CSCI 8810: Project #2

Due on Wednesday, March 4, 2015

Prof. Arabnia 08:00am

Sina Solaimanpour

Contents

I. Introduction

In this paper, I will discuss about the second phase of the ColonD Image processing project. The first phase of the project was about designing the overall structure of the project and implement some basic operations like read/write, gray-scale conversion, thresholding and smoothing images by applying filters. In this phase of the of the project, I will expand the program to include some other filtering operations and another slightly more complicated feature called Connected Components Labelling or CCL.
The following list shows what new features have been added to the program in phase 2:

- Connected Component Labelling (CCL)
- Invert pixel colors
- Robert's operator
- Sobel's operator
- Prewitt's operator
- Kirsch's operator
- Laplacian filter
- Gaussian filter

These features are all implemented and tested with different images, individually and in combination of other operations. In the next sections, I will go over each one of these features in more detail and will present some results from the program.

II. Connected Component Labelling

Connected Component Labelling is a well-known algorithm used to find individual components in an image. The idea behind this version of the algorithm is to check the surrounding pixels (in this case, the west and the north pixels) of a white pixel and assign appropriate label to the pixel in hand. Details of this pixel are as follows:

1. If the north and west pixels are black: Assign a new label to the pixel
2. If west pixel is white and the other one is black: Assign the label of the west pixel to the pixel in hand
3. If north pixel is white and the other one is black: Assign the label of the north pixel to the pixel in hand
4. If both west and north pixels are white: Assign to the pixel in hand, the smaller label of the two and record that the two labels are equivalent.

The algorithm described above is the first pass of the CCL algorithm. After labelling each pixel during the first pass, we need to go over all of the pixels and handle the equivalent labels found in the first pass. As the number of labels during the first pass might get very big, I am using a **Disjoint Set Data Structure** implemented in **Boost framework**. The overall idea of the disjoint set dadta structure is to keep the equivalent elements as a set represented by the smallest member of that set. When a new label is created, it is considered to be a separate set in the bigger set. Later on in the program, when an equivalency is found between two labels, the sets representing those two labels are merged together with a union function. This will allow us to know exactly which labels are equivalent with each other.
During the second pass, we find the set that each label is a member of and use the representing label in that set as the final label.
Just for better visualization of the components, I use a very simple coloring technique which might fail from time to time, depending on the number of the components found and their order, but it will try to assign distinct colors to different components found in the image.
Figure 1 shows the process of labelling the lena's image. First the image is converted to gray scale. Then, I apply some gaussian smoothing to the image and use the simple thresholding feature to convert the image

to a binary image with some disjoint components. At the end, I use the CCL in the program to label the components in the resulting image from the previous steps.

Figure 1: The resulting images from using the CCL algorithm on the lena's image after applying a gray-scale conversion, two gaussian filters and a simple thresholding. Different colors in the last image shows different components found. Here, we have **29** distinct components with average component area of **300.276** pixels.

In the next example, shown in Figure 2, I have tried to show that CCL can be used to find important parts of an image of a human face. The image used in this example was obtained randomly from a search for "Face Image" on Google and was in gray-scale originally. Two Guassian filters have been applied to the image and after that, the image has been converted to binary using a thresholding technique. CCL has been used on the image after inverting the image pixels. The result is very interesting because you can clearly see that the important parts of the face (her eyes, lib, nose and a not very complete boundary around the face) have been highlighted as different components.

III. Invert

Inverting an image has been implemented in the program as an extra feature. It will subtract the pixel value from 255 and use the resulting value as the pixel intensity. This function is illustrated in Figure 3.

IV. Robert's Operator

Robert's operator is acting as an edge detector. The operator is consisting of separate applications of two different 2 x 2 filters. The idea behind this operator is to approximate the gradient of an image. The matrices used as filters are shown below.

$$F1 = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}, F2 = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$$

Figure 2: Applying CCL to an image of a face. Different colors in the last image shows different components found. Here, we have **10** distinct components with average component area of **926** pixels.

Figure 3: Inverting pixel values of the lena's image.

These two filters are applied to the image individually and at the end, the final result would be the summation of both images together. I will present two examples of this operator used on different images.
The first application of the operator was on an image of a bike in front of a brick wall. The operator is applied without any other modifications to the image and the result is shown in Figure 4.

