CSE 573: Introduction to Computer Vision and Image Processing

Instructor: Dr. Junsong Yuan
University at Buffalo

Project 3 Report

Submitted By:
Shreyas Narasimha (UB Person Number: 50289736)

Task 1 – Morphology and Image Processing

Objective: To remove noise from a given binary image using two morphology image processing algorithms.

Input:

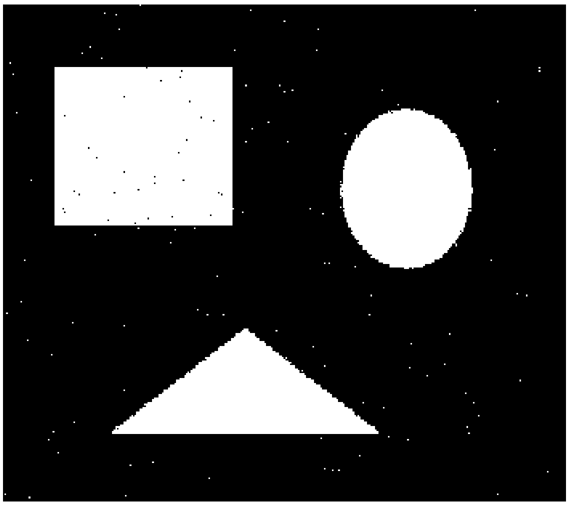

Fig. 1 – A noisy input image

Implementation and Output:

- We use two morphological algorithms **Dilation** and **Erosion**.
- The two algorithms are performed one after the order to perform the morphological operations **Opening** and **Closing**.
- We perform these operations to remove the noise and find the boundaries of the image elements as well.

Erosion:

- To perform erosion, we use a 3x3 kernel containing 255(white).
- The kernel is passed over the entire binary image and only those pixels are left white which allow the pixel centred kernel to match the image pixels behind it completely.

```
def erode(image):
   kernel = np.ones((3,3))
   kernel = kernel*255
   h = int(image.shape[0])
   w = int(image.shape[1])
   temp = image.copy()
   temp = temp*0
   for i in range(1,h-1):
   for j in range(1,w-1):
   if(np.array_equal(kernel,image[i-1:i+2,j-1:j+2])):
   temp[i][j] = 255
   else:
   temp[i][j] = 0
   return temp
```

Dilation:

- To perform dilation, we use a 3x3 kernel containing 255.
- The kernel is passed over the entire image and wherever a white pixel is found, all the pixels around it of the size of the kernel are turned white.

```
def dilate(image):
    h = int(image.shape[0])
    w = int(image.shape[1])
    temp = image.copy()
    temp = temp*0
    for i in range(1,h-1):
        for j in range(1,w-1):
            if(image[i][j] == 255):
                temp[i-1:i+2,j-1:j+2] = 255
    return temp
```

Opening:

- Involves performing erosion and then dilation.

Closing:

- Involves performing dilation and then erosion.

The steps performed are :-

1) Read the image in grayscale using `cv2.imread()`.

2) The grayscale image is **converted** into a binary image using threshold as 127.

3) **Opening** is performed in the image :-

    a. **Opening** refers to performing **Erosion** first and then **Dilating** the Eroded image.

4) **Closing** is performed on the image separately.
    a. **Closing** refers to performing, **Dilation** and then **Erosion.**

5) The **boundary** of the image is found by Eroding the image and then subtracting the binary image with its eroded counterpart.

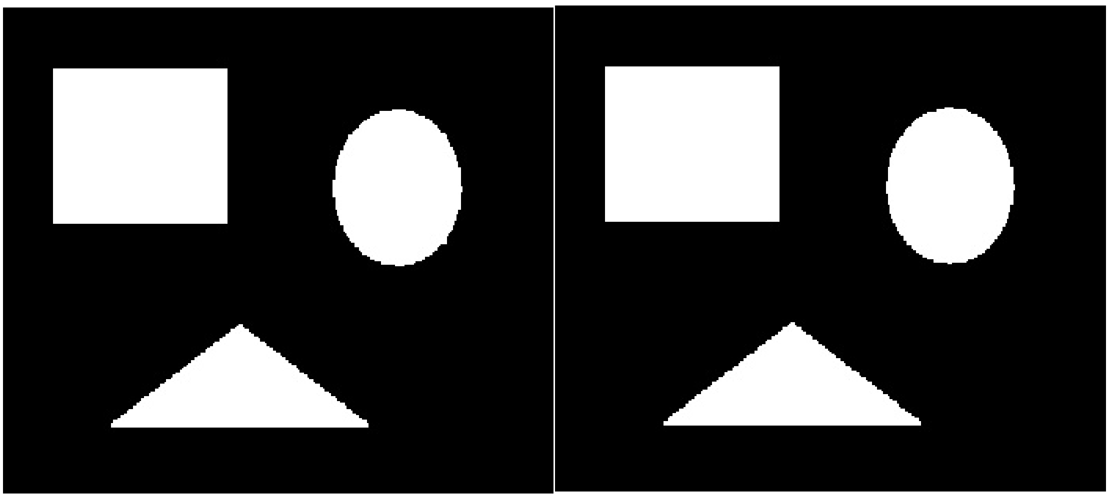

Fig. 2 – Noise removed using a) closing and then opening; b) opening and then closing

