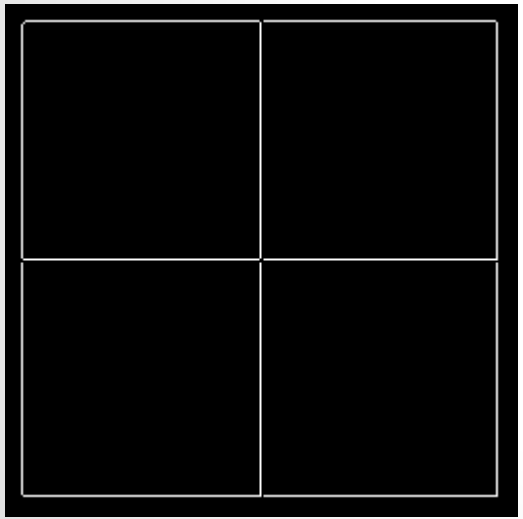
Computer Vision
Spring 2017
Problem Set #2

Ahmad Aldabbagh
aaldabbagh3@gatech.edu

1a: Edge Image

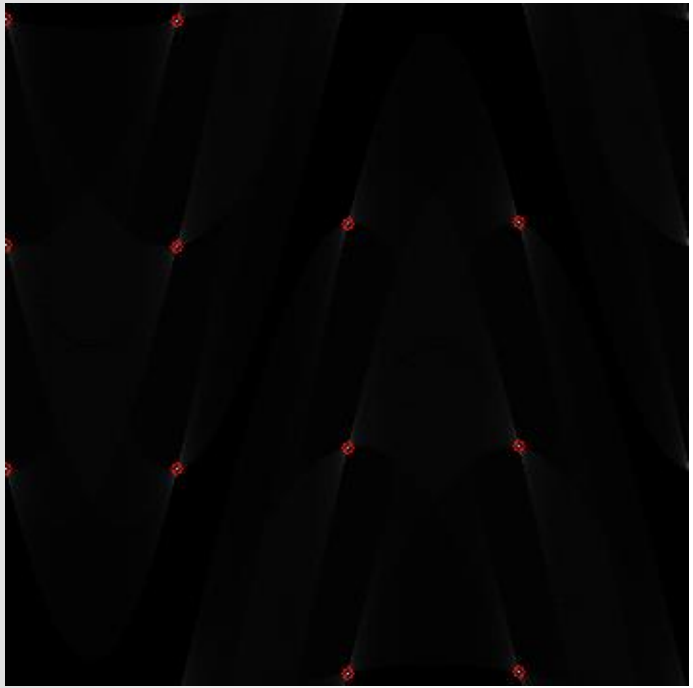


img_edges - ps2-1-a-1.png

2a: Hough Accumulator Array

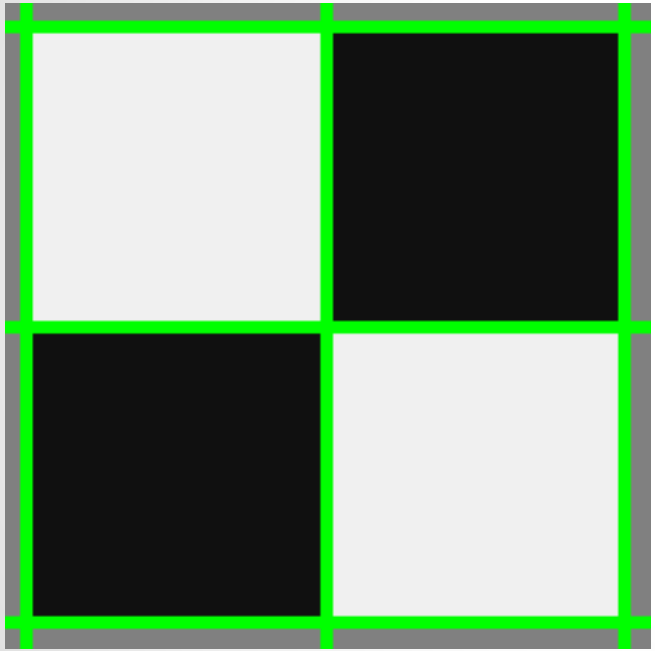
Normalized Accumulator Array - ps2-2-a-1.png

