Digital Image Processing - Lab Session 8

Sofia Noemi Crobeddu
student number: 032410554

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

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL
FACULTÉ DES SCIENCES
ET TECHNOLOGIE

January 19, 2025

Contents

Introduction

In this project segmentation and edge detection are analyzed in the context of image processing. Thresholding is used in both processes, and it is used Otsu's method.

1 Segmentation by thresholding

In this first section of the assignment, it is deepened the **Segmentation in images through thresholding technique**. In general, segmentation is used to divide an image into meaningful regions. The aim in segmentation specifically done with thresholding, is to separate the pixels of an image into two categories, i.e. intensities (values of pixels p) above and below a certain threshold T, in order to highlight certain regions of the image.
In this assignment this method is applied on *se000*, a grayscale image, and the steps for segmentation were taken from [1]. Since in the task it is suggested to use *graythresh()* and *im2bw()* functions that are for MatLab, Otsu's method is here chosen because it is the most similar one in Python coding.
First, the image is loaded and converted in grayscale. After, in order to reduce noise, a structuring element is applied for erosion, followed by a median filter. As in the research paper cited, the mean intensity is then computed to establish an initial threshold range with positive and negative offsets. This offsets is set to 25 after many trials. This mask is applied on the original array image, and later it is performed the difference between the original array image and the one just obtained, in order to maintain the objects of the picture and remove useless parts. Erosion and opening operations are applied again in order to segment better these objects. In addiction, Otsu's thresholding is applied in order to binarize the image just obtained, and additional morphological operations, such as median and Gaussian filtering are performed, to refine the binary mask, ensuring that unwanted artifacts were removed. Finally, a third thresholding is applied, still using the mean intensity as before.
For this first step of liver segmentation the code is shown below, and the resulting image is in Figure 1. As we can see, only the liver and the stomach are now present.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import NoNorm
from PIL import Image
from skimage.filters import threshold_otsu
from skimage.morphology import opening, closing, disk
import cv2

#Load image
image_se000 = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\assignment 8 - IP\\IP7\\IP7\\se000.jpg").convert('L') #Conversion to grayscale

#To array
image_array_se000 = np.array(image_se000)

#Structuring element for Erosion
struct_el_v4 = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))
#Erosion
eroded_v4_se000 = cv2.erode(image_array_se000, struct_el_v4, iterations=1)

#Median filter
median_filtered_se000 = cv2.medianBlur(eroded_v4_se000, 9)

#Calculating the average intensity for the threshold with a positive and negative offset
mean_intensity_se000 = np.mean(median_filtered_se000)
upper_threshold_se000 = mean_intensity_se000 + 25  #Positive offset
lower_threshold_se000 = mean_intensity_se000 - 25  #Negative offset

#Creating the first binary mask based on the first threshold
binary_mask_1 = np.where((median_filtered_se000 >= lower_threshold_se000) & (median_filtered_se000 <= upper_threshold_se000), 1, 0).astype(np.uint8)

#Finding the largest contour
largest_contour_se000 = image_array_se000 * binary_mask_1
largest_contour_se000 = image_array_se000 - largest_contour_se000 #Difference
largest_contour_se000 = cv2.erode(largest_contour_se000, struct_el_v4, iterations=2) #Second erosion
largest_contour_se000 = opening(largest_contour_se000, disk(3)) #Opening

#Function for Otsu's thresholding
def otsu_method(img):
```

```python
    otsu_threshold = threshold_otsu(img)
    #Binarization based on Otsu's method
    binary_image = (img > otsu_threshold).astype(np.uint8)
    return binary_image, otsu_threshold

#Binarization: Otsu's thresholding is used in place of MatLab function graythresh()
BW_se000, level_se000 = otsu_method(largest_contour_se000) #Second thresholding
BW_se000 = image_array_se000 * BW_se000

#Application of a median filter with a 49x49 window
median_filtered2_se000 = cv2.medianBlur(BW_se000, 49)

#Application of a Gaussian filter with a 51x51 window
gaussian_filtered2_se000 = cv2.GaussianBlur(median_filtered2_se000, (51, 51), 0)