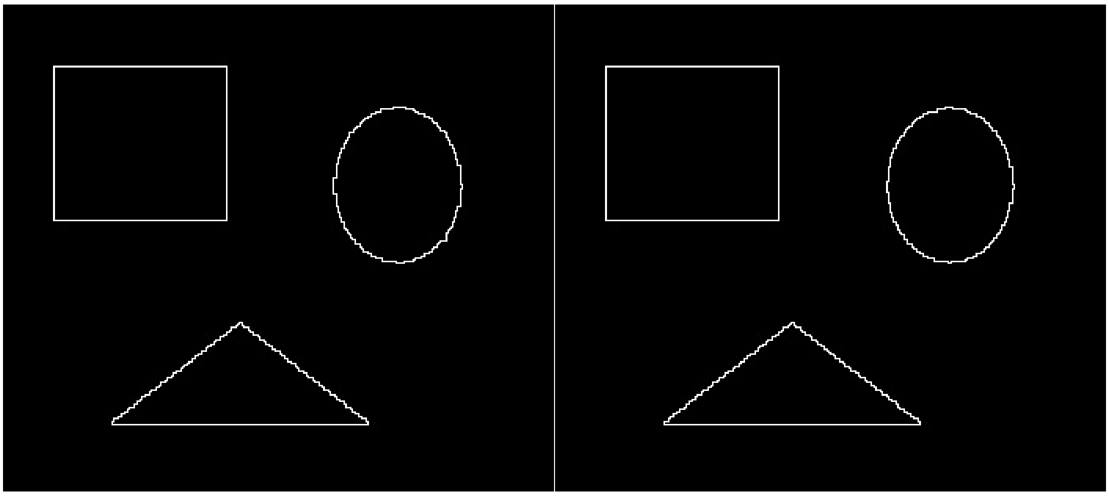

Fig. 3 – Finding the boundary using the results of the previous step

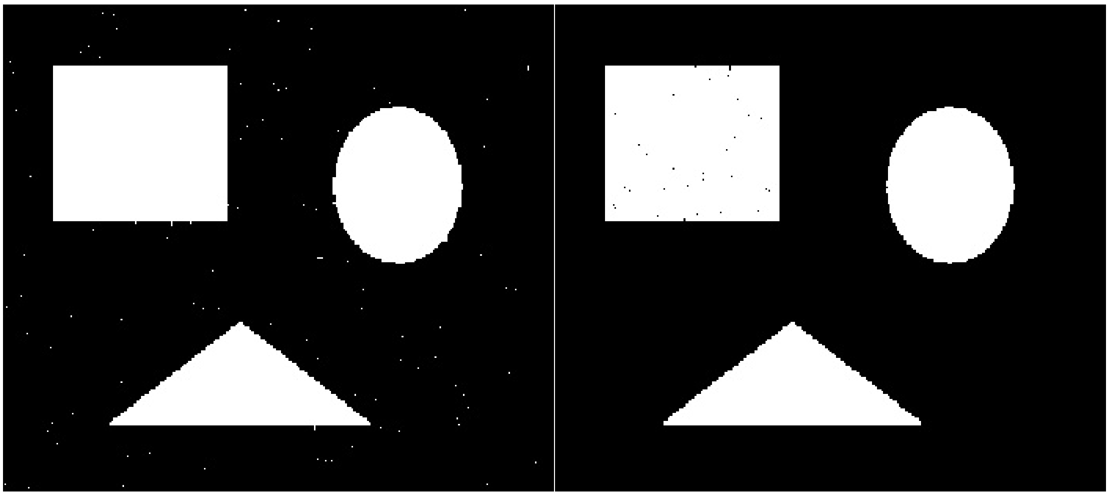

Fig. 4 – Using only a) closing b)opening

Comparison:

- Thus, we see that the outputs of the noise removal are the same.
- We also see that, when only one of each operation is performed, the outputs are different.
- **Closing** is good at removing noise in the background.
- **Opening** is good at removing noise from elements.

Task 2 – Image segmentation and point detection

Objective:

- To detect points in a given image using a point detection algorithm.
- To segment an image using thresholding.

Input:

