CMPEN/EE455: Digital Image Processing I

Computer Project # 2:

Connected-Component Labeling and Set Operations

Ritvik Muralidharan, Georges Junior Naddaf, Zian Wang

Date: 09/22/2017

A. Objectives

This project's main purpose is to:

- Get acquainted with connected-component labeling and processing
- Get acquainted with logical set operations and their implementations (AND,OR,NOT,XOR, min)

B. Methods

For Task 1(a):

We simply iterate over the image and set a threshold of 190. Whenever we detect a pixel that is bright (grey value>=190), we make it completely bright (=255) and if the pixel is not bright, we darken it completely (grey value=0). This will result in an image with distinct bright pixels and will allows us to perform connected-component analysis of the image.

For Task 1(b):

We use the functions specified in the prompt. Bwlabel() labels the image and colors all connected components in unique colors and assigns each a unique integer. Then we use label2rgb() to output and display the image.

For Task 1(c):

For this task, we came up with a brute force solution that is inefficient (exponential runtime). The solution works nonetheless. We begin by converting the image from color to grayscale because

processing the colored image directly posed some difficulties in processing that seemed to emanate from Matlab. We Iterate over the unique values of the connected components. For each value, we go over the fnew image, store the connected component in ftest and nullify all other pixels in ftest. That way, we get a distinct ftest for each connected component. Then for each connected component image, we go over the image and count how many pixels are different than 0. We increment a counter that will eventually represent the size of each connected component. We compare the counter to our stored maximum. If the counter is bigger, we update our store value to represent the maximum and we save the unique identifier of the connected component in codenum. We then go over our labelled image fnew and make sure to represent the biggest component by using codenum while also nullifying all other pixels. This gives us the biggest connected component f1stbw. Keep in mind however that the result image has no colors. To handle that, we create a new rgb image fRGB1 and store f1stbw in it. This image however did not have the same RGB values as the original image - fRGB. We found the pixels of the largest component in fRGB using the pixel location and noted down the specific RGB values. Then, using for loops, we targeted those pixels in fRGB1 and changed their RGB values to the ones from fRGB, while keeping all the other pixels white. We repeat the above process for the remaining 3 largest components by adding a condition to ignore our previous codenums. This code produces the desired results.

For Tasks 2(a) (b):

A AND B represents the set intersection between A and B.
A OR B represents the set union between A and B.
NOT A represents the complement of set A
A XOR B represents the intersection between the union of A and B and the complement of the intersection of A and B.
For each logical operator that we had to implement, we covered the logic with if statements that are fairly straightforward. We added every two pixels of each image and according to the result we got and the operation to implement, we computed the pixel values in the output image.
For Task 2(c):
To implement the min function, we use a simple if/else statement. We traverse images C and D in parallel. If a pixel in image C is smaller that that in image D, we insert it in our output image E. If not, we store the pixel from image D into E.

C. Results

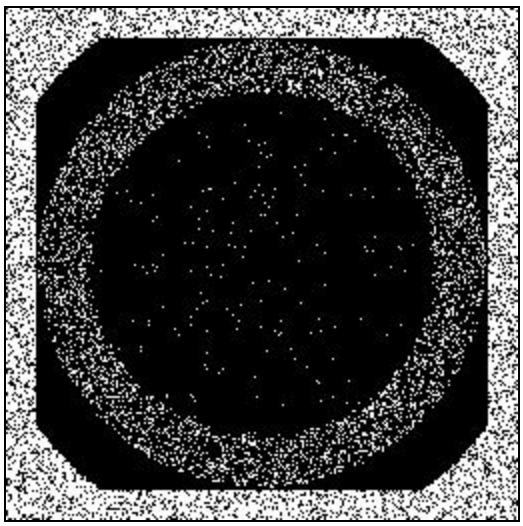

Figure 1. Task 1(a) - fthresh

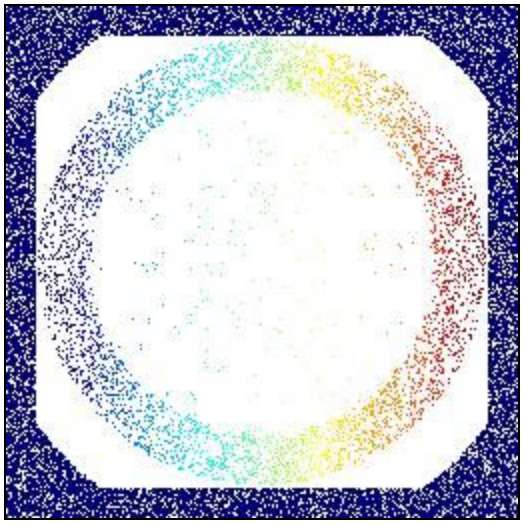

Figure 2. Task 1(b) - fRGB (colored image of connected components)

