B490/659 Project 3 Report

Yu Deng:deng4@indiana.edu

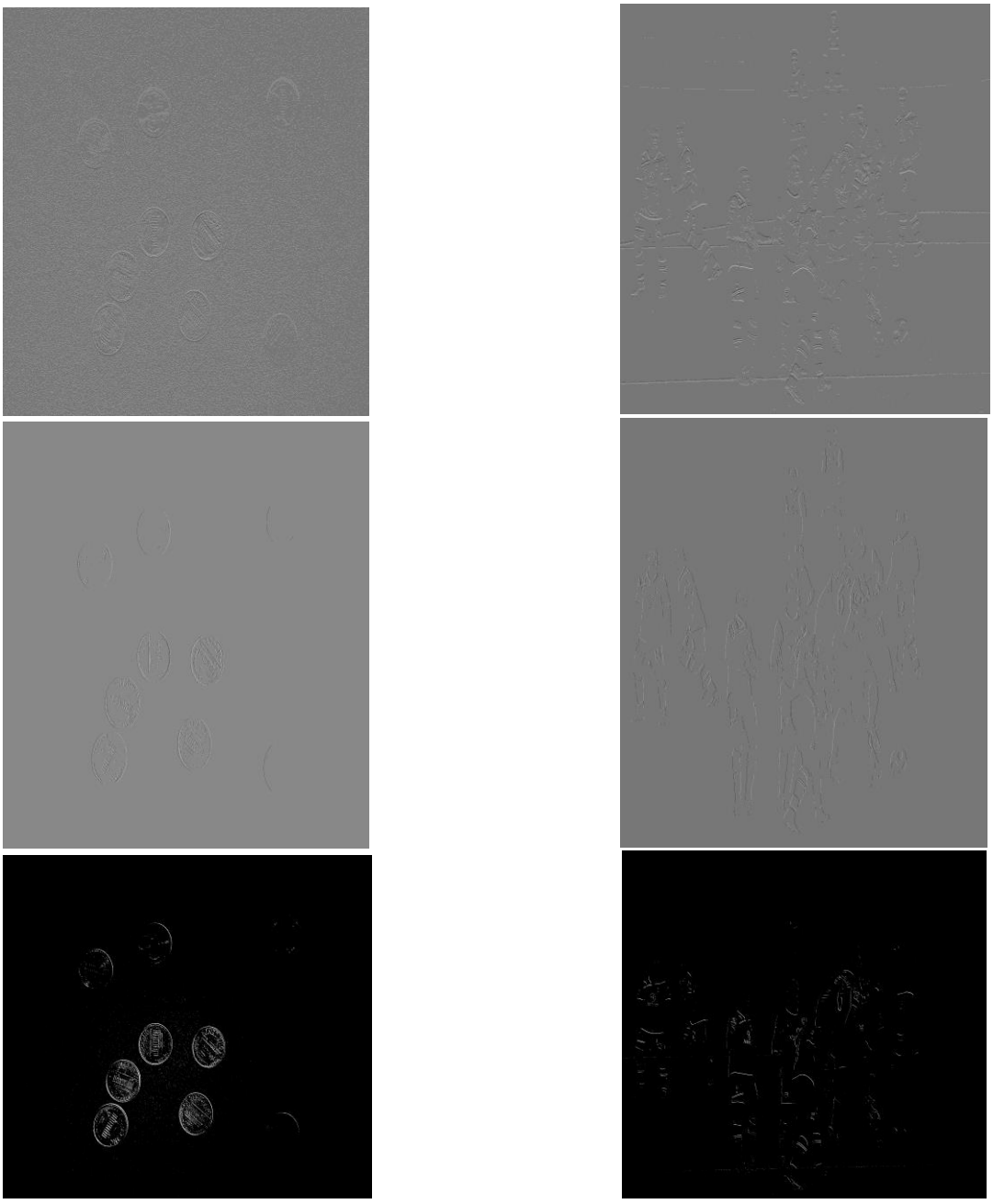
Qiuwei Shou:qiuwshou@indiana.edu

Part 2: Detecting edges and circles

a. Implementation of the Edge Detector

A simple approach is to estimate partial derivatives in the x and y p directions (Ix and Iy) using the Sobel operator, compute the magnitude of the gradient at each pixel, and then threshold Im to produce a binary image.

- Firstly, I use the filter function in the previous project to generate the recommended Gaussian Filter and as well use the recommended Sobel operator to implement this part.
- Secondly, according to the requirement, it asks to estimate the partial derivatives in both directions. So after we got the derivative of the input image on both directions and applying convolution to them. We can get the magnitude of the gradient.
- In the pre-setting, if the pixel is larger than 200, then make it white, otherwise make it black.
- In this part, besides only calculate Ix and Iy, we still produce the image of Sx and Sy in this part (the intermediate images which would be used in finding circles by Hough transform). For your information, Sx is the edge change in X direction and Sy is the edge change in y direction. Sx.jpg and Sy.jpg will be produced and automatically used in the part b.



