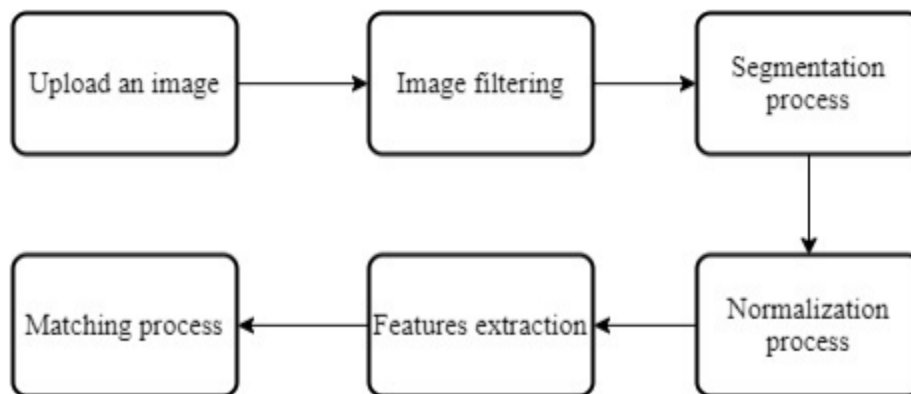


ASSIGNMENT-2: BIOMETRIC SECURITY

IRIS RECOGNITION SYSTEM

Kashish jain, 2021jcs2240

FLOWCHART of proposed system:



Initially two images: image, image2, are given as input from CASIA database.

```
#image = cv2.imread('image_102.png')  
image = cv2.imread('img5.jpg')  
image2 = cv2.imread('img5.jpg')
```

Matching is performed between these given images: image, image2.

ENHANCEMENT: Initially I performed Image enhancement process, in which using cv2.medianblur() function, and np.clip(), I removed noise from the image, also increased the brightness of the image.

To remove noise, cv2.medianblur(img, size_of_filter) is used to replace central element of the image by the median of all the pixel in the kernel area.

To increase brightness, np.clip(pow(i / 255.0, gamma) * 255.0, 0, 255), is used to perform gamma correction function, where the new value= pow(i / 255.0, gamma) * 255.0.

SEGMENTATION: Then in segmentation process, I have used Hough circle transform algorithm to perform segmentation on image.

To detect circles in images, cv2.HoughCircles() function is used. It's first parameter is the input image, and the other parameter is circle detection method.

```
circles = cv2.HoughCircles(img, cv2.HOUGH_GRADIENT, 1, 20,  
                           param1=60, param2=30, minRadius=1, maxRadius=40)
```

mindist=20, which is 4th parameter, is important parameter, used to set the minimum distance between the center(x,y) coordinates of detected circles. If its value is too small then many false circles are detected, if its value is too large then some circles might not be detected.

NORMALIZATION: After performing segmentation on input image, I performed Normalization process. Normalization process converted the circular iris of image into rectangular bock of fixed size.

In this process, polar coordinates of pixels of iris region are used to find cartesian coordinates of pixels.

Detected circles returned by segmentation process are used to determine the polar coordinates.

```
x=int(xp + rp * math.cos(theta[i]))  
y=int(yp + rp * math.sin(theta[i]))
```

This code is used to perform this function, where theta is between 0 and 2pi and xp, rp, yp are the values retuned by segmentation process.

```

while i<phase_width:
    #calculate the cartesian coordinates for the beginning and the end pixels
    x=int(xp + rp * math.cos(theta[i]))
    y=int(yp + rp * math.sin(theta[i]))
    x1=int(xi + ri * math.cos(theta[i]))
    y1=int(yi + ri * math.sin(theta[i]))
    # generate the cartesian coordinates of pixels between the beginning and end pixels
    xspace=np.linspace(x, x1, iris_width)
    yspace=np.linspace(y, y1, iris_width)
    #calculate the value for each pixel
    lis=[]
    for x, y in zip(xspace, yspace):
        if 0 <= int(x) < img.shape[1] and 0 <= int(y) < img.shape[0]:
            lis=lis+[255 - img[int(y), int(x)]]
        else:
            lis=lis+[0]
    iris[:, i]=lis
    i=i+1

return iris

```

This algorithm is used to determine the value of pixels from these cartesian coordinates.

FEATURE EXTRACTION: After performing normalization, features extraction function is performed.

I have used 2Dgabor wavelets, to extract the structure of iris as a sequence of vectors in complex plane. Also phase angles of these vectors are quantized to set the bits in the iris code. These vectors in complex plane, also called as phasors.

```

for i, row in enumerate(patches):
    for (j, p) in product(enumerate(row), enumerate([8, 16, 32])):

        xx, yy=np.meshgrid(np.linspace(0, 1, p.shape[0]), np.linspace(-np.pi, np.pi, p.shape[1]))
        t=np.exp(-w * 1j * (0- yy)) * np.exp(-(xx - 0) ** 2 / alpha ** 2) * np.exp(-(-yy + 0) ** 2 / beta ** 2)

        wavelet= np.array([np.linspace(0, 1, p.shape[0]) for i in range(p.shape[1])]).T * p * np.real(
            t.T), np.array([np.linspace(0, 1, p.shape[0]) for i in range(p.shape[1])]).T * p * np.imag(t.T)
        if True:
            code[3 * i + k, 2 * j + 1]=np.sum(wavelet[1])
        if True:
            code[3 * i + k, 2 * j]=np.sum(wavelet[0]) # calculate the real part

        # code[3 * i + k, 2 * j + 1]=np.sum(wavelet[1]) # calculate the imaginary part
        if mask[i, j].sum() > dr * dtheta * 3 / 4:
            code_mask[3 * i + k, 2 * j]=code_mask[3 * i + k, 2 * j + 1]=1
        else:
            code_mask[3 * i + k, 2 * j]=code_mask[3 * i + k, 2 * j + 1]=0

```

This function is used to implement the equations of 2D gabor wavelets which are then used to apply the 2D gabor wavelets on the image.

So, Firstly, phase information is obtained after applying filter to image using 2D gabor wavelets, then wavelets are generated of different frequency to extract more information.

$$h(\text{Re}, \text{Im}) = \text{sgn}(\text{Re}, \text{Im}) \int \rho \int \varphi I(\rho, \varphi) e^{-j\omega(\theta_0 - \varphi)} e^{-(r_0 - \rho)^2 / \alpha^2} e^{-(\theta_0 - \varphi)^2 / \beta^2} \rho d\rho d\varphi$$

is the equation which is used to extract the phase information. $I(\rho, \phi)$ represents pixel intensity in polar coordinate, α and β represents the parameters of analysis multi scale, r_0 and θ_0 are the coordinates on which wave is applied.

MATCHING PROCESS: After the feature extraction process, I performed matching process, in which I have used hamming distance to measure the dissimilarity between any two iris images. If hamming distance lies in the range [0,0.38], then iris images are considered to be matched, otherwise not.

Formula for hamming distance is given as:

$$\text{HD} = \frac{\|(\text{codeA} \otimes \text{codeB}) \cap \text{maskA} \cap \text{maskB}\|}{\|\text{maskA} \cap \text{maskB}\|}.$$

Where maskA, maskB are mask bit vector and codeA, codeB are phase code bit vectors.

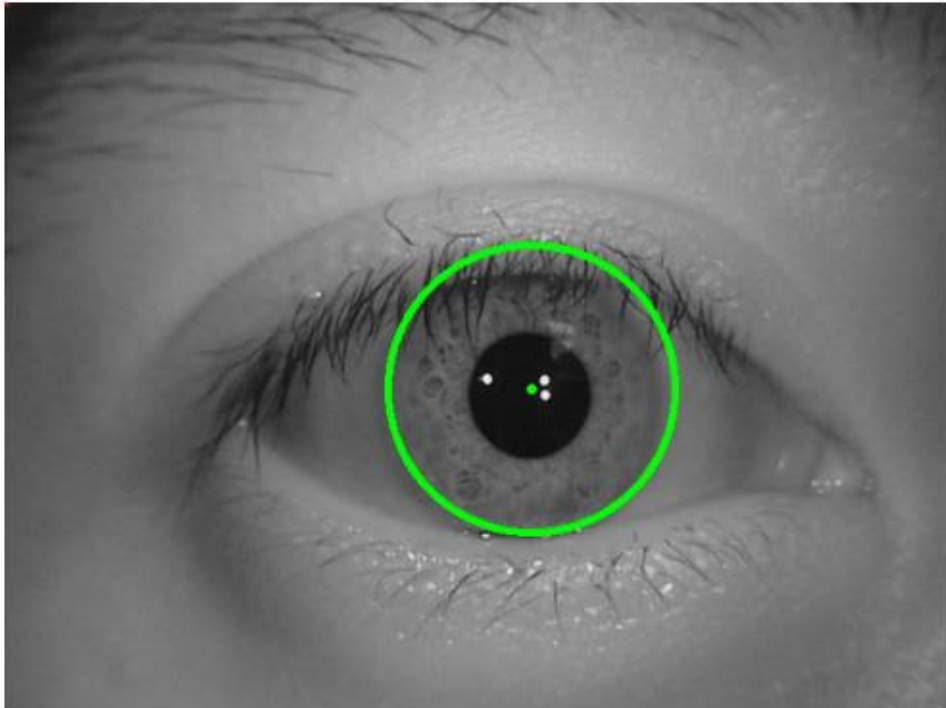
RESULTS:

For the Input images:

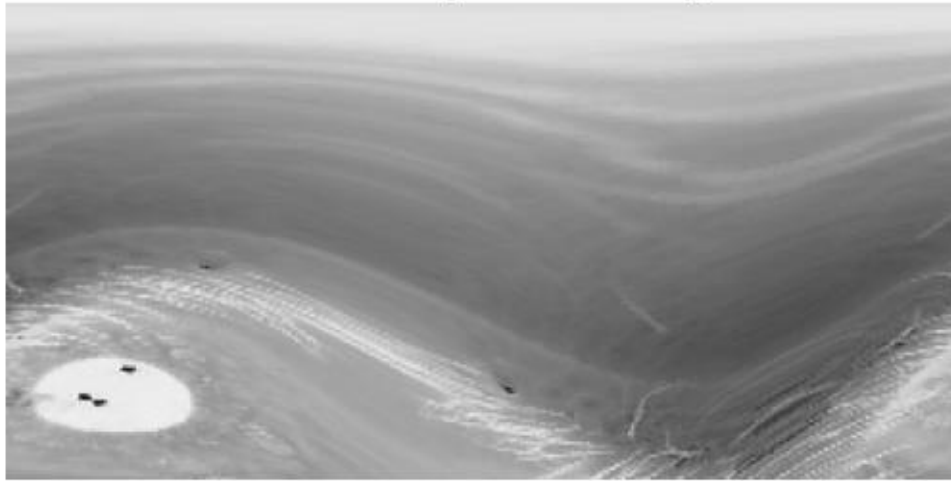
Image1=img5.jpg

Image2=img4.jpg

Segmentation process on image1



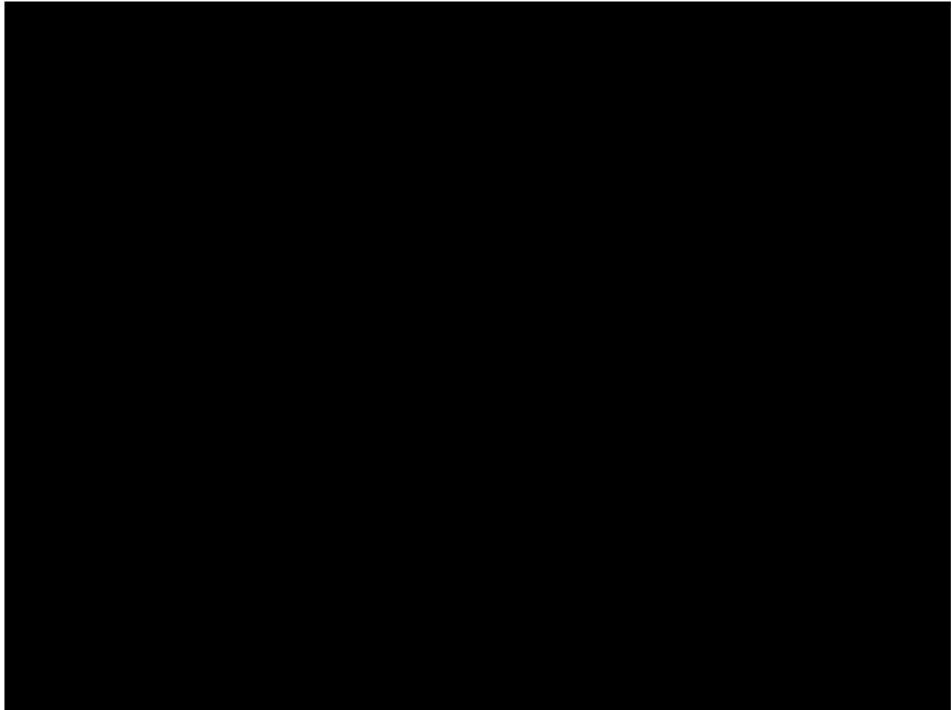
Normalization process on image1



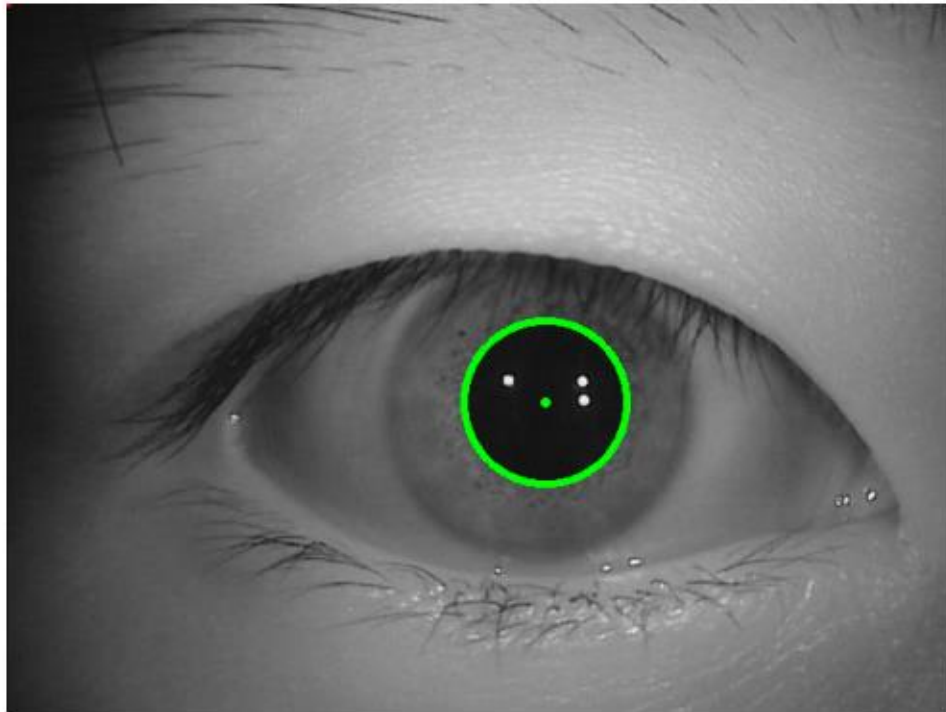
Iris code of image1



Mask code of image1



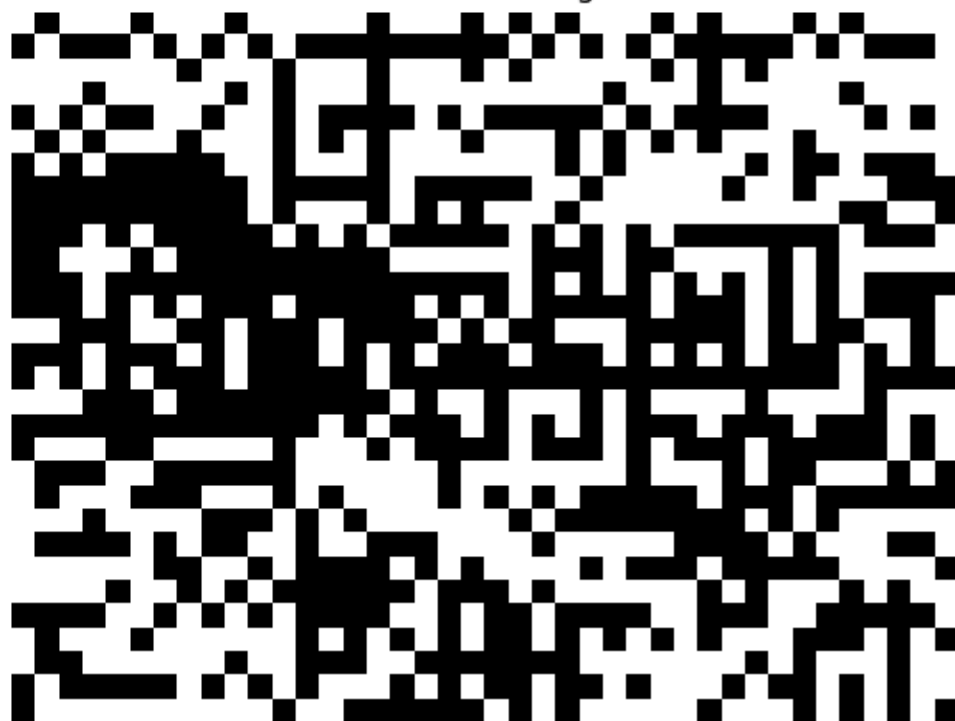
Segmentation process on image2



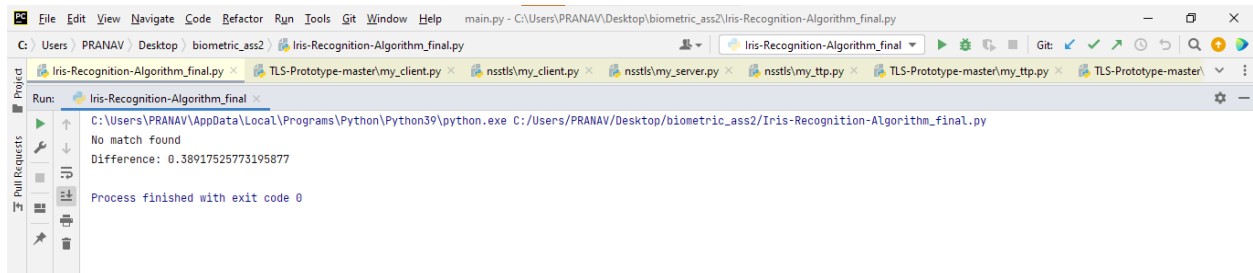
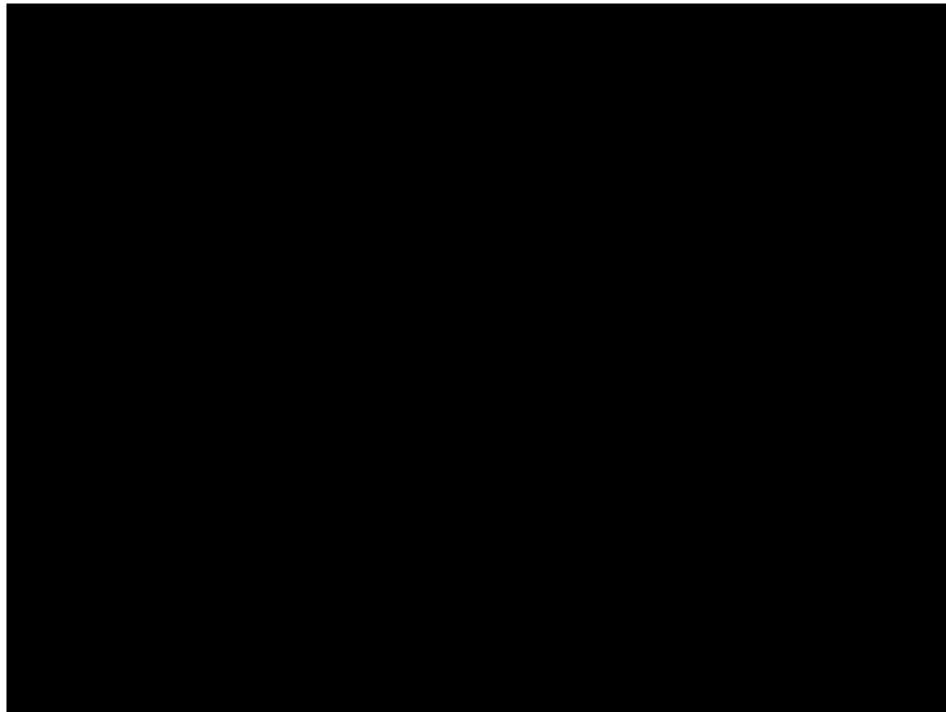
Normalization process on image2



Iris code of image2



Mask code of image2

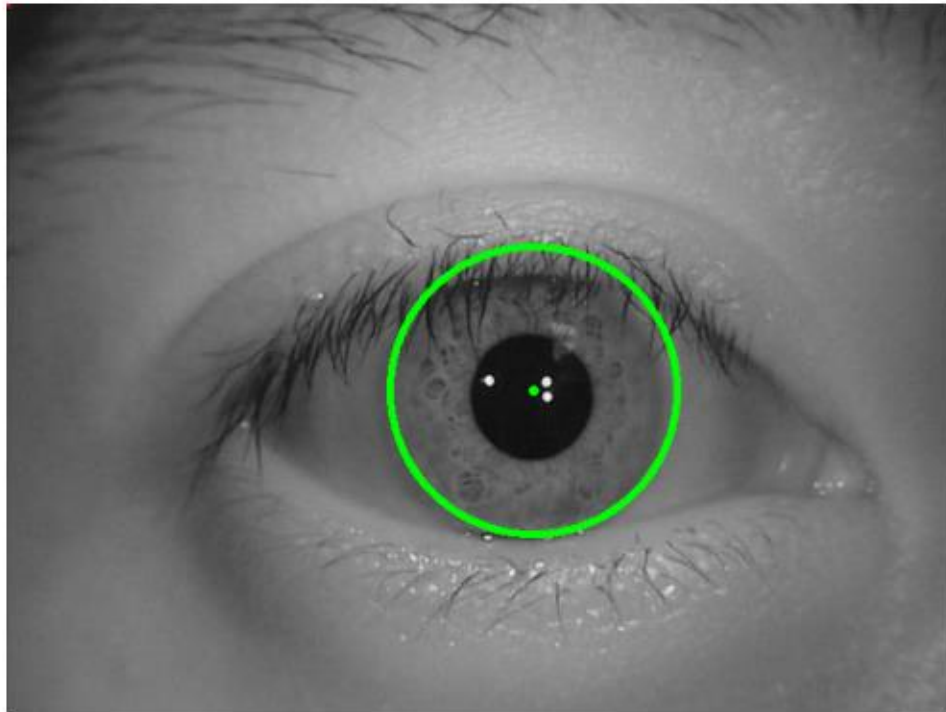


For input images:

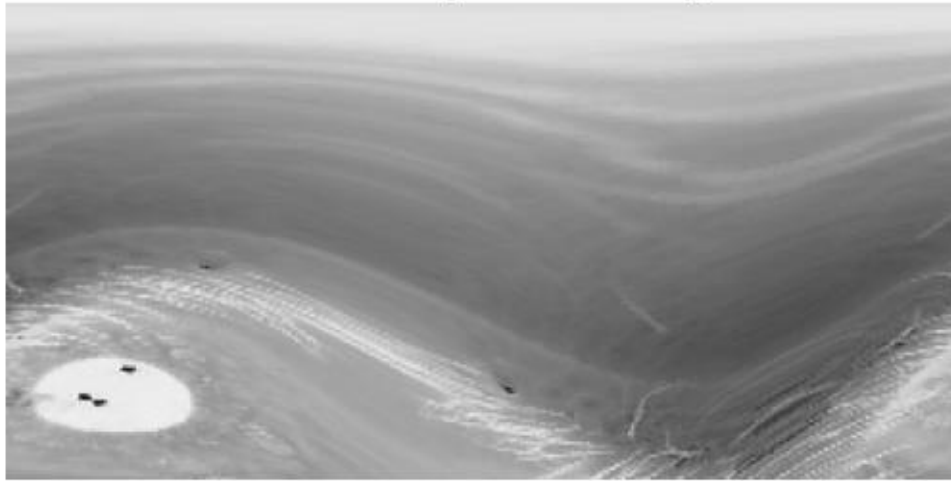
Image1=img5.jpg

Image2=img1.jpg

Segmentation process on image1



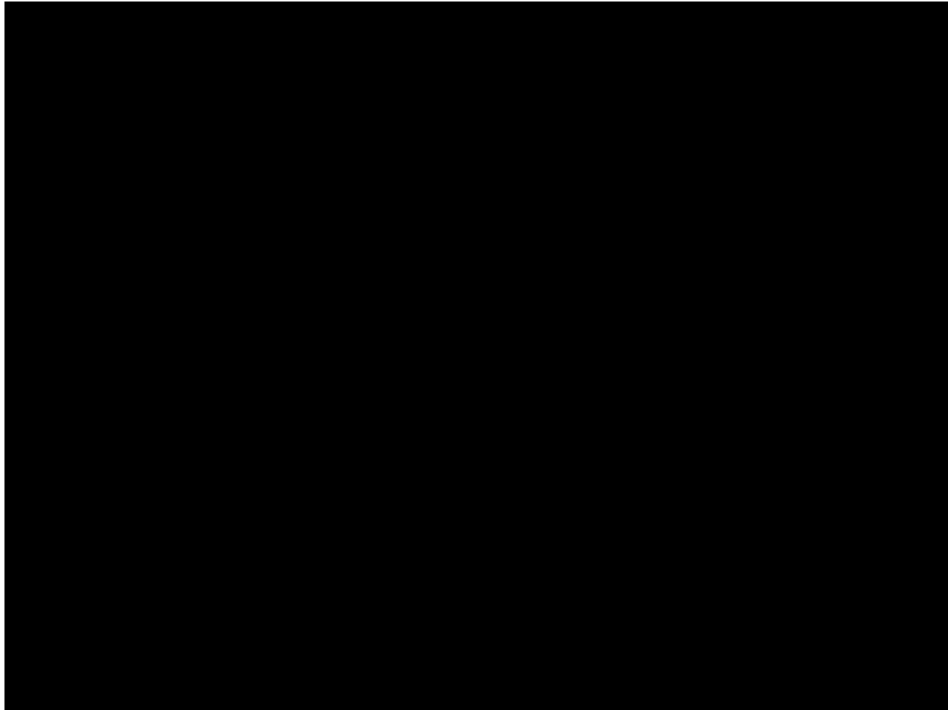
Normalization process on image1



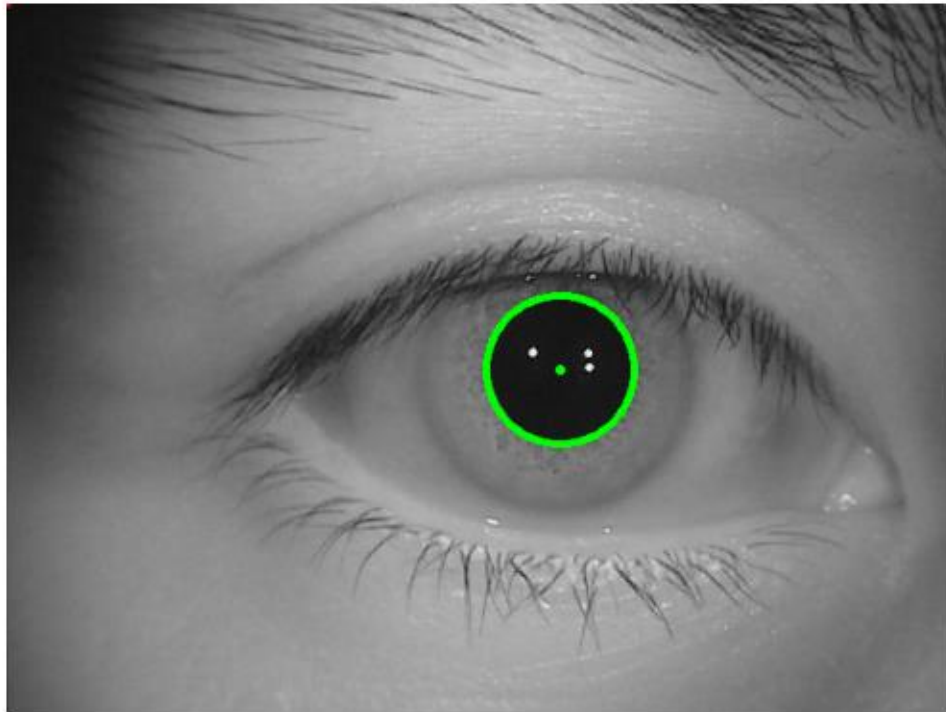
Iris code of image1



Mask code of image1



Segmentation process on image2



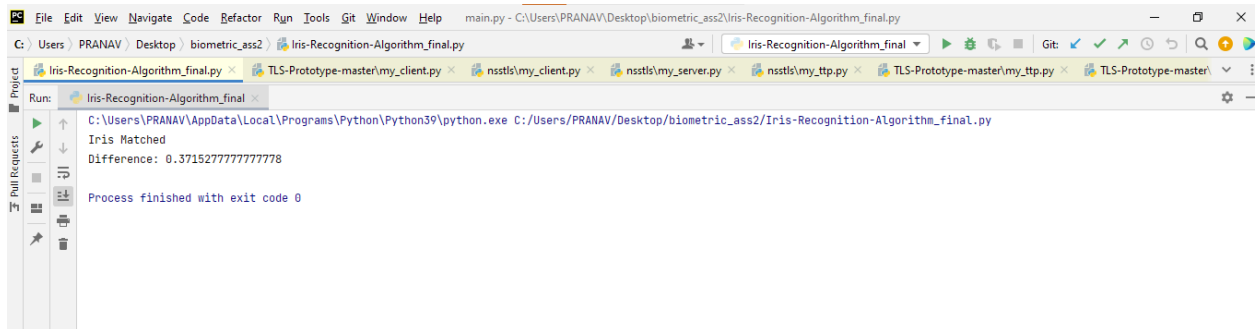
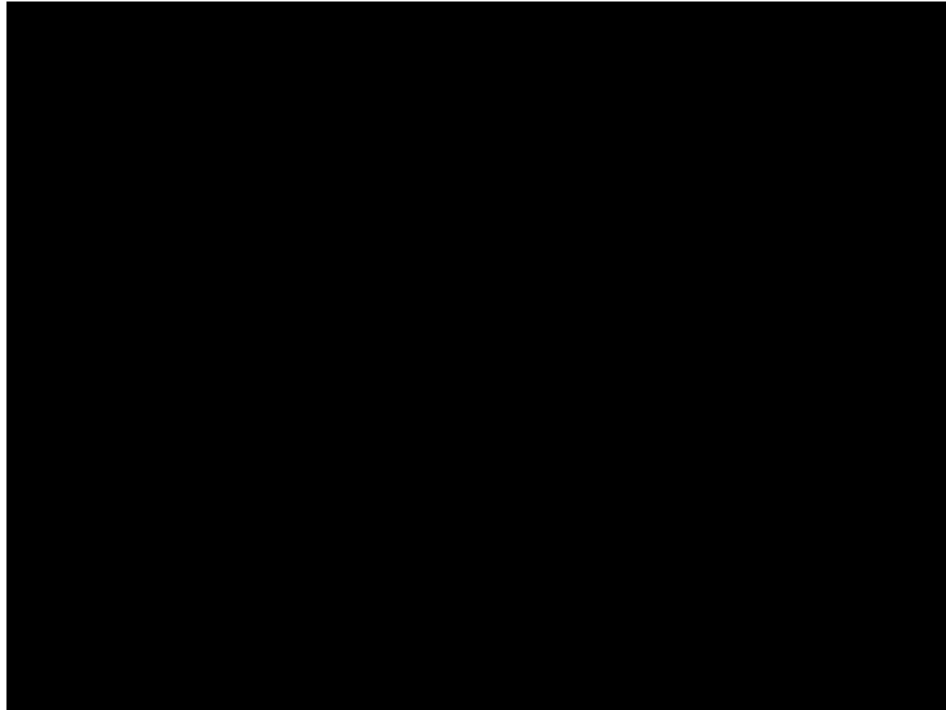
Normalization process on image2



Iris code of image2



Mask code of image2



```
File Edit View Navigate Code Refactor Run Tools Git Window Help main.py - C:\Users\PRANAV\Desktop\biometric_ass2\Iris-Recognition-Algorithm_final.py
C:\Users\PRANAV\Desktop\biometric_ass2\Iris-Recognition-Algorithm_final.py
Iris-Recognition-Algorithm_final.py TLS-Prototype-master\my_client.py nsstls\my_client.py nsstls\my_server.py nsstls\my_ttp.py TLS-Prototype-master\my_ttp.py TLS-Prototype-master\
Run: Iris-Recognition-Algorithm_final
C:\Users\PRANAV\AppData\Local\Programs\Python\Python39\python.exe C:/Users/PRANAV/Desktop/biometric_ass2/Iris-Recognition-Algorithm_final.py
Iris Matched
Difference: 0.3715277777777778
Process finished with exit code 0
```