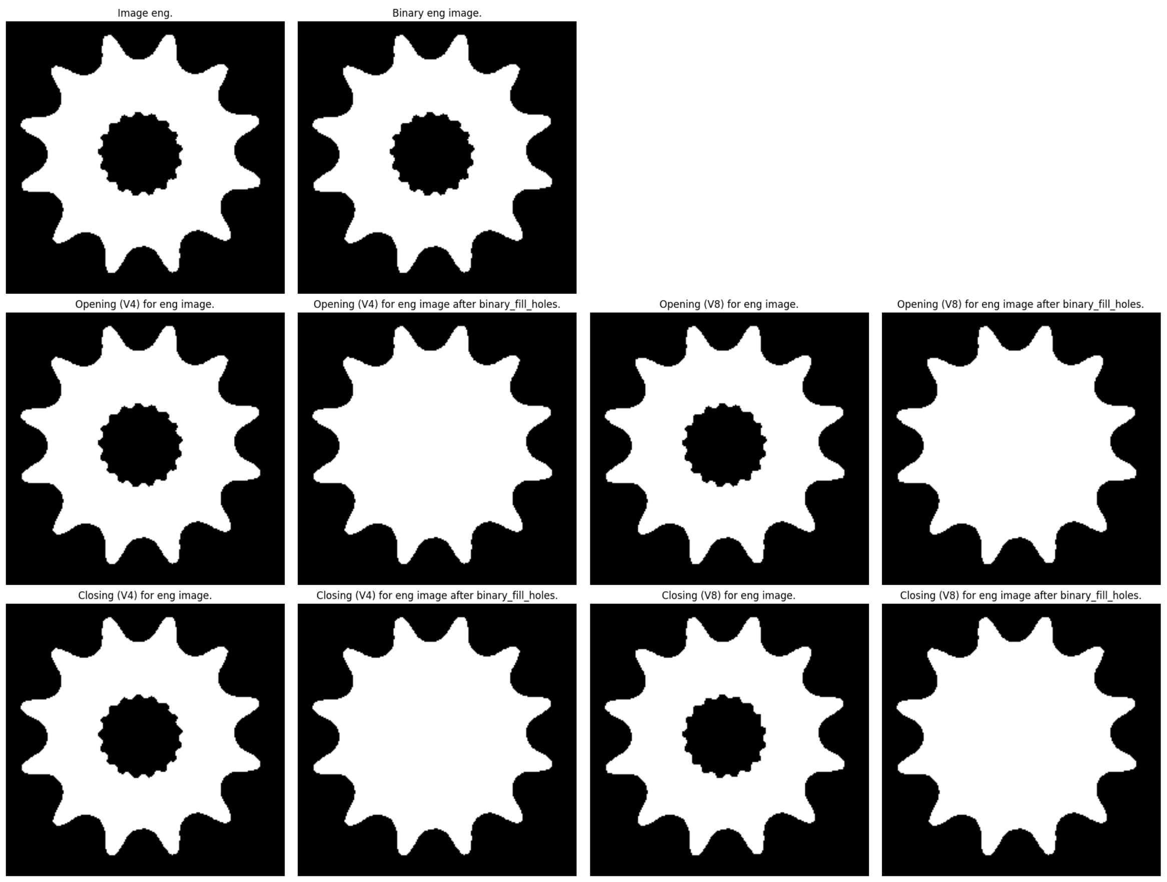
Image eng.
Binary eng image.
Opening (V4) for eng image.
Opening (V4) for eng image after binary_fill_holes.
Opening (V8) for eng image.
Opening (V8) for eng image after binary_fill_holes.
Closing (V4) for eng image.
Closing (V4) for eng image after binary_fill_holes.
Closing (V8) for eng image.
Closing (V8) for eng image after binary_fill_holes.

