Introduction:

I am a native of Hsinchu. Many people say that Hsinchu is a **desert of delicacy** and there are no tourist attractions in Hsinchu, but I strongly disagree, so I will use this assignment to share the food and beauty of Hsinchu that I know.

East door city(東門城)

Task3: Use Hough transform to detect straight lines and cycles.

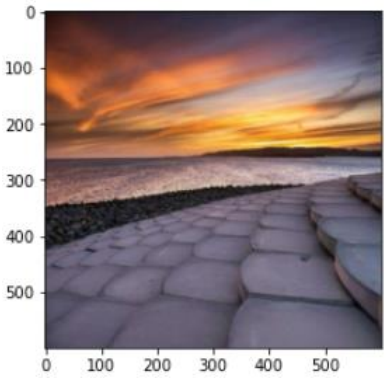
A total of three tourist attractions images and three food image were used, and four image processing methods were used, namely:

1. Smoothing Filters
    A. Gaussian filter
2. Edge Detection
    A. Sobel filter
    B. Threshold
3. Hough transform for lines
4. Hough transform for cycles

Preprocessing:

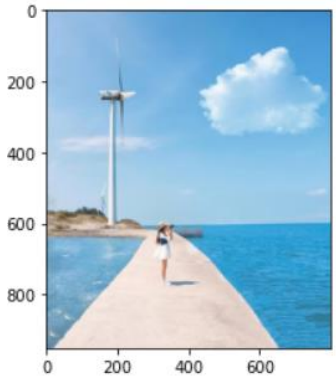
1. Load images

   Use python matplotlib api load images, and change images to array.



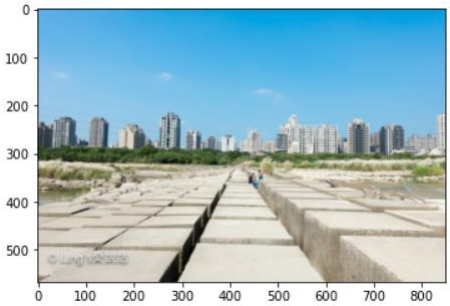
Nanliao Fishing Port

Fish scale ladder(魚鱗天梯)



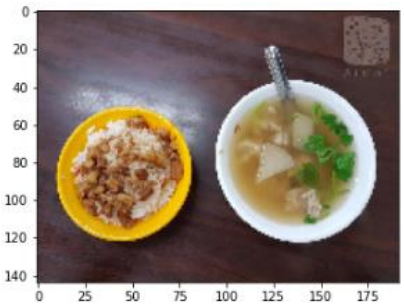
Hsinchu Xiangshan

Voice of the Sea(海之聲)



Hsinchu Zhubei

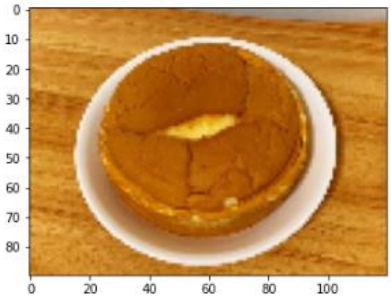
Tofu Rock(豆腐岩)



Afu

Braised Pork Rice

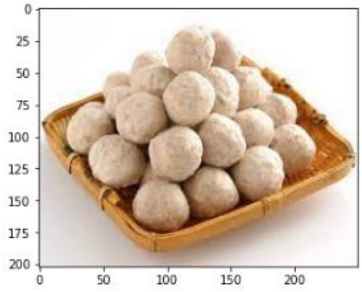
(阿富 魯肉飯)



Xin Meizhen

Pudding Cake

(新美珍 布丁蛋糕)



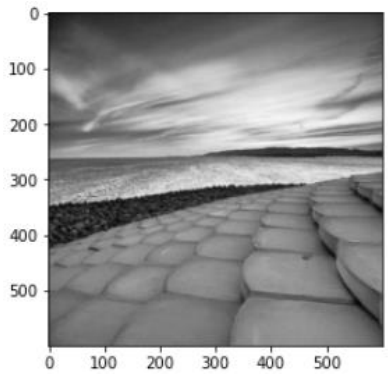
Shijia

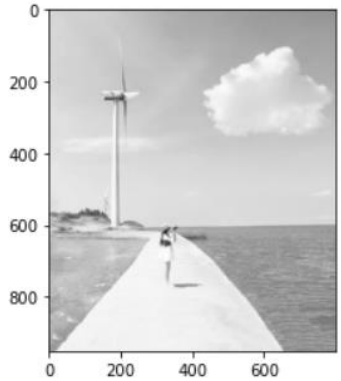
Fish Ball

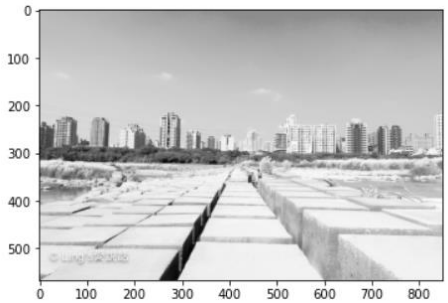
(石家 魚丸)

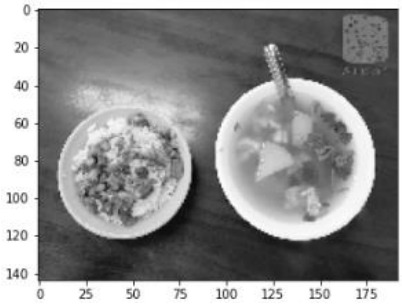
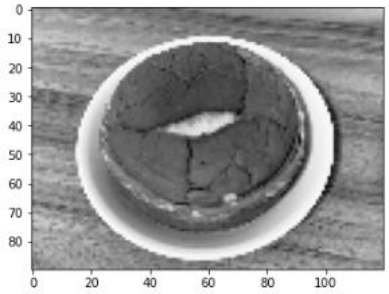
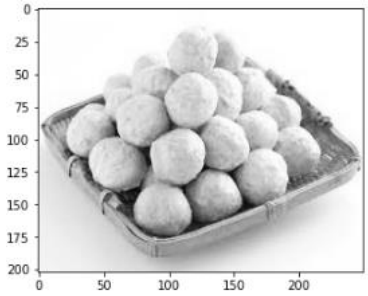
2. Image grayscale

   Use **Y' = 0.299 R + 0.587 G + 0.114 B**，change images from 3-dimension to 1-dimension。







