Project 2

CSE573: Computer Vision

& Image Processing

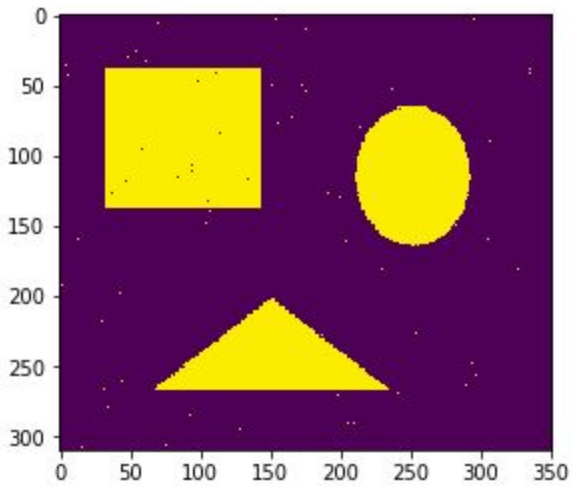
NAME :- PRANAV VIJ
UB NUMBER :- 50290588

Morphology Image Processing

Erosion and dilation are morphological image processing operations. Morphological image processing basically deals with modifying geometric structures in the image. These operations are primarily defined for binary images, but we can also use them on grayscale images. Erosion basically strips out the outermost layer of pixels in a structure, where as dilation adds an extra layer of pixels on a structure.

```
img = cv2.imread('original_imgs/noise.jpg', 0)
show('Original Image', img)
```

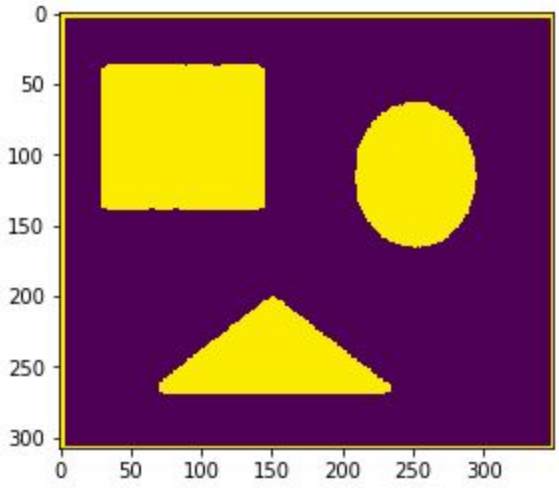
```
Original Image
```



Question 1a)

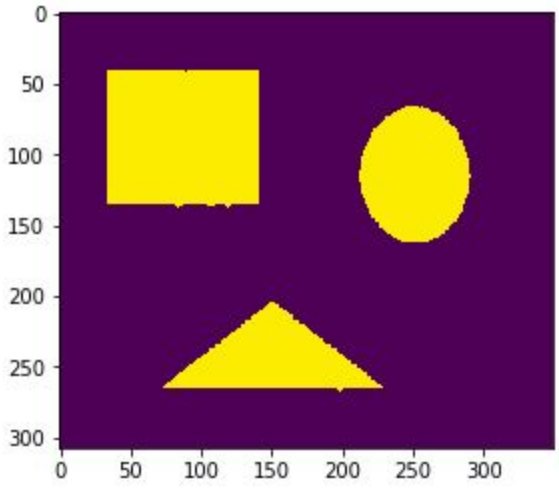
Method 1 (Erosion + Dilation)

Opening



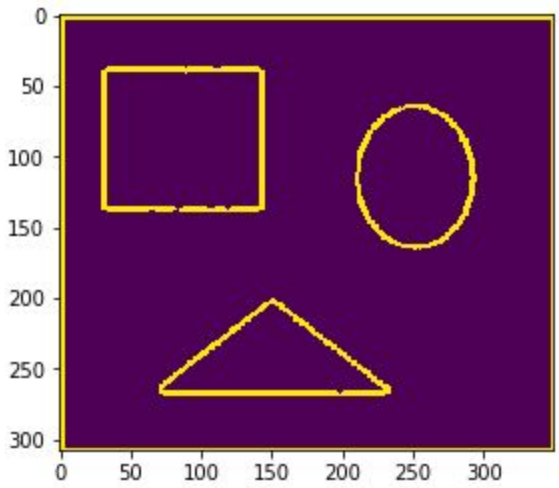
Method 2 (Dilation + Erosion)

