CS6476 - PS1: Edges and Lines

Martin Saint-Jalmes (msaintjalmes3)

In [1]:

```
# Importing Numpy, OpenCV and Matplotlib
import cv2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
```

In [2]:

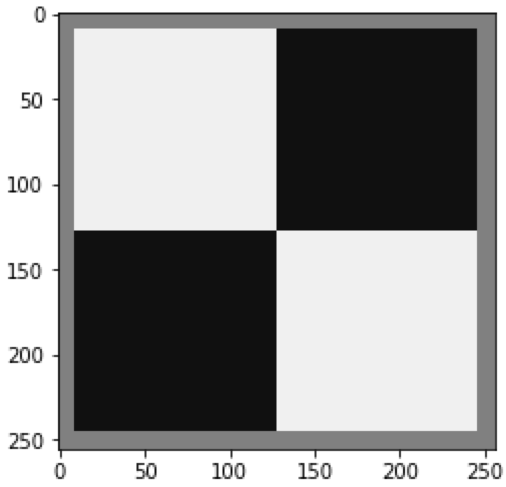
```
#Importing python utilities
import math
from timeit import default_timer as timer
import pickle
```

In [3]:

```
img0 = cv2.imread('./ps1-input0.png', cv2.IMREAD_GRAYSCALE)
plt.imshow(img0, cmap='gray', vmin = 0, vmax = 255)
```

Out[3]:

```
<matplotlib.image.AxesImage at 0x1729e9b0>
```



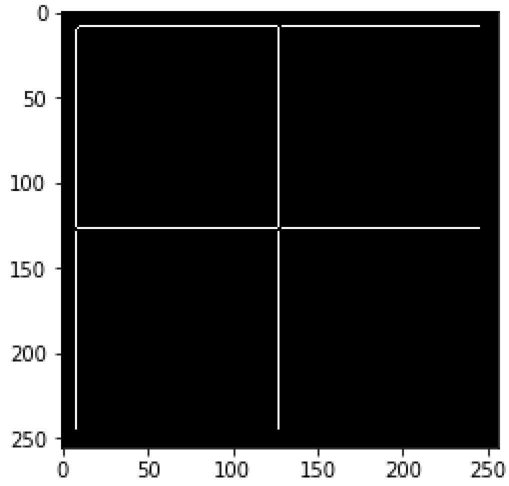
1. Edge image

In [4]:

```
edges = cv2.Canny(img0, 10, 20)
plt.imshow(edges, cmap='gray', vmin = 0, vmax = 255)
```

Out[4]:

```
<matplotlib.image.AxesImage at 0x1740e898>
```



In [5]:

```
edges_cord = np.argwhere(edges == 255)
edges_cord.shape[0] #edges detected
```

Out[5]:

```
1406L
```

2. Hough method for finding lines

In [6]:

```
(y_max, x_max) = img0.shape
rad = lambda deg: deg * math.pi / 180. #lambda for converting to rad
```

In [7]:

```
d_max = int(math.ceil(math.sqrt(x_max ** 2 + y_max ** 2))) #polar transformation
theta_max = 360 #all possible directions
```

In [8]:

```
H = np.zeros((d_max, theta_max))
```

In [9]:

```
for edge in edges_cord:
    for theta in range(-theta_max/2, theta_max/2):
        d = np.abs(edge[1] * math.cos(rad(theta)) + edge[0] * math.sin(rad(theta))) #d
 > 0
        H[int(round(d)), (theta_max/2 + theta)] += 1 #sampling per unit
```

In [10]:

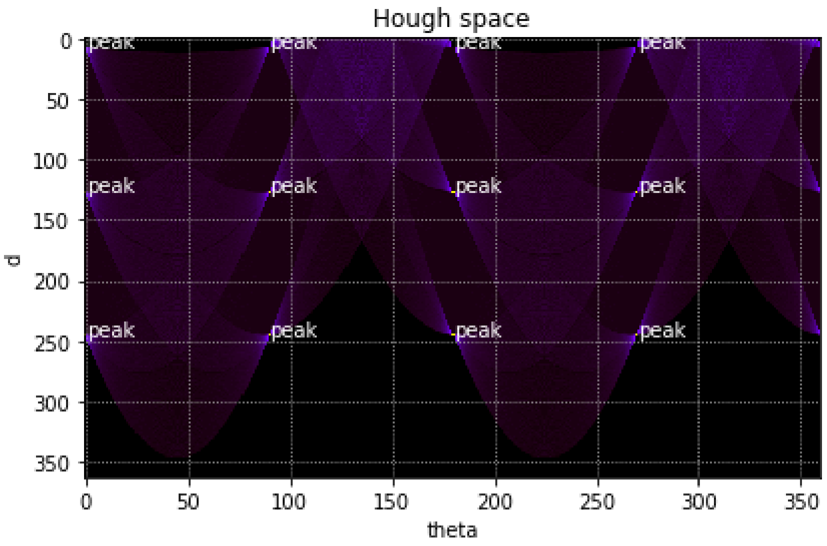
```
max_values = np.argwhere(H > H.max() * 0.5) #thresholding
max_values
```

Out[10]:

```
array([[  8,   0],
       [  8,  90],
       [  8, 180],
       [  8, 270],
       [127,   0],
       [127,  90],
       [127, 180],
       [127, 270],
       [245,   0],
       [245,  90],
       [245, 180],
       [245, 270]], dtype=int64)
```

In [11]:

```
plt.figure()
plt.imshow(H, cmap = 'gnuplot') #a cmap that makes the sinusoids visible
for max in max_values:
    plt.annotate('peak', xy=(max[1], max[0]), color='white', fontsize=10)
plt.grid(True, linestyle = ':')
plt.axis('tight')
plt.xlabel('theta')
plt.ylabel('d')
plt.title('Hough space')
plt.tight_layout()
```



In [12]:

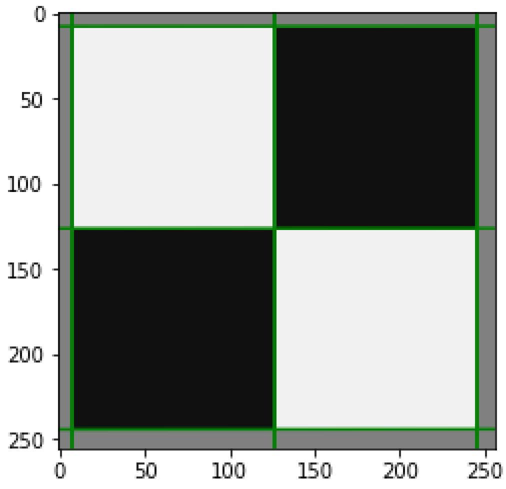
```
def produce_line(peak):
    if peak[1] == 0: #case where theta = 0
        return [peak[0], peak[0], 0, 1000]

    # turning back to cartesian coordinates
    # x = r cos(theta), y = r sin(theta)
    #but we need some margin to define the (infinite) lines/segments and not just point
s
    return [int(peak[0] * np.cos(rad(peak[1] - 90)) - 1000 *(-np.sin(rad(peak[1] - 90
)))), \
            int(peak[0] * np.cos(rad(peak[1] - 90)) + 1000 *(-np.sin(rad(peak[1] - 90
)))), \
            int(peak[0] * np.sin(rad(peak[1] - 90)) - 1000 *(np.cos(rad(peak[1] - 90
)))), \
            int(peak[0] * np.sin(rad(peak[1] - 90)) + 1000 *(np.cos(rad(peak[1] - 90
))))]
    # [xstart, xend, ystart, yend]
```

In [13]:

```
plt.figure()
for line in np.array(map(produce_line, max_values)):
    plt.plot(line[0:2], line[2:4], c = 'g')
plt.imshow(img0, cmap='gray', vmin = 0, vmax = 255)
```

Out[13]:

```
<matplotlib.image.AxesImage at 0x1add14a8>
```



A very straightoforward discretization was chosen for the bin sizes. Since this image is very sharp with distinct lines, we have very clear edges with no noise. As we can see in the Hough accumulator plot, all the sinusoids intersect at very distinct points, corresponding to our peaks (in Hough space, i.e. lines in image space). So the most refined discretization, with one bin for each degree of theta (360) and each increment of the unit of r (with a rounding function, so 363 in total) has been suitable for this part.

3. Noise

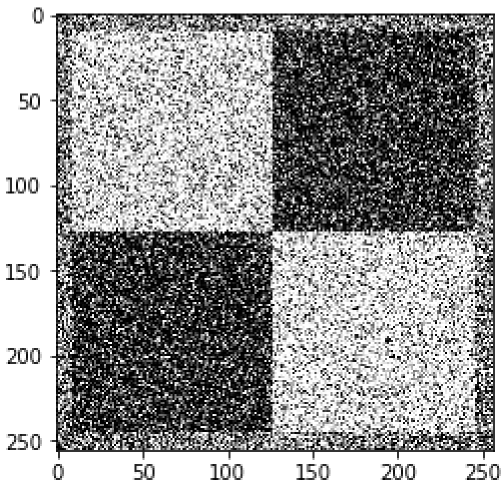
3.a. Smoothed image

In [14]:

```
img0_noisy = cv2.imread('./ps1-input0-noise.png', cv2.IMREAD_GRAYSCALE)
plt.imshow(img0_noisy, cmap='gray', vmin = 0, vmax = 255)
```

Out[14]:

```
<matplotlib.image.AxesImage at 0x1af471d0>
```

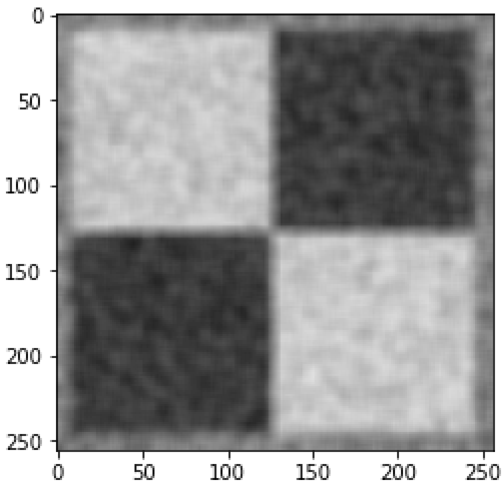


In [15]:

```
img0_blurred = cv2.GaussianBlur(img0_noisy, (11, 11), 4, cv2.BORDER_REFLECT)
plt.imshow(img0_blurred, cmap='gray', vmin = 0, vmax = 255)
```

