**Submitted By**: Anupriya Goyal
**UB Person No.**: 50287108
**UBITName** : anupriya
**Date**: December 3rd ,2018

# Computer Vision and Image Processing (CSE473/573)
# Report for Project 3

## Task1: Morphology Image Processing

### Task1 Part1: Resultant images, explanation of approach used and code snippet for both algorithms:

Two morphology image processing algorithms to remove noises of the image are :

1. Closing followed by one time erosion to remove noise completely
2. Opening followed by one time dilation to remove noise completely

**Explanation when Algorithm 1 and Algorithm2 are applied to image i.e noise.jpg**

1. First read the image i.e noise.jpg (original image in gray scale). Then construct a structuring element which contain all values 1.
   str_elm =[[255,255,255],[255,255,255],[255,255,255]]
   str_elm=np.asarray(str_elm)
2. For closing operation apply dilation followed by erosion. The origin of structuring element is at center so if there is any pixel match with the structuring element then the copy image of original image pixel is set to 255 value otherwise it is set to 0 value.
3. **Algorithm1**:For erosion operation the function is created which will set the pixel value in copy image of original image to 255 only when all pixels in structuring element matches with the pixel in above dilated image otherwise it is set to 0.
4. The two functions that are created i.e dilation(image) and erosion(image) take image as parameter are called again for one more time to perform erosion so as to remove noise from image.
5. **Algorithm2:** Step1 and step2 are followed as it is **.** After that opening operation is performed. For opening operation apply erosion followed by dilation.
6. The two functions that are created i.e erosion(image) and dilation(image) take image as parameter are called again for one more time to perform dilation so as to remove noise from image.

   **Dilation Function:**

```
def dilation(image):
    image1 = image.copy()
    for i in range (1,image.shape[0]-1):
        for j in range (1,image.shape[1]-1):
```

```
            if image[i][j]==str_elm[1][1] or image[i][j-1]==str_elm[1][0] or
image[i][j+1]==str_elm[1][2] or image[i-1][j]==str_elm[0][1] or image[i-1][j-
1]==str_elm[0][0] or image[i-1][j+1]==str_elm[0][2] or image[i+1][j]==str_elm[2][1] or
image[i+1][j-1]==str_elm[2][0] or image[i+1][j+1]==str_elm[2][2]:
                image1[i][j]=255

            else:
                image1[i][j]=0

    return image1
```

**Erosion Fuction**

```
#implemention function for erosion
def erosion(image2):
        image3 = image2.copy()


        for i in range (1,image2.shape[0]-1):
            for j in range (1,image2.shape[1]-1):

                if  image2[i][j]==str_elm[1][1] and image2[i][j-1]==str_elm[1][0] and
image2[i][j+1]==str_elm[1][2] and image2[i-1][j]==str_elm[0][1] and image2[i-1][j-
1]==str_elm[0][0] and image2[i-1][j+1]==str_elm[0][2] and
image2[i+1][j]==str_elm[2][1] and image2[i+1][j-1]==str_elm[2][0] and
image2[i+1][j+1]==str_elm[2][2]:
                        image3[i][j]=255
                else:

                        image3[i][j]=0

        return image3
```

