Task 1: Morphology

Morphological image processing is a collection of non-linear operations related to the shape or morphology of features in an image. Morphological techniques use small shape or template called a **structuring element**. Every structure is binary image with values 0 or 1. Matrix dimension determines size of structuring element. Pattern of one and zeros determines shape of structuring element. Every structuring element has origin which is one of its pixels.

For performing morphological operation, we have used structuring element of matrix of size (3,3).

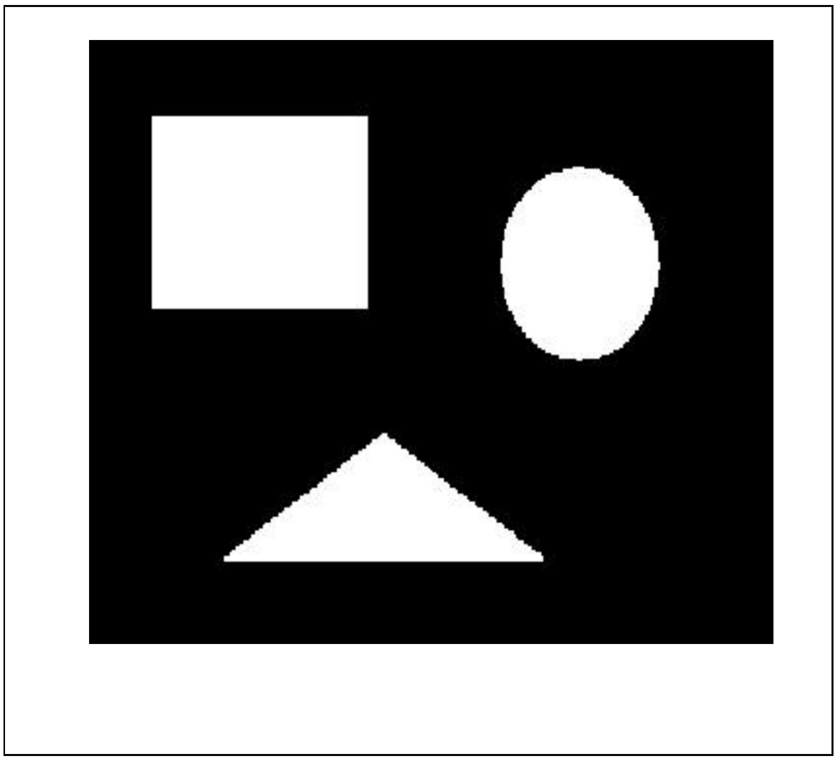
1	1	1
1	1	1
1	1	1

We have used two fundamental morphological operations:

1) Erosion - Erosion of A by B is the set of all points z such that B, translated by z , is contained in A.
2) Dilation - Dilation of A by B then is the set of all z displacements such that and A overlap by at least one nonzero element.

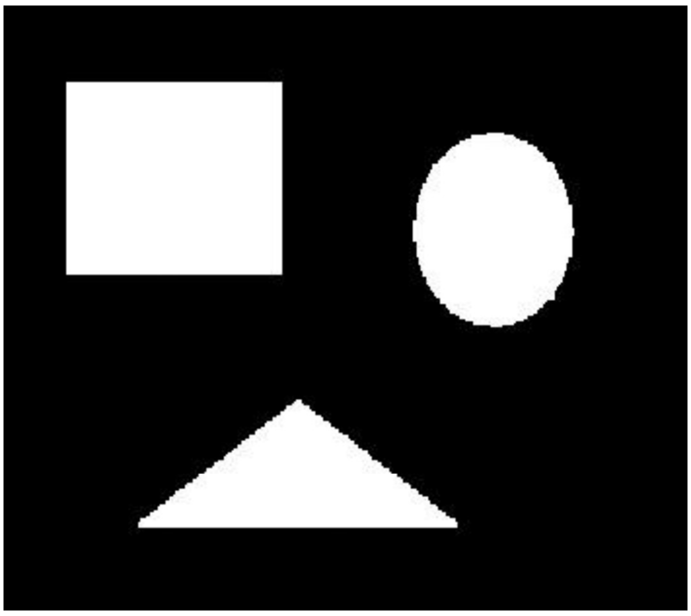
More complicated morphological operations such as opening, and closing are performed by combination of simple morphological operations such as erosion and dilation.

