In [3]:

```
import cv2
import numpy as np
from math import sin, cos, tan, pi
```

In [4]:

```
img0 = cv2.imread('ps1-input0.png', cv2.IMREAD_COLOR)
```

1 In the Problem Set directory there is a Data director with a few images. For this question use the first one ps1-input0.png which looks like this: This is a test image for which the answer should be clear, where the “object” boundaries are only lines.

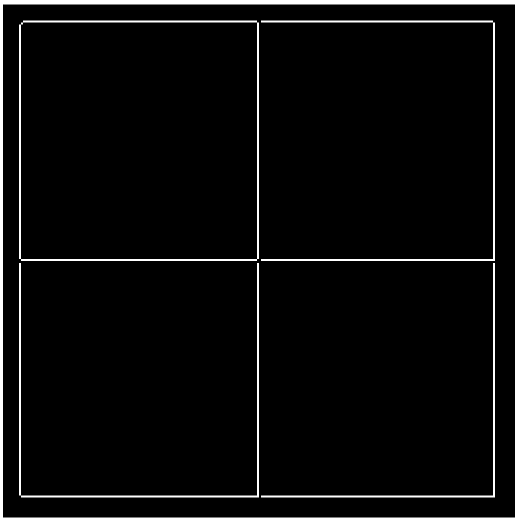
```
a. Do “doc edge” in Matlab and read about edge operators. Using one of you
r choosing – for this image it probably won’t matter much – create an edge
image which is a binary image with white pixels on the edges and black pix
els elsewhere. If your edge operator uses parameters (like ‘canny’) play w
ith those until you get the edges you would expect to see.
Output: the edge image
```

In [5]:

```
img0_edge = cv2.Canny(img0,100,200)
cv2.imwrite('ps1-1-a.png', img0_edge)
```

Out[5]:

```
True
```



2 Write a Hough method for finding lines. Remember to worry about d being negative if θ goes from 0 to π .

```
Apply it to the edge image. Draw the lines in color on the monochrome inte
nsity (not edge) image. The lines can extend to the edge of the images (ak
a infinite lines).
You should see an image that looks like this:
You might get lines at the boundary of the image too depending upon the ed
ge operator you selected (but those really shouldn't be there).
Output: Hough accumulator array image with peaks circled or somehow labele
d.
Output: intensity image with lines drawn on them
Output: written response describing your accumulator bin sizes and why/how
you picked those.
```

In [20]:

```
def hough_line(img, edge_img=None, d_size=250, theta_size=250, peak_prop=0.9, mi
n_vote=100):

    if edge_img is not None:
        edge = edge_img
    else:
        edge = cv2.Canny(img, 100, 200)

    if img is not None:
        img_out = img.copy()

    H = np.zeros((d_size, theta_size))
    theta = np.deg2rad(np.arange(0.0, 180.0, 180.0/theta_size))

    d_max = int(np.hypot(len(img), len(img[0])))
    d_max = round(d_max, 2)
    d_step = d_max / d_size * 2

    distance = np.arange(0-d_max, d_max, d_step)
    true_max_d = 0.0
    true_min_d = 200.0
    true_max_d_index = 0
    true_min_d_index = 100

    y_size, x_size = edge.shape

    for i in range(x_size):
        for j in range(y_size):
            if edge[j][i] >= 200:
                for k in range(len(theta)):
                    ang = theta[k]
                    d = i * cos(ang) + j * sin(ang)
                    d_index = int(d/d_step + 0.5) + int(d_size/2)
                    H[d_index][k] += 1.0
                    #H[d_index - 1 if d_index > 0 else 0][k] += 0.1
                    #H[d_index + 1 if d_index < d_size - 1 else d_size - 1] +=
 0.1

    i, j = H.shape
    m = np.max(H)
    for p in range(i):
        for q in range(j):
            if H[p][q] >= m * peak_prop and H[p][q] > min_vote:
                d = (p - d_size/2) * d_step
                t = theta[q]

                if abs(t - pi/2) > 0.01:
                    if t > 0.01:
                        p1_x = int(d / cos(t) + 0.5)
                        p2_x = int( (d - y_size * sin(t)) / cos(t) + 0.5)
                        p1_y = 0
                        p2_y = y_size
                    else:
                        p1_x = int(d)
                        p2_x = int(d)
                        p1_y = 0
                        p2_y = y_size

                else:
```

```
                    p1_x = 0
                    p2_x = x_size -1
                    p1_y = int(d)
                    p2_y = int(d)

                img_out = cv2.line(img_out, (p1_x, p1_y), (p2_x, p2_y), (100,200
,0), 1)
    if img is not None:
        return img_out, H
    else:
        return H
```

In [21]:

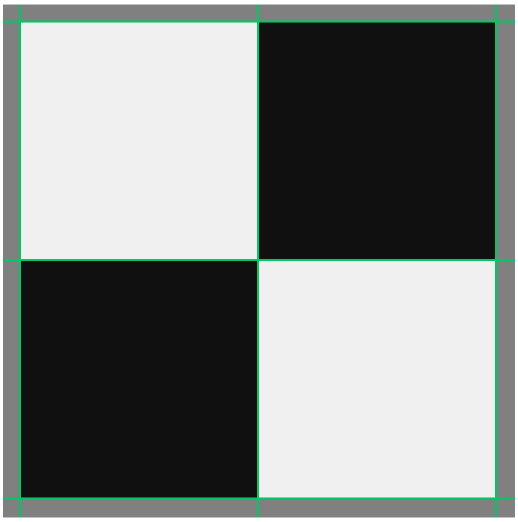
```
img0_hough, h = hough_line(img0, peak_prop=0.95)
cv2.imwrite('ps1-2-a.png', img0_hough)
```

Out[21]:

```
True
```

Answer:

```
I choose the accumulator array size by using the similar size with the ori
ginal image. If the size of accumulator is too small, it is hard to calcul
ate a precise distance.
```



3 Now were going to add noise. For this question use the first one ps1-input0-noise.png .

```
a. This image is the same as before but with noise. Compute a modestly smo
othed version of this image by using a Gaussian filter. Make σ at least a
 few pixels big.
Output: smoothed image
```

In [134]:

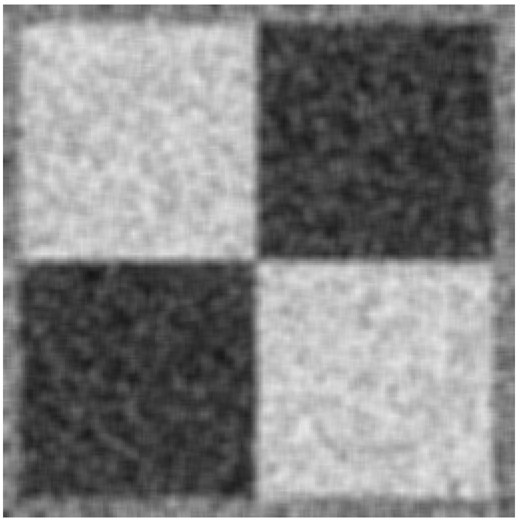
```
img0_noise = cv2.imread('ps1-input0-noise.png', cv2.IMREAD_COLOR)
```

In [135]:

```
img0_noise_filtered = cv2.GaussianBlur(img0_noise, (7,7), 3)
cv2.imwrite('ps1-3-a.png', img0_noise_filtered)
```

Out[135]:

True



b. Using an edge operator of your choosing, create a binary edge image for both the raw monochrome image and the smoothed version above.
Output: the two edge images

