IRIS RECOGNITION

MINI PROJECT developed for Course: Image Processing
under Guidance of: Dr. Dhirendra S. Mishra

29/03/2020

AAKASH SHETTY E034
MEHERANAND TIKKAS E047
JHANVI UDANI E053

B. Tech. | Computer Engineering | Semester VI| Division: E | AY 2019-20

Contents

Biometrics are biological — or physical — measures that can be used to identify people. Fingerprint scanning, facial recognition, and retinal scanners are all examples of biometric technology, but these are just the most known choices. Iris recognition is the most accurate and precise method of biometric identification. Iris recognition system captures a snapshot of an individual's pupil and the iris in the image is segmented and normalised to obtain the distinctive feature. Segmentation is the most important aspect of the process of iris recognition as areas which are incorrectly segmented out as iris regions can compromise biometric models resulting in very poor recognition. Precise iris segmentation is crucial for the effectiveness and accuracy of the subsequent extraction and recognition functionality, and hence the high-performance level of the iris recognition system.

The keys to the development of biometric technologies are multiple, including the result of positive physical proofs and the removal of the need to recall passwords or identifiers that can be quickly overlooked, borrowed or stolen and Increasing demands for security in routine conditions, in particular identity recognition. Owing to its high potential in practical applications, iris detection has quickly become one of the most researched biometric topics and is undoubtedly one of the most accurate biometric identification methods. Iris biometrics makes use of the extremely rich and characteristic texture details between the dark pupil and the white sclera in the annular region.

Segmentation of iris is one of the essential processing measures in an iris identification procedure. The key objective of this iris segmentation phase is to classify the correct area of the iris for recognition purposes. However, upper and lower eyelids, eyelashes, light reflections, shadows obstruct iris texture regions. Iris segmentation thus includes the localization of the eyelids and the removal of the effects of occlusions caused by eyelashes, shadows and light reflections. The performance of the iris segmentation method adopted directly affects the overall efficiency of Iris recognition. On the one hand, it is important for the high quality of the derived iris characteristics used for identification, while, on the other, it is a determining factor of the biometrically real-time response since it is the most time-consuming element of an iris recognition system.

Based upon the description above, there are three major and challenging issues in iris segmentation: (a) Precise pupillary and limbic boundary identification, in particular for non-ideal iris and noisy pictures; (b) appropriate care for occlusions

arising from shadows, eyelids, eyelashes, etc; (c) real-time iris recognition in practical applications.

With the rapid development of emerging technology and the popularisation of biometric applications, manufacturing costs of highly advanced biometric devices, including eye / iris image cameras, have decreased. Around the same time, the quality of the eye / iris photographs obtained has become progressively higher. There is no denying that improved iris image quality will lead to even higher iris recognition systems and even improve iris segmentation algorithms without losing recognition efficiency. It should also be noticed that the occluded iris areas invariantly lose their iris texture information. Thankfully, redundant information provided by the iris features collected will mitigate, if not absolutely eliminate, the negative effect of the occlusion of the iris region.

3. Software /Hardware Used

One of the remaining iris recognition issues is the design of the effective hardware systems. The problem is that the methods that form part of this sort of authentication mechanism are relatively complex. Additionally, next-generation iris detection algorithms should become much more sophisticated to boost the efficiency and performance of currently existing solutions.

Hardware Specifications:

- CPU: Intel Core i5-8210Y (Dual Core, 4 threads, 4MB L3 cache, Base Freq: 1.6Ghz, Boost Freq: 3.6Ghz)
- RAM: 8GB (2,133Mhz LPDDR3)

Software Specifications:

- Operating System: MacOS
- Language Used: Python
- IDE: PyCharm

4. Implementation: Explain the algorithm implemented and its block diagram.

Imported libraries:

matplotlib.pyplot :

pylab is a module within the matplotlib library that was built to mimic MATLAB's global style. It exists only to bring a number of functions and classes from both

NumPy and matplotlib into the namespace, making for an easy transition for former MATLAB users who were not used to needing import statements.

Used for:

In our project pyplot is used for plotting images for visual representation and study. It is used to plots circles to show definition for edges of pupil and iris. We have also used to plot and show images of iris in intermediate processes.

OpenCv :

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.The library has more than 2500 optimised algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognise faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects etc.