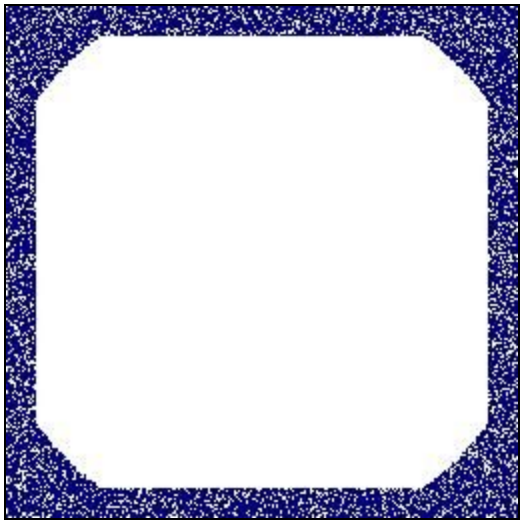

Figure 3. Task 1(c) - fRGB1 (Largest component)

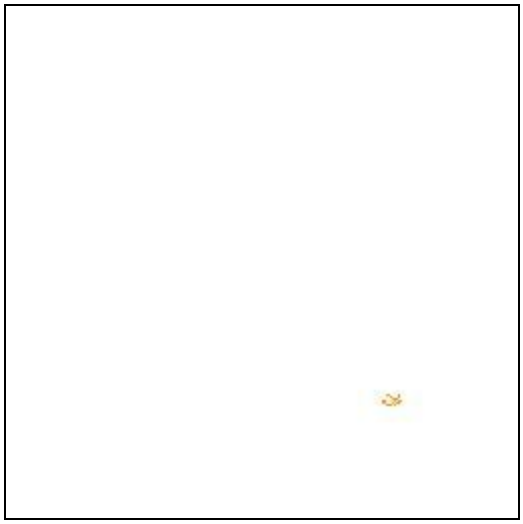

Figure 4. Task 1(c) - fRGB2 (2nd largest component)

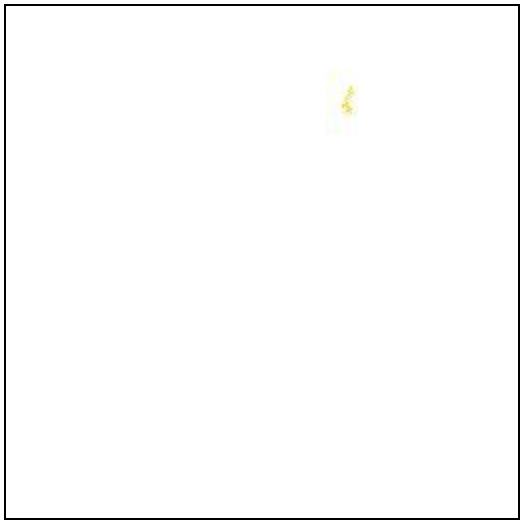

Figure 5. Task 1(c) - fRGB3 (3rd largest component)

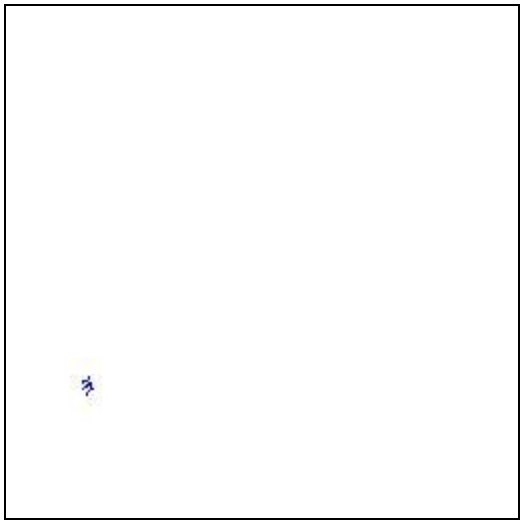

Figure 6. Task 1(c) - fRGB4 (4th largest component)