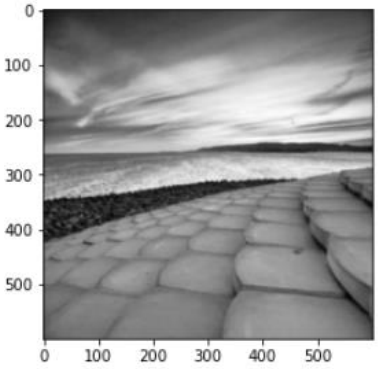




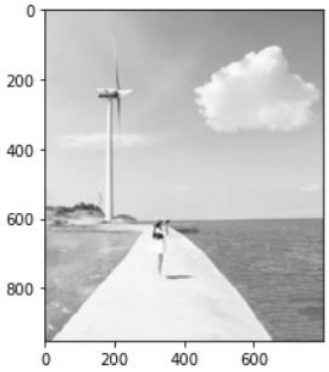
Method detail

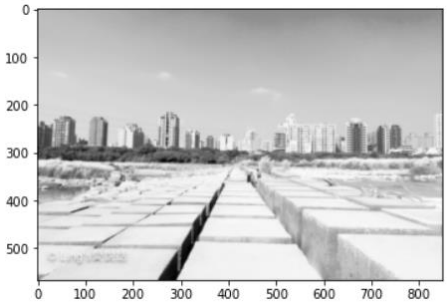
1. Smoothing Filters

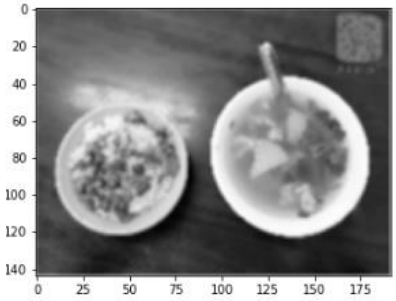
   A. Gaussian filter

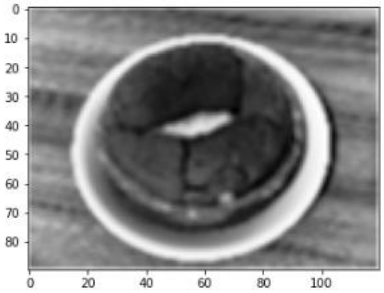
   I use Gaussian filtering to reduce noise and increase σ to make the edges more obvious.

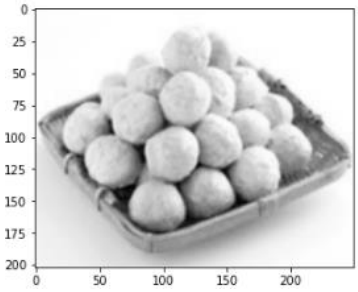

σ = 1


σ = 1


σ = 1


σ = 1.3


σ = 1.3


σ = 1.2

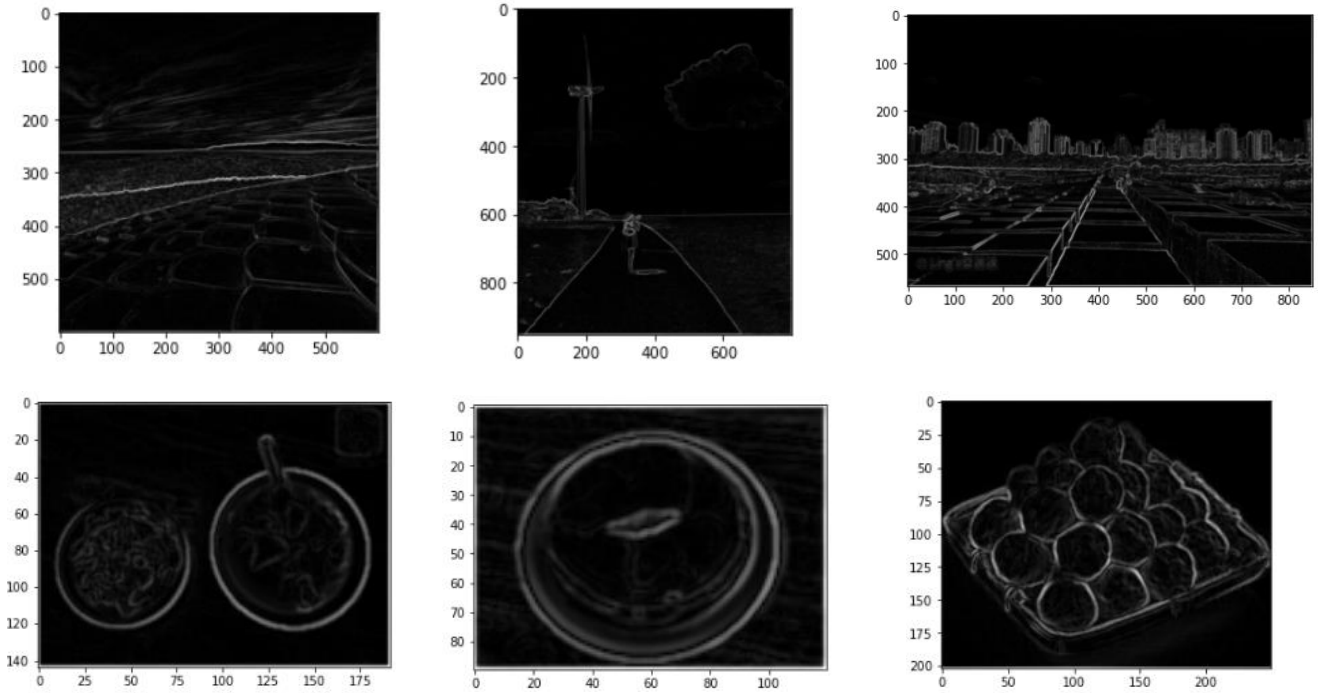
2. Edge Detection

   A. Sobel filter

   Use horizontal gradient、vertical gradient and $|g_x|+|g_y|$ to detect edge.