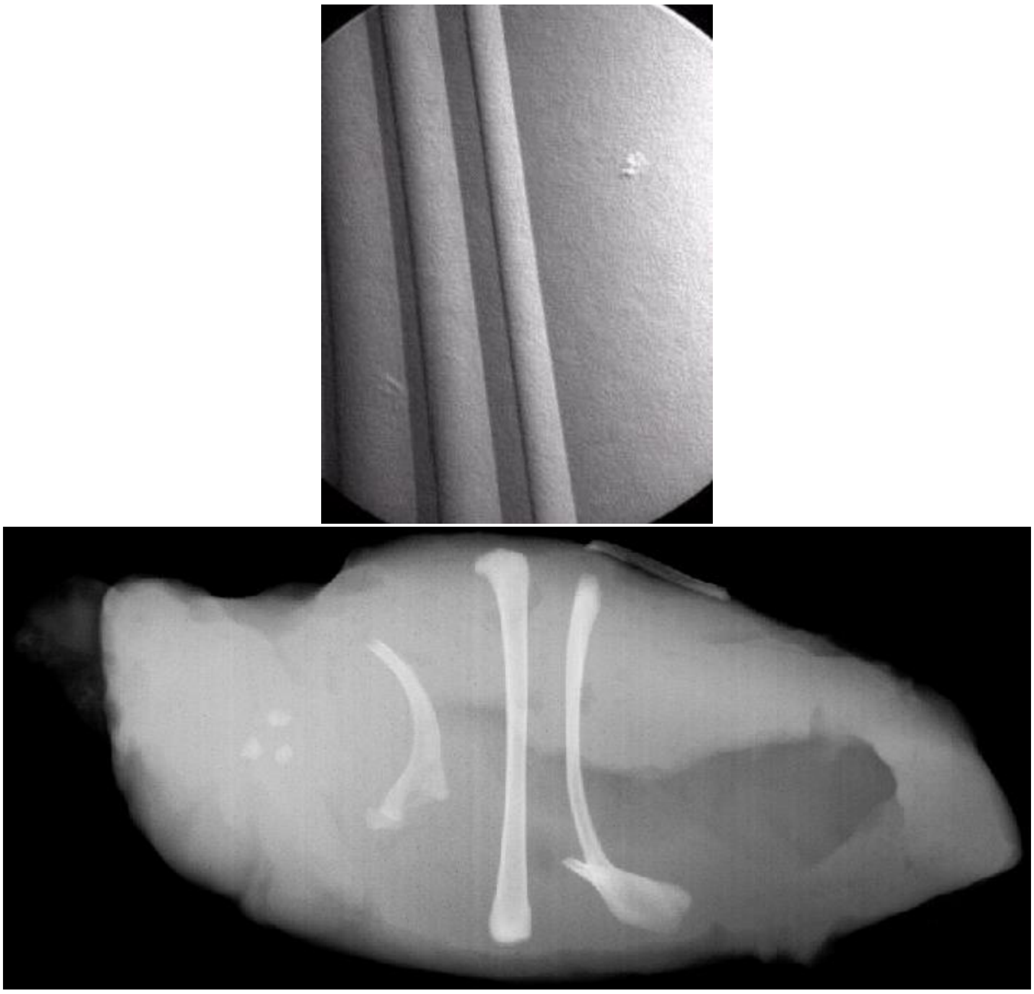

Fig. 5 – Input images

Implementation and Output:

<u>Point Detection</u>:

1) Read the image using `cv2.imread()`.
2) Create a square matrix 5x5 kernel with all values as -1 except the centre pixel which takes the value of square of the kernel size minus one, i.e 24, here.
3) We then convolve the image with the kernel, (the flipping involved doesn't matter since the kernel is symmetric).
4) This results in detecting points after the convolved image is thresholded.
5) We then dilate the image to make the detected point more visible.
6) Kernel Used:

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	24	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

```
def detect_points(image,ks):
    h = int(image.shape[0])
    w = int(image.shape[1])
    temp = image.copy()
    temp = temp*0
    for i in range(int(ks/2),h-int(ks/2)):
        for j in range(int(ks/2),w-int(ks/2)):
            inter = image[i-int(ks/2):i+int(ks/2)+1, j-
int(ks/2):j+int(ks/2)+1]*kernel
            center = np.absolute(np.sum(inter))
            #temp[i][j] = center
            if(center > 2300):
                temp[i][j] = center
return temp
```

Segmentation:

1) Read the image using `cv2.imread()`.
2) We use thresholding to segment the image.
3) To find the optimal global threshold, we use the **heuristic thresholding algorithm**.
4) This threshold is applied on the image to segment it into its different elements.
5) Heuristic thresholding algorithm is as follows:-
    a. Select an initial threshold T.
    b. Segment the image using T. The two groups produced are then averaged, i.e, their pixel values are averages and taken as the new threshold.
    c. The new threshold is calculated as half the average.
    d. The steps are repeated till there is less than a threshold amount of change in T.
6) We then use template matching to find bounding boxes for the detected objects.

```
def segment(image):
    h = int(image.shape[0])
    w = int(image.shape[1])
    t_next = 10
    t1 = 0
    dif = 999
    temp = image.copy() *0
    while(dif > 10):
        t1 = t_next
        count1 = 0
        count2 = 0
        sum1 = 0
        sum2 = 0
```

```
    for i in range(h):
        for j in range(w):
            if(image[i][j] > t1):
                count1 = count1+1
                sum1+= image[i][j]
            else:
                count2 = count2+1
                sum2+= image[i][j]
    if(count1!=0):
        m1 = sum1/(1.0*count1)
    else:
        m1 = 0
    t_next = m1
    dif = t_next - t1
    #print(t_next)

for i in range(h):
        for j in range(w):
            if(image[i][j] > t_next):
                temp[i][j] = 255
#cv2.line(image,(5,5),(45,45),(255,0,0),5)
return temp
```

Outputs:

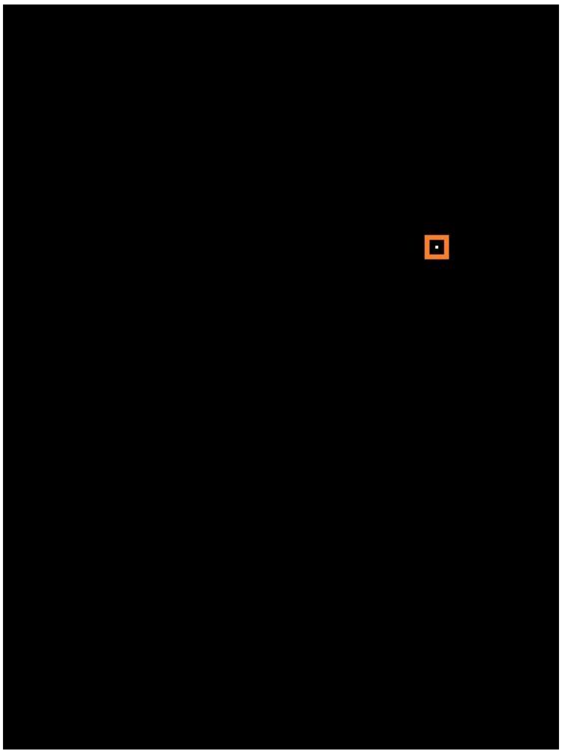

Fig. 6 – Point detection

- The point detected is at position [249,445]

