Project 1

Isabela Telles Furtado Doswaldo - RA: 170012

MC920 - Introduction of Digital Image Processing
University of Campinas

1 Introduction

The goal of the project is to cover subjects of Digital Image Processing, such as the definition of object contour, the image description and classification through its properties, and put it into practice. Therefore, some techniques were applied in Python, along with the following libraries: OpenCV, Scikit Image, Matplotlib and Numpy.

The images used for testing are available in http://www.ic.unicamp.br/~helio/imagens_objetos_coloridos/ and the results presented here are outputs of the program when it was runned with the image *objetos1.png* as argument.

2 Methodology

2.1 Execute Program

First of all, it was necessary to define an argument required to receive the path to the image file. The image should be in the working directory, otherwise a full path of it should be given.

See an example of usage to execute the program if the image is in the working directory:

```
$ python3.5 trabalho_1.py objetos1.png
```

2.2 Read Image

To load the image from a file into a 3D numpy array the OpenCV library's function *cv2.imread()* was used.

2.3 Change Color Space

For some applications of image processing, a color image is not ideal to identify contours and other important features. In that case, it is more appropriate to change its color space to grayscale.

By default, color image loaded by OpenCV is in BGR mode. Then, in order to convert the image to gray color space, the function *cv2.cvtColor()* was used with the flag `cv2.COLOR_BGR2GRAY`.

It is possible to compare the original image with the grayscale image result in Figure 1 e 2, respectively.

2.4 Contours of the Objects

Contours are abstract collections of continuous points along the boundary of objects in an image, having the same color or intensity. They are really useful in shape analysis and object detect recognition.

In order to create an image with the contours of the objects, the first step was to convert the grayscale image, obtained in Subsection 2.3, to a binary image, which provided a better accuracy for the final result. The function used was *cv2.threshold()*, with thresholding type `cv2.THRESH_BINARY_INV`, that assigned a black value to pixels greater than the threshold, and

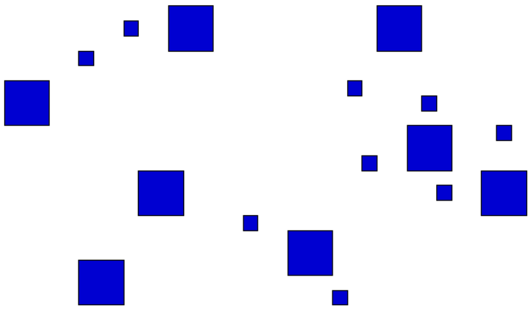
Figure 1: Original Image

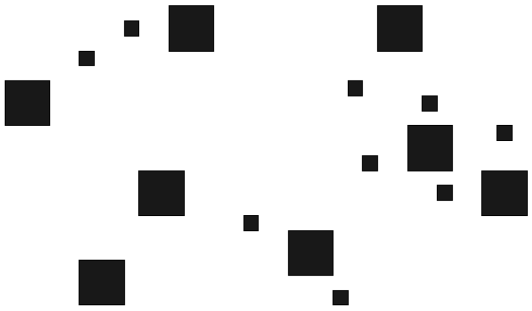
Figure 2: Grayscale Image

a white value to pixels lesser than or equal to threshold. The choice of this type of thresholding was made based on the images available for testing, since all of them had a white background color. Otherwise, if the background of the images were black, the thresholding type necessary would be `cv2.THRESH_BINARY`.

The next step was to actually find the contours in the binary image. Using the function *cv2.findContours()* with the contour retrieval mode as `cv2.RETR_EXTERNAL`, that retrieves only the extreme outer contours, and with the contour approximation mode as `cv2.CHAIN_APPROX_SIMPLE`, that compresses horizontal, vertical, and diagonal segments and leaves only their end points, it was possible to acquire all the contours in the image. The contours were stored in a list, which had each element of it as a numpy array of (x,y) coordinates of boundary points of the object.

The last step consisted in drawing the contours found in a new white image. The Numpy's functions *numpy.empty()* and *numpy.shape()* were used to create an empty 3D numpy array with the same shape of the original image. Since a list of contours was already determined, it was possible to use the function *cv2.drawContours()* to draw red contours of thickness 2 in the white background. Figure 3 shows the final result.

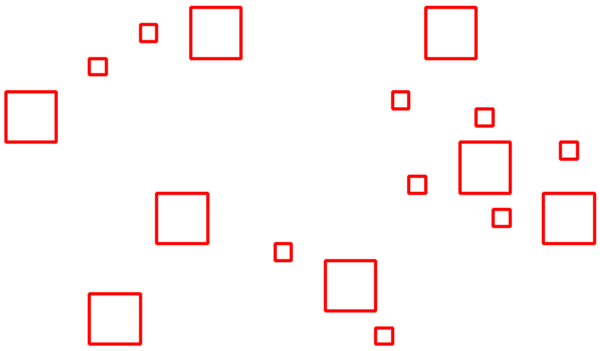
Figure 3: Objects' Contours

2.5 Objects' Properties Extraction

There are some features, such as area, perimeter, centroid, bounding box and etc., that can be extracted from objects of an image when the regions are properly detected. These measured properties can be used, for instance, to classify the objects of the image as you will be able to see in Subsection 2.6.