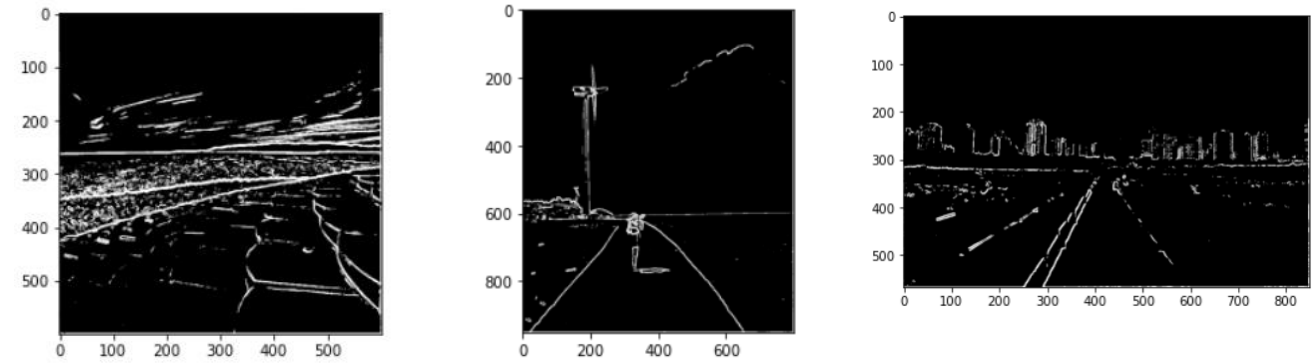
-1	0	1
-2	0	2
-1	0	1

-1	0	1
-2	0	2
-1	0	1

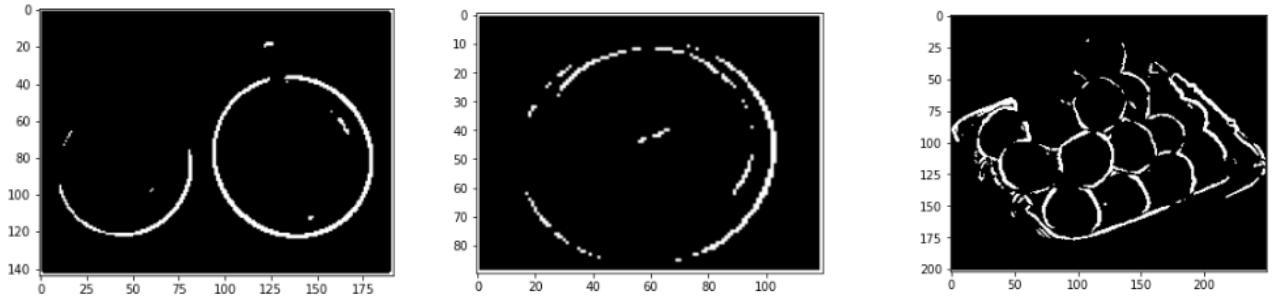


B. Threshold

Normalize each image after the sobel filter to the range [0, 255], and set the threshold according to each image to clearly show the contour.



Threshold=40 Threshold=40 Threshold=75

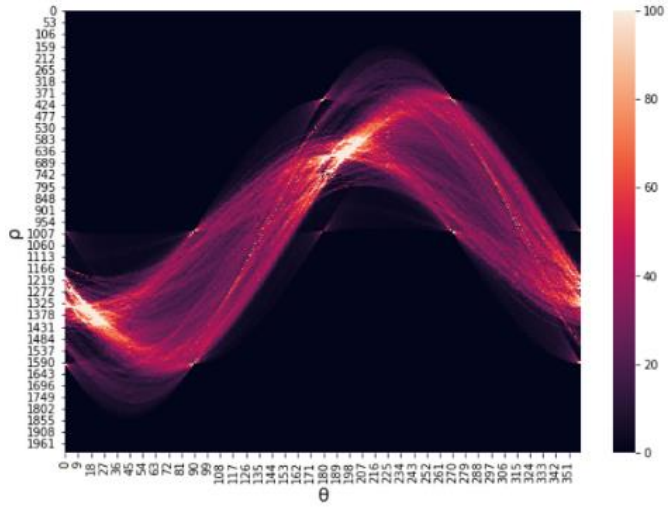


Threshold=50 Threshold=50 Threshold=70

3. Hough transform for lines

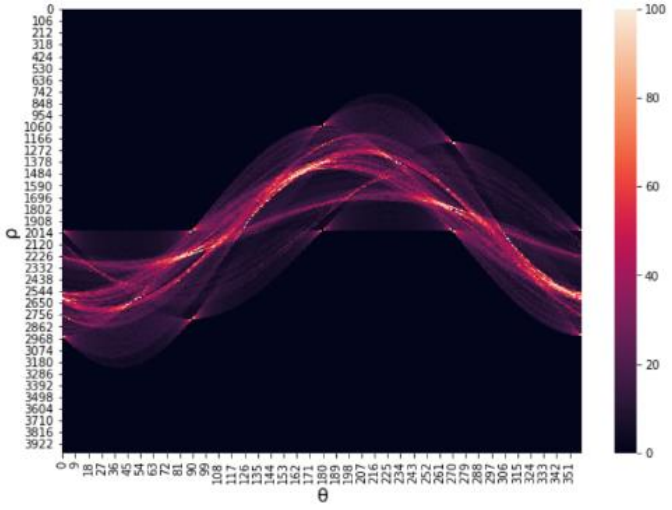
A. Create a cumulative matrix (ρ,θ), record the results of each ρ and θ, and accumulate 1 into the corresponding matrix position. We can use this cumulative matrix to get heatmap. The brighter the place, the more likely it is to be a straight line.

$$\rho = x * \cos(\theta) + y * \sin(\theta)$$



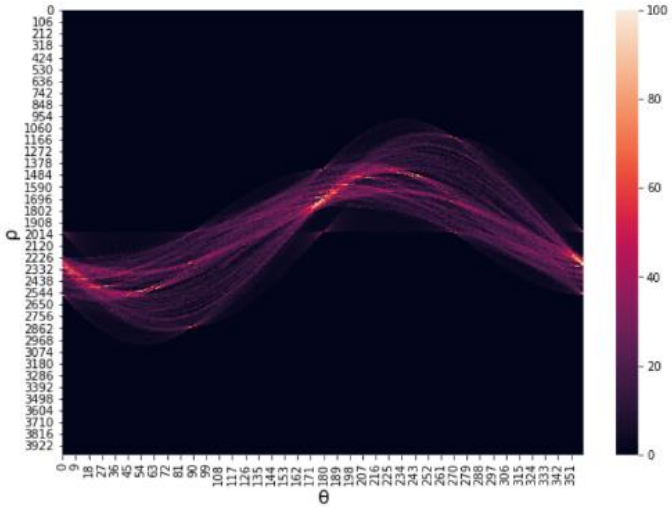
Nanliao Fishing Port

Fish scale ladder(魚鱗天梯)



