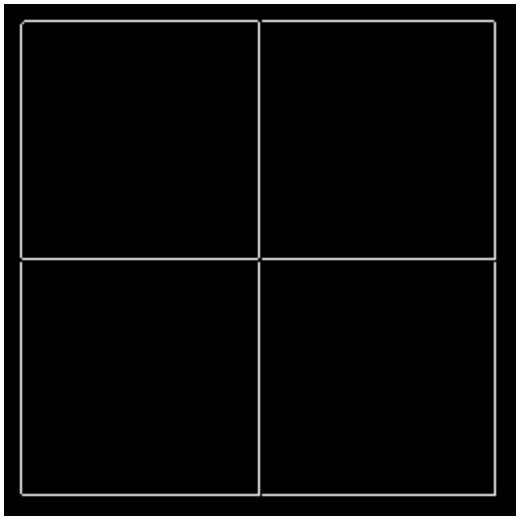
Problem Set 1

Author: Swapnil Ralhan(sralhan3)
GTID: 903049849
02/10/2015

Problem 1

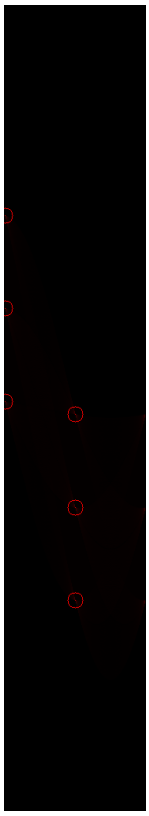


Problem 2

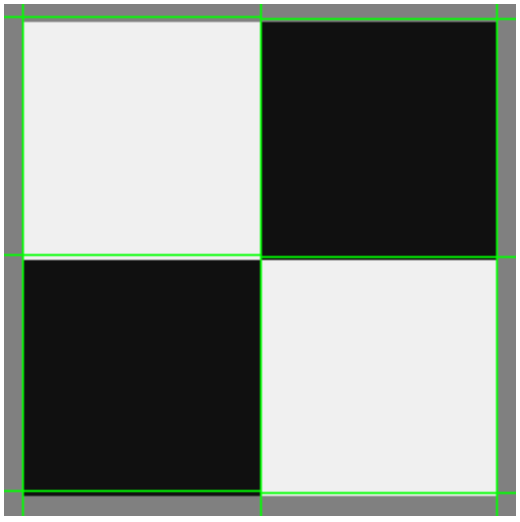
A. Hough Accumulator Array

Points not easily visible due, but peaks highlighted below.

B. Peaks



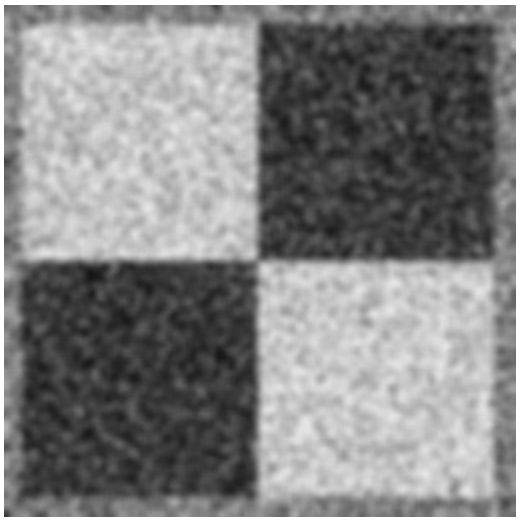
C.



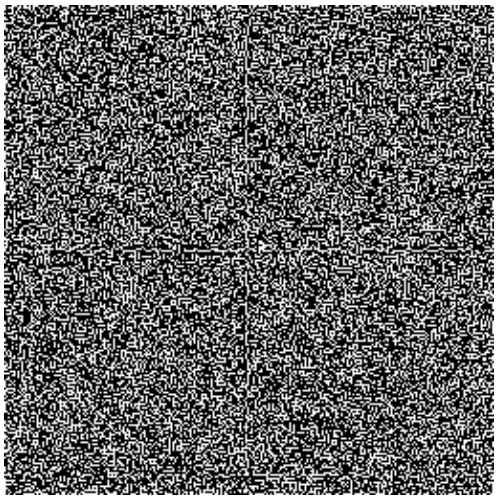
The default values worked well - Rho Resolution - 1.0(bin size is 1 pixel), Threshold = 0.5 * max value of H, Neighborhood size =floor(H.shape / 100.0) * 2 + 1

