UNIVERSITY AT BUFFALO, DEPARTMENT OF COMPUTER SCIENCE

Computer Vision and Image Processing

Project 3 : Project Report

Sagnik Ghosh
UB Person Number : 50289151
December 3, 2018

1 MORPHOLOGY IMAGE PROCESSING

The term morphology has originally evolved from the word "morph" or "shape". In morphological image processing we mostly deal with the shape of the image. Morphological image processing is a collection of non-linear operations related to the shape or morphological of features in an image.

morphological operations rely only on the relative ordering of pixel values, not on their numerical values, which is why they are especially suited for binary images. Morphological operations can also be performed on grey-scale images, since the color intensity values (r,g,b) are unknown and therefore their absolute pixel values are of no interest.

Morphological techniques match an image with a small shape or template called a "structuring element". The structuring element is positioned at all possible locations in the image and it is compared with the corresponding overlapping pixels.

One particular element of a structural element is called the "origin". Any element of the structural element can be set to be the origin.

Different kind of operations are then performed between the values of the structuring element and the corresponding pixel value.

There are two basin "Morphological Operations" in image processing:

1.1 EROSION AND DILATION

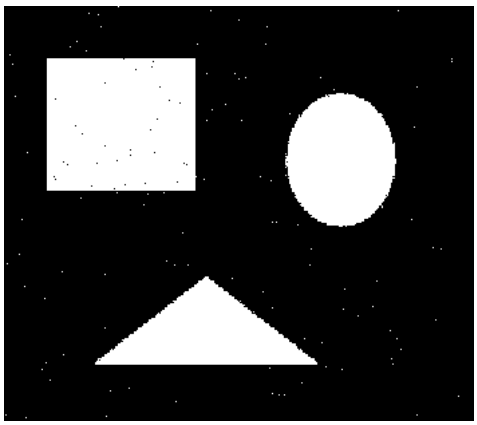
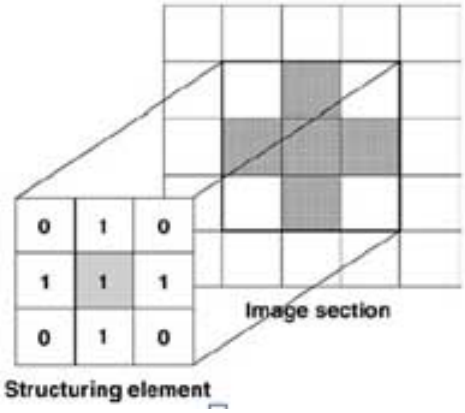


Figure 1.1: Input Image

In "Erosion", if all the values of the structuring element matches with the overlapping pixels values, then the value of the pixel overlapping with the origin is set to 255 and the other overlapping pixel values are set to 0.

Since this operation is basically reducing the number of pixels containing 255 values from the edges of the white objects. This in terms reduces the size of the white object in the image.

The following segment of code explains the erosion operation:

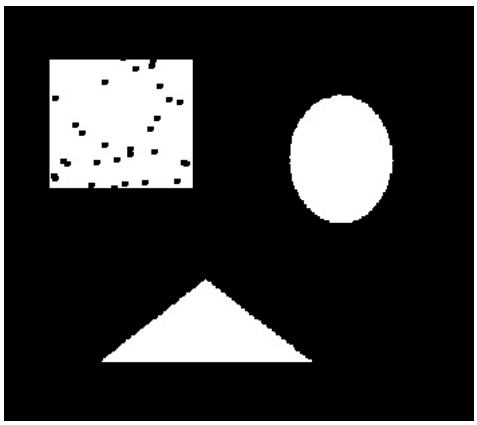



```
r,c = org_img.shape
img = np.zeros(org_img.shape)
img = np.pad(img,(1,1), 'constant')
for i in range (0,r):
    for j in range (0,c):
        if (org_img[i-1][j] == 255 and org_img[i][j] == 255
        and org_img[i+1][j] == 255 and org_img[i-1][j+1] == 255
        and org_img[i][j+1] == 255 and org_img[i+1][j+1] == 255
        and org_img[i-1][j-1] == 255 and org_img[i][j-1] == 255
        and org_img[i+1][j-1] == 255):

                img[i][j]   = 255

        else:
            img[i-1][j] = 0
            img[i+1][j] = 0
            img[i-1][j+1] = 0
            img[i][j+1] = 0
            img[i+1][j+1] = 0
            img[i-1][j-1] = 0
```

```
                img[i][j-1] = 0
                img[i+1][j-1] = 0

    cv.imwrite("erosion.jpg", img)
    return img
```

- **Output Image for Erosion**

As we can see, the the white objects have shrink-ed and the black pixels within the white objects have been enlarged.



