CSC 249 Homework 2 Report

Chris Dalke

February 9, 2017

1 Introduction

For this homework assignment, we were given a grayscale image and were tasked with identifying and labelling the connected components in the image. Then we were given a set of qualifiers to apply the connected components, removing components that were too small or skinny. Lastly, for extra credit, we needed to highlight the heart shape in the image. I completed all of the parts of this assignment and my results and process are described below.

2 How to Run

2.1 Dependencies

Some of my code may require the Image Processing Toolkit to run. If it is not installed, MATLAB will show an error indicating so.

2.2 Running the program

To start the program, please run `Homework02.m`. The program will load the image that was provided for this project from the `Input` folder, and output images representing all stages of the process into the `Output` folder. The files outputted are as follows:

- `0_original`: The original image, scaled up with no filtering
- `1_threshold`: Result of image thresholding
- `2_pass1`: Result of pass 1 of the connected component algorithm
- `3_pass2`: Result of pass 2 of the connected component algorithm
- `4_pass2_labelled`: Connected components with labels and bounding boxes overlaid
- `5_removeSmallRegions`: Result of removing small regions
- `6_removeSkinnyRegions`: Result of removing skinny regions
- `7_detectHeart`: Result of detecting heart

Note: All of the output images have been configured to use a scaled color map where the minimum value is blue and the maximum value is yellow. This makes it easy to identify the unique values for the array. These images are also displayed in this document.

3 Building the Threshold Image

The algorithm I used to build the threshold image iterates through the original image, writing a 0 to a new image if the sampled pixel is below a given threshold and 255 if the sampled pixel is above a given threshold.

The threshold I chose is 60, which I obtained by examining a histogram of the input image.

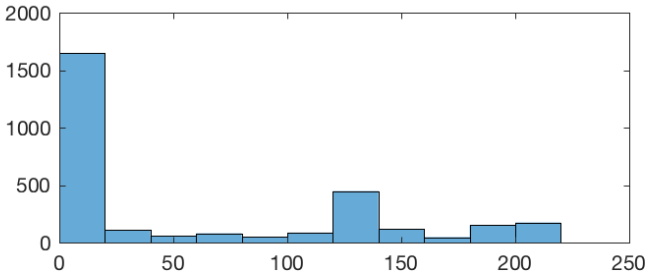

Figure 1: Histogram of Input Image

The histogram was roughly bimodal as shown above, so I chose a number between the two peaks.

The input image is shown on the left below, and the result of the thresholding is shown on the right in false coloring.

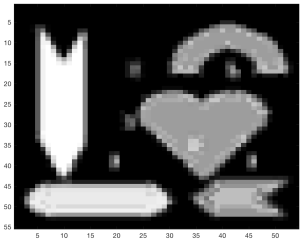

Figure 2: Original Image

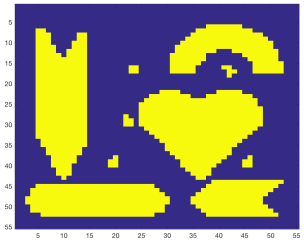

Figure 3: Threshold Image

4 Labeling Connected Components

To label the connected components from the threshold image, I used the Connected-Component Labeling algorithm from graph theory. The algorithm takes in a binary threshold image and outputs an image where each connected region has its own 'label', in this case saved in the form of a unique value on a grayscale image. This algorithm consists of two passes.

The first pass of the algorithm loops through an image row by row. The algorithm carries out a connectivity check on each neighboring pixel that has already been assigned a label. It chooses the minimum of those labels and applies that label to the central pixel. If the pixel has no neighbors with a label, it creates a new label ID and applies that to the current pixel, then continues.

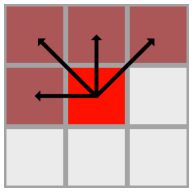

Figure 4: Diagram of Connectivity Check

For this algorithm, we are looking at 8-connectivity, which considers any of the compass directions or diagonals a connection. Since we are progressing row by row, we only have to check the four pixels that have already been checked, which is shown in the figure above.

As we sample from previous pixels, any time two regions are encountered during a check for a pixel both regions are added to an equivalent set. This will help with the merging of regions later.

At the end of this pass, the algorithm will end up with a set of labeled regions that almost represent the connected components, but have issues in several places as seen by the image below.