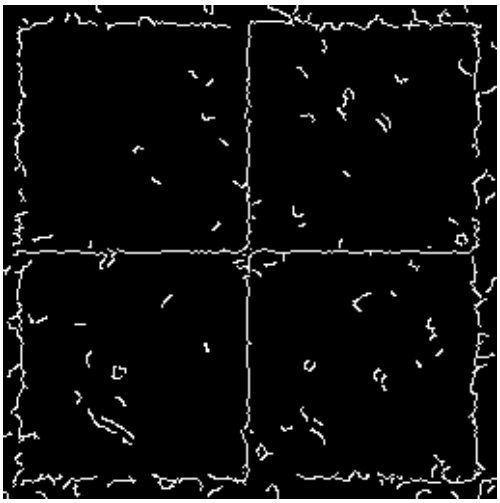
Problem 3

A.



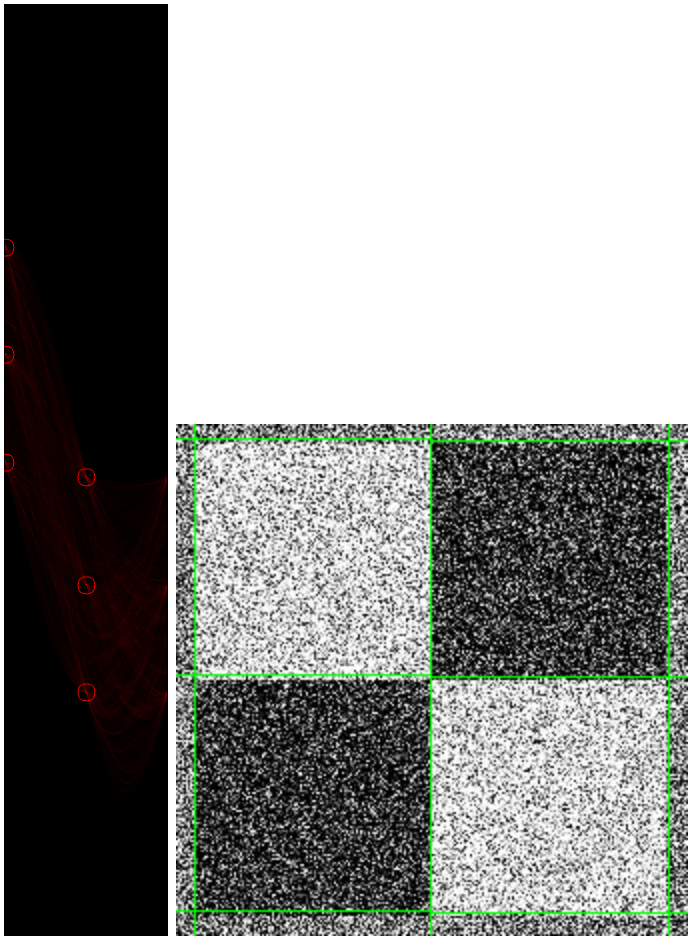
B.





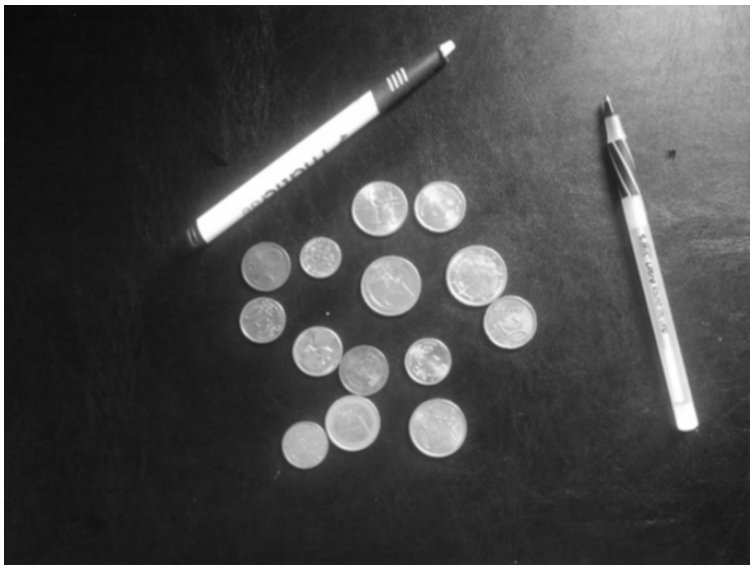
C.

