Problem Set 2: Edges and Lines

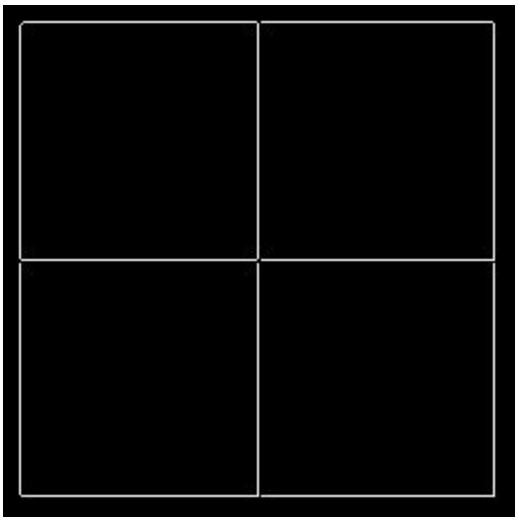
Christopher Owens

gtg819x

901821872

Questions

1. For this question we will use `input/ps2-input0.png`:
    a. Load the input grayscale image (`input/ps2-input0.png`) as `img` and generate an edge image – which is a binary image with white pixels (1) on the edges and black pixels (0) elsewhere.

Output: 

`ps2-1-a-1.png`

2. Implement a Hough Transform method for finding lines. Note that the coordinate system used is as pictured below, with the origin placed at the upper-left pixel of the image and with the Y-axis pointing downwards..

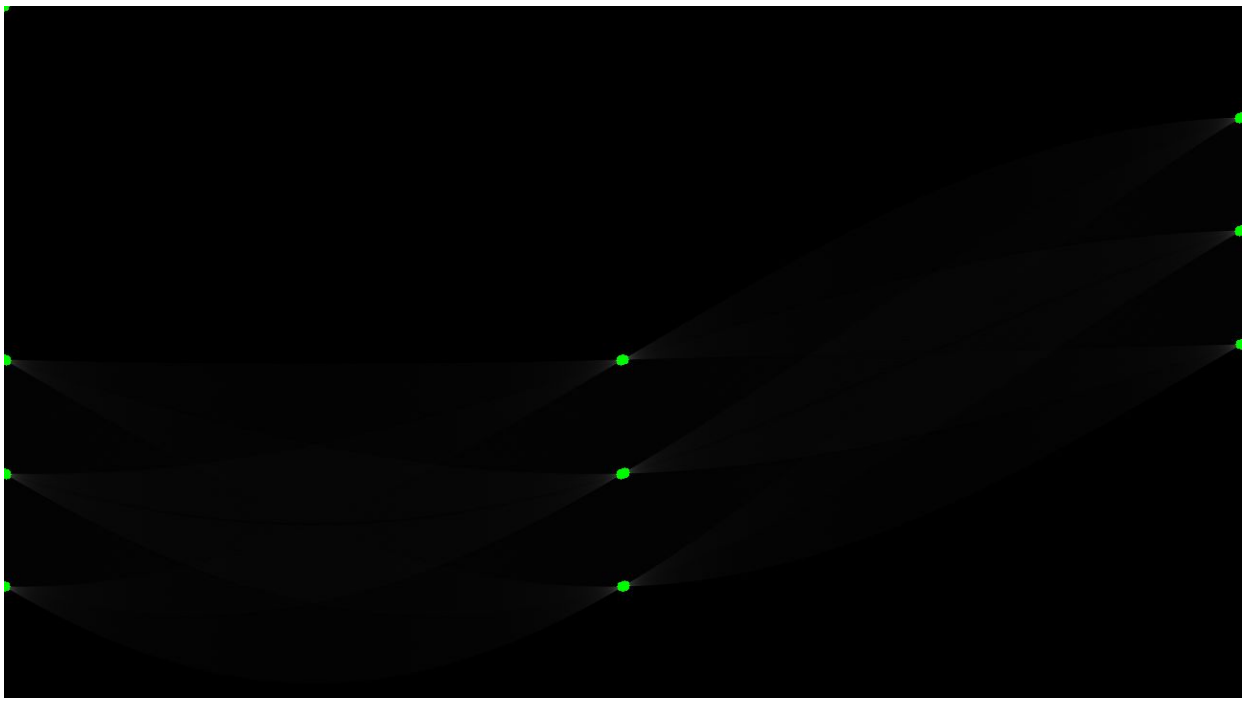
    a. Write a function **`hough_lines_acc()`** that takes a binary edge image and computes the Hough Transform for lines, producing an accumulator array.
    Note that your function should have two optional parameters: `rho_res` (ρ resolution in pixels) and `theta_res` (θ resolution in radians), and your function should return three values: the hough accumulator array H, rho (ρ) values that correspond to rows of H, and theta (θ) values that correspond to columns of H.

Output:

ps2-2-a-1.png

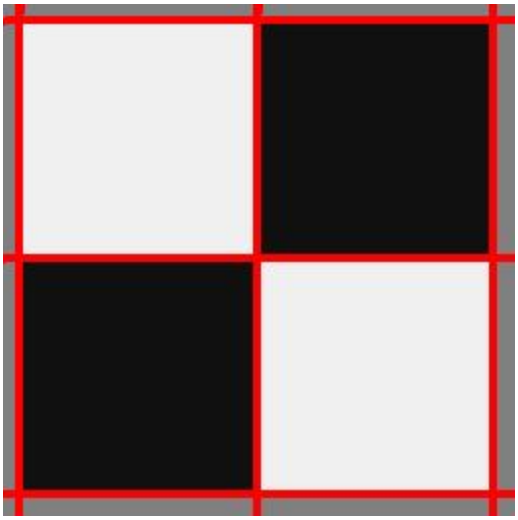
b. Write a function **hough_peaks**( ) that finds indices of the accumulator array (here, line parameters) that correspond to local maxima. It should take an additional parameter Q (integer >= 1) indicating the (maximum) number of peaks to find, and return indices for up to Q peaks.

Output:



ps2-2-b-1.png

c. Write a function **`hough_lines_draw()`** to draw color lines that correspond to peaks found in the accumulator array. This means you need to look up rho, theta values using the peak indices, and then convert them (back) to line parameters in cartesian coordinates (you can then use regular line-drawing functions).