#Calculating the average intensity for the third threshold
mean_intensity2_se000 = np.mean(gaussian_filtered2_se000)
threshold3_mask = np.where(gaussian_filtered2_se000 > mean_intensity2_se000, 1, 0).astype(np.uint8)

#Cleaning the mask with morphological operations
struct_el_large = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (15, 15))
largest_contour2_se000 = closing(threshold3_mask, struct_el_large) #Closing
largest_contour2_se000 = opening(largest_contour2_se000, struct_el_large) #Opening

#Application of the final mask to the image
segmented_se000 = cv2.bitwise_and(image_array_se000, image_array_se000, mask=largest_contour2_se000.astype(np.uint8))

#Plot
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax[0].imshow(image_se000, cmap='gray', norm=NoNorm())
ax[0].set_title("Se000 image.")
ax[0].axis('off')

ax[1].imshow(segmented_se000, cmap='gray')
ax[1].set_title("First segmentation of se000 image.")
ax[1].axis('off')

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

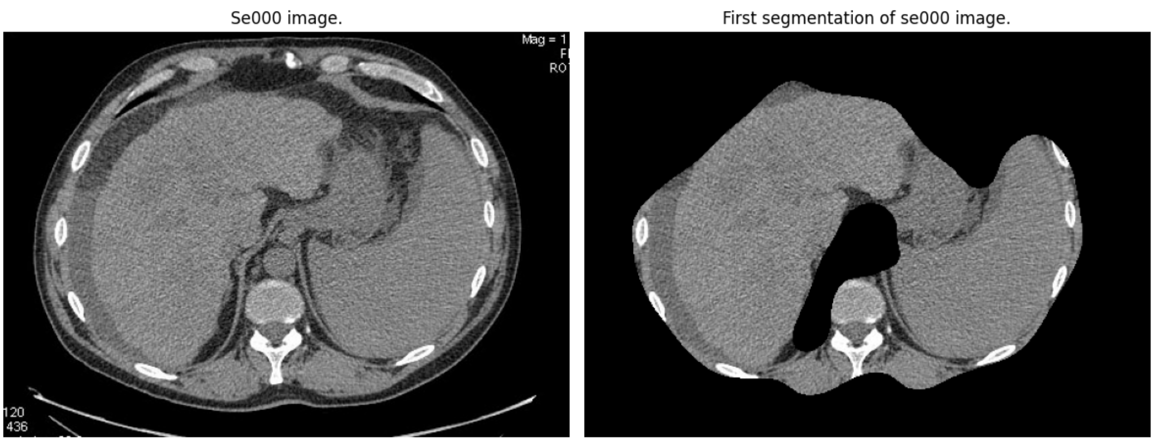

Fig. 1: First step of segmentation in *se000* image.

In order to remove the stomach, a modified bisector is drawn to isolate regions of interest. The idea (i.e. personal method) is that pixels below that line will be put to zero. The code for drawing the modified bisector is shown below and the image with it is in Figure 2.

```python
#Dimensions of the image
height, width = segmented_se000.shape

#Creating a horizontal bisector (horizontal central line)
segmented_se000_delimited = segmented_se000.copy()

```

```python
#Bisector coordinates (after many trials)
start_point = (180, height - 90)  #Starting point
end_point = (width + 40, -100)   #Ending point

#Drawing the bisector
cv2.line(segmented_se000_delimited, start_point, end_point, (255, 0, 0), 1)

#Plot
plt.imshow(segmented_se000_delimited, cmap='gray')
plt.show()
```

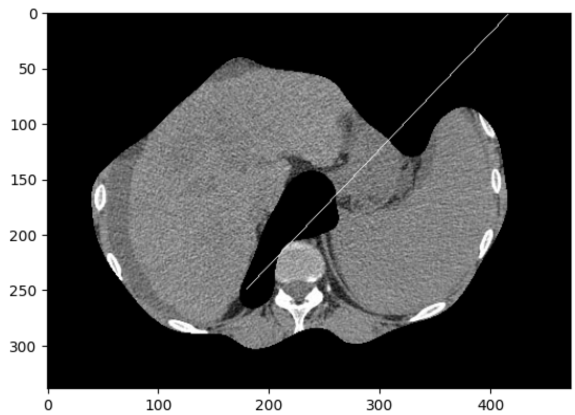

Fig. 2: Segmented *se000* image with modified bisector.