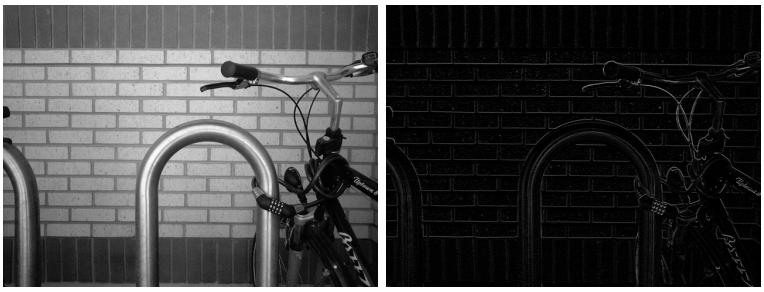

Figure 4: Applying Robert's operator to an image without any other modifications.

Now, I am going to show the resulting image from Robert's operator to the same image but with applying a Guassian filter before the Robert's application. The results are shown in Figure 5.
As you can see, the difference between the results of the Robert's operator to an image with or without the

Figure 5: Applying Robert's operator to an image with Guassian filter applied to it.

Guassian filter is not big. Although, if you pay attention closer, the difference is in the amount of noise shown in two results. The image with Guassian filter applied to it, has much smoother surface which allows us to focus on the important information not getting confused with the noise.

V. Sobel's Operator

Sobel's operator is another edge detector. The operator is consisting of separate applications of two different 3 x 3 filters. The idea behind this operator is to use two filters, one focusing on Horizontal edges and the other one focused on the Vertical edges to find all of the edges in the image. The filters used in this operator is as follows:

$$GY = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, GX = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

These two filters are applied to the image individually and at the end, the final result would be the summation of both images together. I will present two examples of this operator used on different images.
The first application of the operator is on the same image used in the Robert's operator section (The bike's image). The operator is applied without any other modifications to the image and the result is shown in Figure 6. As you can see, the edges are much brighter and stronger comparing to the edges found using Robert's operator.

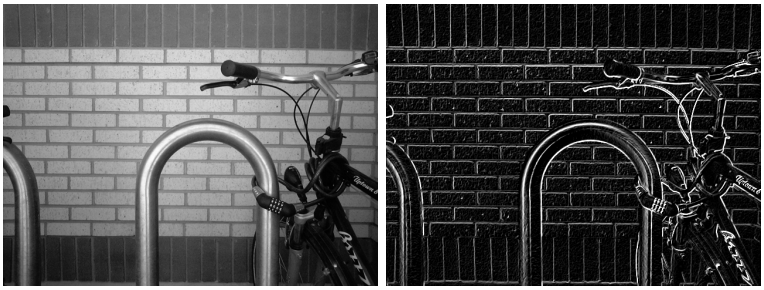

Figure 6: Applying Sobel's operator to an image without any other modifications.

The second example of the Sobel's operator will be on the face image used in the previous sections. This is to illustrate how an edge detection algorithm will perform on an image which contains a face in it. The results can be seen in Figure 7. The face image is first smoothed with an application of a Guassian filter. The result of this operator clearly shows that the important areas of the image are again highlighted and the main purpose of edge detection algorithms are to find and highlight important parts of the images which are the boundaries between different colors in the image.

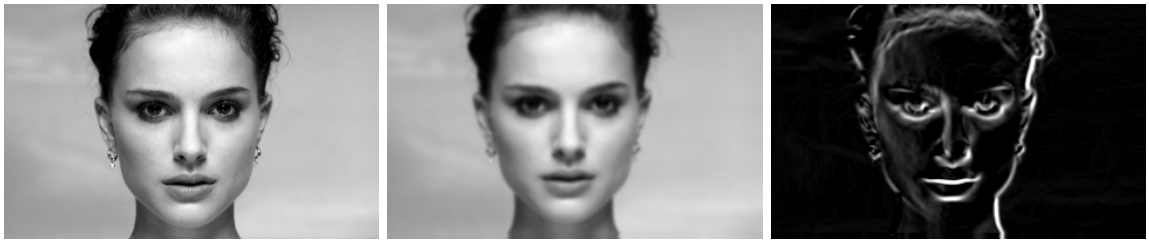

Figure 7: Applying Sobel's operator to a face image with Guassian filter applied to it.