Used for:

In our project pyplot is used for hough circles it has helped us to detect circles and its centre to aid us in creating circles. Used for median Blur to get rid of noises by processing edges. It is used to read images that are entered by the user or to read from database.

Numpy:

Numpy is a library that used for computing scientific/mathematical data. Numpy to create multi-dimensional array(2D, 3D etc), now in order to achieve that, Numpy must be supported in Python so we will import Numpy in the code editor (different code editors have different styles of importing or in-built plugins to import Numpy) Other usages are in:

- Numerical Analysis
- Linear algebra
- Matrix computations

Used for:

In our project numpy is used for managing several calculations like abosule etc. Moreover, it is also used to handle matrix calculations and allows us better management of arrays for its manipulation and better results.

Code Explanation:

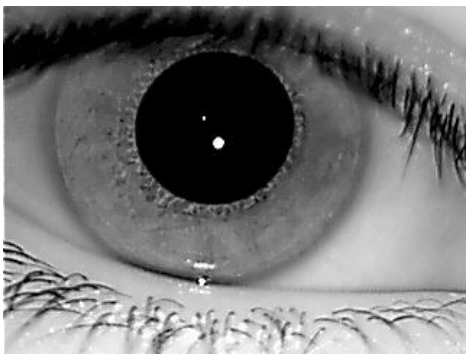
Segmentation:

```
if __name__ == '__main__':
    # data = load_utiris()['data']
    image = cv2.imread('iris1.bmp')
    newImage = image.copy()
    img = preprocess(image)
    img1 = preprocess(newImage)
    x, y, r = find_pupil_hough(img)
    x_iris, y_iris, r_iris = find_iris_hough(img1)
    show_segment(image, x, y, r, x_iris, y_iris, r_iris)
```

Image Acquisition:

This section is a part of the code that helps acquire images from source like a user of datasets. Image is acquired using **IMREAD** for further processing. This image is later used for preprocessing and then is taken for the segmentation process which can help us decide how the images should be dealt for further processing. The functions called in the above images is used to process the input image.

Input Image:

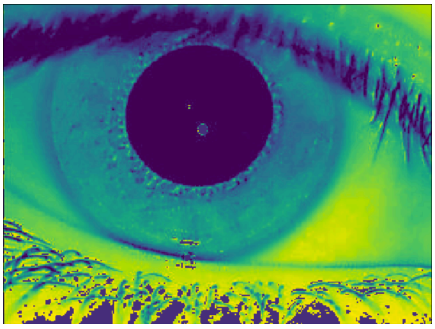


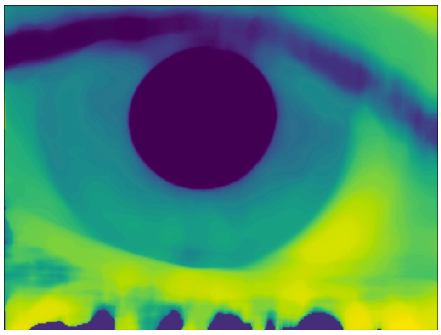
```python
def preprocess(image):

    img = image[:, :, 0].copy()
    img[img > 225] = 30
    return cv2.medianBlur(img, 21)
```

Image Preprocessing:

Image that are acquired are sent for preprocessing where we have used thresholding to segregate the pupil from the rest of the image and then sequentially applied medianBlur as we require a defined edge for the boundaries. It also helps get rid of any noise if present in the image. This all processes is done on a copy of the original image since the original image is required for further varied processes.

Output Image:





```python
def find_pupil_hough(img):

    circles = cv2.HoughCircles(img, cv2.HOUGH_GRADIENT, 1, 20,
                               param1=60, param2=50, minRadius=5, maxRadius=100)
    circles = np.uint16(np.around(circles))
    return circles[0, 0][0], circles[0, 0][1], circles[0, 0][2]
```

Detecting pupil Coordinates:

We need to develop a circle that encapsulates the pupil to get segmentation. Thus, we need the coordinates of the pupil centre and it's radius. For this process we have used the Hough Circle that takes parameters:

(input image, circle detection method, inverse ratio of resolution, Minimum distance between detected centres, Upper threshold for the internal Canny edge detector, Threshold for centre detection, minimum and maximum radius of the circle)

```
Python Console
>>> runfile('/Users/aakash/Desktop/SEM VI/IP/project/iris-recognition-master/rough.py',
144
82
54

>>>
```