2b: Accumulator Array w/Peaks



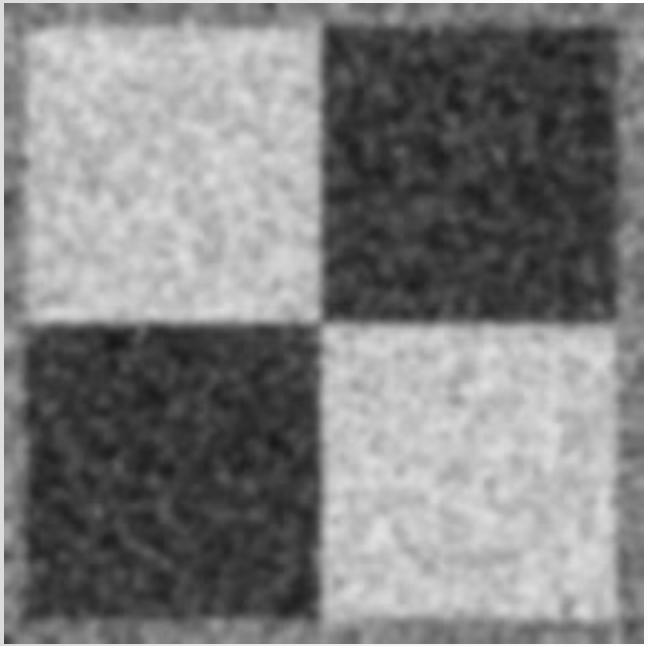
Accumulator Array with peaks highlighted - ps2-2-b-1.png

2c: Image with Lines Drawn



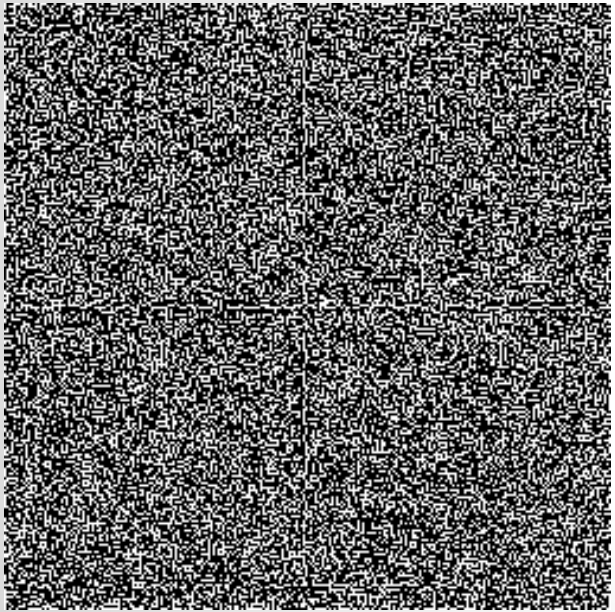
Original grayscale with lines drawn - ps2-2-c-1.png