VI. Prewitt's Operator

Prewitt's operator is again another edge detector operator like the previous two operators. The operator is consisting of separate applications of two different 3 x 3 filters. The idea behind this operator is very similar to the idea used in Sobel's operator which uses two filters, one focusing on Horizontal edges and the other one focused on the Vertical edges to find all of the edges in the image. The filters used in this operator is as follows:

$$GY = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, GX = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

These two filters are applied to the image individually and at the end, the final result would be the summation of both images together. Figure 8 depicts how this operator performs on the image of the bike used in previous sections. The result is very similar to the result of the Sobel's operator, but it can argued that the result of the Prewitt's operator is less biased towards the pixels in the middle of the filter.

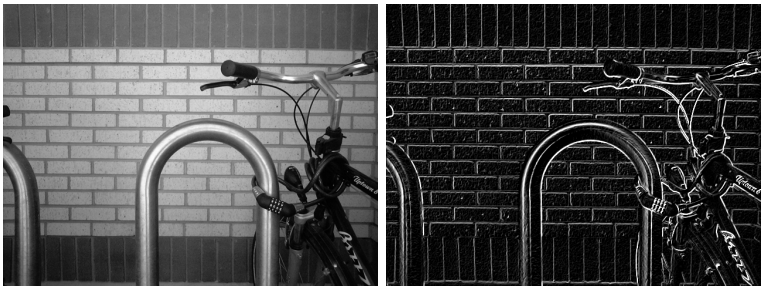

Figure 8: Applying Prewitt's operator to an image without any other modifications.

VII. Kirsch's Operator

Kirsch's operator is another edge detector operator but it acts a little bit differently than the others. The operator is consisting of separate applications of four different 3 x 3 filters. Two of the filters are biased towards the Horizontal and Vertical edges and the other two are simply a 45 degree rotation of these filters. The filters are as follow:

$$F1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, F2 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, F3 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}, F4 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$$

These filters are applied to the image individually and at the end, to combine the results, I find the maximum for each pixel among four pixels of the images and use that value to represent the final pixel value. Beside

combining the resulting images from the four filters, I give the user the option to save all separate and the combined images to the file system. Figure 9 depicts how this operator performs on the image of the bike used in previous sections by showing all four separate images and the combined result at the end. You can obviously see specially in the first two filtered images that the filter are working in favor of the Horizintal and Vertical edges, respectively. The other two filtered images are showing the results for the rotated filters and the last image shows the combined image using the max operator.