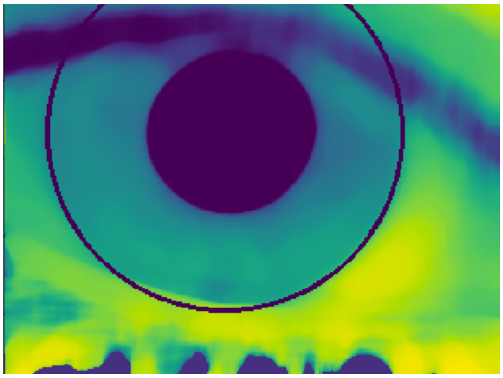
Output:

```
def find_iris_hough(img):

    circles = cv2.HoughCircles(img, cv2.HOUGH_GRADIENT, 1, 20,
                               param1=60, param2=50, minRadius=10, maxRadius=200)
    circles = np.uint16(np.around(circles))


    for i in circles[0,:]:
        # draw the outer circle
        cv2.circle(img, (i[0], i[1]), i[2], (0, 255, 0), 2)

    plt.imshow(img)
    plt.show()
    return circles[0, 0][0], circles[0, 0][1], circles[0, 0][2]
```

Detecting Iris Coordinates:

Similar to the pupil detection iris recognition also makes use of the hough circle to get the radius for circle detection with same parameters except the one for the minimum radius as the radius has to be greater than the pupil's the radius is taken higher. Then the required circle is plotted to get a rough idea of only the iris segment.

Output Image:

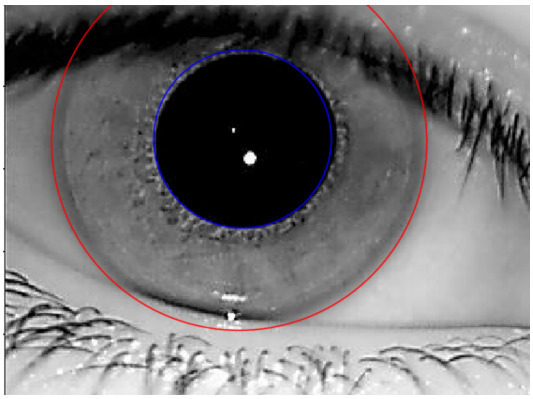


```
def show_segment(img, x, y, r, x2=None, y2=None, r2=None):

    ax = plt.subplot()
    ax.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    segment = plt.Circle((x, y), r, color='b', fill=False)
    ax.add_artist(segment)
    if r2 is not None:
        segment2 = plt.Circle((x2, y2), r2, color='r', fill=False)
        ax.add_artist(segment2)
    plt.show()
```

Display the segments:

We have used a pot and subplot to display the final segmentation with the circle cooler as blue and red to get the final clear segmentation for the viewer. The circles are then plotted with the help of the centre coordinates and the radius for the pupil and the iris. The blue Circle encloses the pupil and the red circle encloses the iris.

Output Image:



Recognition:

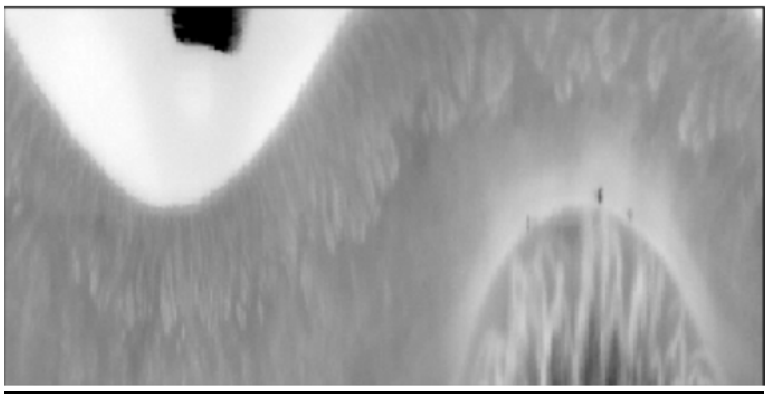
```
if __name__ == '__main__':
    # data = load_utiris()['data']
     image = cv2.imread('iris.bmp')
     image2 = cv2.imread('iris1.bmp')
     print(image.shape)
     print(image2.shape)
     code, mask = encode_photo(image)
     code2, mask2 = encode_photo(image2)
     print(compare_codes(code, code2, mask, mask2))
```

Image Acquisition:

Two Images are taken from the user to compare the iris are the same or different. Using functions the two iris are compared with each other through functions and by encoding and comparing those encoded values.

