2.1

$$\rho = x \cdot cos\theta + y \cdot sin\theta$$

$$\rho = (\frac{x}{\sqrt{x^2 + y^2}} \cdot cos\theta + \frac{y}{\sqrt{x^2 + y^2}} \cdot sin\theta) \cdot \sqrt{x^2 + y^2}$$

Assume that $sin\varphi = \frac{x}{\sqrt{x^2+y^2}}$ and $cos\varphi = \frac{y}{\sqrt{x^2+y^2}}$

$$\rho = (sin\varphi \cdot cos\theta + cos\varphi \cdot sin\theta) \cdot \sqrt{x^2 + y^2}$$

$$\rho = \sqrt{x^2 + y^2} \cdot \sin(\varphi + \theta)$$

2.2

The first reason is that when the line is vertical, the slope is meaningless and will cause a bug in the computation.

The second reason is that Accumulator Array A(m,c) will be very large if the slope is large, which is computation-expensive and we can avoid this by using ρ and θ.

2.3

The range of θ should be (0,2π] and the range of ρ should be [0, $\sqrt{H^2 + W^2}$] if we neglect the negative value of ρ.

2.4

```
% Plot Corresponding Lines in Hough Space
x = [10 15 30];
y = [10 15 30];

for i = 1:3
    theta = 0:pi/180:2*pi;
    rho = sqrt(x(i)^2 + y(i)^2) * sin(theta + atan(x(i)/y(i)));
    plot(theta,rho);
    hold on;
end
```

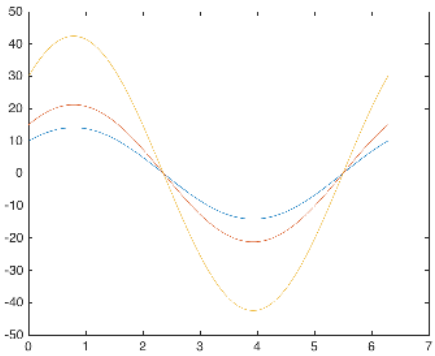


Figure X. Three Lines in Hough Space for (10,10), (15,15) and (30,30)

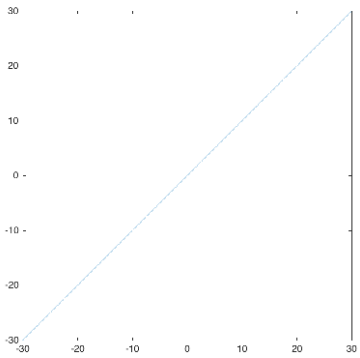


Figure X. Corresponding Line in Original Space for intersection $(0,\frac{3\pi}{4})$

2.5
The higher the dimension of parameter space, the more computational resource it will need.

For example, if we want to find circles in the original x-y space, we need an a-b-r three dimensional parameter space to describe a circle. Then each point in original space will be projected to a circular cone in the 3D space. The complexity increases extremely.

What I will do is to reduce the resolution of parameters so that it may not take so long to get the result. For example, we use rhoRes = 2 and thetaRes = 90/pi as our original resolution. Then maybe I will use aRes = 4, bRes = 4, rRes = 5 to reduce the calculation.

Or we can use simpler/similar/approximate models to reduce the dimension of parameter space.

3.1

```
function [img1] = myImageFilter(img0, hm)

    % Flip the image
    hm = fliplr(hm);
    hm = flipud(hm);

    %Get the size of the filter and the original image
    [h, w] = size(hm);
    [H, W] = size(img0);
    img1 = zeros([H,W]);

    % Construct expand matrix
    a11 = ones((h-1)/2,(w-1)/2) * img0(1,1);
    a13 = ones((h-1)/2,(w-1)/2) * img0(1,W);
    a31 = ones((h-1)/2,(w-1)/2) * img0(H,1);
    a33 = ones((h-1)/2,(w-1)/2) * img0(H,W);
    a12 = repmat(img0(1,:),(h-1)/2,1);
    a32 = repmat(img0(H,:),(h-1)/2,1);
    a21 = repmat(img0(:,1),1,(w-1)/2);
    a23 = repmat(img0(:,W),1,(w-1)/2);

    img_expand = [a11 a12 a13;
                  a21 img0 a23;
                  a31 a32 a33];

    % For each pixel, calculate the filter
    for i = (h+1)/2:H+(h-1)/2
       for j = (w+1)/2:W+(w-1)/2
           img1(i-(h-1)/2,j-(w-1)/2) = sum(sum(img_expand(i-(h-1)/2:i-(h-
1)/2+h-1,j-(w-1)/2:j-(w-1)/2+w-1).*double(hm)));
       end
    end

end
```

I replaced the input ‘h’ with ‘hm’ since I use h as the height for filter.