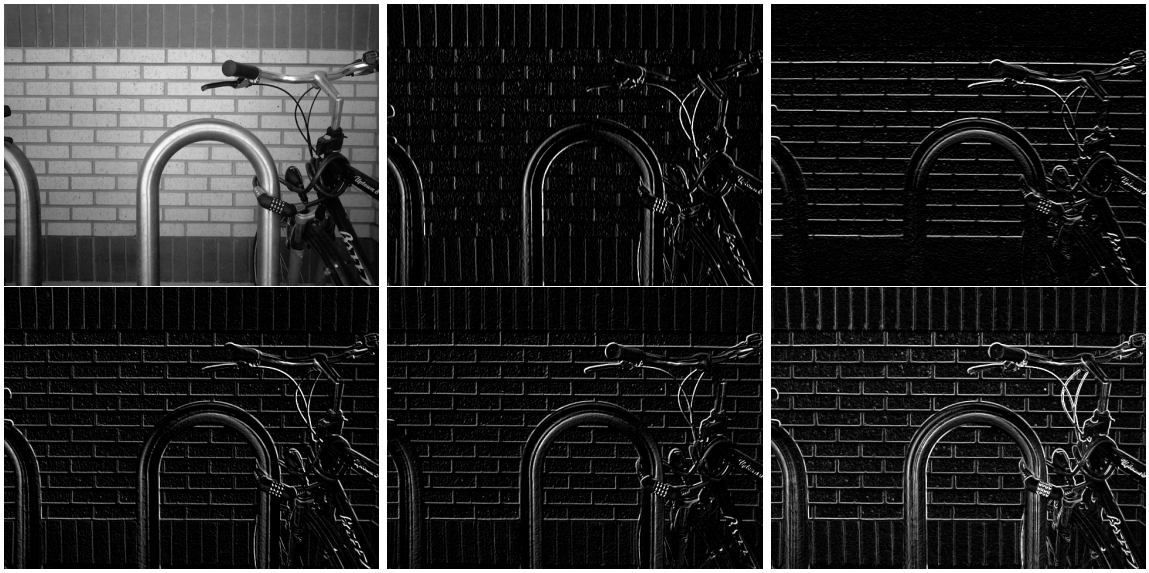

Figure 9: Applying Kirsch's operator to an image without any other modifications.

Figure 10 shows the usage of the Kirsch's operator on Lena's image. Again, all 4 separate and the combined images are shown in this figure. If one pay attention to the third and the fourth images, he can clearly see that these images are the results from the rotated filters because the edges with 45 degree headings are highlighted much stronger than other edges.

VIII. Laplacian Operator

In this section, I have implemented the Laplacian operator on discrete images. This operator creates a new matrix out of an image which can be shown as an image but it might be meaningless because we are mapping the resulting matrix to the visible area with pixel of maximum value of 255. Laplacian operation can be done using different filters but the operator used in ColonD Image Processing program is as follows:

$$F1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

At the rest of this section, I will provide some examples of the Laplacian operator applied to different images. Figures 11, 12 and 13 are depicting these examples.

IX. Guassian Filter

This is another Smoothing (low-pass) filter which is implemented in this program as an extra feature. Some examples of this filter applied once, twice and three times to one image is depicted in Figure 14.

Figure 10: Applying Kirsch's operator to Lena's image without any other modifications.

Figure 11: Applying Guassian filter and Laplacian filter to the Bike's image.

Figure 12: Applying Gray-scale conversion and Laplacian filter to the Lena's image.