First, we have performed opening followed by closing. Opening is performed by erosion followed by dilation and closing is performed by dilation followed by erosion. This is also called as **denoising**.



(res_noise1.jpg)

Now we perform closing followed by opening. It removes noise from image.


(res_noise2.jpg)

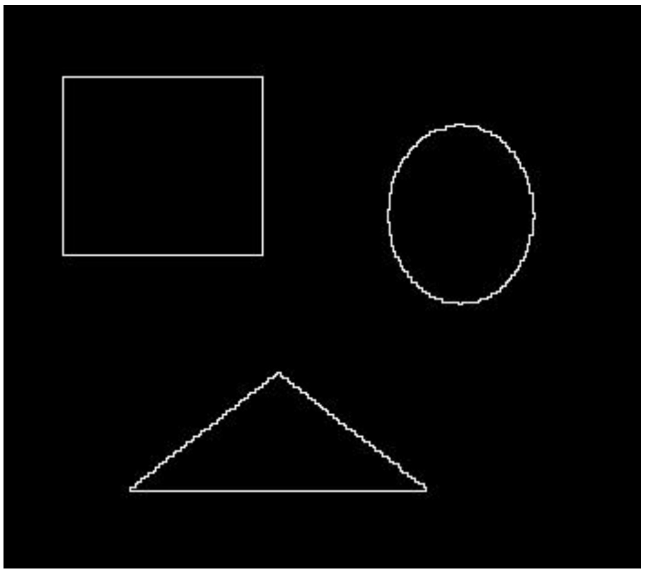
Observation:

1) After performing above operations, we are able to completely **remove noise from image**.
2) This operation has **preserved the shapes of object.**
3) We have seen minor differences in final output of above operations i.e. **shapes of objects deformed slightly.**

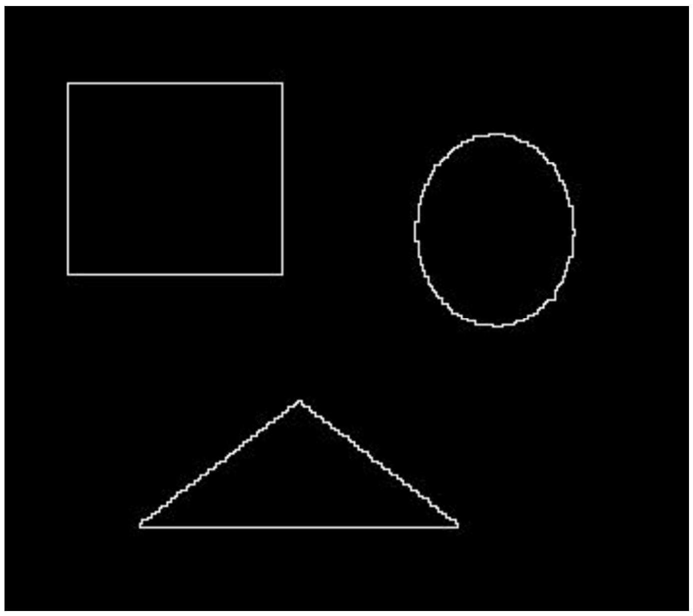
Boundary detection – Morphology

For boundary extraction we have subtracted above outputs res_noise1.jpg and res_noise2.jpg from output obtained from performing erosion on that. The boundary of a set A, denoted by $\beta(A)$, $\beta(A) = A - (A ! B)$

Opening - > closing -> boundary extraction


(res_bound1.jpg)

closing - > opening - > boundary extraction



(res_bound2.jpg)

Task 2: Image segmentation and point detection

2 (a):

We can detect three types of discontinuities in image such as point detection, edge detection and line detection. For detection we move/ run kernel over images.

