Project 3 - Measures

Thales Mateus Rodrigues Oliveira (RA 148051)

***Abstract*—In this project, it was given the task of calculating measures of objects within digital images. To fulfill the requirements, the solution implements four different algorithms, one for each required task from the project description[1]. It was able to generate the outputs needed, and the implementation decisions are explained in this report.**

I. Introduction

The capability of identifying objects within digital images is a great step for automatic processing, segmentation, acquirement of statistic data, and so on. In order to obtain measures from objects within an image, some previous processing can be done to facilitate the calculations, such as identification of the contours of the objects. The goal of this work is to implement algorithms which return measures from objects within an image, and the auxiliar functions to do so. The next sections explains the implemented algorithm, the experiments realized and the output analysis

The code, along with the input files and the report is delivered in the compressed file THALES_MATEUS_RODRIGUES_OLIVEIRA_148051.tar, in the Google Classroom.

II. The Program

The program was implemented with Python 3.7.3. The libraries used and their respective versions are OpenCV 4.1.0, Numpy 1.16.4 and MatPlotLib 3.1.0.

A. How to execute it

The project has a Makefile available to help performing some actions on it. The Makefile has 3 basic instructions: clean, build and exec. Clean instruction removes generated images stored in the **output** folder, the execution code in **bin** folder and the folders itself. The Build instruction creates the **output** and **bin** folders, and moves the source code to **bin**. The Exec instruction executes the code with images in the **input** folder. Listing 1 provides examples of how to execute the three instructions in a terminal.

```
#clean environment, deletes output and bin folders
   and their content
make clean

#prepare the environment for code execution
make build

#executes code
make exec
```

Listing 1. Makefile usage example

ra148051@students.ic.unicamp.br

B. Input

The program does not have an input argument by default, the input images are listed in code, and they are expected to be stored in the **input** folder. Listing 2 shows how images are listed to be executed in code. The $images$ tuple is implemented in $src/main.py$.

The images are expected to be in the *.png* format. The images in this work are expected to have objects in white background.

```
# for inserting other images, add tem to /input
  folder and list them here
images = (
    'image-0',
    'image-1',
    'image-2'
)
```

Listing 2. Input images inside code

C. Output

The output of the program is a series of images describing the prior steps to measures acquirement, the measures themselves in the standard output, and a histogram containing measures' information, for each image input. The output images are saved in the **/output** folder. *<image-name>-grayscale.png* stores the grayscale version of an input image, *<image-name>-contours.png* stores only the contours of the objects in the input image, *<image-name>-labeled.png* stores the labeled objects, and *<image-name>-histogram.png* stores the histogram of areas of an input image.

D. Implementation

The functions which implement the prior and the measures operations are defined in the *src/measures.py* file. The file has two auxiliary functions to calculate the measures: *transform_colors* and *get_contours*, which correspond to questions 1.1 and 1.2 in the project description. The *get_measures* function calculate the measures themselves, and relates to item 1.3 in the project description. The *areas_histogram* function builds the required histogram from item 1.4 from the description. The following items describe each of the implemented functions.

1) Transform Colors: The first prior task in order to obtain the measures was to transform the objects within the input image, which are colored, into their respective grayscale versions. In the output the objects are supposed to be painted in black, while the background remains the same. Listing 3 shows the implementation. Vectorized operations were chosen to make the execution time faster.

```
def transform_colors(img):
  b = g = r = img.copy()
  b[b[:, :, 0] < 255] = 0
  g[g[:, :, 1] < 255] = 0
  r[r[:, :, 2] < 255] = 0
  return b*g*r
```

Listing 3. Transform Colors Implementation

2) Contours of Objects: The second task in order to obtain the measures was obtaining the contours of the objects within the image. The contours of the objects can be used to obtain measures such as area, perimeter and centroid of each object through the application of the chain rule to obtain the contours[2],[3]. Listing 4 shows the implementation of the method. The output is the image with only the contours of the objects drawn in red color, and the contour array itself.

```
def get_contours(img):
  contours_img = np.full_like(img, 255)

  _, thresh = cv2.threshold(img[:, :, 0], 127, 255,
    0)
  contours, _ = cv2.findContours(thresh, cv2.
    RETR_TREE, cv2.CHAIN_APPROX_NONE)
  cv2.drawContours(contours_img, contours[1:], -1,
    (0, 0, 255), 1)
  return contours_img, contours
```

Listing 4. Get Contours Implementation

3) Properties of Objects Extraction: The measures calculation for the objects can be done entirely based on the contours obtained via Chain Rule. OpenCV has already built-in functions for area and perimeter calculation based on the contour (*cv2.contourArea* and *cv2.arcLength*, respectively). OpenCV also has a function for obtaining the moments of the objects (*cv2.moments*). With the moments, the centroid coordinates can be obtained by applying the following relations:

$$c_x = \frac{M_{10}}{M_{00}} \tag{1}$$

and

$$c_y = \frac{M_{01}}{M_{00}} \tag{2}$$

, where (c_x, c_y) are the centroid coordinates in x and y, respectively, M_{00} is the order zero moment, and M_{10} and M_{01} are the first order moments.