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


def unravel_iris(img, xp, yp, rp, xi, yi, ri, phase_width=300, iris_width=150):
    if img.ndim > 2:
        img = img[:, :, 0].copy()
    iris = np.zeros((iris_width, phase_width))
    theta = np.linspace(0, 2 * np.pi, phase_width)
    for i in range(phase_width):
        begin = polar2cart(rp, xp, yp, theta[i])
        end = polar2cart(ri, xi, yi, theta[i])
        xspace = np.linspace(begin[0], end[0], iris_width)
        yspace = np.linspace(begin[1], end[1], iris_width)
        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)]
    return iris
```

Unwrapping Iris:



Encoding the unwrapped iris:

Images are encoded using a mathematical formula. The centre coordinates and the radius is taken as input and using these values a systematic mathematical code is generated. For this linespace is used which generates random values and is applied

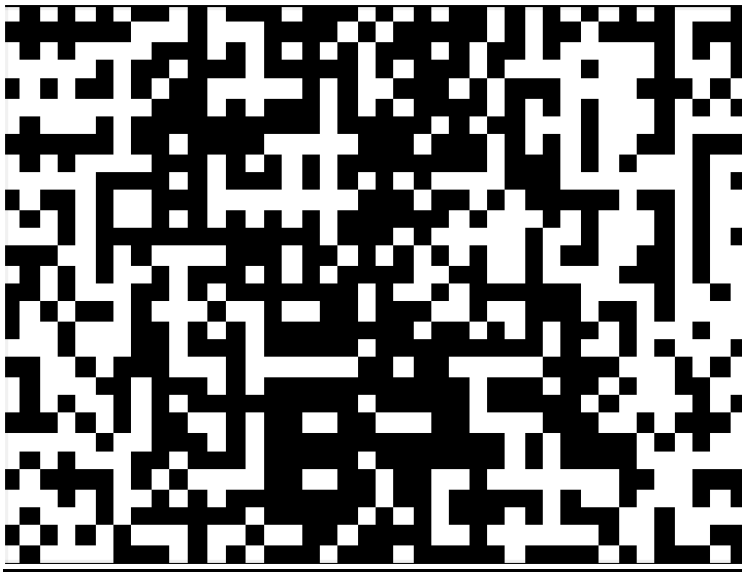
to develop unique pattern for representation of the segmented area. This system of coordinates generated is further sent for processing for feature extraction.

Feature Extraction from Encoded information:

The Gabor feature extraction method is used for the encoded values from the previous step and is later extracted as digital information. This digital information is as unique as the iris. The convolution is done and a code is generated for the development of a monochrome digital information which captures the uniqueness of the image.

```
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)


def gabor_convolve(img, w, alpha, beta):

    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, 0.5, alpha, beta).T), \
           rho * img * np.imag(gabor(xx, yy, w, 0, 0.5, alpha, beta).T)


def iris_encode(img, dr=15, dtheta=15, alpha=0.4):