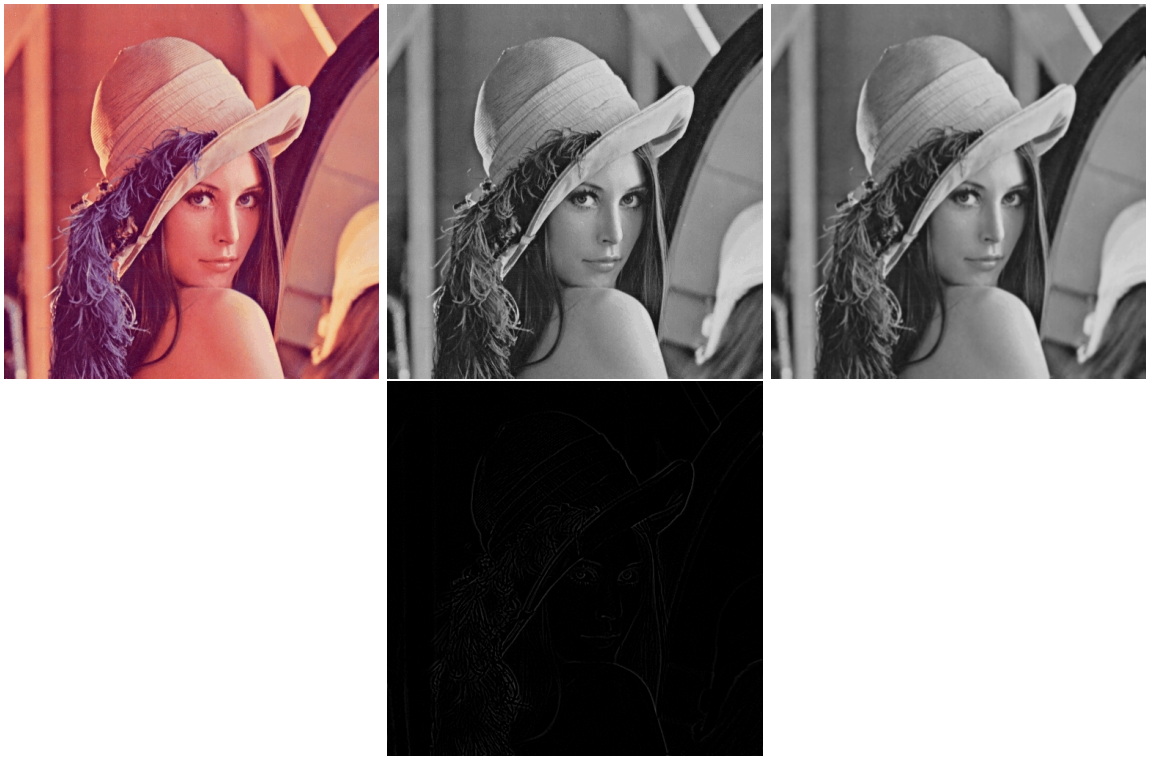

Figure 13: Applying Gray-scale conversion, Guassian filter and Laplacian filter to the Lena's image. Applying just one step of the Guassian filter has a drastic effect on the result of the Laplacian operator. The highlighted areas get faded and less strong.

X. Conclusion

In this phase of the project, I implemented many high-pass filters which try to flatten out parts which have no significant changes in the pixel values but emphasize parts with rapid change in the pixel values.

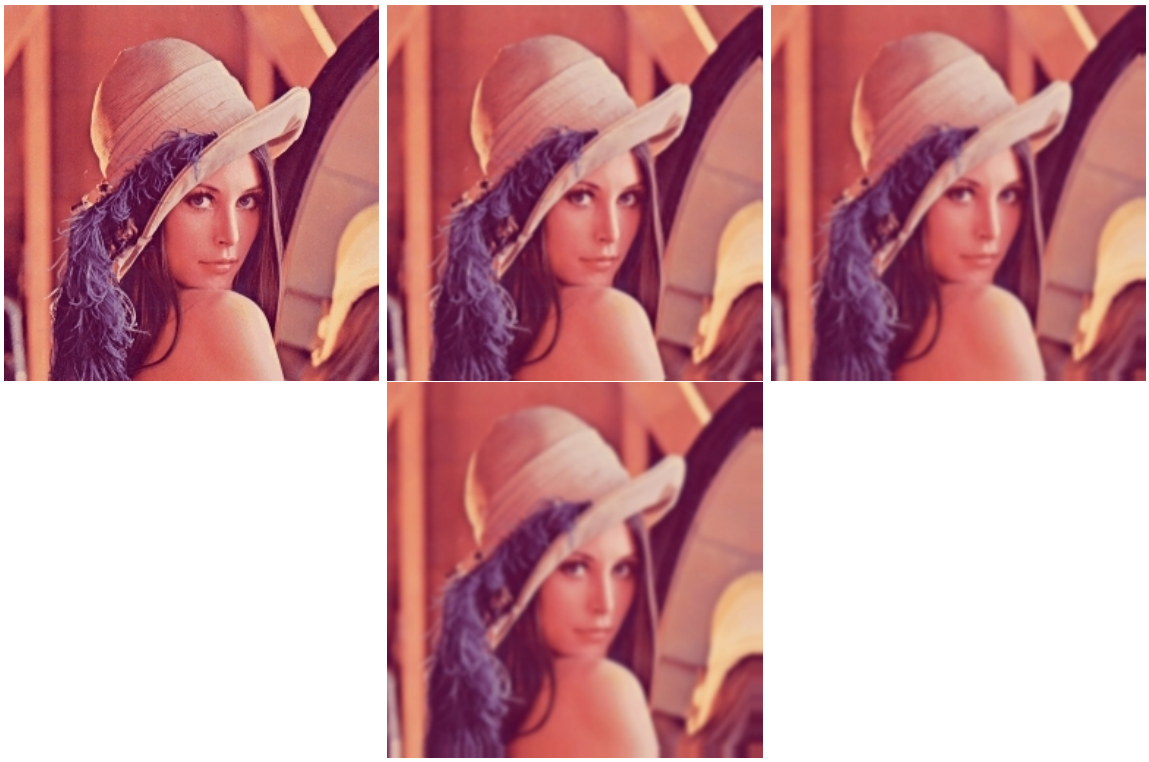

Figure 14: Applying Guassian filter once, twice and three time to the Lena's image.