CS7647 Computer Vision

Assignment 2

Edges and Lines

Kilver, Jacob R
9-7-2015

1. Question 1

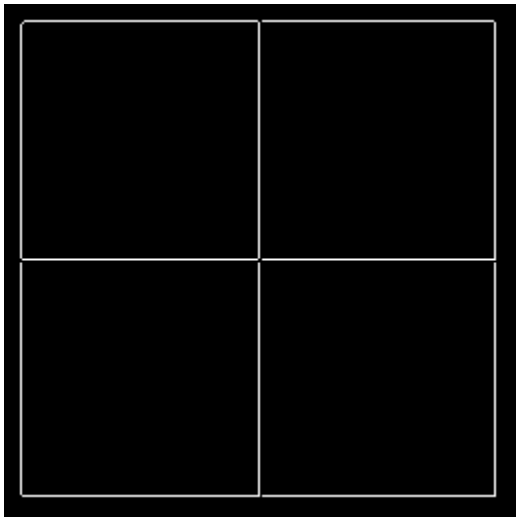

Figure 1: ps2-1-a-1.png

2. Question 2

a. Hough Accumulator

Figure 2: ps2-2-a-1.png

b. Hough Peaks

Figure 3: ps2-2-b-1.png

c. Hough lines

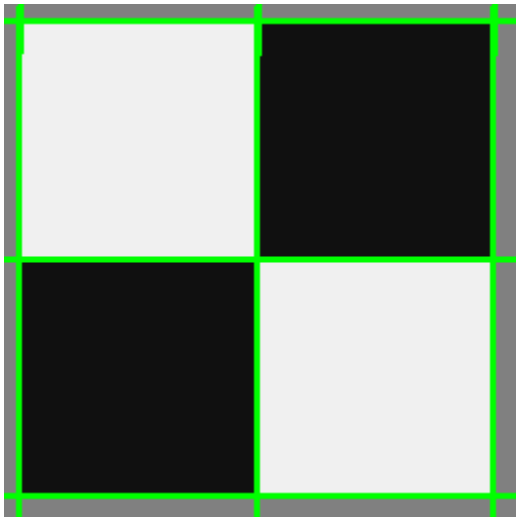

Figure 4: ps2-2-c-1.png

d. Questions

I used the default bin size of $\rho=1$ and $\theta=\pi/90$, so the bin sizes were rather small. This was to ensure accuracy, especially in this case where noise was not present. Peak finding was done in a strict ordering fashion, so the neighborhood size was effectively a size of 1 pixel.

3. Adding noise

a. Smoothed image

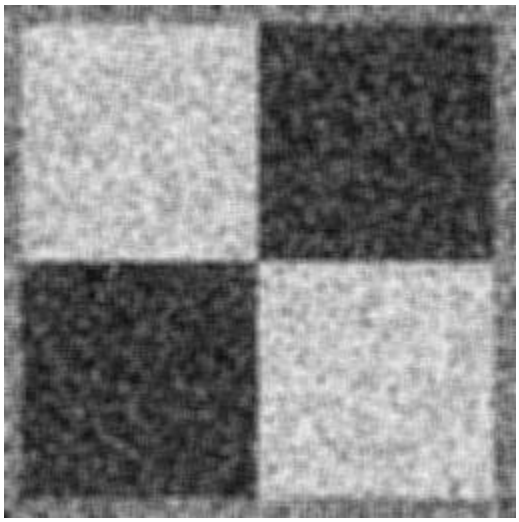

Figure 5: ps2-3-a-1.png

b. Edge images

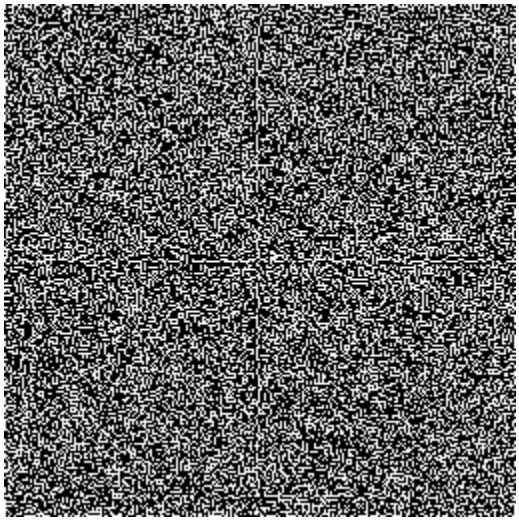

Figure 6: ps2-3-b-1.png (Noisy lines)

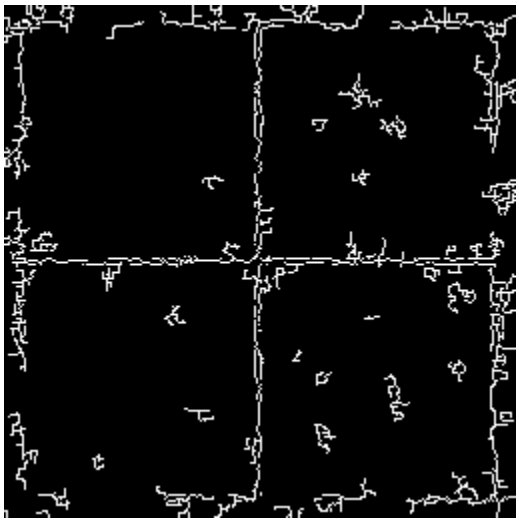

Figure 7: ps2-3-b-2.png (Lines on smoothed image)

