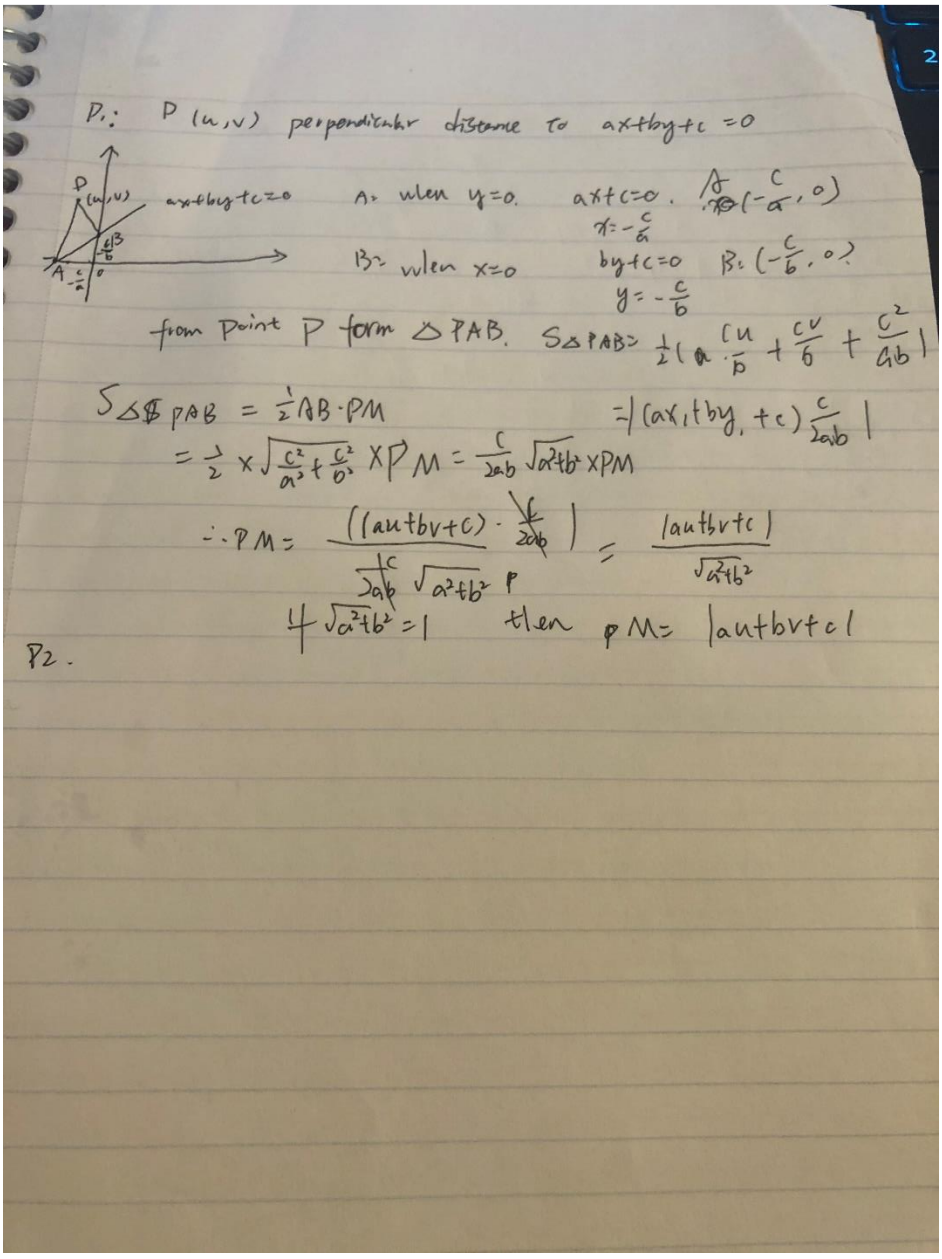


Q2

1)

For each A[a,b,r] = 0;

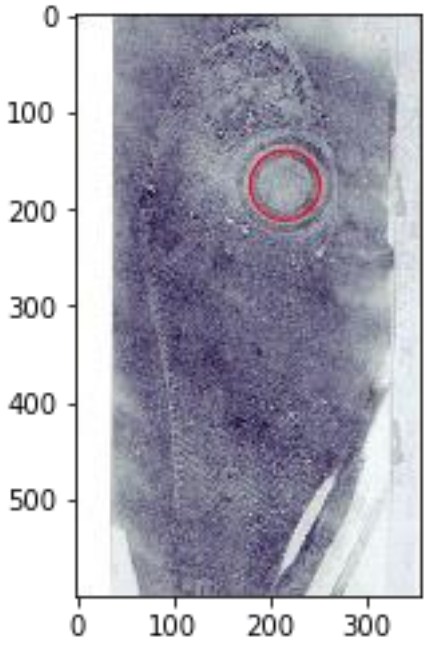
Process the filtering algorithm on image Gaussian Blurring, convert the image to grayscale (grayScaling), make Canny operator, The Canny operator gives the edges on image.

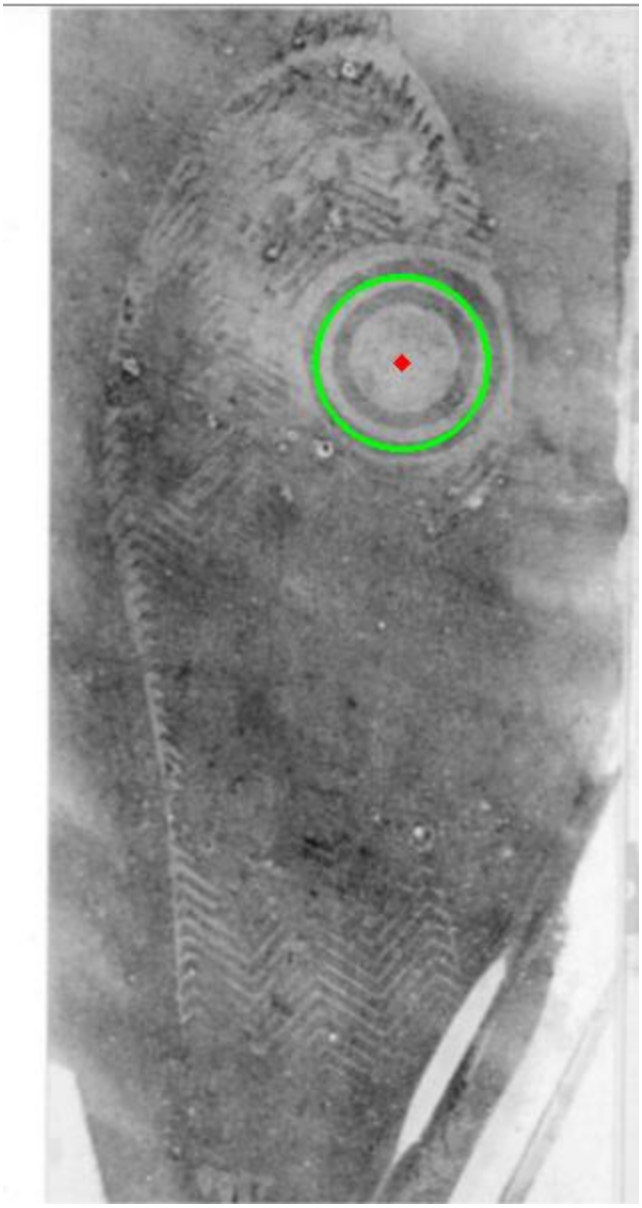
Vote the all possible circles in accumulator.

The local maximum voted circles of Accumulator A gives the circle Hough space.

The maximum voted circle of Accumulator gives the circle.

2)





```
im1=cv2.imread("circle.png")

im1.shape

def gauss(img,size):


    blur = cv2.GaussianBlur(img,(size,size),0)


    return blur

def to_gray(img):
```

```
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return gray

def canny(img,threshold1,threshold2):
    edges = cv2.Canny(img,threshold1,threshold2)
    return edges

def cedge(input):

    input = input.astype('uint8')

    # Using OTSU thresholding - bimodal image
    otsu_threshold_val, ret_matrix = cv2.threshold(input,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

    #lower_threshold = otsu_threshold_val * 0.8
    #upper_threshold = otsu_threshold_val * 1.7

    lower_threshold = otsu_threshold_val * 0.4
    upper_threshold = otsu_threshold_val * 1.3

    print(lower_threshold,upper_threshold)

    #print(lower_threshold,upper_threshold)
    edges = cv2.Canny(input, lower_threshold, upper_threshold)
    return edges
cv2.imshow("img",im1)
cv2.imshow("img_gray",to_gray(im1))
cv2.imshow("img_gauss",gauss(to_gray(im1),3))
cv2.imshow("img_canny",canny(gauss(to_gray(im1),3),75,150))
```

```
cv2.imshow("img_cedge",cedge(gauss(to_gray(im1),3)))
cv2.waitKey(0)
cv2.destroyAllWindows()

def detectCircles(img,threshold,region,radius = None):
    (M,N) = img.shape
    if radius == None:
        R_max = np.max((M,N))
        R_min = 3
    else:
        [R_max,R_min] = radius

    R = R_max - R_min
    #Initializing accumulator array.
    #Accumulator array is a 3 dimensional array with the dimensions representing
    #the radius, X coordinate and Y coordinate resectively.
    #Also appending a padding of 2 times R_max to overcome the problems of overflow
    A = np.zeros((R_max,M+2*R_max,N+2*R_max))
    B = np.zeros((R_max,M+2*R_max,N+2*R_max))

    #Precomputing all angles to increase the speed of the algorithm
    theta = np.arange(0,360)*np.pi/180
    edges = np.argwhere(img[:,:])                                #Extracting all edge coordinates
    for val in range(R):
        r = R_min+val
        #Creating a Circle Blueprint
        bprint = np.zeros((2*(r+1),2*(r+1)))
        (m,n) = (r+1,r+1)                                    #Finding out the center of the blueprint
        for angle in theta:
```

```
            x = int(np.round(r*np.cos(angle)))
            y = int(np.round(r*np.sin(angle)))
            bprint[m+x,n+y] = 1
        constant = np.argwhere(bprint).shape[0]
        for x,y in edges:                                   #For each edge coordinates
            #Centering the blueprint circle over the edges
            #and updating the accumulator array
            X = [x-m+R_max,x+m+R_max]                       #Computing the extreme X values
            Y= [y-n+R_max,y+n+R_max]                        #Computing the extreme Y values
            A[r,X[0]:X[1],Y[0]:Y[1]] += bprint
        A[r][A[r]<threshold*constant/r] = 0

    for r,x,y in np.argwhere(A):
        temp = A[r-region:r+region,x-region:x+region,y-region:y+region]
        try:
            p,a,b = np.unravel_index(np.argmax(temp),temp.shape)
        except:
            continue
        B[r+(p-region),x+(a-region),y+(b-region)] = 1

    return B[:,R_max:-R_max,R_max:-R_max]

def plotHough(A):
    img = cv2.imread("circle.png")
    fig = plt.figure()
    plt.imshow(img)
    circleCoordinates = np.argwhere(A)
    circle = []
    for r,x,y in circleCoordinates:
```

```
        circle.append(plt.Circle((y,x),r,color=(1,0,0),fill=False))
        fig.add_subplot(111).add_artist(circle[-1])
    plt.show()
edges=cedge(gauss(to_gray(im1),3))
cv2.imshow("a",edges)
cv2.waitKey(0)
cv2.destroyAllWindows()

res = detectCircles(edges,14,30,radius=[55,25])
plotHough(res)
img = cv2.imread('circle.png',0)
img = cv2.GaussianBlur(img,(3,3),0)
cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)

circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,20,
                    param1=180,param2=80,minRadius=0,maxRadius=0)

circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv2.circle(cimg,(i[0],i[1]),2,(0,0,255),3)

cv2.imshow('detected circles',cimg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3)

Pros • All points are processed independently, so can cope with occlusion, gaps

Cons • Complexity of search time increases exponentially with the number of model parameters • Quantization: can be tricky to pick a good grid size