**Results when Algorithm 1 i.e closing followed by one time erosion is applied are as follows:**

1. **Closing morphology operation intermediator steps results:**

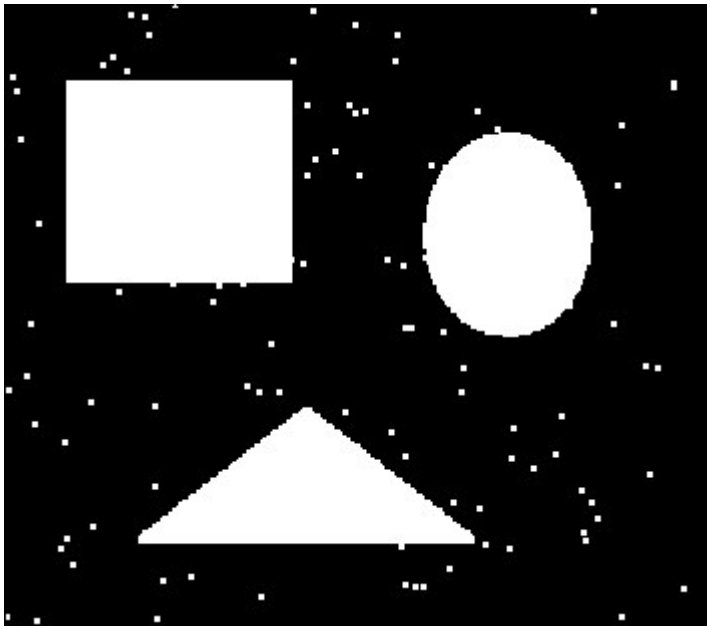

**Figure1. After dilation on original image**

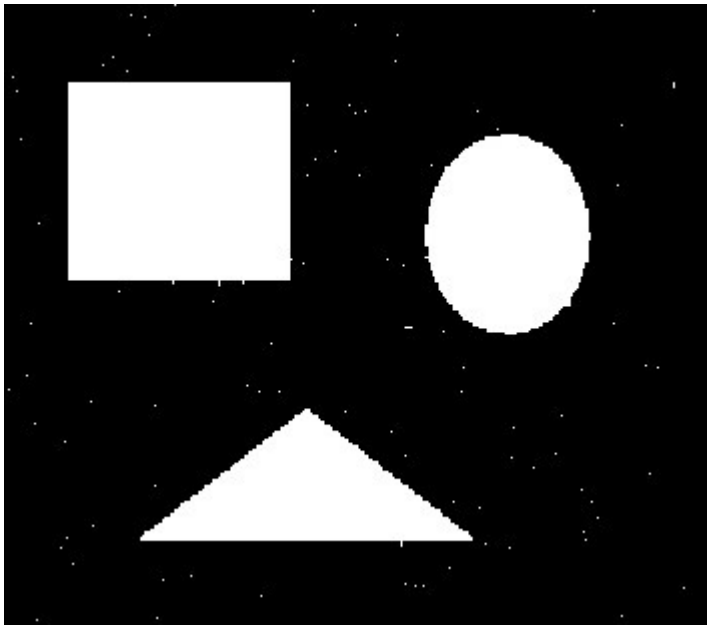

**Figure2: After applying erosion on above dilated image. This completes closing operation**

**Figure3: After applying erosion again on above eroded image. This gives noise removed image.(res_noise1.jpg)**

**Results when Algorithm 2 i.e opening followed by one time dilation is applied are as follows:**

2. **Opening morphology operation intermediator steps results:**

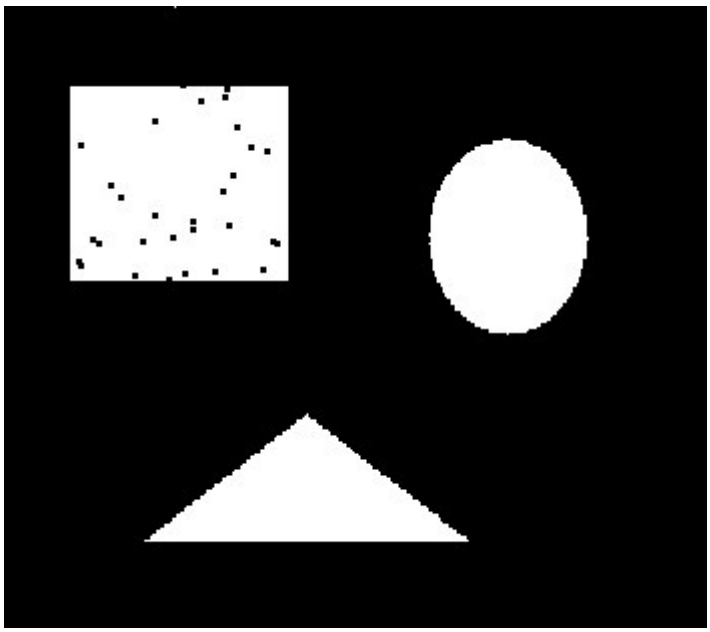

**Figure1. After erosion on original image**

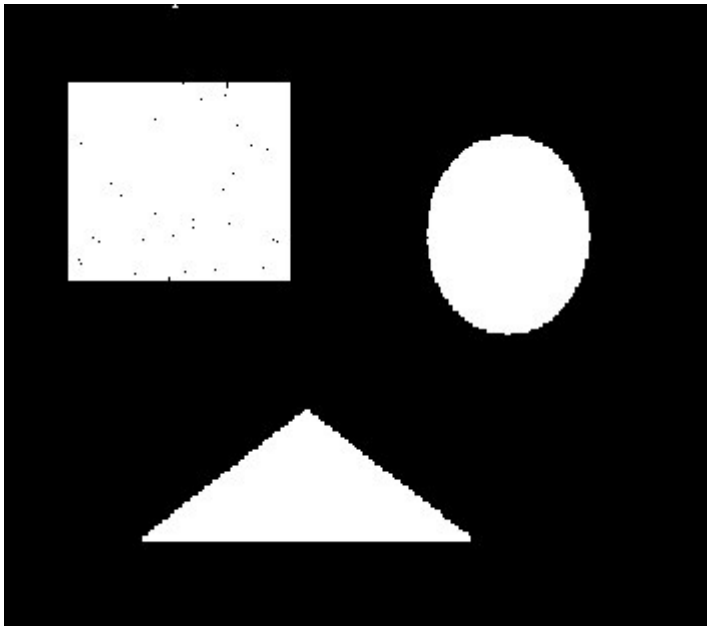

**Figure2: After applying dilation on above eroded image. This completes opening operation**

**Figure3: After applying dilation again on above dilated image. This gives noise removed image.(res_noise2.jpg)**

**Task1 part2:**

The images obtained after applying both algorithms have slight difference. The difference can be seen in corners . In the image res_noise2 (obtained by opening operation) the corners of triangle have become rounded. They have got flattened. While the corners in res_noise1 are sharp and rounded inward(obtained by closing operation).

The opening operation tends to flatten the sharp peninsular projections on the object.