Now that the stomach area is delimited, it has to be zeroed. Here there is the code for transforming that area of pixels in black (i.e. equal to 0). Image resulting is in Figure 3.

```python
#Zero out the pixels below the bisector (part to the right of the line)
for y in range(height):
    for x in range(width):
        #Condition: only if the point is below the bisector
        if x > (y - start_point[1]) * (end_point[0] - start_point[0]) / (end_point[1] - start_point[1]) + start_point[0]:
            segmented_se000_delimited[y, x] = 0

#Application of erosion
segmented_se000_delimited = cv2.erode(segmented_se000_delimited, struct_el_v4, iterations=3)

#Application of Otsu's thresholding
segmented_se000_delimited, _ = otsu_method(segmented_se000_delimited)

#Application of the binary mask with the original image
segmented_liver_se000 = image_array_se000 * segmented_se000_delimited

#Plot of the modified image with zero values below the bisector
plt.imshow(segmented_liver_se000, cmap='gray')
plt.axis('off')
plt.show()
```

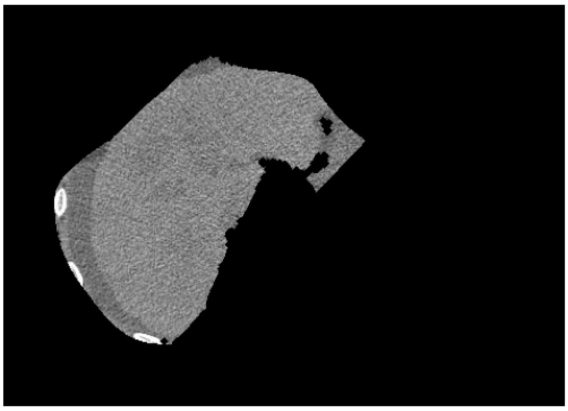

Fig. 3: New segmented *se000* image.

Finally, the process just done is repeated again for refinement. The final segmented liver is then shown in Figure 4.

```
#Dimensions of the image
height, width = segmented_liver_se000.shape

#Creating a new horizontal bisector
se000_new_bisector = segmented_liver_se000.copy()
#New bisector coordinates (lower-right position)
start_point = (255, height - 195)
end_point = (width + 30, -200)

#Drawing the bisector
cv2.line(se000_new_bisector, start_point, end_point, (255, 0, 0), 1)

#Setting all values below the bisector to 0
segmented_liver2_se000 = segmented_liver_se000.copy()

#Zero out the pixels below the bisector as done before
for y in range(height):
    for x in range(width):
        #Condition: only if the point is below the bisector
        if x > (y - start_point[1]) * (end_point[0] - start_point[0]) / (end_point[1] -
    start_point[1]) + start_point[0]:
            segmented_liver2_se000[y, x] = 0

#Plot
plt.imshow(segmented_liver2_se000, cmap='gray')
#plt.title("")
plt.axis('off')
plt.show()
```

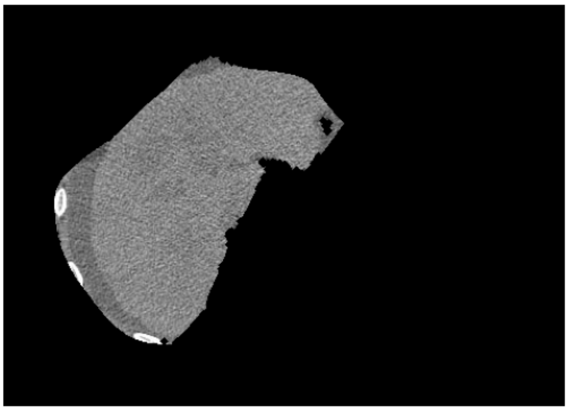

Fig. 4: Segmented liver in *se000* image.

2 Edge function

In this second section of the assignment, it is deepened the **Edge detection technique**. It is a method used to extract edges in an image by identifying significant transitions in light intensity. There are two basic steps:

- maxima extraction, that is to identify regions where intensity changes are most prominent;
- thresholding in order to select only salient edges.