Out[15]:

```
<matplotlib.image.AxesImage at 0x1ae30b70>
```



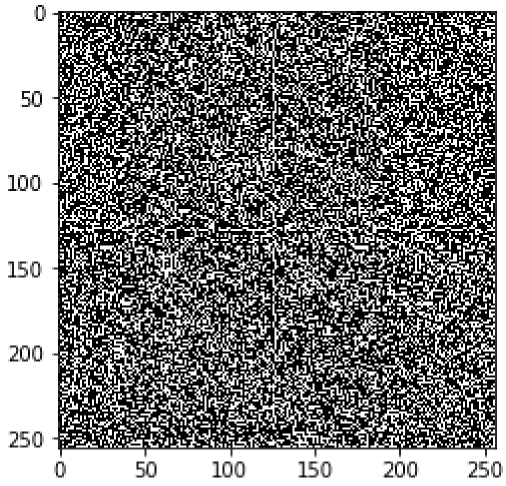
3.b. Edges for regular and smoothed images

In [16]:

```
edges_noisy = cv2.Canny(img0_noisy, 40, 80)
plt.imshow(edges_noisy, cmap='gray', vmin = 0, vmax = 255)
```

Out[16]:

```
<matplotlib.image.AxesImage at 0x177d0978>
```

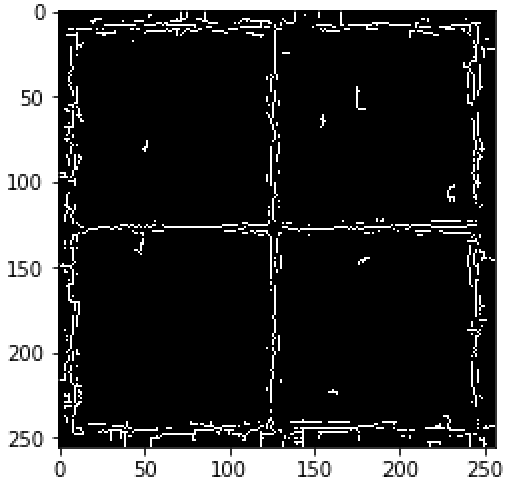


In [17]:

```
edges_blurred = cv2.Canny(img0_blurred, 40, 70) #higher thresholds to reduce noise inside squares
plt.imshow(edges_blurred, cmap='gray', vmin = 0, vmax = 255)
```

Out[17]:

```
<matplotlib.image.AxesImage at 0x177cdb38>
```



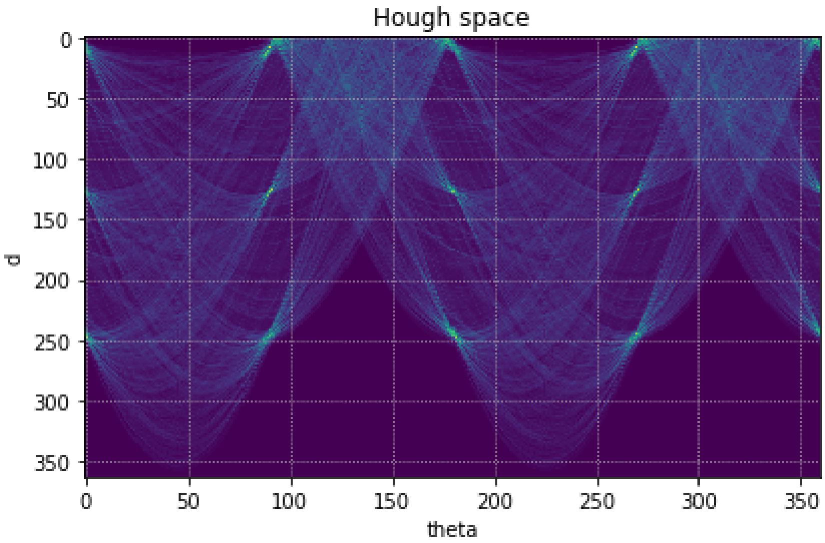
3.c. Hough transformed of the blurred image

In [18]:

```
H = np.zeros((d_max, theta_max))

edges_cord = np.argwhere(edges_blurred == 255)
for edge in edges_cord:
    for theta in range(0, theta_max):
        d = np.abs(edge[1] * math.cos(rad(theta)) + edge[0] * math.sin(rad(theta)))
        H[int(math.ceil(d)), theta] += 1
```

In [19]:

```
plt.figure()
plt.imshow(H)
plt.grid(True, linestyle = ':')
plt.axis('tight')
plt.xlabel('theta')
plt.ylabel('d')
plt.title('Hough space')
plt.tight_layout()
```



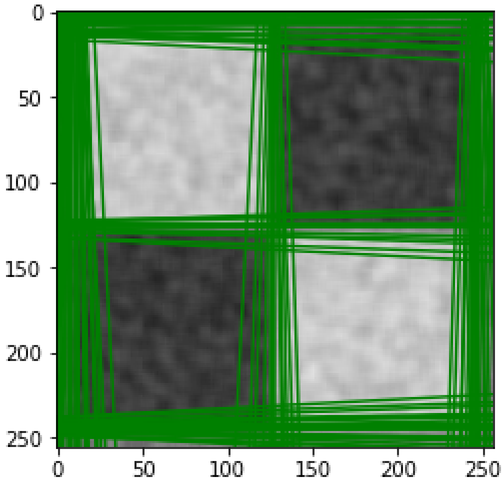
Compared to the previous (noise-free) image, it is difficult to exactly determine the peaks from the previous image. While we have a general idea (both visually and from the previous experience), the bright areas with a lot of votes seem more spread out.
The simple bins (1 for each increment of a degree in theta and each increment in r) are not suitable anymore. We have to rely on larger bins that will be able to detect lines even with noise on the image.
If we continue ploting lines for the peaks in Hough space with this binning, we get a lot of lines that are close to each other (off by only a few pixels):

In [20]:

```
plt.figure()
for line in np.array(map(produce_line, np.argwhere(H > H.max() * 0.5))):
    plt.plot(line[0:2], line[2:4], c = 'g')
plt.imshow(img0_blurred, cmap='gray', vmin = 0, vmax = 255)
```

Out[20]:

```
<matplotlib.image.AxesImage at 0x1bb24198>
```



We can vary the threshold for peak selection (here, only those that have half as much votes in their bins as the most voted one) to reduce the amount of lines that are going to be plotted, but it does not give satisfying results as some edges will have all lines disapearing together while others will keep multiple similar lines with a higher threshold.
We will now define a custom accumulator for larger bins.

In [21]:

```
# setting theta from -90 to +90° to avoid redundant lines & reduce search space
def custom_accumulator(edges, d_max, theta_max = 180, d_step = 1, theta_step = 1):
    H = np.zeros((d_max / d_step, theta_max / theta_step))

    edges_cord = np.argwhere(edges == 255)
    for edge in edges_cord:
        for theta in range(-theta_max/2, theta_max/2):
            d = np.abs(edge[1] * math.cos(rad(theta)) + edge[0] * math.sin(rad(theta)))
            H[int(math.floor(d / d_step)), (theta_max/2 + theta) / theta_step] += 1

    return H
```

In [22]:

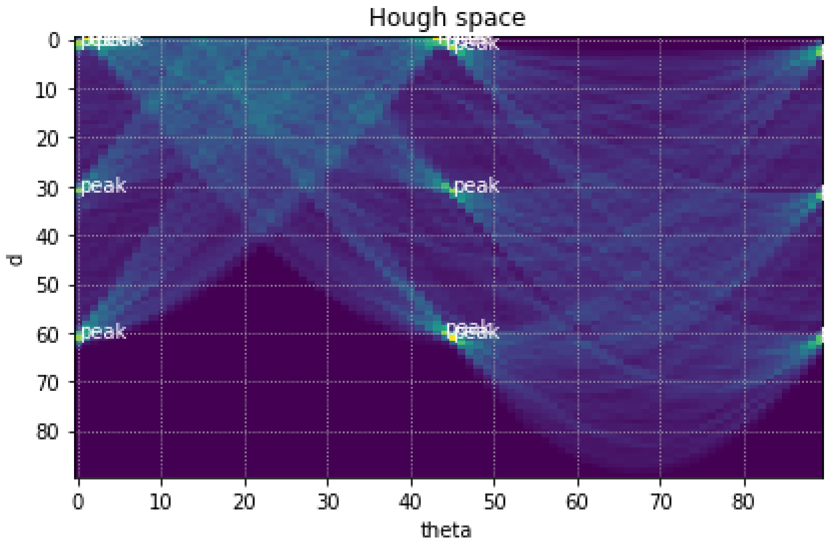
```
def sampling_fix(peak, d_step = 1, theta_step = 1):
    return [int((peak[0] + 0.5)*d_step), int((peak[1] + 0.5)*theta_step)]
```

In [23]:

```
theta_max = 180 #theta from -90 to +90°
H = custom_accumulator(edges_blurred, d_max, theta_max, 4, 2)
max_values = np.argwhere(H > H.max() * 0.71)
```

In [24]:

```
plt.figure()
plt.imshow(H)
for max in max_values:
    plt.annotate('peak', xy=(max[1], max[0]), color='white', fontsize=10)
plt.grid(True, linestyle = ':')
plt.axis('tight')
plt.xlabel('theta')
plt.ylabel('d')
plt.title('Hough space')
plt.tight_layout()
```

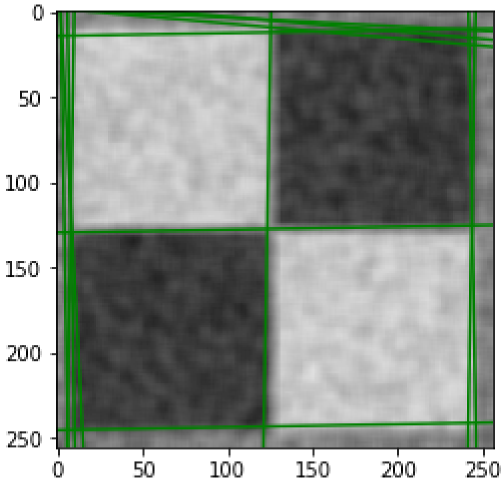


In [25]:

```
plt.figure()
for line in np.array(map(produce_line, [sampling_fix(peak, 4, 2) for peak in max_values
])):
    plt.plot(line[0:2], line[2:4], c = 'g')
plt.imshow(img0_blurred, cmap='gray', vmin = 0, vmax = 255)
```

Out[25]:

```
<matplotlib.image.AxesImage at 0x1cff2ac8>
```



There are a few parameters that were tuned here: the theta and d steps, allowing larger bins as well as the threshold for selecting peaks (given as a ratio of the votes in the most voted bin).
The output image is not perfect, as there is still a couple of lines that shouldn't appear (but can't be removed by varying the threshold parameter without deleting another important line).
This is still an improvement compared to using the standard (1, 1) bin steps that we have seen previously. By plotting the points in Hough space, we can see that the peaks are much less spread out and are clearer (on one point instead of being on a region).