For point detection we run one kernel over image kernel is as follows:

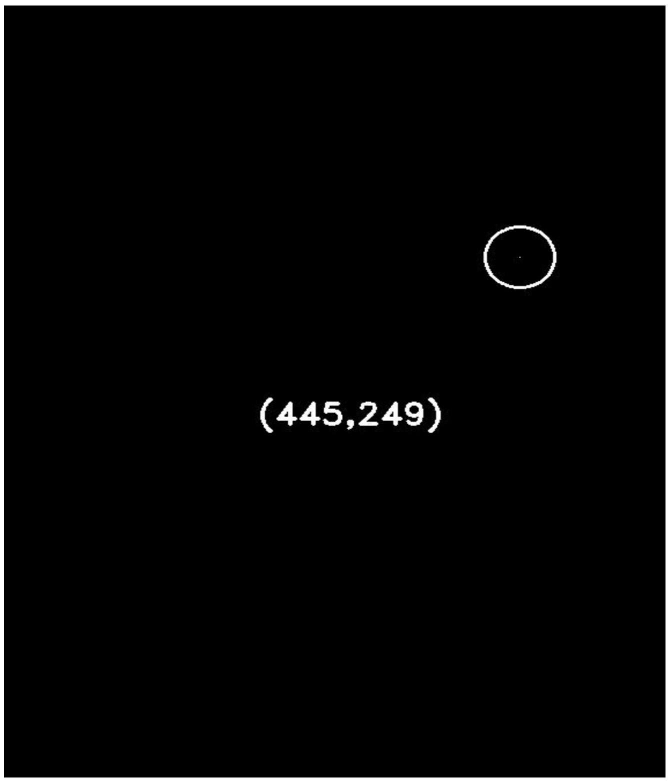
-1	-1	-1
-1	8	-1
-1	-1	-1

1) We perform correlation operation as we run kernel over image. (R)

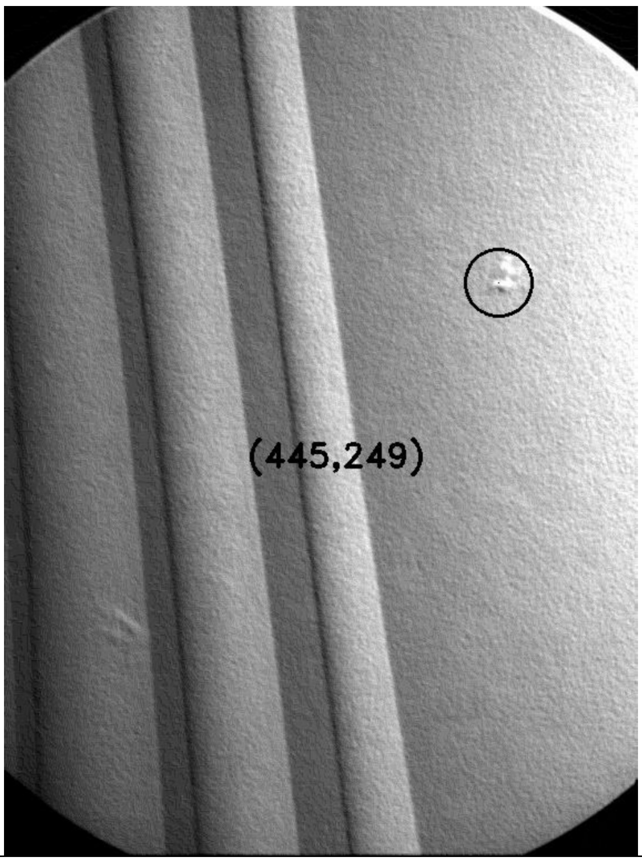
2) We choose arbitrary threshold randomly (T)

3) A point will be located at location on which is mask is centered if |R| > T.

We perform correlation using above kernel and take absolute pixel values. Then we choose threshold value and detect point on given image and circled and marked detected point.