The first image on which it is applied, is T image (grayscale). Since the function *edge()* is for MatLab, in Python other similar techniques are used, such as Sobel, Prewitt, Canny. For this specific first step of second section, Sobel and Prewitt are applied before binarization through thresholding with Otsu's method. Sobel calculates the brightness gradient in both directions, horizontal and vertical, and provides greater noise attenuation than Prewitt. This last one, instead, uses simpler kernels which may be more sensitive to noise.
The code is shown below and the resulting images are in Figure 5. As we can notice, in this case Sobel and Prewitt produce similar results, since they are both local derivative operators which detect changes in intensity along gradients.

```
from skimage import filters

#Load image
image_T = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\assignment 8
     - IP\\IP7\\IP7\\T.png").convert('L') #Conversion to grayscale

#To array
image_array_T = np.array(image_T)

#Since edge() function is for MatLab, here in Python we can use Canny, Sobel, Prewitt and other
    methods.
#Maxima extraction: Application of Sobel for edge detection
edges_sobel_image_T = filters.sobel(image_array_T)
#Maxima extraction: Application of Prewitt for edge detection
edges_prewitt_image_T = filters.prewitt(image_array_T)

#Thresholding: Binarization to get the edges more clearly
binary_sobel_image_T, otsu_threshold_sobel_T = otsu_method(edges_sobel_image_T)
binary_prewitt_image_T, otsu_threshold_prewitt_T = otsu_method(edges_prewitt_image_T)

#Plot
fig, ax = plt.subplots(1, 3, figsize=(18, 6))

#Original image
ax[0].imshow(image_array_T, cmap='gray', norm=NoNorm())
ax[0].set_title("T image.")
ax[0].axis('off')

#Edges of T image - Sobel
ax[1].imshow(binary_sobel_image_T, cmap='gray')
ax[1].set_title("Edges of T image after Sobel and Otsu thresholding.")
ax[1].axis('off')

#Edges of T image - Prewitt
ax[2].imshow(binary_prewitt_image_T, cmap='gray')
ax[2].set_title("Edges of T image after Prewitt and Otsu thresholding.")
ax[2].axis('off')

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

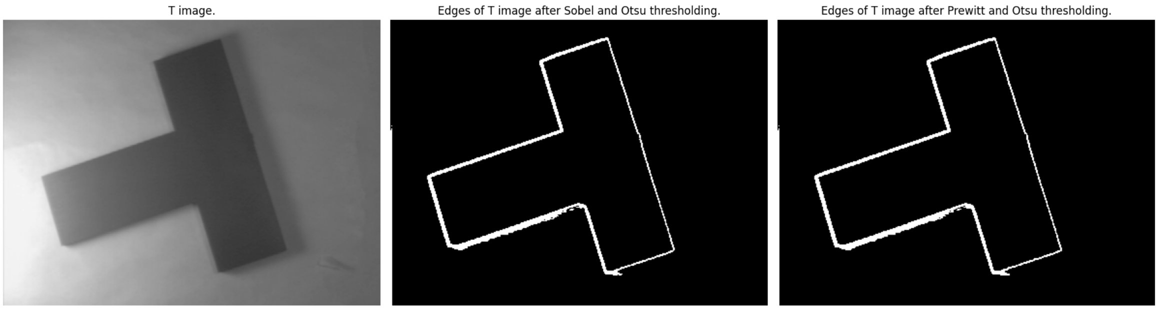

Fig. 5: Edge detection in T image with Sobel and Prewitt.

In the second part of this section, edge detection is applied on *bricks* image, that contains colored blocks.
As in Assignment 3, in order to detect the edges of blue and red bricks, the image is converted to HSV (Hue, Saturation and Value) color space: Hue channel is in position 0 and Saturation in 1. Value channel is not considered since it is useless for this analysis.
To find blue and red masks, it is necessary to limit the Hue channel. Blue and red bounds are calculated based on the Hue system in degrees, from 8 (Appendix). As it is evident, blue is more or less between 194 to 252 degrees, while red has two intervals: the first one between 0 and 60 degrees, the second one between 315 and 360. Hence, relative bounds are obtained simply by dividing the bounds expressed in degrees by 360.
Regarding the level of saturation, instead, the minimal level is maintained at 0.1 after some trials.
Edges detection is here performed using Canny, since it produces thinner edges in the objects. Canny is applied on the two masks, and edges for blue and red objects are obtained separately.
The last part of the task in this section, asked to catch salient edges. It is solved through two different methods.

- Method 1:
  Bricks image is first transformed into grayscale using *rgb2gray* function. After this, Canny edge detection is applied in order to perform maxima extraction. Finally, as done before, binarization through Otsu's thresholding is applied and salient edges are caught.
- Method 2:
  here the three RGB channels are first extracted separately. After, Canny is applied, still separately, on each color channel. Finally, it is performed a combination of the salient edges caught separately, using maximum operation between channels.

The code implemented is shown below, and the resulting images are in Figure 6. As we can notice, in the detection of salient edges, the methods produce a similar result, but the second one is more detailed.