3a: Smoothed Intensity Image

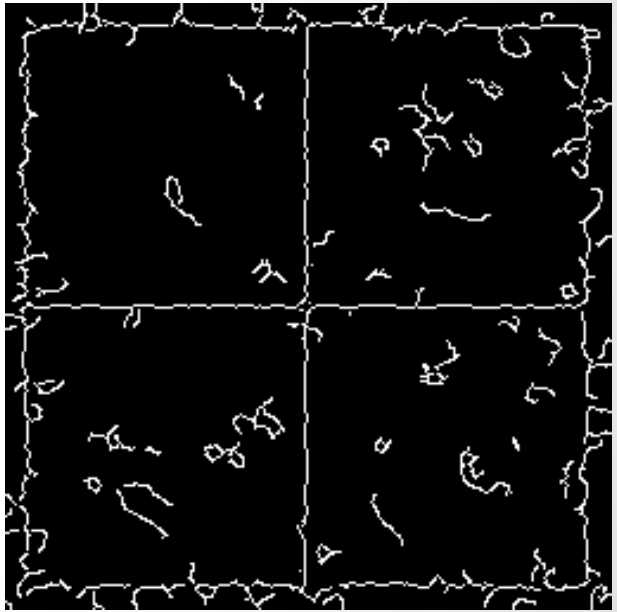


ps2-3-a-1.png

3b: Edge Image Comparison

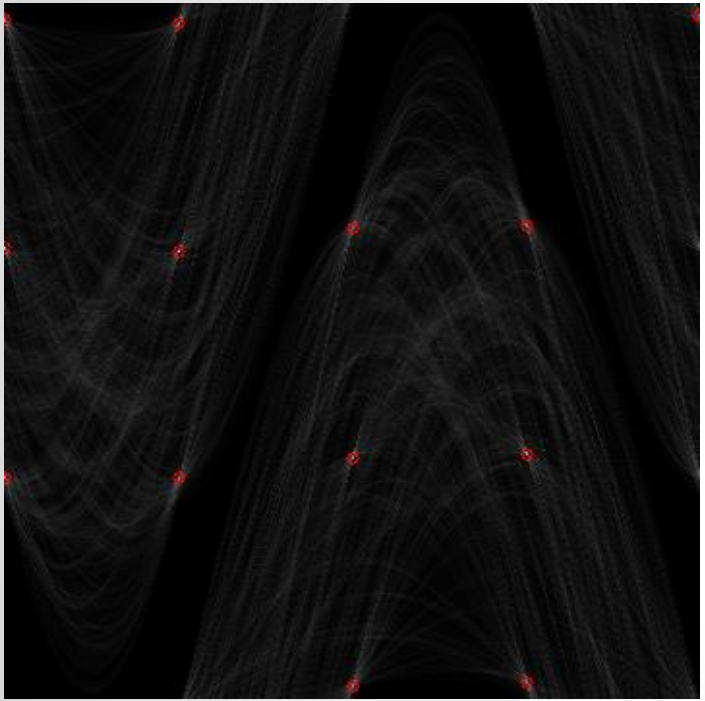


Edge Image (original) - ps2-3-b-1.png

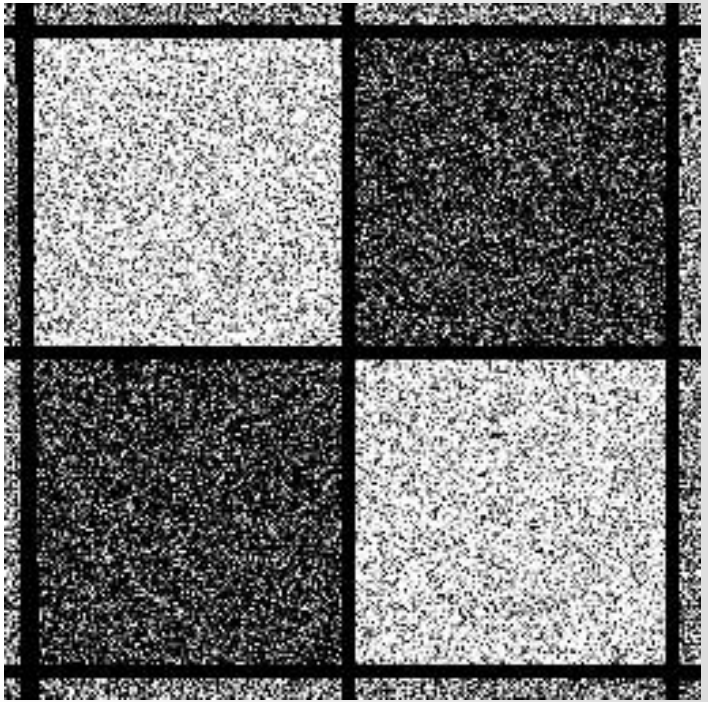