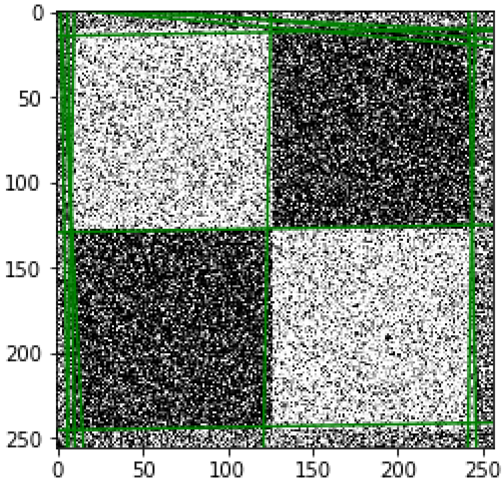
This is the result we have on the original image with noise:

In [26]:

```
plt.figure()
for line in np.array(map(produce_line, [sampling_fix(peak, 4, 2) for peak in max_values
])):
    plt.plot(line[0:2], line[2:4], c = 'g')
plt.imshow(img0_noisy, cmap='gray', vmin = 0, vmax = 255)
```

Out[26]:

```
<matplotlib.image.AxesImage at 0x1ce22470>
```



4. Image with pens and coins

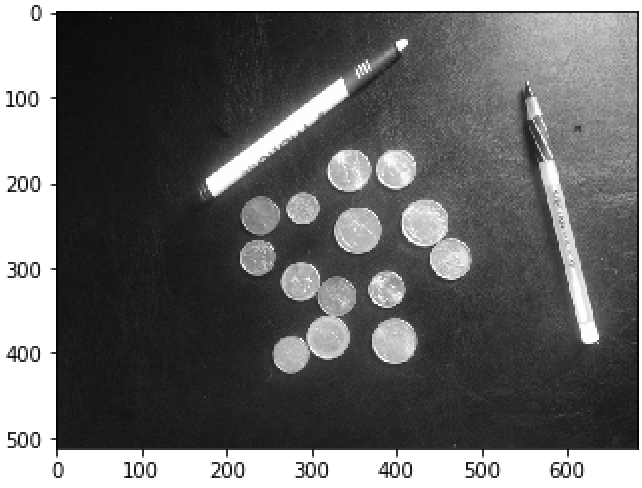
4.a. Smoothing the image

In [27]:

```
img1 = cv2.imread('./ps1-input1.jpg', cv2.IMREAD_GRAYSCALE)
plt.imshow(img1, cmap='gray', vmin = 0, vmax = 255)
```

Out[27]:

```
<matplotlib.image.AxesImage at 0x1d3f8198>
```

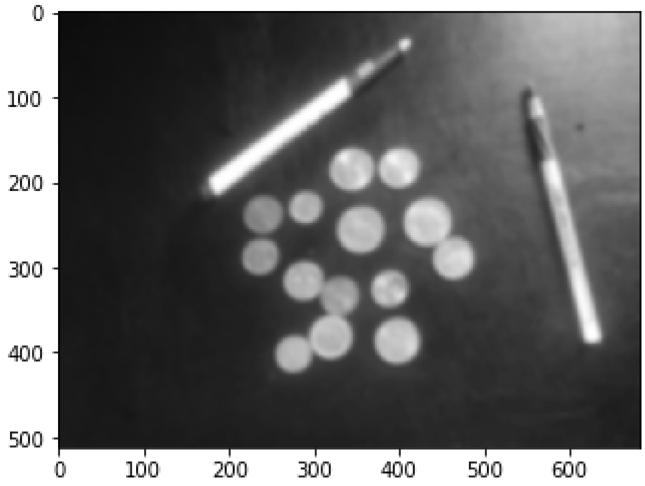


In [28]:

```
img1_blurred = cv2.GaussianBlur(img1, (19, 19), 3, cv2.BORDER_REFLECT)
plt.imshow(img1_blurred, cmap='gray', vmin = 0, vmax = 255)
```

Out[28]:

```
<matplotlib.image.AxesImage at 0x1d64df98>
```



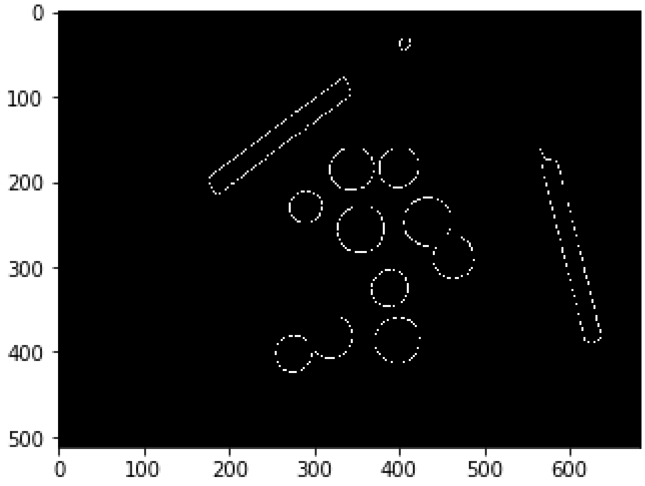
4.b. Edge image

In [29]:

```
img1_edges_blurred = cv2.Canny(img1_blurred, 100, 200)
plt.imshow(img1_edges_blurred, cmap='gray', vmin = 0, vmax= 255)
```

Out[29]:

```
<matplotlib.image.AxesImage at 0x1cd3e898>
```



4.c. Hough for pens

In [30]:

```
(y_max, x_max) = img1.shape

d_max = int(math.ceil(math.sqrt(x_max ** 2 + y_max ** 2))) #polar transformation
theta_max = 180
```

In [31]:

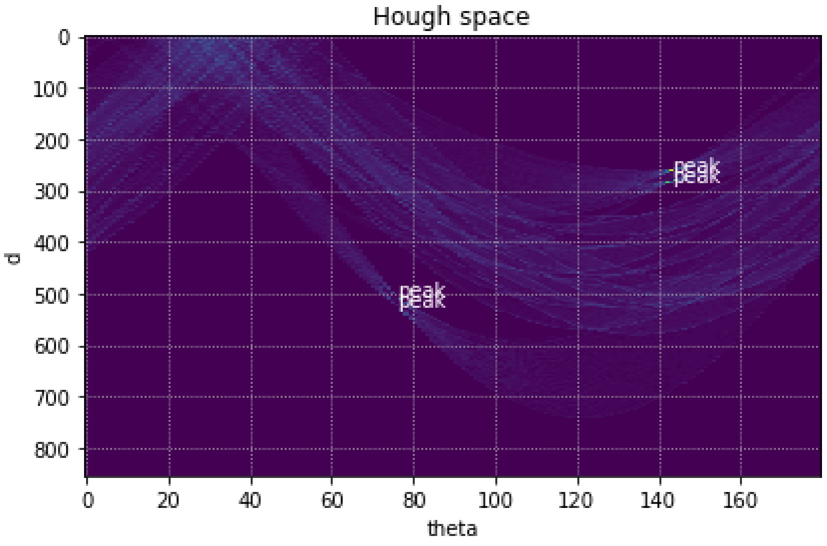
```
H = custom_accumulator(img1_edges_blurred, d_max, theta_max, 1, 1)
max_values = np.argwhere(H > H.max() * 0.8)
```

In [32]:

```
plt.figure()
plt.imshow(H)
for max in max_values:
    plt.annotate('peak', xy=(max[1], max[0]), color='white', fontsize=10)
plt.grid(True, linestyle = ':')
plt.axis('tight')
plt.xlabel('theta')
plt.ylabel('d')
plt.title('Hough space')
plt.tight_layout()
max_values
```

Out[32]:

```
array([[262, 143],
       [284, 143],
       [504,  76],
       [524,  76]], dtype=int64)
```

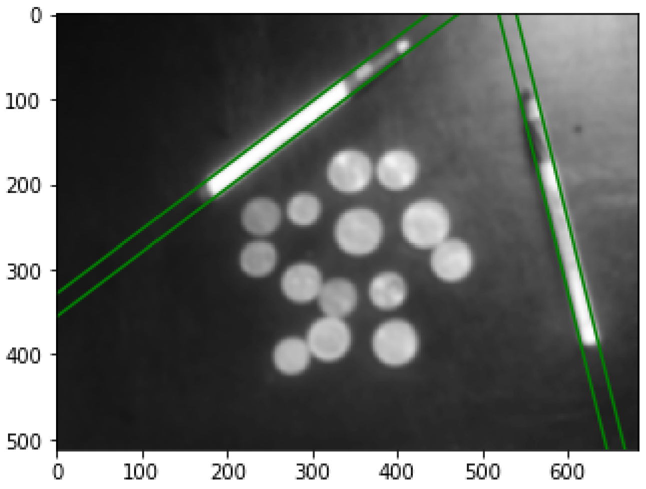


In [33]:

```
plt.figure()
for line in np.array(map(produce_line, [sampling_fix(peak, 1, 1) for peak in max_values
])):
    plt.plot(line[0:2], line[2:4], c = 'g')
plt.imshow(img1_blurred, cmap='gray', vmin = 0, vmax = 255)
```

Out[33]:

```
<matplotlib.image.AxesImage at 0x1b364ac8>
```



Like the first image, this one does not have a lot of noise (and most of it was suppressed thanks to the smoothing step). In this case, we can also rely on a tight binning (resolution of 1 degree and 1 unit in r). There were some lines that were close to the ones present (parallel and off by a few pixels), but it was simpler to change the threshold parameter to get the desired image, as the four lines plotted correspond to maxima in Hough space.
We would not rely on this approach when increasing the threshold makes important lines disappear (like in the previous example) in favor of the "noise" lines.

