Gerom Pagaduan
Dr. Jiang - CS390S
Final Project
12 December, 2019

Final Project Report – Port Detection

Overview

For this project, I worked with Daniel Reuter to create a program that detected and recognized the different electronic ports on a device. The intention was to be able to take a photo of the back of a device and our program will return the number of ports and the type of ports present.

I was mainly focused on the detection portion of the project (Daniel focused on recognition). My goal was to be able to segment each port and return the number of ports present in the picture given to my program.

Methods

My immediate thought when attempting to detect ports was that the colors of the port and the surface that it is on have similar colors. Therefore, I decided that increasing the contrast—and in turn increasing the difference between the ports and its surrounding areas—would be a good idea.

So, I used a *contrast stretch* and compared the two histograms. And upon further analysis of the histogram, although it did not alter as much as I had hoped, it increased the diversity of the pixels. This made it easier to obtain better results during the thresholding portion.

Figure 1: Original Image vs Contrast Stretch

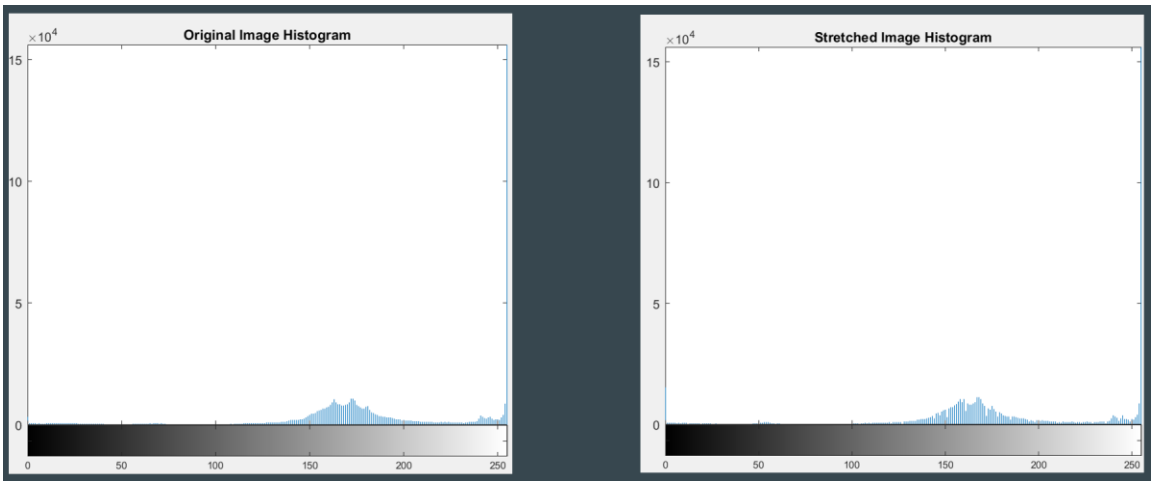

Figure 2: Histogram Comparison

At first, I only thresholded the one image, but it yielded a very messy result and lost a lot of information. In response, I decided to split the color planes into three different ones (Red, Blue, Green), and used a threshold on each color plane.

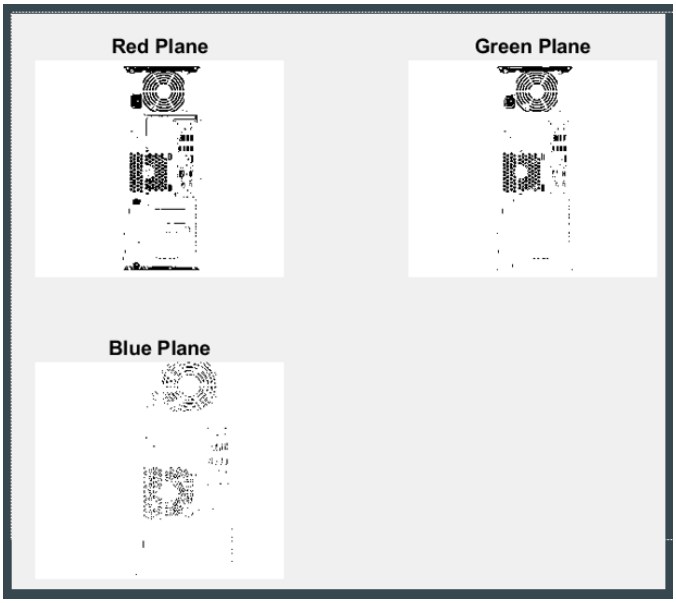

Figure 3: Three Color Planes Thresholded

After I finished thresholding the three planes, I summed up all the images to receive a more complete image. I then inversed the colors so that black was the background color so that I could see the result more clearly.