Edge Image (smoothed) - ps2-3-b-2.png

3c: Hough Transform Smoothed

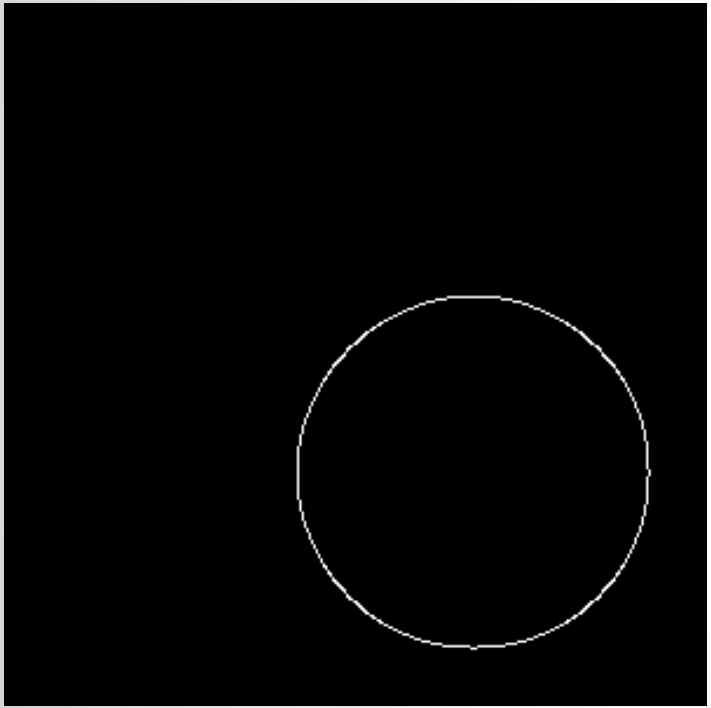


Accumulator Array with peaks highlighted - ps2-3-c-1.png

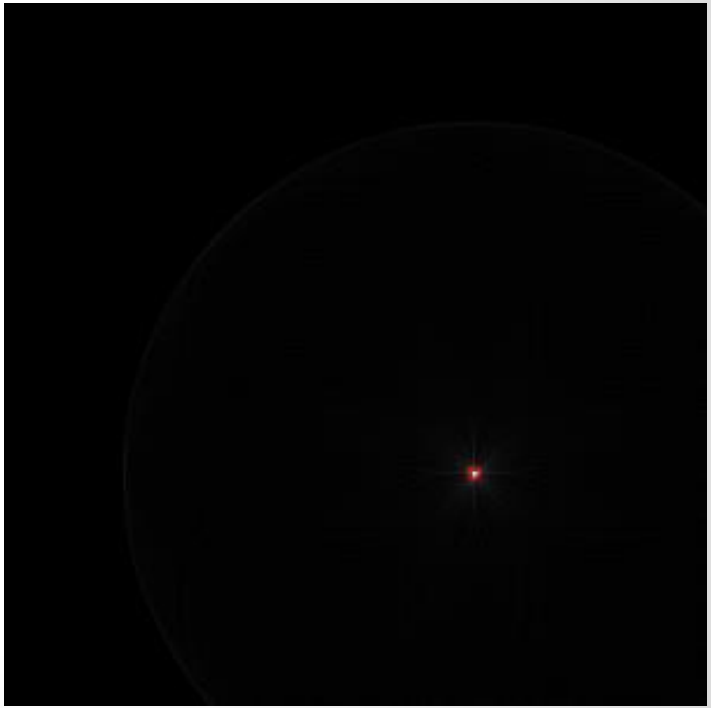


