Computer Vision
Spring 2017
Problem Set # 2

Nikhil Gajendrakumar
nikhil.g@gatech.edu

1a. Edge Image

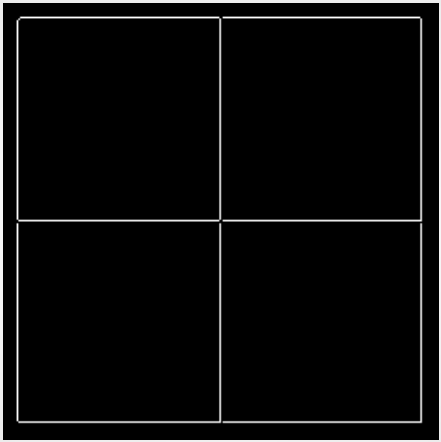


Figure: img_edges - ps2-1-a-1.png

2a. Hough Accumulator Array

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

2b. Accumulator Array w/Peaks

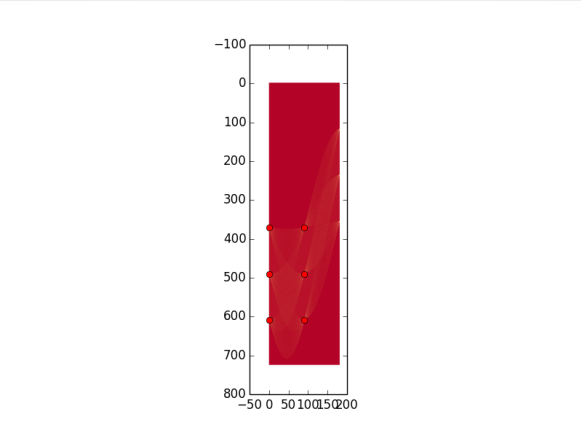


Figure: Accumulator Array with peaks highlighted ps2-2-b-1.png

2c. Image with Lines Drawn

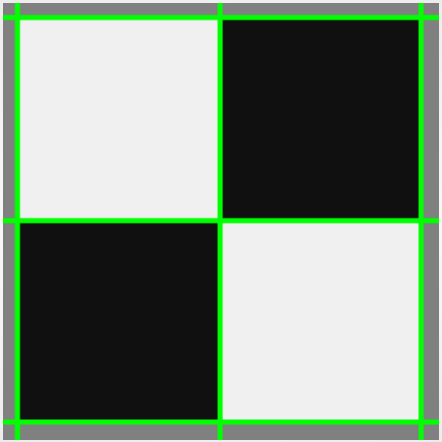


Figure: Original grayscale with lines drawn - ps2-2-c-1.png

3a. Smoothed Intensity Image

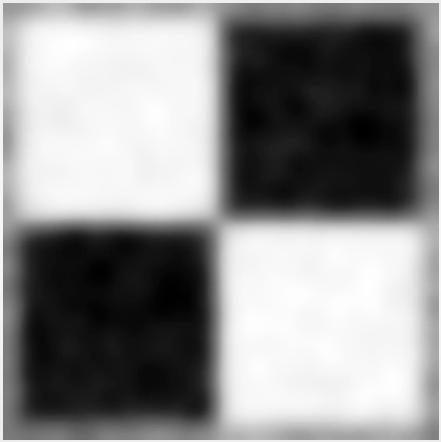
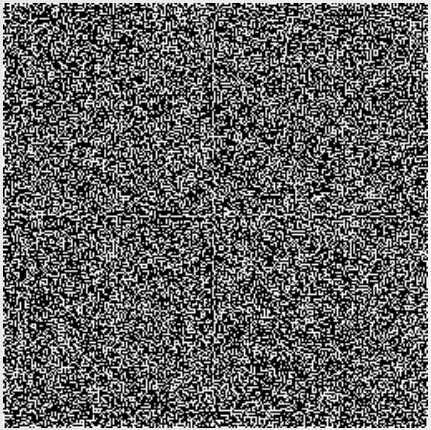
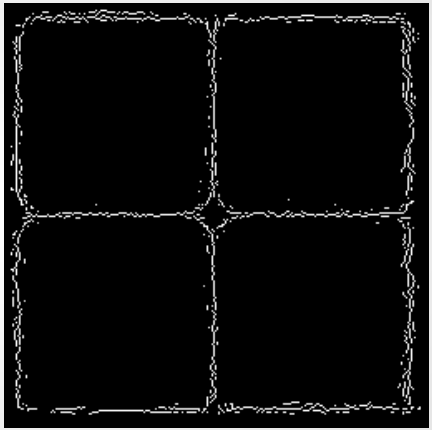