c. Hough lines

Figure 8: ps2-3-c-2.png

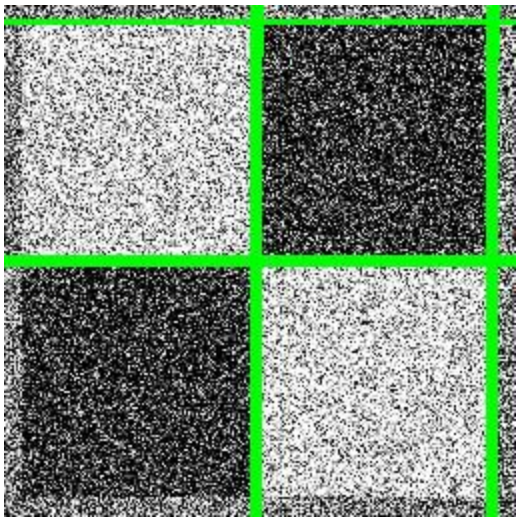

Figure 9: ps2-3-c-2.png

Unfortunately I was unable to find a combination of smoothing and edge finding that could identify the edges on the left and bottom sides. From the edge detection picture (Figure 7), you can see that the edges are not as clear on the left and bottoms sides as on the other sides. I tried more smoothing and adjusting the thresholds on the Canny operator, but neither produced better edge image results than what is included in the code listing. As such, no threshold value for the peak finder could allow the left and bottom lines because the Hough accumulator never found them in the first place.