Fig. 7 – Segmentation

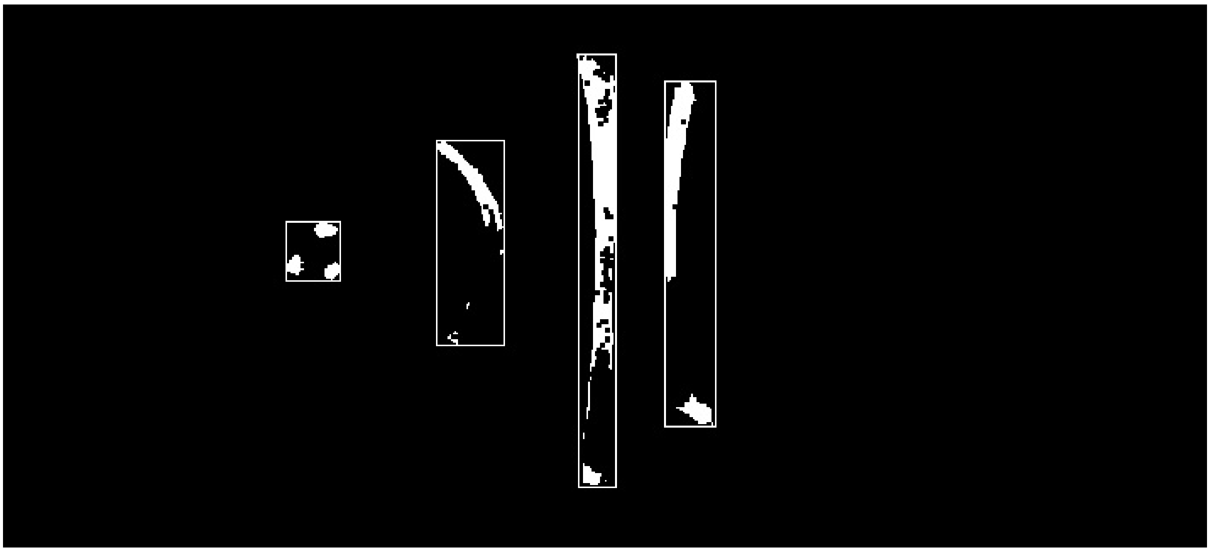

Fig. 8 – Segmented image with bounding boxes

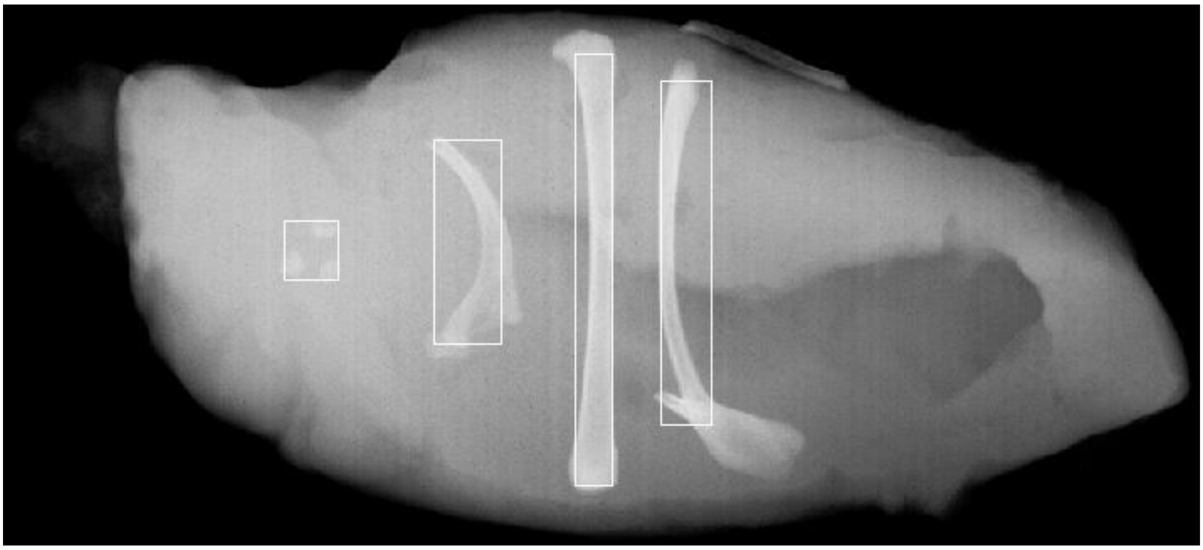

Fig. 9 – Bounding boxes on the input image

```
For object 0

The top left corner(column,row) (167 128)
The top right corner(column,row) (199 128)
The bottom left corner(column,row) (167 163)
The bottom right corner(column,row) (199 163)


For object 1

The top left corner(column,row) (256 80)
The top right corner(column,row) (296 80)
The bottom left corner(column,row) (256 201)
The bottom right corner(column,row) (296 201)


For object 2

The top left corner(column,row) (340 29)
The top right corner(column,row) (362 29)
The bottom left corner(column,row) (340 285)
The bottom right corner(column,row) (362 285)


For object 3

The top left corner(column,row) (391 45)
The top right corner(column,row) (421 45)
The bottom left corner(column,row) (391 249)
The bottom right corner(column,row) (421 249)
```

Fig. 10 – The positions of the bounding boxes

Task 3 – Line Detection using Hough Transform

Objective: To design and implement an algorithm to detect **lines** and **circles** using the Hough transform.

Input:

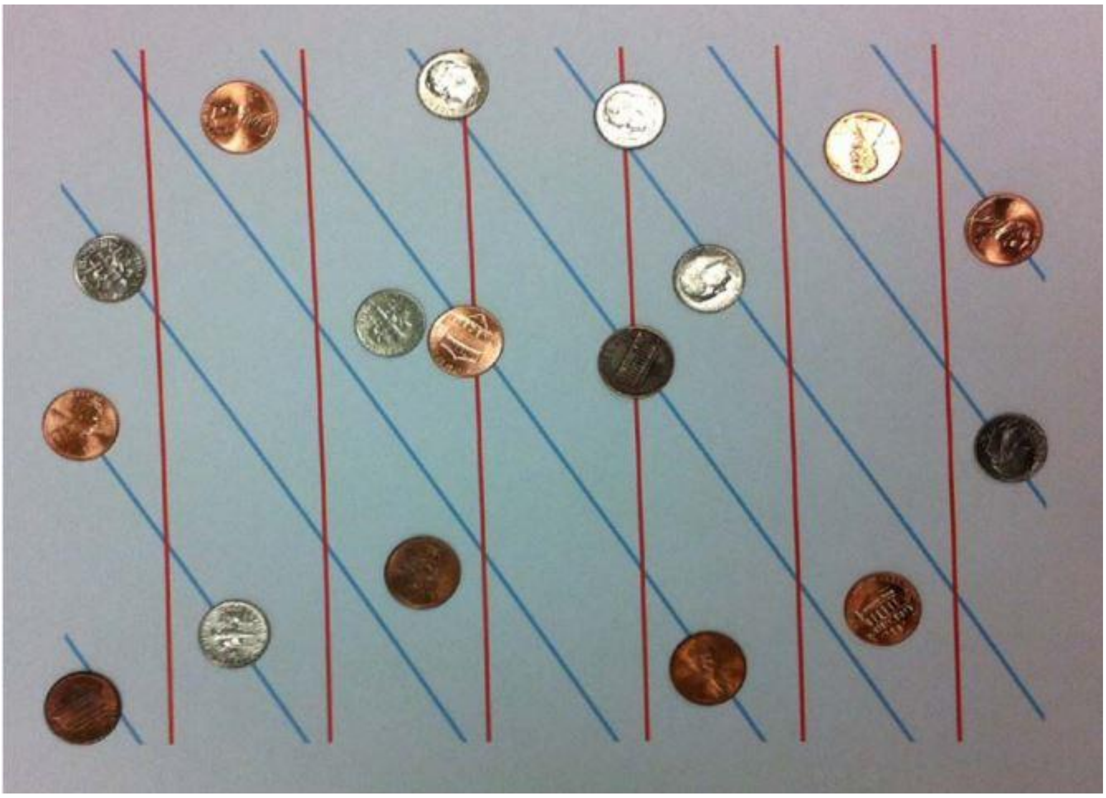

Fig. 11 – Input image

Implementation and Outputs:

1) Read the image using `cv2.imread()`.
2) Convert the input image to an edge detected image. We use a manually written sobel function to do this.
3) **Close, Open** and **Erode** the image to make it more suitable for line detection.
4) Iterate over the image and for all white pixels find all lines passing through it.
5) All these lines increment values corresponding to the (r,ɵ) pairs in an accumulator. Every line can be written of the form $x \cos \theta + y \sin \theta = r$.
6) We then select the lines which have more than a selected threshold in their respective accumulator entry. These are converted into their x, y counterpart and then to lines.
7) There will be several lines and these are consolidated and filtered depending on their angle with the x-axis(ɵ).
8) Similarly, circles are detected and drawn using the circle equation and a 3-D accumulator containing a, b and r (circle equation is ($a = x - r \cos \theta$, $b = y - r \sin \theta$).

```
def hough(image,img):
    h,w = image.shape
    temp = image.copy()
    img1 = img.copy()
    img2 = img.copy()
    points = []
    #temp = cv2.Canny(img,100,200)
    temp = cv2.GaussianBlur(image,(3,3),5)
```

```
    temp = edge_detect(temp)

    temp = erode(opening(closing(temp)))
    angle = 90
    a = np.zeros((int(2*np.sqrt(np.square(h)+np.square(w))),(2*angle)+1))
    for i in range(h):
        for j in range(w):
            #print(temp.shape)
            if(temp[i,j] > 0):
                for k in range(-angle,angle+1):
                    r = (j*np.cos(np.radians(k)) +
i*np.sin(np.radians(k)))+ np.sqrt(np.square(h)+np.square(w))
                    a[int(r),k+angle]+=1

    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            if(a[i,j] > 130):
                points.append([i,j])

    vpoints = []
    spoints = []
    for i in points:
         if( -2 <= i[1]-90 <= -2):
             vpoints.append(i)
         elif((-37 <= i[1]-90 <= -36)):
                spoints.append(i)

    v2 = med(np.asarray(vpoints))
    v1 = med(np.asarray(spoints))
    for i in v2:
        a1 = np.cos(np.radians(i[1]-90))
        b1 = np.sin(np.radians(i[1]-90))
        x0 = a1*(i[0] - np.sqrt(np.square(h)+np.square(w)))
        y0 = b1*(i[0] - np.sqrt(np.square(h)+np.square(w)))
        x1 = int(x0 + 1000*(-b1))
        y1 = int(y0 + 1000*(a1))
        x2 = int(x0 - 1000*(-b1))
        y2 = int(y0 - 1000*(a1))
        cv2.line(img2,(x1,y1),(x2,y2),(0,0,255),2)

    for i in v1:
        a1 = np.cos(np.radians(i[1]-90))
        b1 = np.sin(np.radians(i[1]-90))
        x0 = a1*(i[0] - np.sqrt(np.square(h)+np.square(w)))
        y0 = b1*(i[0] - np.sqrt(np.square(h)+np.square(w)))
        x1 = int(x0 + 1000*(-b1))
        y1 = int(y0 + 1000*(a1))
        x2 = int(x0 - 1000*(-b1))
        y2 = int(y0 - 1000*(a1))
        cv2.line(img1,(x1,y1),(x2,y2),(255,0,0),2)

    return img1,img2
```

For circles:-

```
def chough(image,img):
    h,w = image.shape
    temp = image.copy()
    img1 = img.copy()
    #temp = cv2.GaussianBlur(image,(3,3),2)
    temp = edge_detect(temp)
    #temp = closing(temp)
    temp = dilate(temp)
    angle = 360
    diagonal = np.sqrt(np.square(h)+np.square(w))
    acc = np.zeros((w,h,int(diagonal)))
    #r = 24
    for i in range(h):
        for j in range(w):
            if(temp[i,j] > 200):
                for r in range(23,25):
                    for k in range(angle):
                        a = j - r*np.cos(np.deg2rad(k))
                        b = i - r*np.sin(np.deg2rad(k))
                        if((a>=(w-1)) or (b>=(h-1))):
                            continue
                        else:
                            acc[int(a),int(b),r]+=1
    print(acc)
    points = []
    for i in range(acc.shape[0]):
        for j in range(acc.shape[1]):
            for k in range(acc.shape[2]):
                if(acc[i,j,k] > 330):
                    points.append([i,j,k])
                    #print(acc[i,j,k])
    #points = circon(np.asarray(points))
    points = unique(points)
    for i in points:
        cv2.circle(img1,(int(i[0]),int(i[1])),int(i[2]),(0,255,0),thickness
= 2)
    return img1
```