(point_detected.jpg)

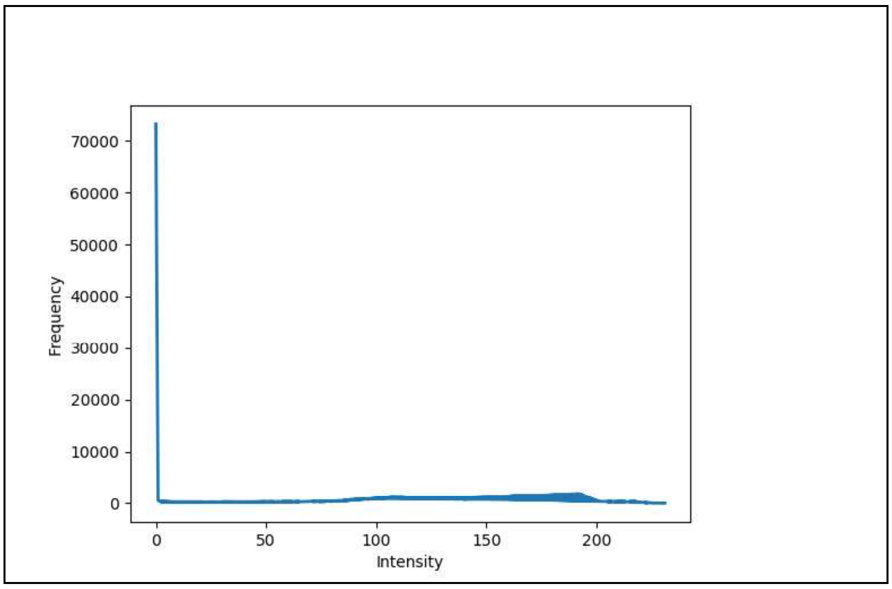


(point_detected_final.jpg)

2 (b):

To find optimal threshold we **draw histogram of given image**. Histogram is drawn without using any histogram library.

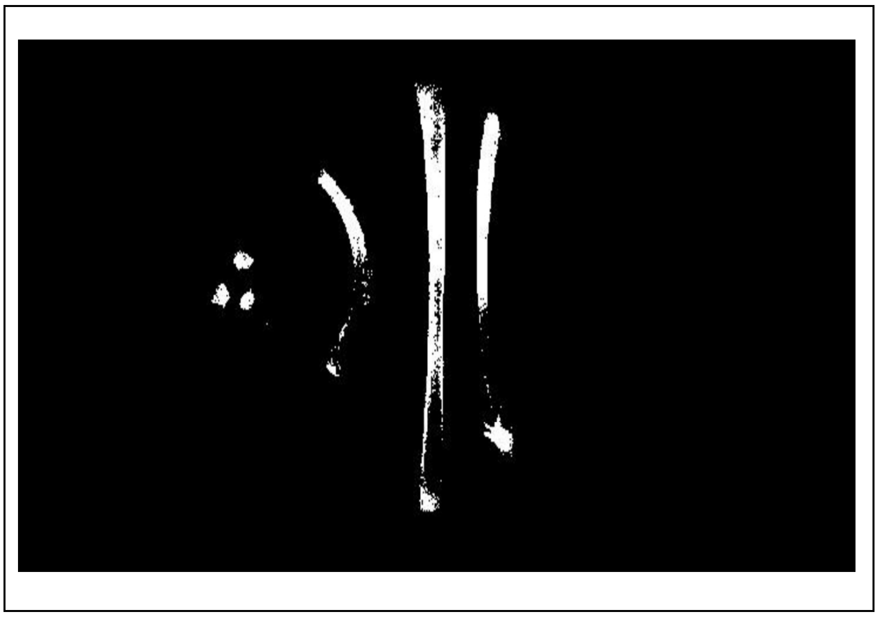
Histogram:



(histogram.png)

After observing above histogram, we can clearly say that pixel with 0 intensity has maximum value and which is background of image we also observe another peek near 200. By choosing threshold value equals to 204 we get below image as output where bones are clearly identified.