ln morphological opening erosion operation removes objects that are smaller than structuring element B and dilation operation restores the shape of remaining objects. The opening by reconstruction method is able to restore the objects completely after erosion applied.

Closing is the complementary operation of opening, defined as dilation followed by erosion. Inward pointing corners are rounded, where the outward pointing corners remain unchanged.

**Task1 Part3:**

**Extracting boundaries using morphology operations:**

The boundary from both noise removed images is obtained by performing erosion on that image and subtracting the eroded image from noise removed image

**Boundary extracted code snippet is given as follows:**

```
img4=erosion(img3)
img5=img3-img4
cv2.imwrite("res_bound1.jpg",img5)
img9=erosion(img8)
img10=img8-img9
cv2.imwrite("res_bound2.jpg",img10)
```

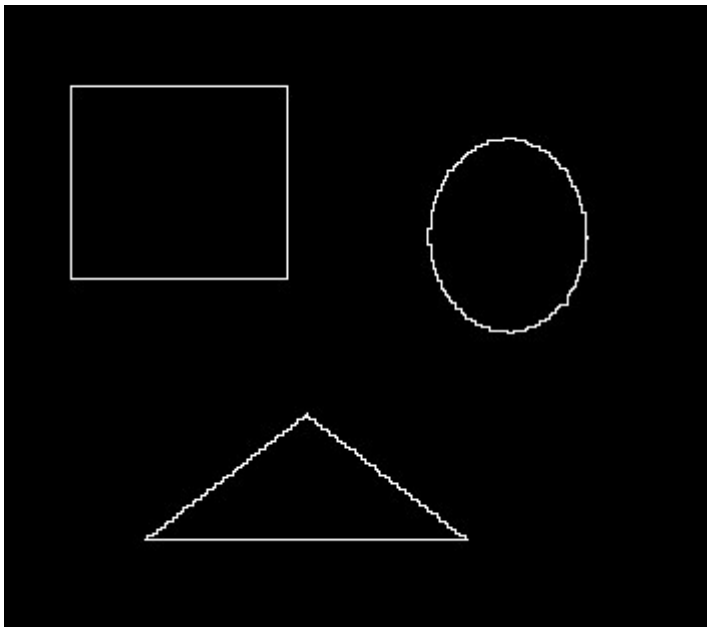

**Figure 4: Boundary extracted from res_noise1.jpg image(res_bound1.jpg)**

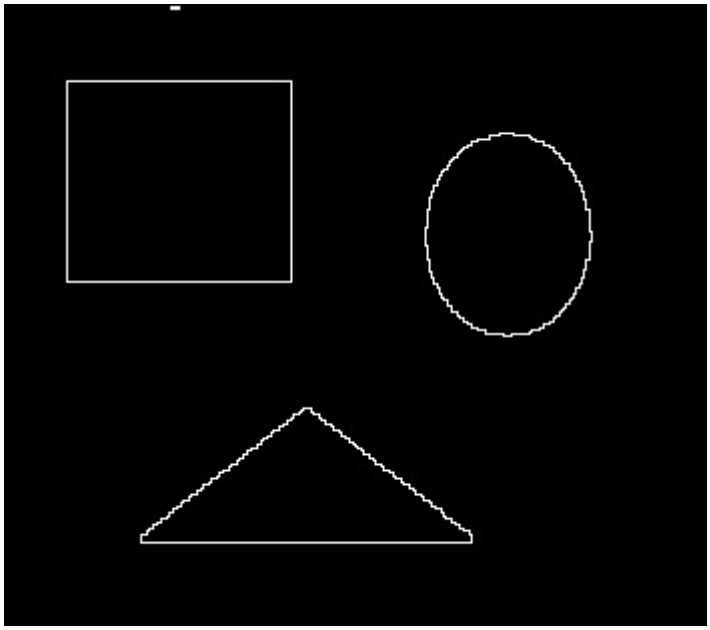

**Figure 5: Boundary extracted from res_noise2.jpg image (res_bound2.jpg)**

**References for task1:**


1. Lecture slide 10Morphological_Image_Processing.pdf (classroom notes)
2. http://www.aishack.in/tutorials/mathematical-morphology/
3. https://en.wikipedia.org/wiki/Opening_(morphology)
4. https://en.wikipedia.org/wiki/Closing_(morphology)
5. https://www.youtube.com/watch?v=P8JRZ4t7EOI&t=2s