Some modest smoothing was enough to find the edges correctly - a 7x7 kernel with std. dev 2. For the edge finding, I used the Canny operator with thresholds 100 and 150. These were derived with some trial and error to see which values keep the edges, while getting rides of the noise.

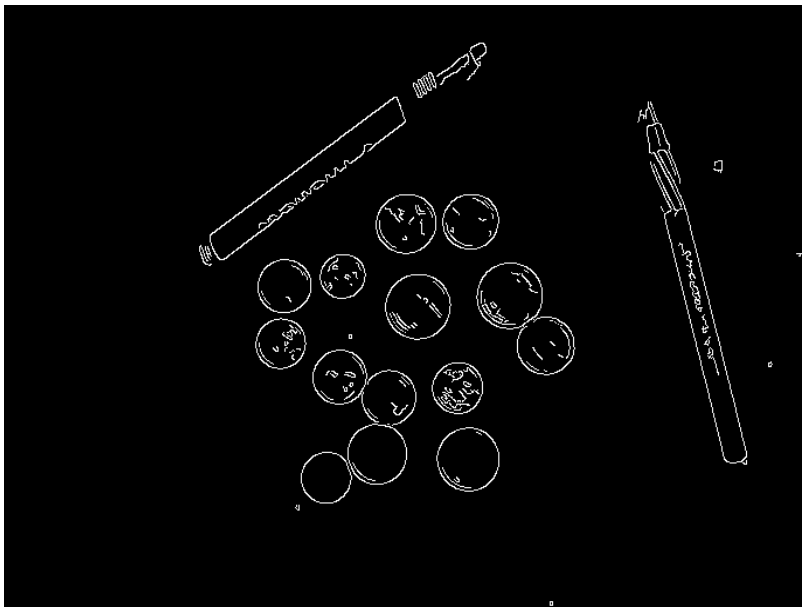


Problem 4

A.

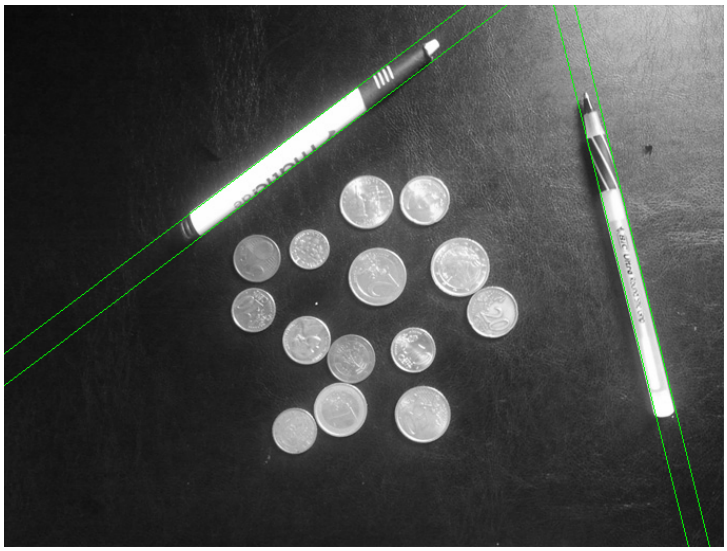


B.



