Computer Vision
Assignment 1

Sami Sellami

February 28, 2019

Problem1:

The task is to implement the adaptive thresholding from scratch, the function sould take the image as input and output a threshold binary image, the solution should also contain a user interface to adjust the parameters of the function. According to the book a practical introduction to computer vision using open CV the adaptive thresholding algorithm is defined as:

1. Divide the image into sub-images
2. For each of the sub-images compute a threshold.
3. For each point in the image determine a threshold by interpolating a threshold value from the four nearest thresholds using bilinear interpolation

We have implemented this algorithm by scratch, however, for the interpolation part, we used a simplified version whick consist of averaging the four neighbooring thresholds computed in step 2 for each pixel and then using this value to do the thresholding.

However, when reading the documentation about the implementation of the algorithm in openCV, it turned out that the algorithm is using a threshold for each pixel which is the average (mean or gaussian) of the neighbooring pixels using a kernel with a specified size.

This is why we implemented the two versions and compared the results to openCV results:
The result of the first method (with block size = 21) is shown in figure

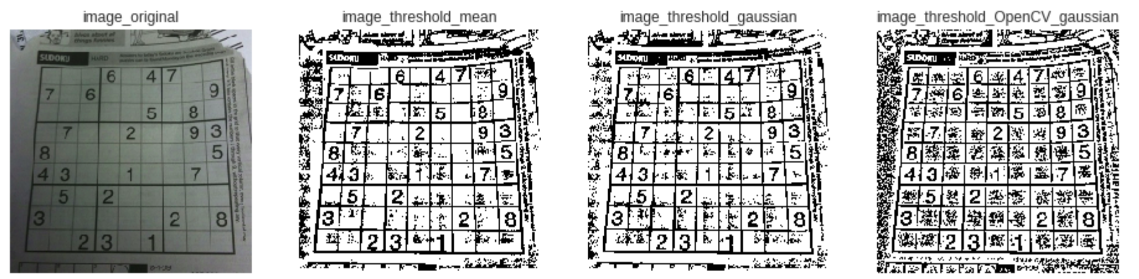

Figure 1: Adaptive thresholding using subimages (mean kernel(left), gaussian kernel(middle) and openCV(right))

we can see that the adaptive thresholding (with mean and gaussian) give satisfiable results, however we can see some differences between them and the openCV result

show the results of the second method (threshold computed for each pixel) and the openCV results (with block size = 11)

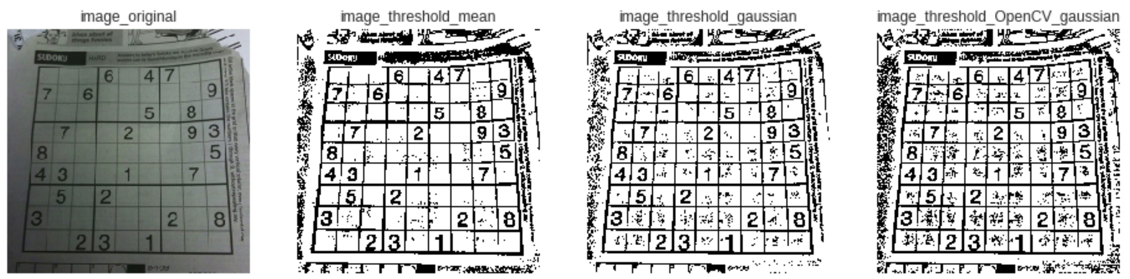

Figure 2: Adaptive thresholding using thresholding in each pixel (mean kernel(left), gaussian kernel(middle) and openCV(right))

We can see that this second method perform better than the first one and is closer to the image obtained with openCV library

Problem2:

The task is to calculate an objects size by photo and order objects according to their size, if two objects have the same size, one should sort them from left to right, we should also show size values (area, perimeters, heights, widths) in selected units (mm, cm, m, pixels)

For this we followed the steps shown in the following algorithm:

Procedure: Objects size calculation and ordering

```
Input: colored image
Output: image with objects sizes and order
 1.  blur the image
 2.  threshold the image and apply morphological operation
 3.  if the paper is detected as forground
 4.      apply contours and approximation to obtain the corners of the paper
 5.      apply perspective transformation to extract the paper from the image
 6.      threshold the image and apply morphological operation
 7.      if the objects are detected as forground
 8.          apply contour to obtain informations about contour, height and perimeter
 9.          apply connected components labeling to obtain the area of each object
10.          sort the parameters by the key rank_by and convert them to the desired unit
11.          draw the informations in the final image
12.      else adjust the thresholding parameters and redo steps 7-10
13.  else adjust the thresholding parameters and redo steps 3-10
14.  return parameters and final image
```