3.2

```
    % For each pixel, calculate the filter
  for i = 1:H*W
       row = 1 + floor((i-1)/W); % coordinates in original matrix
       col = mod(i,W);
       if col == 0
           col = W;
       end
       eRow = row + (h-1)/2; % coordinates in expanded matrix
       eCol = col + (w-1)/2;
       img1(row,col) = sum(sum(img_expand(eRow-(h-1)/2:eRow-(h-1)/2+h-
1,eCol-(w-1)/2:eCol-(w-1)/2+w-1).*double(hm)));
    end
```

This is the different part from 'myImageFilter'. First I calculated 'row' and 'col', the indexes of original matrix, using only one variable 'i'. After that, I transform 'i' into expanded indexes 'eRow' and 'eCol'.

3.3 Edge Detection

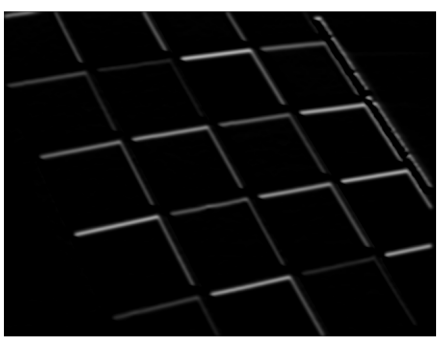
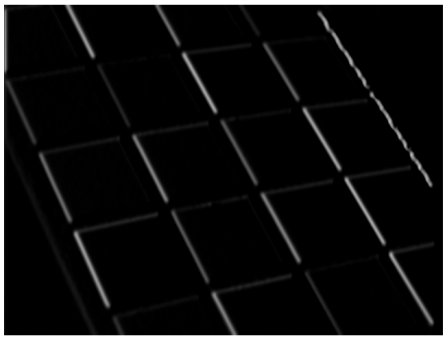



Figure X. Ix and Iy before NMS

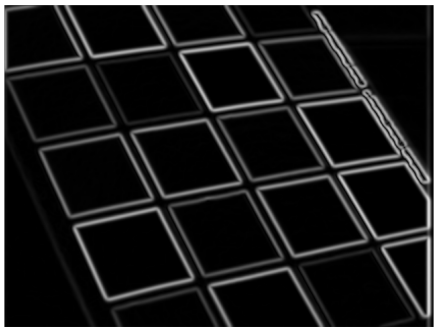
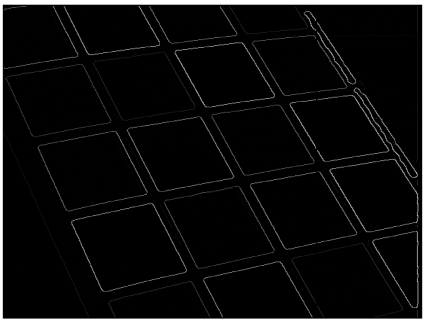



Figure X. Im before NMS and Im after NMS

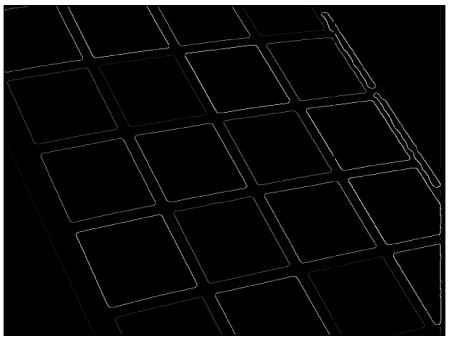
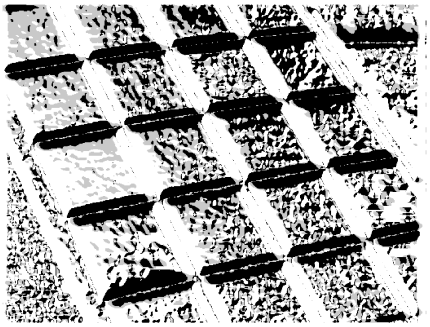