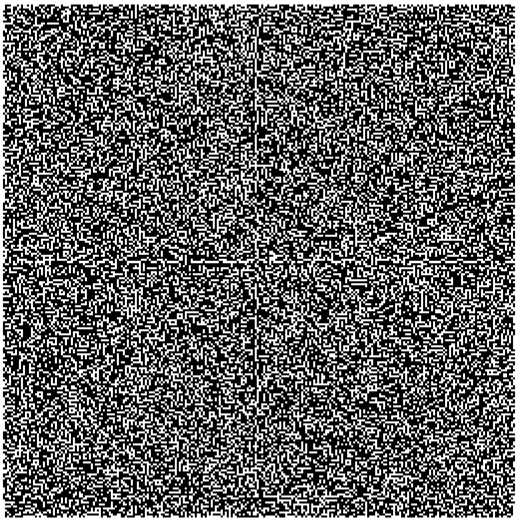
In [136]:

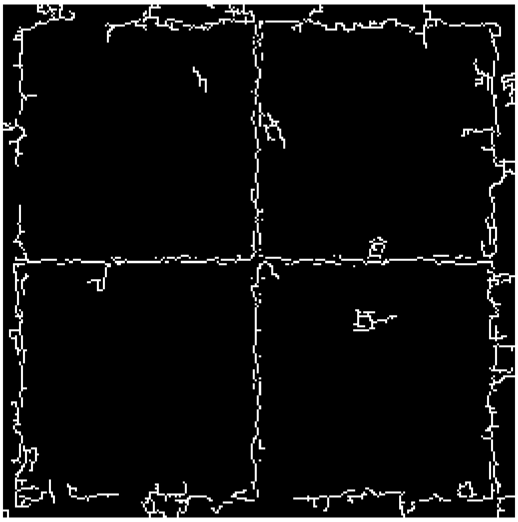
```
img0_noise_edge = cv2.Canny(img0_noise, 200, 250)
img0_noise_filtered_edge = cv2.Canny(img0_noise_filtered, 50, 160)

cv2.imwrite('ps1-3-b-1.png', img0_noise_edge)
cv2.imwrite('ps1-3-b-2.png', img0_noise_filtered_edge)
```

Out[136]:

True





```
c. Now apply your Hough method to the smoothed version of the edge image.
 Your goal is to adjust the filtering, the edge finding, and the Hough alg
orithms to find the lines as best you can in this test case

Output: Hough accumulator array image with peaks circled or somehow labele
d.
Output: intensity image (original one with the noise )with lines drawn on
 them
Output: describe what you had to do to get the best result you could.
```

In [137]:

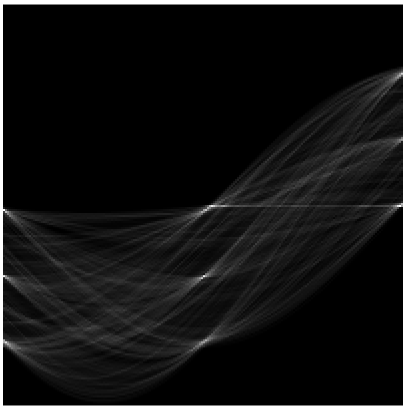
```
# Apply Hough method
img0_noise_filtered_hough, hough_img0_noise = hough_line(
    img = img0_noise_filtered, edge_img=img0_noise_filtered_edge, d_size=200, th
eta_size=200, peak_prop=0.8)
cv2.imwrite('ps1-3-c.png', img0_noise_filtered_hough)
```

Out[137]:

```
True
```

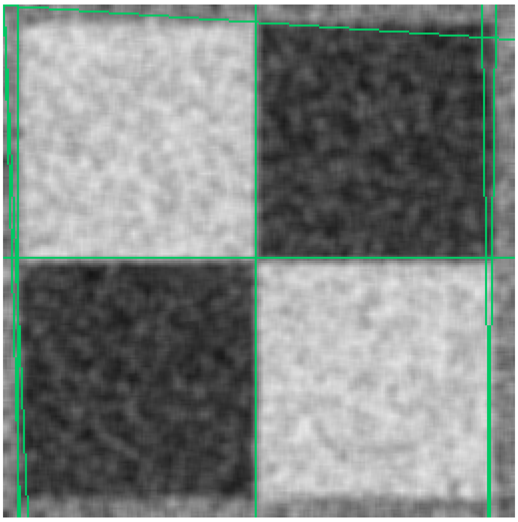
In [144]:

```
# Create accumulator array image
hough_max = np.max(hough_img0_noise)
hough_img0_noise[:,:] = (1.0 - hough_img0_noise/hough_max) *255.0
cv2.imwrite('ps1-3-c-1.png', hough_img0_noise)
```



Answer:

A small size of sigma can keep noises; a larger sigma can reduce noises. We should use a small but non-zero sigma to reduce some noises in the center of the image. Then the thresholds of the Canny edge function should be adjusted. The low threshold is smaller than the usual value to leave enough lines, which have no relationship with other lines.



4 For this question use the first one ps1-input1.jpg .

```
a. This image has objects in it whose boundaries are circles (coins) or li
nes (pens). For this question you're still finding lines. Create a monochr
ome version of the image and compute a modestly smoothed version of this i
mage by using a Gaussian filter. Make σ at least a few pixels big.
Output: smoothed image
```

In [6]:

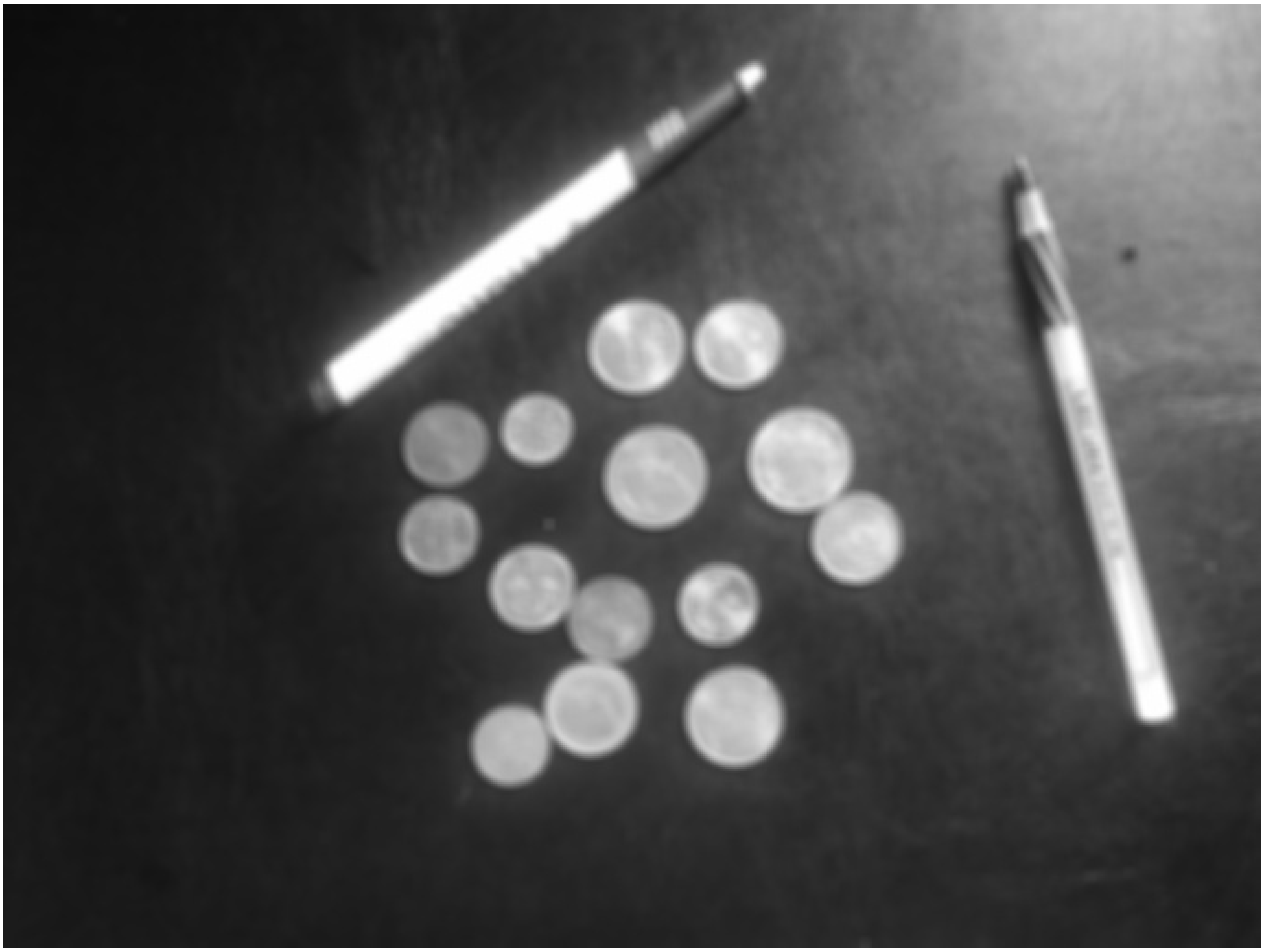
```
img1 = cv2.imread('ps1-input1.jpg')
img1_mono = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
```