Closing



Question 1b)

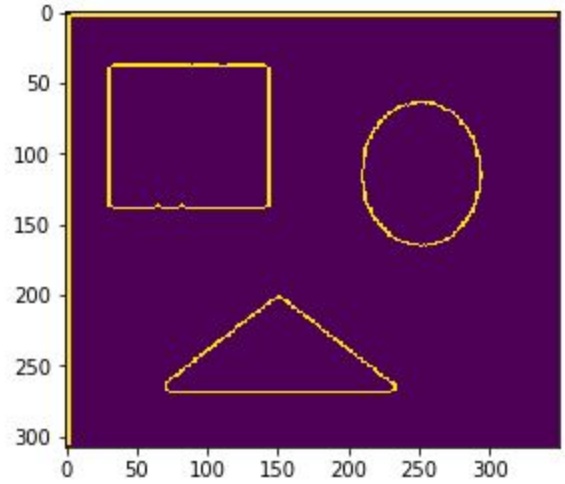
Comparison



Question 1c)

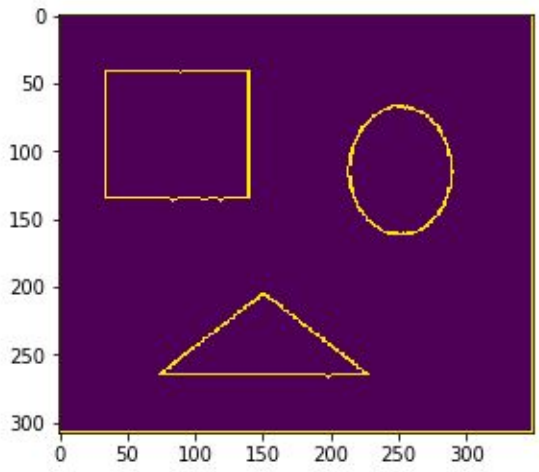
```
boundary_1 = result_1 - erode(result_1, kernel_method_1)
show('res_bound1',boundary_1)
```

res_bound1



```
boundary_2 =  erode(result_2, kernel_method_1) - result_2
boundary_2 = compliment_A(compliment_A(np.asarray(boundary_2).tolist()))
show('res_bound2', boundary_2)
```

```
res_bound2
```



CODE

```
def show(image_name, img):
    print(image_name)
    imgplot = plt.imshow(img)

def normalize_matrix(A):
    row_a_size = len(A)
    col_a_size = len(A[0])
    max_x = 0
    min_x = A[0][0]
    for i in range(0, row_a_size):
        max_x = max(max(A[i]), max_x)
        min_x = min(min(A[i]), min_x)
    for i in range(0, row_a_size):
        for j in range(0, col_a_size):
            A[i][j] = int(255 * ((max_x - A[i][j])/(max_x - min_x)))
    return A

def add_padding(a, n = 3):
    row_len = len(a)
    col_len = len(a[0])
    MATRIX_1 = []
```