Figure: ps2-3-a-1.png

3b. Edge Image Comparison

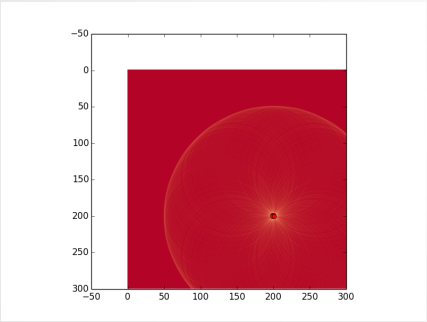


Edge Image (original) - ps2-3-b-1.png

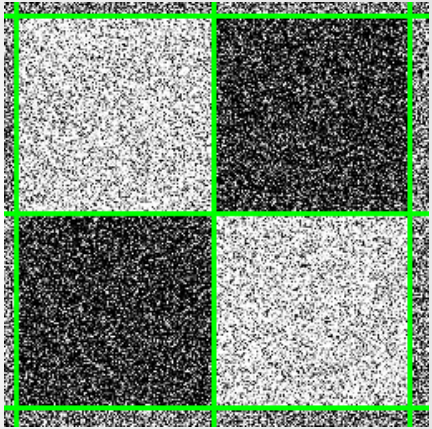


Edge Image (smoothed) - ps2-3-b-2.png

3c. Hough Transform Smoothed

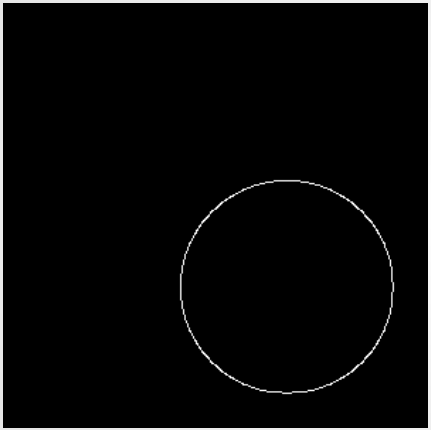


Accumulator Array with peaks highlighted - ps2-3-c-1.png

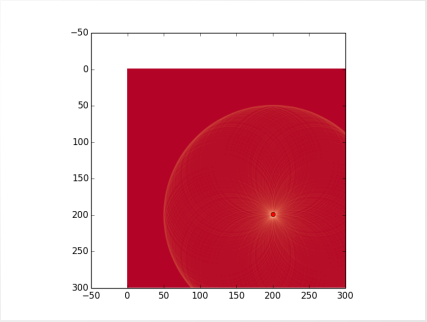


Original intensity image with lines drawn - ps2-3-c-2.png

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

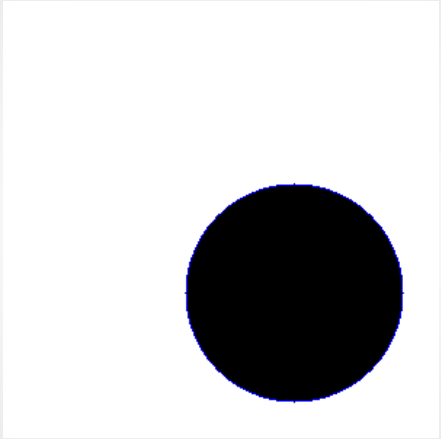
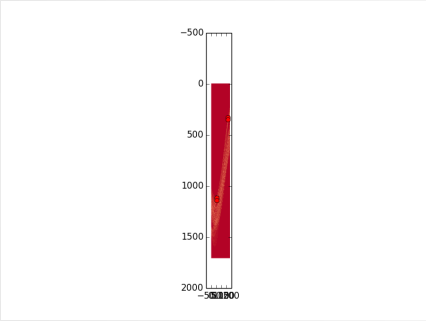
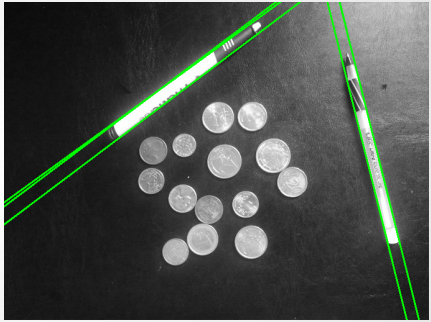