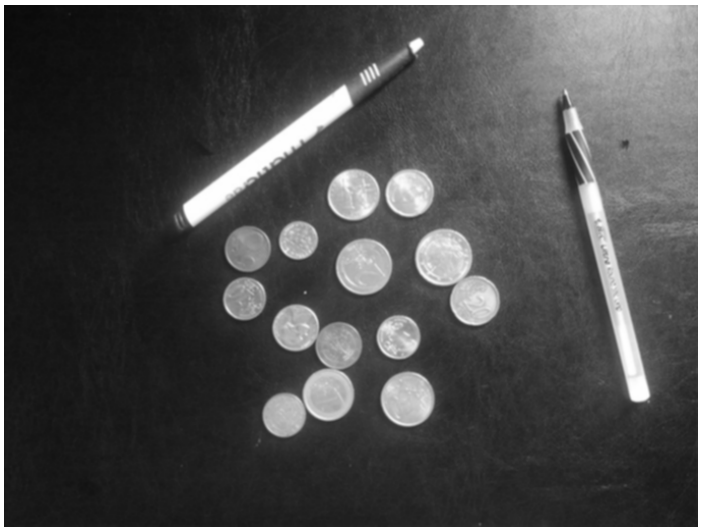
C.

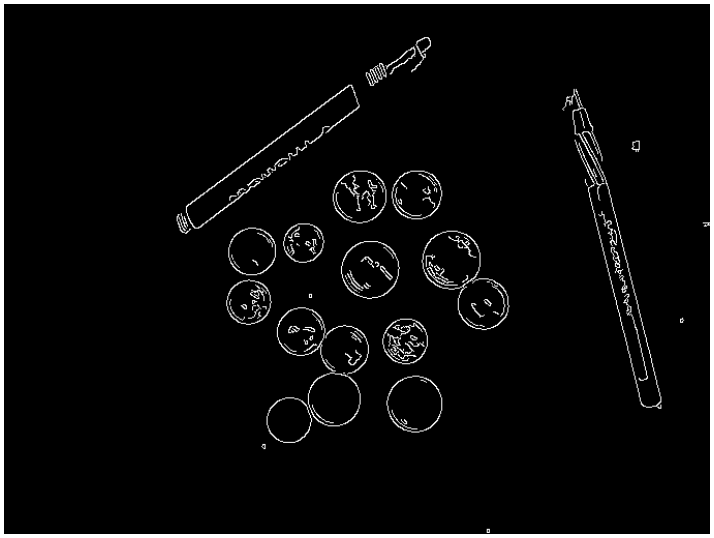
I was able to find the 4 lines representing the edges of the pens. Picking the top 4 lines, with a modified neighborhood size of floor(H.shape / 150.0) * 2 + 1. The default values were finding multiple lines closeby for a single edge of a pen. Thus, we increased the neighborhood size to eliminate closeby lines.