In [8]:

```
img1_filtered = cv2.GaussianBlur(img1_mono,(7,7), 3)
cv2.imwrite('ps1-4-a.jpg', img1_filtered)
```

Out[8]:

True



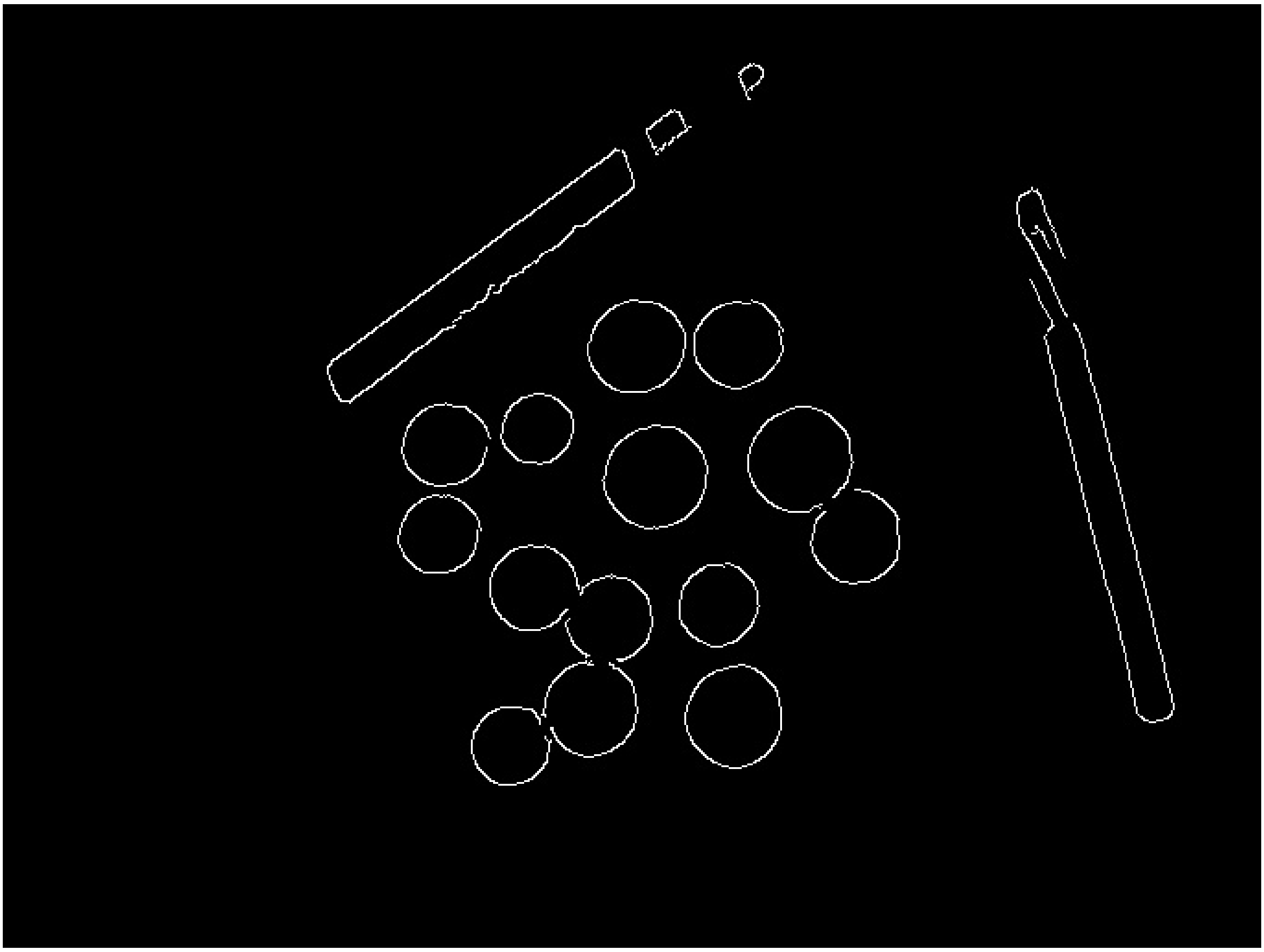
b. Using an edge operator and parameters of your choosing, create an edge image for the smoothed version above.
Output: the edge image

In [9]:

```
img1_filtered_edge = cv2.Canny(img1_filtered,100,200)
cv2.imwrite('ps1-4-b.jpg', img1_filtered_edge)
```

Out[9]:

True



c. Apply your Hough algorithm to find the lines along the pens. Draw the lines in color on the smoothed monochrome intensity (not edge) image. The lines can extend to the edge of the images (aka infinite lines).
Output: Hough accumulator array image with peaks circled or somehow labeled.
Output: intensity images with lines drawn on them
Output: describe what you had to do to get the best result you could

In [12]:

```
img1_mono_colored = cv2.cvtColor(img1_filtered, cv2.COLOR_GRAY2BGR)
img1_mono_hough, h_img1_mono = hough_line(img1_mono_colored, edge_img=img1_filtered_edge, d_size=600, theta_size=600, peak_prop=0.7, min_vote=100)
cv2.imwrite('ps1-4-c-2.png', img1_mono_hough)
```

Out[12]:

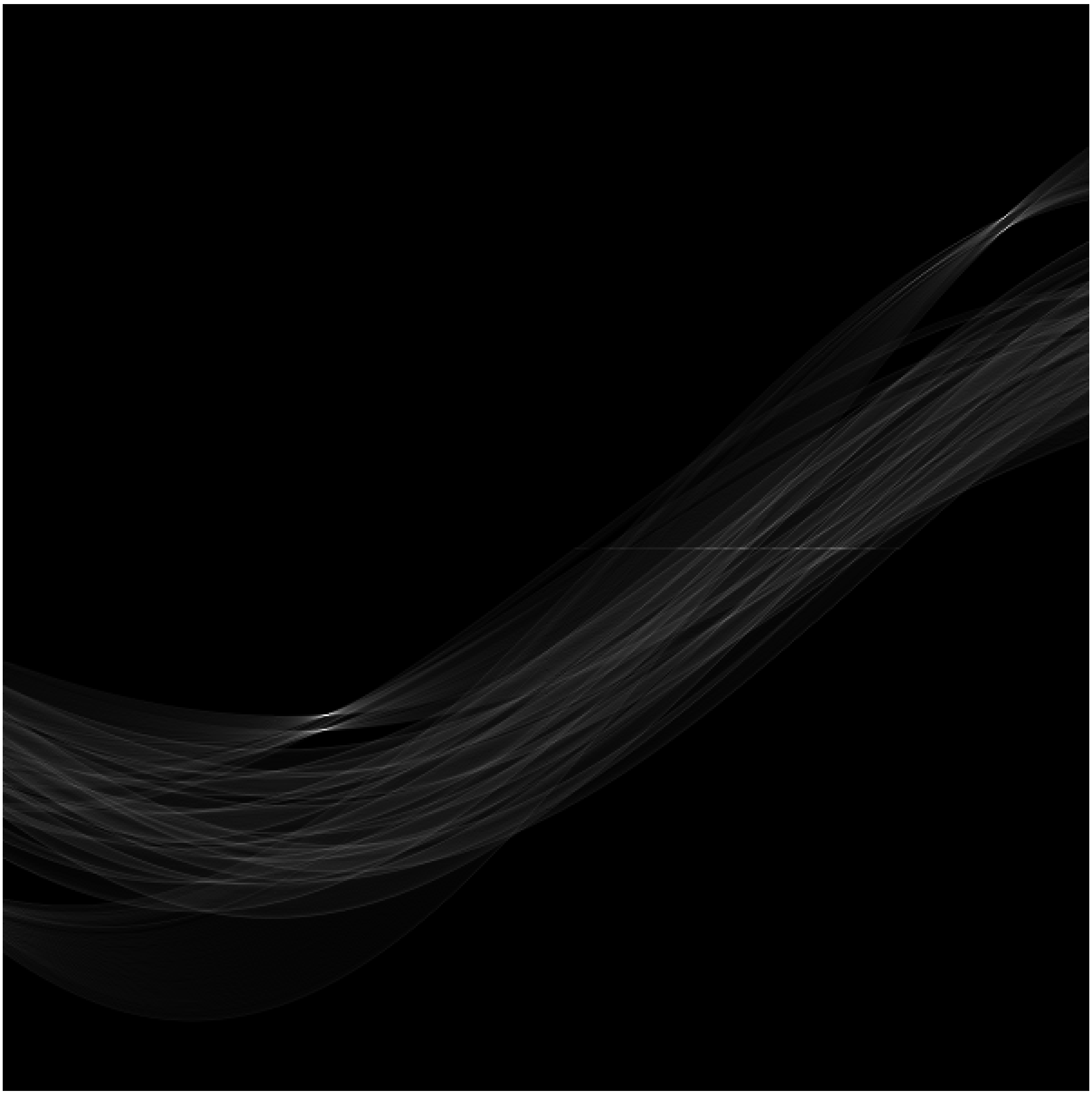
```
True
```

In [148]:

```
# Create accumulator array image
hough_max1 = np.max(h_img1_mono)
h_img1_mono = (1.0 - h_img1_mono/hough_max1) *255.0
cv2.imwrite('ps1-4-c-1.png', h_img1_mono)
```

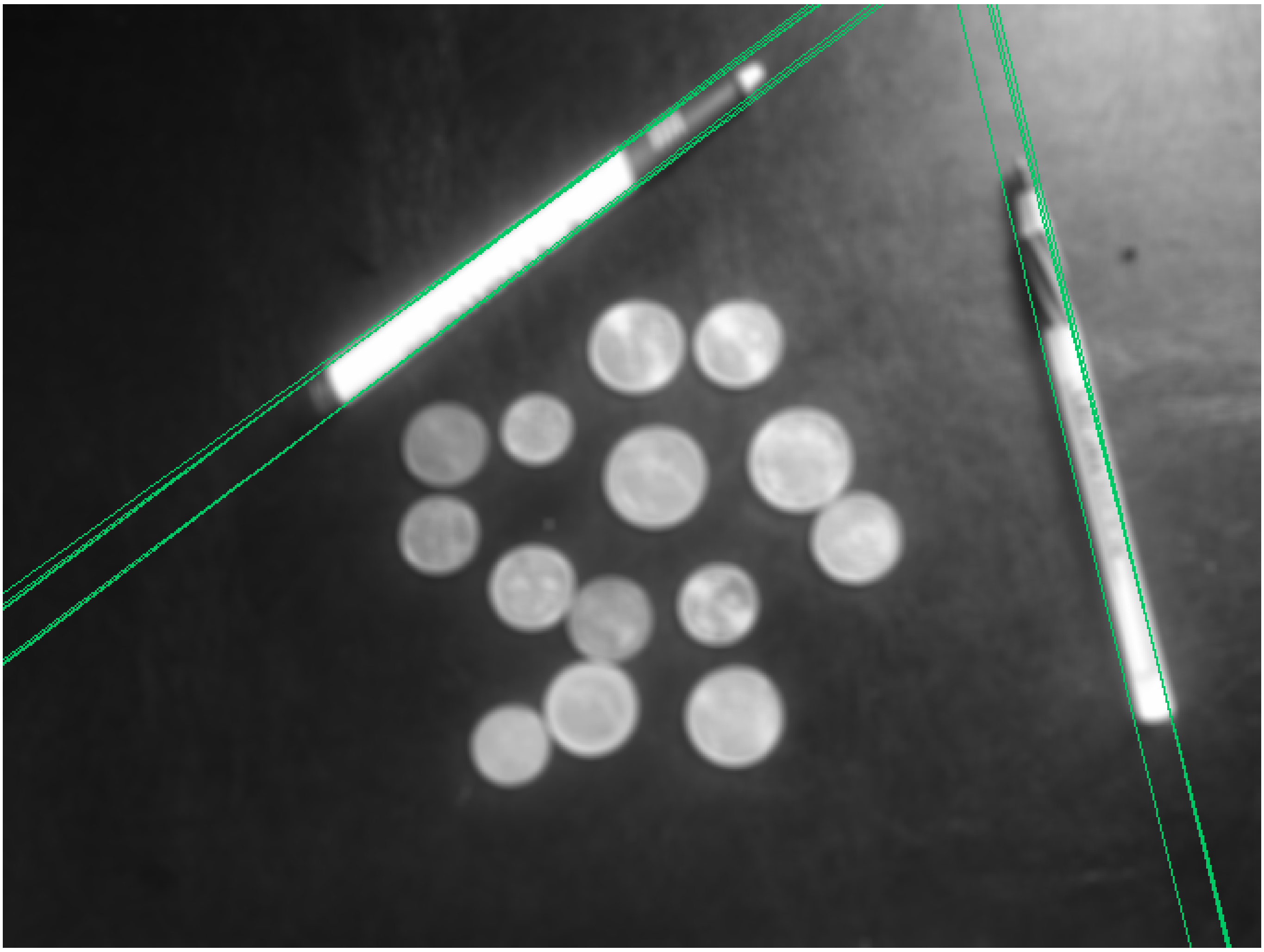
Out[148]:

```
True
```



Answer:

The filter should have some small values of sigma to remove noise edges. When building the hough space, I tried to add some smaller values to neighboring places, but that did not work well. So I only use thresholds to limit the hough space values to be counted as lines in the image.



```
5 Now for circles. Write a circle finding version of the Hough transform.
 You can implement either the single point method or the point plus gradie
nt method. THIS PART IS (SOMETIMES) HARDER THAN IT LOOKS – LEAVE EXTRA TIM
E FOR THIS!!!! TO TEST THIS YOU MIGHT MAKE YOUR OWN TEST IMAGE. If you fin
d your arrays getting too big (hint, hint) you might try make the range of
radii very small to start with and see if you can find one size circle. Th
en maybe try the different sizes.

a. Using the same original image as above. Smooth it, find the edges, find
the circles.
Output: edge image and images with the circles drawn in color
Output: describe what you had to do to find the circles.
```