Now let's dive deeper into the algorithm implementation:

- **step1-2:** These steps consist of applying bluring to get rid of noise and the high frequencies present in the table, then applying thresholding and morphological operaiotions (closing or opening) to extact the paper as forground, for this an interface was created to adjust the necessary parameters (threshold value, type of morphological operation and the block size used, use_otsu ...etc) until the best binary image is found
  When the code runs it propose to the user to continue or to abort the compilation and adjust the parameters once again before relaunching the code, for some images (images 1, 11, 13, 18 for example), the best parameters have already been set, and one should only use the check box *use_defaul_parameters*

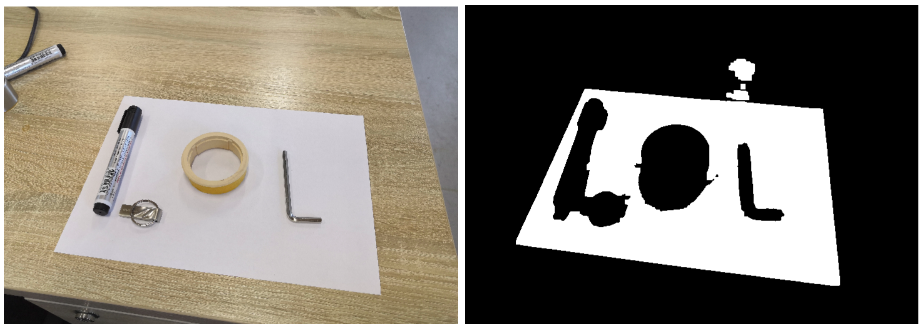

Figure 3: Image before and after applying the threshold and the morphological operations

  In figure one can see clearly the contour of the paper which is gonna be useful to apply perspective transformations

- **step 3-5: Paper extraction** These steps aims to obtain the image of the paper containing the objects to be sorted, for this we find contours and then approximate and filter them to obtain a contour with four points corresponding to the corners of the paper, after that, using perspective transformation we were able to obtain an image with the size of the paper and containg the objects, it should be noted that the order and position of the corners cannot be known in advance, this is why we implemented a small code for matching the points from source image to the destination; the smallest values of x and y correspond to the upper left corner ....etc **NB:** in order to remove the contours that are inside other contours (in the duck tape for example), we used the ***heirarchy*** variable returned with ***cv2.findContours*** to keep only the contours that have no parents