4. More difficult lines

a. Smoothed image

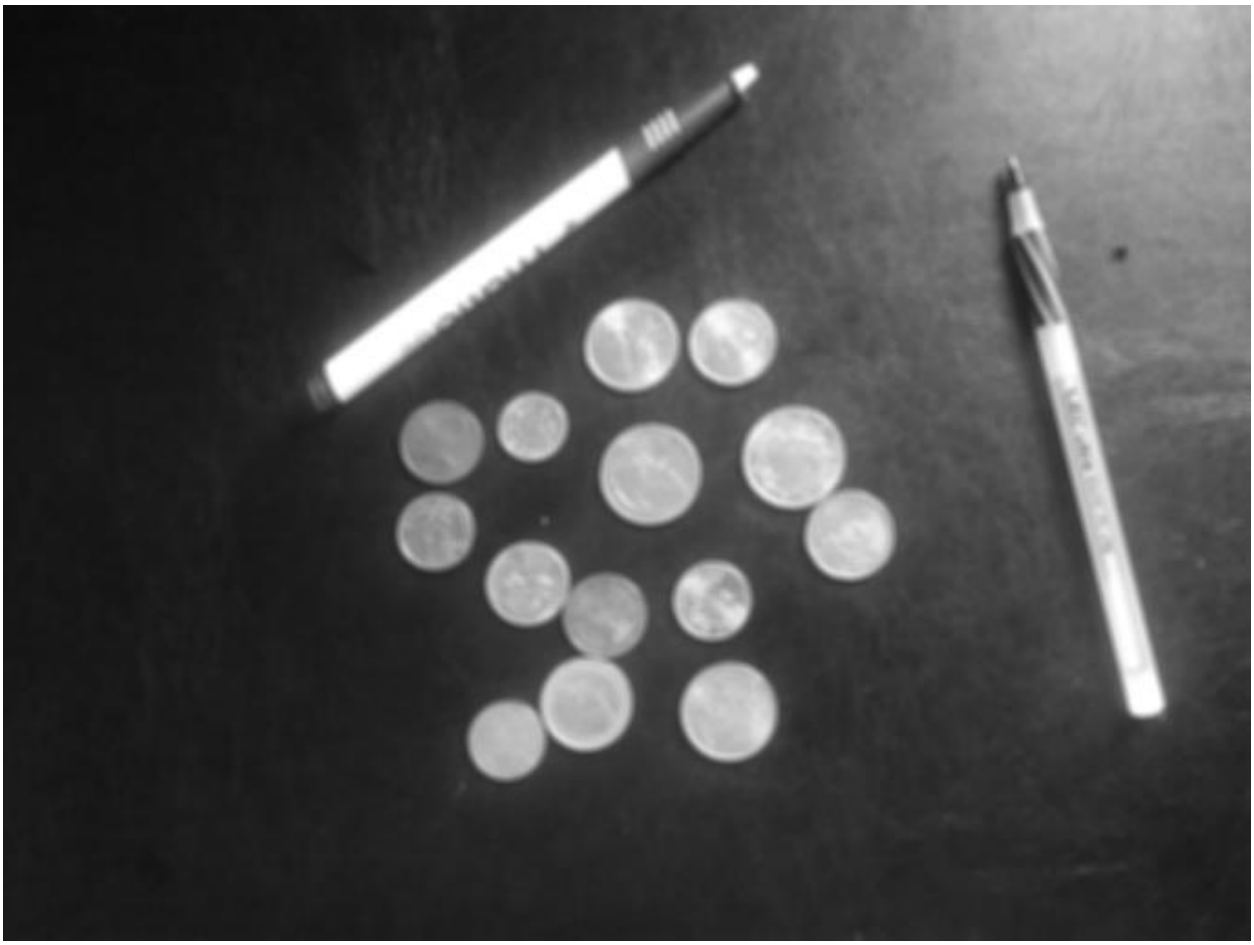

Figure 10: ps2-4-a-1.png

b. Line image

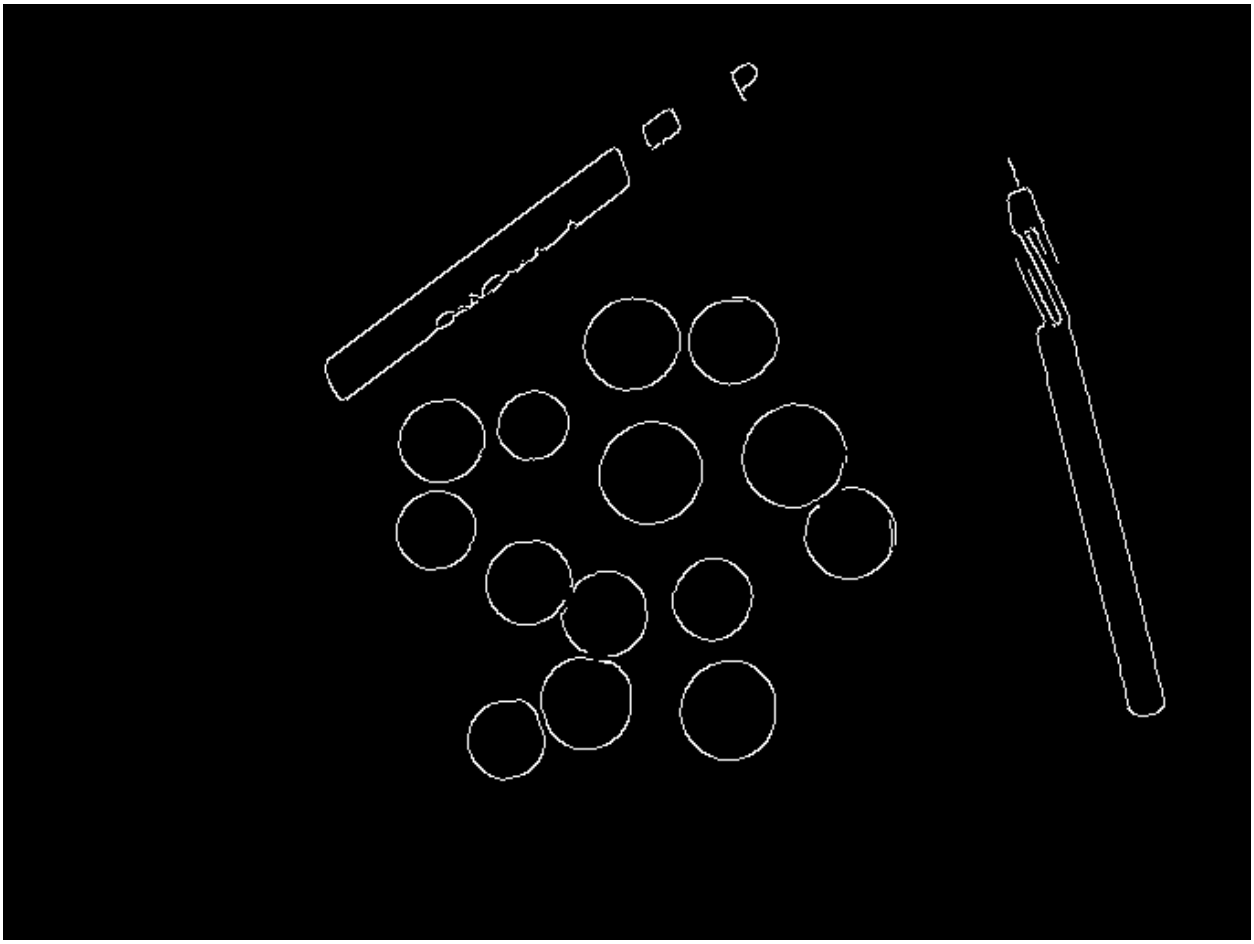

Figure 11: ps2-4-b-1.png

c. Hough lines

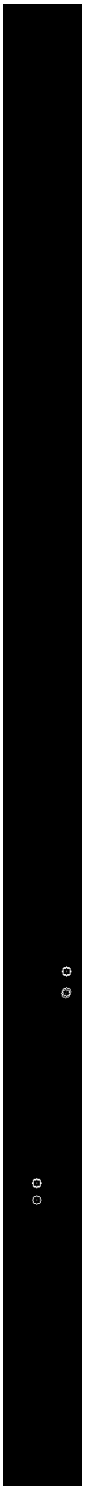

Figure 12: ps2-4-c-1.png

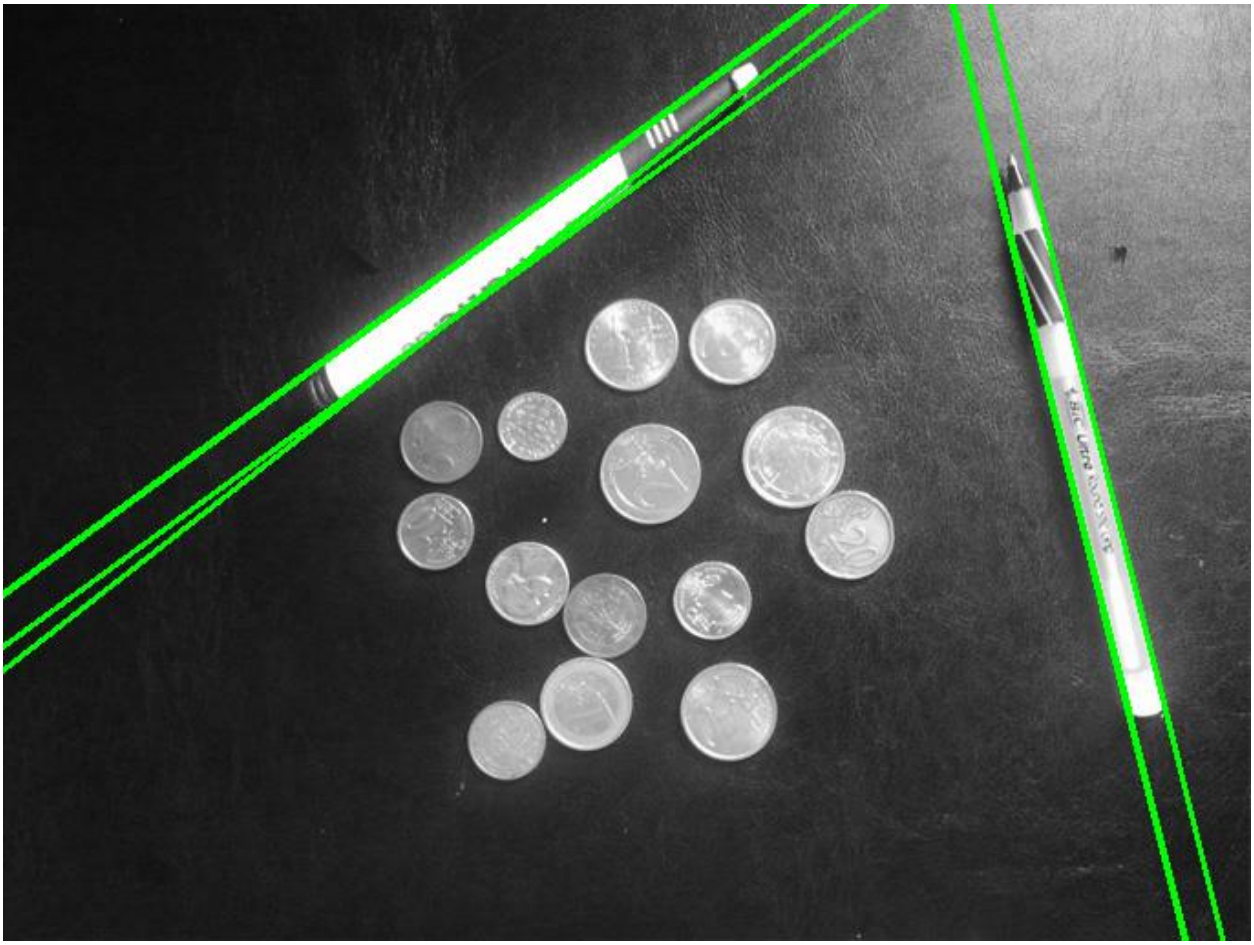

Figure 13: ps2-4-c-2.png

There were not that many lines present in this image, so not much image manipulation was required to obtain good results. The image was smoothed using a GaussianBlur operation. Then the edges were found using a Canny operator with a upper threshold of 200 and a lower threshold of 100. This found all the important edges in the blurred image (and maybe a few unimportant ones). Finally, the Hough accumulator was applied to the edge image, and the peaks were found using a threshold of 75.