Hsinchu Xiangshan

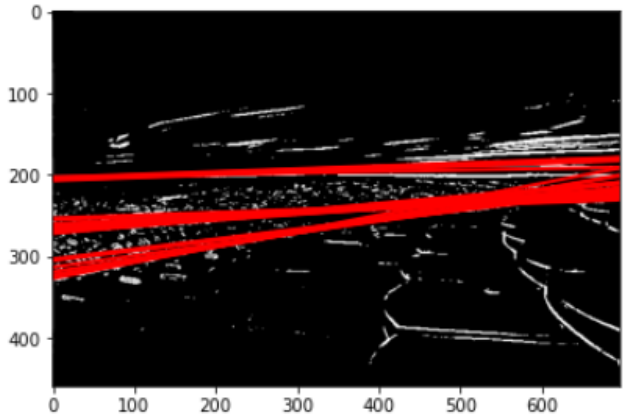
Voice of the Sea(海之聲)



Hsinchu Zhubei

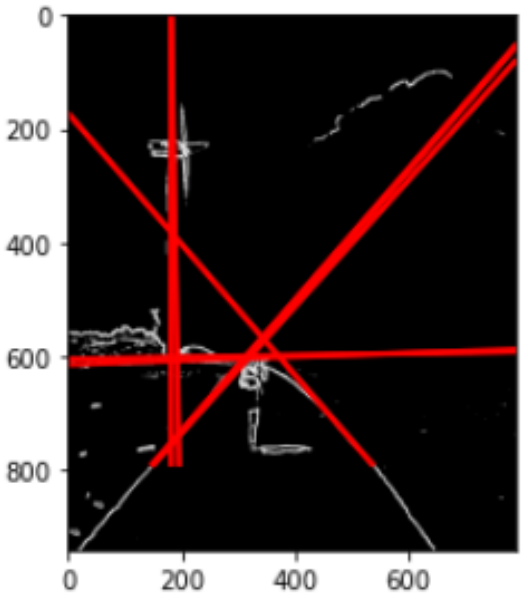
Tofu Rock(豆腐岩)

B. Get the max value of cumulative matrix (ρ, θ), and use the (ρ, θ) to plot straight line.



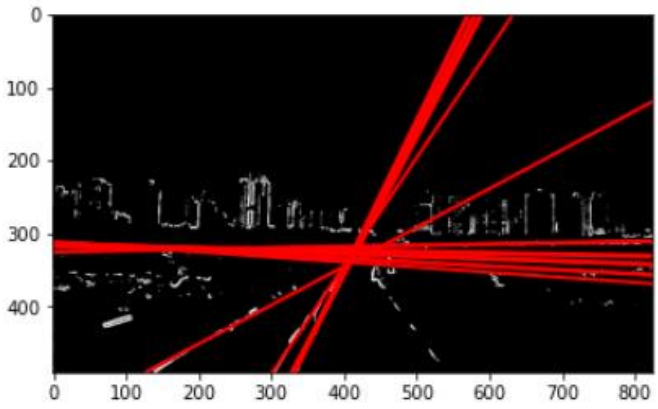
Nanliao Fishing Port

Fish scale ladder(魚鱗天梯)



Hsinchu Xiangshan

Voice of the Sea(海之聲)

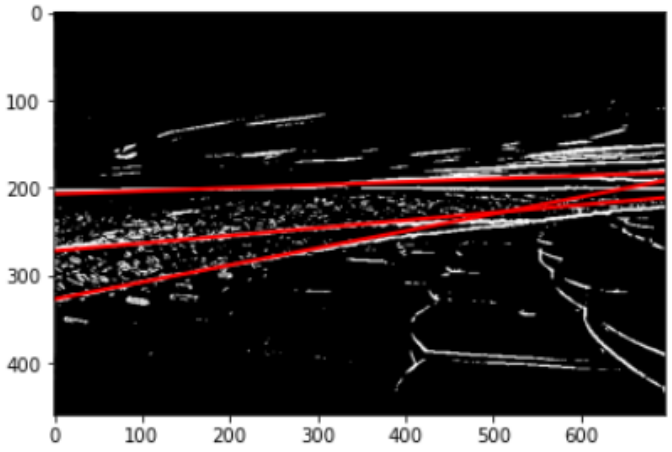


Hsinchu Zhubei

Tofu Rock(豆腐岩)

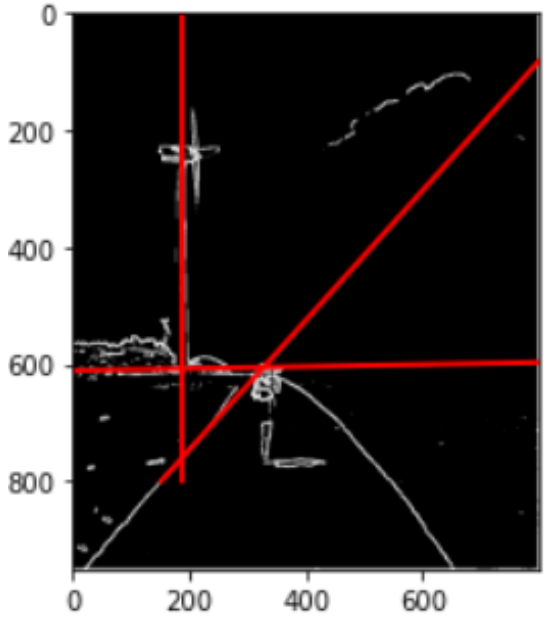
C. Delete the adjacent straight lines and keep the only straight line.

Compute every line the sum ρ+θ. If (ρ1+θ1) – (ρ2+θ2) < diff, only one of the straight lines is keep. The diff is a parameter. Use above function can reduce the number of straight lines.



