Image Regions
CSC 449

Objectives

The requirements of this assignment are:

- Determine a threshold to segment a given image
- Label the connected components in the image
- Perform region selection and removal of certain areas based on given criteria

Files Included

1. script.m - invokes all the Matlab procedures required for this assignment
2. segment.m - reads image, builds histogram of pixel values and output binary map
3. label_cc.m - label the connected components using 8-neighbors algorithm
4. process_labels.m - detects labeled regions that do not meet the criteria
5. clean_labels.m - removes labeled regions that are marked as disqualified
6. draw_labels.m - draws the labels with bounding boxes and tags

Implementation Detail and Observation

We read the image file and used Matlab's function for collecting counts of pixel values in histogram array. We drew those counts in a histogram using different bin size as shown below.

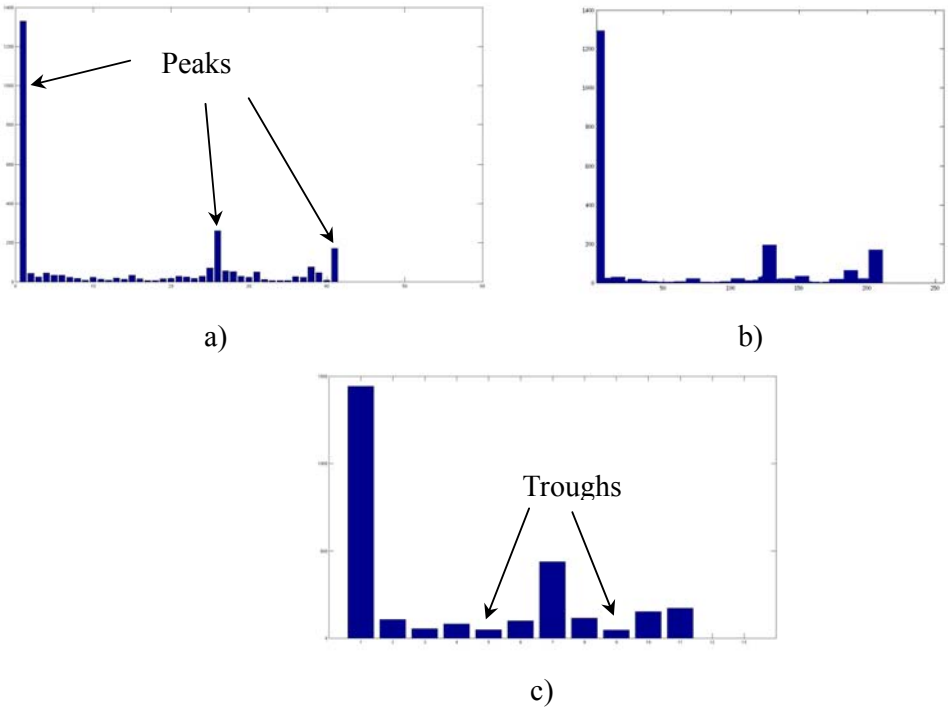


Figure 1. Histogram obtained using bin size of a) 5 b) 10 and c) 20. The pixel values of 0-5, 125-135 and 200-205 stood out as the most frequently observed values. On the other hand, the pixel values of 70-80 and 150-160 occur least frequently.

The original image's pixel values range from 0-205 and we noticed three peaks and two troughs as shown in figure1. Therefore, we used the values at troughs as starting point for threshold values, and observed the resulting binary maps to determine whether they are suitable for proceeding

further with labeling. The binary maps created with different threshold values are shown below.

a) b) c)

Figure 2. Binary maps obtained by using threshold values of a) 50 b) 70 and c) 150.

We observed that using threshold value ’50’ results in some small, white dots being connected to the bigger regions. On the other extreme, the threshold value of ‘150’ disintegrates the heart and cone/semicircle shapes into small pieces. The best result is obtained by using ‘70’ as threshold value as it divides up all the regions from each other perfectly.