Fig. 3: Opening and Closing for *eng* image.

```python
#Load the image eng2
image_eng2 = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 4 - IP\\IP4_v2 1\\IP4_v2\\eng2.png").convert("L")
#To array
image_array_eng2 = np.array(image_eng2)

#Binarization
binary_image_eng2, otsu_threshold_eng2 = otsu_method(image_array_eng2)
```

```python
#Opening
opened4_eng2 = cv2.morphologyEx(binary_image_eng2, cv2.MORPH_OPEN, struct_el2_v4) #(V4)
opened8_eng2 = cv2.morphologyEx(binary_image_eng2, cv2.MORPH_OPEN, struct_el2_v8) #(V8)

#Closing
closed4_eng2 = cv2.morphologyEx(opened4_eng2, cv2.MORPH_CLOSE, struct_el2_v4)
closed8_eng2 = cv2.morphologyEx(opened8_eng2, cv2.MORPH_CLOSE, struct_el2_v8)

#Filling the gaps
opened4_eng2_post = binary_fill_holes(opened4_eng2).astype(np.uint8) * 255
opened8_eng2_post = binary_fill_holes(opened8_eng2).astype(np.uint8) * 255
closed4_eng2_post = binary_fill_holes(closed4_eng2).astype(np.uint8) * 255
closed8_eng2_post = binary_fill_holes(closed8_eng2).astype(np.uint8) * 255

#Creating the grid for the graphs
fig = plt.figure(figsize=(20, 12))
gs = GridSpec(3, 4, figure=fig)

#First row with two columns
ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[0, 1])

#Second row with four columns
ax3 = fig.add_subplot(gs[1, 0])
ax4 = fig.add_subplot(gs[1, 1])
ax5 = fig.add_subplot(gs[1, 2])
ax6 = fig.add_subplot(gs[1, 3])

#Third row with four columns
ax7 = fig.add_subplot(gs[2, 0])
ax8 = fig.add_subplot(gs[2, 1])
ax9 = fig.add_subplot(gs[2, 2])
ax10 = fig.add_subplot(gs[2, 3])

#Plots
ax1.imshow(image_eng2, cmap='gray', norm=NoNorm())
ax1.set_title("Image eng2.")
ax1.axis('off')

ax2.imshow(binary_image_eng2, cmap='gray')
ax2.set_title("Binary eng2 image.")
ax2.axis('off')

ax3.imshow(opened4_eng2, cmap='gray')
ax3.set_title("Opening (V4) for eng2 image.")
ax3.axis('off')

ax4.imshow(opened4_eng2_post, cmap='gray')
ax4.set_title("Opening (V4) for eng2 image after binary_fill_holes.")
ax4.axis('off')

ax5.imshow(opened8_eng2, cmap='gray')
ax5.set_title("Opening (V8) for eng2 image.")
ax5.axis('off')

ax6.imshow(opened8_eng2_post, cmap='gray')
ax6.set_title("Opening (V8) for eng2 image after binary_fill_holes.")
ax6.axis('off')

ax7.imshow(closed4_eng2, cmap='gray')
ax7.set_title("Closing (V4) for eng2 image.")
ax7.axis('off')

ax8.imshow(closed4_eng2_post, cmap='gray')
ax8.set_title("Closing (V4) for eng2 image after binary_fill_holes.")
ax8.axis('off')

ax9.imshow(closed8_eng2, cmap='gray')
ax9.set_title("Closing (V8) for eng2 image.")
ax9.axis('off')

ax10.imshow(closed8_eng2_post, cmap='gray')
```

```python
ax10.set_title("Closing (V8) for eng2 image after binary_fill_holes.")
ax10.axis('off')

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

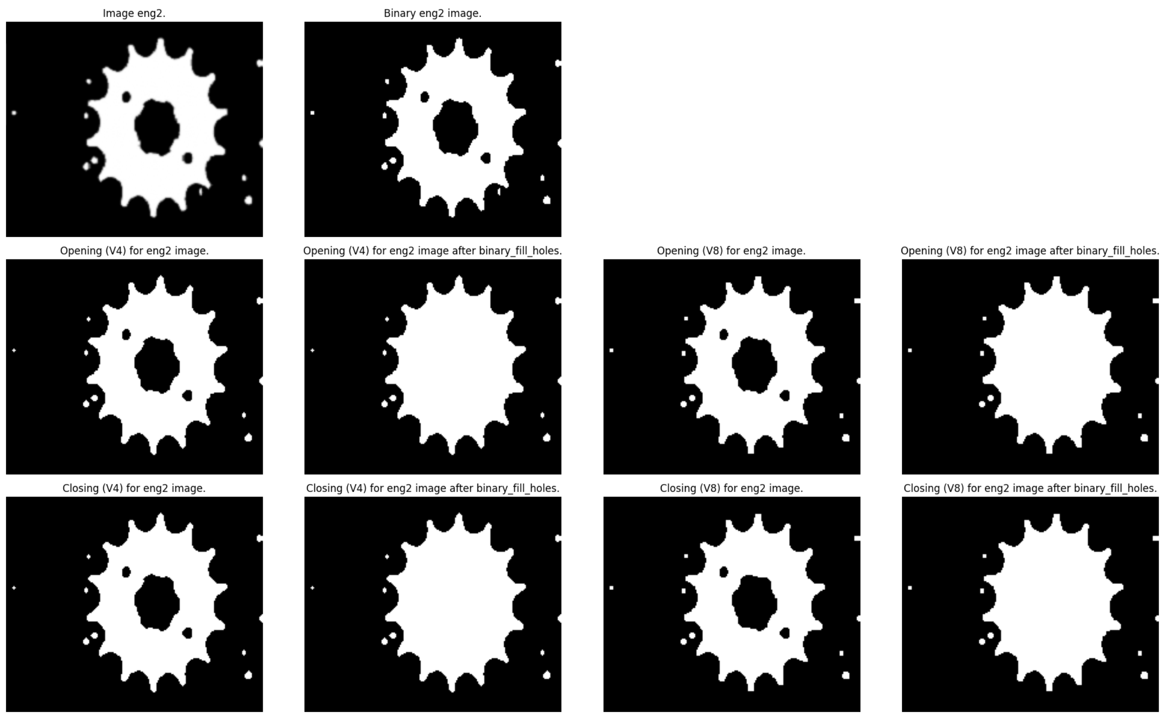
Image eng2.
Binary eng2 image.
Opening (V4) for eng2 image.
Opening (V4) for eng2 image after binary_fill_holes.
Opening (V8) for eng2 image.
Opening (V8) for eng2 image after binary_fill_holes.
Closing (V4) for eng2 image.
Closing (V4) for eng2 image after binary_fill_holes.
Closing (V8) for eng2 image.
Closing (V8) for eng2 image after binary_fill_holes.

Fig. 4: Opening and Closing for *eng* image.

In this final step we apply Otsu's method on the original image *I12* and on the image after the application of **top-hat transformation**. This technique enhances bright regions that are smaller than the structuring element, so basically it isolates bright details like small objects or features from the background. The structural element here, is not the same found for the techniques applied before, but it is instead a disk-shaped structuring element. The radius of the disk is set to 40.
The code to perform this analysis is shown below, and Figure 5 is the result. As we can see, in the original image the white objects are not well defined, while in the binarized one they tend to disappear. In the binarized image after top-hat, instead, the objects are very evident and the result is good.