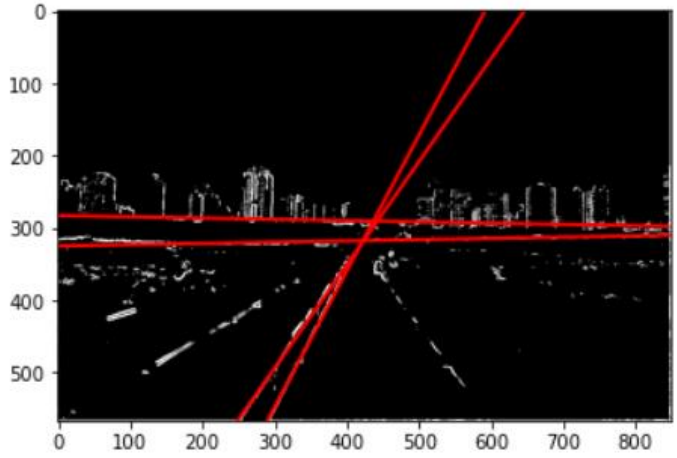
Nanliao Fishing Port
Fish scale ladder(魚鱗天梯)

It is a new attraction in Nanliao. Hsinchu has nine winds(九降風). You can play Windsurfing and kites in that area. It is a tourist attraction suitable for families.



Hsinchu Xiangshan
Voice of the Sea(海之聲)

The Voice of the Sea is very close to the sea, and you can watch the wind turbines up close. It is a suitable place for couples to take.



Hsinchu Zhubei
Tofu Rock(豆腐岩)

Tofu Rock is a new attraction in Touqian Creek(頭前溪) in Hsinchu, which is famous for the wave-removing blocks that delay the deepening of the river bed, and are similar to blocks of tofu. Exception, the row of house prices on the river bank is the most expensive place in Zhubei.

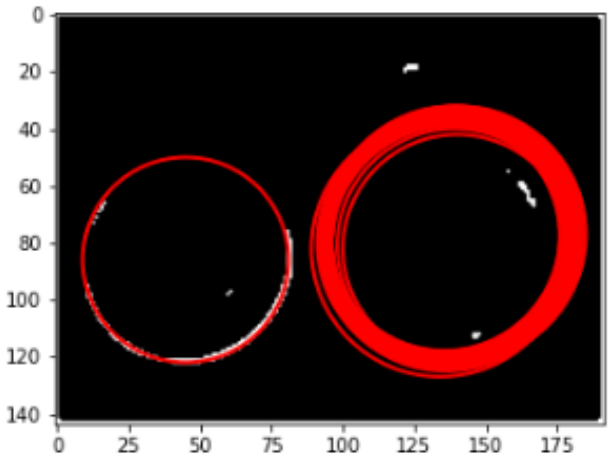
4. Hough transform for cycles

   A. Create a cumulative matrix (a, b, radius), record the results of each a, b, radius and accumulate 1 into the corresponding matrix position.

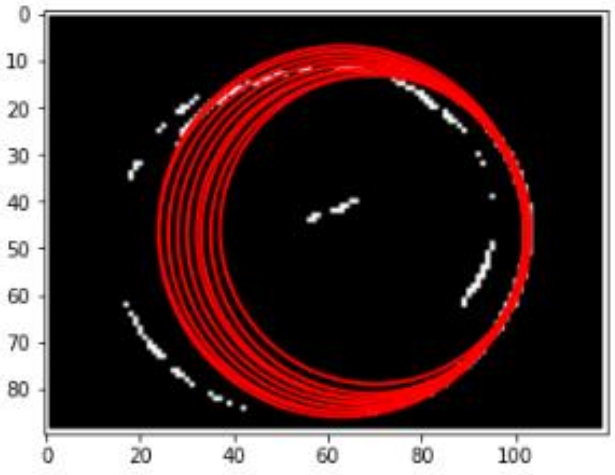
$$a = x - r * cos(\theta)$$

$$b = y - r * sin(\theta)$$

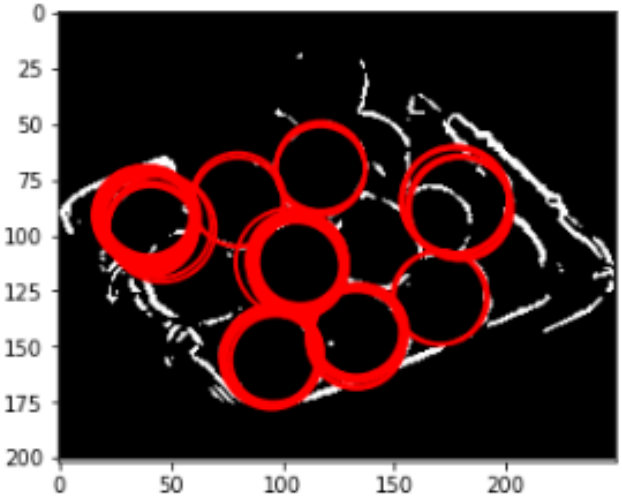
   B. Get the max value of cumulative matrix (a, b, radius), and use the (a, b, radius) to plot cycle.



Afu
Braised Pork Rice
(阿富 魯肉飯)

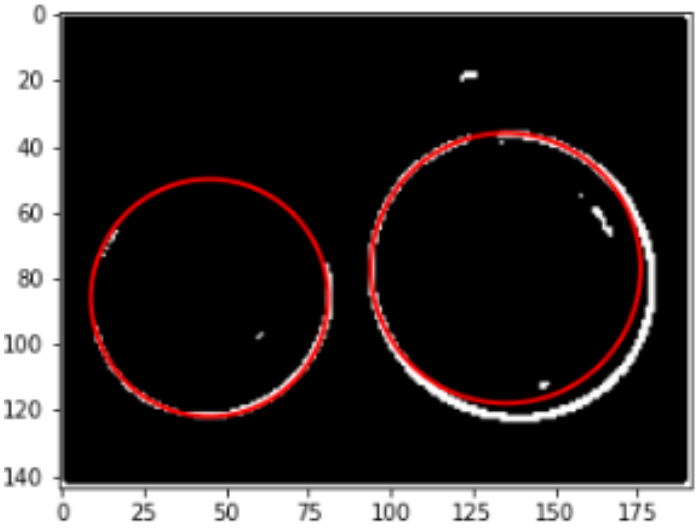


Xin Meizhen
Pudding Cake
(新美珍 布丁蛋糕)