These images above shows the original picture, Sx, Sy and the final edge detector output for two different images. The first one is the 1.jpg from the test images. The second image is a random image shows a certain moment on a soccer game.

b. Circle Detector using Hough Transform

The general idea is to let each image pixel vote for a set of points in the Hough Space. Once the votes are accumulated, we can detect circles in the image by finding peaks in the Hough Space. And before that, we have to make sure that the Hough accumulated space should be three-dimensional (row, column coordinates and the radius). And to reduce the size of the space we have to assume that we are only interested in circles having radius within a reasonable range.

- Firstly, preset the minimum radius to be 10 and the maximum radius to be 60. And set threshold value to be 2 to determine whether a circle is detected when doing voting.
- Then we define the parameter a,b,r to create an accumulator in 3d, and since in the previous part we already store the Sx.jpg and the Sy.jpg so in this part, we can directly use them to calculate the direction map.
- To calculate the directional map, we make use of the already calculated convolutions of the image with the horizontal and vertical Sobel filter. For every detected edge point [i,j] we calculate the angle θ as

$$\theta[i,j] = \tan^{-1}\left(\frac{\partial f}{\partial y}[i,j] \Big/ \frac{\partial f}{\partial x}[i,j]\right) = \tan^{-1}\frac{\text{sobel}_{vert}[i,j]}{\text{sobel}_{horiz}[i,j]}$$

  Using the function atan2 in C gives angles in the interval $(-\pi,\pi)$. However, since edges with angles θ and $\theta + \pi$ have got the same direction, we can shift all θ[i,j]'s so that they lie in $(-\pi/2,\pi/2)$.
- Then we do the accumulation into (a,b)-space using circular Hough transform. The idea of the Hough transform is that perpendiculars to edge points of a circle cross in the center of the circle. Therefore, if we draw perpendicular lines to every edge point of our edge map, we should obtain bright 'hot spots' in the center of the circles. We therefore draw following line segments into the (a,b) space:

$$\left.\begin{aligned} a &= r\sin\theta \\ b &= r\cos\theta \end{aligned}\right\} \text{ where } r \in (minr, maxr)$$
$$A(i \pm a, j \pm b) \leftarrow A(i \pm a, j \pm b) + E(i,j)$$

  This transform will create spots with higher brightness in places where centers of circles should be found. However, these spots can be quite diffused and dimmed.
- Then we do the implementation in the r-space. After the localization of the circles' centers, the circles' radii have to be found. This is done by accumulating in a one-dimensional space which coordinate is the radius of concentric circles with the given center. For every (discrete) radius in the desirable range (minr, maxr), the sum of edge strengths E() for the points P along the circle with the given radius is calculated:

$$R(r) = \sum_{P \in \text{circle}(r)} E(P)$$

- The r-space is then thresholded, which creates separate intervals with non-zero values. Finally, we can generate the result image by finish the implementation by both (a,b) space and the r-space.

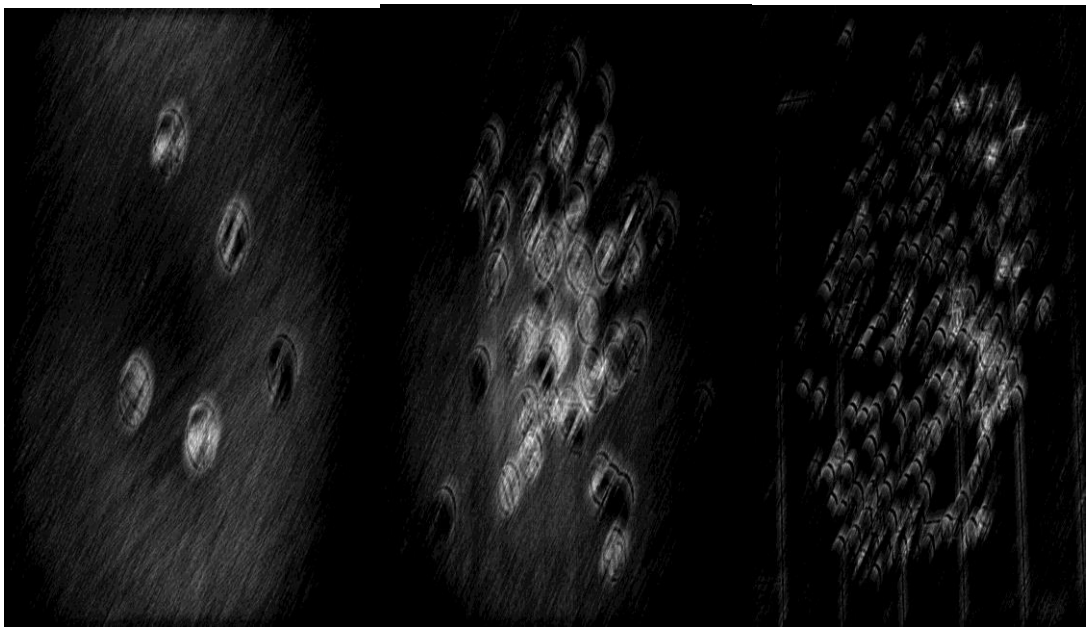
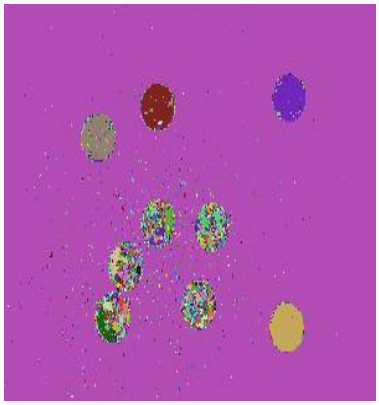


