Universitat Autònoma de Barcelona

Fundamentals Of Computer Vision
Project Report

Morphological operators and Feature Detectors

Students:
Júlia Garcia, 1630382
Josep Maria Rocafort, 1636790

Professor:
Jorge Bernal

$9^{th} december 2022$

Contents

1 Introduction

The purpose of this report is to provide efficient solutions for two different problems: Pasta Image and Product Image. During the process, knowledge acquired in class is applied by using Morphological operators and Feature Detectors. The following sections explain in detail the methods that had been used in each case to achieve the desired result.

2 Pasta image

2.1 Problem

In this problem, we were given an image in which different types of pasta appeared. The goal of this first challenge was to count the number of elements of each type and provide separate images with only one kind of pasta.

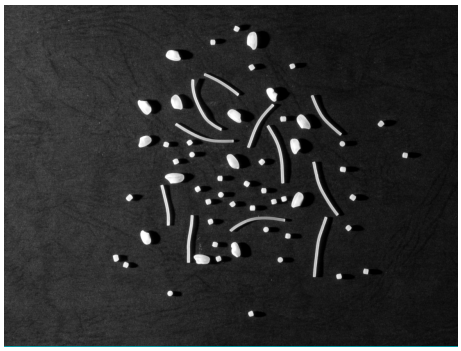


Figure 1: Original Pasta Image

2.2 Obtaining isolate images

In this section, we provide the steps we followed to obtain the desired images.

First of all, we needed a defined separation between the objects and the background, so we converted the original pasta image (Grayscale Image) to Binary (pixels can have one of exactly two colors, in this case, black and white).

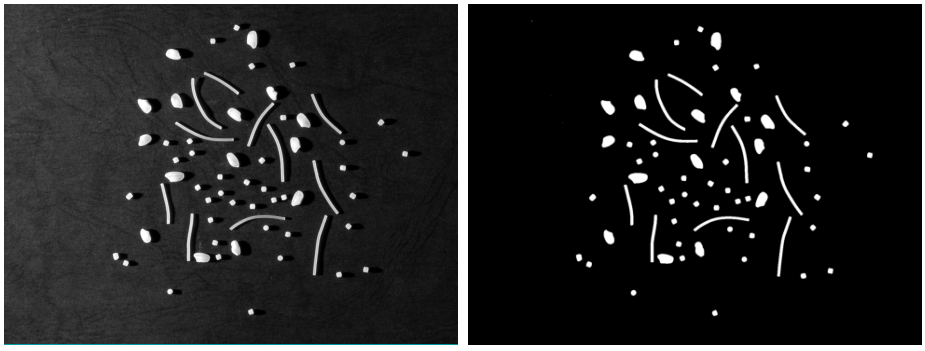


Figure 2: Original Pasta Image (left) and Binary Image (right)

At this point of the project, we started to apply morphological operations to the obtained Binary Image. The idea was to compute an Erosion with different Kernels depending on the type of pasta.

We started with the bigger type of pasta (grains of rice). As said, we computed an Erosion followed by a Dilation (to recover the loss of pasta size in the previous operation), in other words, we computed an Opening to the Binary Image.

Listing 1: Obtaining Grains of Rice Image

```
kernel = np.ones((17, 17), np.uint8)

img_erosion = cv2.erode(blackAndWhiteImage, kernel, iterations=1)
pasta = cv2.dilate(img_erosion, kernel)
```

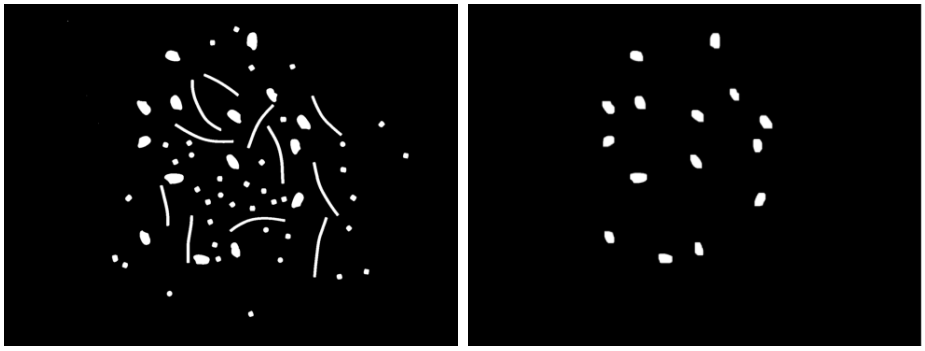


Figure 3: Binary Image (Input) and Grains of Rice Image (Output)

Then, we simplified the problem by computing a subtraction of both previous images. In this way, we obtained the original image without the grains of rice.