In [122]:

```
def hough_circle(img, edge_img=None, gradient=False, r_size=100, r_max=100, peak
_prop=0.9, min_vote=10, decal=0.1, min_r=3):

    if edge_img is not None:
        edge = edge_img
    else:
        edge = cv2.Canny(img, 100, 200)

    if img is not None:
        img_out = img.copy()

    y_size, x_size = edge.shape
    H = np.zeros((y_size, x_size, r_size))

    # r_max = np.hypot(x_size, y_size) / 2
    r_list = np.arange(1, r_max, r_max/r_size)

    for xi in range(x_size):
        for yi in range(y_size):
            if edge[yi][xi] >= 200:
                if gradient:
                    pass
                else:
                    for r in range(1, r_size):
                        for deg in range(0, 360):
                            theta = np.deg2rad(deg)
                            ai = int(xi - r * cos(theta)+0.5)
                            bi = int(yi + r * sin(theta)+0.5)
                            r_ind = int((r - 1) * r_size / (r_max - 1))
                            if 0 <= ai < x_size and 0 <= bi < y_size:
                                H[bi][ai][r_ind + 1 if r+1 < r_max else r_max -
1] += decal
                                H[bi][ai][r_ind] += 1.0
                                H[bi][ai][r_ind - 1 if r_ind - 1 >= 0 else 0] +=
decal
    m = np.max(H)
    for a in range(x_size):
        for b in range(y_size):
            for r in range(min_r, r_size):
                if H[b][a][r] >= m * peak_prop and H[b][a][r] >= min_vote:
                    center_x = a
                    center_y = b
                    radius = int(r_list[r])

                    img_out = cv2.circle(img_out, (center_x, center_y), radius,
(0,255,0), 1)
    if img is not None:
        return img_out, H
    else:
        return H
```

In [17]:

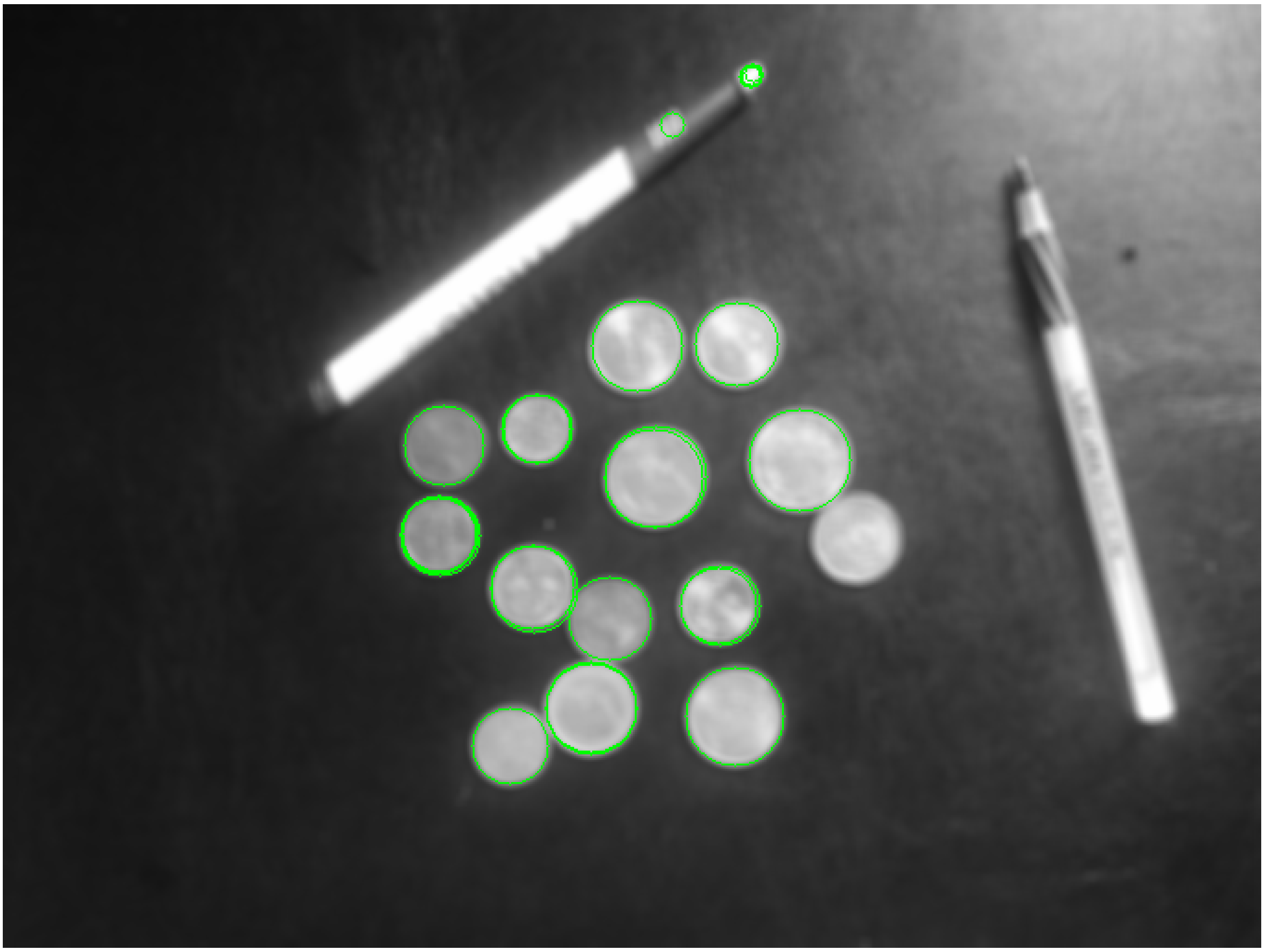
```
img1_mono_hough_circle, h_img1_circle = hough_circle(
    img1_mono_colored, edge_img=img1_filtered_edge,
    r_size=50, r_max=50,
    peak_prop=0.7, min_vote=150)
cv2.imwrite('ps1-5-a.png', img1_mono_hough_circle)
```

Out[17]:

```
True
```

Answer:

I did the finding circle Hough transform by testing different radius sizes. We need a hough space with three dimensions: x of the center, y of the center, and the radius. For each edge point in the original image, we try different sizes of angle theta and various size of radius to find reasonable centers of that point and add values to the hough space. After the hough space (an accumulator array) is built, find the center's max values and draw circles on the original image.



6 More realistic images. Now that you have Hough methods working, we’re go
ing to try them on images that have clutter in them: visual elements that
 are not part of the objects to be detected. The image to use is ps1-input
2.jpg
a. Apply your line finder. Use a smoothing filter and edge detector that s
eems to work best in terms of finding all the pen edges. Don’t worry (unti
l b) about whether you arefinding other lines.
Output: the smoothed image you used with the Hough lines drawn on them.

In [25]:

```
img2 = cv2.imread('ps1-input2.jpg')
```

In [94]:

```
img2_filtered = cv2.GaussianBlur(img2,(13,13), 3)
cv2.imwrite('ps1-6-a-1.jpg', img2_filtered)
img2_filtered_edge = cv2.Canny(img2_filtered,50,100)
cv2.imwrite('ps1-6-a-2.jpg', img2_filtered_edge)
```

Out[94]:

True

In [96]:

```
img2_pen_lines, h_img2_line = hough_line(img2_filtered, edge_img=img2_filtered_e
dge,
                                          d_size=600, theta_size=300, min_vote=50
, peak_prop=0.6)
cv2.imwrite('ps1-6-a.jpg', img2_pen_lines)
```

Out[96]:

True

b. Likely the last step found lines that are not the boundaries of the pen
s. What are the problems present?
Output: written response.

Answer:

The lines found are not only boundaries of the pens, but also boundaries of books and line patterns on the book.