## Task2: Image segmentation and point detection:

This image is used for point detection and histogram is also plotted for this image
http://sse.tongji.edu.cn/linzhang/DIP/demoCodes/ch8/pointDetec/turbine-blade.jpg

**Steps performed to obtain the point detected image are as follows:**

1. Read the image which is given in for point detection and then plot the histogram
2. For plotting histogram implemented the function for it:
3. The code snippet is as follows:
4. for i in range (0,256):
5. count=0
6. for p in range (0,img.shape[0]):
7. for q in range (0,img.shape[1]):
8. if img[p][q]==i:
9. count=count+1
10. arr.append(count)
11. x = np.array([i for i in range(256)])
12. a = np.array(arr)
13. plt.bar(x, a, width=1, bottom=None, align='center', data=None)
14. plt.title("histogram")
15. plt.show()
16. From the histogram determine the threshold value and applied approximate threshold to the given image
17. Used mask to find the R values i.e where R is summation of $w_i * z_i$
18. Move this mask on whole image and obtain value of R for each pixel point in image.
19. The formulation measures the weighted difference between the center point and its neighbors. • A point has been detected at the location on which the mask is centered if R >= T

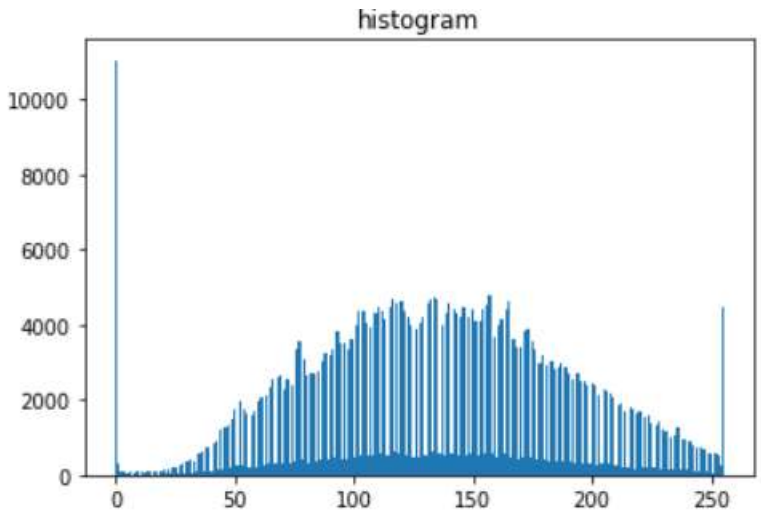


**Figure6: Histogram for turbine-blade.jpg(lossless image used)**

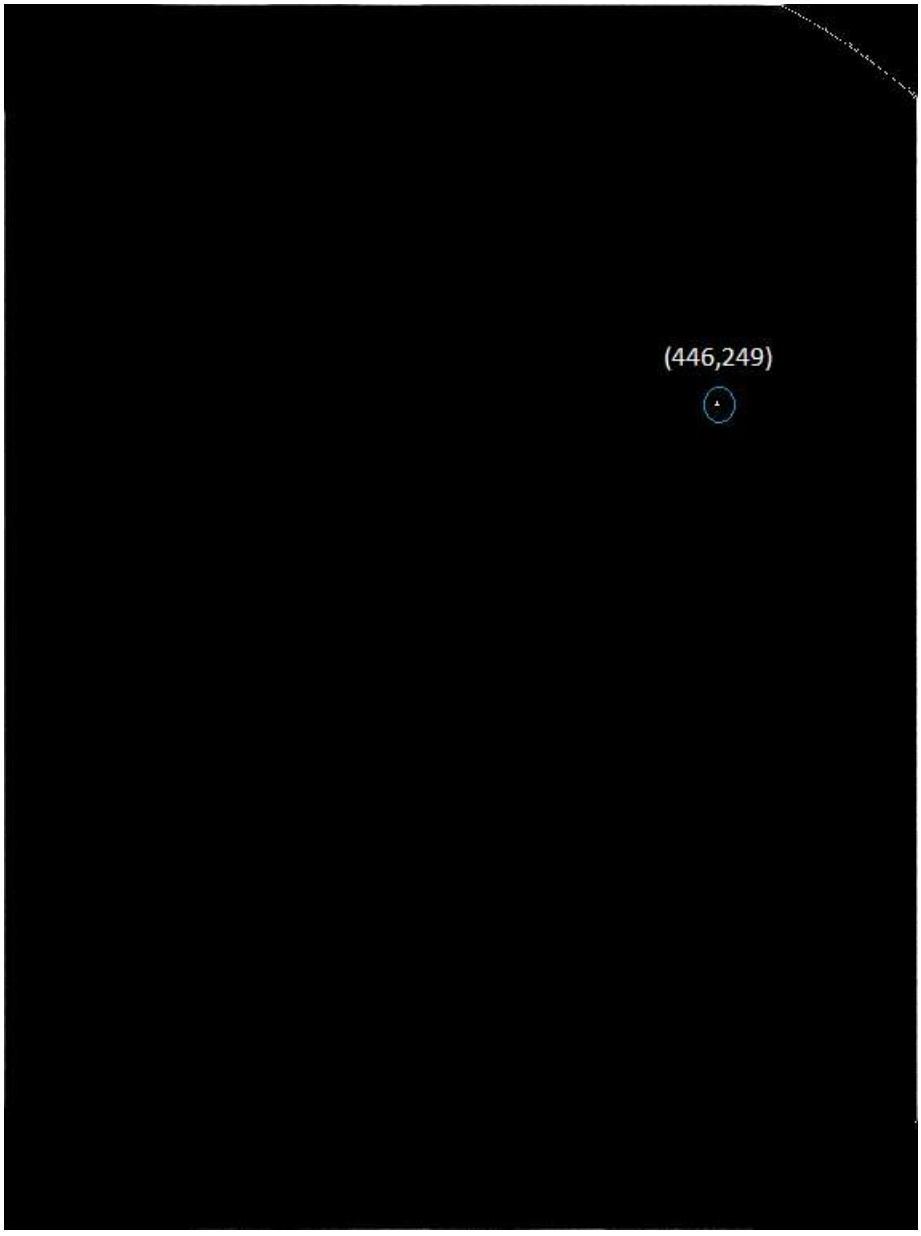