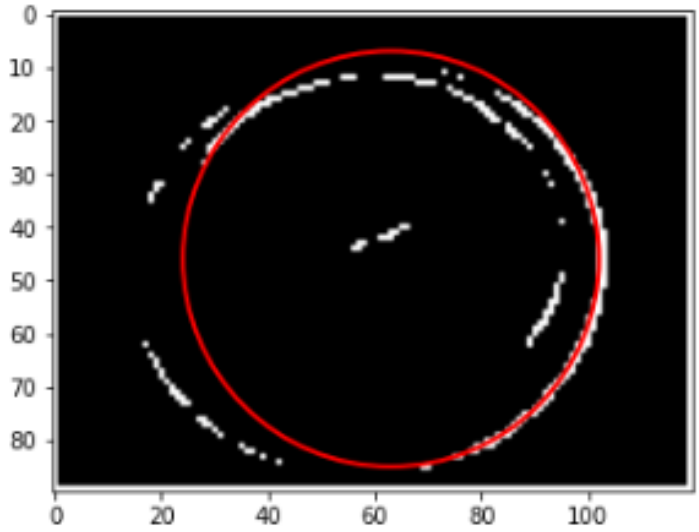
Shijia
Fish Ball
(石家 魚丸)

C. Delete the adjacent cycle and keep the cycle.



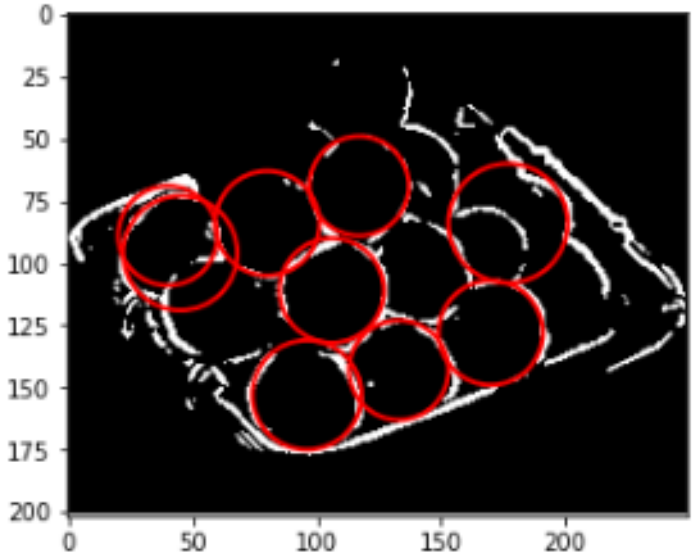
Afu
Braised Pork Rice
(阿富 魯肉飯)

Located next to the Chenghuang Temple(城隍廟), it is a simple dish that is super delicious with half-boiled eggs.



Xin Meizhen
Pudding Cake
(新美珍 布丁蛋糕)

Located in the alley of Xionglin(芎林), the aroma of the cake from the afternoon is very attractive. It is a must-eat ancient cake in Hsinchu.



Shijia
Fish Ball
(石家 魚丸)

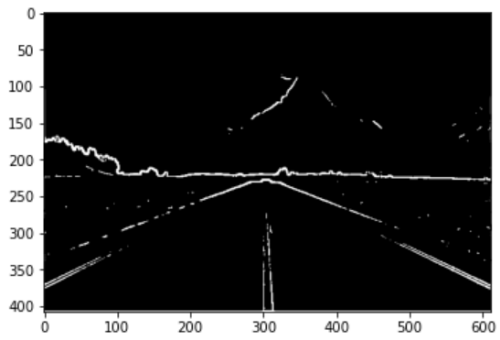
Shijia Fish Ball is a 70-year-old store that insists on hand-making the freshest Hsinchu wrapped fish balls.

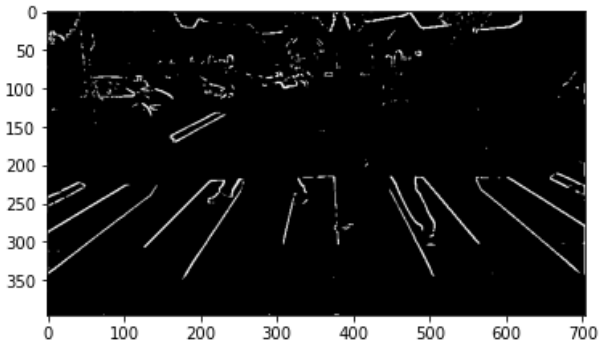
Result

The food and beauty of Hsinchu will definitely not be lost to other regions, but there is no good publicity skills, and the government has not planned tourism facilities well. I hope that the most suitable tourist attractions in Taiwan in the future will be Hsinchu City's first place.

Additional try

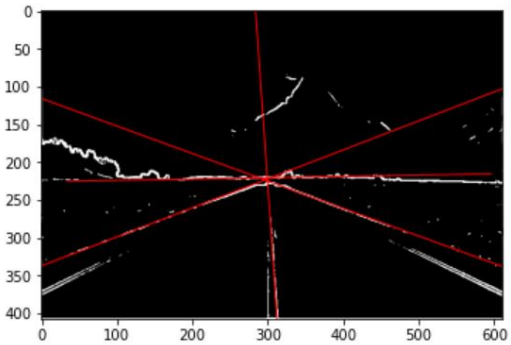
I also use teacher’s image to try Hough transform for line and cycle. But the edge detected by the sobel filter is very rough. Maybe next time, I will use other edge detection methods, such as Canny algorithm to improve Hough transform.