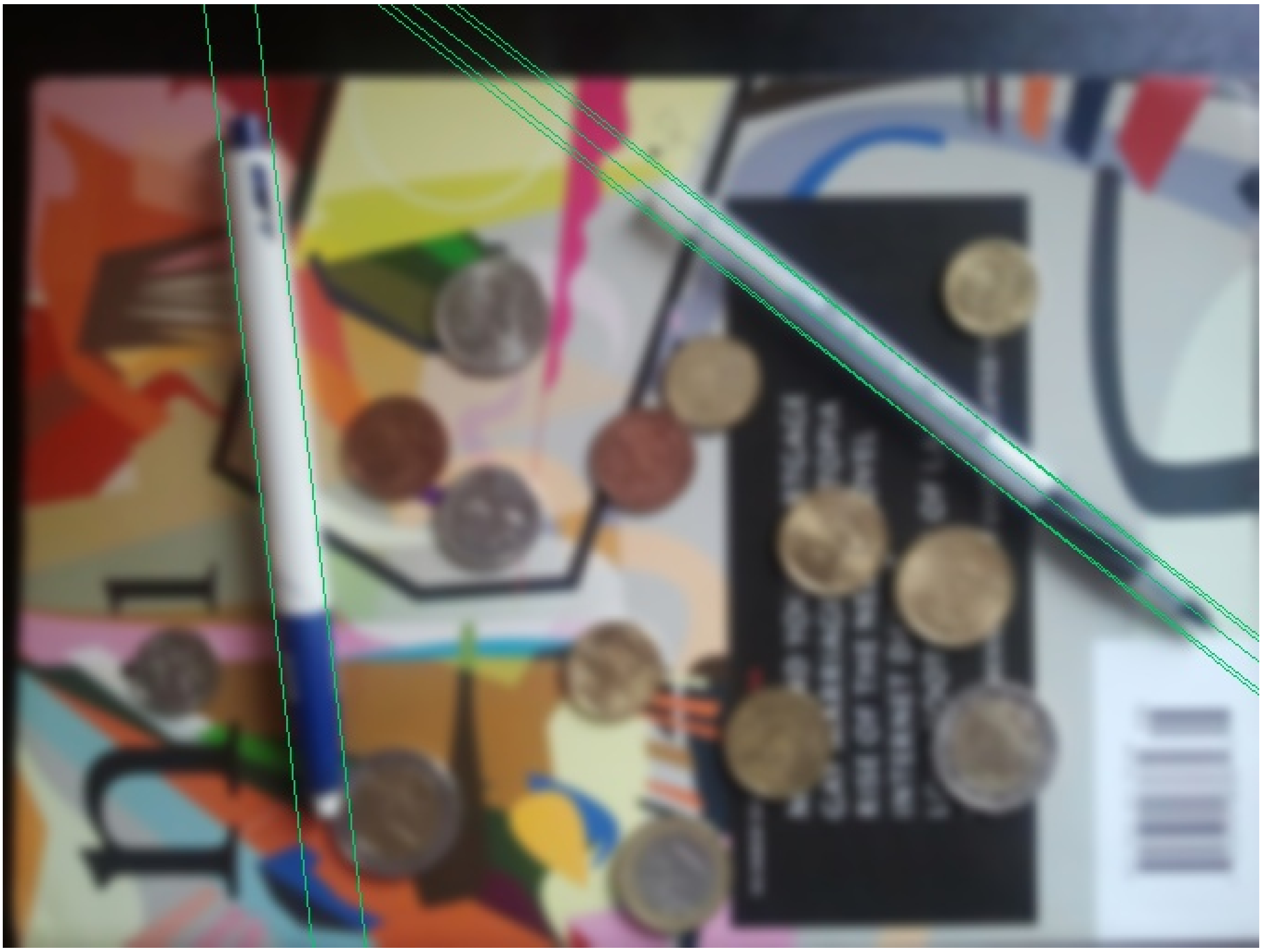
```
c. Attempt to find only the lines that are the *boundaries* of the pen. Th
ree operations you need to try are better thresholding in finding the line
s (look for stronger edges), checking the minimum length of the line, look
ing for nearby parallel lines
Output: Smoothed image with new Hough lines drawn.
```

In [97]:

```
img2_pen_lines2, h_img2_line2 = hough_line_pen(img2_filtered, edge_img=img2_filt
ered_edge,
                                             d_size=600, theta_size=300, min_vote=10
0, peak_prop=0.6)
cv2.imwrite('ps1-6-c.jpg', img2_pen_lines2)
```

Out[97]:

```
True
```



In [78]:

```
def hough_line_pen(img, edge_img=None, d_size=250, theta_size=250, peak_prop=0.9
, min_vote=100):

    if edge_img is not None:
        edge = edge_img
    else:
        edge = cv2.Canny(img, 100, 200)

    if img is not None:
        img_out = img.copy()

    H = np.zeros((d_size, theta_size))
    theta = np.deg2rad(np.arange(0.0, 180.0, 180.0/theta_size))

    d_max = int(np.hypot(len(img), len(img[0])))
    d_max = round(d_max, 2)
    d_step = d_max / d_size * 2

    distance = np.arange(0-d_max, d_max, d_step)
    true_max_d = 0.0
    true_min_d = 200.0
    true_max_d_index = 0
    true_min_d_index = 100

    y_size, x_size = edge.shape

    for i in range(x_size):
        for j in range(y_size):
            if edge[j][i] >= 200:
                for k in range(len(theta)):
                    ang = theta[k]
                    d = i * cos(ang) + j * sin(ang)
                    d_index = int(d/d_step + 0.5) + int(d_size/2)
                    H[d_index][k] += 1
                    # H[d_index - 1 if d_index > 0 else 0][k] += 0.01
                    # H[d_index + 1 if d_index < d_size - 1 else d_size - 1] +=
 0.01

    i, j = H.shape
    m = np.max(H)
    for p in range(i):
        for q in range(j):
            if H[p][q] >= m * peak_prop and H[p][q] >= min_vote:
                has_par = False
                for i in range(1, 50):
                    val1 = H[p-i if p-i > 0 else 0][q]
                    if val1 >= m * peak_prop and val1 >= min_vote:
                        has_par = True
                        break
                    val2 = H[p+i if p+i < d_size-1 else d_size-1][q]
                    if val2 >= m * peak_prop and val2 >= min_vote:
                        has_par = True
                        break
                if has_par:
                    d = (p - d_size/2) * d_step
                    t = theta[q]

                    if abs(t - pi/2) > 0.01:
                        if t > 0.01:
```

```
                        p1_x = int(d / cos(t) + 0.5)
                        p2_x = int( (d - y_size * sin(t)) / cos(t) + 0.5)
                        p1_y = 0
                        p2_y = y_size
                    else:
                        p1_x = int(d)
                        p2_x = int(d)
                        p1_y = 0
                        p2_y = y_size

                else:
                    p1_x = 0
                    p2_x = x_size -1
                    p1_y = int(d)
                    p2_y = int(d)

                img_out = cv2.line(img_out, (p1_x, p1_y), (p2_x, p2_y), (100
,200,0), 1)
    if img is not None:
        return img_out, H
    else:
        return H
```

7 Finding circles on the same clutter image.
a. Apply your circle finder. Use a smoothing filter that seems to work bes
t in terms of finding all the coins.
Output: the smoothed image you used with the circles drawn on them.