Original intensity image with lines drawn - ps2-3-c-2.png

4a: Single-Point Method

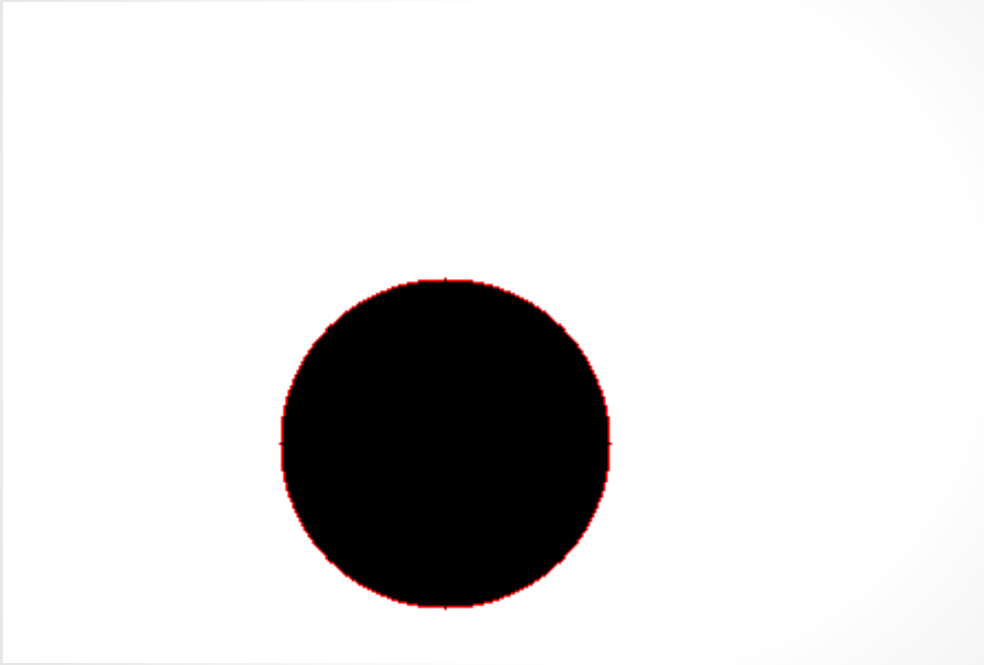


Edge Image - ps2-4-a-1.png



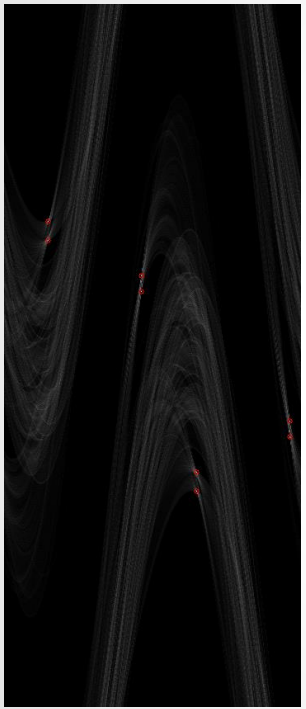
Accumulator Array with peaks highlighted - ps2-4-a-2.png