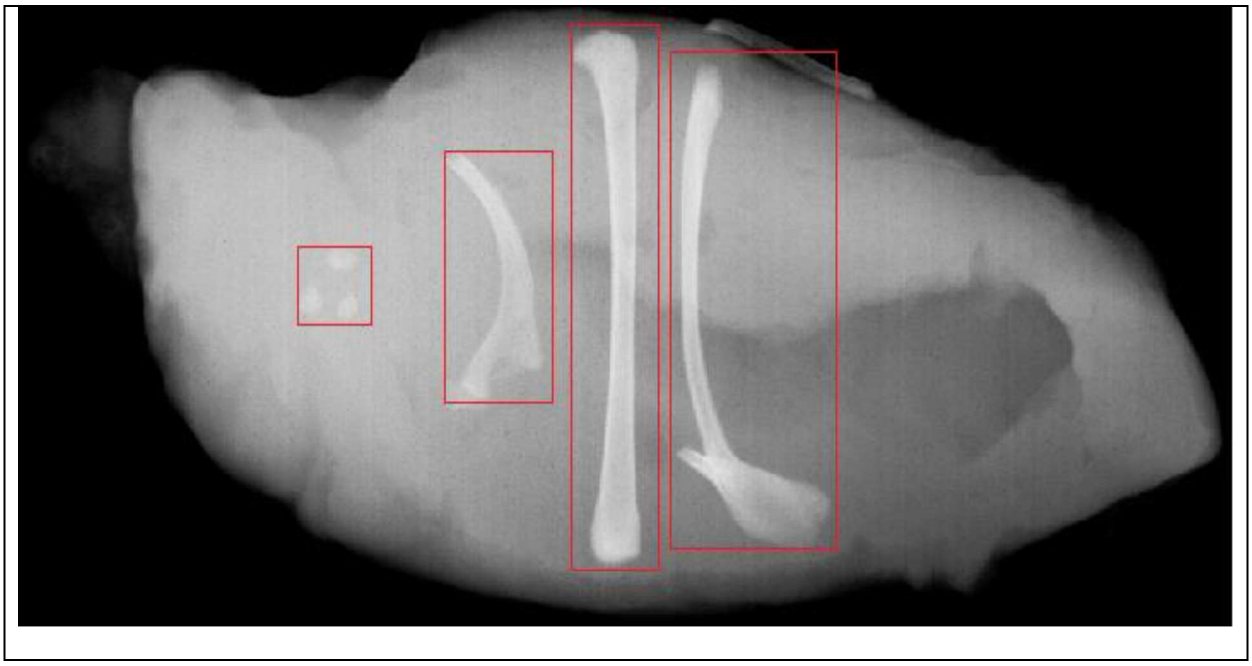
(T = 204)



(thresold.jpg)

Bounded Image:

We draw bounded boxes around object that we detected and corner coordinates of all squares are noted.



(bounding_box.jpg)

Bounded box coordinates –

Box 1 – top-left - (164,127), top-right-(207,127), bottom-left (164,167), bottom – right (207,167)

Box 2 – top- left - (250,78), top -right - (314,78), bottom-left- (250,207), bottom-right (314,207)

Box 3 – top-left -(325,14), top-right -(376,14), bottom-left (325,293), bottom-right (376,293)

Box 4 – top -left -(383,28), top-right -(480,28), bottom-left (383,282), bottom-right (480,282)

Task 3: Hough Transform

To perform Hough transform we will run **sobel mask** over image to detect edges in x direction and y direction. We perform **convolution** operation. After running sobel mask we threshold an obtained output to form binary image. Below is the image obtained after threshold is applied. Sobel operator as follows.

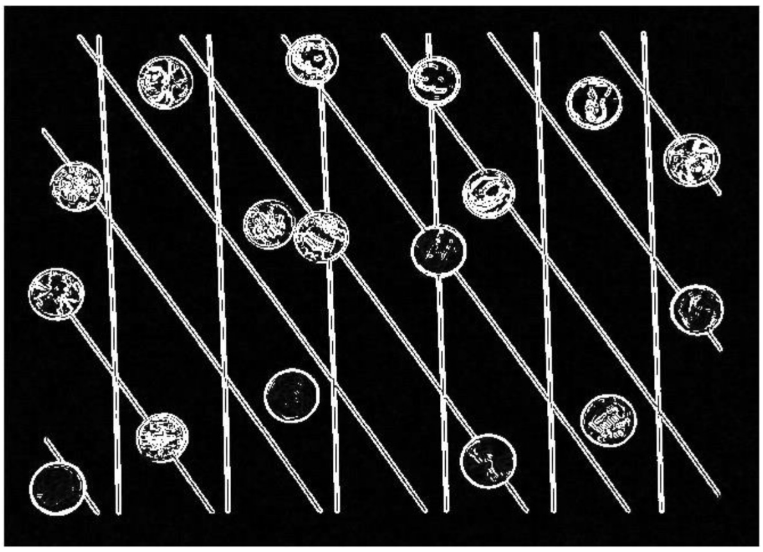
X – direction :

