Digital Image Processing - Lab Session 5

Sofia Noemi Crobeddu
student number: 032410554

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

December 9, 2024

Contents

Introduction

In this assignment we analyze two images, *chro* and *coins*. We focus in particular in labeling, cleaning of the border and the analysis on the surface area of the connected components. We also estimate the radius starting from the area and the perimeter.

1 Analysis

We start this section with the analysis of *chro* image. More specifically, we remove the objects located in the borders, since they could disrupt the analysis, introducing inaccuracies and misrepresenting the image content.
Before doing this, we load the image and transform it in gray scale using *convert('L')*. After this, we use Otsu's method to binarize the image, in order to separate foreground objects from the background based on the intensity distribution. The function *label* from *skimage* library is applied to the binary image, in order to assign unique labels to connected components. Finally, it is applied the function *clear_border* to remove the objects at the labels of the image, i.e. the objects that are incomplete and may skew subsequent analysis.
Counting the number of removed objects, we have that:

- the number of objects in the original image is 10;
- the number of objects after labeling is 6;
- the number of removed objects is 4.

The code is shown below, and Figure 1 shows the original image, the binarized, the labeled and *chro* without the removed objects.

```python
image_chro = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 5 - IP\\IP5_v2\\IP5_v2\\chro.bmp").convert('L')

#To array
image_array_chro = np.array(image_chro)

#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_chro, otsu_threshold_chro = otsu_method(image_array_chro)

#Labeling
labeled_image_chro, num_objects_before = label(binary_image_chro, return_num=True)

#Removing the objects on the label
cleared_image_chro = clear_border(binary_image_chro)
cleared_image_chro, num_objects_after = label(cleared_image_chro, return_num=True)

#Counting the number of removed objects
objects_removed_chro = num_objects_before - num_objects_after
```

```python
#Plot results
fig, ax = plt.subplots(1, 4, figsize=(20, 15))
ax[0].imshow(image_chro, cmap='gray', norm=NoNorm())
ax[0].set_title("Original chro image.")
ax[0].axis('off')

ax[1].imshow(binary_image_chro, cmap='gray')
ax[1].set_title("Binary chro image.")
ax[1].axis('off')

ax[2].imshow(labeled_image_chro, cmap='nipy_spectral')
ax[2].set_title("Chro image with labeled objects.")
ax[2].axis('off')

ax[3].imshow(cleared_image_chro, cmap='nipy_spectral')
ax[3].set_title("Chro image after labeling and cleaning.")
ax[3].axis('off')
```

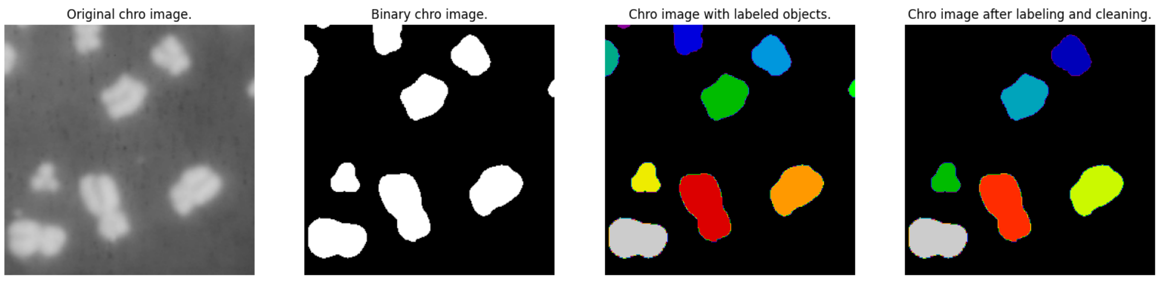
Original chro image.
Binary chro image.
Chro image with labeled objects.
Chro image after labeling and cleaning.

Fig. 1: Remotion of disruptive objects in *chro* image.

In this second part of the assignment we analyze *coins* image. We want to classify and count the coins in the image, based on their surface area. Coins are divided into two classes:

- Class 1 comprehends coins with a surface area between 1500 and 2000 pixels;
- Class 2 comprehends coins with a surface area between 2300 and 2800 pixels.

After loading the image and transforming it into a binary one through Otsu's method, we also ensure that no holes are present inside the objects, using *binary_fill_holes* function. After this, we label the image as we did for the analysis before in *chro*, and we save the labeled image as *L*.
The code of this first step is shown below, and Figure 2 is the resulting images obtained.

```
image_coins = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 5 - IP\\IP5_v2\\IP5_v2\\coins.bmp").convert('L')

#To array
image_array_coins = np.array(image_coins)

#Binarization
binary_image_coins, otsu_threshold_coins = otsu_method(image_array_coins)

#Removing the internal holes
filled_image_coins = binary_fill_holes(binary_image_coins)

#Labeling
L = label(filled_image_coins)

#Plot results
fig, ax = plt.subplots(1, 4, figsize=(20, 15))
ax[0].imshow(image_coins, cmap='gray', norm=NoNorm())
ax[0].set_title("Original coins image.")
ax[0].axis('off')

ax[1].imshow(binary_image_coins, cmap='gray')
ax[1].set_title("Binary coins image.")
ax[1].axis('off')

ax[2].imshow(filled_image_coins, cmap='gray')
ax[2].set_title("Coins image with filled holes.")
ax[2].axis('off')

ax[3].imshow(L, cmap='gray')
ax[3].set_title("Coins image with labeled objects.")
ax[3].axis('off')
```