Figure X. Output Im and Io

3.4 Hough Transformation

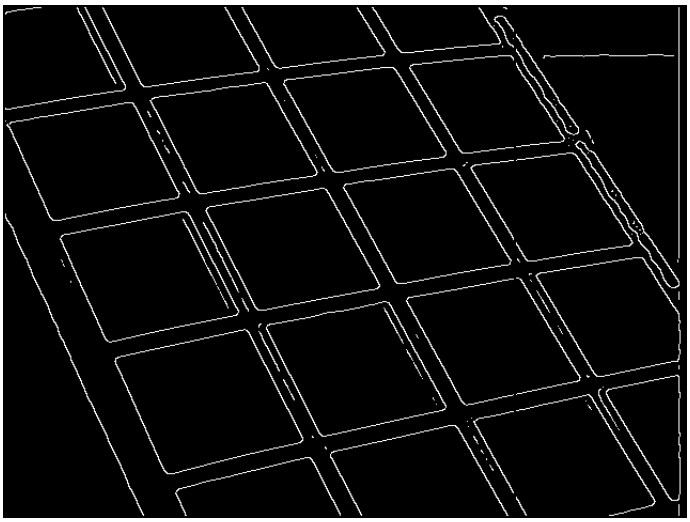


Figure X. Threshold Image

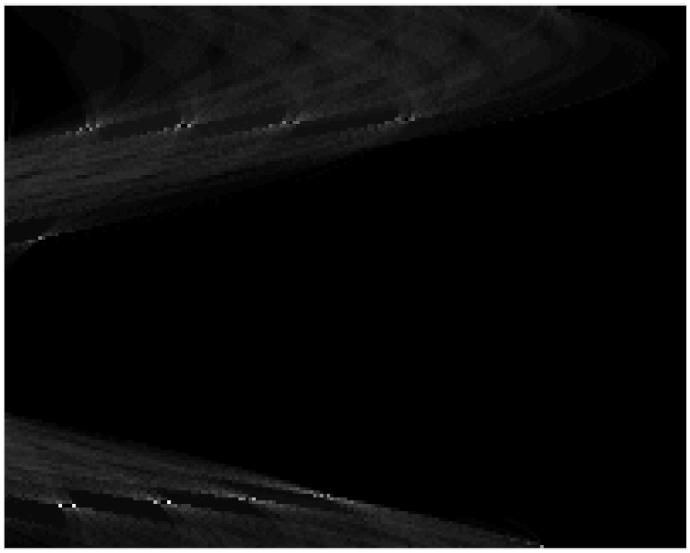


Figure X. Hough Space

3.5 Finding Lines

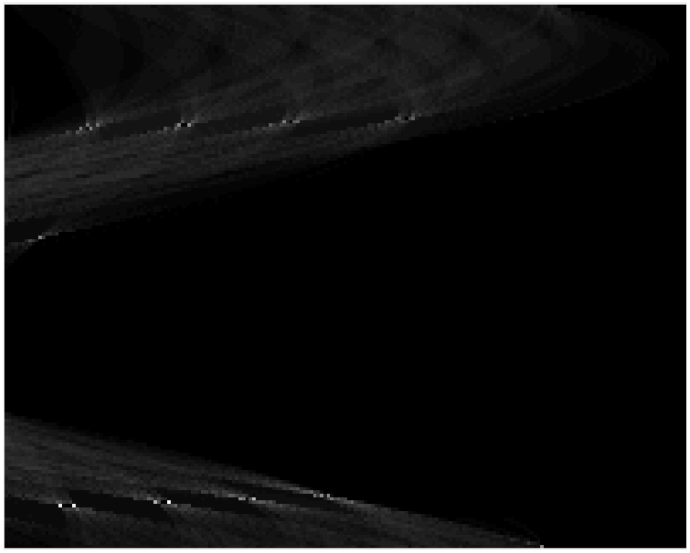


Figure X. Hough Space

By applying NMS and find the peak values and their indexes, I get two arrays rhos and thetas. Then I can visualize them using houghLines.

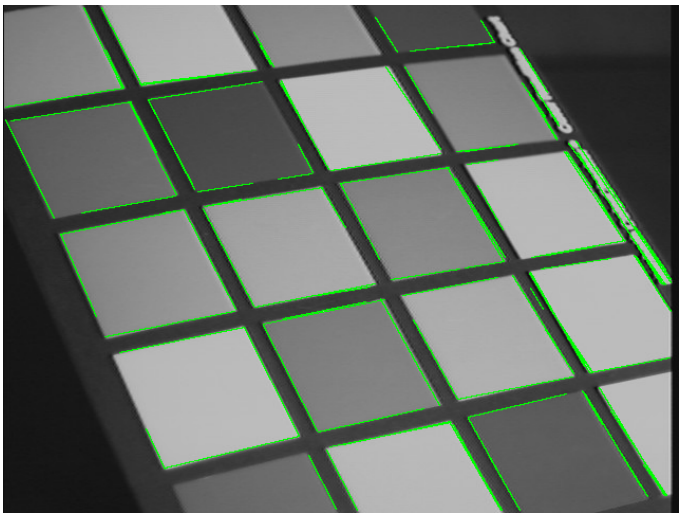


Figure X. Visualization

3.6 Fitting Line Segments for Visualization

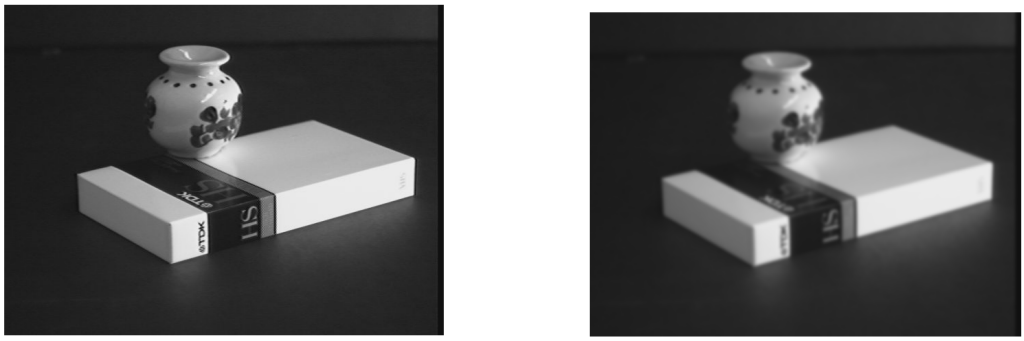


Figure X. Original Image and Image after Gaussian

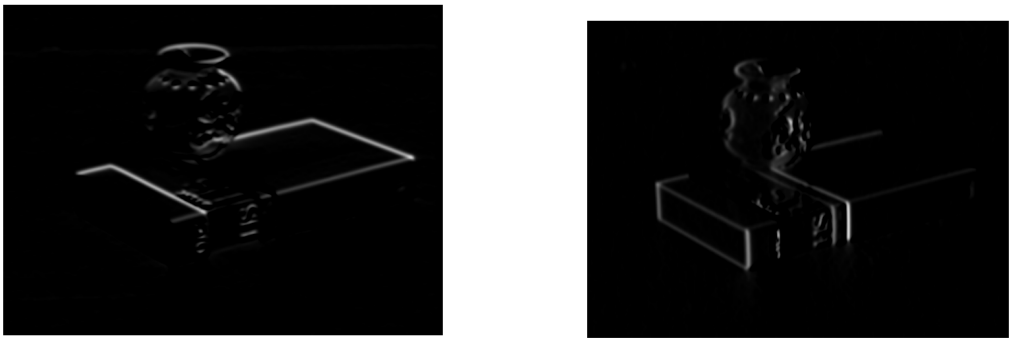


Figure X. Ix and Iy before NMS

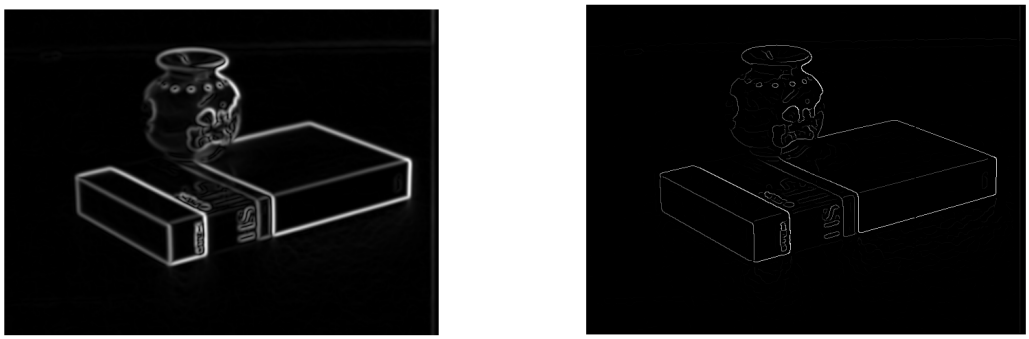


Figure X. Im before NMS and Im after NMS

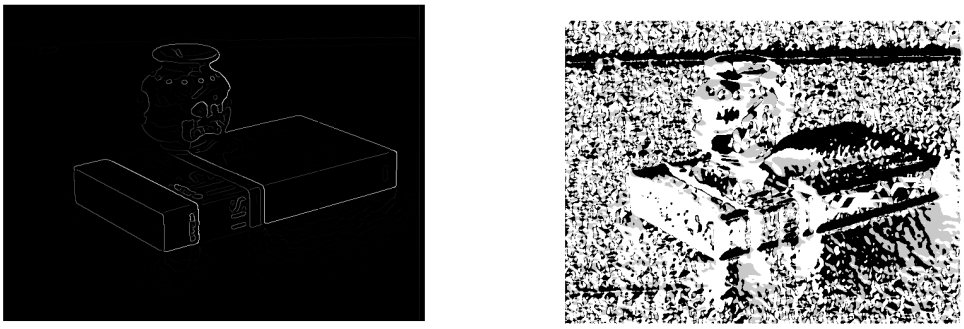
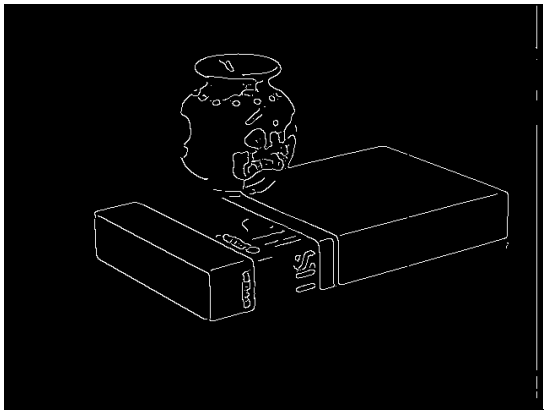


Figure X. Output Im and Io



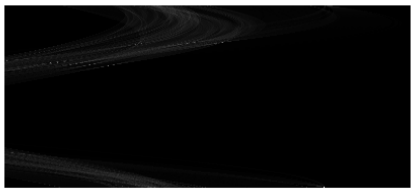


Figure X. Threshold Image and Hough Space

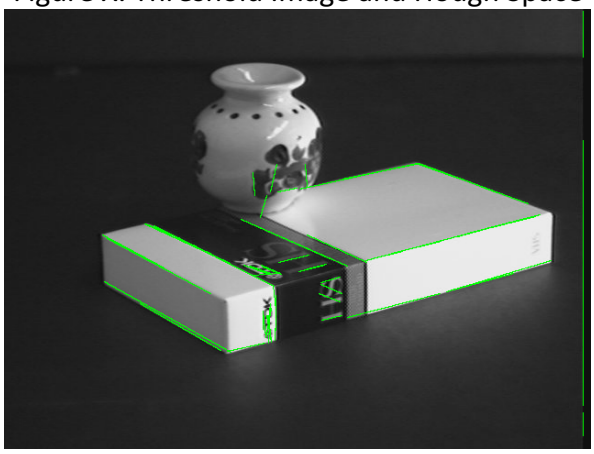


Figure X. Visualization

4. Experiments

My code doesn't work well on all of the images with a single set of parameters.
Each parameter will make a difference in finding lines.

I found that how I choose the origin of x-y coordinate actually play an important role.

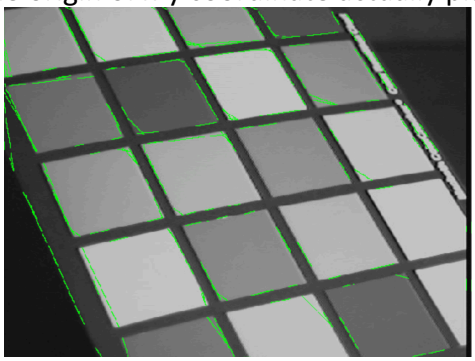


Figure X. Output before change the coordinate

This is something I got when I set the origin to the middle of the image.
And below is what I got with origin at left-top corner.

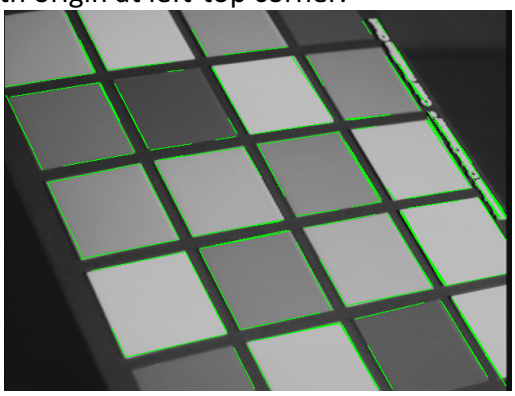


Figure X. Output after change the coordinate

But I think this is due to the way others implement houghLines function in matlab. It won't be a problem if I write it by my self.

a) sigma

Sigma is the parameter for Gaussian filter, which is used to blur the image. As a result, the larger the sigma, blurrier the image will be. At first, increase sigma will cancel some noise in the image.

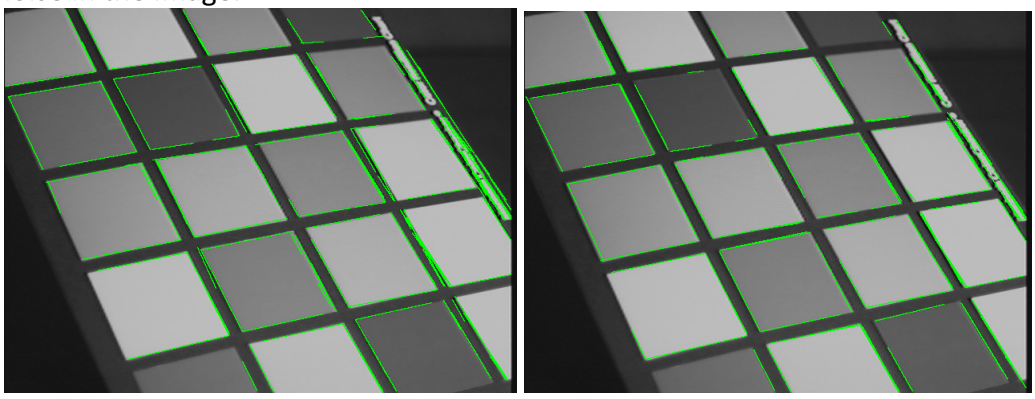


Figure X. sigma = 1 and sigma = 3

As you can see, some noise between squares disappeared after increasing sigma from 1 to 3. But a great increase in sigma will cause important information loss.

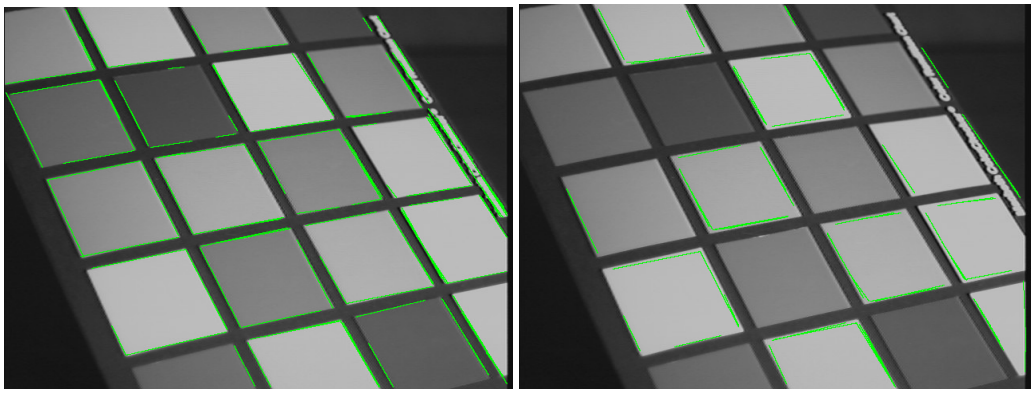


Figure X. sigma = 3 and sigma = 10

Under this circumstance, edges were canceled by the big Gaussian matrix.

b) threshold

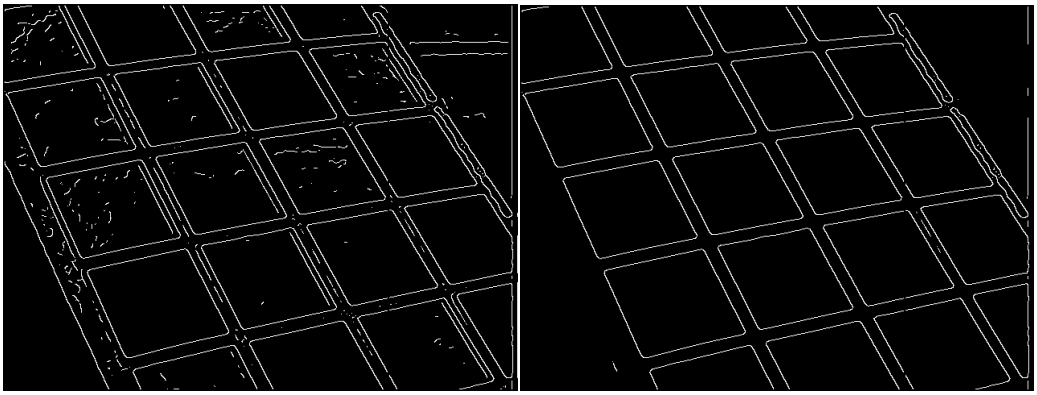


Figure X. threshold = 0.01 and threshold = 0.1

Threshold is used to keep points with strong confident on edges.

c) rhoRes and thetaRes

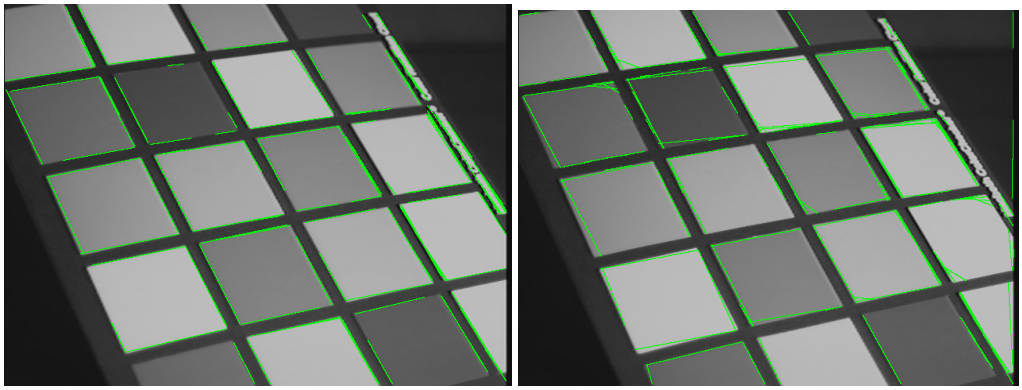


Figure X. rhoRes = 3 and rhoRes = 10

A large rhoRes will cause a small accumulator and a shift in the final result since the resolution is low. The accuracy will decrease if we use a rhoRes like this.

thetaRes plays the same role here.

However, a high resolution doesn't work as well.

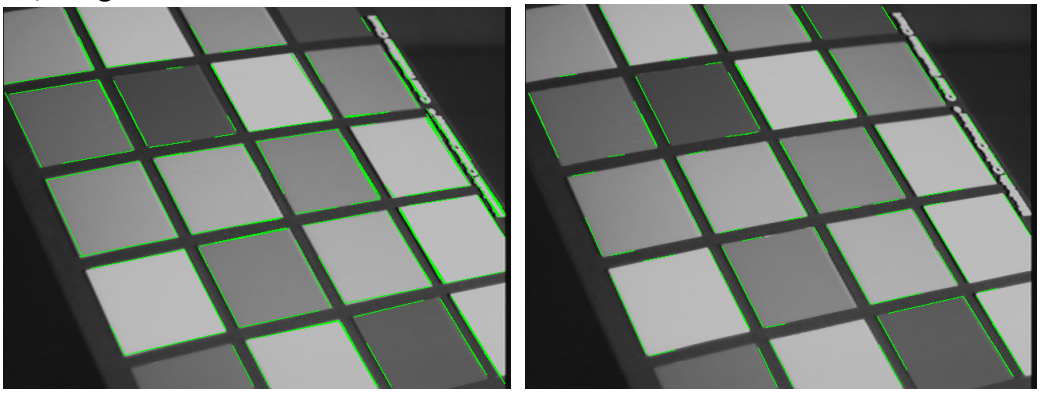


Figure X. rhoRes = 3, thetaRes = 90/pi and rhoRes = 0.5, thetaRes = 180/pi

A high resolution will result in separate line segments since the matrix is so big, maximum points were more 'flat' and distributed.