**Figure7 :Detected  point in given image (labeled with coordinate and circled with blue circle)**

**Coordinate of detected point are: (446,249)**

**Task2 Part2 Segmentation**

**Steps performed to get segmented image(bone pieces from original image ) are as follows:**

1. Read the imput segment.jpg image in gray scale
2. Segment the image on the basic of optimal threshold
3. Threshold is determined by plotting histogram and using global thresholding method
4. The algorithm for global threshold is implemented. Its code snippet is as follows:

```
5. def threshold_calculate(img,threshold,T0):
6.     G1=[]
7.     G2=[]
8.     pixel_sum1=0
9.     pixel_sum2=0
10.    for i in range(img.shape[0]):
11.        for j in range(img.shape[1]):
12.            if img[i][j]>threshold:
13.                G1.append(img[i][j])
14.            else:
15.                G2.append(img[i][j])
16.
17.
18.    for i in range (len(G1)):
19.        pixel_sum1=pixel_sum1+G1[i]
20.
21.    if(len(G1) != 0):
22.        avg1=(pixel_sum1)/(len(G1))
23.    else:
24.        avg1 = 0
25.    for j in range (len(G2)):
26.        pixel_sum2=pixel_sum2+G2[j]
27.
28.    avg2=(pixel_sum2)/(len(G2))
29.
30.    new_threshold=(0.5)*(avg1+avg2)
31.
32.
33.    if abs(new_threshold-threshold) >T0:
34.        threshold_calculate(img,new_threshold,T0)
35.
36.    return new_threshold
```

37. After getting segmented image bounding box along with cpoordinate of bounding box are represented in the final image.