Figure: Original monochrome image with circles drawn in color- ps2-4-b-1.png

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

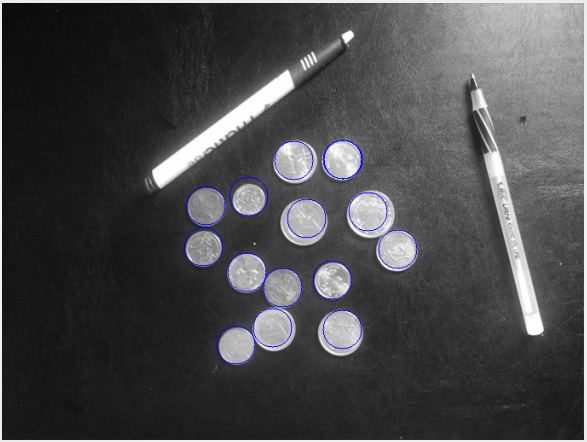


Figure: Original monochrome image with circles drawn - ps2-5-b-1.png

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

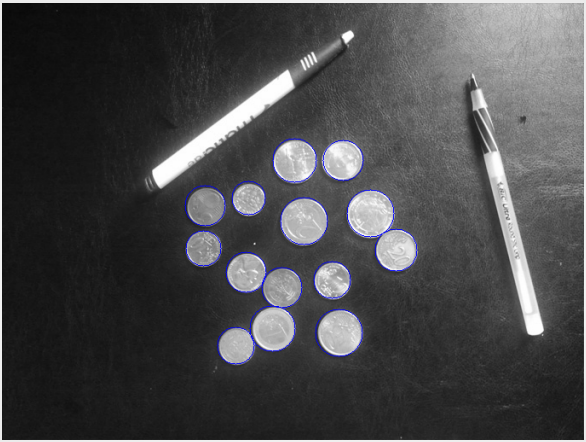


Figure: Original monochrome image with circles drawn - ps2-5-c-1.png

6a: Images with Clutter (pens)

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

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

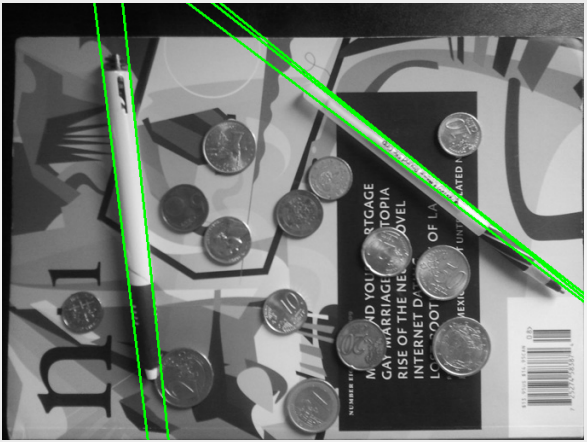


Figure: Smoothed image with lines drawn - ps2-6-b-1.png

6c: Images with Clutter (coins)