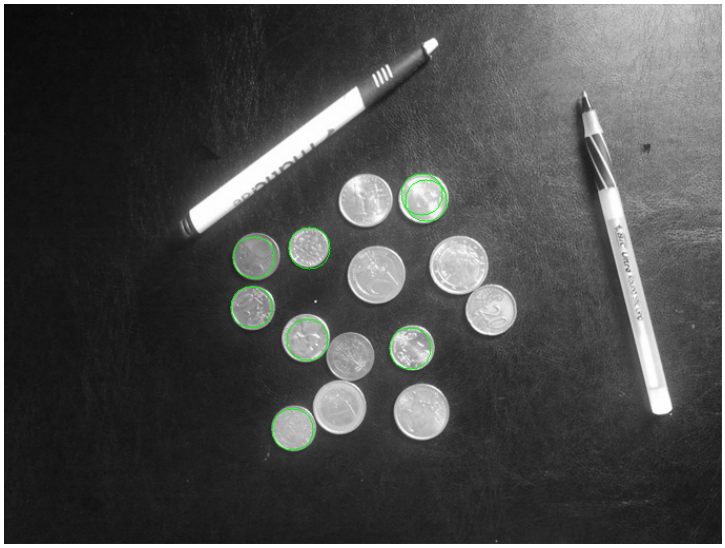


Problem 5

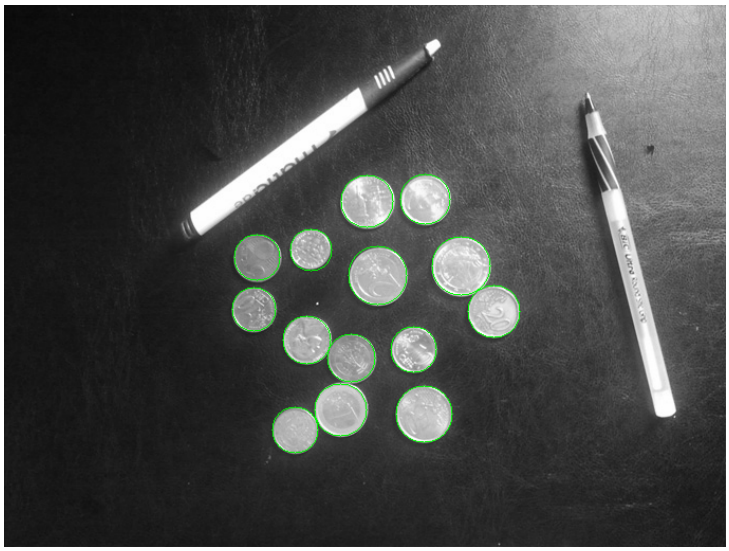
A.







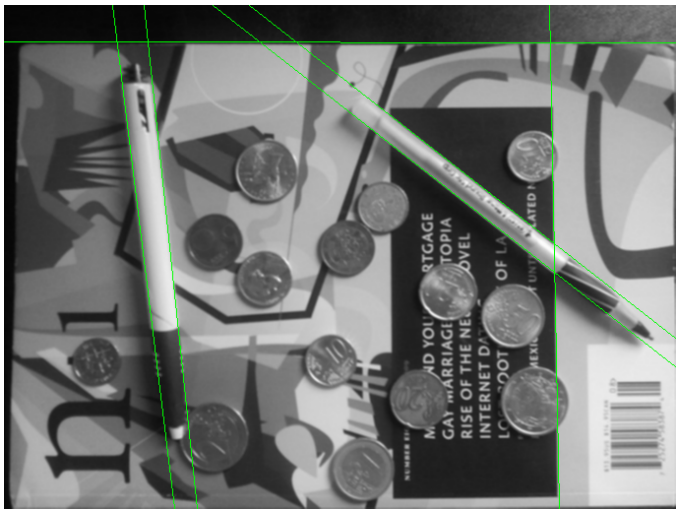
B.



As you can see, we detect all 14 circles. I used the point + gradient method, which made things much simpler as the arrays were much smaller, and the algorithm much faster. I created an extra method that would return all local optima for an accumulator array, put in a priority queue with the value of the accumulator. Then the calling method(find_circles), would put all the values into a common priority queue, and will go through them one at a time, in order of priority(value of the accumulator).
The only tweak I had to make was to avoid circles that were too close together. I used the boundary as minimum of the radius range, but it can be tweaked depending on what level of overlap we'd like to allow.

Problem 6

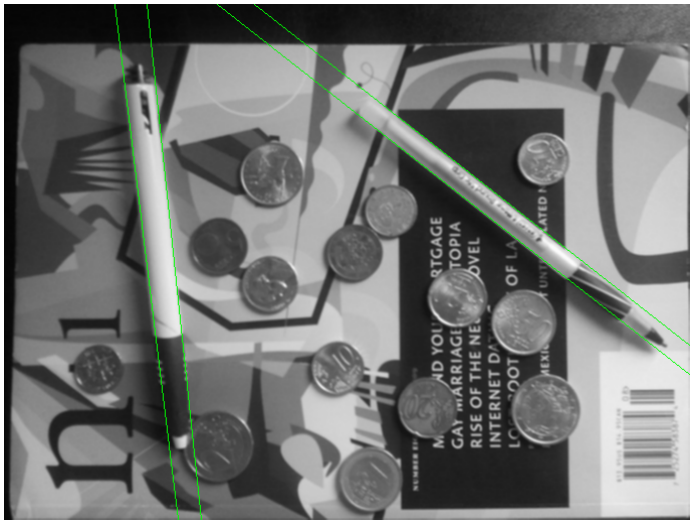
A.



B.

The line finder found lines other than the edges of the pen. For instance it found the edge of the book, or the edge of a section on the book cover. We need to get rid of these false positives.