Using the Scikit Image library, it was possible to apply the function *skimage.measure.label()* in the binary image, obtained in Subsection 2.4, which returned a labeled image, that is, an image in which the foreground regions each have a numeric label and the background regions all have label zero. The connected regions were found by the function mentioned above using the concept of connectivity, which says two pixels are connected when they are neighbours and have the same value.

Then, for each region, the perimeter and the area were listed:

```
region:  0          perimeter:  190        area:  2352
region:  1          perimeter:  190        area:  2352
region:  2          perimeter:  62         area:  272
region:  3          perimeter:  62         area:  272
region:  4          perimeter:  188        area:  2304
region:  5          perimeter:  62         area:  272
region:  6          perimeter:  64         area:  289
region:  7          perimeter:  190        area:  2352
region:  8          perimeter:  64         area:  289
region:  9          perimeter:  64         area:  289
region:  10         perimeter:  190        area:  2352
region:  11         perimeter:  190        area:  2352
region:  12         perimeter:  64         area:  289
region:  13         perimeter:  62         area:  272
region:  14         perimeter:  188        area:  2304
region:  15         perimeter:  190        area:  2352
region:  16         perimeter:  62         area:  272
```

And Figure 4, as a complement, shows each region labeled. To write the number of each region in the image, it was necessary to use the following function of Matplotlib: *matplotlib.pyplot.text()*.

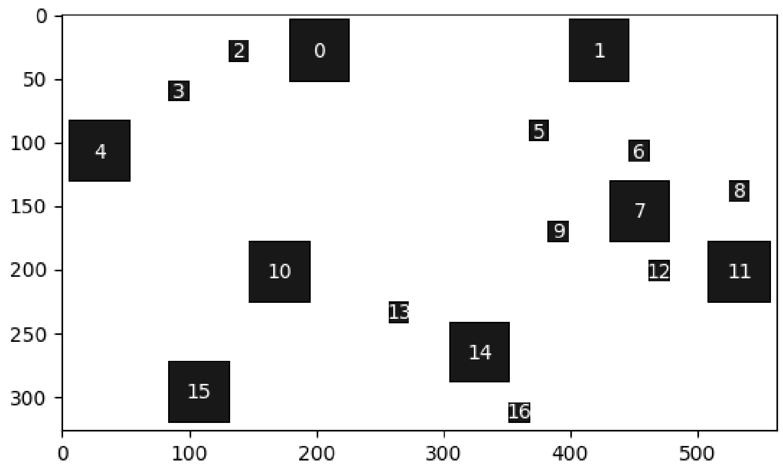

Figure 4: Objects' Contours

2.6 Histogram of Objects' Area

An image analysis can be done by an object-based classification. It was possible, for example, to classify the objects of the image in Figure 1 by their area. Regions with area lesser than 1500 pixels were classified as small; regions with area greater than or equal to 1500 pixels and lesser than 3000 were classified as medium; regions with area greater than 3000 were classified as big.

In order to do that classification, it was used a histogram of objects' area. The function *matplotlib.pyplot.hist()* created this histogram, showed in Figure 5, since a numpy array of objects' areas and an array of bins (0, 1500, 3000, maximum area) were provided. An array of the number of objects in each class (small, medium, big) were returned by the function, as it is possible to verifiy below:

```
number of small regions:  9
number of medium regions:  8
number of big regions:  0
```

The histogram was organized in three columns, each one correspondingly to one of the three classes.

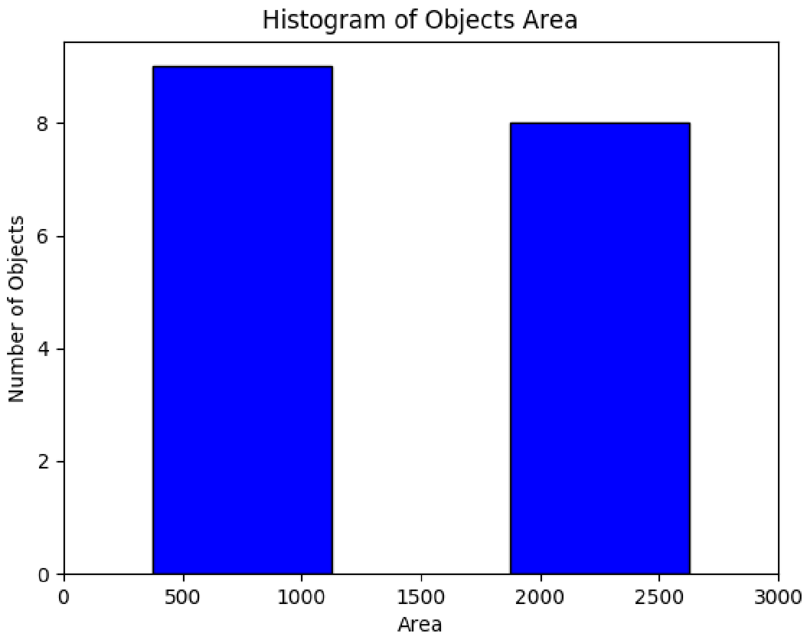

Figure 5: Histogram of Objects' Area