```
from skimage import feature
from skimage.color import rgb2hsv
from skimage.color import rgb2gray

#Load image
image_bricks = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
     assignment 8 - IP\\IP7\\IP7\\bricks.jpg")

#To array
image_array_bricks = np.array(image_bricks)

#Conversion RGB to HSV
#- Hue : [0 ,1]. As hue increases from 0 to 1, the color transitions from red to orange, yellow,
     green, cyan, blue, magenta, and finally back to red.
#- Saturation : [0 ,1]. 0 = neutral shade , 1 = maximum saturation.
#- Value : maximum value among the red, green, and blue components of a specific color.
hsv_bricks = rgb2hsv(image_array_bricks)
#Extraction of Hue and Saturation channels
hue = hsv_bricks[:, :, 0]  #Hue channel
saturation = hsv_bricks[:, :, 1] #Saturation channel

```

```
#Thresholds for blue in HSV space:
#The Hue range has 360 degrees. The color blue goes more or less between 194 to 252 degrees . Hence, 194/360 = 0.54 , while 252/360 = 0.70.
lower_hue_blue = 0.54
upper_hue_blue = 0.70

#Thresholds for red in HSV space:
#Red colo has two intervals:
#- the first interval goes more or less between 0 to 60 degrees . Hence, 0/360 = 0.0 , while 60/360 = 0.17;
#- the second interval goes more or less between 315 to 360 degrees . Hence, 315/360 = 0.86 , while 360/360 = 1.
lower_hue_red1 = 0.0
upper_hue_red1 = 0.05
lower_hue_red2 = 0.875
upper_hue_red2 = 1.0

minimum_saturation = 0.1 #Minimum saturation to exclude white/bright pixels --> 0.1 after some trials.

#Masks for blue and red points in the image
blue_mask = (hue >= lower_hue_blue) & (hue <= upper_hue_blue) & (saturation >= minimum_saturation)
red_mask = ((hue >= lower_hue_red1) & (hue <= upper_hue_red1) | (hue >= lower_hue_red2) & (hue <= upper_hue_red2)) & (saturation >= minimum_saturation)

#Here we use Canny method: application of Canny for edge detection
edges_blue = feature.canny(blue_mask)
edges_red = feature.canny(red_mask)

#Now for the last part of the task regarding salient edges detection of the entire image, we use 2 methods.
#METHOD 1:
#First we transform the image into grayscale
gray_image_bricks = rgb2gray(image_array_bricks)

#Maxima extraction: Application of Canny for edge detection
edges_image_bricks = feature.canny(gray_image_bricks)

#Thresholding: Binarization to get the edges more clearly
binary_edges_image_bricks, otsu_threshold_edges_bricks = otsu_method(edges_image_bricks)

#METHOD 2:
#We extract separately the channels from the image array
R_bricks = image_array_bricks[:, :, 0] #red
G_bricks = image_array_bricks[:, :, 1] #green
B_bricks = image_array_bricks[:, :, 2] #blue

#Maxima extraction: Application of Canny for edge detection
edges_R_bricks = feature.canny(R_bricks)
edges_G_bricks = feature.canny(G_bricks)
edges_B_bricks = feature.canny(B_bricks)