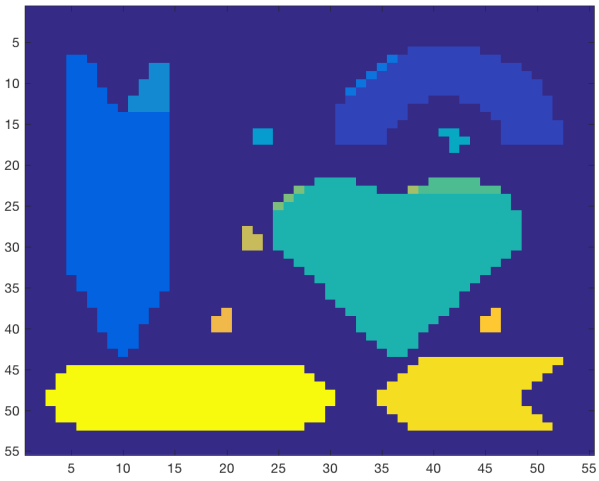

Figure 5: Result of Pass 1 of Connected Component Algorithm

The issues stem from the fact that the algorithm has no way to know if two parts of an image are actually connected until it scans further into the image, by which time it can't go backwards and fix this.

With the help of the sets that were recorded during the first pass, the algorithm calculates the disjoint sets for all of the regions. Using this, it is possible to create a new set of labels such that any labels that were touching in the first pass can now be merged together.

Pass 2 of the algorithm applies these new labels, which results in a cleanly labeled image.

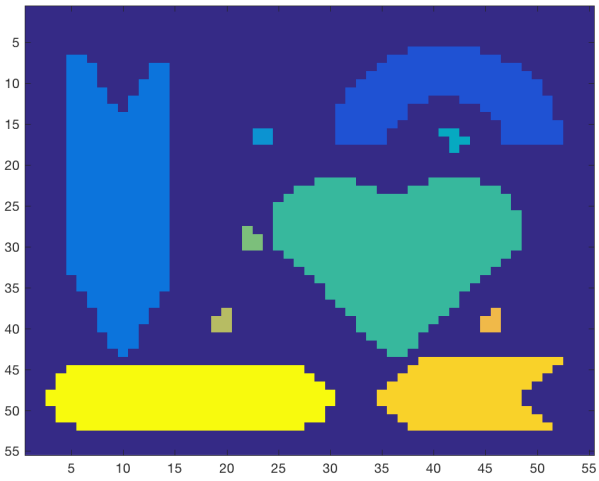

Figure 6: Result of Pass 1 of Connected Component Algorithm

5 Performing Region Selection

For this step of the assignment I created a series of functions that help work with the region image. I created functions that build a list of the current labels in an image, and a function that identifies how many unique images are in the label. Another function takes in an input and a label, and returns a bounding box for the region of pixels with that label.

The bounding box algorithm works by looping through the entire image and determining the minimum and maximum X and Y positions for a specified label, which can then be turned into a list containing the X position, Y position, width, and height of the region.

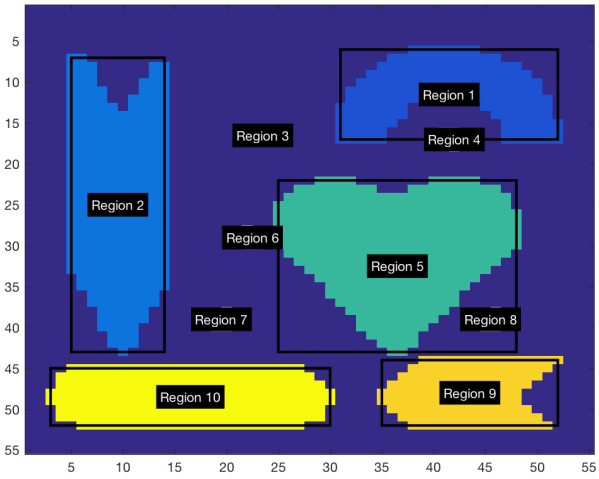

Figure 7: Connected Components Labeled by Regions

5.1 Removing Small Regions

To remove the small regions, I calculated the per-pixel area of each region. My code loops through the image and counts the number of pixels with a certain label. If a label has less than 10, the pixels in the region are overwritten with a label of 0, removing the region.

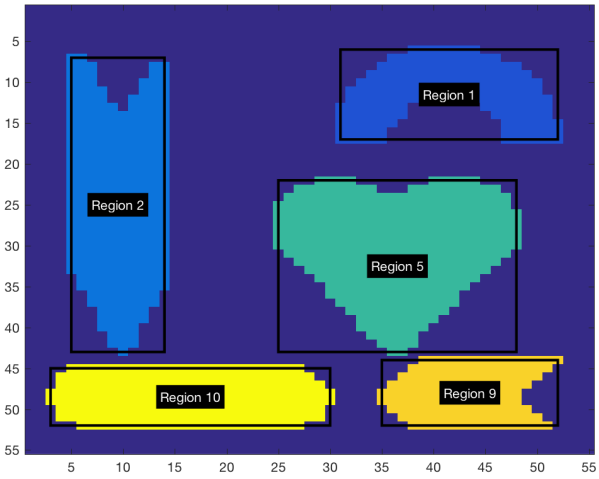

Figure 8: Result of Removing Small Regions

5.2 Removing Skinny Regions

To remove skinny regions, I loop through all of the labels in the image and calculate the bounding box for the label region. If the aspect ratio ($width/height$) is above 3:1, the pixels in the region are overwritten with a value of 0, removing the region.

