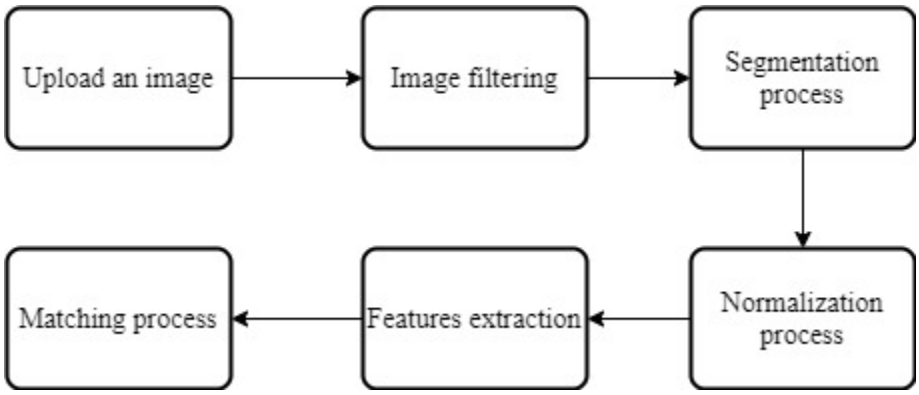
Implementation of the recognition algorithm

The data flow of the recognition algorithm is presented in the following figure:



In filtering process noise from the image is removed and the brightness of image is increased.

- cv2.medianBlur(img, 5) - cv.medianBlur() takes the median of all the pixels under the kernel area and the central element is replaced with this median value. The size of the filter is determined by the second parameter;
- np.clip(pow(i / 255.0, gamma) * 255.0, 0, 255) – this line implement the gamma correction function : new_value=255*(old_value/255)$^{\gamma}$.

If $\gamma>1$ brightness increase and if $\gamma<1$ brightness decrease.

Segmentation process is implemented using Hough circle transform.

circles = cv2.HoughCircles(img, cv2.HOUGH_GRADIENT, 1, 20, param1=60, param2=30, minRadius=20, maxRadius=100)

Parameters of this function are :

- img - grayscale input image;
- cv2.HOUGH_GRADIENT - detection method;
- 1 - represents the inverse ratio of the accumulator resolution to the image resolution. If this is 1, the accumulator has the same resolution as the input image;
- 20 - minimum distance between the centers of the detected circles;
- param1=60 - method specific parameter. It is the higher threshold of the two passed to the Canny edge detector;
- param2=30 - method specific parameter. It is the accumulator threshold for the circle centers at the detection stage;
- minRadius=20 - minimum circle radius;

- maxRadius=100 – maximum circle radius.

Normalization – after the segmentation process the circular iris region is transformed into a fixed size rectangular block. To do this, we calculate the cartesian coordinates for every pixel from iris region using polar coordinates of these pixels. The polar coordinates are determined using the two circles detected in the segmentation process, one who delimits the iris region and another one delimits the pupil region.

```
def polar2cart(r, x0, y0, theta):
    x = int(x0 + r * math.cos(theta))
    y = int(y0 + r * math.sin(theta))
    return x, y
```

r, x0, y0, are the parameters returned by segmentation process and theta has the value between 0 and 2 π.

The function below calculates the Cartesian coordinates and the value of the pixels determined by these coordinates.

```
for i in range(phase_width):
    #calculate the cartesian coordinates for the beginning and the end pixels
    begin = polar2cart(rp, xp, yp, theta[i])
    end = polar2cart(ri, xi, yi, theta[i])
    #calculate the cartesian coordinates of pixels between the beginning and end pixels
    xspace = np.linspace(begin[0], end[0], iris_width)
    yspace = np.linspace(begin[1], end[1], iris_width)
    #calculate the value for each pixel
    iris[:, i] = [255 - img[int(y), int(x)]
                  if 0 <= int(x) < img.shape[1] and 0 <= int(y) < img.shape[0]
                  else 0
                  for x, y in zip(xspace, yspace)] #assign the cartesian coordinates
return iris
```

Features extraction – this process is implementing using 2D Gabor wavelets. This process uses complex-valued 2D Gabor wavelets to extract the structure of the iris as a sequence of phasors (vectors in the complex plane), whose phase angles are quantized to set the bits in the iris code. The main idea of this method is that: firstly we construct two-dimensional Gabor wavelet, and take it to filter the image, and after that we get phase information. This method use only the phase

information in the pattern because the phase angles are assigned regardless of the image contrast. The phase information is extracted by applying the following equation:

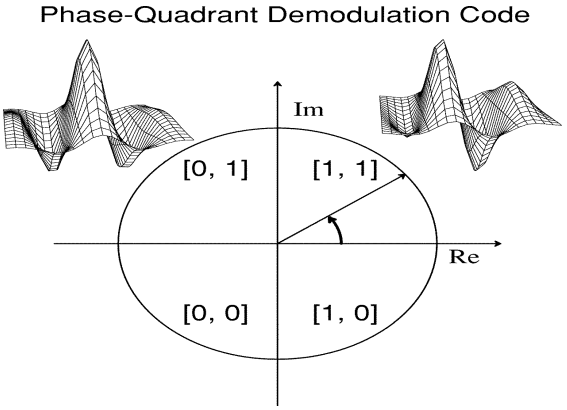
$h(\mathrm{Re},\mathrm{Im}) = \mathrm{sgn}(\mathrm{Re},\mathrm{Im}) \int \rho \int \varphi\, I(\rho,\varphi) e - j\omega(\theta 0 - \varphi)\, e - (r0 - \rho)2/\alpha\, 2e - (\theta 0 - \varphi)2/\beta 2\rho\, d\rho\, d\varphi$

Where I(ρ, φ) represents the intensity of the picture in a polar coordinate. ω is the throbbing of the wavelet of Gabor, r0 and θ0 are the coordinates of the point around of which one applies the wave, and finally α and β are the parameters of the analysis multi scale.

```
#implementation of  equation for 2D Gabor wavelets

def gabor(rho, phi, w, theta0, r0, alpha, beta):
    return np.exp(-w * 1j * (theta0 - phi)) * np.exp(-(rho - r0) ** 2 / alpha ** 2) *\
        np.exp(-(-phi + theta0) ** 2 / beta ** 2)


# application of the 2D Gabor wavelets on the image

def gabor_convolve(img, w, alpha, beta):
    # generate the parameters
    rho = np.array([np.linspace(0, 1, img.shape[0]) for i in range(img.shape[1])]).T
    x = np.linspace(0, 1, img.shape[0])
    y = np.linspace(-np.pi, np.pi, img.shape[1])
    xx, yy = np.meshgrid(x, y)
    return rho * img * np.real(gabor(xx, yy, w, 0, 0, alpha, beta).T), \
        rho * img * np.imag(gabor(xx, yy, w, 0, 0, alpha, beta).T)
```

To extract more information we generate wavelets with different frequency:

```
for k, w in enumerate([8, 16, 32]):
```



Either φ the phase of transformed it of 2D Gabor wavelets, and h codes it generated:

- if $0 < \varphi < \pi/2$ then h = [1,1] ;
- if $\pi/2 < \varphi < \pi$ then h = [0,1] ;
- if $\pi < \varphi < 3\pi/2$ then h = [0,0] ;
- if $3\pi/2 < \varphi < 2\pi$ then h = [1,0] .

In matching process is used Hamming Distance (HD) as the measure of the dissimilarity between any two irises, whose two phase code bit vectors are denoted {codeA, codeB} and whose mask bit vectors are denoted {maskA, maskB}:

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

If the calculated distance is in the range [0, 0.38] we consider a match.