5. Hough circles

a. Smoothed image

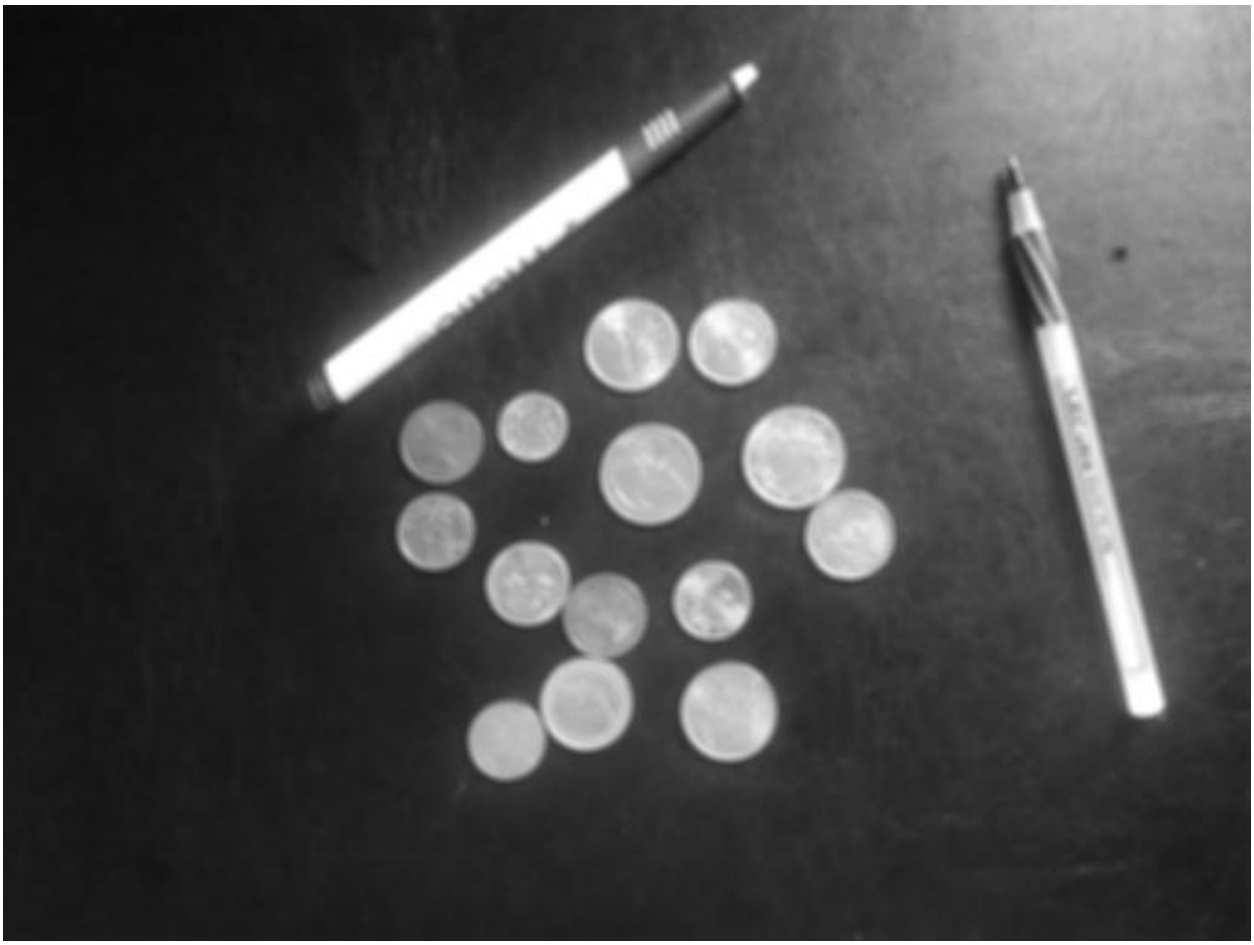

Figure 14: ps2-5-a-1.png

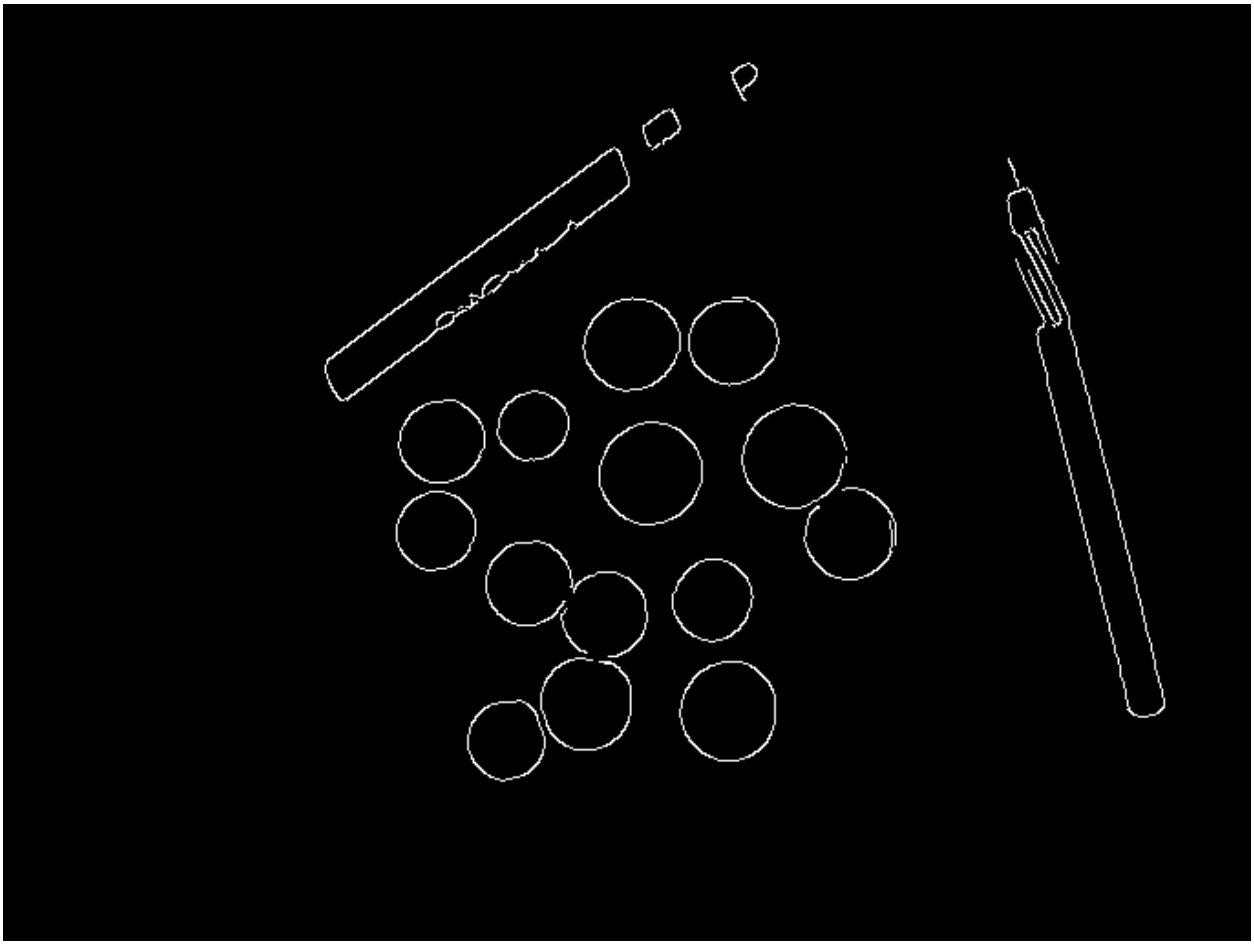

Figure 15: ps2-5-a-2.png

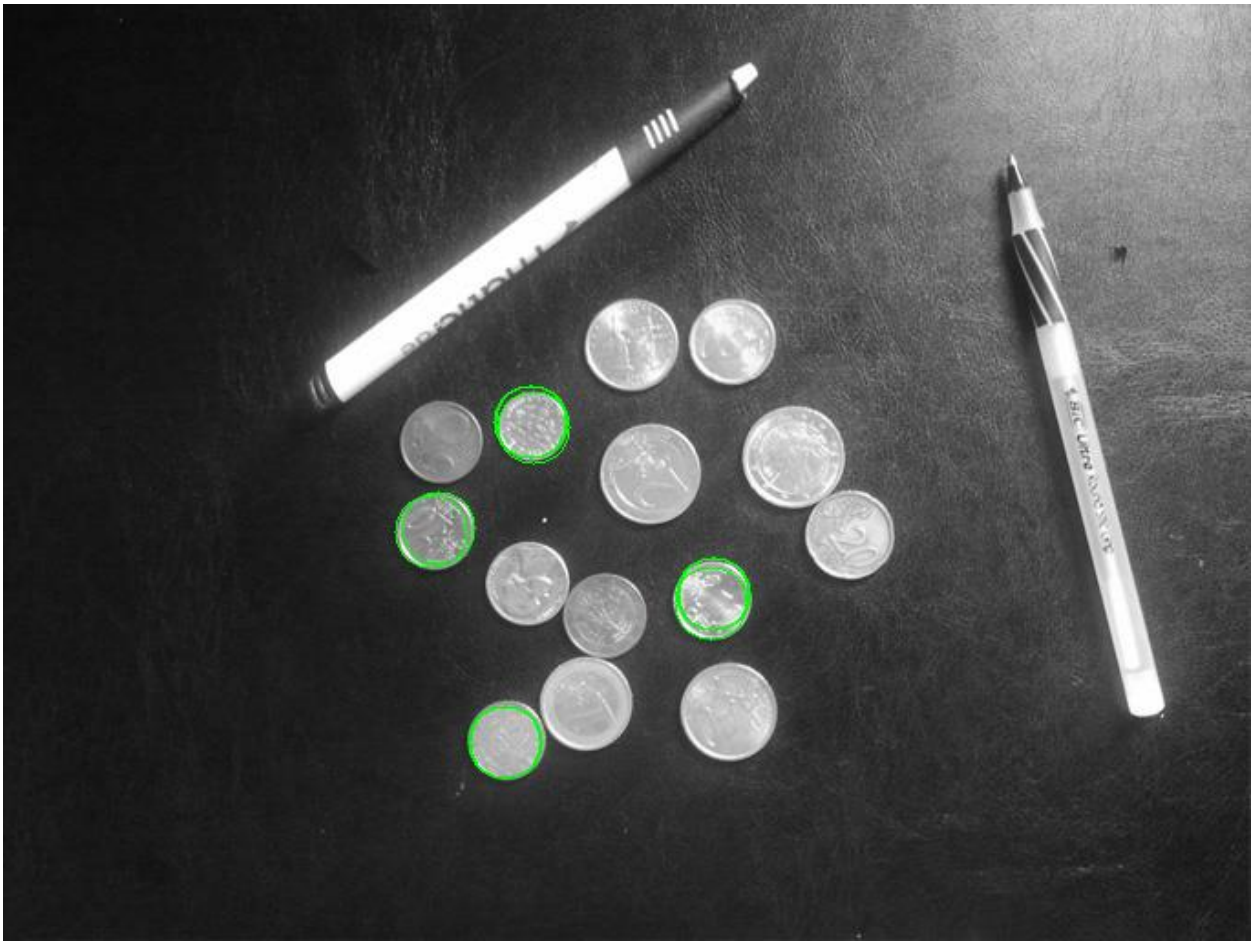

Figure 16: ps2-5-a-3.png

b. Finding circles

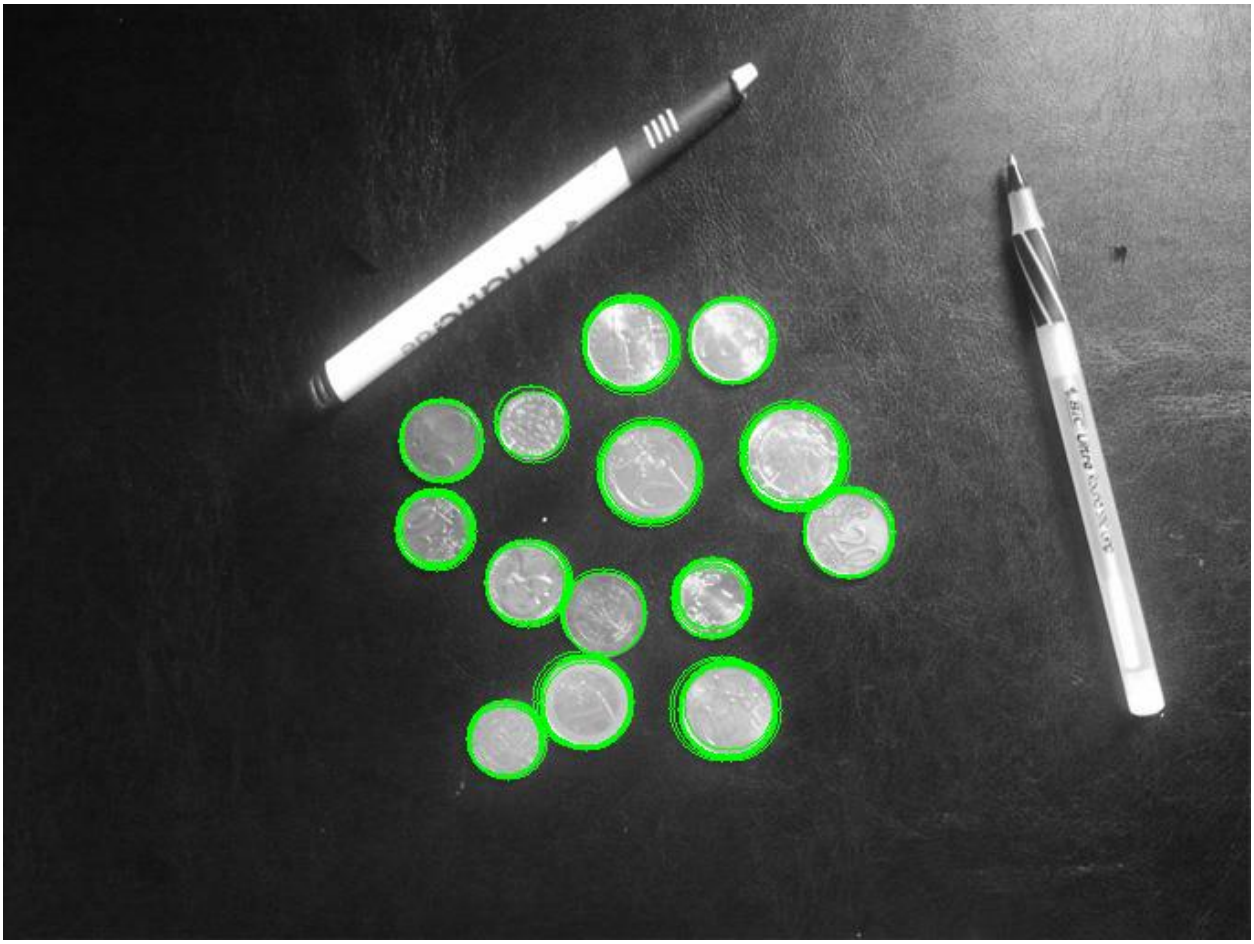

Figure 17: ps2-5-b-1.png

The Hough accumulator for circles was more straightforward to implement than it seemed at first, especially for just a single radius. I created two vectors of the same size holding all the x- and y-coordinates for the edge pixels. Then, using the non-gradient version of the accumulator, I went through every possible gradient direction and created indices into the Hough accumulator array. Since the equation used to calculate the indices could exceed the maximum size of the accumulator array, I had to add a threshold to prevent "index out of bounds" errors. I used the same peak finder as before to find the centers of the circles.