```python
#Load the image I12
image_I12 = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\assignment
     4 - IP\\IP4_v2 1\\IP4_v2\\I12.bmp").convert("L")
#To array
image_array_I12 = np.array(image_I12)
```

```python
from skimage.morphology import disk, white_tophat

#Binarization using Otsu
binary_image_I12, otsu_threshold_I12 = otsu_method(image_array_I12)

#Application of white top-hat transform
radius = 40
struct_el_disk = disk(radius)  #disk structuring element
tophat_image_I12 = white_tophat(image_array_I12, footprint=struct_el_disk)

#Plot
fig, ax = plt.subplots(1, 3, figsize=(12, 4))

#Original image
ax[0].imshow(image_array_I12, cmap='gray', norm=NoNorm())
ax[0].set_title("I12 image.")
```

```
ax[0].axis('off')

#Otsu binarization
ax[1].imshow(binary_image_I12, cmap='gray')
ax[1].set_title("Binary I12 image with Otsu thresholding.")
ax[1].axis('off')

#Top-hat image
ax[2].imshow(tophat_image_I12, cmap='gray')
ax[2].set_title("White top-hat transformation of I12 image.")
ax[2].axis('off')

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

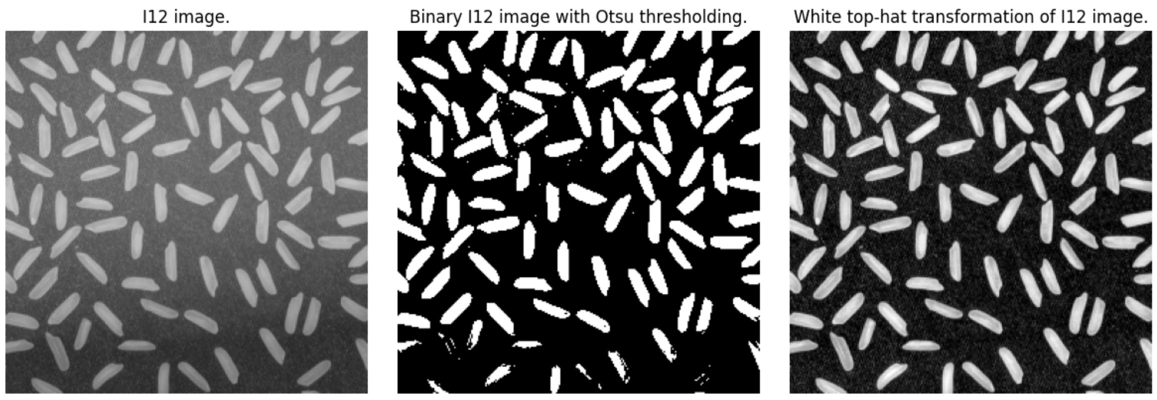
I12 image.
Binary I12 image with Otsu thresholding.
White top-hat transformation of I12 image.

Fig. 5: Top-hat transformation on *I12* image.

References

[1] YouTube: Morphological Image Processing - Top Hat Transform and Applications (2022). https://www.youtube.com/watch?v=WQK_oOWW5Zo&t=485s

[2] MathWorks: Structuring Elements (2024). https://fr.mathworks.com/help/images/structuring-elements.html

[3] Battiato, S.: La Morfologia Matematica (2012). https://www.dmi.unict.it/\texttildelowbattiato/mm1112/Parte%209%20-%20La%20Morfologia%20Matematica.pdf

[4] Khanal, S.: Relationships between pixels: Neighbours and connectivity. Medium (2023)

[5] Anshul: Opening morphological operation: Image processing. Medium (2023)

[6] Sasasulakshi: Opencv morphological dilation and erosion. Medium (2023)

[7] Top-hat transform (2024). https://en.wikipedia.org/wiki/Top-hat_transform

[8] Majidzadeh, F.: Digital Image Processing and Pattern Recognition, (2023)

[9] YouTube: Video on Color Detection (2024). https://www.youtube.com/watch?v=qe4fpdNzfxU