Figure 4: Resulting image after Substraction

We repeated the Erosion-Dilation process, but this time we applied the operations to the newly obtained image in order to get rid of spaghetti elements. Therefore, we obtained an image with only Grains of Quinoa.

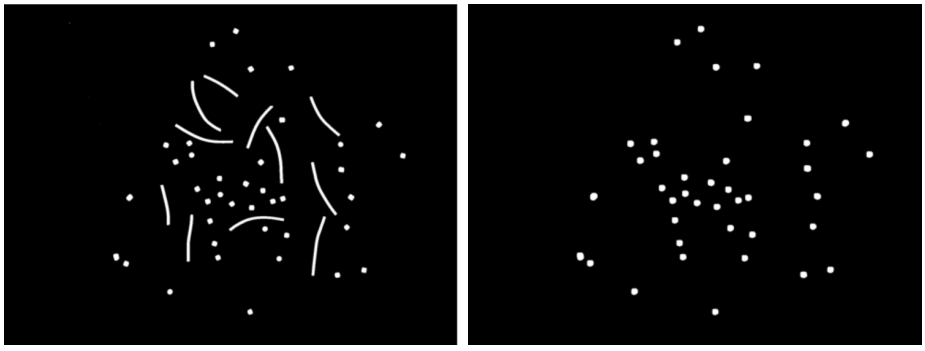


Figure 5: Binary Image (Input) and Grains of Quinoa Image (Output)

Finally, to obtain the spaghetti image, we computed a Subtraction between both previous images.

Figure 6: Resulting image after Substraction

2.3 Counting the number of elements

In this section, we provide the steps we followed to obtain the number of elements of each kind.

In order to perform the task, we decided to use Connected Component Labeling (Blob detection for Binary Images). This is an algorithm to detect and count the number of connected regions (blobs) in a binary image. We used 8-Connectivity to make sure that pixels with touching edges or corners were connected, and assumed that no elements were in contact.

Listing 2: Connected Component Labeling

```
def blob_detection(Image, StructingElement):
    ...
    return IDS, cnt_label
```

The explained algorithm was applied to the following images.