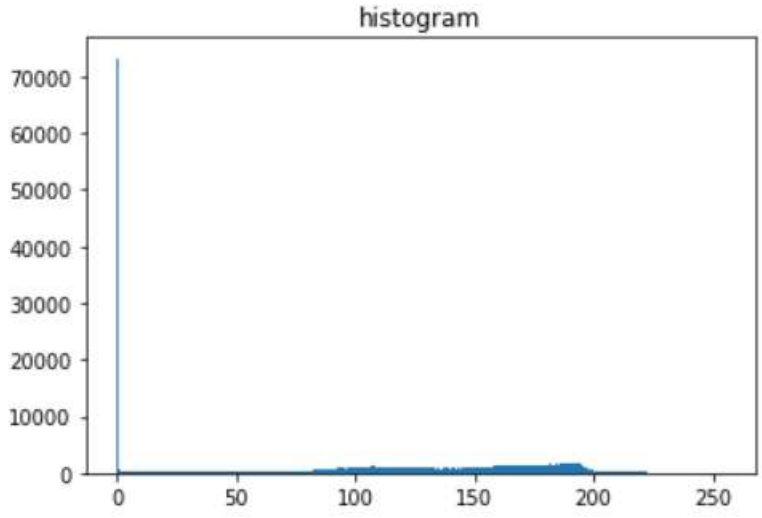


**Figure8: Histogram for segment.jpg**

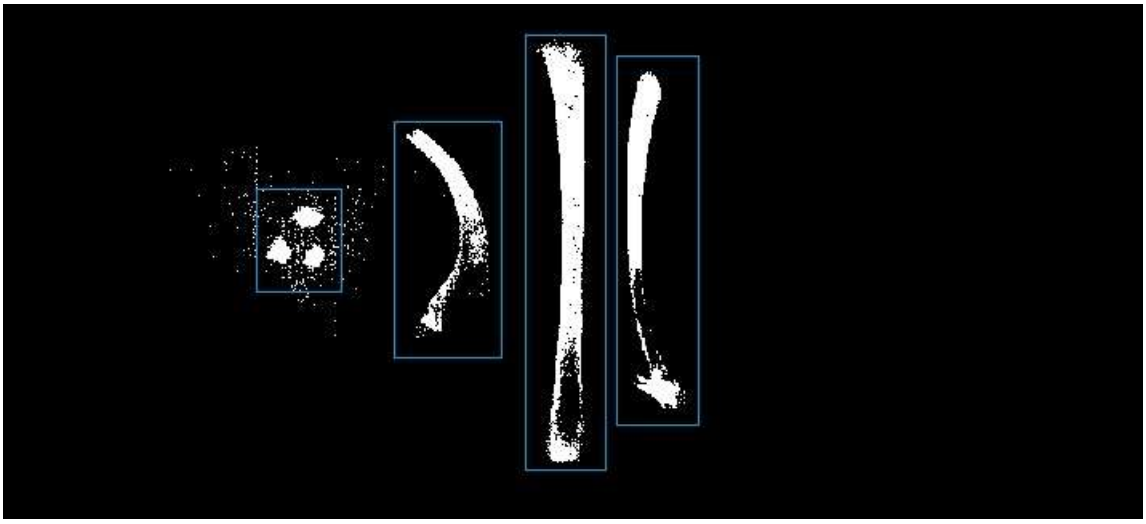

**Figure9: Segmented image with bounding box**

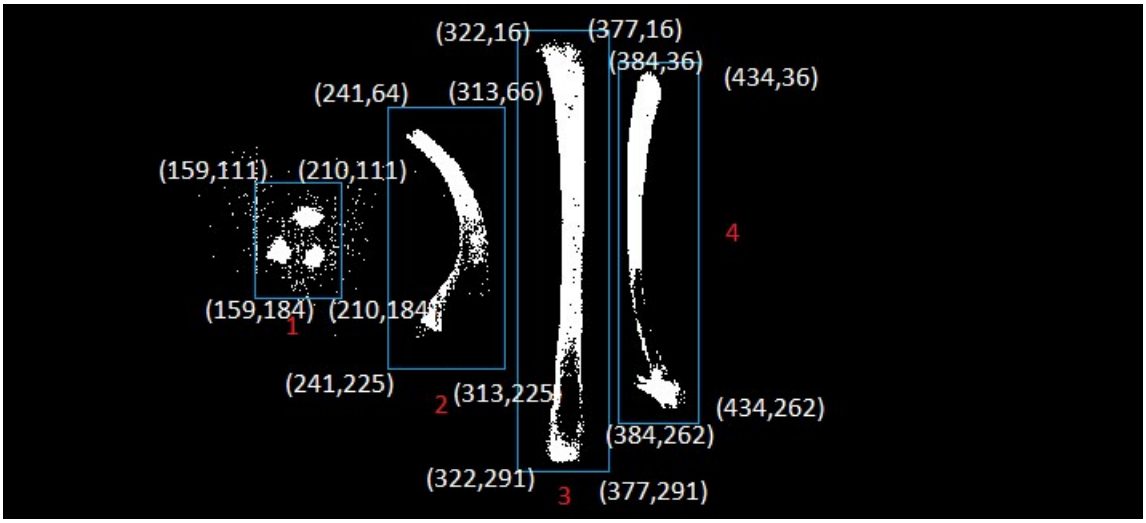


**Figure10: Segmented image with bounding box and coordinates labelled on it**

**Coordinates of bounding box1 (containing 3 small pieces of bones) :**

Upper left corner coordinate is : (159,111)
Upper right corner coordinates is: (210,111)
Lower left corner coordinate is: (159,184)
Lower right corner coordinate is: (210,184)