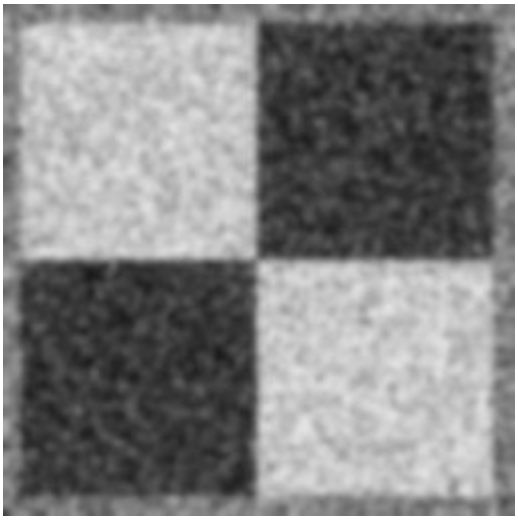
Output: 

ps2-2-c-1.png

d. What parameters did you use for finding lines in this image?
Output: Text response describing your accumulator bin sizes, threshold, and neighborhood size parameters for finding peaks, and why/how you picked those.
   i. theta_res=pi/45 - I made this a little bigger than the default so it would run more quickly. It still gives a very fine bin size
   ii. rho = 1 - it worked with the default, so I didn't change it
   iii. I looked for the top 30 peaks. I upped the number of peaks until I found all the lines. Since this is a simple image, I figured it wouldn't find any stray lines and bumped it up by 10 each time.

3. Now we're going to add some noise.

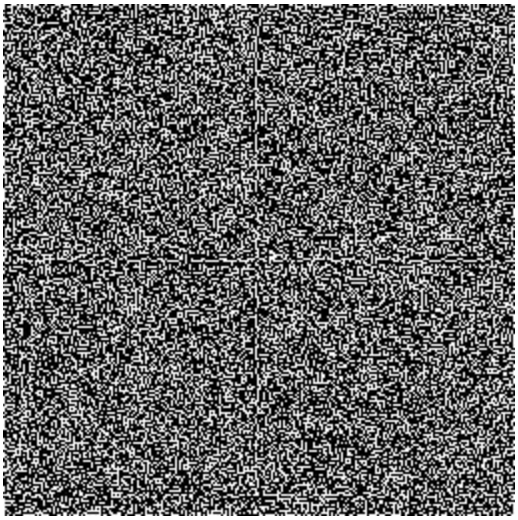
a. Use `ps2-input0-noise.png` - same image as before, but with noise. Compute a modestly smoothed version of this image by using a Gaussian filter. Make σ at least a few pixels big.

Output: 

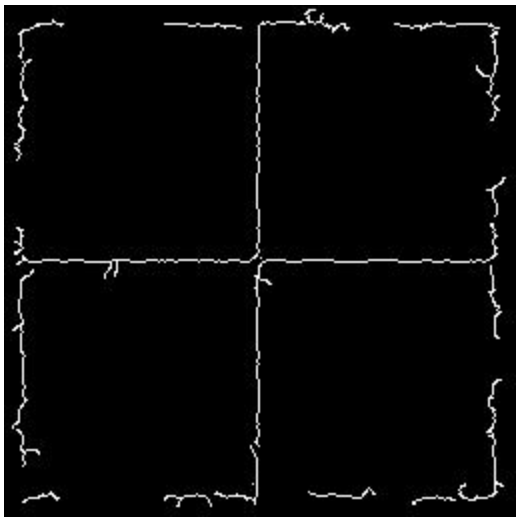
Smoothed image: `ps2-3-a-1.png`

b. Using an edge operator of your choosing, create a binary edge image for both the original image (`ps2-input0-noise.png`) and the smoothed version above.

Output:

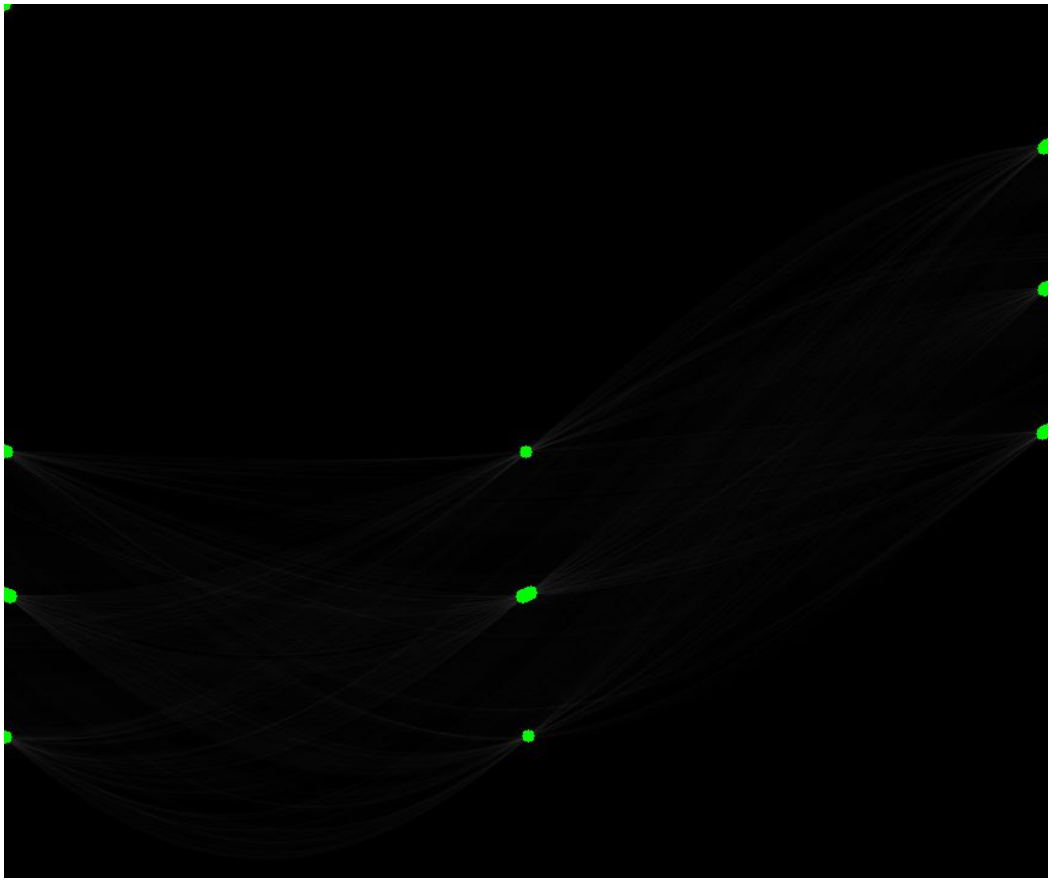


`ps2-3-b-1.png` (from original)