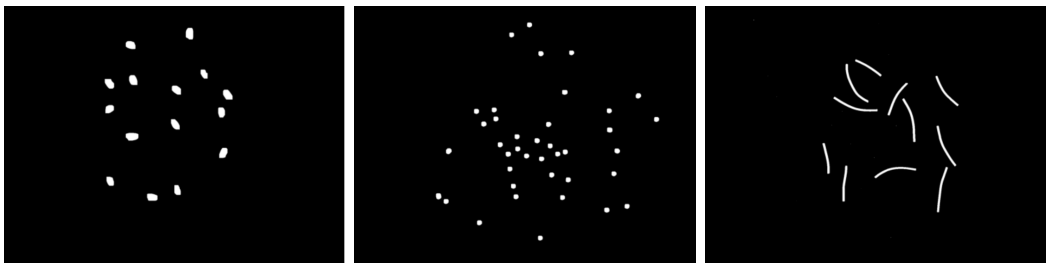


Figure 7: Separated pasta images

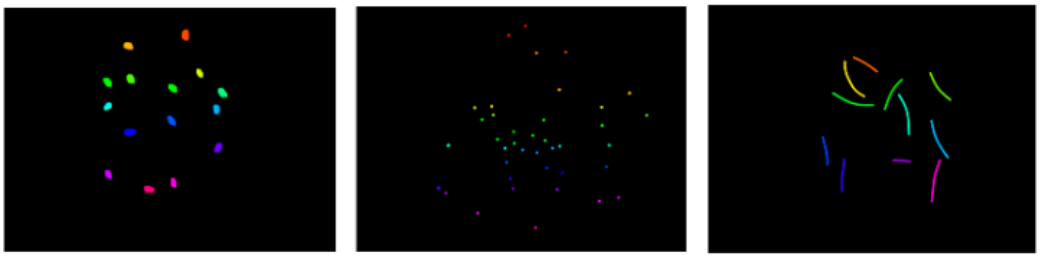
After this process, we realized that the results were not as expected because the images were noisy. So, we tried applying blurring and smoothing techniques, but it didn't work. For this reason, we applied a threshold where only elements with bigger areas were considered.

Listing 3: Area Thresholding

```
def area_threshold(IDS, pix_thr):
    ...
    return IDS_new, cnt_newidx-1
```

The obtained output was as follows:

```
detected 15 pasta objects in image
detected 39 grains objects in image
detected 11 spaguetti objects in image
```



3 Product image

In this problem, we were given the following product image. The main goal of this first challenge was to obtain the reference number that appears on the image.

Figure 8: Reference Product Image

3.1 Localizing and detecting the numbers

In this section, we provide the steps we followed to localize the numbers on the input image.

First of all, we manually extracted a cutout of the Reference Product Image where the text "REF" appeared. The image obtained would be used to locate the desired numbers in any input image since they are always accompanied by "REF". *Note that this image is saved in the python script in a variable as an array of arrays, the purpose of this is to have the script be standalone and avoid unnecessary processing and memory reading the image from a jpg.*

Figure 9: REF Image

We then used OpenCV for template matching along the entire input image. The point with the highest correlation would be where the text REF was in the input image. Then, using a hard-coded shape window we extracted an image with the numbers to be analyzed at the detected reference point. *We assume that input images scale and rotation is constant*

Listing 4: Template Macthing

```
res = cv.matchTemplate(image,template,cv.TM_CCOEFF_NORMED)
```

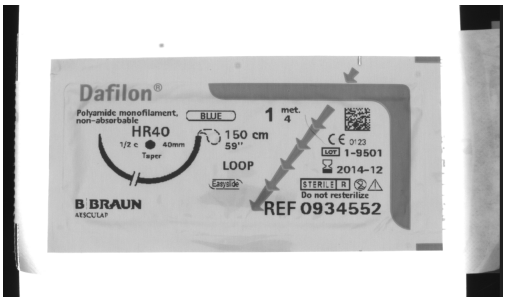


Figure 10: Input Image

Figure 11: Detected number to analyze (Binary Image)

3.2 Identify the numbers

In this section, we provide the steps we followed to identify the value of each detected number.

We wanted a separate image for each number. This process to cut out the digits consisted in squishing the obtained binary image into an array with the sum of pixel values in every column, with this array we could detect spaces where it was non-zero, thus our numbers.

Figure 12: Some obtained digits

The approach we decided to use to check what number we had was to check the similarity of each cutout digit with our templates. This measure is the mean of the pixels that are the same in both images.
To be able to perform this approach we need to have the digit image the same shape as the templates. Therefore, we first cut out the edges to the digit and all the templates. Then, we added borders/padding to each image where the axis was smaller than the one to match.

Listing 5: Obtaining template shaped image

```
def crop_to_edges(image):
    ...
    return fin_image
```

Finally, to perform the template matching we defined a function that detects what number we have; for a provided digit image and list of template images it returns the similarity with all templates, the image of the decided matching template and what number that is.

Listing 6: Deducing the number

```
def eval_digits(digit,templates):
    ...
    return eval_digit, templates[match], match
```

The obtained result was satisfactory, so we exactly obtained the number that appeared in the input image.

Figure 13: Program final output

4 Conclusions

The purpose of this project was to apply the knowledge learned in class and propose a solution for both suggested problems. Although we had some complications, we achieved the goal in both cases. Therefore, based on the obtained results, we can conclude that we have been able to understand and apply some of the algorithms that are used in Image Processing. We are happy with the result, nevertheless, we believe we could improve the execution of the solutions in the future.