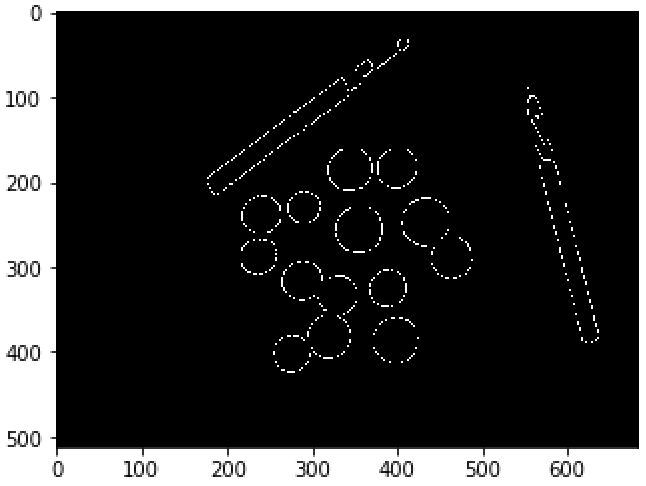
5. Circles

In [34]:

```
img1_edges_blurred = cv2.Canny(img1_blurred, 70, 100)
plt.imshow(img1_edges_blurred, cmap='gray', vmin = 0, vmax= 255)
```

Out[34]:

```
<matplotlib.image.AxesImage at 0x1d8f06d8>
```



We're generating a new edge image with lower thresholds for the Canny detector in order to better capture the edges of the coins (which is the main focus of this part).

In [35]:

```
def circles_accumulator(edges, r_min, r_max):
    edges_cord = np.argwhere(edges == 255)

    r_range = range(r_min, r_max + 1)
    a_max = np.abs(int(round(edges_cord[:,0].max() + r_max + 1)))
    b_max = np.abs(int(round(edges_cord[:,1].max() + r_max + 1)))
    H = np.zeros((a_max, b_max, len(r_range)))

    for e, edge in enumerate(edges_cord):
        #print(str(e*100./edges_cord.shape[0]) + '%')
        for r in r_range:
            for theta in range(0, 360):
                a = np.abs(int(round(edge[0] + r*math.sin(rad(theta)))))
                b = np.abs(int(round(edge[1] - r*math.cos(rad(theta)))))
                H[a,b,r - r_min] += 1

    return H
```

A new accumulator function was necessary.

In [36]:

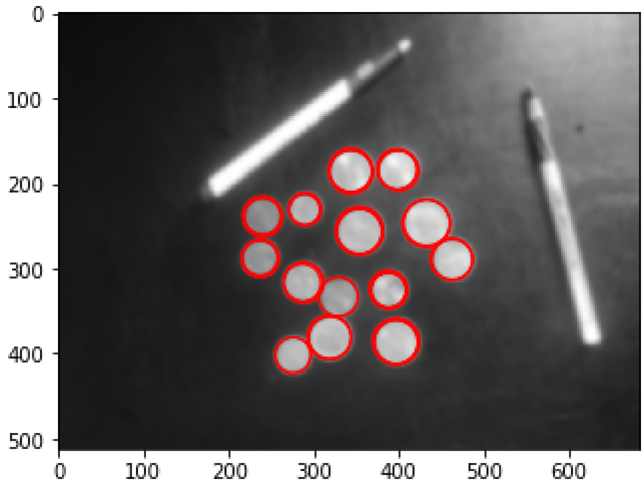
```
H = circles_accumulator(img1_edges_blurred, 15, 50)
```

In [37]:

```
max_values = np.argwhere(H > H.max() * 0.6)

plt.figure()
for circle in max_values:
    c = plt.Circle((circle[1], circle[0]), circle[2] + 15, color='r', fill = False)
    plt.gcf().gca().add_artist(c)
plt.imshow(img1_blurred, cmap='gray', vmin = 0, vmax = 255)
```

Out[37]:

```
<matplotlib.image.AxesImage at 0x1e166cc0>
```



The search for circles was performed using the single point method. There are three parameters in Hough space: a, b and r. The first two correspond to the location of a possible center in image space, while r is the radius of this circle. By having each edge (most are points on the contour of a circle) vote, we get cones in Hough space for possible circles and radii. For each circle in the image, since each "edge" point is voting for a candidate center, the intersections of the cones in Hough space correspond to the most likely center + radius in the image.

The parameters that were tested and changed for this part are the Canny detector threshold (so that the circles would be present in the edge image), the Hough accumulator threshold (`max_values = np.argwhere(H > H.max() * 0.6)`, 0.6 is our relative threshold to the highest intensity value of the Hough accumulator) and the range of possible radii in the image. By lowering the r_min parameter, we have some false positives at the ends of each pen, since they are also round. Increasing the minimum threshold (to circles of a minimum 15 px radius) therefore eliminates these false positives. The higher threshold r_max allows us to limit this (expensive) search in three dimensions for possible circles. The coins in the image only have an 18 to 28 radius, so limiting the search to a 50 pixel radius seemed reasonable.

6. Realistic images

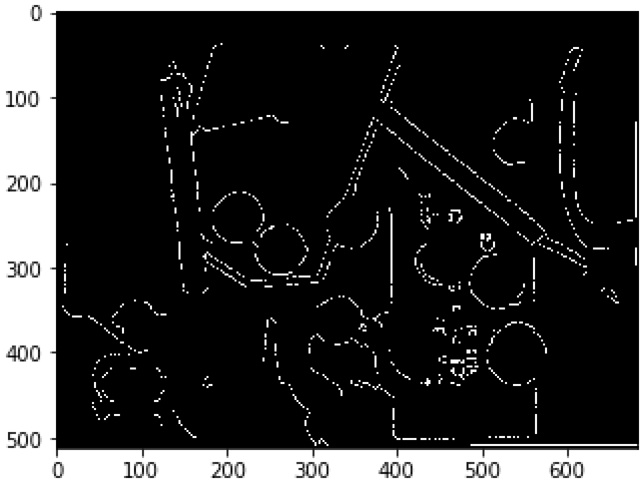
6.a. Lines (input 2)

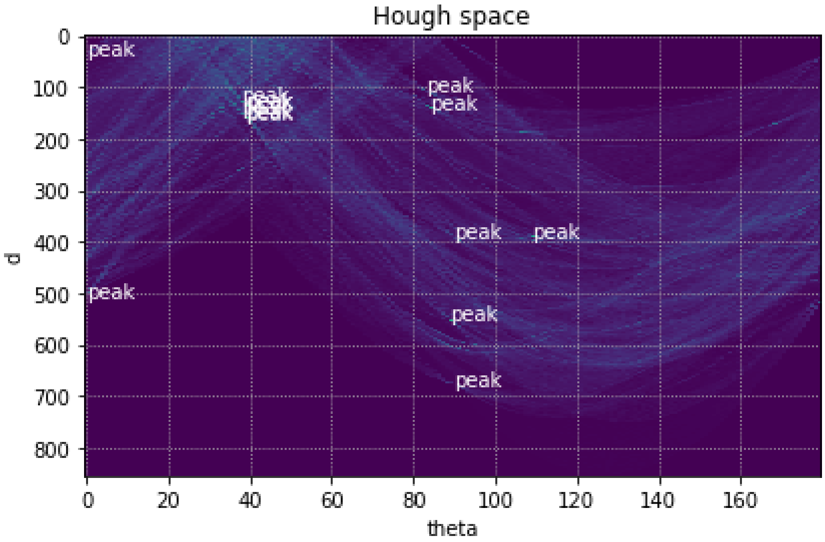
In [38]:

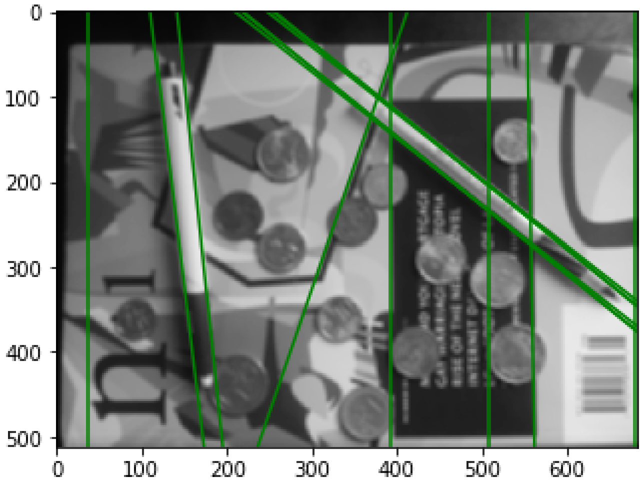
```
#imports for interactive jupyter
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

In [39]:

```
#Interact commented for report generation (no output shown)
#@interact(threshold = (0.0,1.,0.01), step_d = (1,20), step_theta = (1,20), \
#          canny_1 = (10,150,10), canny_2 = (50,300,10), \
#          gauss_sigma = (1,10), filter_size = (5,31,2))
def img2_pens(threshold = 0.53, step_d = 1, step_theta = 1, canny_1 = 60, canny_2 = 200
, gauss_sigma = 4, filter_size = 7):
    img2_pens = cv2.imread('./ps1-input2.jpg', cv2.IMREAD_GRAYSCALE)
    img2_blurred = cv2.GaussianBlur(img2_pens, (filter_size, filter_size), gauss_sigma,
 cv2.BORDER_REFLECT)

    theta_max = 180
    (y_max, x_max) = img2_pens.shape

    d_max = int(math.ceil(math.sqrt(y_max**2 + x_max**2)))

    img2_edges_blurred = cv2.Canny(img2_blurred, canny_1, canny_2)
    plt.imshow(img2_edges_blurred, cmap='gray', vmin = 0, vmax = 255)