After the measures calculation, the objects within the image are labeled, that label is placed in the image (the brightness of the image is changed to make the labels more readable), and the calculated properties are printed in the standard output. The return of the function is the labeled image and the calculated areas, which are used in the following step. Listing 5 shows the main lines of the implementation.

```
def get_measures(img, contours):
  ...
  # modify image colors to make the labels more
    readable
  hsv_img = cv2.cvtColor(img.copy(), cv2.
    COLOR_BGR2HSV)
  hsv_img[..., 2] = np.multiply(hsv_img[..., 2],
    1.1).astype(np.uint8)
  output_img = cv2.cvtColor(hsv_img, cv2.
    COLOR_HSV2BGR) + 100

  idx = 0
  for contour in contours:
      # calculate required measures
      perimeter = cv2.arcLength(contour, True)
      area = cv2.contourArea(contour)
      moments = cv2.moments(contour)
      c_x = int(moments['m10']/moments['m00'])
      c_y = int(moments['m01']/moments['m00'])
      areas.append(area)

      # label image
      offset_x = 10 if idx > 9 else 5
      cv2.putText(output_img, str(idx), (c_x -
    offset_x, c_y + 5), font, 0.5, (0, 0, 0), 2)

      print("Regiao %2d: area: %6.1f perimetro: %9.5
    f " % (idx, area, perimeter))
      idx += 1
  return output_img, areas
```

Listing 5. Properties Extraction Implementation

4) Histogram of Areas: The last step was to generate a histogram of Areas of the objects, classifying the objects according to this specific constraint: Small regions are the ones with area less than 1500, Medium regions are the ones with area between 1500 and 3000 and Big regions are the ones with area greater than 3000. The number of objects with falls in each category is also displayed in the standard output. Matplotlib has a build-in function for histogram calculation (*matplotlib.pyplot.hist*) that was used for both the plot and the standard output. Listing 6 shows the implementation.

```
def areas_histogram(areas, img_name):
  counts, _, _ = plt.hist(areas, [0, 1500, 3000,
    4500], color='#0504aa',
                                alpha=0.7)
  plt.xlabel("Area")
  plt.ylabel("Numero de objetos")
  plt.title("Numero de objetos por area")
  plt.grid(axis='y', alpha=0.75)

  print("Classificacao dos objetos baseado em suas
    respectivas areas:")
  print("Numero de regioes pequenas: %d" % counts
    [0])
  print("Numero de regioes medias: %d" % counts[1])
  print("Numero de regioes grandes: %d" % counts[2])

  plt.savefig('output/' + img_name + "-histogram" +
    '.png')
  plt.show()
```

Listing 6. Histogram of Areas Implementation

III. Experiments

The *src/main.py* file executes the test pipeline. The idea is the following: for each input image, execute the four implemented functions mentioned in the previous section, and save their respective outputs. The input images are stored in **input** folder as mentioned before, and their names and dimensions are listed in table I. The input images are shown in figure 1. The generated output images are saved in *png* extension.

Image Names	Dimensions (width x height)
image-0.png	563 x 327
image-1.png	563 x 343
image-2.png	238 x 238

TABLE I
Input images used in experiments

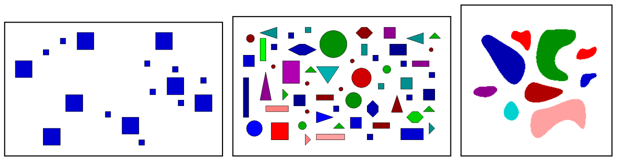

Fig. 1. Input images used in experiments. a) image-0 b) image-1 c) image-2

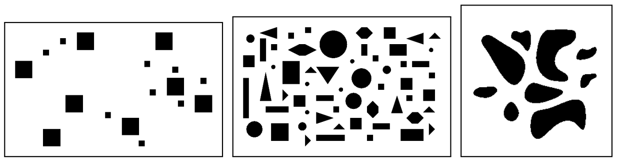

Fig. 2. Transform Colors Results. a) image-0 b) image-1 c) image-2

As we have 3 input images, 4 implemented functions, we have 12 images of output, and 6 output text blocks. The output images are stored in the **output** folder.