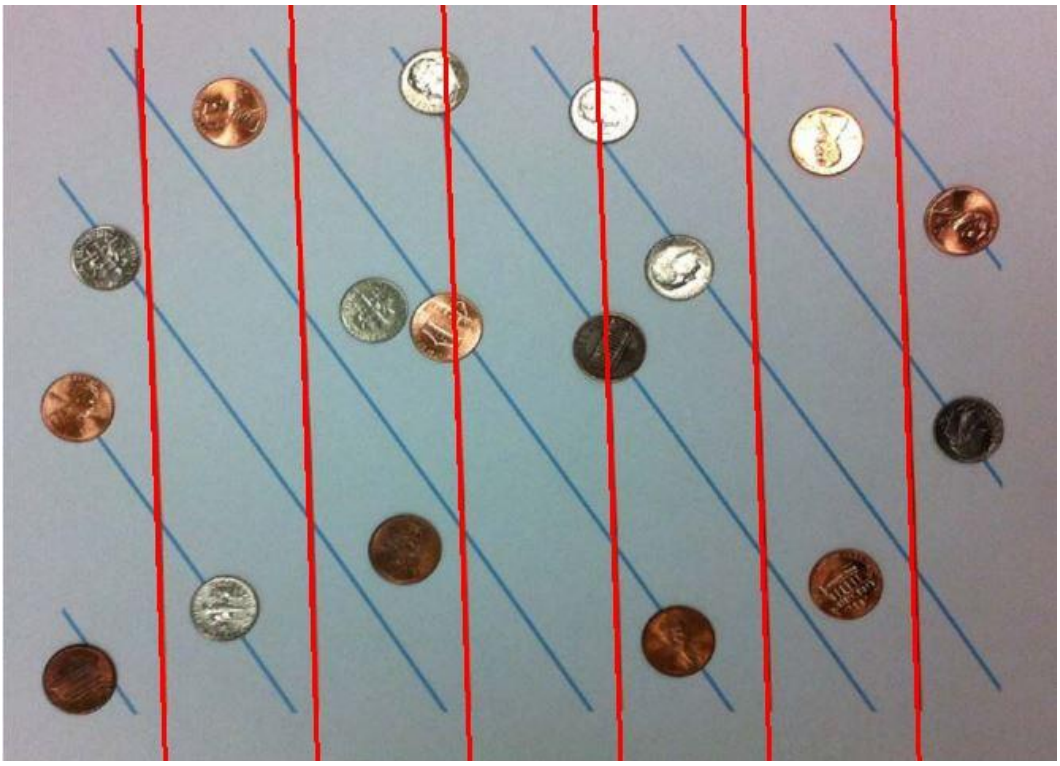

Fig. 12 – Red lines detected

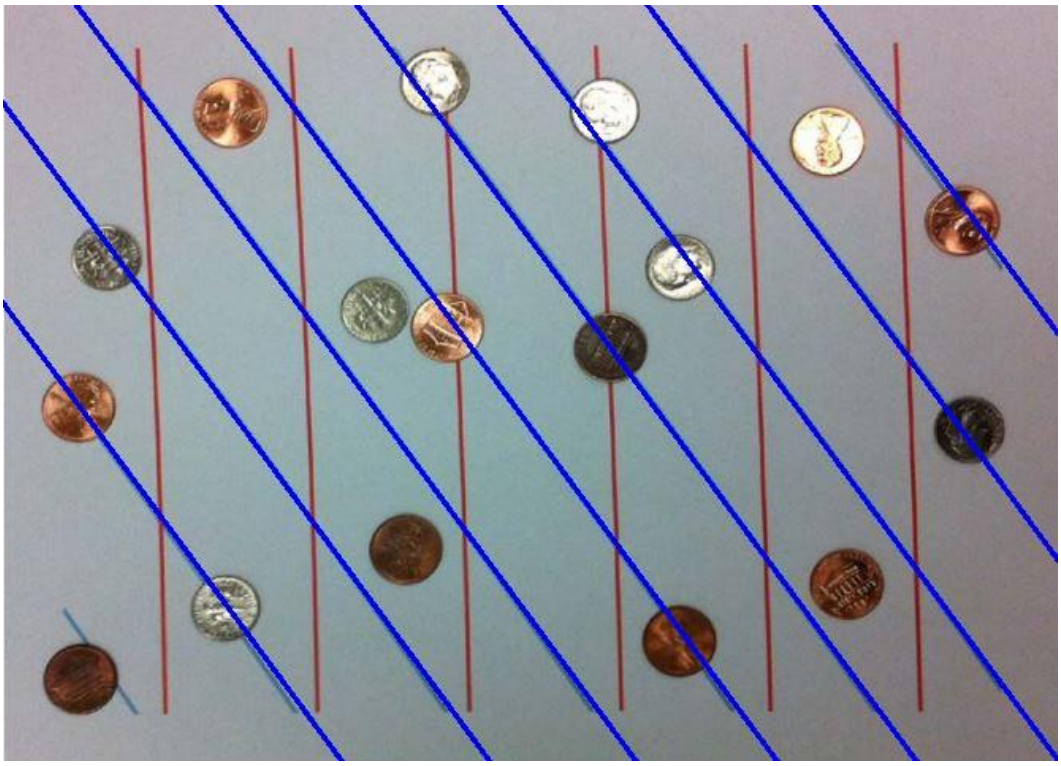

Fig. 13 – Blue lines detected

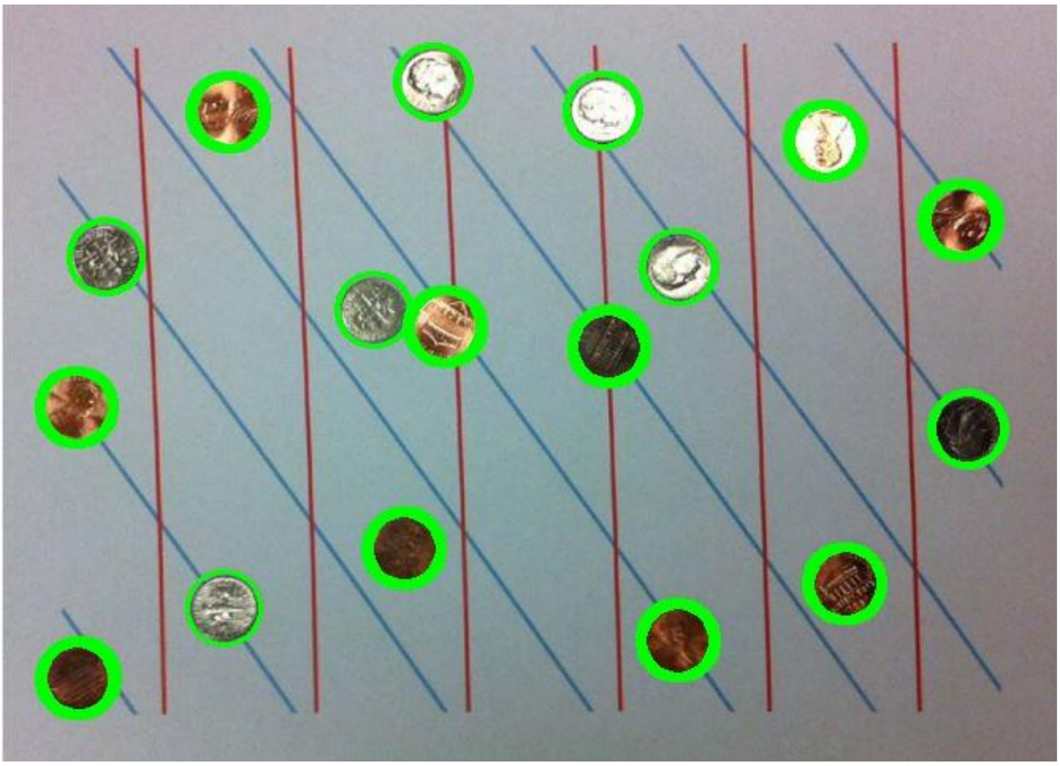

Fig. 14 – Coins detected

More on Hough Transform:-

- A line is a collection of points.
- We represent the line as a point in the mc – space.
- A line can be written as $y = mx + c$
- Thus, one pair of m and c represents an entire line in the xy – plane.
- For, each point, all the lines passing through the point are represented in the mc – space.
- All these lines (-90 degrees to 90 degrees) are represented as a set of points in the mc – space, i.e, another line in the mc space.
- This is done for all the points in the edge detected image to get a mc – space with several lines, the intersection point of these lines represents the line in the xy – plane which is common to the points.
- Using the Cartesian equation poses a problem, i.e, it is not applicable when the line is vertical, so we use the polar equation, i.e, $x \cos \theta + y \sin \theta = r$
- This, turns the mc – space to the r, θ space where the lines instead become sinusoids. The sinusoid with the maximum intensity represents the detected line in the xy – plane.