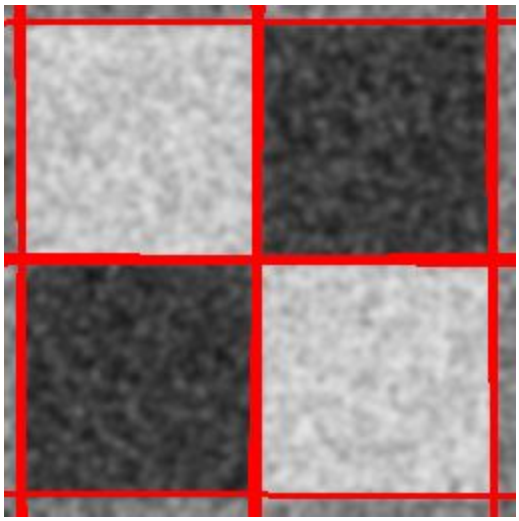
ps2-3-b-2.png (from smoothed)

c. Now apply your Hough method to the smoothed version of the edge image. Your goal is to adjust the filtering, edge finding, and Hough algorithms to find the lines as best you can in this test case.
Output:
- Hough accumulator array image with peaks highlighted:



ps2-3-c-1.png

- Intensity image (original one with the noise) with lines drawn on them:



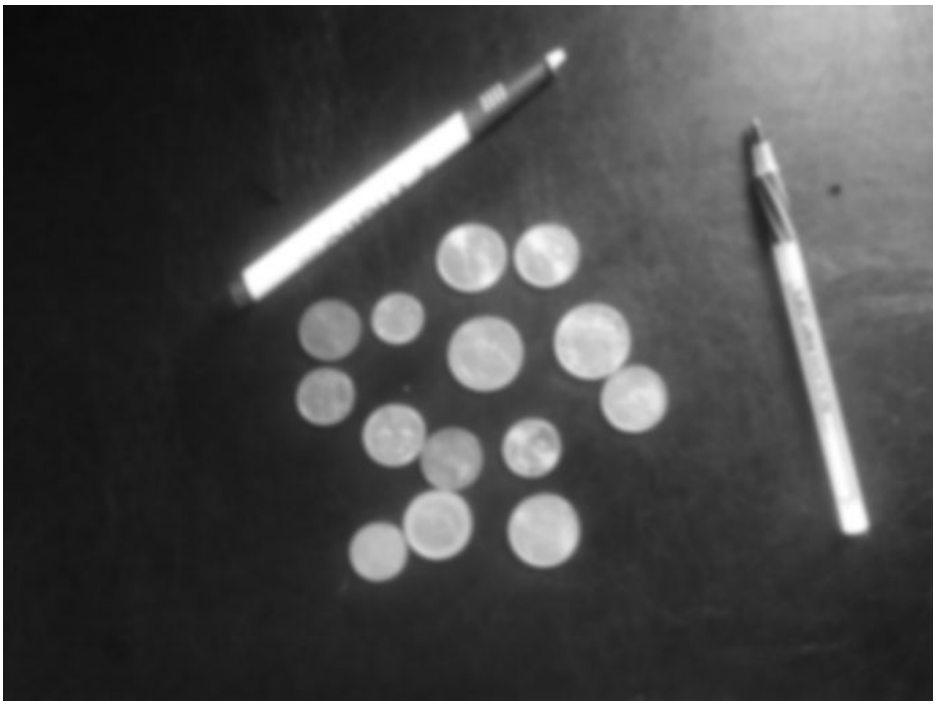
ps2-3-c-2.png

- Text response: Describe what you had to do to get the best result you could.

This one was pretty straightforward. I applied a 15x15 gaussian kernel with a sigma of 2 to blur most of the noise. I had to bump the number of peaks up to get all the lines.

4. For this question use: `ps2-input1.png`
   a. This image has objects in it whose boundaries are circles (coins) or lines (pens). For this question, you are still focused on finding lines. Load/create a monochrome version of the image (you can pick a single color channel, or use a built-in color-to-grayscale conversion function), and compute a modestly-smoothed version of this image by using a Gaussian filter. Make σ at least a few pixels big.

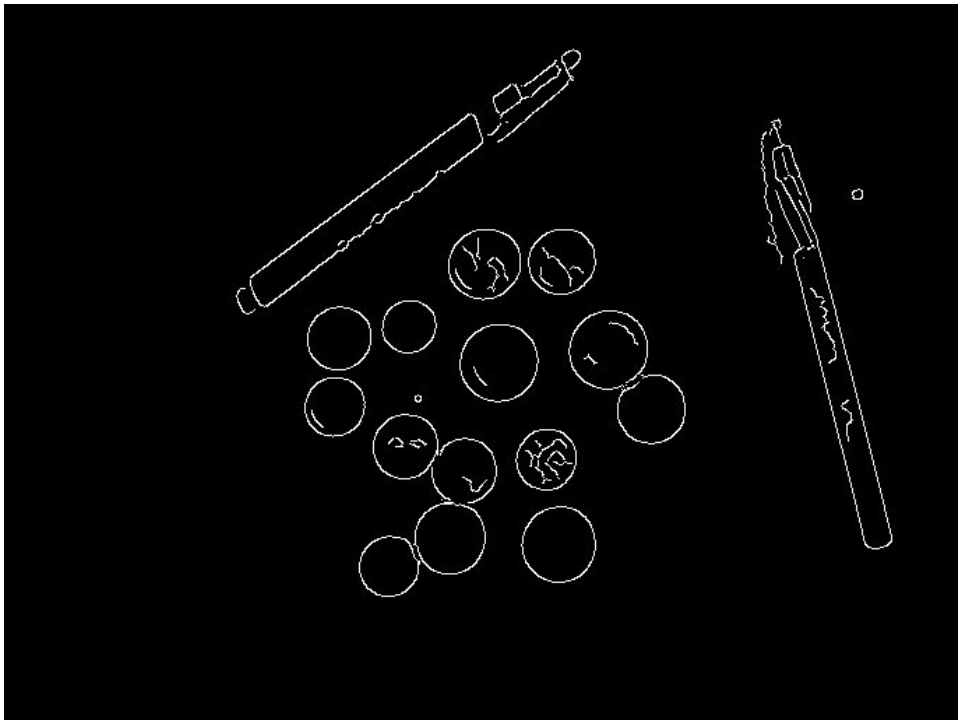
Output: Smoothed monochrome image:



ps2-4-a-1.png

b. Create an edge image for the smoothed version above.

Output: Edge image:



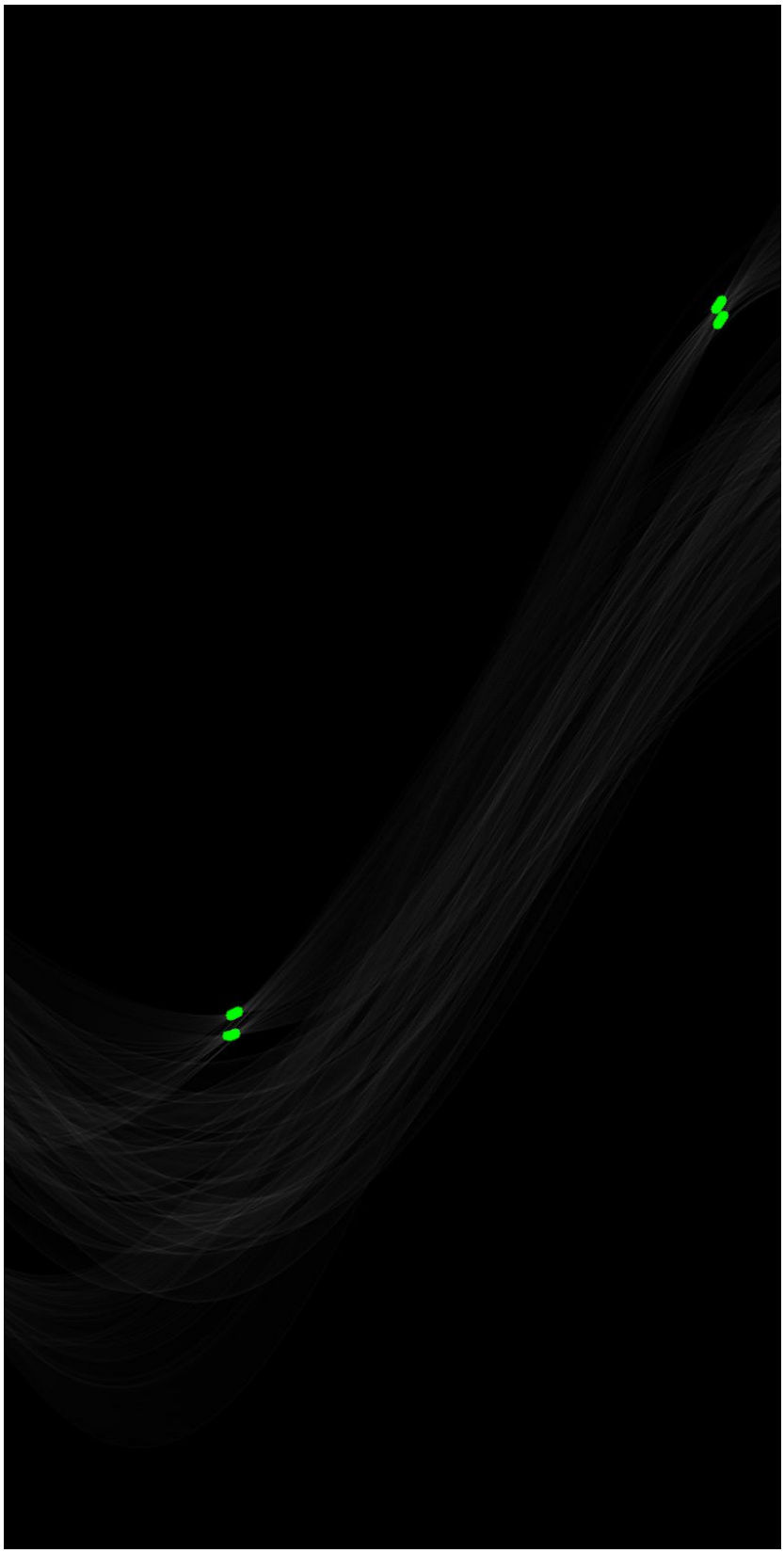
ps2-4-b-1.png

c. Apply your Hough algorithm to the edge image to find lines along the pens. Draw the lines in color on the original monochrome (i.e., not edge) image. The lines can be drawn to extend to the edges

of the original monochrome image.

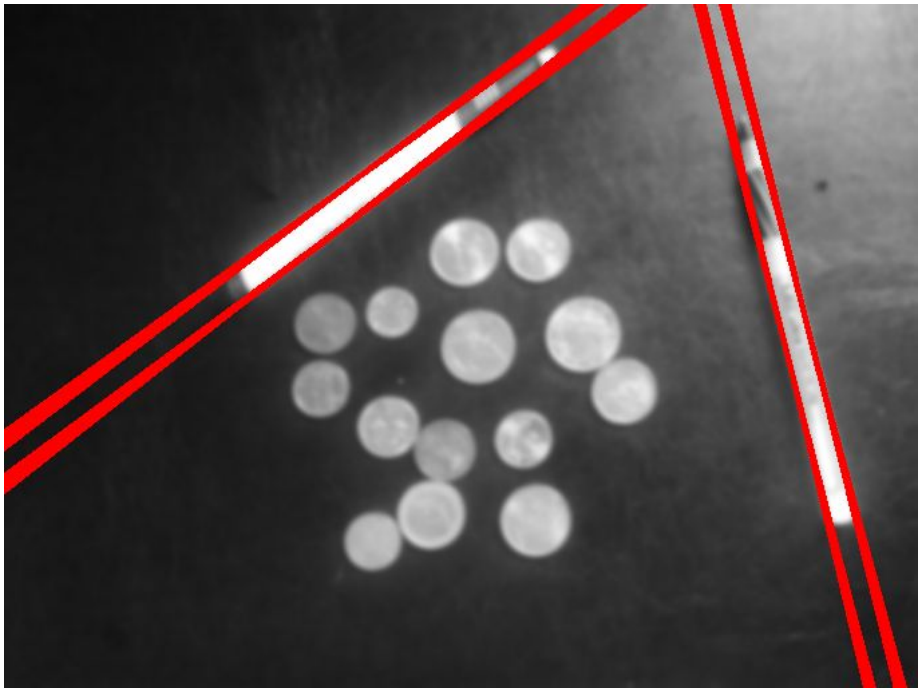
Output:

- Hough accumulator array image with peaks highlighted:



ps2-4-c-1.png

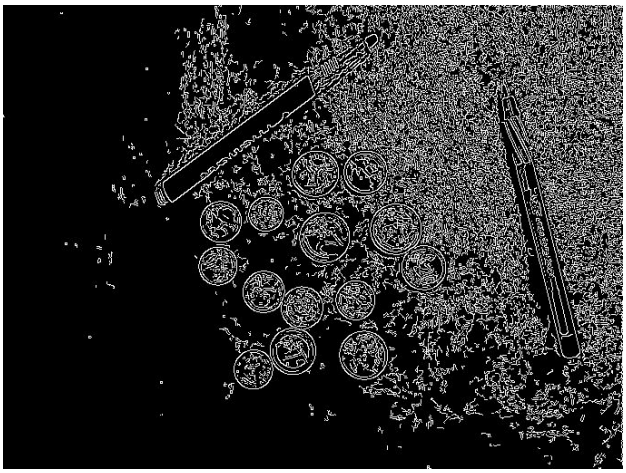
- Original monochrome image with lines drawn on it:



`ps2-4-c-2.png`

Text response: Describe what you had to do to get the best result you could.

i. The main thing I did was increase how much I blurred the image to get rid of the 'edges' created by the light reflecting on the table. This was my first attempt at edges:

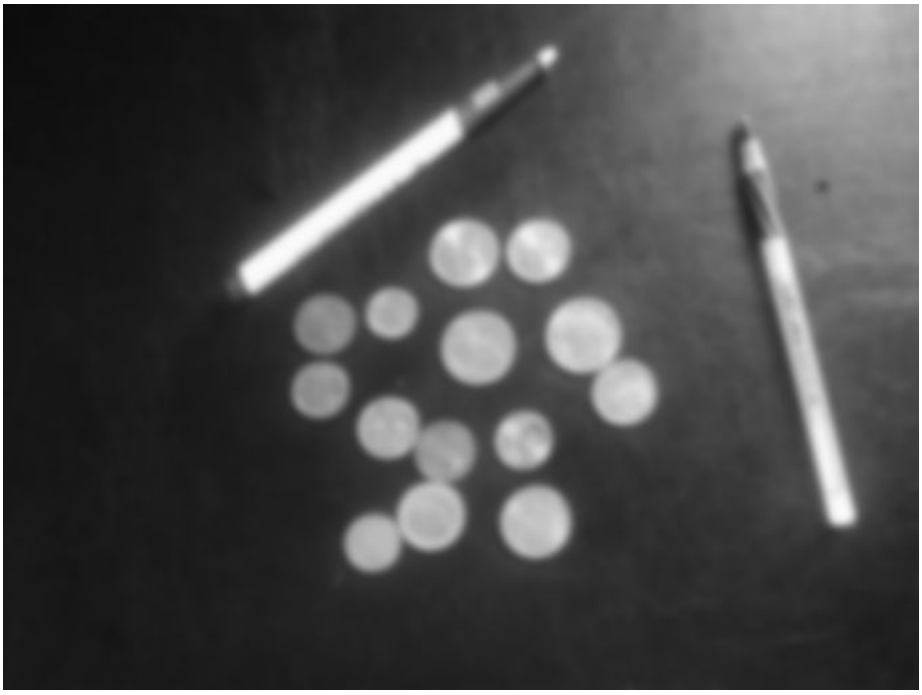


5. Now write a circle finding version of the Hough transform. You can implement either the single point method or the point plus gradient method.

   a. Implement **`hough_circles_acc()`** to compute the accumulator array for a given radius. Using the same original image (monochrome) as above (`ps2-input1.png`), smooth it, find the edges (or directly use edge image from 4-b above), and try calling your function with radius = 20:

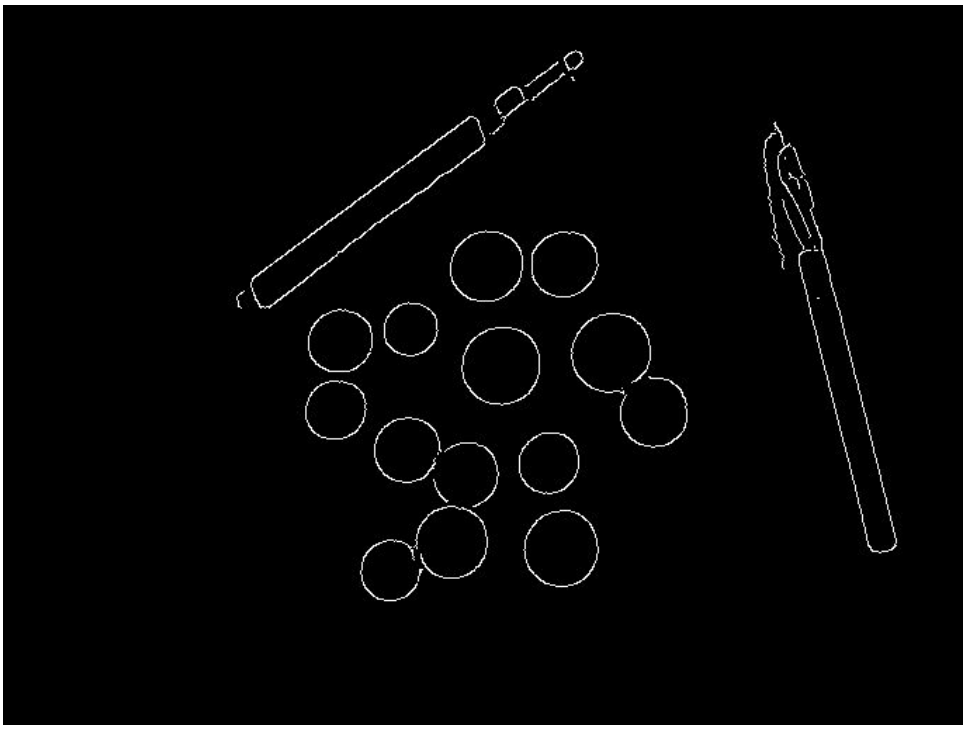
Output:

- Smoothed image:



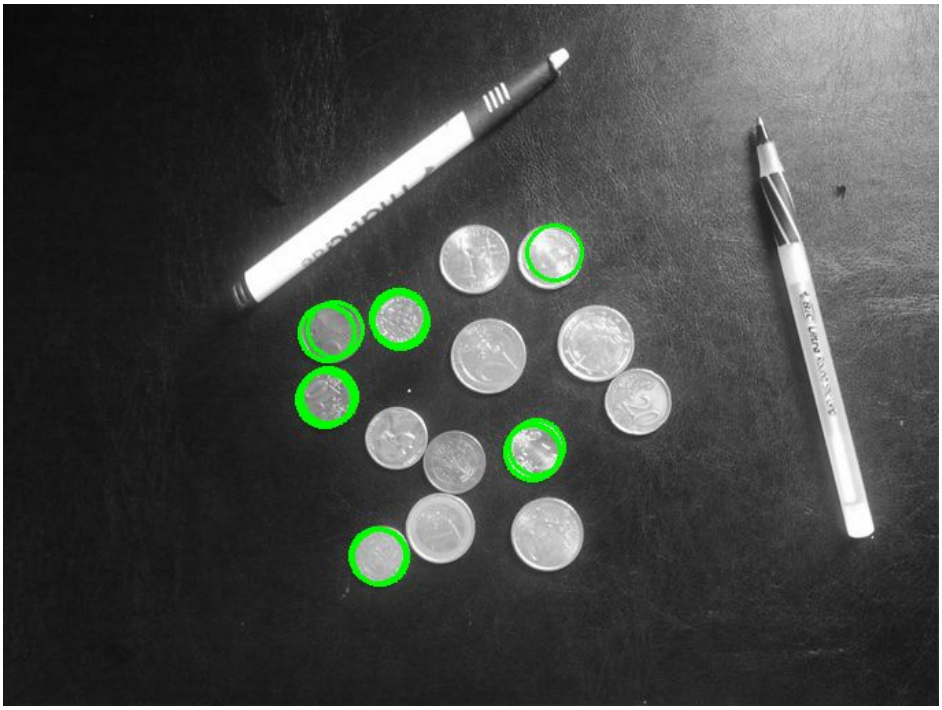
`ps2-5-a-1.png` (this may be identical to `ps2-4-a-1.png`)

- Edge image:



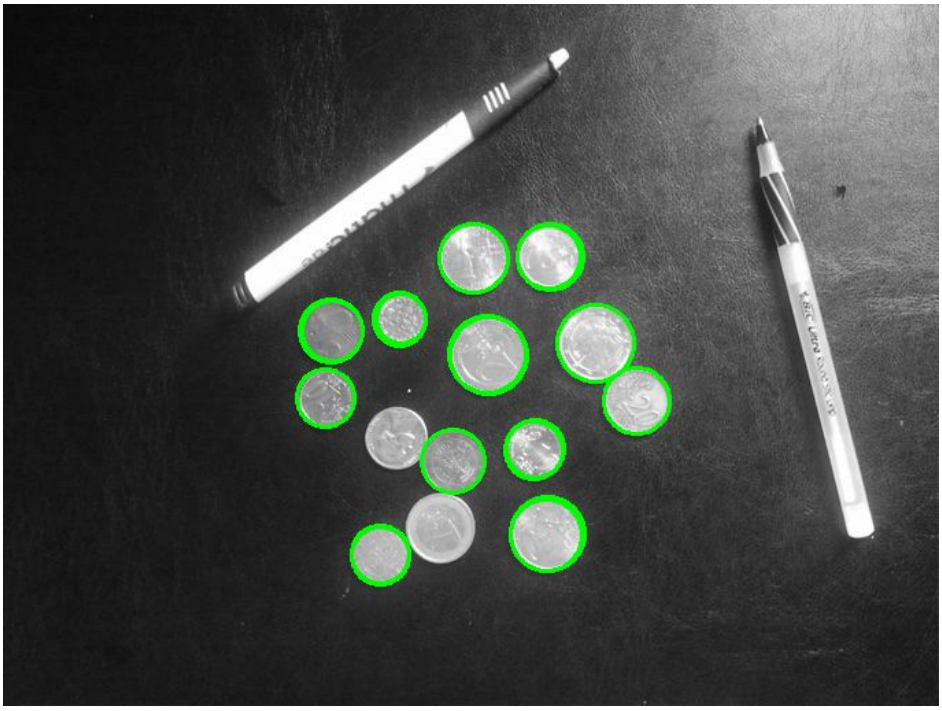
`ps2-5-a-2.png` (this may be identical to `ps2-4-b-1.png`)

- Original monochrome image with the circles drawn in color:



ps2-5-a-3.png

b. Implement a function **`find_circles()`** that combines the above two steps, searching for circles within a given (inclusive) radius range, and returning circle centers along with their radii
Output:
- Original monochrome image with the circles drawn in color:

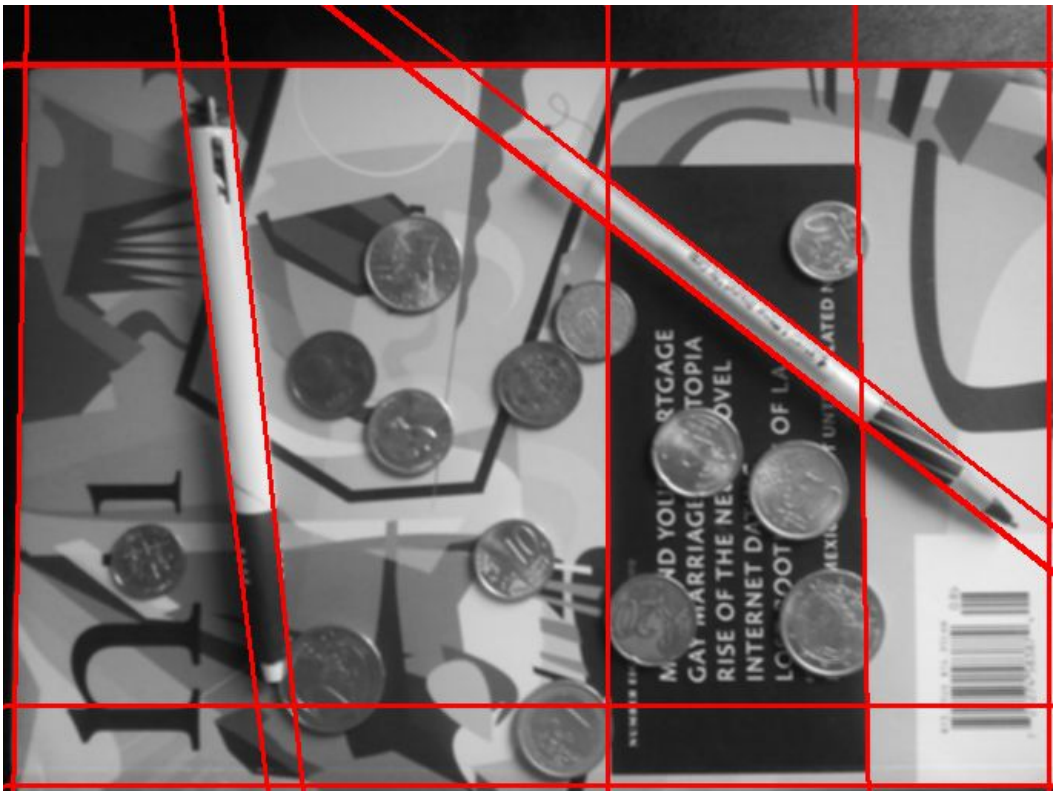


ps2-5-b-1.png

- Text response: Describe what you had to do to find circles.

I mostly played with the radius range. I also had to get rid of as much noise as possible in the edge image. The faces on the coins caused many circles to vote for the center to be slightly askew of where it should be.

6. More realistic images. Now that you have Hough methods working, we're going to try them on images that have *clutter*--visual elements that are not part of the objects to be detected. Use: `ps2-input2.png`
   a. Apply your line finder. Use whichever smoothing filter and edge detector that seems to work best for finding all pen edges. Don't worry (until 6b) about whether you are finding other lines as well.
   Output: Smoothed image you used with the Hough lines drawn on them:

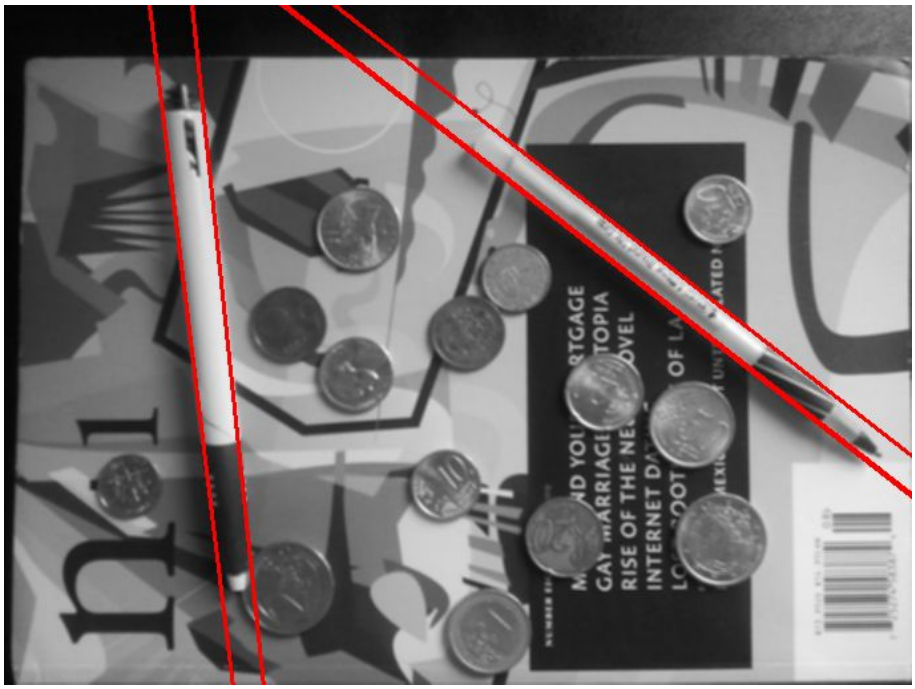


`ps2-6-a-1.png`

   b. Likely, the last step found lines that are not the boundaries of the pens. What are the problems present?
   Output: Text response
      i. The biggest problem I had was that the left pen is not the strongest edge . There are also many long lines, especially the black text area.