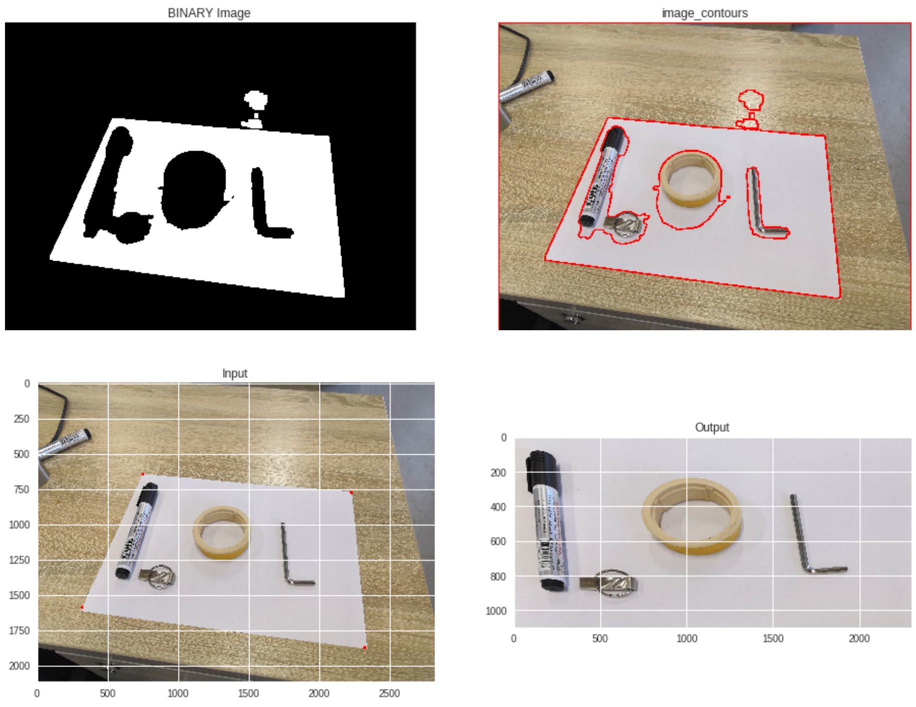

Figure 4: Contours application and perspective transformations

- **step 6-11: Objects detection and size calculations** In these steps, we first apply thresholding and (opening or closing) to isolate the objects of the image as forground, it is important to have homogeneous objects to be able to extract the best area estimation, then we apply contours detections to obtain the information about the perimeters, after that we use the function ***cv2.minAreaRectangle*** to obtanin a bounding rotated rectangle that will give us the height and width.
Regarding the area we used connected components labeling because the contour may not give correct informations(in the case of an object inside another one for example), to have the correspondance between the areas (from CCL) and the sizes obtained with contours, we order the contours informations from left to right using *centroids*, we do the same for the areas using of the *center* information given by CCl, and then we join the two informations.
The next step is to sort the parameters by the key **rank_by**, if two objects are the same (have the same area with some threshold for example) we order them from left to right. Then we convert them to the desired **unit**, for this we simply apply the formula

$$dist_{units} = pixel \times (\frac{width_{paper}}{shape[0]} + \frac{height_{paper}}{shape[1]})/2$$

where shape is the resolution of the image containing the paper (we did the average to take into accout that the pixel is not square)
Finally, we draw the informations on the final image with object 1 corresponding to the smallest (wrt rank_by) and object n to the biggest

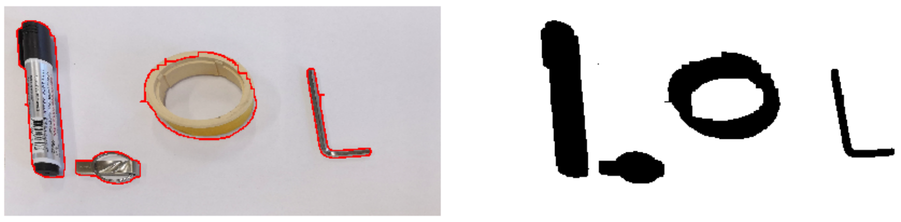

Figure 5: Thresholding and objects contours detection

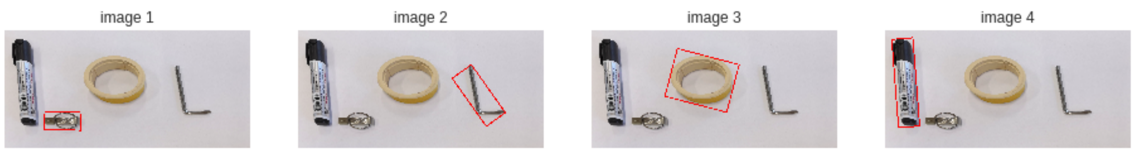

Figure 6: Bounding rectangle application

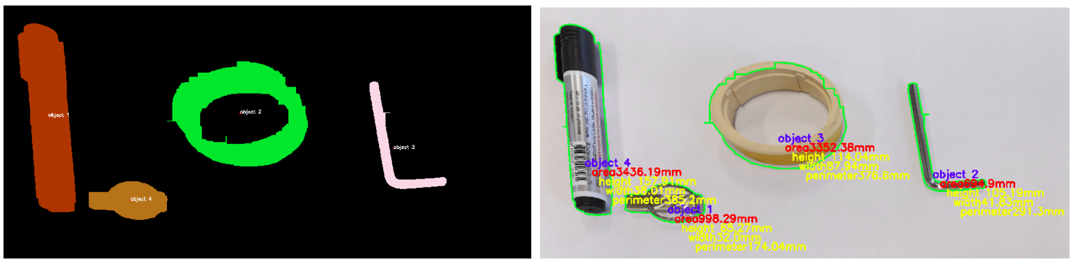

Figure 7: Connected component labelling and final image with size informations