In case of the "dilation" operation, if only the origin value of the structuring element is same as the overlapping element, all the overlapping pixel values of the image is set to 255. This in terms increases the size of the white object in the image.

Since this operation is basically increasing the number of pixels containing 255 values from the edges of the white objects. This in terms increases the size of the white object in the image.

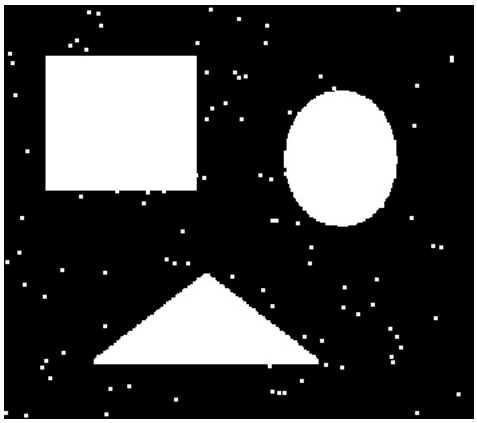
The following segment of code explains the erosion operation:

```
import cv2 as cv
import numpy as np

def dilation(org_img):

    r,c = org_img.shape
    img = np.zeros(org_img.shape)
    img = np.pad(img,(1,1), 'constant')
    for i in range (0,r):
```

```
            for j in range (0,c):
                if (org_img[i][j] == 255):
                    img[i-1][j]   = 255
                    img[i][j]     = 255
                    img[i+1][j]   = 255
                    img[i-1][j+1] = 255
                    img[i][j+1]   = 255
                    img[i+1][j+1] = 255
                    img[i-1][j-1] = 255
                    img[i][j-1]   = 255
                    img[i+1][j-1] = 255

        cv.imwrite('dilation.jpg', img)
    return img
```

- **Output Image for Dilation**

As we can see, the the blac objects have enlarged and the white pixels within the black area have also been enlarged.



1.2 Opening and Closing Operation

Based on these two basic morphological operations (erosion and dilation) several other operations can be performed. Opening and closing is nothing but combination of erosion and dilation performed in a different sequence.

In "opening" operation, first we perform erosion, on the input image and then on the resulting image, we perform dilation.

In contrast, in case of "closing" operation, first, we perform dilation on the input image and then we perform erosion on the resulting image.

Basically both the "opening" and "closing" operations returns the image objects nearky to the original size but with reduced noise.

In this project we were supposed to perform two combinations of opening and closing, as in, "opening after closing" and "closing after opening". First "opening after closing" was performed and then on the resulting image, "closing after opening" was performed.

- **Output Image**

Figure 1.2: Closing after Closing

Figure 1.3: Opening after Closing

2 POINT DETECTION

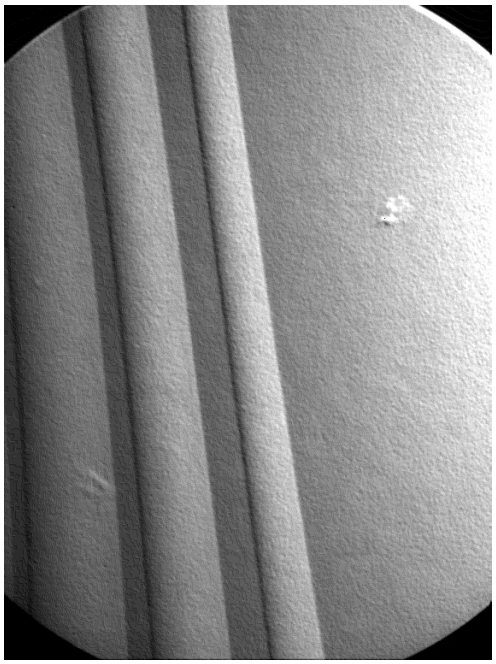


Figure 2.1: Input Image

In this problem we are supposed to detect the porosity or porous or holes on the input image. This porous or holes can be identified by the inconsistency in intensity of pixels from the neighbouring pixels.