    H = custom_accumulator(img2_edges_blurred, d_max, theta_max, step_d, step_theta)
    max_values = np.argwhere(H > H.max() * threshold)

    plt.figure()
    plt.imshow(H)
    for max in max_values:
        plt.annotate('peak', xy=(max[1], max[0]), color='white', fontsize=10)
    plt.grid(True, linestyle = ':')
    plt.axis('tight')
    plt.xlabel('theta')
    plt.ylabel('d')
    plt.title('Hough space')
    plt.tight_layout()
    plt.show()

    plt.figure()
    for line in np.array(map(produce_line, [sampling_fix(peak, step_d, step_theta) for
peak in max_values])):
        plt.plot(line[0:2], line[2:4], c = 'g')
    plt.imshow(img2_blurred, cmap='gray', vmin = 0, vmax = 255)

#Force function output with best parameters for report generation (no output shown othe
rwise)
img2_pens()
```







6.b. Issues with line detection

One of the issues was to select appropriate Canny parameters as well as smoothing parameters (filter size and gaussian σ in order for the left pen to be sufficiently present in the edge image. Even with the best parameters found, we can see that the laft pen's shape is not well refined. There were generally not any issues with the right pen. However if this step is not correctly done, then the results we get by using the Hough accumulator will be disappointing. Most of the time, it resulted in only either the left border of the left pen or its right border being correctly detected.

That said, the other constraint we have is to limit the number of false positives. To some extent, varying the Canny parameters had its importance to remove unnecessary lines in the edge image, but it was mostly due to the selection of an appropriate accumulator threshold (here only bins that had 53% as much votes as the most voted for bin were kept).

Regarding false positives, they are of several types:

- The edge of the book, on the right part of the image
- The left and right borders of the black part of the book (which is logical given the difference in color intensities between these two parts)
- A black diagonal line on the printing of the book (for the same reasons) which is especially well detected on the Canny edge image
- Two vertical lines due to prining on the book (the "n" and the text on the black patch)
- Parallel lines on the borders of the right pen. Using the same technique as with the 1st (noisy) image didn't work: while one could expect a looser binning (for d, increasing the `step_d` therefore decreasing the number of bins in the Hough accumulator) to merge the detected lines, it actually has the opposite effect in this particular image: non-maxima that were previously eliminated through thresholding end up merging in single bins and causing new lines to be detected. This can be seen on the Hough accumulator plot: there is a very large region around $[0, 170]_d \times [30, 50]_\theta$ where lines intersect but non-maxima have been eliminated. Merging values in this space results in more peaks being detected (and that are difficult to remove with higher thresholding without removing peaks corresponding to other line areas of the image).

6.c. Improving accuracy

While some operations have already been attempted to reduce false positives (including binning size for parallel lines and higher thresholding for the Hough accumulator), we will focus on other approaches to get the expected result.

In [40]:

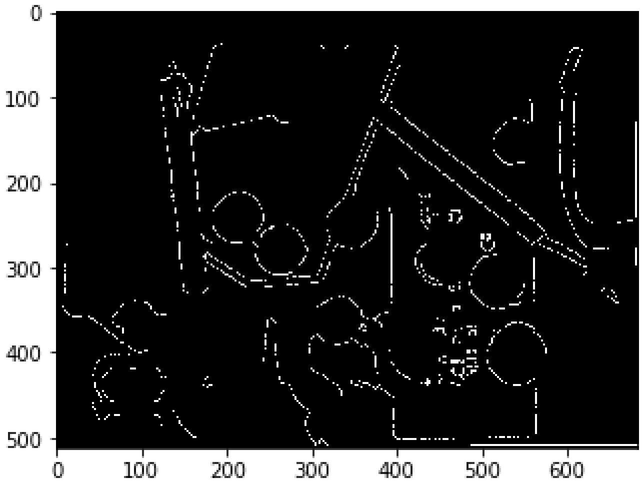
```
def max_filter(accumulator, d_area = 10, theta_area = 10):
    new_acc = accumulator.copy()
    (d_boundary, theta_boundary) = accumulator.shape
    for d in range(d_area, d_boundary - d_area + 1):
        for t in range(theta_area, theta_boundary - theta_area + 1):
            argmax = np.unravel_index(np.argmax(new_acc[d-d_area:d+d_area, t-theta_area
:t+theta_area]),(2*d_area,2*theta_area))
            max = new_acc[d-d_area + argmax[0], t-theta_area + argmax[1]]
            new_acc[d-d_area:d+d_area, t-theta_area:t+theta_area] = 0
            new_acc[d-d_area + argmax[0], t-theta_area + argmax[1]] = max
    return new_acc
```

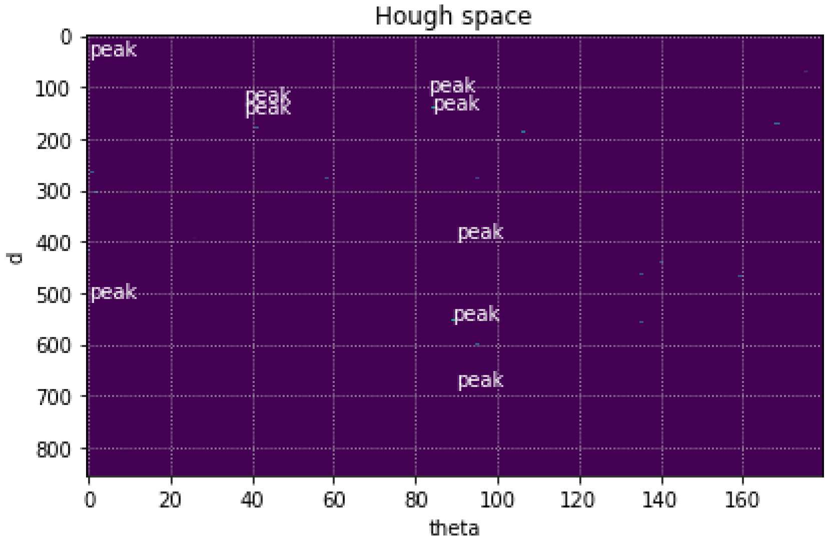
In [41]:

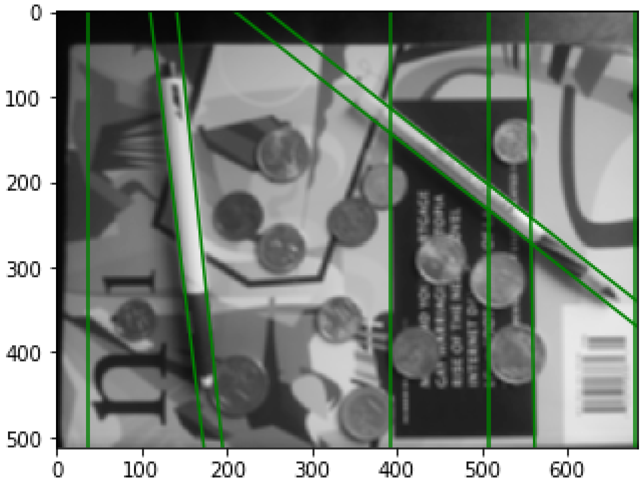
```
def neighborhood_coord(point, border):
    l = []
    for i in range(-1, 2):
        for j in range(-1, 2):
            if not ((i == j and j == 0) or point[0] + i < 0 or \
                    point[1] + j < 0 or point[0] + i >= border[0] or point[1] + j >= border[1]):
                l.append([point[0] + i, point[1] + j])
    return l

def minimum_length_2(edge_im):
    new_im = edge_im.copy()
    edges_cord = np.argwhere(new_im == 255)
    for edge in edges_cord:
        if np.max([new_im[tuple(index)] for index in neighborhood_coord(edge, new_im.shape)]) == 0:
            new_im[tuple(edge)] = 0
    return new_im
```

In [42]:

```
#Interact commented for report generation (no output shown)
#@interact(threshold = (0.0,1.,0.01), step_d = (1,20), step_theta = (1,20), \
#          canny_1 = (10,150,10), canny_2 = (50,300,10), \
#          gauss_sigma = (1,10), filter_size = (5,31,2))
def img2_pens(threshold = 0.53, step_d = 1, step_theta = 1, canny_1 = 60, canny_2 = 200
, gauss_sigma = 4, filter_size = 7):
    img2_pens = cv2.imread('./ps1-input2.jpg', cv2.IMREAD_GRAYSCALE)
    img2_blurred = cv2.GaussianBlur(img2_pens, (filter_size, filter_size), gauss_sigma,
 cv2.BORDER_REFLECT)

    theta_max = 180
    (y_max, x_max) = img2_pens.shape

    d_max = int(math.ceil(math.sqrt(y_max**2 + x_max**2)))

    img2_edges_blurred = cv2.Canny(img2_blurred, canny_1, canny_2)
    plt.imshow(img2_edges_blurred, cmap='gray', vmin = 0, vmax = 255)

    img2_edges_blurred = minimum_length_2(img2_edges_blurred) #remove isolated points

    H = custom_accumulator(img2_edges_blurred, d_max, theta_max, step_d, step_theta)
    H = max_filter(H, 10, 10) #suppressing local non-maxima
    max_values = np.argwhere(H > H.max() * threshold)

    plt.figure()
    plt.imshow(H)
    for max in max_values:
        plt.annotate('peak', xy=(max[1], max[0]), color='white', fontsize=10)
    plt.grid(True, linestyle = ':')
    plt.axis('tight')
    plt.xlabel('theta')
    plt.ylabel('d')
    plt.title('Hough space')
    plt.tight_layout()
    plt.show()

    plt.figure()
    for line in np.array(map(produce_line, [sampling_fix(peak, step_d, step_theta) for
peak in max_values])):
        plt.plot(line[0:2], line[2:4], c = 'g')
    plt.imshow(img2_blurred, cmap='gray', vmin = 0, vmax = 255)