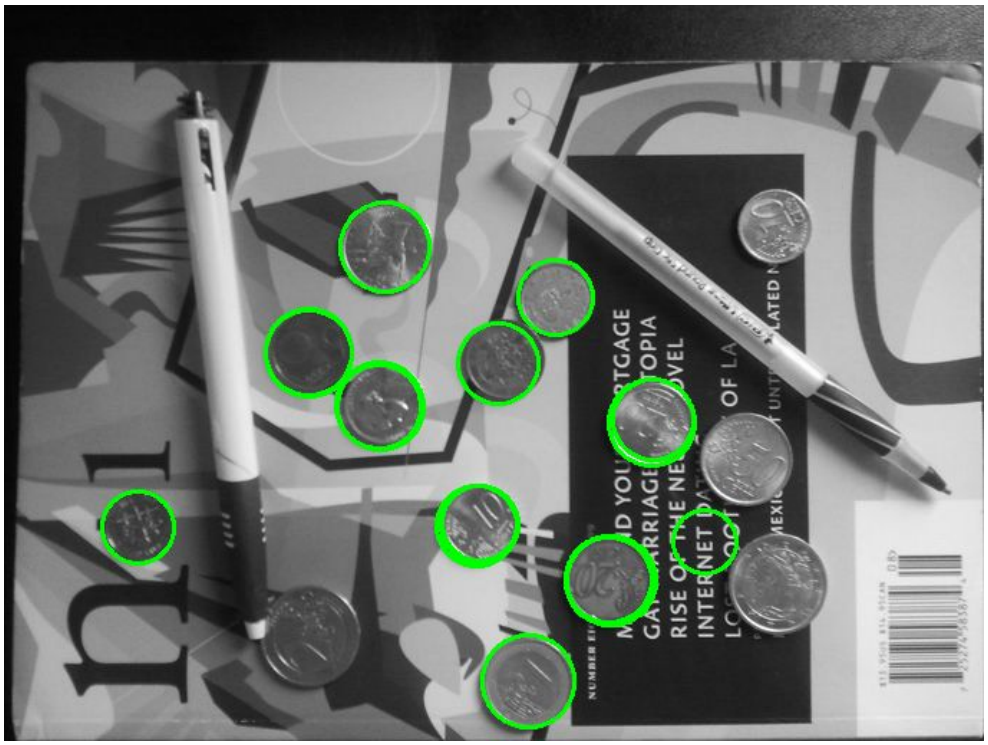
   c. Attempt to find only the lines that are the *boundaries* of the pen. Three operations you need to try are: better thresholding in finding the lines (look for stronger edges); checking the minimum length of the line; and looking for nearby parallel lines.

Output: Smoothed image with new Hough lines drawn:



ps2-6-c-1.png

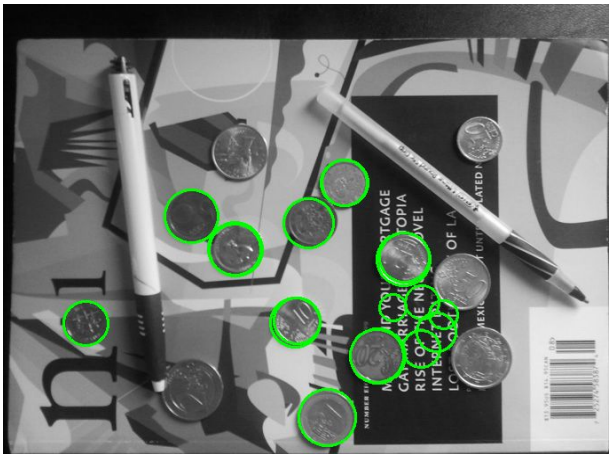
7. Finding circles on the same clutter image (`ps2-input2.png`).
    a. Apply your circle finder. Use a smoothing filter that you think seems to work best in terms of finding all the coins.
    Output: the smoothed image you used with the circles drawn on them:



ps2-7-a-1.png

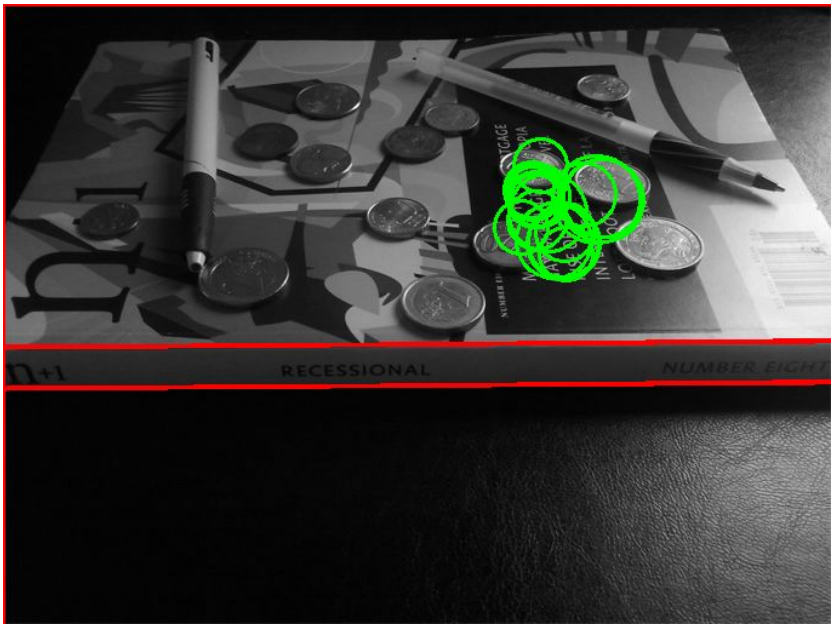
b. Are there any false positives? How would/did you get rid of them?
Output: Text response (if you did these steps, mention where they are in the code by file, line no., and also include brief snippets)

I got many false positives on my first go around:



Most of these were due to the text producing so many edges they acted like circles in the voting. I got rid of most of them by adjusting the radius range from 15-45 to 22-50. I also tried to blur out the text much like I did for the noise created by the reflection on the table, but the text was too defined for this to work without getting rid of most other edges.

8. Sensitivity to distortion. There is a distorted version of the scene at `ps2-input3.png`.
   a. Apply the line and circle finders to the distorted image. Can you find lines? Circles?
   Output: Monochrome image with lines and circles (if any) found:



`ps2-8-a-1.png`

   b. What might you do to fix the circle problem?
   Output: Text response describing what you might try

i. I would attempt to find ellipses instead of circles. This seems much harder/time consuming than finding circles because an ellipse cannot be defined using a center point and a radius. It can however be defined with two points and the sum of the distance from each of those points (distance from point a to edge + distance from point b to same point on the edge must be the same for all points on the edge). I believe this could be transformed into hough space for the voting and then back out for identification. It will probably be much more complicated, I believe we would need another dimension, so optimization would be key.

c. EXTRA CREDIT: Try to fix the circle problem (**THIS IS HARD**).
Output:
- Image that is the best shot at fixing the circle problem, with circles found: `ps2-8-c-1.png`
- Text response describing what tried and what worked best (with snippets).