To solve this problem first we have to run a point detection algorithm using "Laplacian filter". The Laplacian kernel can be described as follows:

The first step of this algorithm is to perform element wise multiplication between the Laplacian kernel elements and the overlapping image pixel values and then the results are added. The final value is then replaced with the center pixel value.

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

The centering element value and all the other element values of the kernel will always add up to 0. This is because, if the overlapping pixel values of the image is consistent or all the pixels have same value then after multiplication and addition, the resulting value would be zero, or the pixel at the center will become black.

On the other hand, if the overlapping pixel values are inconsistent or the centering pixel has higher intensity, then the process will increase its intensity by multiple times whereas the neighbouring pixel values will become negative.

After These steps the pixels of our image will contain either very low intensity value or very high intensity value. After this step the image will look like this:

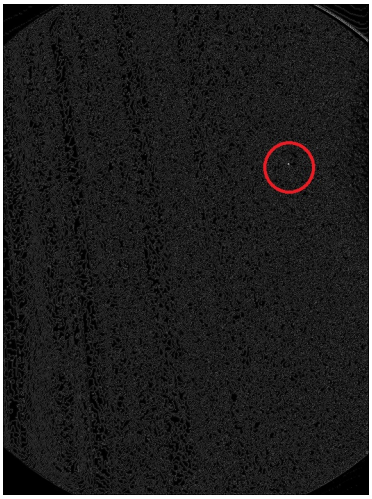


Figure 2.2: After applying Laplacian Filter

Our desired pixel or the pixel of the porous which we desire to detect is standing out among almost all other pixels will distinctively high intensity value.

In order to detect the porous in the image precisely, we will have to normalize the and plot a "histogram " which will contain all the all the intensity values between 0 and 255 and the number of pixels of those intensity values.

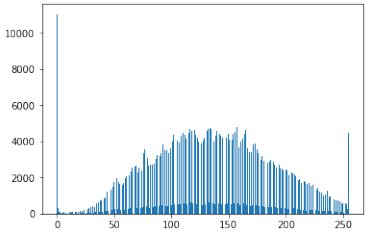


Figure 2.3: Histogram

From the above histogram we can clearly detect a drop in the number of pixels after the intensity of 128. There for a threshold was set at 128 ,i.e. only the pixels which have intensity greater than 128will be printed. The resulting image of that is the following in which the pixel of the porous is clearly visible.

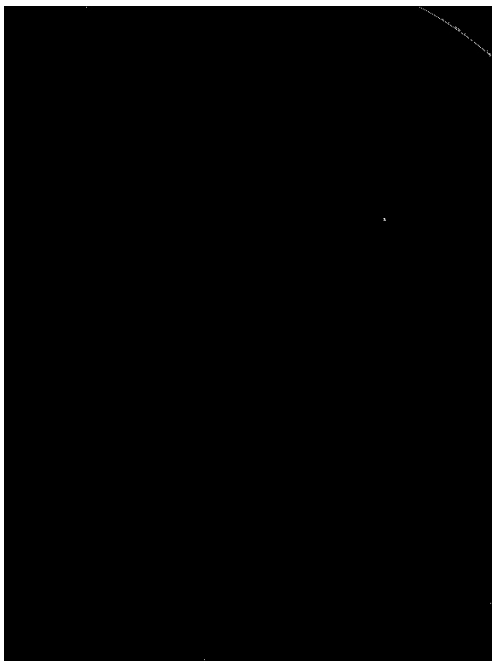


Figure 2.4: Output : Porosity Detected

2.1 IMAGE SEGMENTATION

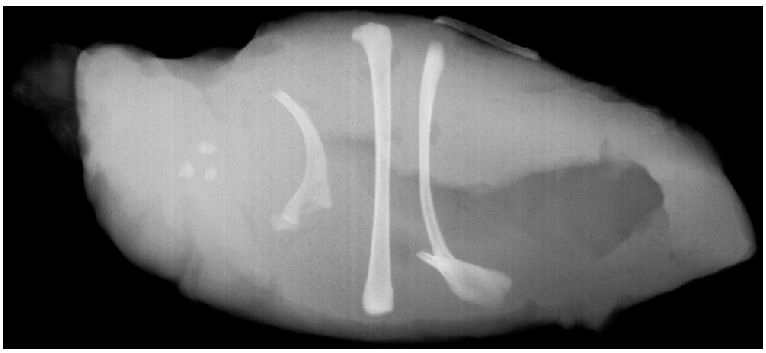


Figure 2.5: Input Image

In this problem the following steps have been followed:

- The Input image was loaded and has been converted into a binary image.
- An array with 255 elements was created with all zero values. The number of pixels with different intensity are then counted and the count is then stored in the array.
- Now the array contains the number of pixels with different intensity staring from 0 to 255. This array is then used to plot a histogram as shown below:

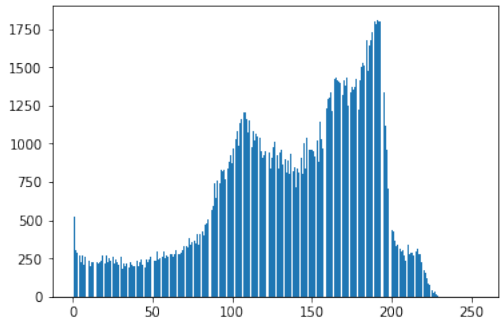


Figure 2.6: Output : Porosity Detected

From the above histogram it can be observed that there is a sudden drop in the number of pixels after the intensity value of 204. Therefore the pixels which had intensity value greater than 204 were plotted and the end result is shown in the below image:

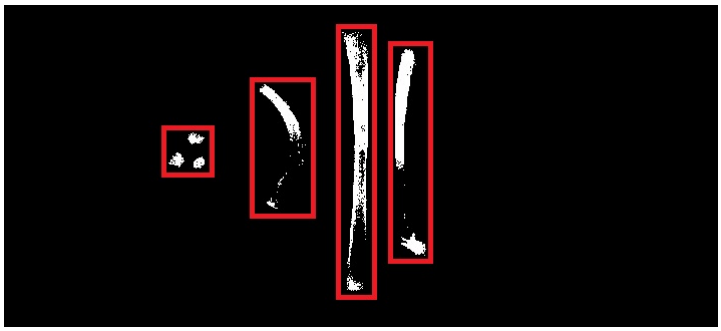


Figure 2.7: Histogram

The segments detected in this problem has been enclosed with the red boxes and the Coordinates for the four corner points of the bounding boxes are the followings (Boxes are mentioned from left to right):

Box 1	(207,169)	(256,169)	(207,217)	(256,217)
Box 2	(308,210)	(310,310)	(308,347)	(370,347)
Box 3	(369,291)	(406,291)	(369,562)	(406,562)
Box 4	(425,255)	(466,255)	(425,473)	(466,473)

3 HOUGH TRANSFORM

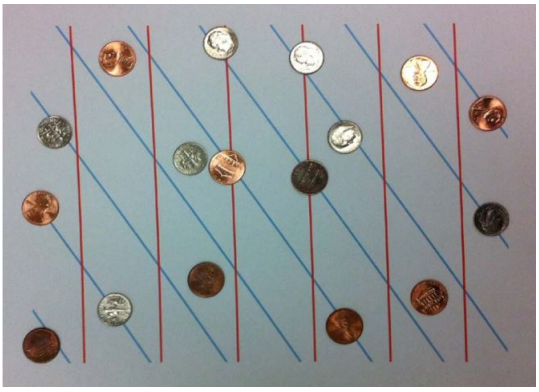


Figure 3.1: Input Image

3.1 WHAT IS HOUGH TRANSFORM?

The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing.[1] The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

3.2 STRAIGHT LINE DTECTION:

In this project Hough transform is used to detect straight lines. In general, the straight line y = mx + b can be represented as a point (b, m) in the parameter space.

However, vertical lines pose a problem. They would give rise to unbounded or infinite values of the slope parameter m. Thus, for computational reasons, we use the normal form of the straight lines, which is:

$$y = xCos\theta + ySin\theta$$

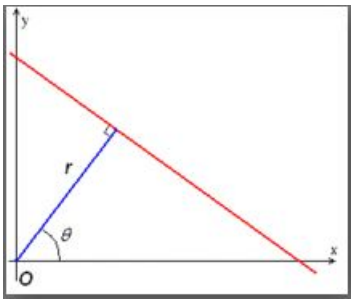


Figure 3.2: Straight Line Polar Equation

where r is the distance from the origin to the closest point on the straight line, and {θ} (theta) is the angle between the x axis and the line connecting the origin with that closest point.

It is therefore possible to associate with each line of the image a pair (r,θ). The (r,θ) plane is sometimes referred to as Hough space for the set of straight lines in two dimensions.