4b: Point-Plus Method

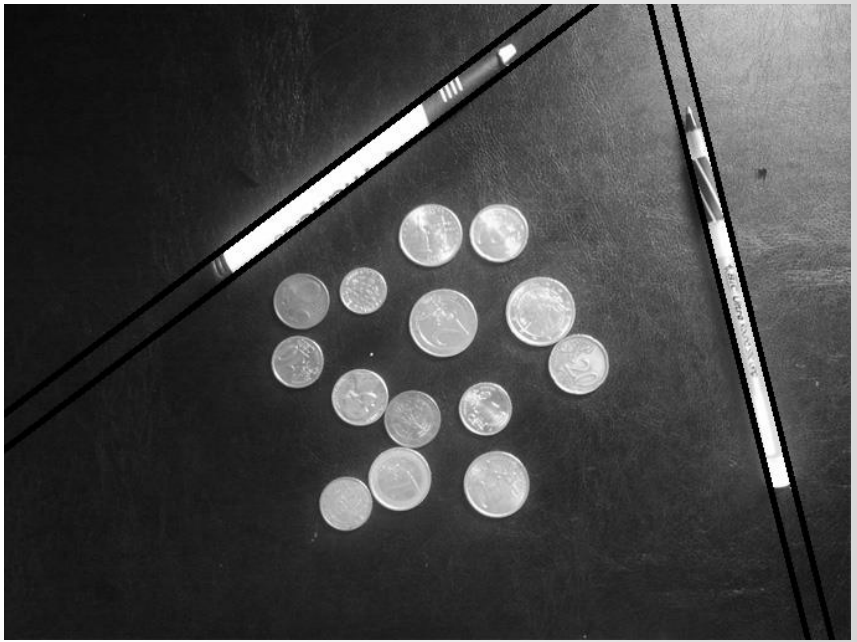


original monochrome image with circles drawn in color- ps2-4-b-1.png

5a: Coins and Pens

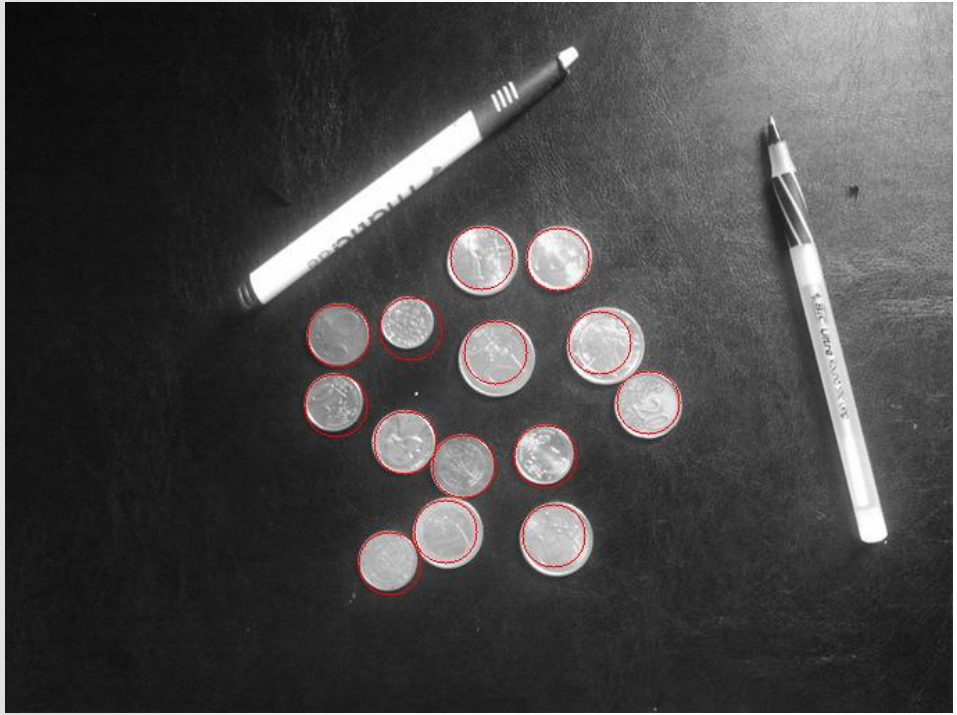


Accumulator Array with peaks highlighted - ps2-5-a-1.png



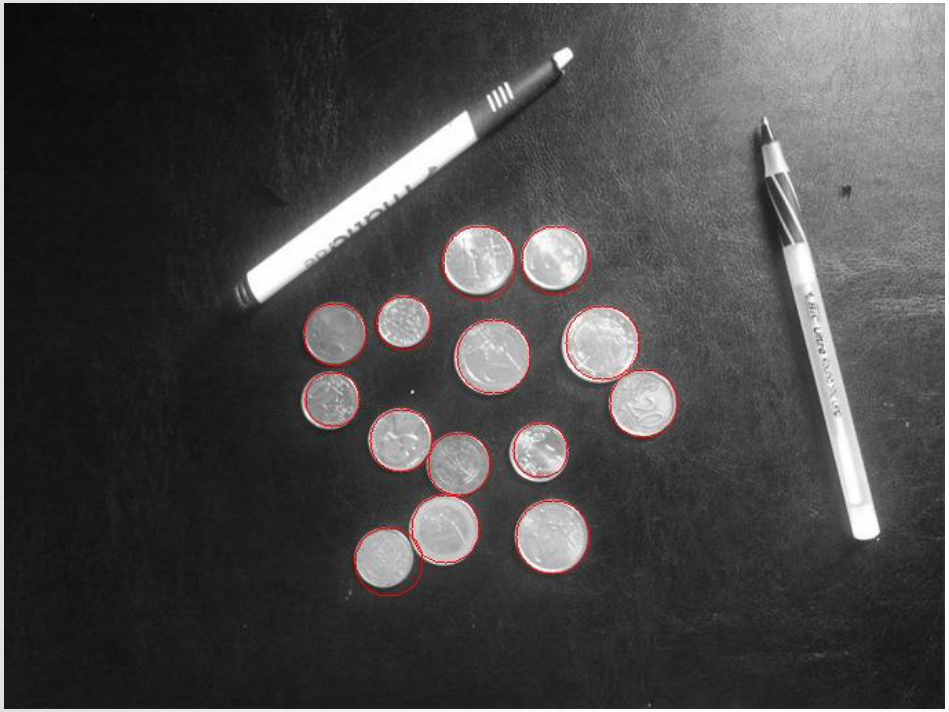
Monochrome image with lines drawn - ps2-5-a-2.png

5b: Coins and Pens: Circles



Original monochrome image with circles drawn - ps2-5-b-1.png

5c: Coins and Pens: Circles (cont.)