#Force function output with best parameters for report generation (no output shown othe
rwise)
img2_pens()
```







Following the three possible operations, we get a better image the (almost-parallel) lines along the right pen were successfully removed using local non-maxima suppression (`max_filter`) and `minimum_length_2` removes any edge point from the Canny edge image that is not linked to another in its neighborhood (left, right, up, down or diagonally).

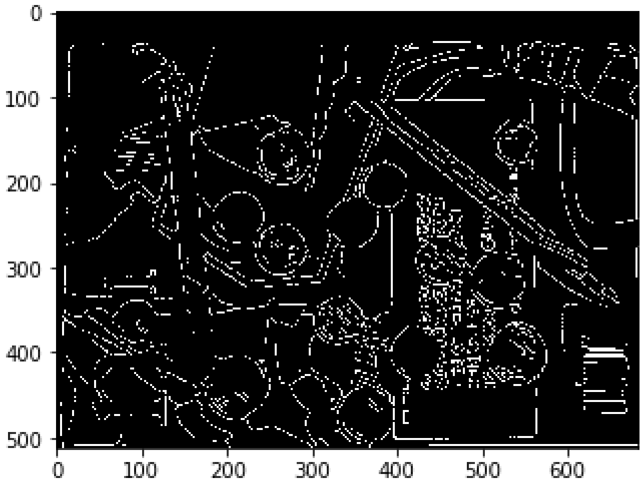
This second function did not have any influence in this case however as the chosen Canny parameters were already such that no such isolated edge existed. Increasing the minimum length (from 2 to n) probably wouldn't have helped in this case: there are clear lines on the edge image (hundreds of pixels long) for the black patch and the boders of the book.

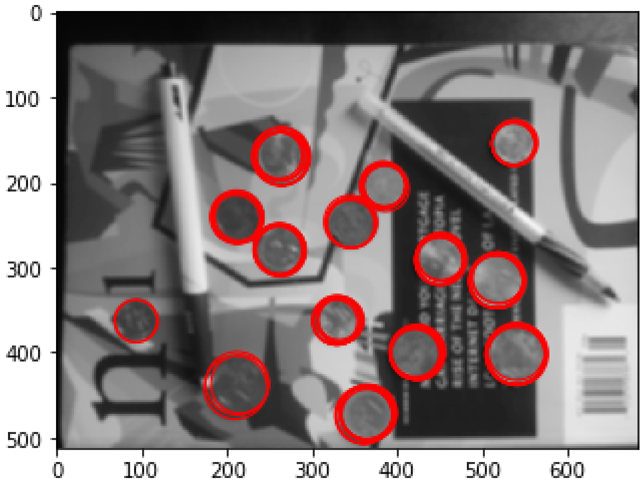
7. Circles (input 2)

7.a. Smoothing & finding

In [43]:

```
#Interact commented for report generation (no output shown)
#@interact(threshold = (0.0,1.,0.01), r_min = (1,30), r_max = (5,70), \
#          canny_1 = (10,150,10), canny_2 = (50,300,10), \
#          gauss_sigma = (1,10), filter_size = (5,31,2))
def img2_coins(threshold = 0.67, r_min = 25, r_max = 45, canny_1 = 40, canny_2 = 60, ga
uss_sigma = 3, filter_size = 7):
    start = timer()

    img2_coins = cv2.imread('./ps1-input2.jpg', cv2.IMREAD_GRAYSCALE)
    img2_blurred = cv2.GaussianBlur(img2_coins, (filter_size,filter_size), gauss_sigma,
 cv2.BORDER_REFLECT)

    theta_max = 360
    (y_max, x_max) = img2_coins.shape

    d_max = int(math.ceil(math.sqrt(y_max**2 + x_max**2)))

    img2_edges_blurred = cv2.Canny(img2_blurred, canny_1, canny_2)
    plt.imshow(img2_edges_blurred, cmap='gray', vmin = 0, vmax = 255)

    H = circles_accumulator(img2_edges_blurred, r_min, r_max)

    max_values = np.argwhere(H > H.max() * threshold)

    plt.figure()
    for circle in max_values:
        c = plt.Circle((circle[1], circle[0]), circle[2] + r_min, color='r', fill = Fal
se)
        plt.gcf().gca().add_artist(c)
    plt.imshow(img2_blurred, cmap='gray', vmin = 0, vmax = 255)

    end = timer()
    #print(end-start)

#Force function output with best parameters for report generation (no output shown othe
rwise)
img2_coins()
```





7.b. False positives

Just like with the line case, the whole detection process was encapsulated in a function that is used with Jupyter's interactive widgets capabilities. The parameters that were used to determine the optimal detector finding all the coins were the following:

- `filter_size` and `gauss_sigma`: the size of the smoothing filter and corresponding σ gaussian blur
- `canny_1` and `canny_2`: the two thresholds for Canny edge detection
- `r_min` and `r_max`: the interval defining the radii that candidate circles might have
- `threshold`: a relative threshold to select peaks in the Hough accumulator, as a proportion of the most voted for bin of the accumulator

The most crucial false positives were eliminated through trial and error for different combinations of all these parameters. Notably:

- A circle was previously found inside the "n" shape of the book
- A circle was found between multiple coins: as the text in the black patch of the book accounted for quite a few edges, a cricle was found between the lower three coins of the patch. It was supported by the edges from the coins and from the text, linking edges between the coins
  These false alarms were eliminated with better thresholding.

The result shown here is not perfect: there is a lot of overlap in the coins. Because of the smoothing and Canny parameters chosen, the imprint on the coins as well as the book background (shapes, text seen on the edge image) sometimes cause multiple circles to be found very close to each other with centers and radii varying only by a few pixels. This result was kept because the small coin on the left is very hard to capture and increasing detection thresholds would result in losing its detected circle. This is the best result that was found when looking for **all** of the coins, but smoother results ignoring that particular coin gives us much better accuracy for the other coins (less overlap and pixel shifts).

A possible solution would be to implement posterior non-maxima local suppression (like the `max_filter` for lines):

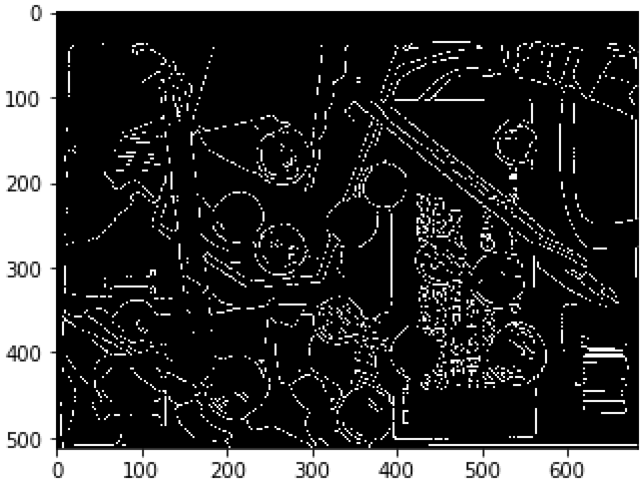
- if two circles around the same values of (a,b) have a similar r then we could manually null (setting the accumulator's bins to 0) the ones that are non-maxima (only in each neighbourhood).
- the same procedure could be applied for small shifts in circle centers (i.e. a and/or b off by a few units) where close bins all have a high number of votes.

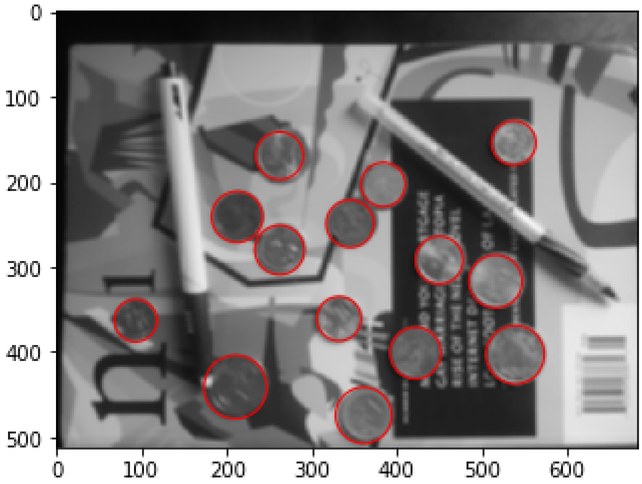
In [44]:

```
def max_filter_circle(accumulator, a_area = 10, b_area = 10, r_area = 10):
    new_acc = accumulator.copy()
    (a_boundary, b_boundary, r_boundary) = accumulator.shape
    for a in range(a_area, a_boundary - a_area + 1):
        for b in range(b_area, b_boundary - b_area + 1):
            for r in range(r_area, r_boundary - r_area + 1):
                argmax = np.unravel_index(np.argmax( \
                    new_acc[a-a_area:a+a_area, b-b_area:b+b_area, r-r_area:r+r_area]),(2*a_area,2*b_area,2*r_area))
                max = new_acc[a-a_area + argmax[0], b-b_area + argmax[1], r-r_area + argmax[2]]
                new_acc[a-a_area:a+a_area, b-b_area:b+b_area, r-r_area:r+r_area] = 0
                new_acc[a-a_area + argmax[0], b-b_area + argmax[1], r-r_area + argmax[2]] = max
    return new_acc
```

In [45]:

```
#Interact commented for report generation (no output shown)
#@interact(threshold = (0.0,1.,0.01), r_min = (1,30), r_max = (5,70), \
#          canny_1 = (10,150,10), canny_2 = (50,300,10), \
#          gauss_sigma = (1,10), filter_size = (5,31,2))
def img2_coins(threshold = 0.67, r_min = 25, r_max = 45, canny_1 = 40, canny_2 = 60, gauss_sigma = 3, filter_size = 7):
    start = timer()

    img2_coins = cv2.imread('./ps1-input2.jpg', cv2.IMREAD_GRAYSCALE)
    img2_blurred = cv2.GaussianBlur(img2_coins, (filter_size,filter_size), gauss_sigma, cv2.BORDER_REFLECT)

    theta_max = 360
    (y_max, x_max) = img2_coins.shape

    d_max = int(math.ceil(math.sqrt(y_max**2 + x_max**2)))

    img2_edges_blurred = cv2.Canny(img2_blurred, canny_1, canny_2)
    plt.imshow(img2_edges_blurred, cmap='gray', vmin = 0, vmax = 255)