3.3 Implementation

In the process of implementing the straight line detection using Hough transform, the following steps are followed:

- The Input image was loaded and has been converted into a binary image.
- Then a vertical edge detection algorith was e=run on the input image and the output image was normalized using the following segment of code:

```
largest = 255
smallest = 0
r,c = img.shape

count = 0

img1 = np.zeros(img.shape)
kernel = [[-1,0,1],[-2,0,2],[-1,0,1]]
```

```
for x in range(1,r-1):
    for y in range(1,c-1):
        img1[x][y]= img[x-1][y-1] * kernel[0][0] + \
                    img[x-1][y] * kernel[0][1] + \
                    img[x-1][y+1] * kernel[0][2] + \
                    img[x][y-1] * kernel[1][0] + \
                    img[x][y] * kernel[1][1] + \
                    img[x][y+1] * kernel[1][2] + \
                    img[x+1][y-1] * kernel[2][0] + \
                    img[x+1][y] * kernel[2][1] + \
                    img[x+1][y+1] * kernel[2][2]

    if (img1[x][y] > largest):
        largest = img1[x][y]

for x in range(0,r):
    for y in range(0,c):
        img1[x][y] = (np.abs(img1[x][y]) / np.abs(largest))*255
        if (img1[x][y] > 80):
            img1[x][y] = 255
            count += 1
        else:
            img1[x][y] = 0

cv2.imwrite('edge.jpg', img1)
```

- The output after running the vertical edge detection algorithm is shown below:

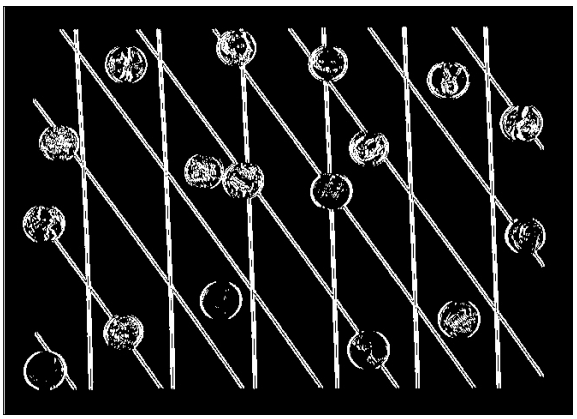


Figure 3.3: Edge Detected Image

- After this the accumulator matrix was created and initialized with all zeros. The accumulator matrix is supposed to take with r values in the y-axis and θ values in the x axis. The following is the accumulator saved in an image form:

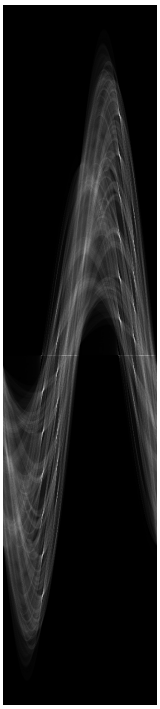


Figure 3.4: Accumulator

- The accumulator creates a phase signal that corresponds to the input frequency. In this case the input frequency (number of votes of each coordinate) creates a sinusoidal wave. The polar equation of a straight lines nothing but a representation of sine wave.
- Now an iteration was run on the entire image and for each non-zero pixel of the image was converted into 360 (r, θ) [for each value of θ from 0 to 360 or -180 to 180] pairs and in each iteration the voting was done on the accumulator matrix (The index for appropriate (r, θ) value was increased by 1). Since we are considering full sinusoidal wave, we cannot discard the negetive values of "r".
- following segment of code describes the creation of accumulator matrix and the voting process:

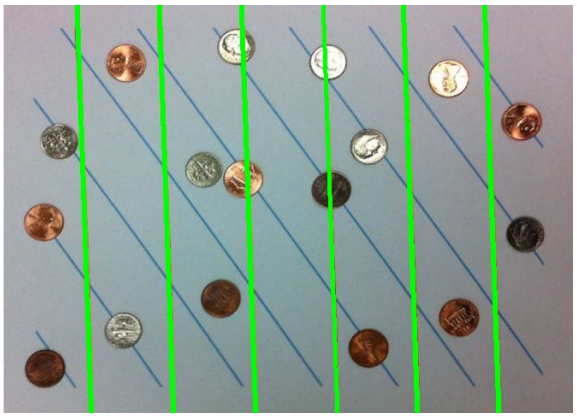
```
###<<<<<<----------------------------create accumulator---------------------------->>>>>>

d = np.sqrt((r**2)+(c**2))
d = int(d)
```

```
accumulator = np.zeros(((d*2),360))
print(accumulator.shape)


for x in range(0,r):
    for y in range(0,c):
        if (img1[x][y] == 255):
            for deg in range (-180,180):
                r = x*np.cos(np.deg2rad(deg)) + y*np.sin(np.deg2rad(deg))
                r = int(r)
                accumulator[r+d][deg] = accumulator[r+d][deg]+1

cv2.imwrite("accumulator.jpg",accumulator)

###<<<<<<----------------------selecting highest voted points------------------------>>>>>

idx = np.unravel_index(np.argsort(accumulator.ravel())[-3000:], accumulator.shape)
```

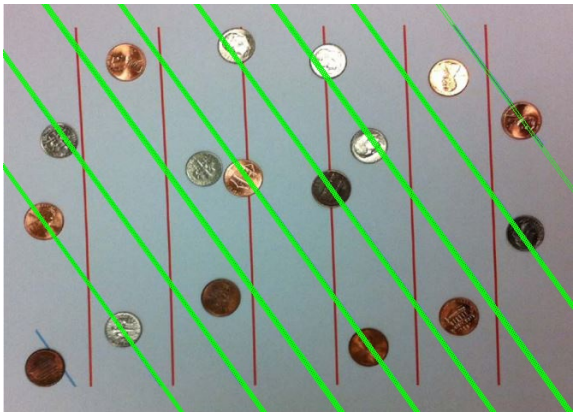
- After that, the highest voted coordinates (r, θ) were extracted and each (r, θ) in the polar space was then converted into numerous (x,y) points in the cartesian image space, using the following equation:

$$y = (y - xCos\theta)/Sin\theta$$

- The In the final step, the precise angles for the vertical lines and the diagonal lines were determined and all the resulting points for vertical and diagonal lines separately, on the original image.
- Two output images are shown below:



: Figure 3.5
red$_l ine.jpg$



: Figure 3.6
blue$_l ine.jpg$

4 Bonus Point

This is similar to the previous problem, in this problem, instead of straight lines, we are supposed to detect the circles in the input image. To do that, we have to consider the equation of a circle instead of that of the straight line:

$$(x-a)^2 + (y-b)^2 = r^2$$

where (a,b) is the coordinate of the center of the circle, and r is the distance of the center from the origin.
In the polar plain, the x and y can be represented as the following equations:
x = x - r.Cosθ
y = y - r.Sinθ
All steps followed after this was same as the straight line detection algorithm:

1. The image was loaded and converted into grey-scale form.
2. Edges were detected using "horizontal" edge detection algorithm.
3. Then the image was normalized and a threshold was applied, above which all the values were set to 255 and below which all the pixels were set to zero.
4. This step was followed in order to turn the image into binary form and to remove all the straight lines and to keep only the circles. After this step, the image was like the following:

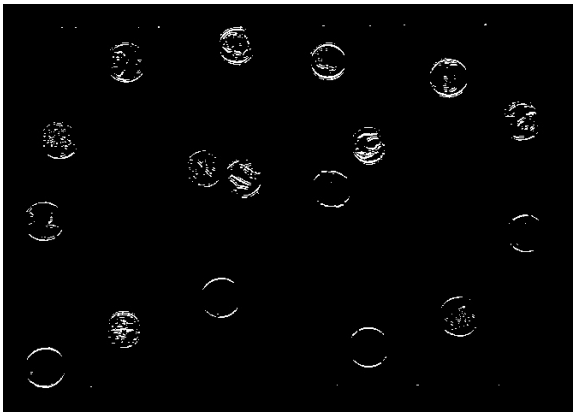


Figure 4.1: Edge Detected Image

5. Then the accumulator was created, but the difference is that instead of (r, θ) value the voting will be done using (a,b) values. Then the maximum voted points were extracted. The following is the image of the accumulator saved as an image:

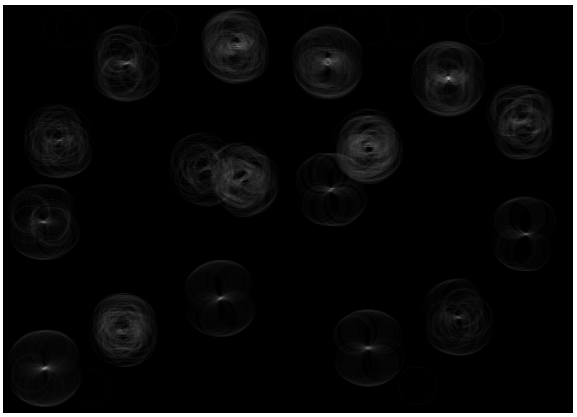


Figure 4.2: Accumulator

6. These points will be the coordinates of the centers of the circles. Thus circles with fixed diameter were plotted around these extracted coordinates and the following output image was generated:

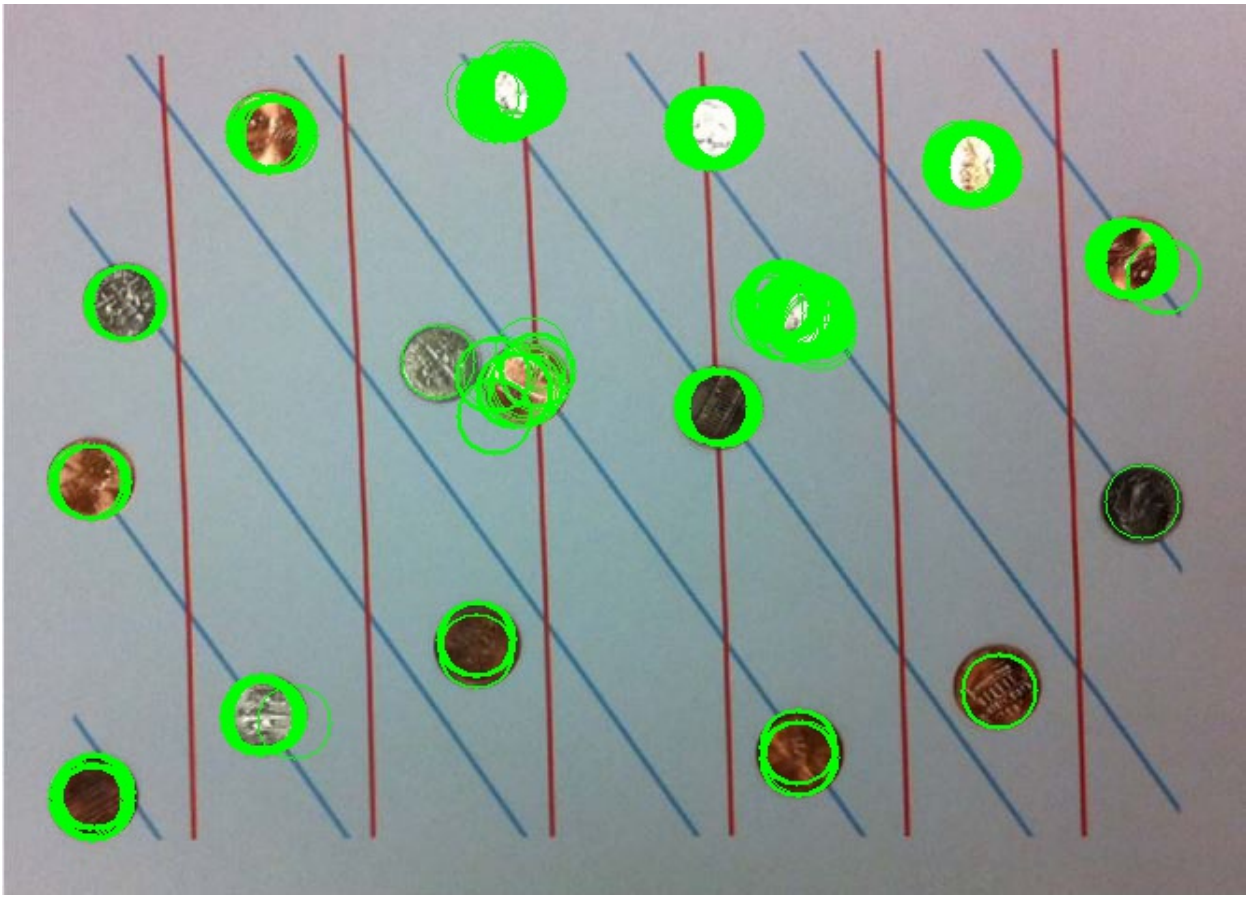


Figure 4.3: coin.jpg