Once I obtained the sum of the planes, I started to reduce the noise. To do this, I used the `imfill` function, using `'holes'` as the parameter. My intention in using this was to fill in insignificant white space. It worked, but there were still many frayed edges and random particles. To fix the issue, I decided to apply the structuring element functions, `strel`, using `'disk'` as the shape—I used disk because I thought it was the most versatile shape out of all the possible ones we could use—and it rounded out the edges of the image and omitted all the floating particles, leaving behind only the important ones.

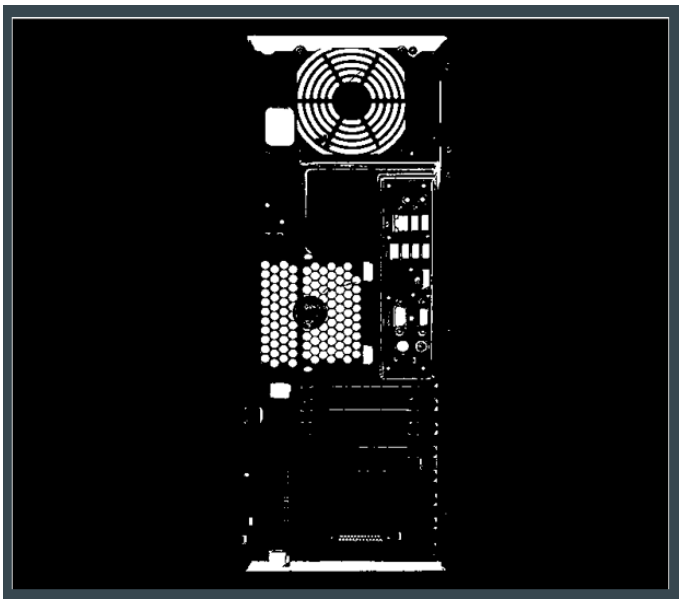

Figure 4: Image after using imfill

The end result was that there were a few ports that were lost and the final image did not yield enough information.

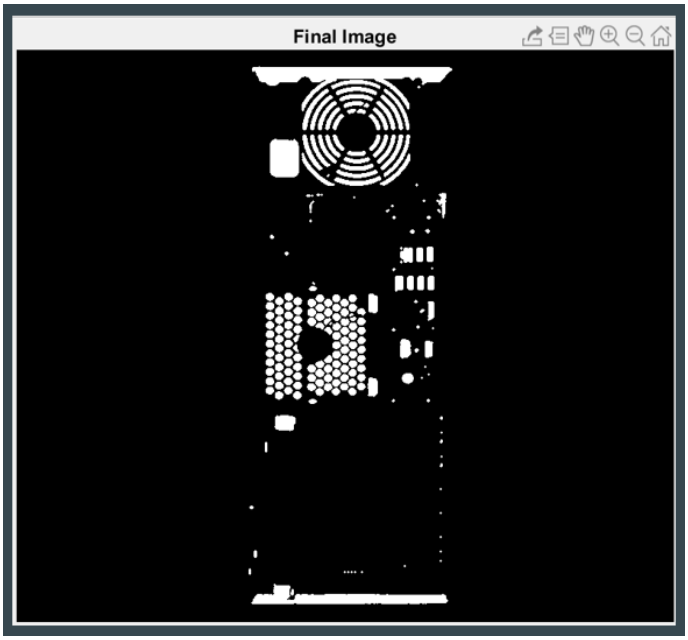

Figure 5: Image after using strel

I figured that because I was manipulating the RGB plane, there was a more efficient way to stretch the image's properties. So, I used the *decorrelation stretch*. This colorshifted the whole image and made it easier to divide.