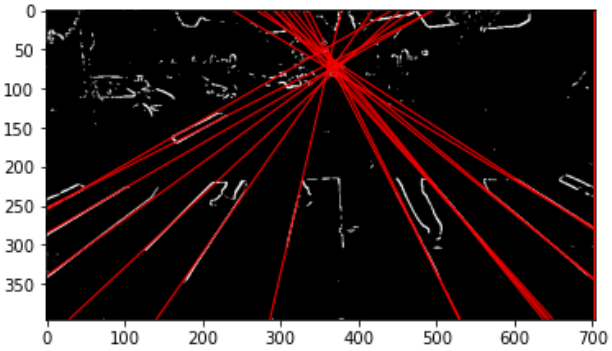


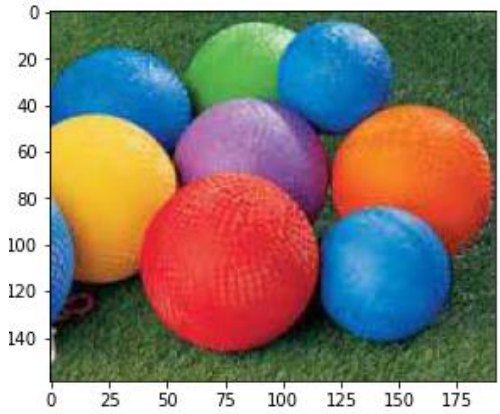


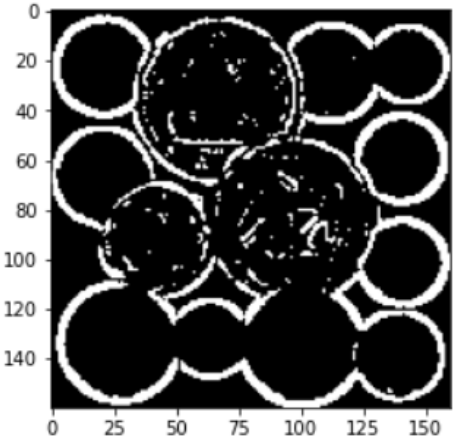
Threshold=70

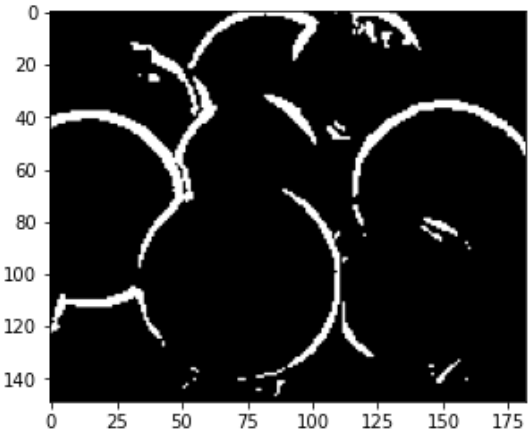
Threshold=100





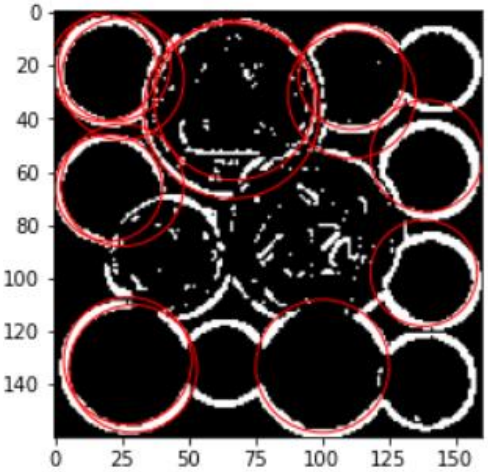


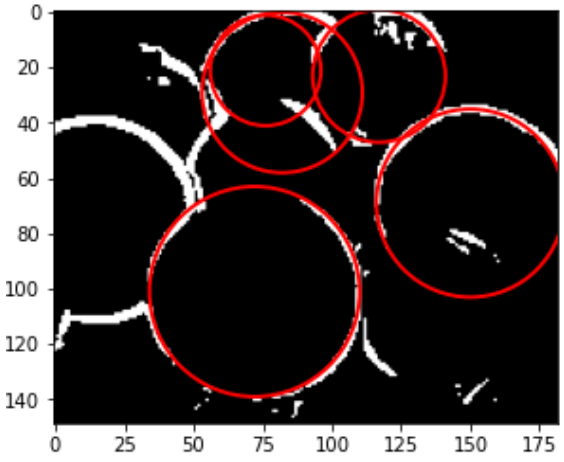




Threshold=90

Threshold=40





Code:

Image preprocessing

A. load images

```
def readImg2Gray(path, resize=False):
    image     = plt.imread(path)
    if resize:
        image = cv2.resize(image, None, fx=0.2, fy=0.2)

    image_gray = rgb2gray(image)
    showImg(image)
    showImg(image_gray)
    return image_gray
```

B. to gray

```
#  Y' = 0.299 R + 0.587 G + 0.114 B
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
```

1. Smoothing Filters

A. Gaussian filter

```
def gaussian_filter(a, kernel, sigma):
    '''
    padding: padding size
    kernel : kernel size
    stride : stride
    sigma  : smooth is control by σ，σ more bigger is more smooth and more blur
    '''
    padding = kernel//2
    kernel = kernel
    stride = 1   # same size of input and output

    # create empty matrix
    temp = np.zeros((a.shape[0] + 2*padding, a.shape[1] + 2*padding))
#     temp = np.full(shape=(a.shape[0] + 2*padding, a.shape[1] + 2*padding), fill_value=255)
    blur = np.zeros((a.shape[0], a.shape[1]))

    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            temp[i+padding][j+padding] = a[i][j]

    # gaussian kernel
    x, y = np.mgrid[-(kernel//2):(kernel//2)+1, -(kernel//2):(kernel//2)+1]
    gaussian_kernel = (1/(2*pi*sigma**2))*np.exp(-(x**2+y**2)/(2*(sigma**2)))

    # gaussian filter
    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            blur[i][j] = (temp[stride*i:stride*i+kernel, stride*j:stride*j+kernel]*gaussian_kernel).sum()

    return blur
```

2. Edge Detection

A. Sobel filter

```
def Filter(a, Type, kernel=3):
    '''
    padding: padding size
    kernel: kernel size
    stride: stride
    '''
    padding = kernel//2
    kernel = kernel
    stride = 1   # same size of input and output

    # create empty matrix
    temp = np.zeros((a.shape[0] + 2*padding, a.shape[1] + 2*padding))
#     temp = np.full(shape=(a.shape[0] + 2*padding, a.shape[1] + 2*padding), fill_value=255)    # create background is black
    blur = np.zeros((a.shape[0], a.shape[1]))
    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            temp[i+padding][j+padding] = a[i][j]

    # kernel
    if Type == "sobel_h":
        KERNEL = np.array([[-1,-2,-1],
                           [0,0,0],
                           [1,2,1]])
    elif Type == "sobel_v":
        KERNEL = np.array([[-1,0,1],
                           [-2,0,2],
                           [-1,0,1]])
    elif Type == "prewitt_h":
        KERNEL = np.array([[-1,-1,-1],
                           [0,0,0],
                           [1,1,1]])
    elif Type == "prewitt_v":
        KERNEL = np.array([[-1,0,1],
                           [-1,0,1],
                           [-1,0,1]])
    else:
        KERNEL = np.array([[0,-1,0],
                           [-1,4,-1],
                           [0,-1,0]])

    # filter
    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            blur[i][j] = (temp[stride*i:stride*i+kernel, stride*j:stride*j+kernel]*KERNEL).sum()

    return blur
```

```
def sobel(image):
    sobel_h = Filter(image, Type="sobel_h", kernel=3)
    sobel_v = Filter(image, Type="sobel_v", kernel=3)
    sobel   = np.absolute(sobel_h) + np.absolute(sobel_v)
    return sobel
```