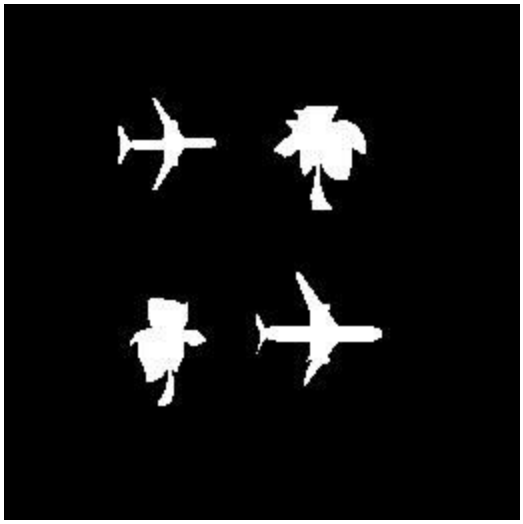

Figure 7. Task 2(b) - A AND B

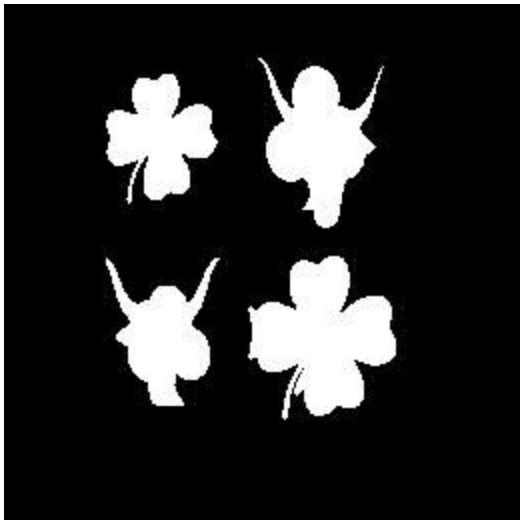

Figure 8. Task 2(b) - A OR B

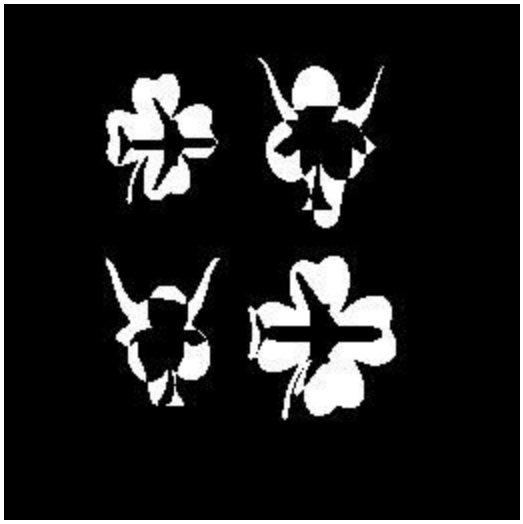

Figure 9. Task 2(b) - A XOR B

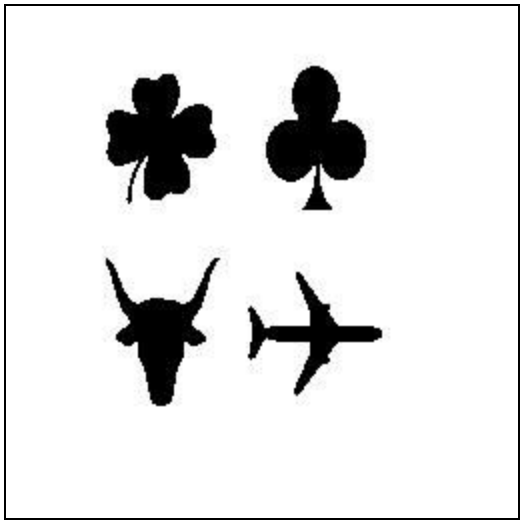

Figure 10. Task 2(b) - NOT (A)

Figure 11. Task 2(c) - Image E, where E = min(C,D)

D. Conclusions

This project allowed us to better understand connected components in images and how logical operators work in the context of image processing:

- Matlab allows us to detect connected components in images and process them. This is a very important application in the field of image processing. It can lead to many conclusions about images depending on what we are looking for. In this example, we performed brightest-region extraction.
- Matlab logical operators are fairly straightforward in the sense that they allow us to perform core operations in a simple way. Implementing them gave us a good idea of how useful they can be when mixing images.