Finding multiple radii was relatively straightforward once the steps described above were complete. The most difficult part was finding a data structure to keep everything organized. I settled on a vector of radii and a list containing the (x,y) pairs for the circle centers. The list was used because it could allow for varying lengths depending on how many centers were found for a given radius. Unfortunately, this list then had to be converted to numpy array to pass to the method for visualizing the circles.

6. Finding lines with clutter

a. Finding lines

Figure 18: ps2-6-a-1.png

b. Discussion

The line finder was not broken, per se, but it did find many lines that are present in the image, just not the lines that we want. There are unimportant lines that do not need to be highlighted.

c. Looking for better results

Figure 19: ps2-6-c-1.png

Unfortunately, my attempts at better smoothing and better thresholding were not able to locate the lines on the second pen, while they did eliminate some of the other lines. Since the lines on the second pen were not found, looking for nearby parallel lines would not have helped finding the lines for the second pen. Furthermore, checking the minimum length of the line would have allowed the lines at the edges of the book to pass. Finding lines in cluttered environments can be quite a challenge.

7. Finding circles with clutter

a. Finding circles

Figure 20: ps2-7-a-1.png

b. Discussion

The circle finder could not locate all the circles due to the image clutter. However, the circle finder did locate a false positive in the 'n' of the book cover (see Figure 21). There is a fairly decent and uncluttered half circle present there, and the circle finder located it. There are a few ways that could be used to prevent this, but I was unable to implement neither of them. First, a better threshold might be able to prevent this. Since there is only a half circle present, the center would not get as many votes as a full circle would. However, in this image filled with clutter, such a strategy might filter out other circles as well. A second option would be similar to checking the distance of the line discussed for finding Hough lines. The ratio between the radius and the circumference of the circle could be checked to make sure that only circles where most of the shape is actually present are selected.