Once we obtained the threshold value, we created a binary map representing ‘1’ for regions whose pixel values are above the threshold and ‘0’ for those whose values are equal to or below the threshold. We used this binary map as input to our labeling algorithm. The labeling algorithm utilizes 8 neighboring pixels to label the pixel in focus. It can be informally described as:

First pass:

For each pixel in column ‘c’ and row ‘r’ of image ‘I’,

- If all eight neighbors are background pixels (that is, pixels with value of zero), I(c,r) is assigned a new label.

- If there is one neighboring pixel with a non-zero label, assign this label to the current pixel, I(c,r).

- If more than one neighboring pixels is non-zero, pick the smallest of them and assign the label belonging to that label to the current pixel, I(c,r). Then enter all the labels detected in this stage into table, which we refer to as *equivalence table*, created to store these connected labels.

Second pass:

For each pixel in column ‘c’ and row ‘r’, of image ‘I’,

- If the pixel is labeled, look up the label in the equivalence table and walk along all entries in table to obtain the smallest label termed as equivalent to the current label. Then, assign that smallest label as the label of the current pixel.

In practice, we only scanned four neighbors of the pixels as shown in figure 3. The reason is to avoid redundant computation since we will be visiting the remaining neighboring pixels in the next computation cycles.

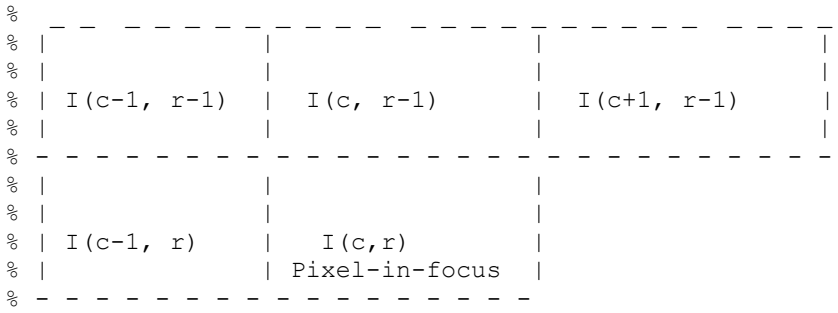


Figure 3. Initially, we visited the right and top neighboring pixels to avoid redundant computation.

Also, when we traverse the image pixels, we chose to do so in columnar fashion (that is, visiting pixels along the column-wise order from left to right). The decision is based on the knowledge that Matlab's interpreter orders the arrays—in our case, the image array—in the memory in the order of columns, thereby achieving faster access rates through cache-prefetching for such traversal.

However, we did not realize that this change in order of traversal, combined with our approach of scanning only four previously visited neighbors, would affect our labeling results in a subtle way. For example, if we are to label the given image using above traversal and neighbor scanning method, we will assign two different labels to the semi-circle in the top right corner of the image although it is clearly a connected region to any observer's eye.