In [104]:

```
img2_circle_filtered = cv2.GaussianBlur(img2, (9,9), 2)
img2_circle_filtered_edge = cv2.Canny(img2_circle_filtered, 50, 100)
cv2.imwrite('ps1-7-a-1.jpg', img2_circle_filtered)
cv2.imwrite('ps1-7-a-2.jpg', img2_circle_filtered_edge)
```

Out[104]:

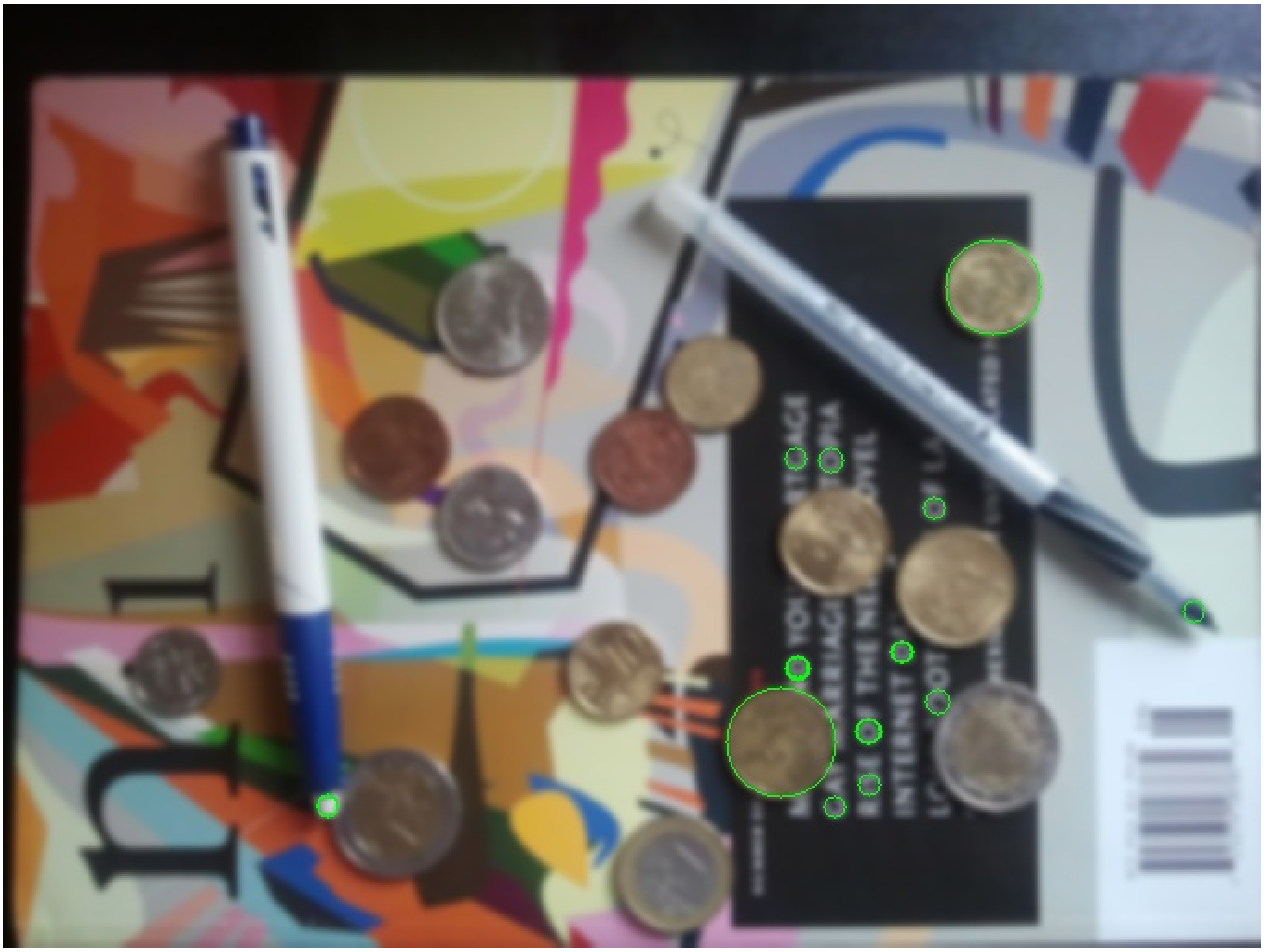
True

In [129]:

```
img2_circle, h_img2_circle = hough_circle(img2_circle_filtered, edge_img=img2_ci
rcle_filtered_edge,
                                          r_size=100, r_max=100, peak_prop=0.6,
min_vote=100, decal=0.5, min_r=5)
cv2.imwrite('ps1-7-a.jpg', img2_circle)
```

Out[129]:

True



```
b. Are there any false alarms? How would/did you get rid of them?
Output: written response (if you did these steps mention that they are in
 the code)
```

I think the problem is that with a colored background, the edge of coins are nor so clear as it was on a black background. So when adding values to the hough space, I also added some smaller values to the nearby. And some small, not circle patterns were detected as the circle. So I also added a parameter to limit the smallest radius.

```
8 Sensitivity to distortion. There is a distorted version of the scene at
 ps1-input3.jpg
a. Apply the line and circle finder to the distorted image. Can you find l
ines? The circles?
Output: Image with lines and circles (if any) found.
```

In [109]:

```
img3 = cv2.imread('ps1-input3.jpg')
```

In [120]:

```
img3_lines, h_img3_lines = hough_line(img3_filtered, edge_img=img3_canny,
                                     d_size=1000, theta_size=300, peak_prop=0.8
, min_vote=50)
cv2.imwrite('ps1-8-a-1.jpg', img3_lines)
```

Out[120]:

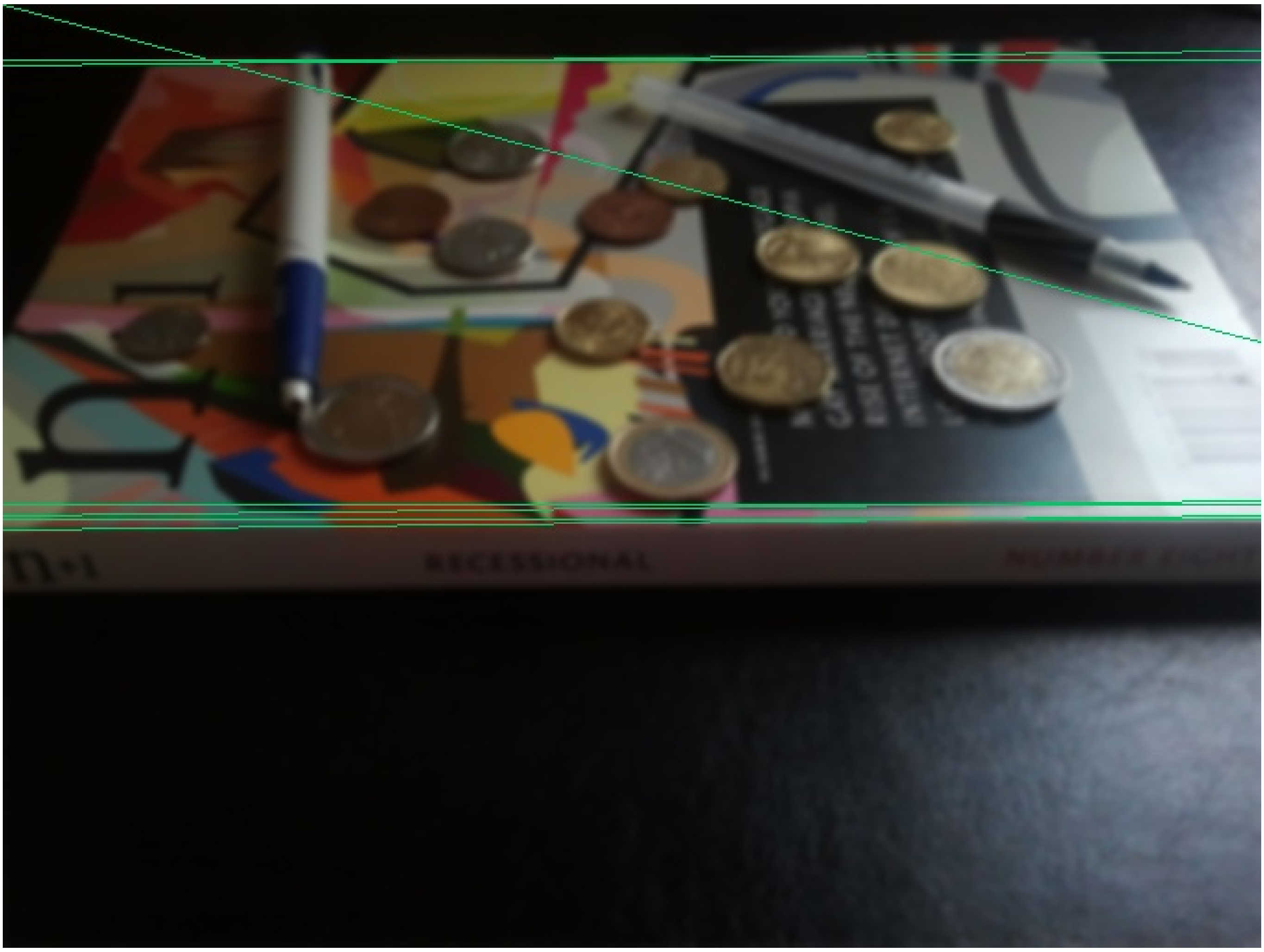
```
True
```

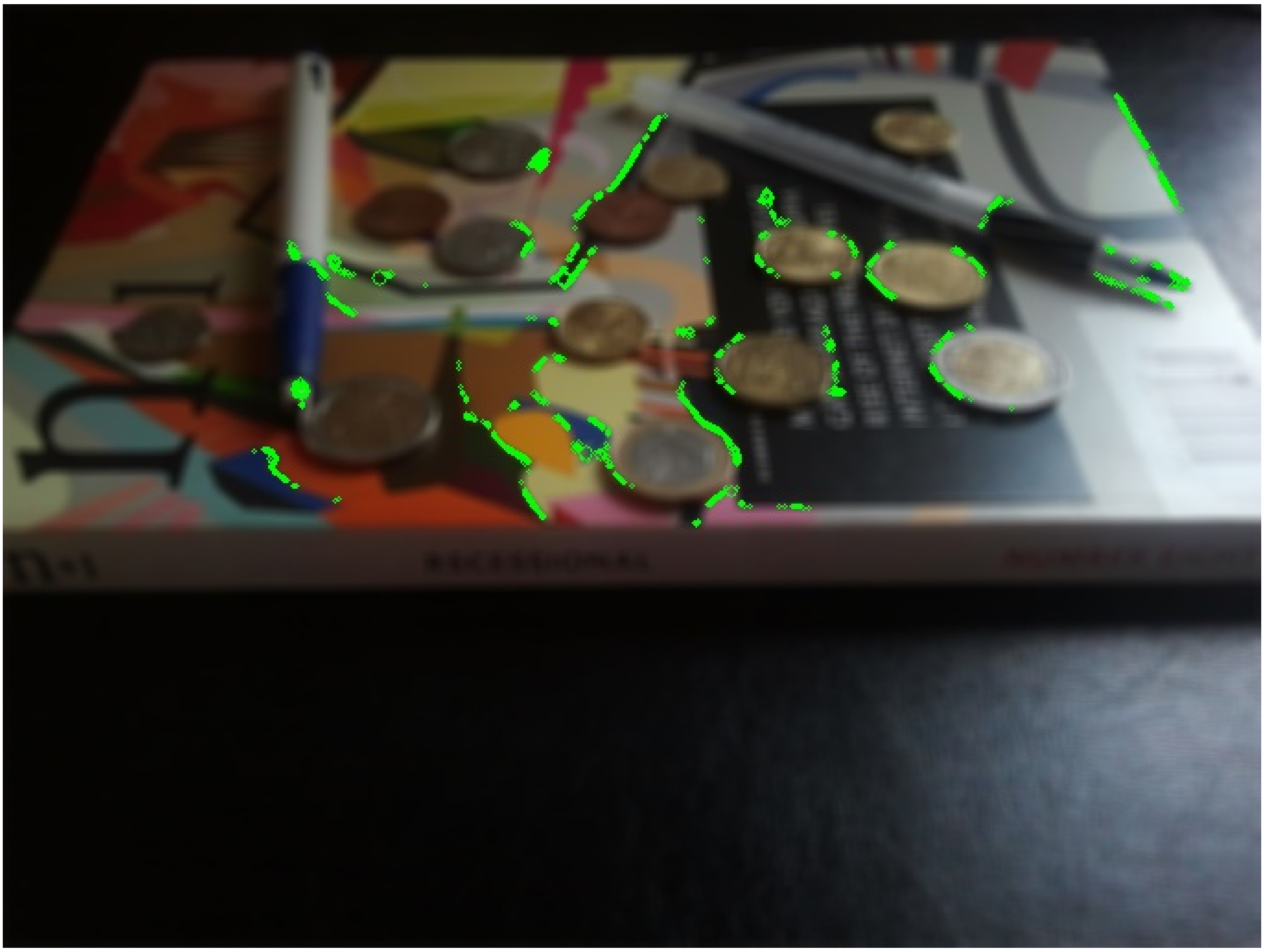
In [156]:

```
img3_circles, h_img3_circle = hough_circle(img3_filtered,edge_img=img3_canny,
                                          r_size=100, r_max=100, peak_prop=0.6, m
in_vote=200, min_r=1)
cv2.imwrite('ps1-8-a-2.jpg', img3_circles)
```

Out[156]:

```
True
```





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

Answer:

The circles are distorted; it looks like an ellipse but may not a standard ellipse. Maybe we can use the formula of an ellipse to build the hough space. x^2/a^2 + y^2/b^2 = 1, so we need parameters a and b as another two dimensions to calculate the ellipse center. But the four dimension needs a significant amount of time.

```
c. Try to fix the circle problem THIS IS HARD.
Output: Written response describing what tried and what worked best. Outpu
t: Image that is the best shot at fixing the circle problem.
```

In [149]:

```
def hough_ellipse(img, edge_img=None, ra_size=100, rb_size=100, r_max=100, peak_
prop=0.9, min_vote=10, decal=0.1):

    if edge_img is not None:
        edge = edge_img
    else:
        edge = cv2.Canny(img, 100, 200)

    if img is not None:
        img_out = img.copy()

    y_size, x_size = edge.shape
    H = np.zeros((y_size, x_size, ra_size, rb_size))

    # r_max = np.hypot(x_size, y_size) / 2
    ra_list = np.arange(1, r_max, r_max/ra_size)
    rb_list = np.arange(1, r_max, r_max/rb_size)

    for xi in range(x_size):
        for yi in range(y_size):
            if edge[yi][xi] >= 200:
                for ra in range(1, ra_size):
                    for rb in range(1, rb_size):
                        for deg in range(0, 360):
                            theta = np.deg2rad(deg)
                            ai = int(xi - ra * cos(theta) + 0.5)
                            bi = int(yi + ra * sin(theta) + 0.5)

                            ra_ind = int((ra - 1) * ra_size / (r_max - 1))
                            rb_ind = int((rb - 1) * rb_size / (r_max - 1))
                            if 0 <= ai < x_size and 0 <= bi < y_size:
                                # H[x_a][y_b][r_ind + 1 if r+1 < r_max else r_ma
x - 1] += 0.5

                                H[bi][ai][ra_ind][rb_ind] += 1.0
                                # H[x_a][y_b][r_ind - 1 if r_ind - 1 >= 0 else
 0] += 0.5
    m = np.max(H)
    for a in range(x_size):
        for b in range(y_size):
            for ra in range(ra_size):
                for rb in range(rb_size):
                    if H[b][a][ra][rb] >= m * peak_prop and H[b][a][ra][rb] > mi
n_vote:
                        center_x = a
                        center_y = b
                        radius_a = int(ra_list[ra])
                        radius_b = int(rb_list[rb])
                        img_out = cv2.ellipse(img_out, (center_x, center_y), (ra
dius_a, radius_b), (0,255,0), 1)
    if img is not None:
        return img_out, H
    else:
        return H
```

In [ ]: