Manju Karthik Shivashankar | Person ID : 50286752

(1)(a)

Algorithm :1 – Opening followed by closing

First we need to read the image, Opening is just another name of erosion followed by dilation. Here we have implemented erosion first, this removed most of the white noise on the black background. However the following dilation took care of the black noise on the white foreground.

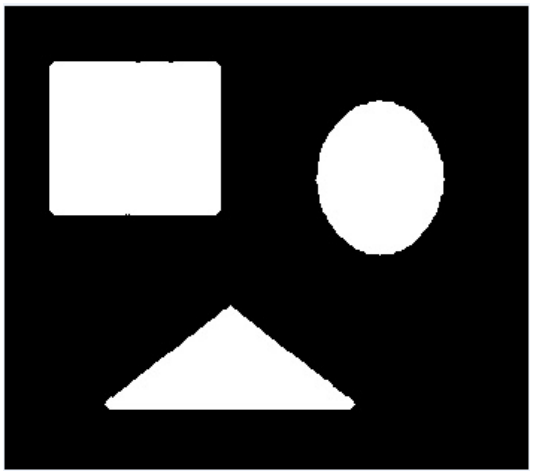


Fig1: res_noise1.jpg

Algorithm :2 – Closing followed by Opening

First we need to read the image, perform Closing- is just another name of dilation followed by erosion. Here we have implemented dilation first, this removed most of the black noise on the white background. However the following erosion took care of the white noise on the black background.

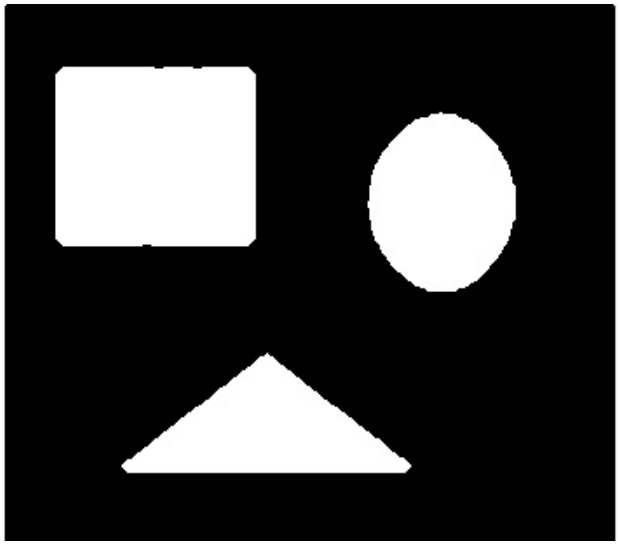


Fig2: res_noise2.jpg

(1)(b)

Comparing the results

When we compare the results of the image obtained from algo1 and algo2, it must be noted that they are almost same, except very few pixels which when eroded first and dilated then resulted in a small difference. But from a distance they both look the same.

(1)(c)

Once the noise has been removed using the two algorithms, we need to find the boundary by just subtracting the final result of the algorithms with the initial noise removed image. This gives us the boundaries of the images.

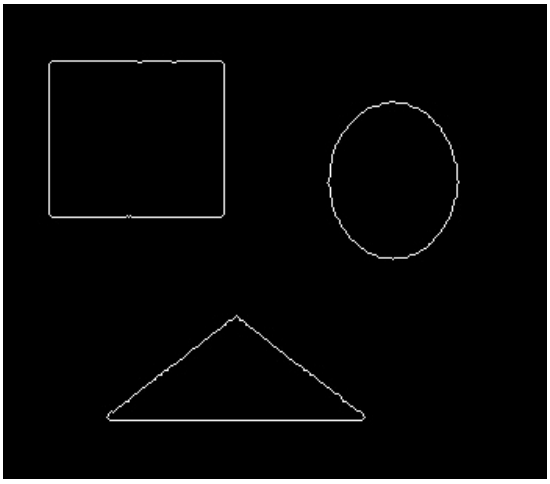


Fig3: res_bound1.jpg

When the two algorithms were implemented it was observed that, there were not many differences in the results except a few pixels. If we carefully observe then it will be evident that the two boundary images are almost same but have small differences.

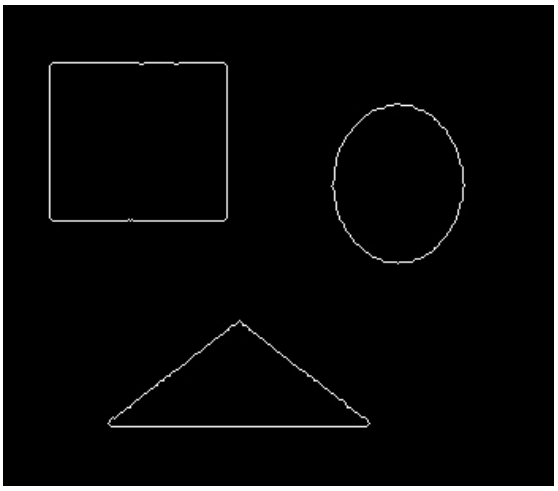


Fig4: res_bound2.jpg

(2)(a) For point detection we used the point detection mask. We then iterated every pixel and performed our operation. Then we normalized the image and displayed as shown below. The point is successfully detected and the co-ordinates are displayed too.

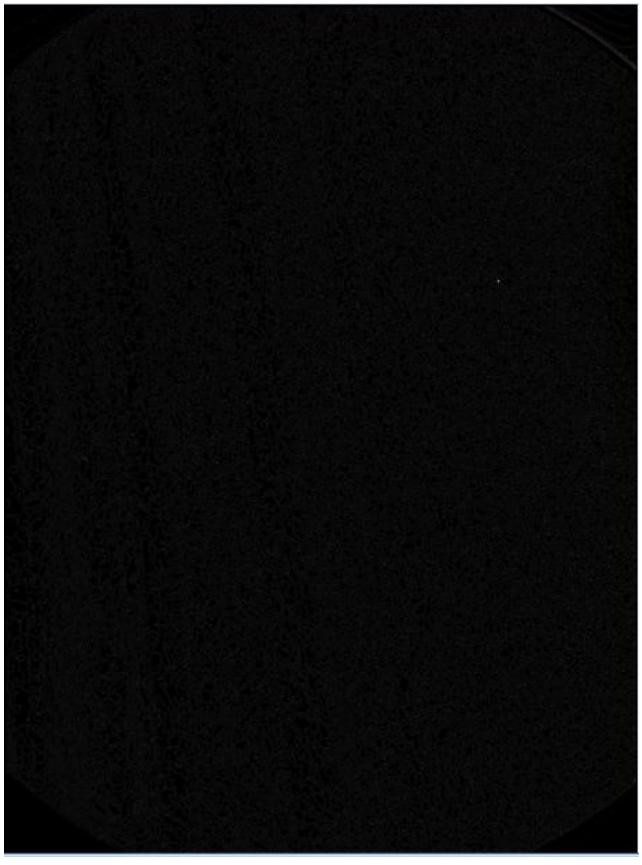


Fig5: detected.jpg

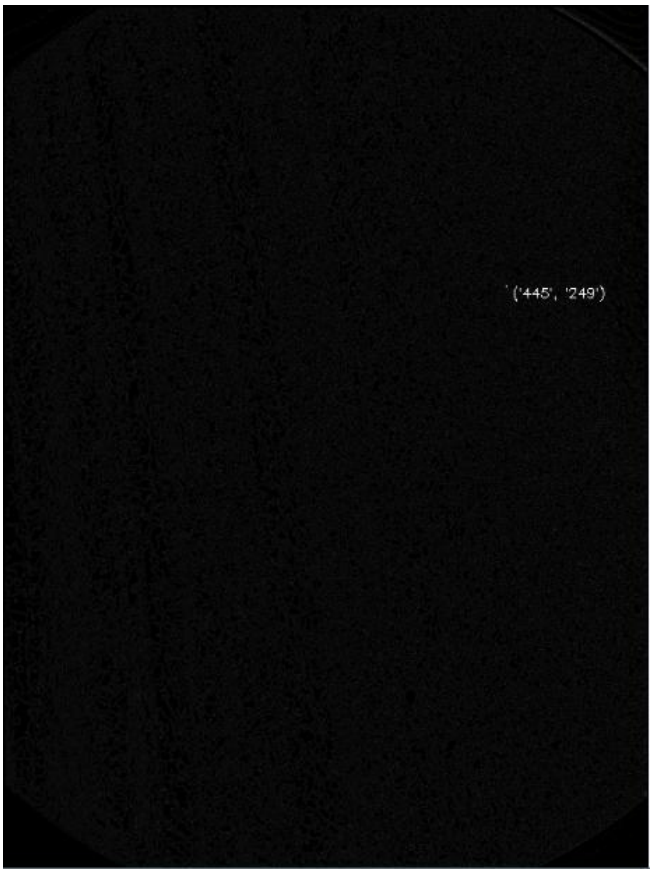
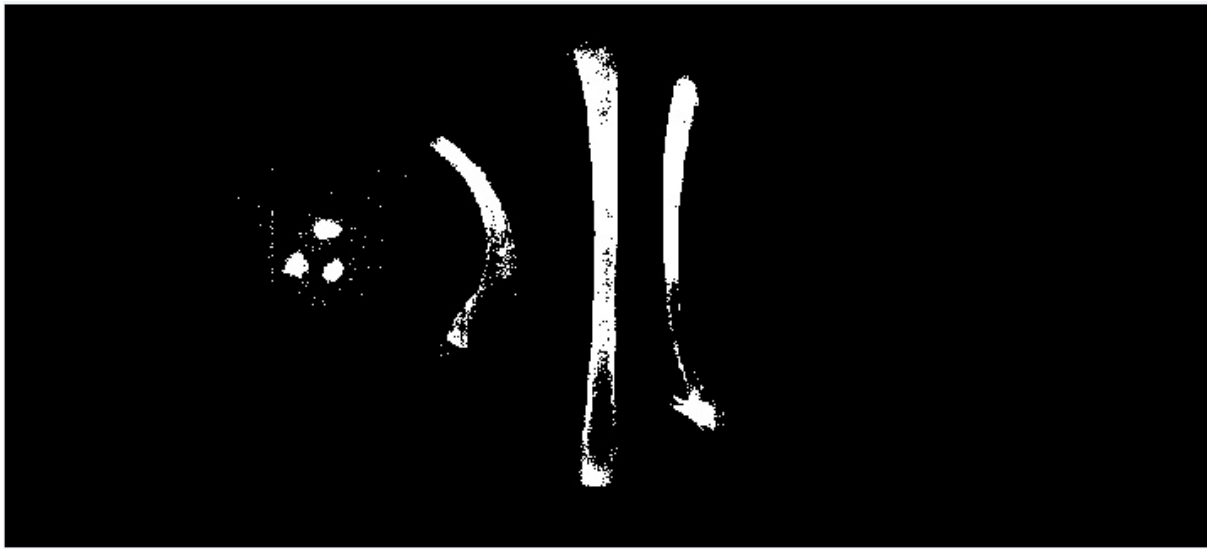


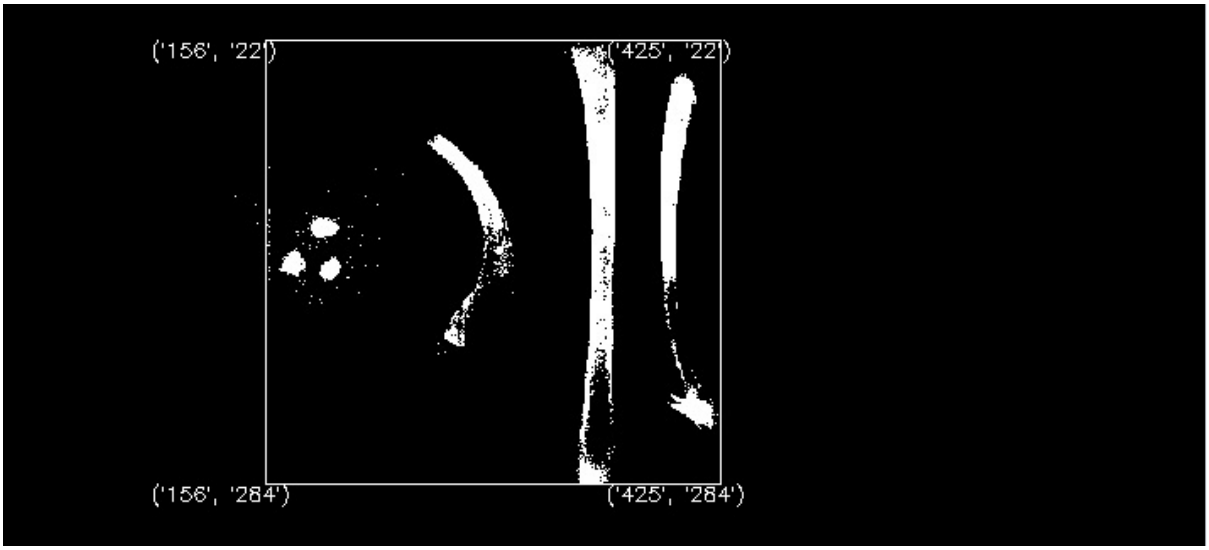
Fig6: detected1.jpg (showing point coordinates)

(2)(b)

In this task I have implemented heuristic thresholding. The image is read in gray scale and then we start by taking the values of pixels above and below an initial threshold. Then we divide the sum of pixel values by their count, later we take the average of these two to find the new threshold. This is done until optimal threshold is reached.



Note: The bounding box also has been coded, the bounding box is not drawn using any tool. It has been implemented from the scratch.



The coordinates as seen in the image are :

x1,y1 = (156,22)

x2,y2 = (425,22)

x3,y3 = (156,284)

x4,y4 = (425,284)

(3)(a)

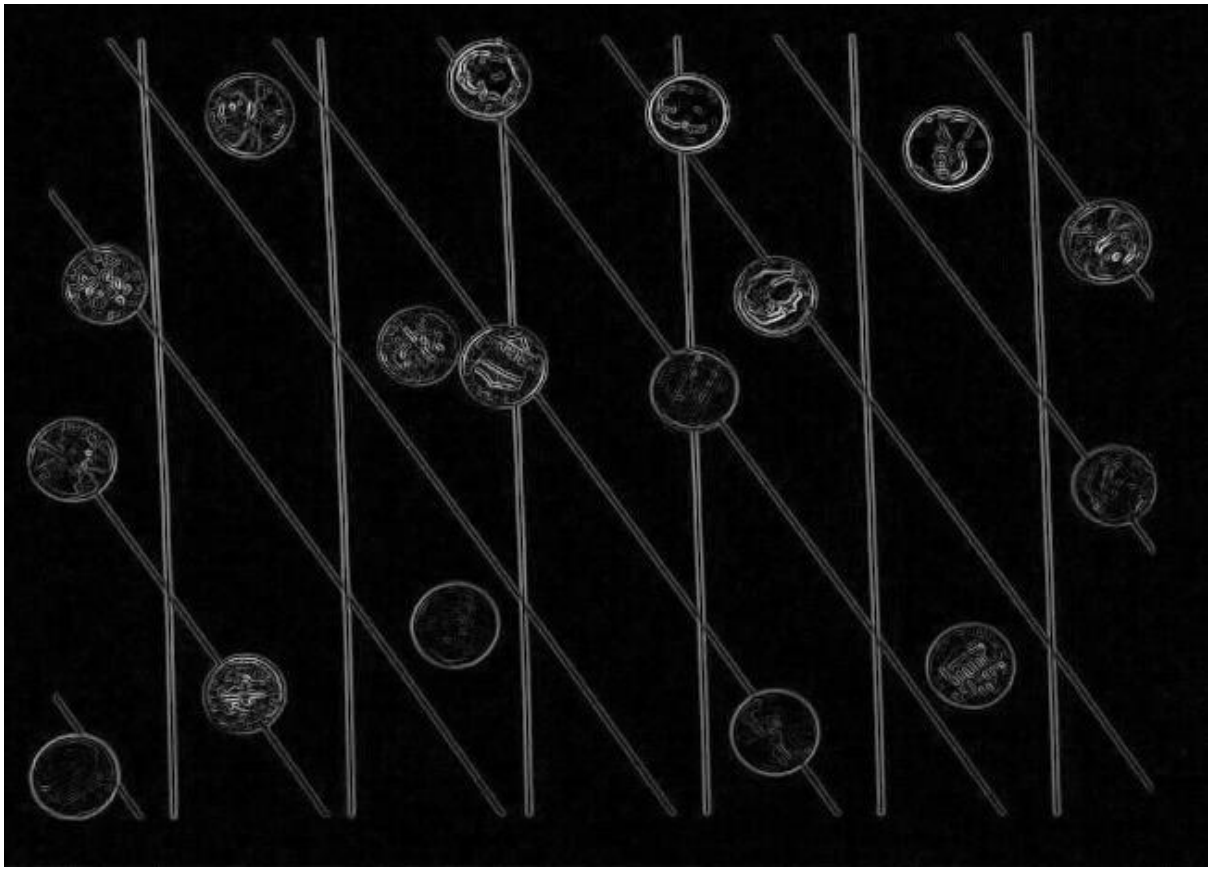
A line can be represented as y = mx+c or in parametric form, as rho = xcos(theta)+ysin(theta)
where rho is the perpendicular distance from origin to the line, and theta is the angle formed by this perpendicular line and horizontal axis measured in counter-clockwise.

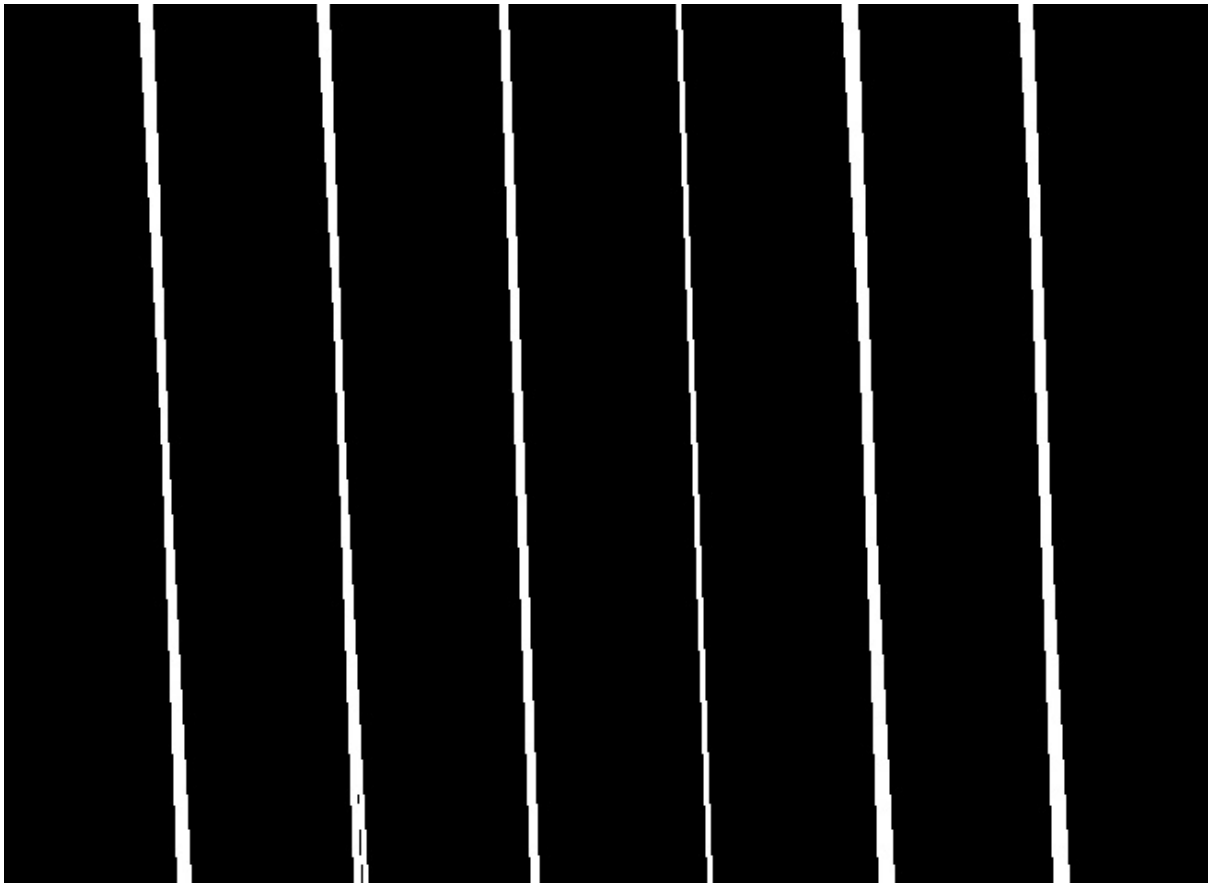
So if line is passing below the origin, it will have a positive rho and angle less than 180. If it is going above the origin, instead of taking angle greater than 180, angle is taken less than 180, and rho is taken negative. Any vertical line will have 0 degree and horizontal lines will have 90 degree.

Now let's see how Hough Transform works for lines. Any line can be represented in these two terms, (rho,theta). So first it creates a 2D array or accumulator (to hold values of two parameters) and it is set to 0 initially. Let rows denote the rho and columns denote the theta. Size of array depends on the accuracy you need. Suppose you want the accuracy of angles to be 1 degree, you need 180 columns. For rho, the maximum distance possible is the diagonal length of the image. So taking one pixel accuracy, number of rows can be diagonal length of the image.

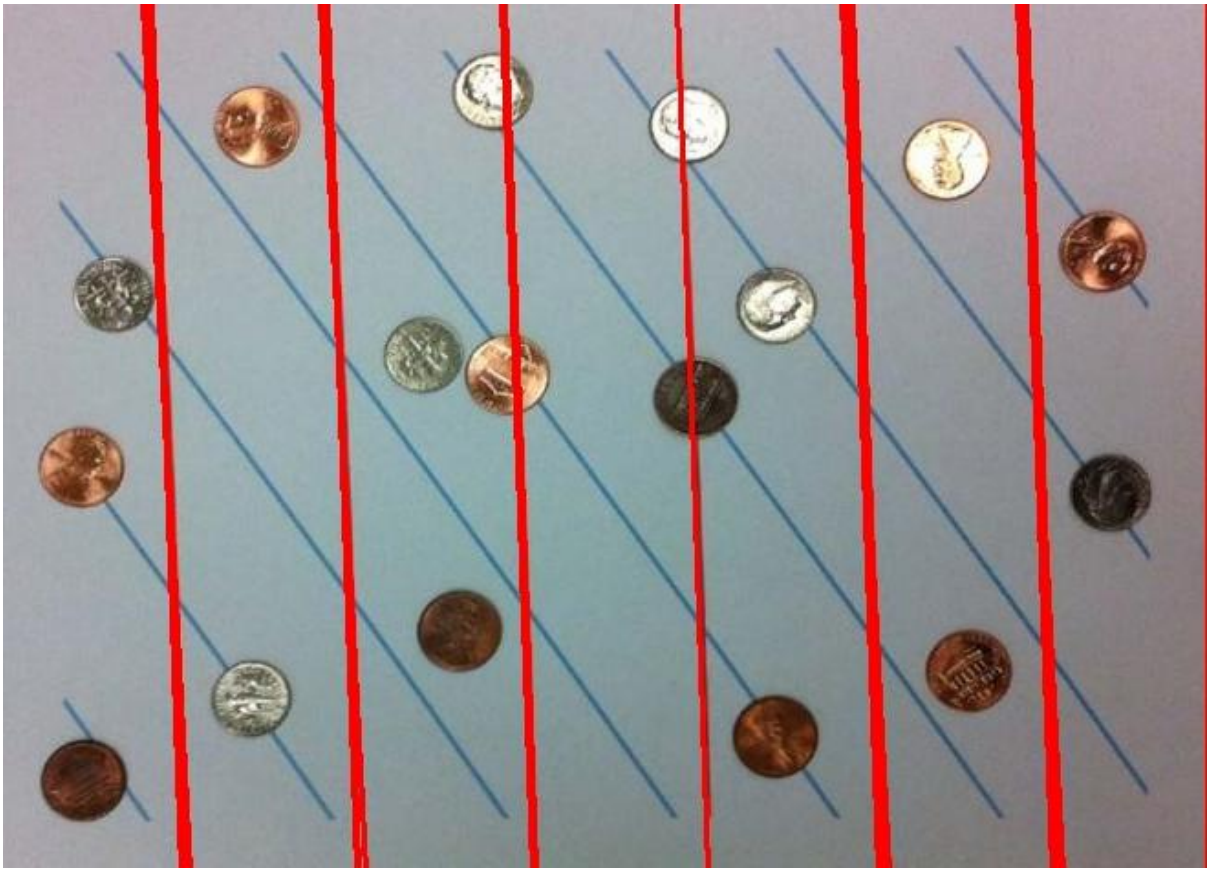
Consider a 100x100 image with a horizontal line at the middle. Take the first point of the line. You know it's (x,y) values. Now in the line equation, put the values 0 to 180 and check the rho you get. For every (rho,theta) pair, you increment value by one in our accumulator in its corresponding (rho,theta) cells. So now in accumulator, the cell (50,90) = 1 along with some other cells.

Now take the second point on the line. Do the same as above. Increment the values in the cells corresponding to (rho,theta) you got. This time, the cell (50,90) = 2. What you actually do is voting the (rho,theta) values. You continue this process for every point on the line. At each point, the cell (50,90) will be incremented or voted up, while other cells may or may not be voted up. This way, at the end, the cell (50,90) will have maximum votes. So if you search the accumulator for maximum votes, you get the value (50,90) which says, there is a line in this image at distance 50 from origin and at angle 90 degrees.



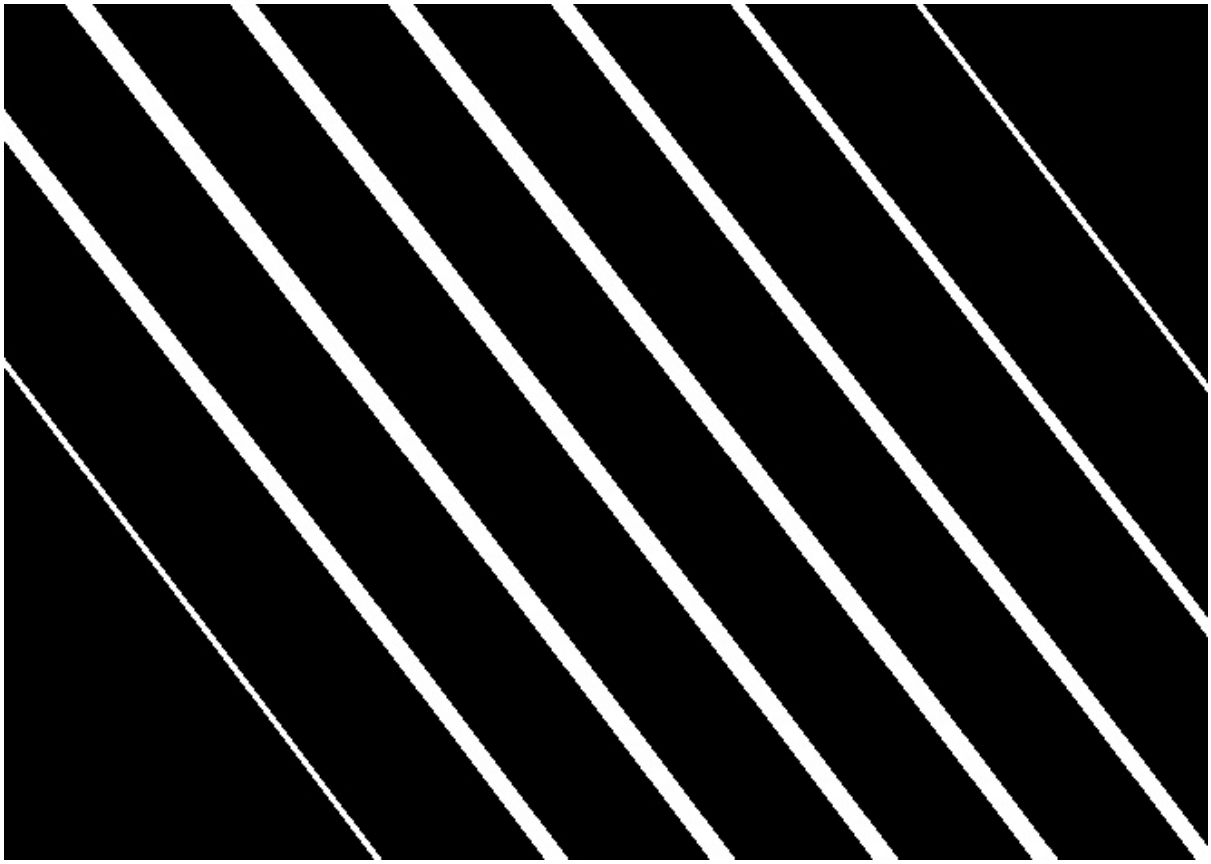


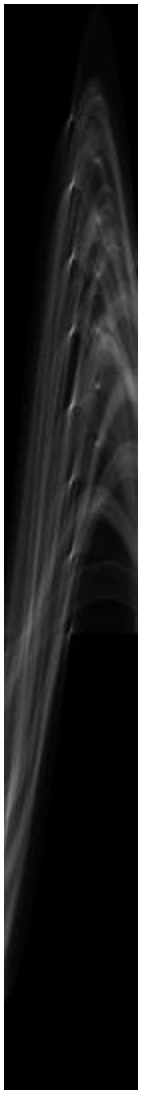
Applied edge detection over sharpened(twice) image.

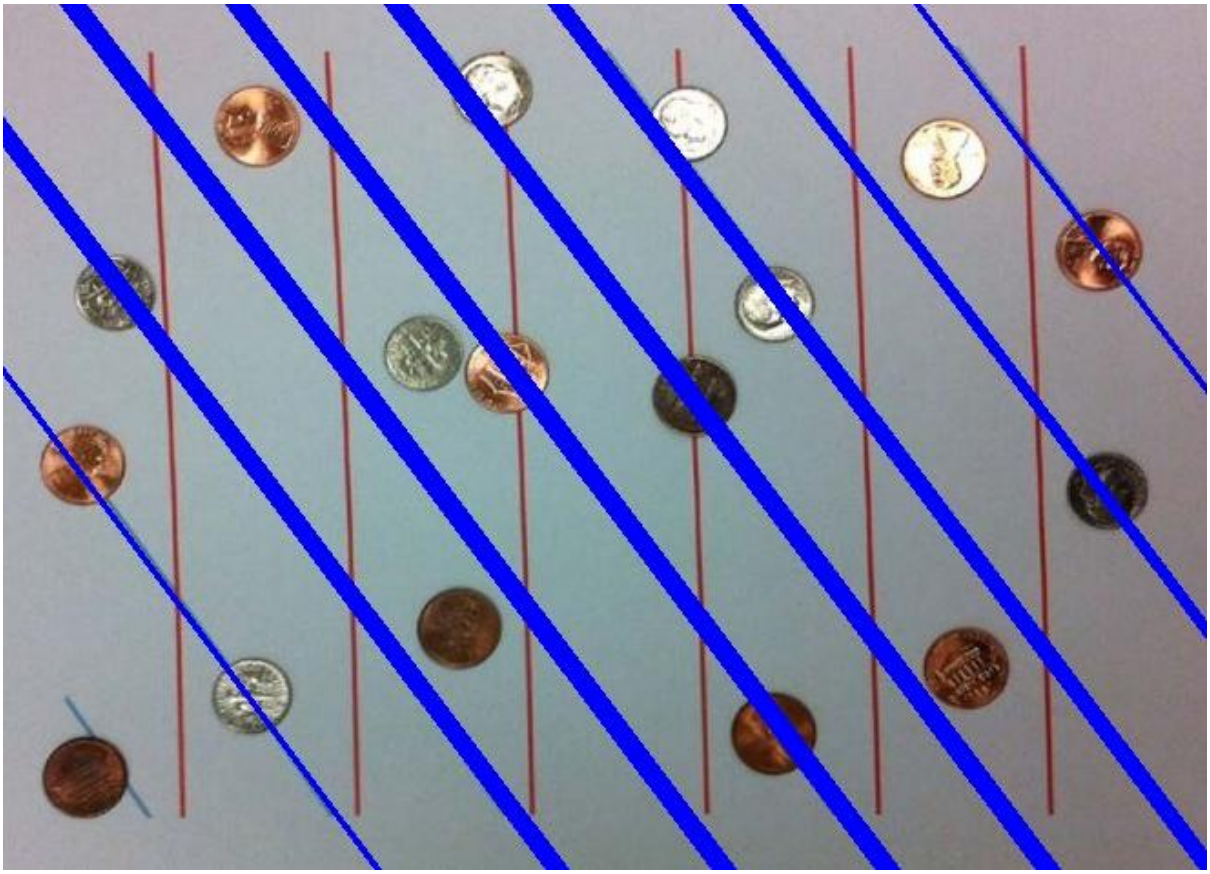


(3)(b)

To prep the image to detect diagonal lines, we need to first extract the diagonal lines using edge detection. Here the below images that you see are the processed images using first a sobel filter and then a angle 45 degree mask. Finally we find a threshold that removes all other noise and the highlighted diagonal lines are obtained.







(3)(c)

$x = a + R\cos\theta$

$y = b + R\sin\theta$

So, every point in the xy space will be equivalent to a circle in the ab space (R isn't a parameter, we already know it). This is because on rearranging the equations, we get:

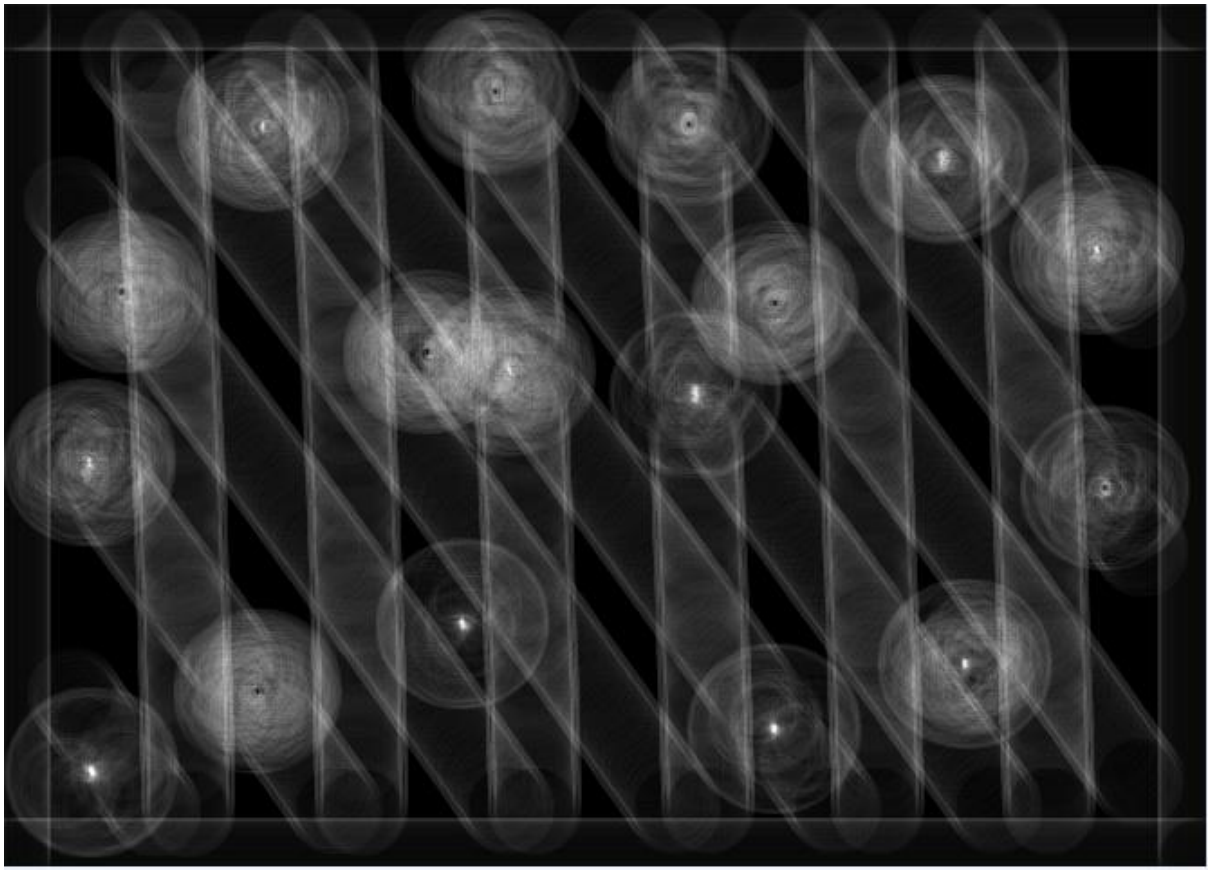
$a = x_1 - R\cos\theta$

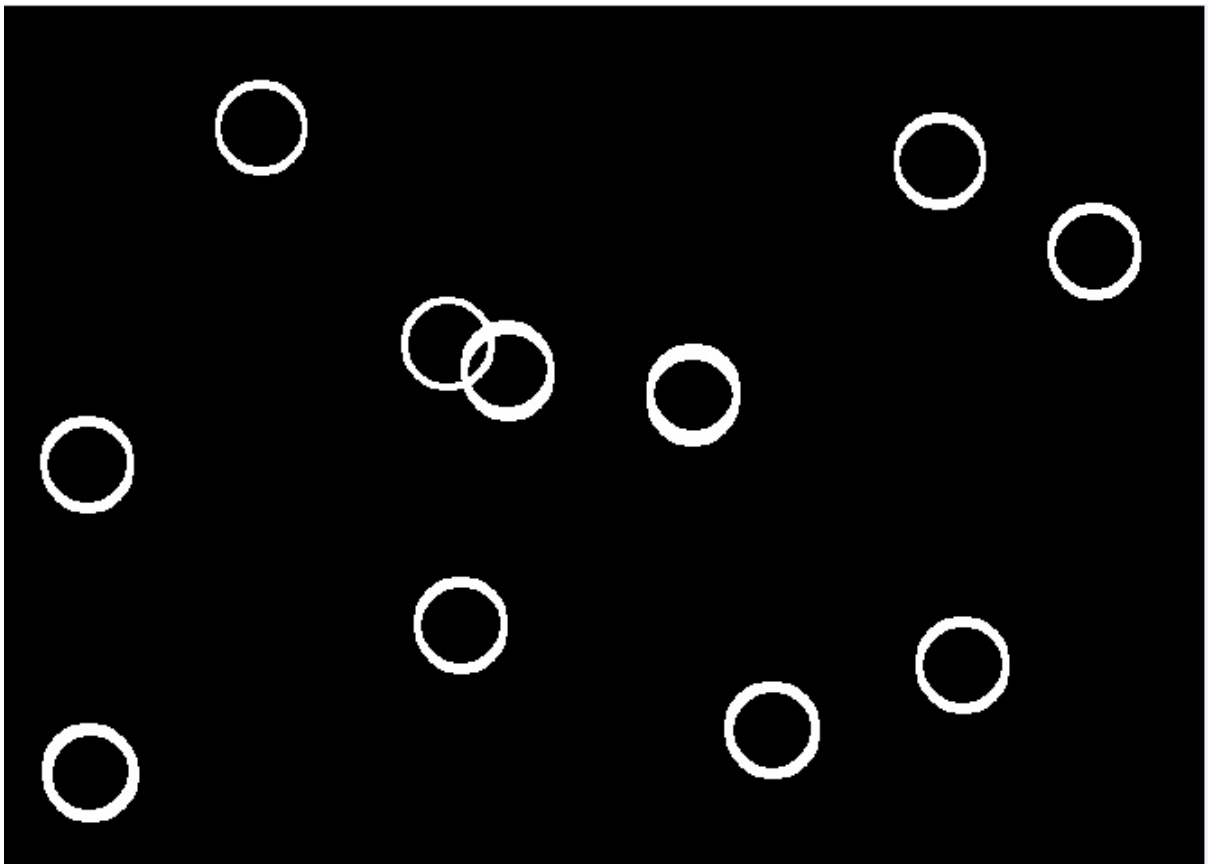
$b = y_1 - R\sin\theta$

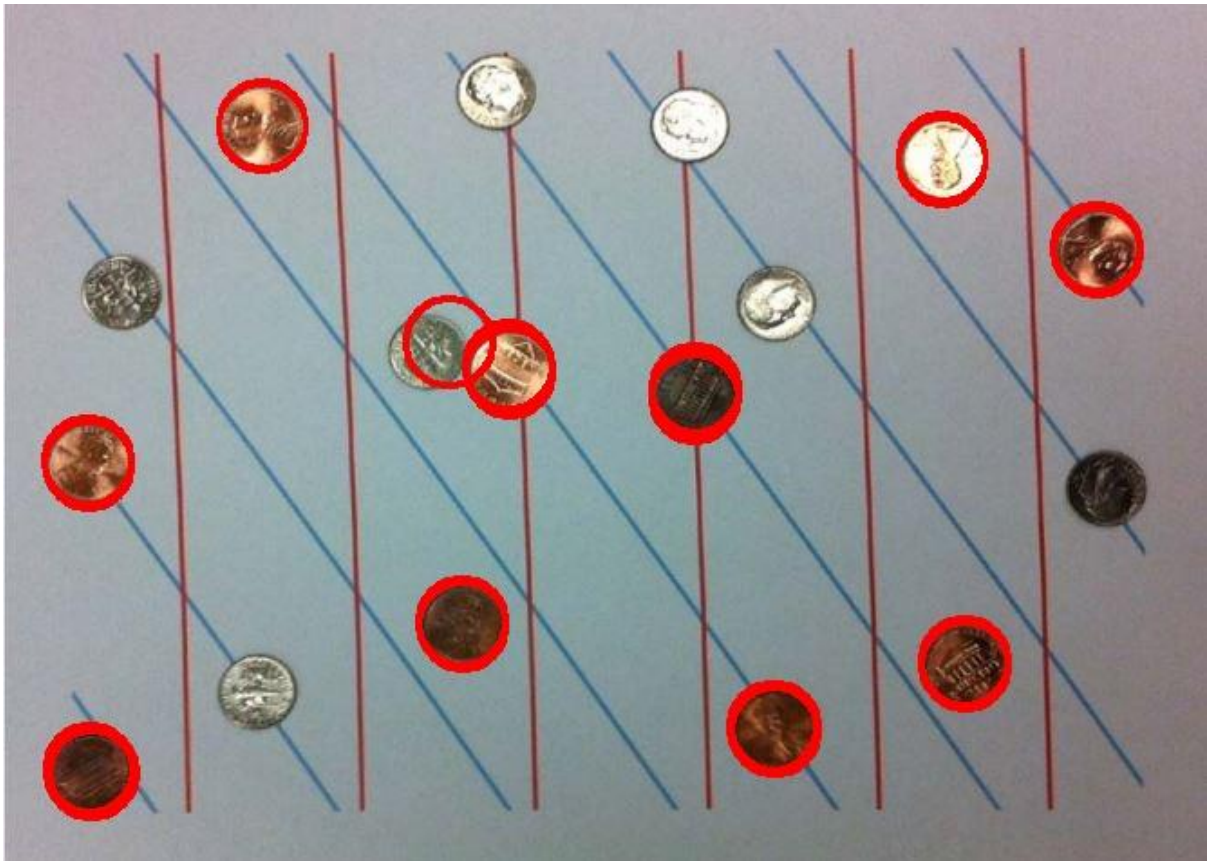
for a particular point (x_1, y_1). And θ sweeps from 0 to 360 degrees.

So, the flow of events is something like this:

1. Load an image
2. Detect edges and generate a binary image
3. For every 'edge' pixel, generate a circle in the ab space
4. For every point on the circle in the ab space, cast 'votes' in the accumulator cells
5. The cells with greater number of votes are the centres







In conclusion, we detected the following things :

(i) Number of vertical lines(red) = 6
(ii) Number of diagonal lines(blue) = 8
(iii) Number of coins = 11

References:

http://www.aishack.in/tutorials/circle-hough-transform/

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html