#Now we combine the three edges from the three channels, in order to obtain the final edge
all_edges_bricks = np.maximum(np.maximum(edges_R_bricks, edges_G_bricks), edges_B_bricks)
#Thresholding: Binarization to get the edges more clearly
all_edges_bricks, tr_method2 = otsu_method(all_edges_bricks)

#Plot
fig, ax = plt.subplots(2, 3, figsize=(18, 12))

#Original image
ax[0, 0].imshow(image_array_bricks)
ax[0, 0].set_title("Bricks image.")
ax[0, 0].axis('off')

#Blue edges
ax[0, 1].imshow(edges_blue, cmap='gray')
ax[0, 1].set_title("Blue edges detection in bricks image.")
ax[0, 1].axis('off')

#Red edges
ax[0, 2].imshow(edges_red, cmap='gray')
ax[0, 2].set_title("Red edges detection in bricks image.")
ax[0, 2].axis('off')

#Edges METHOD 1
ax[1, 0].imshow(binary_edges_image_bricks, cmap='gray')
ax[1, 0].set_title("Edges detection in bricks image with Canny.")
ax[1, 0].axis('off')

#Edges METHOD 2
```

```
ax[1, 1].imshow(all_edges_bricks, cmap='gray')
ax[1, 1].set_title("Edges detection in bricks image with combination of channel edges.")
ax[1, 1].axis('off')

#Hiding the last subplot since it is not present
ax[-1, -1].axis('off')

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

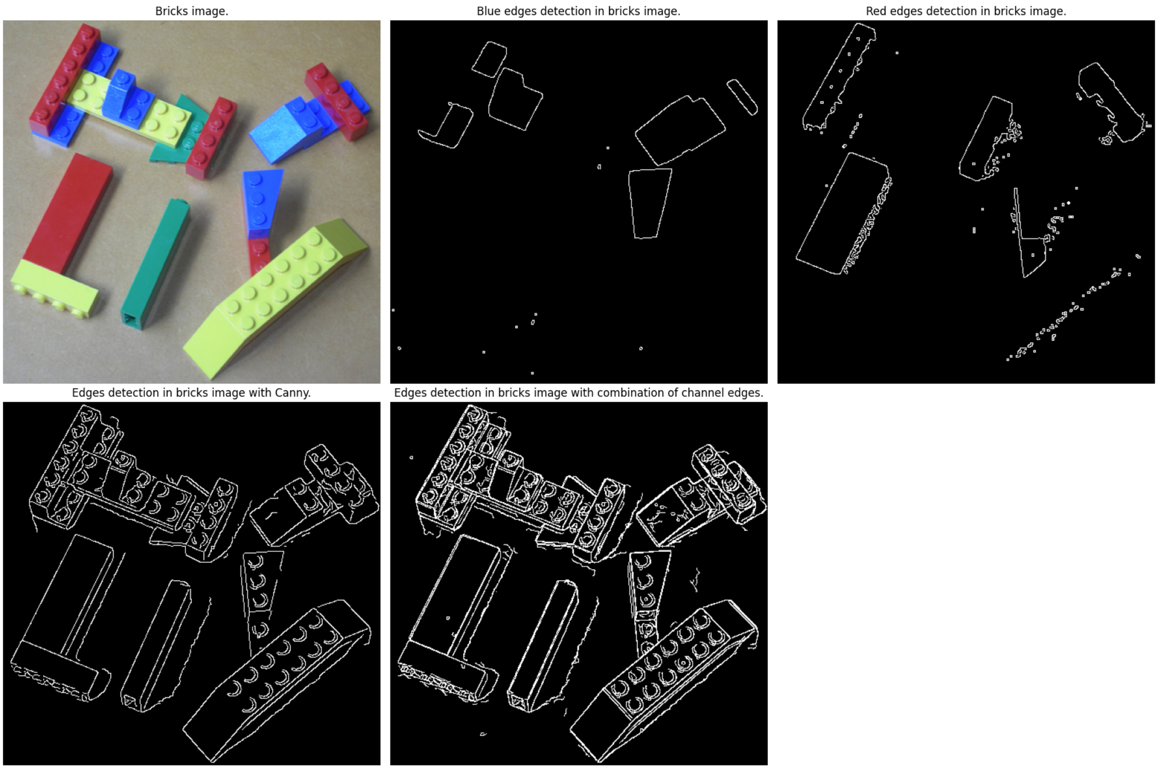

Fig. 6: Edge detection in *bricks* image.