    mask = view_as_blocks(np.logical_and(100 < img, img < 230), (dr, dtheta))
    norm_iris = (img - img.mean()) / img.std()
    patches = view_as_blocks(norm_iris, (dr, dtheta))
    code = np.zeros((patches.shape[0] * 3, patches.shape[1] * 2))
    code_mask = np.zeros((patches.shape[0] * 3, patches.shape[1] * 2))
    for i, row in enumerate(patches):
        for j, p in enumerate(row):
            for k, w in enumerate([8, 16, 32]):
                wavelet = gabor_convolve(p, w, alpha, 1 / alpha)
                code[3 * i + k, 2 * j] = np.sum(wavelet[0])
                code[3 * i + k, 2 * j + 1] = np.sum(wavelet[1])
                code_mask[3 * i + k, 2 * j] = code_mask[3 * i + k, 2 * j + 1] = \
                    1 if mask[i, j].sum() > dr * dtheta * 3 / 4 else 0
    code[code >= 0] = 1
    code[code < 0] = 0
    return code, code_mask
```

Output:

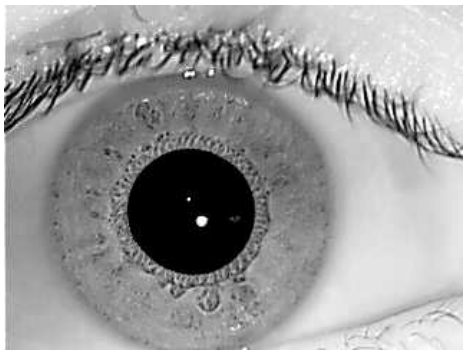


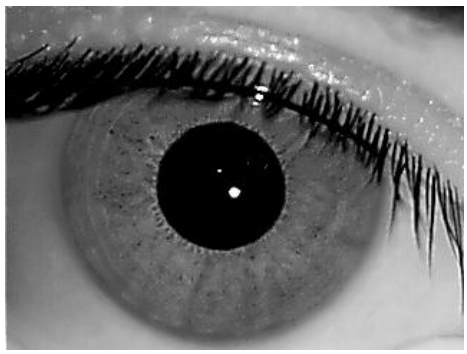
```
def compare_codes(a, b, mask_a, mask_b, rotation=False):
    return np.sum(np.remainder(a + b, 2) * mask_a * mask_b) / np.sum(mask_a * mask_b)
```

Comparing two different iris:

This above code compares between two entered images of iris and returns the difference. If the difference is seen to be 0, then iris is a match, otherwise it is a mismatch.

Input Images:





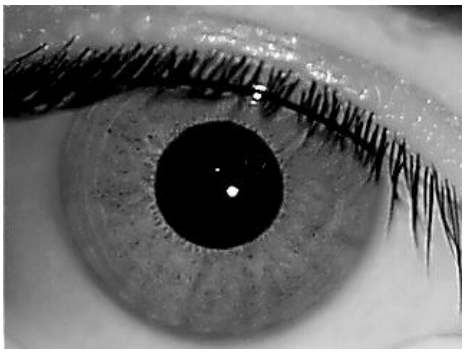
```
if __name__ == '__main__':
    # data = load_utiris()['data']
    image = cv2.imread('iris.bmp')
    image2 = cv2.imread('iris3.bmp')
    code, mask = encode_photo(image)
    code2, mask2 = encode_photo(image2)
    # print(compare_codes(code, code2, mask, mask2))
    if compare_codes(code, code2, mask, mask2) == 0:
        print("Iris Matched")
        print("Differece: 0")
    else:
        print("No Match Found")
        print("Difference: "+str(compare_codes(code, code2, mask, mask2)))
```

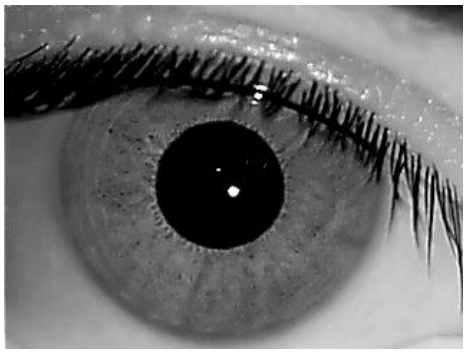
Different Iris:

```
if __name__ == '__main__':
    # data = load_utiris()['data']
    image = cv2.imread('iris3.bmp')
    image2 = cv2.imread('iris3.bmp')
    code, mask = encode_photo(image)
    code2, mask2 = encode_photo(image2)
    # print(compare_codes(code, code2, mask, mask2))
    if compare_codes(code, code2, mask, mask2) == 0:
        print("Iris Matched")
        print("Differece: 0")
    else:
        print("No Match Found")
        print("Difference: "+str(compare_codes(code, code2, mask, mask2)))
```