-1	0	1
-2	0	2
-1	0	1

Y – direction :

-1	-2	-1
0	0	0
1	2	1

Sobel operator output:

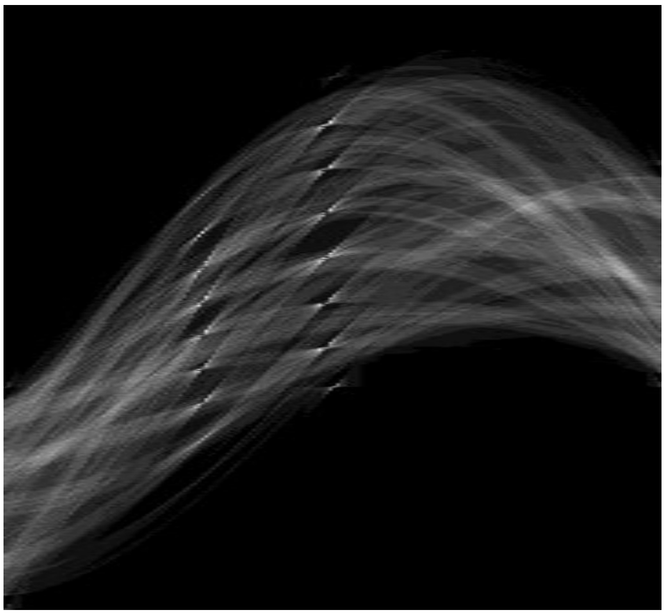


(sobel_result.jpg)

Now our goal is transform given line in hough space. A line in image space can be expressed in the form of cartesian coordinate system or polar coordinate system. For hough transform we will express line in polar form, so equation of line becomes

$$\boxed{r = x \cos\theta + y \sin \theta}$$

We transform from (x, y) space to (r, θ)For each (x, y) pair we can plot the family of lines that goes through it and we get **sinusoid** as follows:

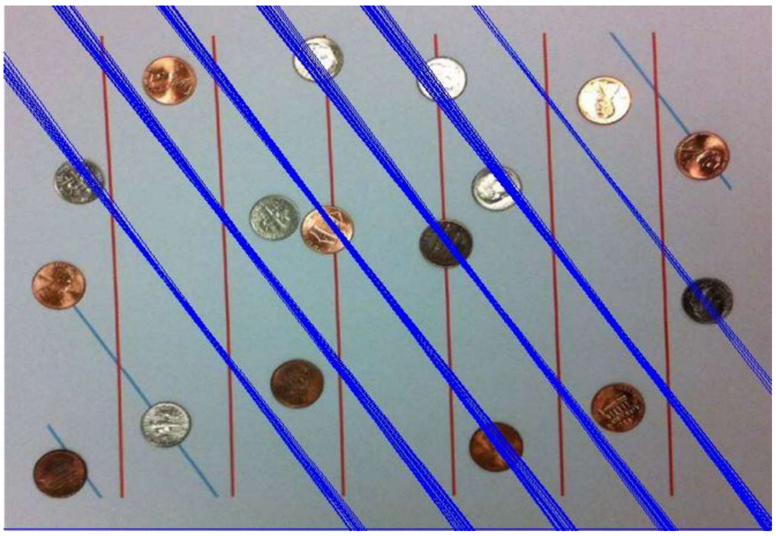


(sinusoid.jpg)

We perform this operation for all points in image and find out sinusoid curves in (r, θ) space. We can observe that each curve intersects with some other curve. **If the curves intersect in (r, θ) space** for two different points it indicates that both **points lie on same line.**