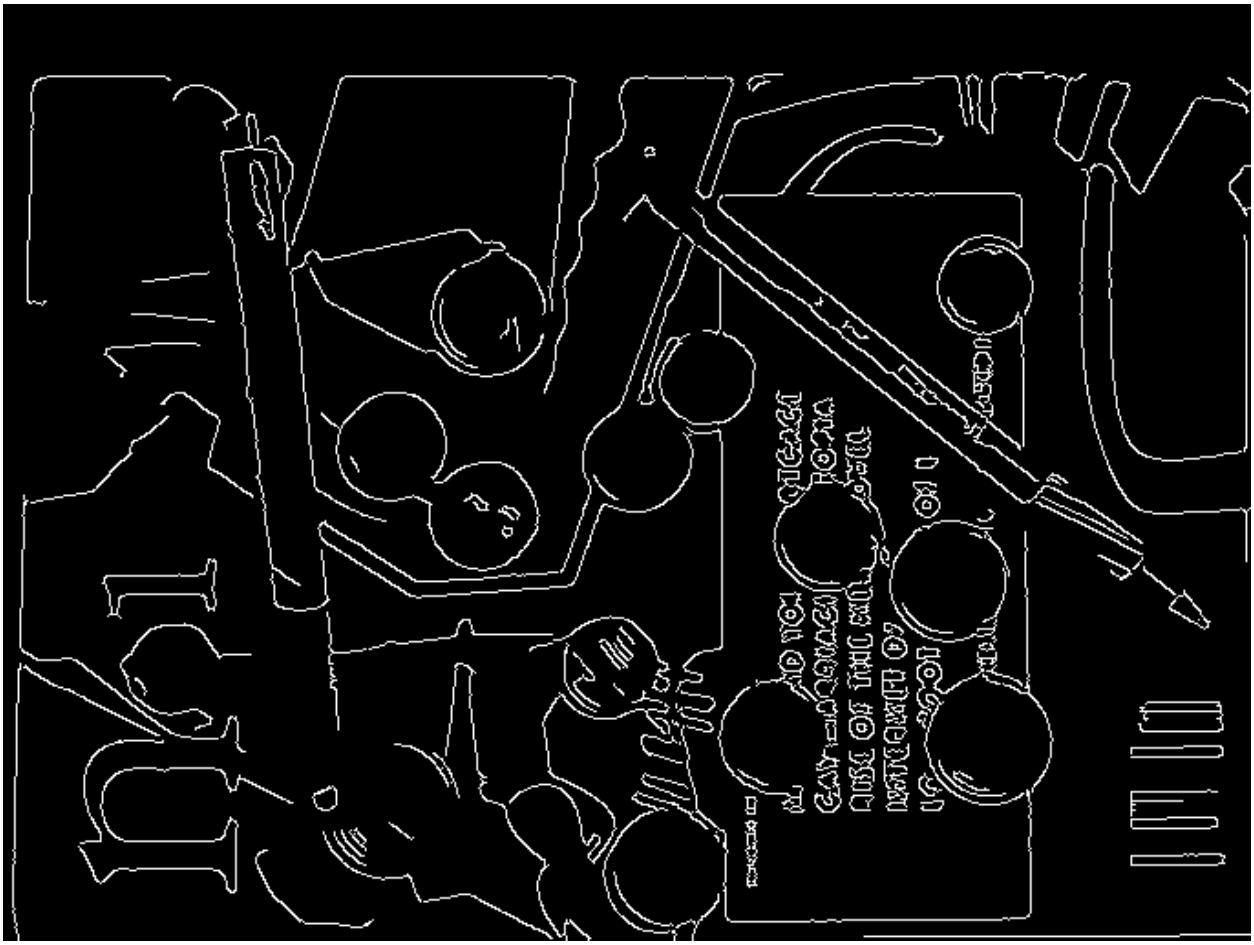

Figure 21: edges in clutter image

8. Distortion

a. Finding lines and circles

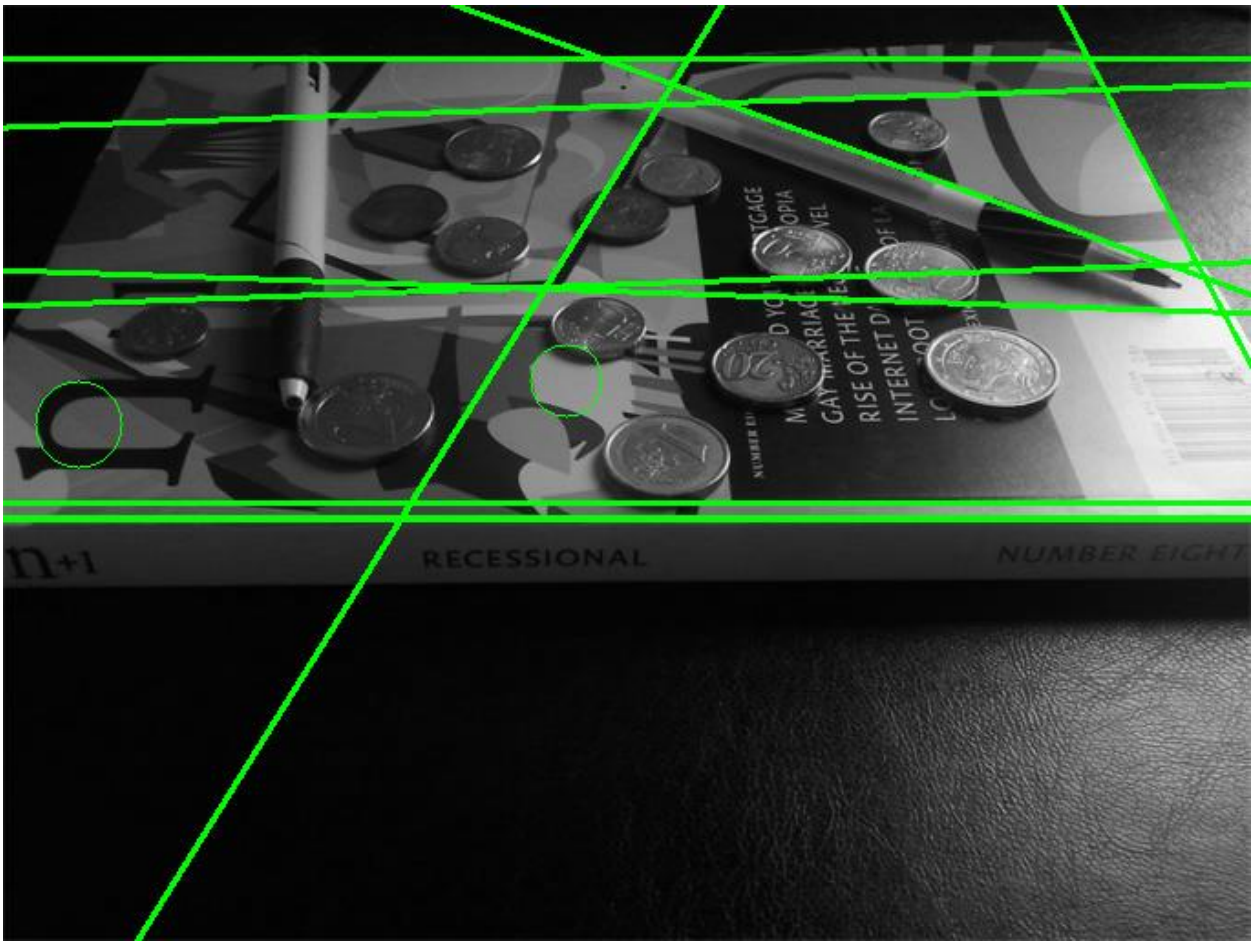

Figure 22: ps2-8-1.png

b. Fixing problems

In addition to the methods described above for eliminating false positives, the primary issue in this distorted image is that the circles have become ellipses. The circle finder will not be able to find these ellipses because they do not fit into the equation of the circle implemented in the Hough accumulator function for circles. A new Hough accumulator function for ellipses would need to be created for this problem. I would expect the ellipse finder to take even longer than the circle finder since there is an additional free parameter (major and minor axis versus radius).