```
Python Console
>>> runfile('/Users/aakash/Desktop/SEM VI/IP/project/iris-recognition-master/recognition.py', wdir='
No Match Found
Difference: 0.47354497354497355
```

Similar Iris:

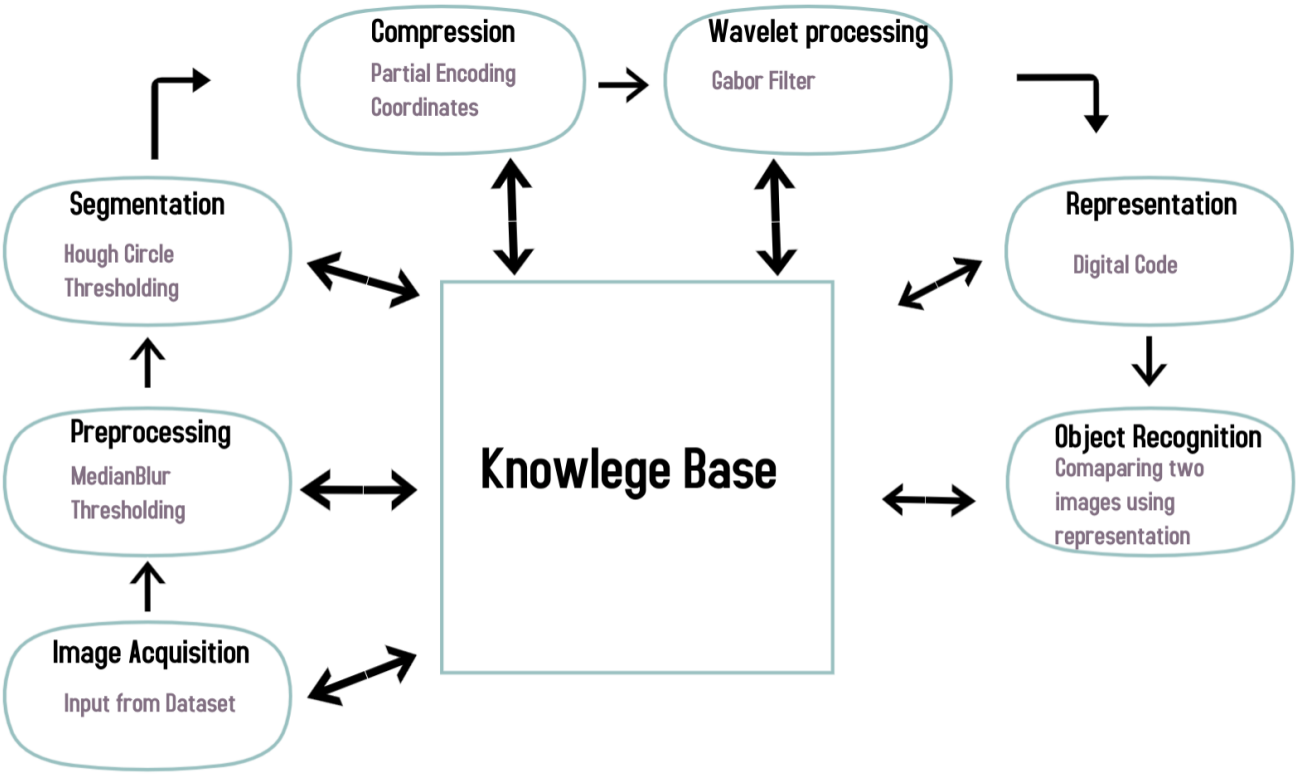




Output:

```
Python Console
>>> runfile('/Users/aakash/Desktop/SEM VI/IP/project/iris-recognition-master/recognition.py', wdir='/Users/aa
Iris Matched
Differece: 0
```

BLOCK DIAGRAM:



5. Image Databases used: Explain the database content in detail.

The images used from the dataset UTIRIS V4. These images are all infrared images in .bmp format. The images are acquired from scanners and contain the eye and the eyelids so that not much of preprocessing time is wasted for finding the pupil and iris.

6. Stages of image processing:

a) Image acquisition:

To deal with images and before analysing them the most important thing is to capture the image. This is called as Image acquisition. We have acquired images

from the UTIRIS open Database. It has 40 images or eyes, both left and right, and this fit our algorithm best.

b) Pre-processing:

- Resizing the images.
- Thresholding
- Applied medianBlur.

c)Techniques applied:

- Applied Hough transform on the image first:

The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing.[1] The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

Snippet of code:

```
def find_pupil_hough(img):
    """Finds the pupil using Hugh transform.