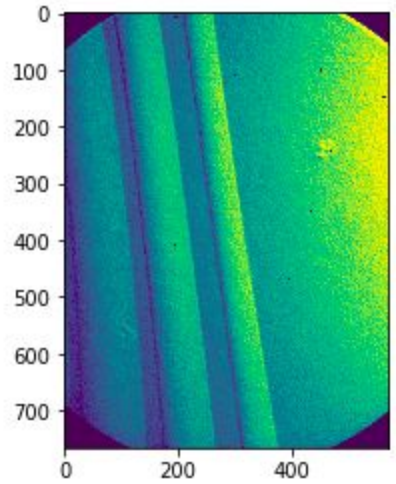
```
    padding_len = int((n - 1)/2)
    for i in range(0, padding_len):
        MATRIX_1.append([0]*(col_len + n - 1))
    for i in range(0, row_len):
        MATRIX_1.append([0]*padding_len + a[i] + [0]*padding_len)
    for i in range(0, padding_len):
        MATRIX_1.append([0]*(col_len + + n - 1))
    return MATRIX_1

def dilate(img_padded, B):# WHERE A IS BIGGER THAN B
    A = np.asarray(img_padded).tolist()
    B = np.asarray(B).tolist()
    n = len(B)
    padding = int((n - 1)/2)
    row_a_size = len(A)
    col_a_size = len(A[0])
    MATRIX_SCANNED = []

    for i in range(0, row_a_size):
        MATRIX_SCANNED.append([0]*col_a_size)
    for i in range(padding, row_a_size - padding):
        for j in range(padding, col_a_size - padding):
            for k in range(0, n):
                for l in range(0, n):
                    if A[i - padding + k][j - padding + l] != 0 and B[k][l] != 0:
                        MATRIX_SCANNED[i][j] = 1
                        break
    return MATRIX_SCANNED

def compliment_A(A):
    row_a_size = len(A)
    col_a_size = len(A[0])

    MATRIX_SCANNED = []
    for i in range(0, row_a_size):
        MATRIX_SCANNED.append([0]*col_a_size)

    for i in range(0, row_a_size):
        for j in range(0, col_a_size):
            if A[i][j] == 0:
                MATRIX_SCANNED[i][j] = 1
    return MATRIX_SCANNED

def erode(img_padded, B):
    A = np.asarray(img_padded).tolist()
    B = np.asarray(B).tolist()
    n = len(B)
```

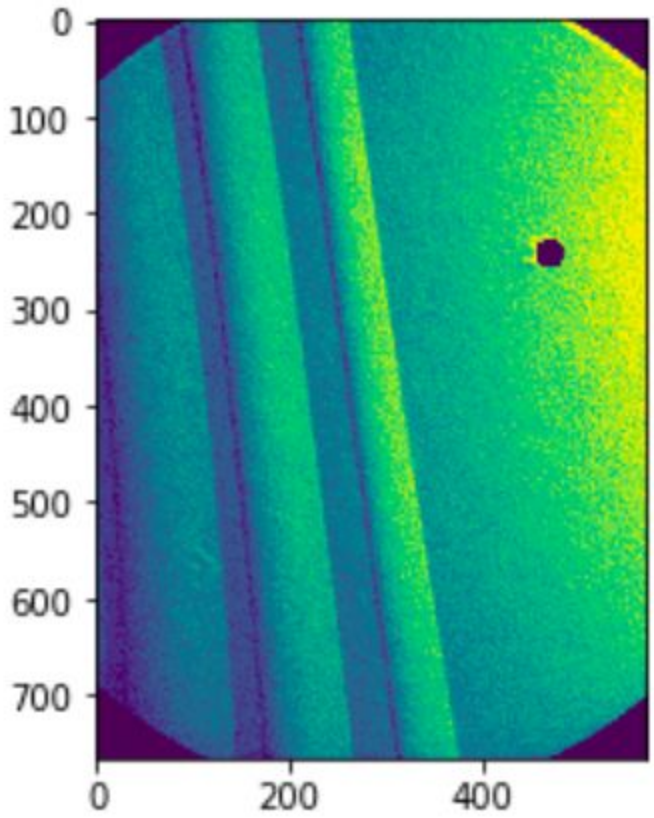
```
    complement_Ac = compliment_A(A)
    erode = dilate(np.asarray(complement_Ac), B)
    erode_c = compliment_A(erode)
    return erode_c

def remove_padding(img, pad):
    n = int((pad - 1)/2)
    matrix = []
    col = len(img[0])
    row = len(img)
    c,r = col - 2*n, row - 2*n
    for i in range(n, r):
        matrix.append(img[i][n: c])
    return matrix

img = cv2.imread('original_imgs/noise.jpg', 0)
show('Original Image', img)

#### Method 1 Erosion + Dilation  ####
kernel_method_1 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))

img_padding = add_padding(np.asarray(img).tolist(), kernel_method_1.shape[0])

erosion = erode(img_padding, kernel_method_1)
dilation = dilate(erosion, kernel_method_1)
dilation = dilate(dilation, kernel_method_1)

result_1 = remove_padding(dilation, kernel_method_1.shape[0])
show('Opening', result_1)

#### Method 2 Dilation + Erosion ####
dilation = dilate(img_padding, kernel_method_1)
erosion = erode(dilation, kernel_method_1)
erosion = erode(erosion, kernel_method_1)

result_2 = remove_padding(erosion, kernel_method_1.shape[0])
show('Closing', result_2)




result_1 = np.asarray(result_1)
result_2 = np.asarray(result_2)

comparison = result_1 - result_2
show('Comparison', comparison)
```

```
boundary_1 = result_1 - erode(result_1, kernel_method_1)
show('res_bound1',boundary_1)

boundary_2 =  erode(result_2, kernel_method_1) - result_2
boundary_2 = compliment_A(compliment_A(np.asarray(boundary_2).tolist()))
show('res_bound2', boundary_2)
```

Image segmentation and point detection

```
Multiplication
Highlighted Image
```



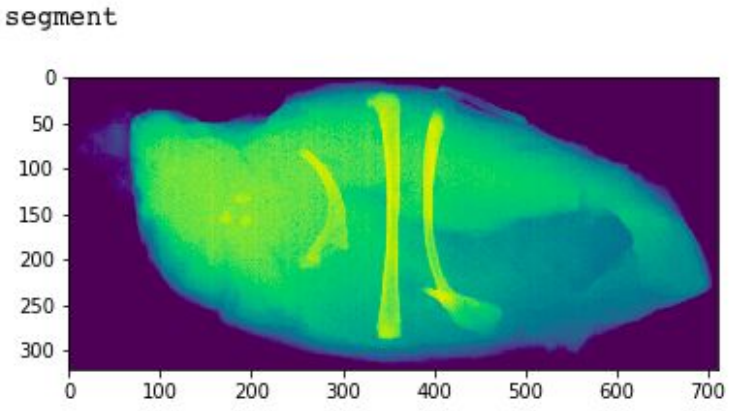


CODE

```
def point_detection(A, B, n, threshold, upper_threshold):# WHERE A IS BIGGER THAN B
    padding = int((n - 1)/2)
    row_a_size = len(A)
    col_a_size = len(A[0])
    MATRIX_SCANNED = []
    points = []
    for i in range(0, row_a_size):
        MATRIX_SCANNED.append([0]*col_a_size)
    for i in range(padding, row_a_size - padding):
        for j in range(padding, col_a_size - padding):
            for k in range(0, n):
                for l in range(0, n):
                    MATRIX_SCANNED[i][j] += B[k][l]*A[i - padding + k][j - padding
+ l]
            if threshold > MATRIX_SCANNED[i][j] or MATRIX_SCANNED[i][j] >
upper_threshold:
                MATRIX_SCANNED[i][j] = 0
            else:
```

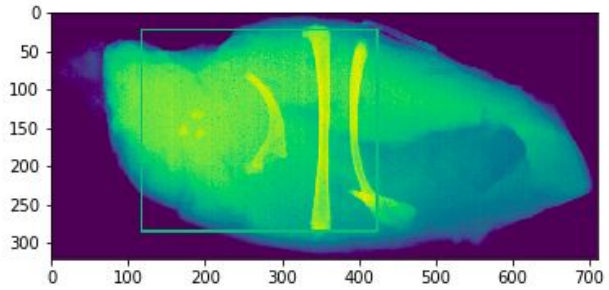
```
#                 print(MATRIX_SCANNED[i][j], i, j)
                MATRIX_SCANNED[i][j] = 255
                points.append((i,j))
    return np.asarray(MATRIX_SCANNED), points
img_point = cv2.imread('original_imgs/turbine-blade.jpg', 0)
B = [[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]]
point_padding = add_padding(img_point.tolist())
mult, points = point_detection(point_padding, B, 3, 248, 249)
show('Multiplication', mult)
for t in points:
    cv2.circle(img_point, t, 2, (0,0,255), -1)
show('Highlighted Image', img_point)
```

Segmentation by Thresholding



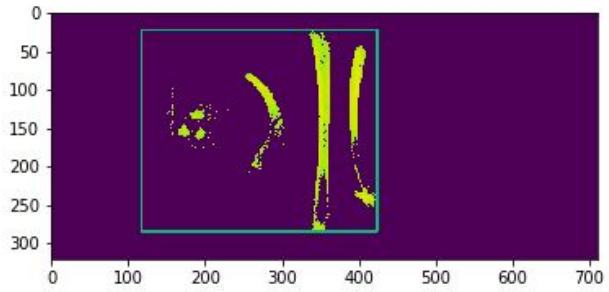
```
show('Drawing Rectangle on Original', cv2.rectangle(display_segm
```

```
Drawing Rectangle on Original
```




```
show('Drawing Rectangle', cv2.rectangle(img_segment, (start_r, s
```

```
Drawing Rectangle
```



CODE

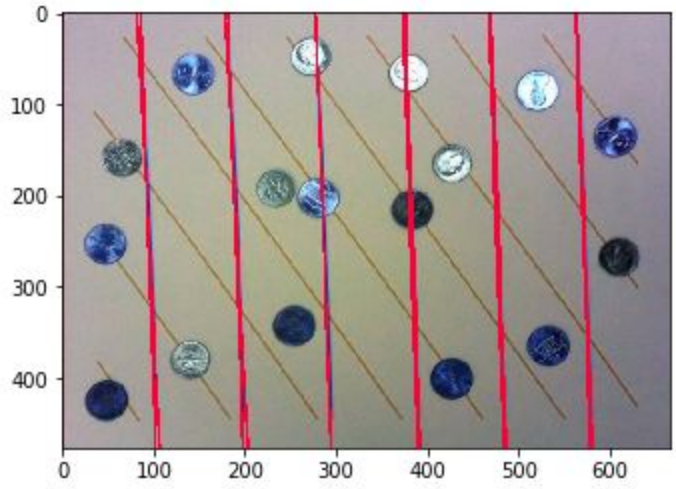
```
def thresholding(img, threshold):
    r,c = len(img), len(img[0])
    for i in range(0, r):
        for j in range(0, c):
            if threshold > img[i][j]:
                img[i][j] = 0
    return img

display_segment = cv2.imread('original_imgs/segment.jpg', 0)
img_segment = cv2.imread('original_imgs/segment.jpg', 0)
show('segment', img_segment)
r, c = img_segment.shape
```

```
thresholding_img = thresholding(img_segment, 200)
show('thresholding', thresholding_img)
start_r, end_r = 0,0
start_t, end_t = 0,0

for j in range(0, c):
    for i in range(0, r):
        start_r = max(start_r, thresholding_img[i][j])
    if start_r > 0:
        start_r = j
        break
for j in range(c - 1, -1, -1):
    for i in range(0, r):
        end_r = max(end_r, thresholding_img[i][j])
    if end_r > 0:
        end_r = j
        break
for i in range(0, r):
    for j in range(0, c):
        start_t = max(start_t, thresholding_img[i][j])
    if start_t > 0:
        start_t = i
        break
for i in range(r - 1, -1, -1):
    for j in range(0, c):
        end_t = max(end_t, thresholding_img[i][j])
    if end_t > 0:
        end_t = i
        break
show('Drawing Rectangle on Original', cv2.rectangle(display_segment, (start_r,
start_t),(end_r, end_t),(155,100,0),2))
show('Drawing Rectangle', cv2.rectangle(img_segment, (start_r, start_t),(end_r,
end_t),(155,100,0),2))
```

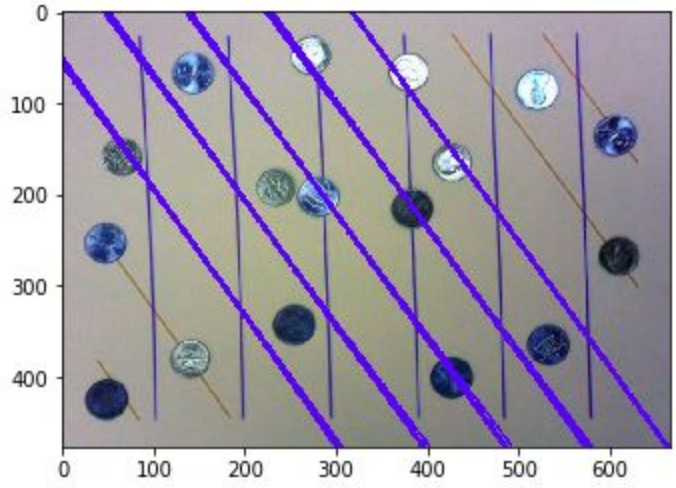
Hough transform

Result

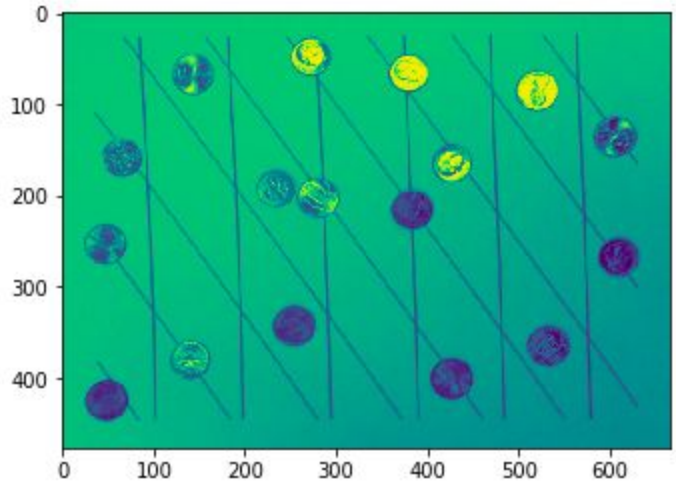
red line



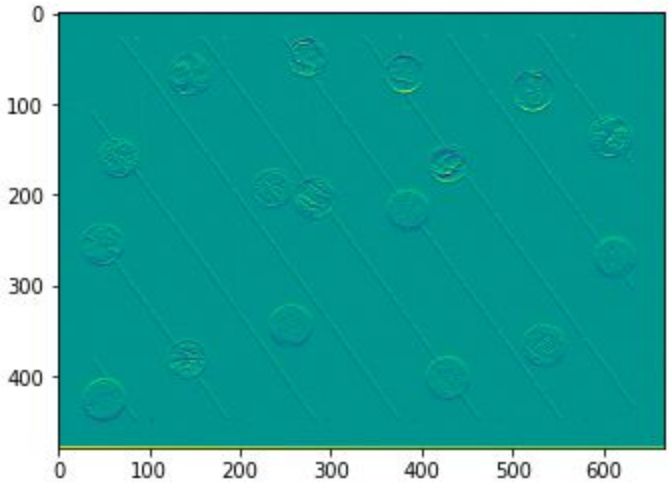
Blue line



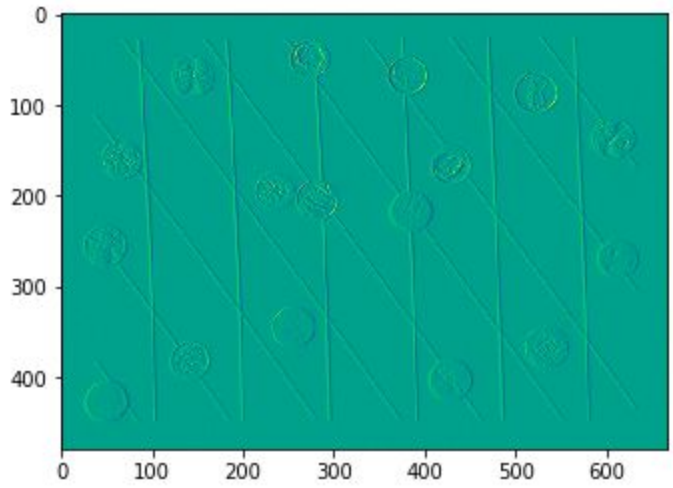
Original Gray Scale Image
(477, 666)



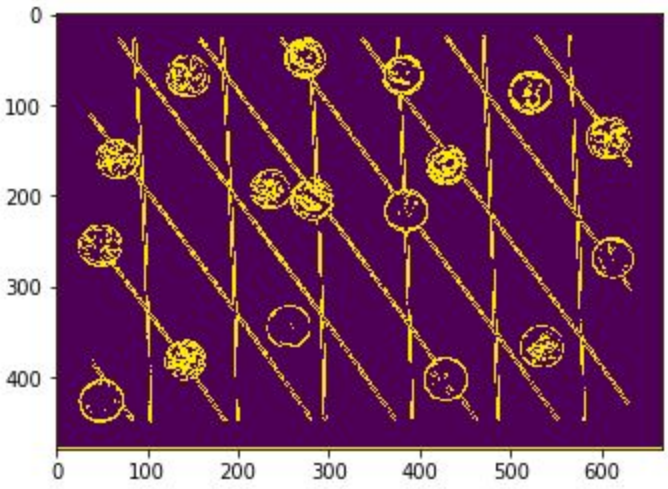
sobel_x



sobel_y



img_edge



RHO & THETA

```
point_red
```

```
[[0.0, 0.0],
 [-562.0, 3.1066861152648926],
 [-182.0, 3.1066861152648926],
 [-82.0, 3.0892326831817627],
 [-470.0, 3.1066861152648926],
 [-467.0, 3.1066861152648926],
 [-376.0, 3.1066861152648926],
 [-86.0, 3.1066861152648926],
 [-278.0, 3.1066861152648926],
 [-564.0, 3.1066861152648926],
 [-373.0, 3.1066861152648926],
 [-178.0, 3.0892326831817627]]
```

```
point_blue
```

```
[[-183.0, 2.5132741928100586],
 [-42.0, 2.5132741928100586],
 [-186.0, 2.5132741928100586],
 [-39.0, 2.5132741928100586],
 [-258.0, 2.5132741928100586],
 [-111.0, 2.5132741928100586],
 [-114.0, 2.5132741928100586],
 [-255.0, 2.5132741928100586],
 [35.0, 2.5132741928100586],
 [28.0, 2.5307273864746094],
 [32.0, 2.5132741928100586],
 [-35.0, 2.4958207607269287],
 [-177.0, 2.4958207607269287],
 [-107.0, 2.4958207607269287]]
```

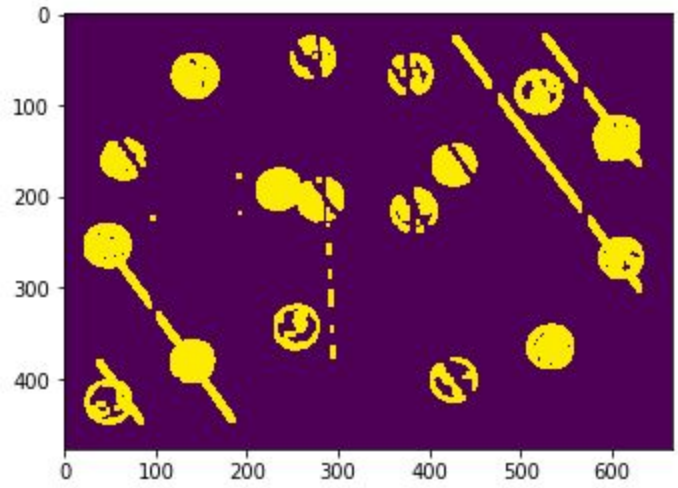
CODE

```
img_hough = cv2.imread('original_imgs/hough.jpg', 0)
show('Original Gray Scale Image', img_hough)
print(img_hough.shape)

sobel_x = np.asarray(matrix_mult_rect_scan(add_padding(img_hough.tolist()) ,
SOBEL_X, 3))
show('sobel_x',sobel_x)

sobel_y = np.asarray(matrix_mult_rect_scan(add_padding(img_hough.tolist()) ,
SOBEL_Y, 3))
show('sobel_y',sobel_y)
img_hough = np.asarray(add_padding(img_hough.tolist()))

img_edge = np.sqrt(np.square(sobel_x) + np.square(sobel_y))
t = 100
img_edge[img_edge >= t] = 255
img_edge[img_edge < t] = 0
show('img_edge', img_edge)

thetas = None
rhos = None

def hough_line(img):
    thetas = np.deg2rad(np.arange(-90.0, 90.0))
    width, height = img.shape
    diag_len = np.ceil(np.sqrt(width * width + height * height))   # max_dist
    rhos = np.linspace(-diag_len, diag_len, diag_len * 2.0)

    # Cache some resuable values
    cos_t = np.cos(thetas)
    sin_t = np.sin(thetas)
    num_thetas = len(thetas)

    # Hough accumulator array of theta vs rho
    accumulator = np.zeros((int(2 * diag_len), num_thetas), dtype=np.uint64)
    y_idxs, x_idxs = np.nonzero(img)  # (row, col) indexes to edges

    # Vote in the hough accumulator
    for i in range(len(x_idxs)):
        x = x_idxs[i]
        y = y_idxs[i]
        for t_idx in range(num_thetas):
        # Calculate rho. diag_len is added for a positive index
```

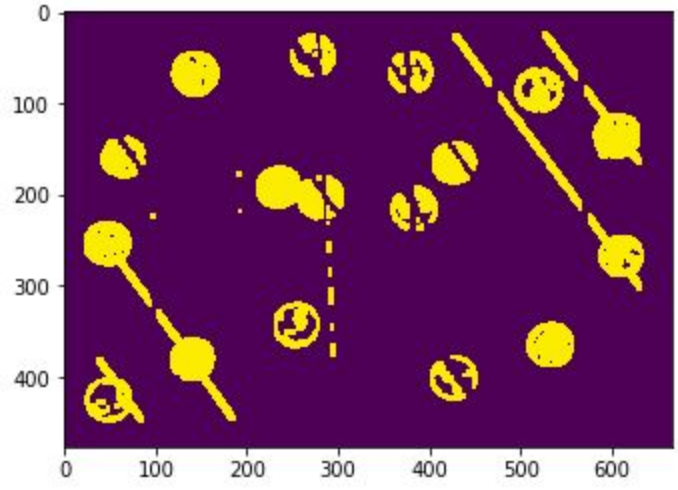
```
            rho = int(round(x * cos_t[t_idx] + y * sin_t[t_idx]) + diag_len)
            accumulator[rho, t_idx] += 1
    return accumulator

accumulator = hough_line(img_edge)
acc = accumulator.tolist()
print('Point of Maximum value')
max_value = np.amax(accumulator)
print(max_value)

points_of_maxima = []
start = int(max_value)
end = start - 400

for val in range(start, end, -1):
    for i in range(0, accumulator.shape[0]):
        for j in range(0, accumulator.shape[1]):
            if acc[i][j] == val:
                points_of_maxima.append((val, i, j))

print(len(points_of_maxima))

print(img_hough.shape)
for val, rhos, thetas in points_of_maxima:
    a = np.cos(thetas)
    b = np.sin(thetas)
    x0 = a*rhos
    y0 = b*rhos
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(img_hough,(x1,y1),(x2,y2),(0,0,255),2)

show('Hough Transformation', img_hough)
```

BONUS

red line



red line



CODE

```
for line in lines:
    rho,theta = line[0][0], line[0][1]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    mid_x, mid_y = int((x1 + x2)/2),int((y1+y2)/2)
    try:
        cv2.line(edges,(x1,y1),(x2,y2),(0, 0, 0),5)
    except Exception as e:
        print(e)
        print(mid_x, mid_y)

kernel = np.ones((5,5),np.uint8)
dilation = cv2.dilate(edges, kernel,iterations = 4)
erosion = cv2.erode(dilation, kernel,iterations = 5)
show('red line', erosion)
circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, 1.2, 150)
output = img.copy()
print(circles)
if circles is not None:
    circles = np.round(circles[0, :]).astype("int")

    for (x, y, r) in circles:
        if r < 50:
            cv2.circle(output, (x, y), r, (0, 255, 0), 4)
    show('Circle Detection', np.hstack([img, output]))
```

REFERENCES

1. https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html
2. https://github.com