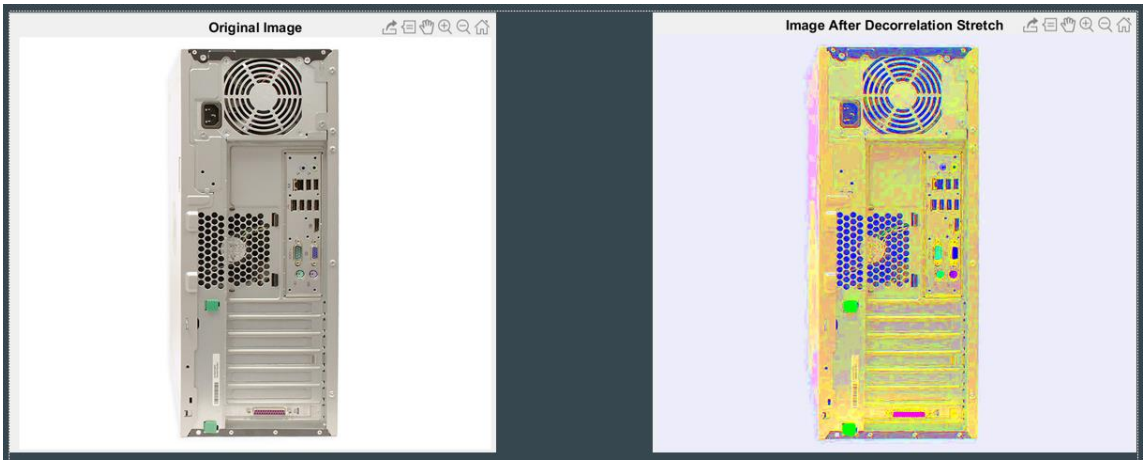

Figure 6: Decorrelation Stretched Image

I noticed that most of the ports were now blue in the image, so I set the threshold of those as 0.0 since I wanted to keep all those elements, while I adjusted the red and green until I obtained satisfactory results. I then performed the same segmentation and noise reduction techniques as I did for the contrast stretch, and this time, the thresholded image was able to keep all of the ports.

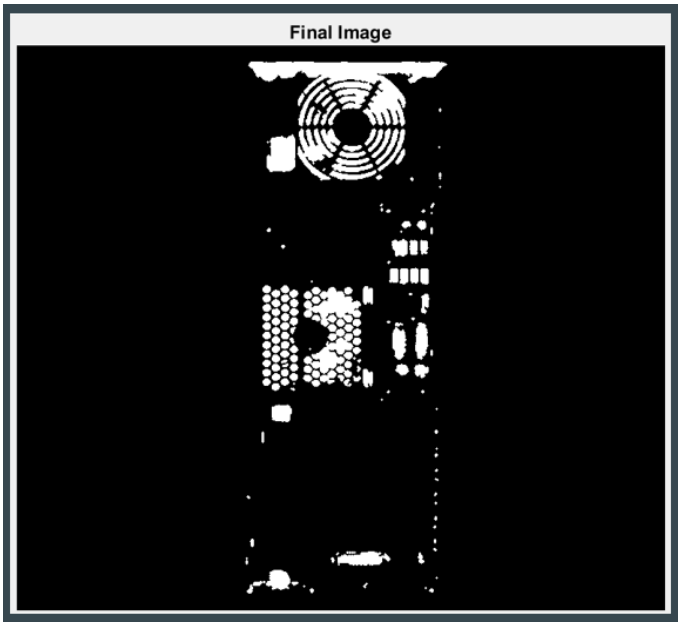

Figure 7: After using strel on decorrelated image

Challenges and Possible Solutions

My biggest challenge for my portion of the project was isolating the ports from the rest of the image—mainly, the other non-port holes. The image kept the normal holes, some which

overlapped with the ports, which made the thresholding image retain more information than it needed.

To remedy this, I was thinking of using more edge detection methods, such as sharpening the edges of the image or using Canny edge detection to find the ports easier.

Another method was suggested to me, which was to use some light filters/patterns over the image. I was told that this would be ideal because the reflectivity of the ports would be stronger than the normal holes. So then, once there is more light on the ports, I would be able to threshold them easier using a higher threshold value. I also think this would be the best course of action, but I am still unsure of how to implement it.

Next Steps

After I fix the issues with isolating the ports in the image, I will then use `regionprops` to find the properties of the image, box the ports, and return the number of ports. Then, it would prepare each port for image recognition via template matching.