    :param img: Image of an eye
    :return: x, y coordinates of the centre of the pupil and its radius
    """
    circles = cv2.HoughCircles(img, cv2.HOUGH_GRADIENT, 1, 20,
                               param1=60, param2=50, minRadius=5, maxRadius=100)
    circles = np.uint16(np.around(circles))
    return circles[0, 0][0], circles[0, 0][1], circles[0, 0][2]
```

- Image segmentation:

  Algorithm:

  a. Read image of an eye
  b. Starting x coordinate
  c. Starting y coordinate

d. minr: Minimal radius
e. maxr: Maximal radius
f. step: The difference between two consecutive radii in the search space
g. sigma: The amount of blur of integral values before selecting the optimal radius
h. center_margin: The maximum distance from x0, y0 reached to find the optimal segment centre
i. segment_type: Either 'iris' or 'pupil' used to optimize the search
j. jump: The difference between two consecutive segment centres in the search space
k. return: x, y of the segment centre, radius of the segment and integral value matching the optimal result

- Gabor Filter:

  A Gabor filter is a linear filter used for texture analysis, which means that it basically analyses whether there are any specific frequency content in the image in specific directions in a localized region around the point or region of analysis.

  Frequency and orientation representations of Gabor filters are claimed by many contemporary vision scientists to be similar to those of the human visual system, though there is no empirical evidence and no functional rationale to support the idea.

  They have been found to be particularly appropriate for texture representation and discrimination. In the spatial domain, a 2D Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave.

  Some authors claim that simple cells in the visual cortex of mammalian brains can be modeled by Gabor functions. Thus, image analysis with Gabor filters is thought by some to be similar to perception in the human visual system.

d)Hurdles faced:

We were initially unable to find a way to place the red circle of recognition exactly on the iris.

Resolution:

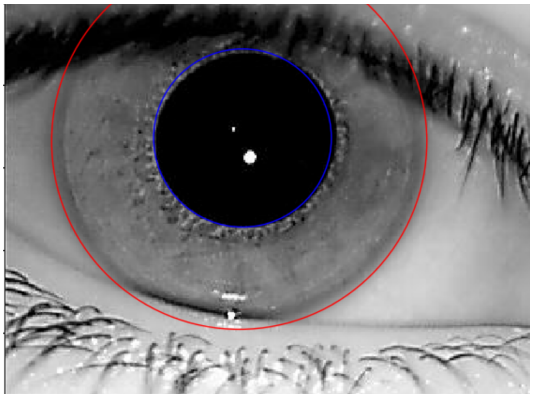
We figured out that our input coordinates were not favourable to our results and we updated them.

e) No post processing was required for this algorithm.

f) Results:

We have been able to achieve desired output.

Segmentation:



Recognition:

```
Python Console
>>> runfile('/Users/aakash/Desktop/SEM VI/IP/project/iris-recognition-master/recognition.py', wdir='
No Match Found
Difference: 0.47354497354497355
```

7. Result Analysis:

Results while highly dependent on the resolution several images did not produce the expected result, some produced red circles on the pupil while some were way out of the iris boundary. Results varies as the parameter of the hour circle function is changes which gives us denser or scarcer circles. The differences would have highly varied if we would have considered a different encoding algorithm or the coordinate generating system, but the final result of recognition would have been unaffected (i.e if the iris were different or same). Result are seen to best where maximum portion of the image is covered by the Iris and pupil and not the eyelids and unnecessary background. Results are slightly affected if the person being scanned is not looking directly into the scanner, Because the circle does not cover the iris and pupil as it should in normal cases.

8. Conclusion :

Image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as image objects). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. We have tried to implement iris recognition using image segmentation, since, we felt that segmentation is a simple algorithm in image processing but is versatile as well. We read about iris recognition and realised that it has been tried with segmentation, but we did not find conclusive results. Hence, we decided to pre-process the image and use segmentation and we were able to achieve the desired results.

9.Limitations :

- Lack of an extensive database, which could give us more accurate results.
- In terms of the images, we had to deal with the problem of bad illumination.
- Segmentation Accuracy is highly depended on the Resolution of the images.

Future Scope:

- With the help of MATLAB, these results could have been more accurate, since MATLAB has multiple in-built libraries and functions that would have made pre-processing much easier, and maybe, more layers could have been added to it.
- Further scope for development with low resolution images.
- Can reduce the storage size by better coordinate representation and encoding methods.

10.References :

[1] https://utiris.wordpress.com/2014/03/04/university-of-tehran-iris-image-repository/

[2] https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic3.htm

[3] https://ijcsmc.com/docs/papers/May2014/V3I5201499a84.pdf

[4]https://www.researchgate.net/post/Gabor_filter_and_its_applications

[5]https://docs.opencv.org/3.4/d4/d70/tutorial_hough_circle.html

[6]http://ijarece.org/wp-content/uploads/2017/09/IJARECE-VOL-6-ISSUE-9-1003-1007.pdf