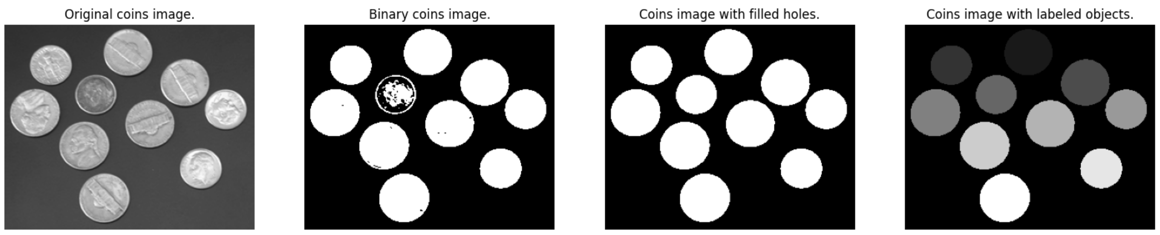
Original coins image.
Binary coins image.
Coins image with filled holes.
Coins image with labeled objects.

Fig. 2: Labeling on *coins* image.

Now we can obtain the properties of the objects, i.e. the surface areas of all connected components, using *regionprops* function. We store them in the variable *STATS*.

```
#Calculating the properties of the objects
STATS = regionprops(L)
```

The area data from *STATS* is now converted into the array *TAB*. This array, basically, contains the surface area of each connected component in the image. *TAB* is composed by the following areas:

$$2510, 1854, 2544, 1810, 2651, 1864, 2594, 2680, 1893, 2753.$$

```
#Converting the areas of a numpy array
TAB = np.array([region.area for region in STATS])
```

The plot of *TAB*, done as specified in the task, is in Figure 3. It shows the surface area of each object as a function of its index. As we can see that there are separate classes.

```
#Plot
plt.plot(TAB, linestyle='none', marker='*')
plt.xlabel('Index of the component')
plt.ylabel('Area')
plt.show()
```

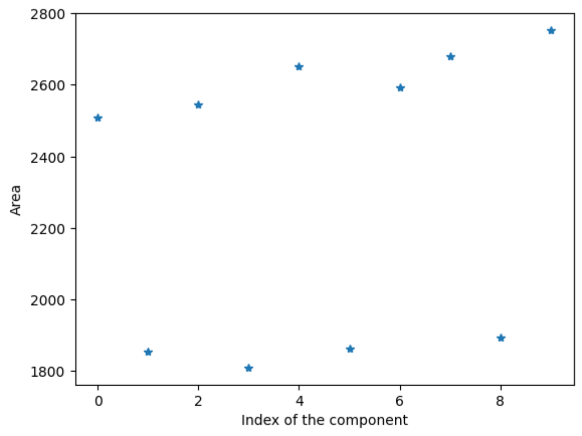

Fig. 3: Distribution of the areas of TAB.

Now we can select the indices of components whose surface areas fall within the range for Class 1. They are the following ones: 1, 3, 5, 8.

```
#Finding the indeces of the objets in Class 1
class1_idx = np.where((TAB >= 1500) & (TAB <= 2000))[0]
```

The number of objects belonging to Class 1, is then calculated using *size* function.

```
#Counting the objects in Class 1
class1_count = np.size(np.where((TAB >= 1500) & (TAB <= 2000))[0])
```

Here instead, we automatic calculate the number of objects using *sum*. We obtain that Class 1 has 4 objects, Class 2 has 6 objects.

```
#Counting automatically the coins for each class
class1_count = np.sum((TAB >= 1500) & (TAB <= 2000))
class2_count = np.sum((TAB >= 2300) & (TAB <= 2800))
```

In this final step we catch also the perimeters, always from *STATS*. The areas is equal to the array *TAB*, while the perimeters are the following ones:

$$184.50966799, 158.36753237, 186.16652224, 156.36753237, 189.92388155, 158.61017306, 187.92388155, 190.75230868, 16$$

```
#Extracting the areas and the perimeters
areas = np.array([region.area for region in STATS])
perimeters = np.array([region.perimeter for region in STATS])
```

We can now conclude the assignment estimating the radius from the areas and the perimeters. We use the following formulas:

- Radius from the Area A is defined as
$$R = \sqrt{\frac{A}{\pi}};$$
- Radius from the Perimeter P is defined as
$$R = \frac{P}{\pi}.$$

```
#Calculating the radius using the formulas
rad_area = np.sqrt(areas / np.pi)
rad_perimeter = perimeters / (2 * np.pi)

#Results
print("Radius calculated from the Area", rad_area)
print("Radius calculated from the Perimeter:", rad_perimeter)
```

The final outputs that we have are:

- $$28.26584183, 24.29293167, 28.45663983, 24.00293511, 29.04891578,$$
$$24.35835848, 28.73492378, 29.20737056, 24.54711011, 29.60248497$$
for the radius calculated from the Area;
- $$29.36562571, 25.2049756, 29.62932225, 24.88666572, 30.22732456,$$
$$25.24359307, 29.90901467, 30.35917283, 25.65513376, 30.67748272$$
for the radius calculated from the Perimeter.

References

[1] Community, S.O.: Add Extra Properties to Regionprops in Skimage. Accessed: 2024-12-08. https://stackoverflow.com/questions/63481913/add-extra-properties-to-regionprops-in-skimage

[2] Developers, S.-I.: Regionprops Example in Scikit-Image. (2023). Accessed: 2024-12-08. https://scikit-image.org/docs/0.24.x/auto_examples/segmentation/plot_regionprops.html

[3] Community, S.O.: Change Connection Definitions for Clear Border in Python. Accessed: 2024-12-08. https://stackoverflow.com/questions/70193514/change-connection-definitions-for-clear-border-in-python

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

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