Original monochrome image with circles drawn - ps2-5-c-1.png

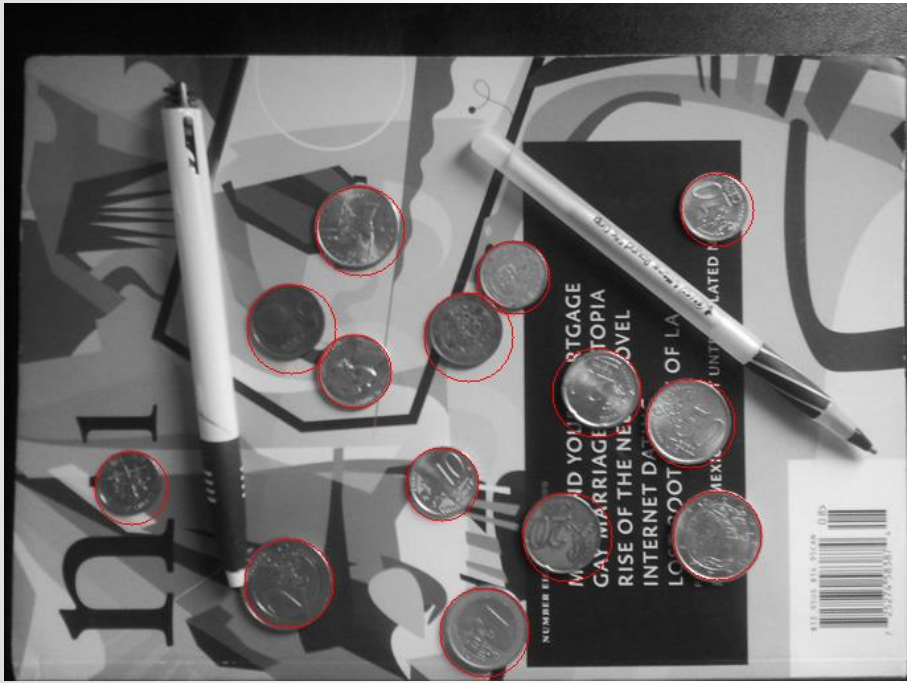
6a: Images with Clutter (pens)

Smoothed image with lines drawn - ps2-6-a-1.png

6b: Images with Clutter (pens), part 2

Smoothed image with lines drawn - ps2-6-b-1.png

6c: Images with Clutter (coins)



Smoothed image with circles drawn - ps2-6-c-1.png

7: Discussion

a. **For each of the methods, what sorts of parameters did you use for finding lines in an image? Circles? Did any of the parameters radically change? (Describe your accumulator bin sizes, threshold, and neighborhood size parameters for finding peaks, and why/how you picked those.)**

For the hough line method, the resolution (bin size) was extremely important to be able to detect the lines and circles correctly. I found that decreasing the resolution degraded the quality of detecting the correct lines. For both line and circle finding, changing the canny edge parameters were important as we sometimes need to remove or keep some details before applying the hough functions. After finding the edges, choosing the right thresholds and neighborhood deltas were extremely important as well. My accumulator bin sizes were 1 for rho and 1 deg for theta, these values were set empirically based on the results obtained by experimenting on the images. As for the thresholds and neighborhood size, these were chosen depending on the problem/image and were also chosen based on the best results. In some instances, we would like to eliminate close by peaks so increasing the neighborhood size would be useful. Also, applying a large threshold would eliminate peaks with a small vote count. Some of the values of threshold and nhood delta respectively are Q2:[120, (25,25)]; Q3:[135, (10,10)]; Q5b:[120, (25,25)].

7: Discussion

b. What differences do you see when finding the edges in a noisy vs. regular image?

The main thing I noticed was in a noisy image, a smoothing filter is required to at least capture part of the edges in the image. Also, it is more difficult to fine tune the canny edge parameters in the noisy image while trying to retain as much information about the lines as possible. When we examine the resulting edges from both, we notice that the regular image has straight and sharp edges that clearly correlates to image while the noisy image has less distinctive edges and are quite crooked with the presence of points that do not represent edges. The thresholds in the regular image's canny edge detector were larger than that of the noisy image.

c. Is there any perceived difference in time or computational cost between the single-point and the point-plus gradient method implementations?

Yes it is quite noticeable that the point-plus method runs much quicker than the single point method. This is quite expected since the single point method loops through a large array angles (360 degress) for every x,y point while the gradient method only runs twice (angle and angle+180)

7: Discussion

d. For question 5, were you able to find all the circles? Describe what you had to do to find circles.

Yes I was able to find all the circles. I had to tweak the value of my threshold and nhood values to capture all the circles and get rid of multiple detections and false positives.

e. In question 6, for a cluttered image, you likely found lines that are not the boundaries of the pen. What problems did you face to overcome this?