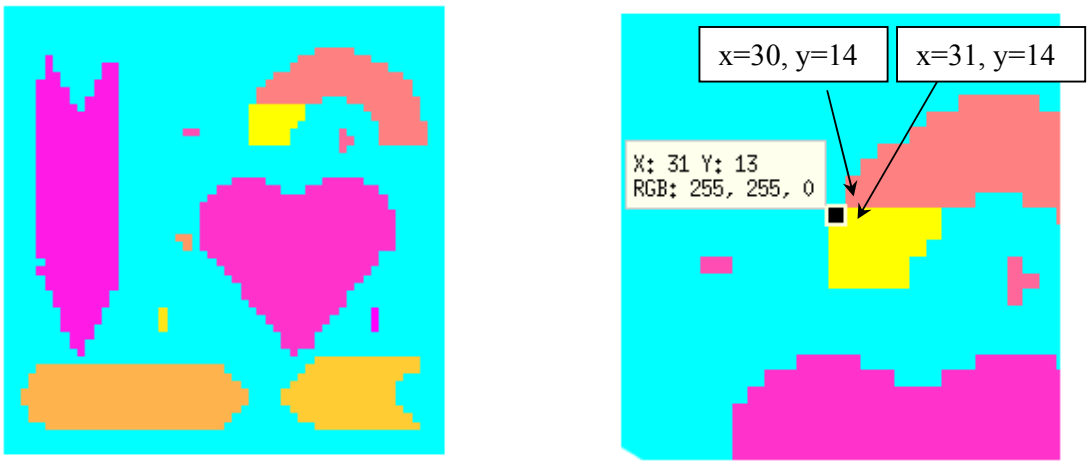


Figure 4. a) Our initial approach to labeling incorrectly assigned two different labels to the semi-circle. b) Close-up image showing the value and coordinates of the pixel where our original algorithm failed to properly assign the same label to the region.

In figure 4 b), the pixel located at x=31 and y=13 (refer to as point 'A') was scanned before the one at x=30 and y=14 (refer to as point 'B'). As a result, point 'A' was assigned a new label (suppose it is label number '5'). After the algorithm scanned column 13 and reached point 'B', none of its neighbors—according to our scanning approach—was labeled, so it was assigned a new label (suppose it is label number '6'). When it arrived at x=31 and y =14 (refer to as point 'C'), the algorithm detected two neighbors with labels '5' and '6'; therefore, it was assigned '5' to point 'C'. As a result, the bottom-left portion of the semi-circle bears a different label than the rest of its region.

In order to fix this, we adapted our original way of neighbor scanning to our approach of visiting pixels in column-wise fashion. That is, we consulted the neighbors in previously visited column and row to assign the new label as shown in figure 5 below. This solved the problem of incorrectly assigning two different labels to the same region when faced with edges in image region (see figure 6 a).

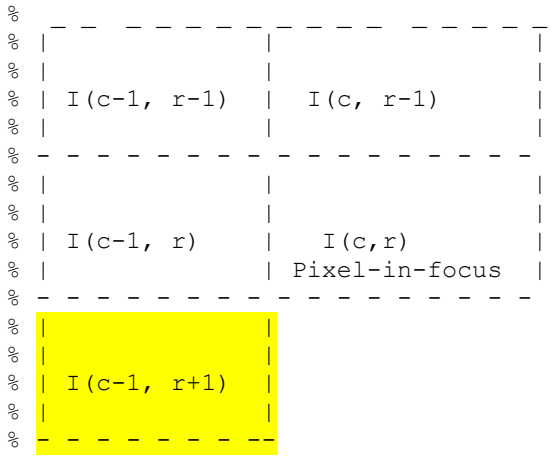


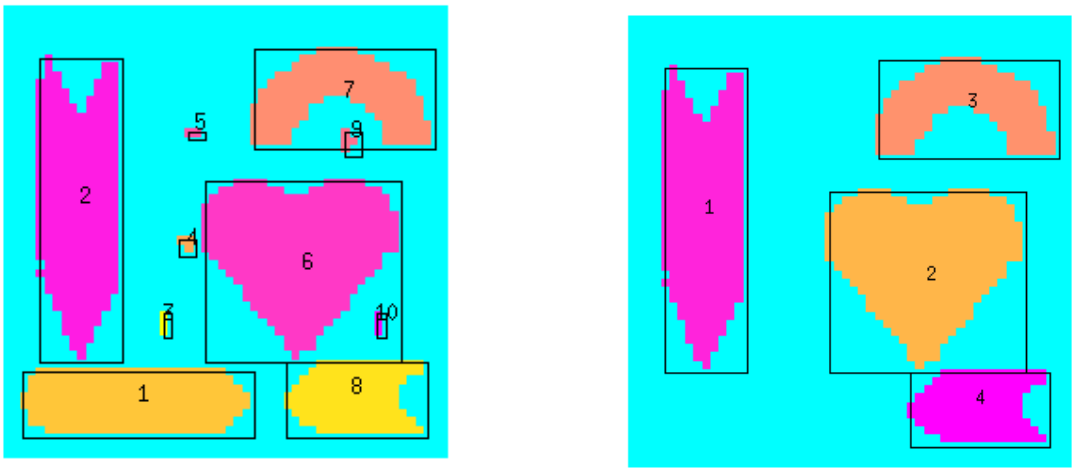
Figure 5. We modified our scanning approach by visiting four neighbors in column-oriented fashion.