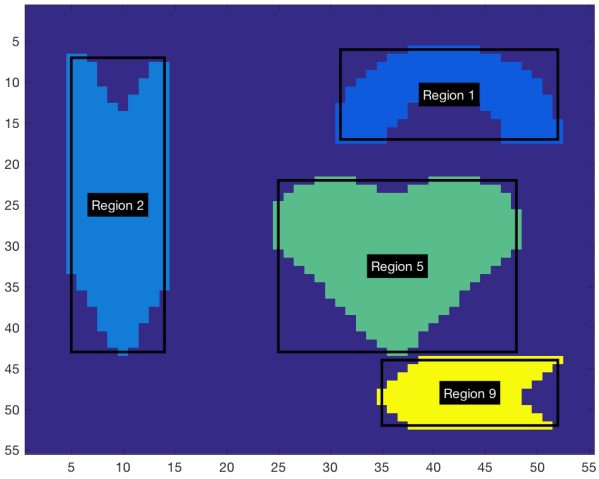

Figure 9: Result of Removing Skinny Regions

6 Extra Credit

For extra credit, I detected the heart shape and highlighted it. To detect the heart shape, I used two main qualifiers. Looping through all the regions, I first checked if the aspect ratio was between 0.8 and 1.2, which indicates that the shape is roughly square. This narrowed out the longer shapes.

Then, I used a sampling method. I created a 9x9 black and white image of a heart shape that I sampled pixels from the region image against.

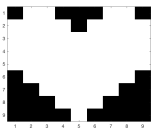

Figure 10: Heart Sample Image

The algorithm samples the pixels from the sample image in equivalent positions inside the bounding box of each region and counts the number of pixels that mismatches the sample image. If the number of samples that don't match the heart shape is low, the shape can be considered a heart. A visualization of the sampling is shown below, zoomed in on a region that passes the check as well as a region that does not pass the check. Samples that match are shown in

green, and samples that do not match are shown in red. The heart shape image has very few incorrect samples, which means it passed the test.

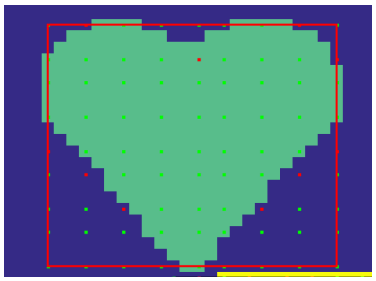

Figure 11: Heart Sampling (Pass, Errors)

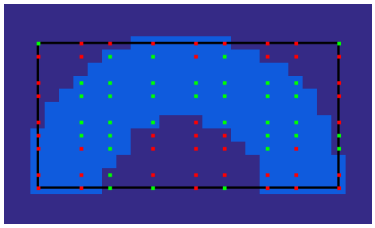

Figure 12: Heart Sampling (Fail)

The result of this algorithm is that the heart in the image is highlighted, as seen below.

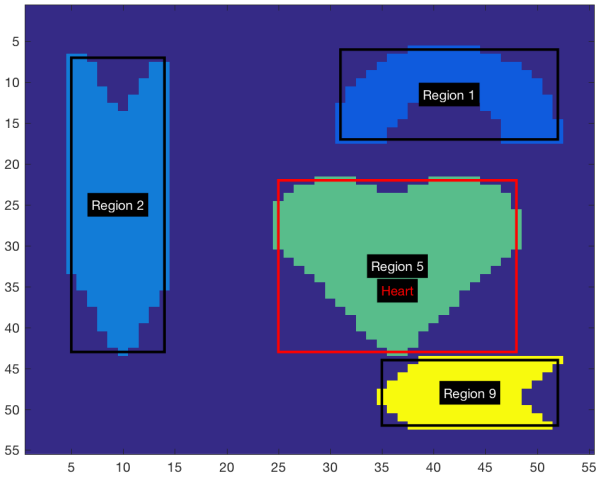

Figure 13: Result of Highlighting the Heart

7 Academic Honesty

I did not collaborate on this project and all work is my own.