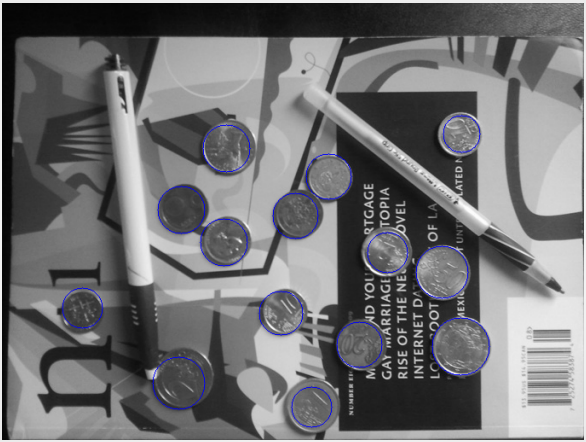


Figure: Smoothed image with circles drawn - ps2-6-c-1.png

7a: 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.)

In order to find both lines and circles, I used Canny edge detector with Threshold1 = 100, Threshold2 = 100. I picked these parameters by Trial-and-error method as they captured majority of the edges. I used Gaussian Kernel of size 5x5 to blur the image. Again, kernel size was chosen by trial-and-error method.
In order to Find Lines,
Accumulator size = d x theta = image diagonal distance * 180 (degrees) . d = diagonal distance because that's the longest possible edge and theta = 180 because image is interpreted as ranging from -90 degrees to +90 degrees.
Threshold = 200. This is again chosen by lot of trial-and-error and fine tuning.
neighborhood size = 4 x 4 because edges in the given image are fairly apart and this provides non-maximal suppression .

7a: Discussion

I had to increase the Gaussian kernel size to 41 x 41 in order to filter lot of noise in ps2-input0-noise.png
I also reduced the Canny Threshold1 = 10, Threshold2 = 40 in order to handle noise
In order to Find Circles,
Accumulator size = same as size of the image (we are told to follow this)
Threshold = 250. This is again chosen by lot of trial-and-error and fine tuning.
neighborhood size = 75 x 75 because circles in the given image are fairly apart and this provides non-maximal suppression
I had to increase the Gaussian kernel size to 41 x 41 in order to filter lot of noise in ps2-input0-noise.png
I also reduced the Canny Threshold1 = 10, Threshold2 = 40 in order to handle noise.

7a: Discussion

I reduced the Hough threshold to 145 in order to capture all the coins in image ps2-input2.png. Since that image has lot of different shapes, votes of circles were distributed among the neighboring bins, reducing the threshold will help in capturing all the coins.

I also reduced neighborhood size to 50×50, again in order handle votes of circles being distributed among the neighboring bins.

7b:. Discussion

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

In the noisy image, we have to first filter/smoothen the image with a bigger Gaussian Kernel. Canny edge detection thresholds will have to be reduced for noise image, and Hough transform neighbor hood size is increased for a noisy image to obviously account for distributed voting to the neighboring bins.

7c: Discussion

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

Yes, point-plus gradient method is much faster than the single-point method because we compute only one gradient in point-plus method instead of looping over all the thetas between 0 to pi

7d: 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 out all the circles in the question 5, with both radius = 23 and interacting over radius between 18 to 30. In both cases I had to reduce the Hough transform threshold to 145 and neighborhood delta to 50 x 50.

7e: Discussion

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?

I found out all the line segments using rho and theta, and considered only line segments with slope between range 0.7 to 1.0 and -4 to -7. This helped me to find the correct line segments.

7f: 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

Yes, I did find one false positive near the top left corner of the image. I got rid of it by filtering the image, before finding edges, using gaussian filtering with kernel size 5 x 5 and reducing the hough transform threshold to 100 and radius ranging from 18 to 35.

```
from matplotlib import pyplot as plt
import numpy as np
import math

img = cv2.imread(PATH_in + "ps2-input1.png", 0)
img1 = cv2.GaussianBlur(img, (5,5), 0)
img1 = cv2.Canny(img1, 100, 200)
img1 = img1 / 255.0

peaks = find_circles(img, img1, range(18, 35, 1), 100,
                                    nhood_delta=(30, 30))
plt.imsave(PATH_out + "ps2-6-c-1.png",
                draw_circles(img, peaks),
                cmap="coolwarm_r")
```