B. Threshold

```
def scale(image, threshold=70):
    shape = image.shape
    image_scaled = minmax_scale(image.ravel(), feature_range=(0,255)).reshape(shape)
    image_scaled[image_scaled >= threshold] = 255
    image_scaled[image_scaled < threshold] = 0
    return image_scaled
```

3. Hough transform for lines

   A. Create a cumulative matrix (ρ,θ), record the results of each ρ and θ, and accumulate 1 into the corresponding matrix position.

```
def cumulativeMatrix(edges, Maxrho):
    """
    theta = angle * (pi / 180)
    rho   = x * cos(theta) + y * sin(theta)
    :return A:(rho, angle) Cumulative Matrix
    """
    A = np.zeros((2*Maxrho, 360))              # rho range(-2000~2000)
    whiteIndexs = np.argwhere(edges == 255)    # whiteIndexs: gray scale=255 index
    print("calculate point number:", len(whiteIndexs))

    # cumulative
    for index in whiteIndexs:
        x, y = index[0], index[1]
        for angle in range(360):
            theta = angle * (pi / 180)
            rho   = int(x * cos(theta) + y * sin(theta) + 0.5 )   # round(rho)
            A[rho + Maxrho][angle] += 1
    return A
```

```
def lineInfo(A, linenumber=30):
    """
    Get the (rho, angle) of max value in cumulative matrix.

    :linenumber: number of line
    :return: list [[rho, angle],[rho, angle]]
    """
    sortedvalue = np.sort(A, axis=None)[::-1][:linenumber]

    ras = []
    for sv in sortedvalue:
        ra = np.argwhere(A==sv)[0].tolist()
        if ra not in ras:
            ras.append(ra)

    return ras
```

B. Get the max value of cumulative matrix (ρ, θ), and use the (ρ, θ) to plot straight line.

```
def line_point(rho, theta):
    """
    :rho = x*cos(theta)+y*sin(theta)
    :return y_plot:type list
            x_plot:type list
    """
    x = np.linspace(0, edges.shape[1], 10000)     # np.linspace(開始,結束,樣本數)
    y = (rho - x * cos(theta)) / sin(theta)

    x_plot, y_plot = [], []
    for i in range(len(x)):
        if y[i] > 0 and x[i] < edges.shape[0] - 1 and y[i] < edges.shape[1]-1:
            x_plot.append(x[i])
            y_plot.append(y[i])

    return y_plot, x_plot
```

C. Delete the adjacent straight lines and keep the only straight line.

```
def deleteAdjacentLine(ras, diff=30):
    """
    Delete the adjacent lines and keep the only straight line.
    :method: if (rho2 + angle2)-(rho1 + angle1) < diff then delete line [rho2,angle2].

    :diff: maximum difference
    :ras: [[rho, angle],[rho, angle]]
    :return: [[rho, angle],[rho, angle]]
    """
    wantdeletes = []
    sorted_ras = sorted(ras)
    for i in range(len(sorted_ras)):
        for j in range(i+1, len(sorted_ras)):
            if abs(sum(sorted_ras[j]) - sum(sorted_ras[i])) < diff:
                wantdeletes.append(sorted_ras[j])
    for wantdelete in wantdeletes:
        if wantdelete in sorted_ras:
            sorted_ras.remove(wantdelete)
    return sorted_ras
```

4. Hough transform for cycles

A. Create a cumulative matrix (a, b, radius), record the results of each a, b, radius and accumulate 1 into the corresponding matrix position.

```
def cumulativeMatrix3D(edges, padding=2000):
    """
    :padding: padding the cumulative matrix record more information.
    :method:
        radius: >10
        angle : 0~360
        a = x - r * cos(angle*(pi/180))
        b = y - r * sin(angle*(pi/180))
    :return A:(a, b, radius) Cumulative Matrix
    """
    restore = padding//2

    radius = min(edges.shape[0], edges.shape[1])//2          # radius=minimum image length or width
    A      = np.zeros((edges.shape[0] + padding, edges.shape[1] + padding, radius))   # A padding=2000
    whiteIndexs = np.argwhere(edges == 255)                  # whiteIndexs:(x,y) gray scale=255 index
    print("calculate point number:", len(whiteIndexs))

    # cumulative
    n = 0
    for index in whiteIndexs:                        # (x,y)
        x, y = index[0], index[1]
        for r in range(20, radius):                  # radius
            for angle in range(360):
                a = int(x - r * cos(angle*(pi/180)) + 0.5)
                b = int(y - r * sin(angle*(pi/180)) + 0.5)

                if a >= A.shape[0] + restore:              # a, b bigger than padding
                    continue
                if b >= A.shape[1] + padding:
                    continue

                A[a + restore, b + restore, r] += 1

        if n%1000==0:
            print("{} finish".format(n))
        n += 1
    return A
```

B. Get the max value of cumulative matrix (a, b, radius), and use the (a, b, radius) to plot cycle.

```
def lineInfo(A, cyclenumber=30):
    """
    Get the (rho, angle) of max value in cumulative matrix.

    :cyclenumber: number of line
    :return: list [[a,b,r], [a,b,r]]
    """
    sortedvalue = np.sort(A, axis=None)[::-1][:cyclenumber]
    abrs = []
    for sv in sortedvalue:
        abr = np.argwhere(A==sv)[0].tolist()
        if abr not in abrs:
            abrs.append(abr)

    return abrs
```

C. Delete the adjacent cycle and keep the cycle.

```
def deleteAdjacentCycle(abrs, diff=30):
    """
    Delete the adjacent Cycles.
    :method: if (a2 + b2 + r2)-(a1 + b1 + r1) < diff then delete cycle [a2,b2,r2].

    :diff: maximum difference
    :ras: [[a,b,r], [a,b,r]]
    :return: [[a,b,r], [a,b,r]]
    """

    wantdeletes = []
    sorted_abrs = sorted(abrs)
    for i in range(len(sorted_abrs)):
        for j in range(i+1, len(sorted_abrs)):
            if abs(sum(sorted_abrs[j]) - sum(sorted_abrs[i])) < diff:    #
                wantdeletes.append(sorted_abrs[j])

    for wantdelete in wantdeletes:
        if wantdelete in sorted_abrs:
            sorted_abrs.remove(wantdelete)
    return sorted_abrs
```