    H = circles_accumulator(img2_edges_blurred, r_min, r_max)

    H = max_filter_circle(H,10,10,10)

    max_values = np.argwhere(H > H.max() * threshold)

    plt.figure()
    for circle in max_values:
        c = plt.Circle((circle[1], circle[0]), circle[2] + r_min, color='r', fill = False)
        plt.gcf().gca().add_artist(c)
    plt.imshow(img2_blurred, cmap='gray', vmin = 0, vmax = 255)

    end = timer()
    #print(end-start)

#Force function output with best parameters for report generation (no output shown otherwise)
img2_coins()
```



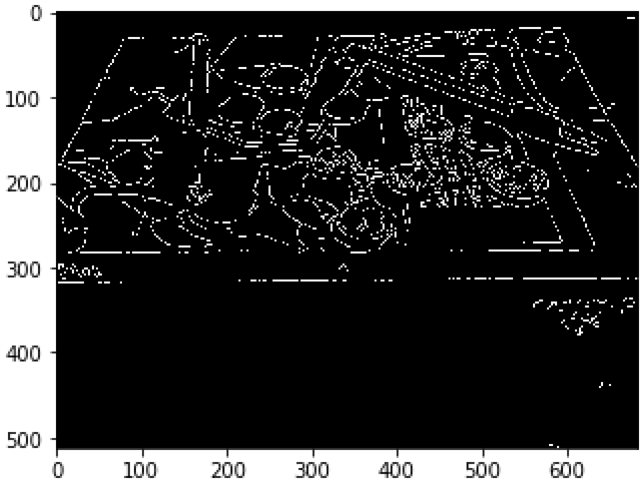


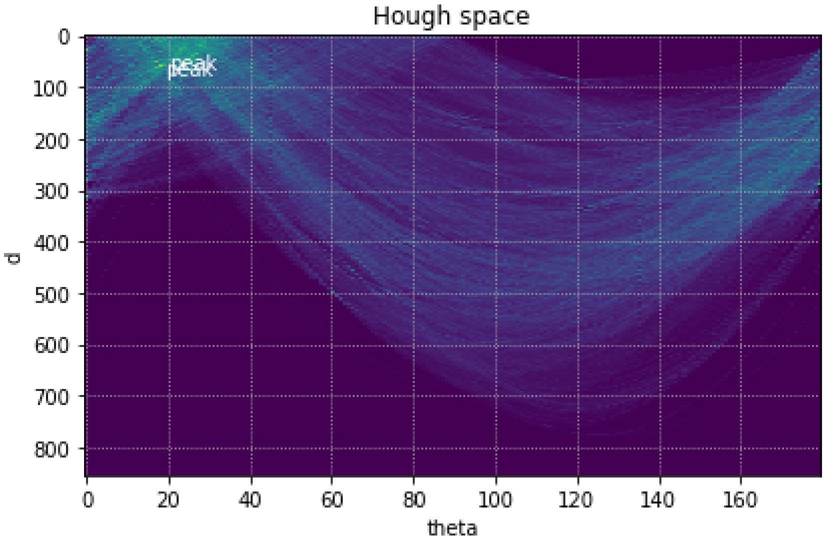
8. Sensitivity to distortion

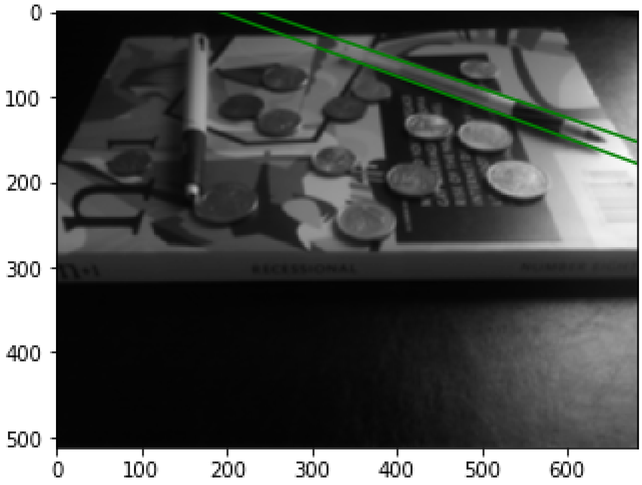
8.a. Line and circle finders on the distorted image (input 3)

In [46]:

```
#Interact commented for report generation (no output shown)
#@interact(threshold = (0.0,1.,0.01), step_d = (1,20), step_theta = (1,20), \
#          canny_1 = (10,150,10), canny_2 = (50,300,10), \
#          gauss_sigma = (1,10), filter_size = (5,31,2))
def img3_pens(threshold = 0.84, step_d = 1, step_theta = 1, canny_1 = 20, canny_2 = 50,
 gauss_sigma = 2, filter_size = 7):
    img3_pens = cv2.imread('./ps1-input3.jpg', cv2.IMREAD_GRAYSCALE)
    img3_blurred = cv2.GaussianBlur(img3_pens, (filter_size, filter_size), gauss_sigma,
 cv2.BORDER_REFLECT)

    theta_max = 180
    (y_max, x_max) = img3_pens.shape

    d_max = int(math.ceil(math.sqrt(y_max**2 + x_max**2)))

    img3_edges_blurred = cv2.Canny(img3_blurred, canny_1, canny_2)
    plt.imshow(img3_edges_blurred, cmap='gray', vmin = 0, vmax = 255)

    H = custom_accumulator(img3_edges_blurred, d_max, theta_max, step_d, step_theta)
    max_values = np.argwhere(H > H.max() * threshold)

    plt.figure()
    plt.imshow(H)
    for max in max_values:
        plt.annotate('peak', xy=(max[1], max[0]), color='white', fontsize=10)
    plt.grid(True, linestyle = ':')
    plt.axis('tight')
    plt.xlabel('theta')
    plt.ylabel('d')
    plt.title('Hough space')
    plt.tight_layout()
    plt.show()

    plt.figure()
    for line in np.array(map(produce_line, [sampling_fix(peak, step_d, step_theta) for
peak in max_values])):
        plt.plot(line[0:2], line[2:4], c = 'g')
    plt.imshow(img3_blurred, cmap='gray', vmin = 0, vmax = 255)

#Force function output with best parameters for report generation (no output shown othe
rwise)
img3_pens()
```





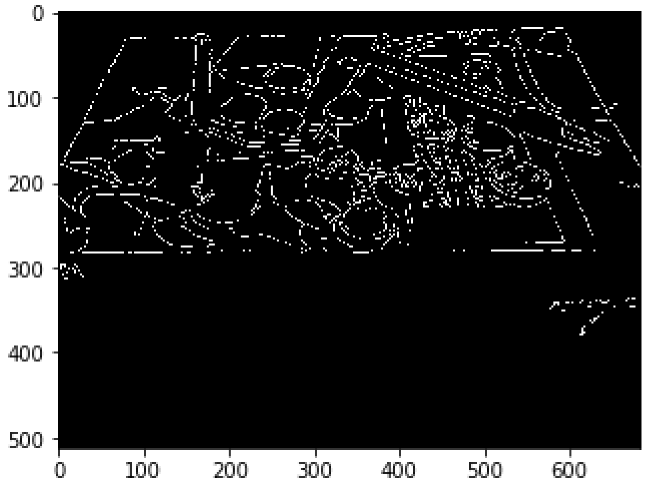


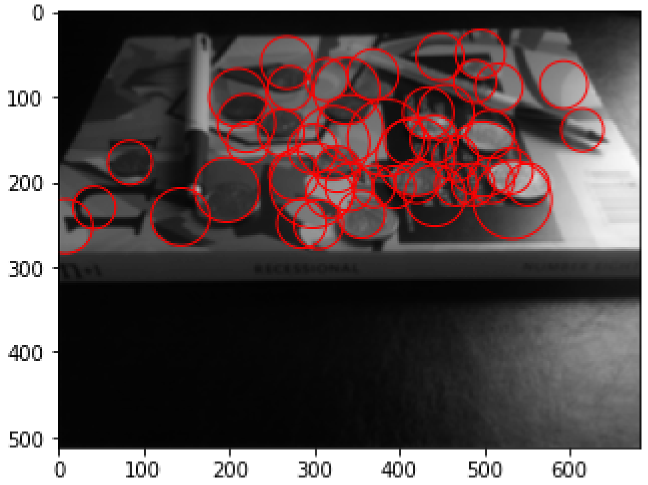
The left pen is particularly difficult to capture, as with the perpective projection, it's contours aren't well laid out in lines. As it can be seen in the edge image, the borders have segments of small straight vertical lines in groups that are each off by a few pixels (instead of a clear, diagonal line). As such, the borders are seen as multiple independent lines for each segment due to perspective.

While it is ultimately possible to retrieve the lines of the contour of the left pen, they will only be supported by a few edges, which means that we would have to lower the `threshold` to about .35 to accept these few votes, resulting in an end image with a lot of false positives as well. A choice has been made to show a clearer image, with only the detection of the right pen.

In [47]:

```
#Interact commented for report generation (no output shown)
#@interact(threshold = (0.0,1.,0.01), r_min = (1,30), r_max = (5,70), \
#          canny_1 = (10,150,10), canny_2 = (50,300,10), \
#          gauss_sigma = (1,10), filter_size = (5,31,2))
def img3_coins(threshold = 0.67, r_min = 25, r_max = 50, canny_1 = 20, canny_2 = 60, gauss_sigma = 2, filter_size = 7):
    start = timer()

    img3_coins = cv2.imread('./ps1-input3.jpg', cv2.IMREAD_GRAYSCALE)
    img3_coins_blurred = cv2.GaussianBlur(img3_coins, (filter_size,filter_size), gauss_sigma, cv2.BORDER_REFLECT)

    theta_max = 360
    (y_max, x_max) = img3_coins.shape

    d_max = int(math.ceil(math.sqrt(y_max**2 + x_max**2)))

    img3_coins_edges_blurred = cv2.Canny(img3_coins_blurred, canny_1, canny_2)
    plt.imshow(img3_coins_edges_blurred, cmap='gray', vmin = 0, vmax = 255)