That is correct. The first thing done was to apply a smoothing filter. This along with parameter tuning, reduced the lines that are not related to the pen. However, there were some lines remaining. To remove them, I looked at the peaks and noticed that the pen edges should be parallel (have very close theta) and close to each other (difference in rho should be small). I created a function that puts an upper limit on difference in rho and theta between peaks and eliminates those that are far or do not have a close peak to form a pair.

7: Discussion

f. Likewise in question 6, there may have been false positives for circles. Did you find such false positives? How would/did you get rid of them? If you did these steps, mention where they are in the code by file, line no., and also include brief snippets

To try to cover the different sizes of the coins, I used different radii values. I also played with the edge detection thresholds to remove most of the image's edges while trying to retain as much of the coins as possible. Removing false positives was achieved by playing with the threshold and nhood values. Finally, I created a function to remove intertwined or overlapping circles which is called "remove interwined circles". The function can be found in line 420 of experiment.py.

Snippet can be found in next page.

7: Discussion

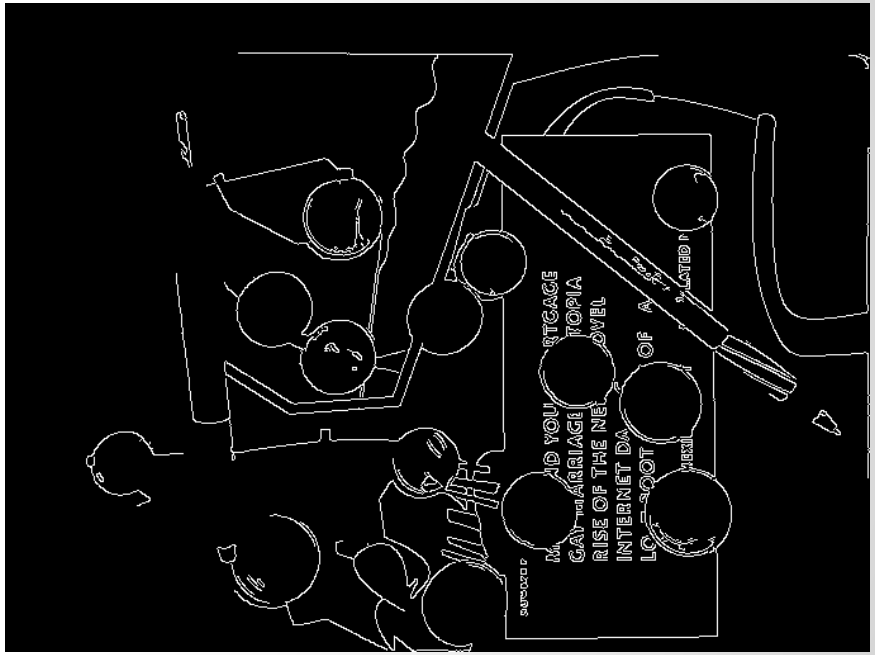
```
def remove_interwined_circles(circles,thresh=50):
    new_circles =[]
    for i in range(len(circles)):
        for j in range(len(circles)):
            if j == 0:
                has_neighbor = False
            if i !=j:
                circle1_x = circles[i][0]
                circle1_y = circles[i][1]
                circle1_r = circles[i][2]

                circle2_x = circles[j][0]
                circle2_y = circles[j][1]
                circle2_r = circles[j][2]

                dist =
euclid(circle1_x,circle1_y,circle2_x,circle2_y)
                if dist<thresh:
                    has_neighbor = True
                    if circle1_r>=circle2_r:

new_circles.append([circle1_x,circle1_y,circle1_r])
        if not has_neighbor:

new_circles.append([circle1_x,circle1_y,circle1_r])
    return new_circles
```



Highlighted coin edges using canny thresholds

8: CHALLENGE PROBLEM

Results of Challenge Problem a or b - ps2-8-[a/b]-1.png

8: Details about the implementation

Provide specific details about how you accomplished your results. You may add more slides if needed.

References:

- http://opencvexamples.blogspot.com/2013/10/line-detection-by-hough-line-transform.html
- https://www.mathworks.com/help/images/ref/houghpeaks.html?refresh=true
- http://me.umn.edu/courses/me5286/vision/Notes/2015/ME5286-Lecture9.pdf
- http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html
- http://www.ltu.se/cms_fs/1.36192!/e0005e_lecture05_hough_transform.dvi.pdf
- https://piazza.com/class/ixpb4h3cvua2gp?cid=206