**Coordinates of bounding box2 (left big bone) :**

Upper left corner coordinate is : (241,64)
Upper right corner coordinates is: (313,64)
Lower left corner coordinate is: (241,225)
Lower right corner coordinate is: (313,225)

**Coordinates of bounding box3 (containing big middle bone) :**

Upper left corner coordinate is : (322,16)
Upper right corner coordinates is: (377,16)
Lower left corner coordinate is: (322,291)
Lower right corner coordinate is: (377,291)

**Coordinates of bounding box4(containing right big bone) :**

Upper left corner coordinate is : (384,36)
Upper right corner coordinates is: (434,36)

Lower left corner coordinate is: (384,262)
Lower right corner coordinate is: (434,262)

**References:**


1. Lecture slide 12_Image_Analysis (classroom notes)
2. https://matplotlib.org/users/pyplot_tutorial.html


## Task3 : Hough Transform

### Task1 part1 : Hough Transform for detecting lines:

No. of diagonal lines detected are: 7 (diagonal lines which are detected are represented by white color).

No. of vertical lines detected are: 6 (vertical line which are detected are represented by green color).

### Algorithm used to implement hough transform

Using Polar parameters:

xcos(theta)- ysin(theta ) =d

**Basic Hough Transform algorithm used for implementation is as follows:**

1. Initialize H[d,theta]=0
2. for each edge point in image I(x,y)
   for theta=[theta min to theta max]
   d= xcos(theta)- ysin(theta )
   H[d,theta]+=1
3. Find the values of (d,theta) where H[d,theta] is maximum
4. The detected lines in the image is given by
   d= xcos(theta)- ysin(theta )

**Steps performed in hough transformation to detect lines:**

1. Read image as input hough.jpg
2. Perform edge detection using sobel operator on the original image.
3. Sobel edge detection is implemented from scratch using using edge operator.
4. Pass this edge detected image to hough transform function so that accumulator values can be returned.
5. Then loop over all accumulator and theta values to get the equation of lines and used function o separate vertical lines from diagonal lines:
6. if(angle1<=-90.00 and angle1>-93):
7. cv2.line(original_img_1,(x1,y1),(x2,y2),(0,255,0),1)
8. #print("vertical")

```
9.
10.  else:
11.              cv2.line(original_img_1,(x1,y1),(x2,y2),(255,255,255),1)
```

**Figure11: This shows the result of hough transform on lines. It contains both diagonal and vertical lines detection. The diagonal lines are detected by white color and vertical lines by green color.**

**Code snippet for hough transform:**

```
def hough_trans(image):
    w = image.shape[0]
    h = image.shape[1]
    y_nzv, x_nzv = np.nonzero(image)
    angle = np.deg2rad(np.arange(-90.0, 90.0))
```

```
    cos_value = np.cos(angle)
    sin_value = np.sin(angle)
    total_angles= len(angle
    length_diagonal = int(np.ceil(np.sqrt(w * w + h * h))
    value_acc = np.zeros((2 * length_diagonal, total_angles),dtype=np.uint64)
    values = np.linspace(-length_diagonal, length_diagonal, length_diagonal * 2.0)
    for i in range(len(x_nzv)):
        x=x_nzv[i]
        y=y_nzv[i]
        for t in range(total_angles):
            value=int(round(x * cos_value[t] + y * sin_value[t]) + length_diagonal)
            value_acc[value,t]+=1
    return value_acc, angle, values
```

**References:**


1. http://me.umn.edu/courses/me5286/vision/Notes/2015/ME5286-Lecture9.pdf
2. https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html
3. https://alyssaq.github.io/2014/understanding-hough-transform/
4. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html


**Task3 Part3: Bonus Question:**

**Hough Transform to detect circles in the image:**

No. of circle detected are 7 using hough transform algorihm for circle detection:

Algorithm used to detect coins in the image is same as used to detect circle in the image using hough transform:

```
For every edge pixel(x,y):
    For each possible raduis value r:
        For each possible gradient direction theta:
            Use estimated gradient
            a=x-r(cos(theta))
            b=y+r(sin(theta))
            H[a,b,r]+=1
        End
    End
End
```

**Steps performed for coin detection:**

1. Read the imput image hough.jpg
2. Find the gaussian blur fo the image
3. Determine the edge detected using edge detection method
4. Choose radii as 100
5. Call hough circle detection function which takes input as image,radii and accumulator array)
6. Returned values will contain a,b,r and use them to represent circle(coins) on original image.