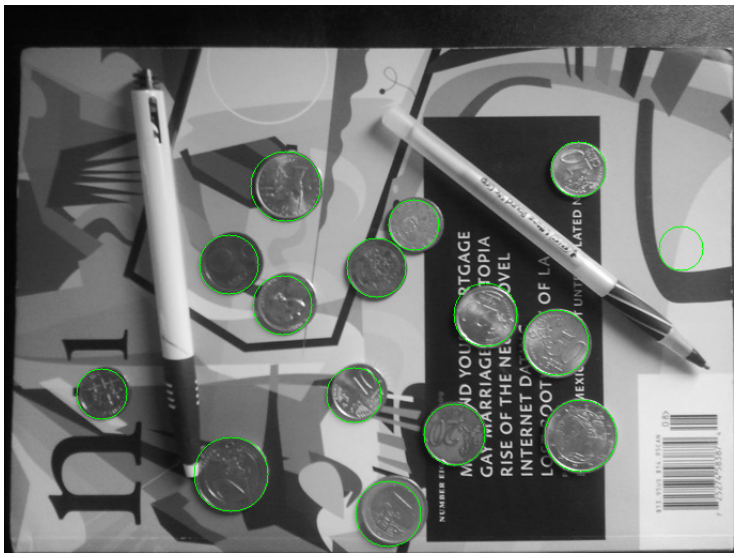
C.



This required not much work. First, I simply got the top 6 lines(better threshold), which got me the lines as shown above. Then I applied a filtering heuristic. I kept pairs of lines that were within 50 pixels of each other, and which were parallel(within 2 degrees). This found us the above 2 lines.

Problem 7

A.

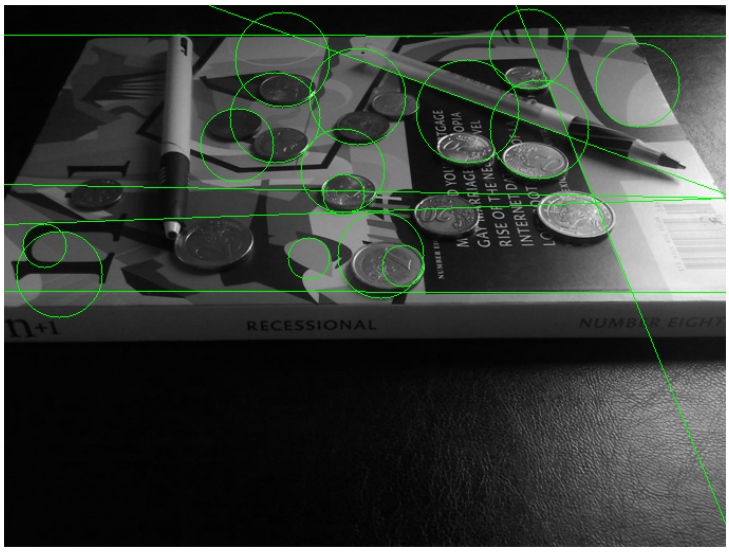


B.

As we can see, there is one false positive, the partial circle formed by the shape on the book. The reason this happens is that the shape on the book is very solid, which forms a strong edge in the edge image. This edge contributes sufficient pixels to the center for our circle recognizer to think that there is a circle there. One thing that can be done is to necessitate edge pixels on opposite(or some definition of sufficient) sides of the circle being formed. For instance, we could have the variation in the points that contribute to this circle be at least π radians.

Problem 8

A.



B.

As we can see, we found some circles and lines, though the accuracy was pretty off. In terms of lines, we found the more prominent lines, such as the edges of the book, an edge of a pen, and the edge formed by the shape on the book. However we also found false positives, such as a number of lines going through the book, as well some false negatives, such as no edges found for the pen.

In terms of circles, we can see that our recognition is far from correct. The main reason is that due to distortion, the circles are no longer spherical, but slightly oblong/elliptical. Thus, a way to fix it, is to use the equation of an ellipse to find circles(doable in $O(N^3)$ time). Alternatively, we can take into account the error, and have each circle vote for both the predicted center, as well as neighboring bins, to take into account that the circle may be deformed in that direction.