- This is done by incrementing the r, θ counter in an accumulator whenever it is encountered and then selecting the r, θ with the maximum values.
- A similar rationale is used for circle detection but with the circle equation as mentioned before.

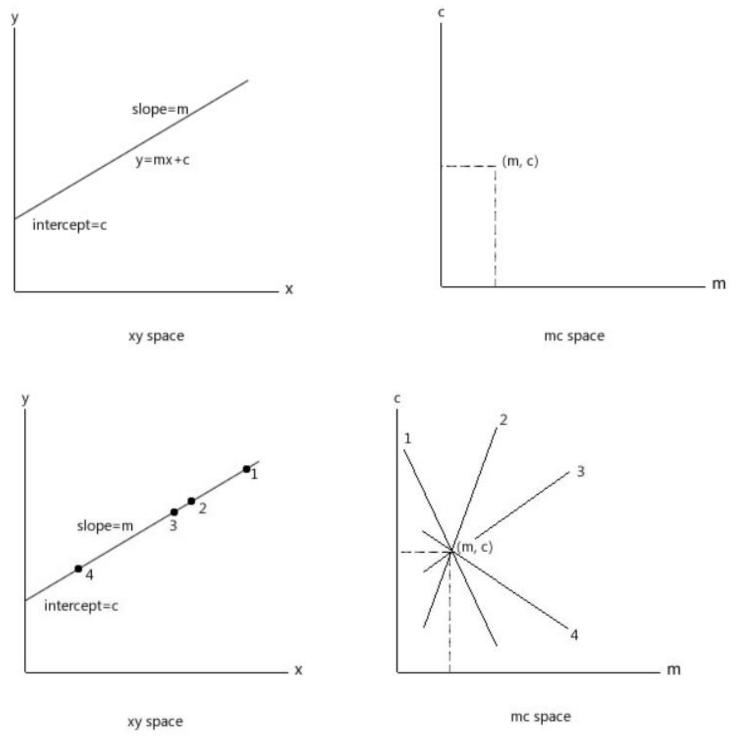

Fig. 15 – Line in the xy plane represented as a point in the mc Hough space

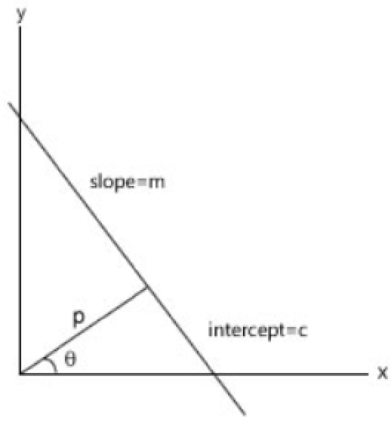

Fig. 16 – The r, ɵ representation of a line

Conclusion:

- We learned how to:-
  - Remove noise from a binary image by using morphological operations.
  - Find the boundaries of an image using morphological operations.
  - Detect points within an image.
  - Segment an image i.e, separate the background and different parts of the foreground.
  - Use Hough transform to detect lines and circles in a given image.

References

1) https://stackoverflow.com/questions/7624765/converting-an-opencv-image-to-black-and-white

2) https://docs.opencv.org/3.1.0/dc/da5/tutorial_py_drawing_functions.html

3) https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.mean.html

4) https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html

5) https://www.youtube.com/watch?v=4zHbI-fFIlI - Thales Sehn Körting

6) https://stackoverflow.com/questions/28077733/numpy-sin-function-in-degrees

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