3 Canny detector

For this last section, another type of technique is applied on T image, still for edge detection as in previous one. Before, Sobel and Prewitt were applied on this specific image. Here, as for *bricks* image, Canny is applied in order to detect the edges. As we can see from Figure 7, compared to 5 we have slimmer edges, but the result is really the same of Sobel and Prewitt application. Thresholding is still performed useing Otsu's method.

```
#Maxima extraction: Application of Canny for edge detection
edges_canny_image_T = feature.canny(image_array_T)

#Thresholding: Binarization to get the edges more clearly
binary_canny_image_T, otsu_threshold_canny_T = otsu_method(edges_canny_image_T)

#Plot
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

#Original image
ax[0].imshow(image_array_T, cmap='gray', norm=NoNorm())
ax[0].set_title("T image.")
ax[0].axis('off')

#Edges of T image - Canny
ax[1].imshow(binary_canny_image_T, cmap='gray')
ax[1].set_title("Edges of T image after Canny and Otsu thresholding.")
```

```
ax[1].axis('off')
```

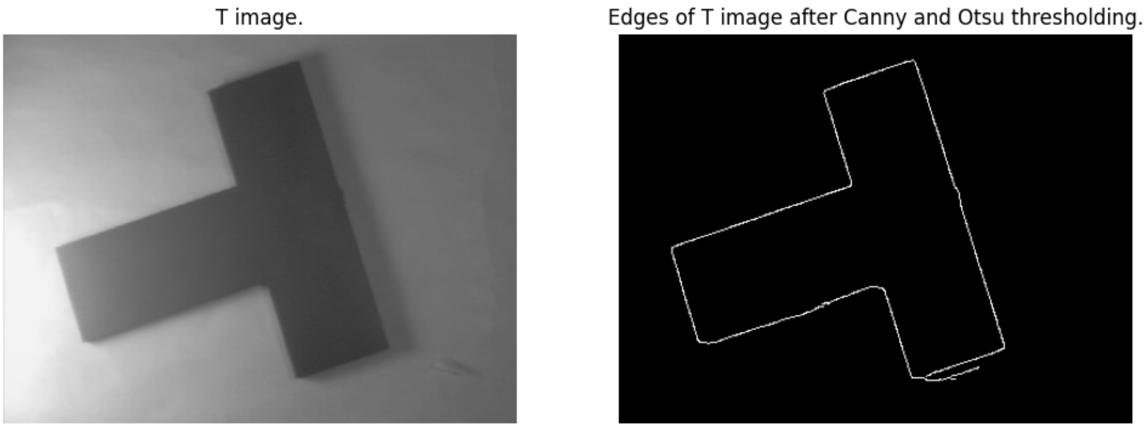

Fig. 7: Edge detection in T image with Canny.

Appendix

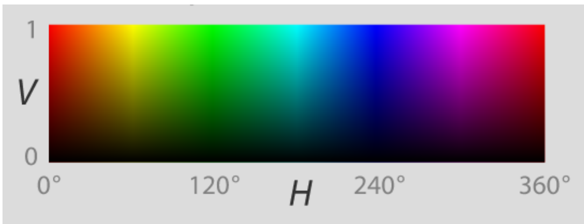

Fig. 8: Hue system color.

References

[1] Sadeque, Z.A., Khan, T.I., Hossain, Q.D., Turaba, M.Y.: Automated detection and classification of liver cancer from ct images using hog-svm model. ResearchGate (2019)

[2] Overflow, S.: An OpenCV function similar to MATLAB's graythresh (2013). https://stackoverflow.com/questions/16057023/an-opencv-function-similar-to-matlabs-graythresh

[3] Overflow, S.: MATLAB - How to Detect Green Color on Image (2016). https://stackoverflow.com/questions/37684903/matlab-how-to-detect-green-color-on-image

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