    H = circles_accumulator(img3_coins_edges_blurred, r_min, r_max)

    H = max_filter_circle(H,10,10,10)

    max_values = np.argwhere(H > H.max() * threshold)

    plt.figure()
    for circle in max_values:
        c = plt.Circle((circle[1], circle[0]), circle[2] + r_min, color='r', fill = False)
        plt.gcf().gca().add_artist(c)
    plt.imshow(img3_coins_blurred, cmap='gray', vmin = 0, vmax = 255)

    end = timer()
    #print(end-start)

#Force function output with best parameters for report generation (no output shown otherwise)
img3_coins()
```





8.b. How to fix the circle problem

The issue with trying to find circles on the image is that the coins won't be detected as circles (because they're not). Since the picture was taken from the side with an angle instead of from the top, coins appear as ellipses (perspective). This can be easily checked with any image processing software: the coins are wider than they are tall. So the use of an accumulator relying on votes for a center from edges equidistant to it makes no sense: the edges of the coins are from varying distances to their center.

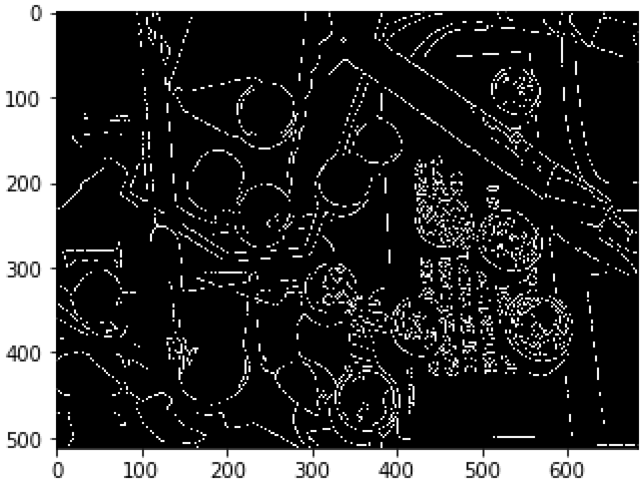
There are two approaches that we could imagine to fix the circle problem:

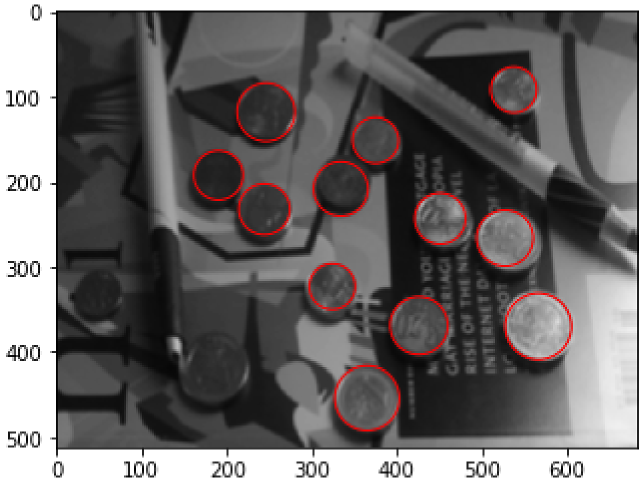
- Try to "undistort" the image and re-establish scale where coins appear as circles
- Change the accumulator algorithm so that is can recognize ellipses instead of circles.
  While the equation of circle is: $(x-a)^2 + (y-b)^2 = r^2$ (where we have been looking for a, b, r in Hough space), that of an ellipse is $\frac{(x-a)^2}{r_1^2} + \frac{(y-b)^2}{r_2^2} = 1$

8.c. Fixing the circle problem

In [48]:

```
#Interact commented for report generation (no output shown)
#@interact(threshold = (0.0,1.,0.01), r_min = (1,30), r_max = (5,70), \
#          canny_1 = (10,150,10), canny_2 = (50,300,10), \
#          gauss_sigma = (1,10), filter_size = (5,31,2))
def img3_coins_ps(threshold = 0.67, r_min = 25, r_max = 50, canny_1 = 20, canny_2 = 60,
 gauss_sigma = 2, filter_size = 7):
    start = timer()

    img3_coins_ps = cv2.imread('./ps1-input3-ps.jpg', cv2.IMREAD_GRAYSCALE)
    plt.imshow(img3_coins_ps, cmap='gray', vmin = 0, vmax = 255)
    img3_coins_ps_blurred = cv2.GaussianBlur(img3_coins_ps, (filter_size,filter_size),
gauss_sigma, cv2.BORDER_REFLECT)

    theta_max = 360
    (y_max, x_max) = img3_coins_ps.shape

    d_max = int(math.ceil(math.sqrt(y_max**2 + x_max**2)))

    img3_coins_ps_edges_blurred = cv2.Canny(img3_coins_ps_blurred, canny_1, canny_2)
    plt.imshow(img3_coins_ps_edges_blurred, cmap='gray', vmin = 0, vmax = 255)

    H = circles_accumulator(img3_coins_ps_edges_blurred, r_min, r_max)

    H = max_filter_circle(H,10,10,10)

    max_values = np.argwhere(H > H.max() * threshold)

    plt.figure()
    for circle in max_values:
        c = plt.Circle((circle[1], circle[0]), circle[2] + r_min, color='r', fill = Fal
se)
        plt.gcf().gca().add_artist(c)
    plt.imshow(img3_coins_ps_blurred, cmap='gray', vmin = 0, vmax = 255)

    end = timer()
    #print(end-start)

#Force function output with best parameters for report generation (no output shown othe
rwise)
img3_coins_ps()
```





A very crude edit on an image editing software to try to distort and "undo" perspective gives us this result.

Now, looking at the ellipses accumulator.

Knowing that the equation of an ellipse is $\frac{(x-a)^2}{r_1^2} + \frac{(y-b)^2}{r_2^2} = 1$, we can construct two equations for a and b similar to those found in the case of the circle:

$a = x - r_1 cos(\theta)$ and $b = y + r_2 sin(\theta)$

In [49]:

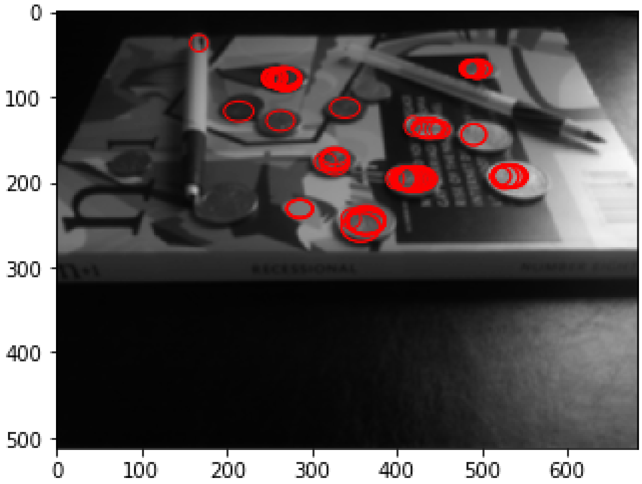
```python
def ellipses_accumulator(edges, r_min, r_max):
    edges_cord = np.argwhere(edges == 255)

    r_range = range(r_min, r_max + 1)
    a_max = np.abs(int(round(edges_cord[:,0].max() + r_max + 1)))
    b_max = np.abs(int(round(edges_cord[:,1].max() + r_max + 1)))
    H = np.zeros((a_max, b_max, len(r_range), len(r_range)))

    for e, edge in enumerate(edges_cord):
        for r1 in r_range:
            for r2 in r_range:
                for theta in range(0, 360):
                    a = np.abs(int(round(edge[0] + r2*math.sin(rad(theta)))))
                    b = np.abs(int(round(edge[1] - r1*math.cos(rad(theta)))))
                    H[a,b,r1 - r_min,r2 - r_min] += 1

    return H
```

In [50]:

```python
#as the computation time for this part was already very long, no optimization here.
img3_coins = cv2.imread('./ps1-input3.jpg', cv2.IMREAD_GRAYSCALE)
img3_coins_blurred = cv2.GaussianBlur(img3_coins, (7,7), 2, cv2.BORDER_REFLECT)

theta_max = 360
(y_max, x_max) = img3_coins.shape

d_max = int(math.ceil(math.sqrt(y_max**2 + x_max**2)))

img3_coins_edges_blurred = cv2.Canny(img3_coins_blurred, 20, 50)
plt.imshow(img3_coins_edges_blurred, cmap='gray', vmin = 0, vmax = 255)

H = ellipses_accumulator(img3_coins_edges_blurred, 10, 40)
```

In [51]:

```
#Interact commented for report generation (no output shown)
#@interact(threshold = (0.0,1.,0.01))
def img3_coins_2(threshold = 0.77): #threshold only, too long to compute
    max_values = np.argwhere(H > H.max() * threshold)

    plt.figure()
    for circle in max_values:
        e = Ellipse((circle[1], circle[0]), circle[2] + 20, circle[3] + 20, color='r',
fill = False)
        plt.gcf().gca().add_artist(e)
    plt.imshow(img3_coins_blurred, cmap='gray', vmin = 0, vmax = 255)

#Force function output with best parameters for report generation (no output shown othe
rwise)
img3_coins_2()
```



While still getting an idea of where most coins are, this algorithm is terribly inefficient. Using the single point method for circles wasn't especially computationally easy, it remained reasonable (about 15 minutes on a laptop). The "naive" algorithm implemented here actually adds complexity to the problem with an extra dimension to vote in (and computations are now made in the order of several hours).

Optimized versions of the Hough transform exist (reducing the complexity of the search):
Yonghong Xie and Qiang Ji, "A new efficient ellipse detection method," Object recognition supported by user interaction for service robots, Quebec City, Quebec, Canada, 2002, pp. 957-960 vol.2. doi: 10.1109/ICPR.2002.1048464

However, a more fundamental problem lies with the perspective of the image. For example, the bottom right coin, due to being on the sides, is more affected by perspective. It appears slightly distorted, with a larger right side (ressembling an egg).