Once we labeled the regions in the image, we purged certain regions that do not meet the following criteria:

1. Regions having area of above 10
2. Regions having aspect ration (width/height) of below 3:1

We calculated the area to count the number of pixels for each labeled region. We derived aspect ratios by obtaining the highest and lowest 'x' and 'y' coordinates of each labeled region and extrapolating it as a bounding box to calculate the ratio between width and height of each.

Finally, we put bounding boxes and labels to the respective regions as shown in figure 6 b.



a) b)

Figure 6 a) Ten different regions are detected after modifying our neighbor-scanning approach. b) Only four regions remained after eliminating the ones with smaller-than-threshold area and aspect ratios.

Conclusion

Although implementing 8-connected labeling algorithm sounds simple, we faced a couple of challenges aside from the subtle bug discussed in the previous section. The first of them is determining the optimum threshold for region separation. Our approach—of looking at the histogram and choosing the lowest values between the peak pixel values—is naïve and would not scale well if the original image contain a lot of shades and/or if we cannot detect clear peaks in the histogram. We found a couple of different approaches using simple mathematics [1,2] and tried one of them[1]; however, our Naïve approach in this assignment worked better because we were able to pick and choose the threshold value by trial and error whereas the other approach we tried gave us the value of '105', which yields less optimal results in region separation.

Another challenge we faced was finding the smallest, connected label in the equivalence table. We implemented equivalence table by using a nx2 array in Matlab. However, we found that traversing the array is not as easy as calling a hash function, so we implemented the table again by calling Java's Hash data structure within Matlab. But this approached complicated the code and we decided to use Google to find a simpler implementation. We finally found [3] which implemented equivalence table using a flatted Matlab array. The good part of [3]'s approach is that it dynamically updates the nodes in the table to smallest values possible; this results in a simpler code when resolving the labels in the second pass of our algorithm.

We also explored ways of matching shapes in images and found a few research work related to that by well-known vision researchers [4, 5, 6]. We read and understood the approach by Belongie and co. [7] the best and found the companion demo from one of Sergie's website. However, we did not have enough time to port everything into our code and therefore, decided to try it out as one of our future projects.

References

[1] Gray image thresholding using Triangle Method. URL:
http://www.mathworks.com/matlabcentral/fileexchange/28047-gray-image-thresholding-using-the-triangle-method

[2] URL: http://robotics.eecs.berkeley.edu/~sastry/ee20/thcode.html

[3] IMAGE_COMPONENTS – Connected Components of an Image or Matrix. URL:
http://people.sc.fsu.edu/~jburkardt/m_src/image_components/i4mat_components.m
draw_labels.m

[4] Matching with Shape Contexts by Serge Belongie. URL:
http://www.eecs.berkeley.edu/Research/Projects/CS/vision/shape/sc_digits.html

[5] Shape Matching Previously by Kristen Grauman. URL:
http://www.cs.utexas.edu/~grauman/courses/fall2009/slides/lecture21_shape.pdf

[6] Computer Vision Group - University of Berkeley. URL:
http://www.eecs.berkeley.edu/Research/Projects/CS/vision/shape/

[7] S. Belongie, J. Malik and J. Puzicha. Shape Context: A new descriptor for matching and object recognition. NIPS 2000.