IV. Discussion

This section is organized in five parts. The first four analyzes the results of each function for all the input images, and the fifth sums up the implementation as a whole.

A. *Transform colors*

The results of the method for all input images are shown in figure 2. As it can be seen, the results are satisfatory, and they can be used for the contour calculation of each object.

B. *Contours of Objects*

The results of the method for all grayscale images obtained in the previous subsection are shown in figure 3. The painted contours seem to fit the objects in their respective original images, which validates the implementation. The obtained contours can be used in the following function.

C. *Properties of Objects Extraction*

The figure results of the method for all images and their contours are show in figure 4. As mentioned before, some brightness effect was applied in order to make the label readable. The label is placed in a way to center it related to the centroid position. The text outputs are shown in listing.

V. Conclusion

The properties extraction from objects within images was possible thanks to the approaches presented in the academia. The usage of different input images made it possible to validate the implementation for different object shapes. Future work could be done to generalize the methodology for different background and object colors/textures.

References

[1] H. Pedrini, “Introduction to digital image processing, project 3 description,” 2019.
[2] H. Pedrini, “Introduction to digital image processing, representation class,” pp. 5–14, 2019.
[3] H. Pedrini, “Introduction to digital image processing, representation class,” pp. 40–41, 49–50, 60–67, 2019.

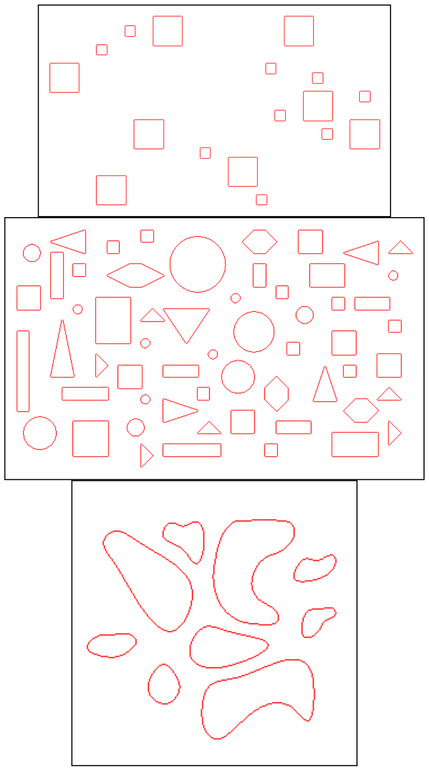

Fig. 3. Contours of Objects Results. a) image-0 b) image-1 c) image-2

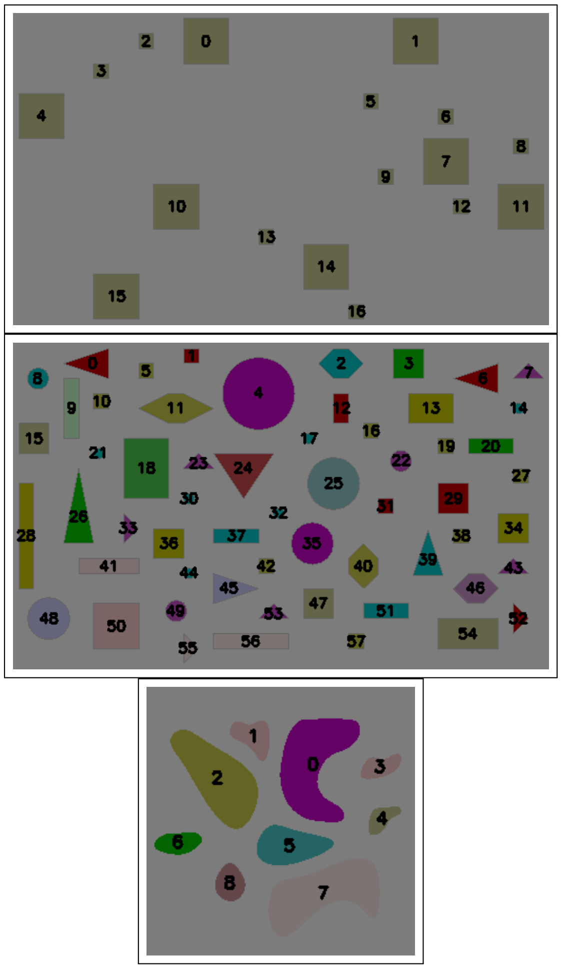

Fig. 4. Properties of Objects Figure Results. a) image-0 b) image-1 c) image-2