**Code snippet for coin detection is:**

```
def hough_trans_circle(x0,y0,r_d):
    cord_x = r_d
    cord_y=0
    cond_fact = 1-cord_x
    while(cord_y<cord_x ):
        if(cord_x + x0<h and cord_y + y0<w):
            container_st [ cord_x + x0,cord_y + y0,r_d]+=1;
        if(cord_y  + x0<h and cord_x + y0<w):
            container_st [ cord_y  + x0,cord_x + y0,r_d]+=1;
        if(-cord_x + x0<h and cord_y + y0<w):
            container_st [-cord_x + x0,cord_y  + y0,r_d]+=1;
        if(-cord_y  + x0<h and cord_x + y0<w):
            container_st [-cord_y  + x0,cord_x + y0,r_d]+=1;
        if(-cord_x + x0<h and -cord_y  + y0<w):
            container_st [-x + x0,-cord_y  + y0,r_d]+=1;
        if(-cord_y  + x0<h and -cord_x + y0<w):
            container_st [-cord_y  + x0,-x + y0,r_d]+=1;
        if(cord_x + x0<h and -cord_y  + y0<w)
            container_st [ cord_x + x0,-cord_y  + y0,r_d]+=1;
        if(cord_y  + x0<h and -cord_x + y0<w):
```

```
    container_st [ cord_y  + x0,-cord_x + y0,r_d]+=1;
cord_y+=1
if(cond_fact<=0):
    cond_fact += 2 * cord_y + 1
else:
    cord_x=cord_x-1;
    cond_fact+= 2 * (cord_y - cord_x) + 1
```

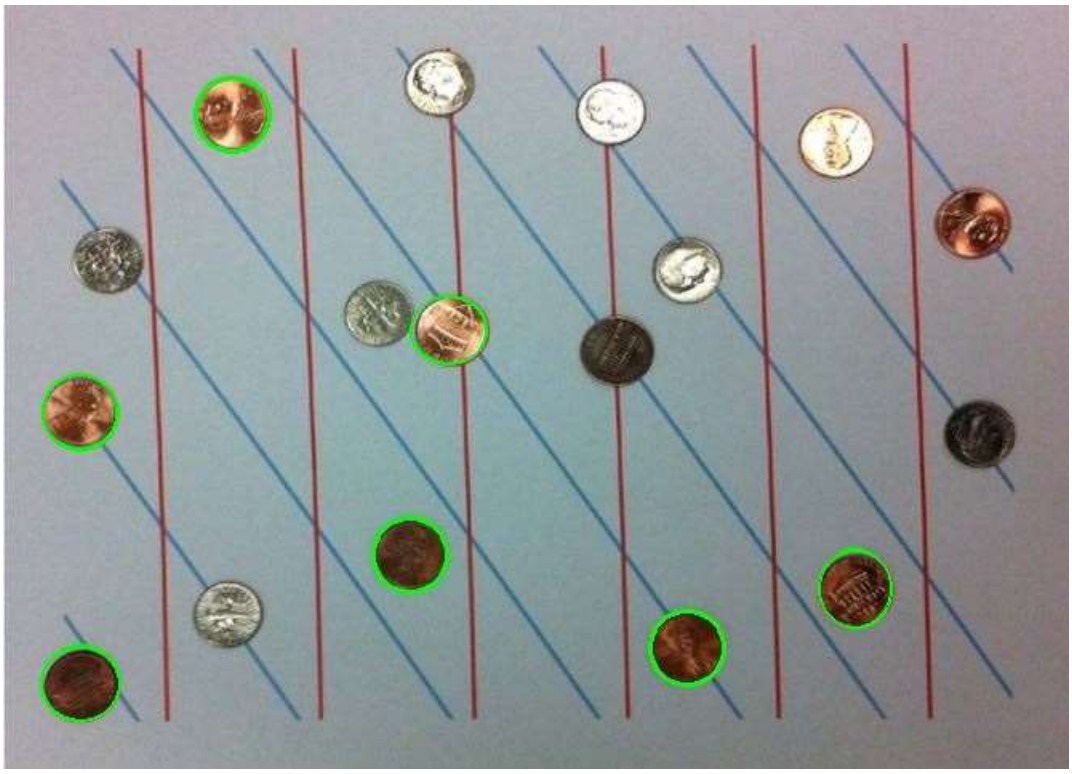

**Figure12: It shows the no. of coins detected when hough transformation is applied to it.**

**References:**


1. https://en.wikipedia.org/wiki/Circle_Hough_Transform
2. http://www.aishack.in/tutorials/circle-hough-transform/
3. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghcircles/py_houghcircles.html
4. https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/