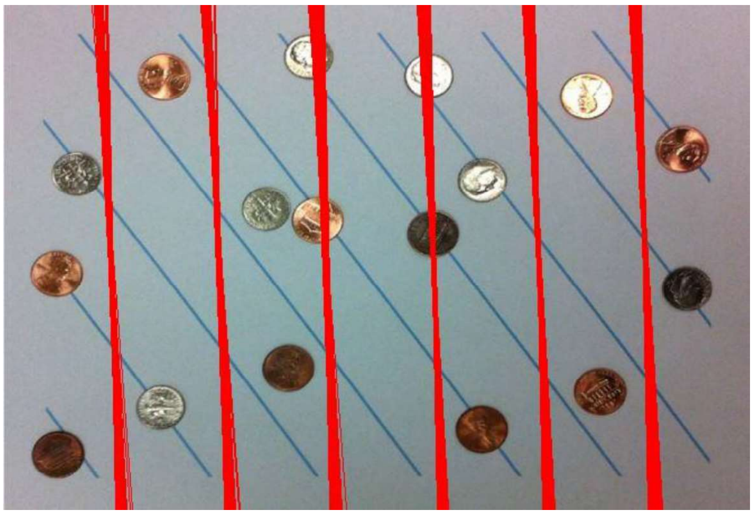
Hence, we detect lines by finding intersection of curves. But we need to set some **threshold** for number of intersections. If the number of intersections is above threshold value, then it's detected as line. For keeping track of number of intersections we use **voting array**.

Blue Lines - > We have **detected 6 blue lines**.



(blue_lines.jpg)

Red Lines - > we have **detected 6 red lines**.



(red_lines.jpg)

Bonus Task:

Now our task is to detect circles in image. Before moving ahead with detecting circles, we have to detect edges in image that we will detect using sobel operator.

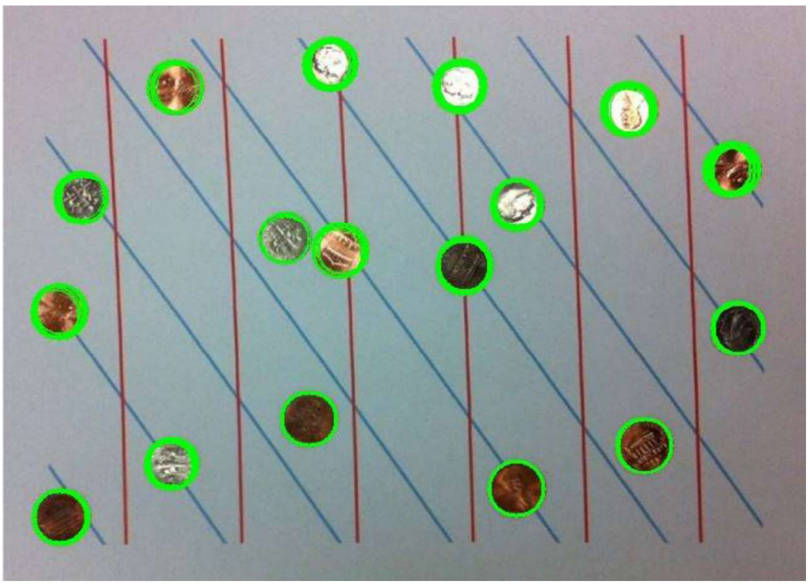
Finding circle is same as detecting lines. A circle is defined by center and radius i.e. (a, b) & r.

Center can be represented as **a = x- r*cosθ , b = y + r*sinθ**. θ varies from 0 to 360 and that completes a circle with radius r. So our goal is to find tuple of (a, b,θ).

After finding tuple of (a, b,θ).we will **filter out negative values of a and b** i.e. tuple with positive values of a and b are kept and other are ignored. We will keep track of tuple values using accumulator array.

Finally we will chose some random threshold on the basis of max value found from accumulator array. In our case threshold = 0.7 * max(accumulator). Tuple values which are above threshold are chosen and circles are drawn using this points and radius.

Coins -> **17 coins are detected**



(coins.jpg)

References:

1. https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html

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