Figure above is test images 3,4,10 and the output image after the “circle detector” operation. From the result we can see that the result is pretty good when the number of circles is small. When the number of coins is really large, the result of circle detector tends to be somehow blurred, which is caused by the overlap of coins. And the light spot is designed to indicate the center of circles after Hough Transform. We can see that where there are lot of coins, that spot is pretty bright.

Part 3 Finding Circular Regions

a. Implementation of simple segmentation problem

- In the pre-setting, all the images that are larger than 300*300 are resized. Keep the longest side to be 300 long maximum. And set the threshold of the average difference to be 1.5
- For every pixel, link it to the node below and to the right of it (if possible). Then assign a PID to every pixel, and put the current pixel and the pixel linked it to into a pair. After that we calculate the edge cost using the formula in the requirement. Finally we put the pairs in a map such that we generate a “score” for the node pairs.
- Then we sort the image by every pixel and store each value to a vector. Since the simple algorithm calculates the average difference between pixels. Do merge when the difference exceed one and a half of itself until all other value between the neighborhoods is less than the threshold.
- After that, do the real segmentation by doing sort over the vectors that store pixel values. Each time update the PID after doing merge.
- Assign random color to each segmentation created. Then output the final image



The figures above is the result after simple segmentation

b. Modified Segmentation Algorithm

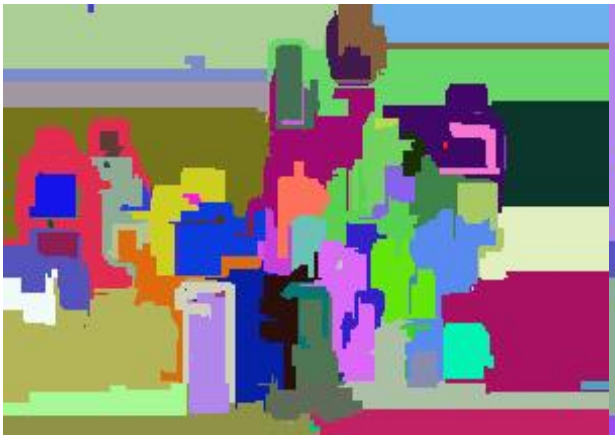
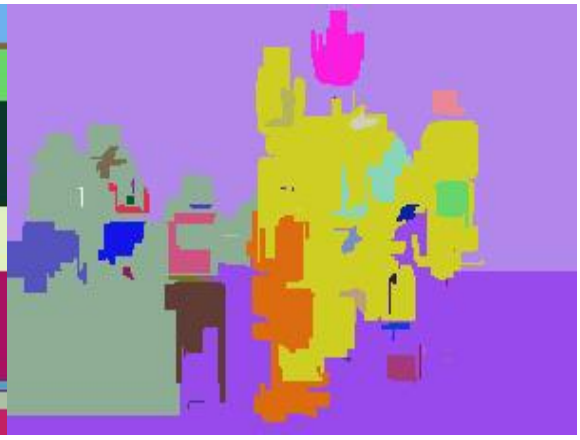
First, apply a Gaussian kernel to each RGB color plane of the image ahead of time, in order to reduce noise in the edge weights before segmentation. Second, instead of simply joining connected components based on edge weight, use a slightly more complicated rule. That is, join two connected component using

$$w_e \leq \min(N(C_1) + \frac{k}{|C_1|}, N(C_2) + \frac{k}{|C_2|}),$$

Where we denotes the weight of edge e, k is a constant parameter, |C| denotes the size of component C (i.e., number of vertices in the component), and N(C) denotes the maximum edge weight in the minimum spanning tree of C.

- In the pre-setting, besides the previous part, a constant parameter k is introduced as an additional input.
- When calculate the threshold, using the equation above to implement.



The Figures above shows the modified segmentation with k=50,500 and 5000

c. Detecting Circular Regions

We failed to complete part c in the code.

But the logic and idea to implement this part would be like this

- Calculates the area of every segment and stores them in a map by counting how many pixels belong to each segment.
- Calculate the center of gravity of each segment.

- Calculates the perimeter of each segment by using the algorithm introduced in the Burger & Burge book.
- Determine a reasonable threshold about detecting potential circles to accept or deny segment regions.
- Detect circles in a image

Part 4 Finding Circular Regions

We failed to complete part c in the code.

But the logic and idea to implement this part would be like this

- First, use segmentation algorithm to determine what radiuses used in the Hough transform later on
- Use Hough Transform to implement a Hough Detector used to detect potential circles with pre-setting radius
- Use the output of the Hough Detector to determine the coin numbers