1ABSTRACT

2 A biometric system provides automatic recognition of an individual based on some sort of unique feature or characteristic possessed by the individual. Biometric systems have been developed based on fingerprints, facial features, voice, hand geometry, handwriting, the retina and the one presented in this thesis, the iris.

3 The objective of the project is to implement an open-source iris recognition system in order to verify the claimed performance of the technology. The development tool used is MATLAB, and emphasis is on the software for performing recognition, and not hardware for capturing an eye image.

Iris recognition analyzes the features that exist in the colored tissue surrounding the pupil, which has 250 points used for comparison, including rings, furrows, and freckles. Iris recognition uses a regular video camera system and can be done from further away than a retinal scan. It has the ability to create an accurate enough measurement that can be used for Identification purposes, not just verification.

The probability of finding two people with identical iris patterns is considered to be approximately 1 in 10^{52}(population of the earth is of the order 10^{10}). Not even one-egged twins or a future clone of a person will have the same iris patterns. The iris is considered to be an internal organ because it is so well protected by the eyelid and the cornea from environmental damage. It is stable over time even though the person ages. Iris recognition is the most precise and fastest of the biometric authentication methods.

CHAPTER - 1

BIOMETRIC TECHNOLOGY

1.1 INTRODUCTION:

A biometric system provides automatic recognition of an individual based on some sort of unique feature or characteristic possessed by the individual. Biometric systems have been developed based on fingerprints, facial features, voice, hand geometry, handwriting, the retina and the one presented in this thesis, the iris.

Biometric systems work by first capturing a sample of the feature, such as recording a digital sound signal for voice recognition, or taking a digital colour image for face recognition. The sample is then transformed using some sort of mathematical function into a biometric template. The biometric template will provide a normalised, efficient and highly discriminating representation of the feature, which can then be objectively compared with other templates in order to determine identity. Most biometric systems allow two modes of operation. An enrolment mode for adding templates to a database, and an identification mode, where a template is created for an individual and then a match is searched for in the database of pre-enrolled templates.

A good biometric is characterized by use of a feature that is; highly unique, stable and be easily captured and the chance of any two people having the same characteristic will be minimal. Also the feature does not change over time .In order to provide convenience to the user, and prevent misrepresentation of the feature.

1.2 ADVANTAGES OF USING BIOMETRICS:

- Easier fraud detection
- Better than password/PIN or smart cards
- No need to memorize passwords
- Requires physical presence of the person to be identified
- Unique physical or behavioral characteristic
- Cannot be borrowed, stolen, or forgotten
- Cannot leave it at home

1.3 TYPES OF BIOMETRICS:

* Physical Biometrics

- Finger print Recognition
- Facial Recognition
- Hand Geometry
- IRIS Recognition
- DNA

*Behavioral Biometrics

- Speaker Recognition
- Signature
- Keystroke
- Walking style

1.4 IRIS RECOGNITION SYSTEM:

The purpose of 'Iris Recognition', a biometrical based technology for personal identification and verification, is to recognize a person from his/her iris prints. In fact, iris patterns are characterized by high level of stability and distinctiveness. Each individual has a unique iris.

The probability of finding two people with identical iris patterns is considered to be approximately 1 in 10^{52}(population of the earth is of the order 10^{10}). Not even one-egged twins or a future clone of a person will have the same iris patterns.It is stable over time even though the person ages. Iris recognition is the most precise and fastest of the biometric authentication methods.

1.5 OUTLINE OF THE REPORT:

The objective is to implement an open-source iris recognition system in order to verify the claimed performance of the technology. The development tool used is MATLAB, and emphasis will be only on the software for performing recognition, and not hardware for capturing an eye image.

In the chapter 2, we will discuss about the IRIS Recognition system, its strengths and weakness. In the chapter 3, we will discuss about the implementation of IRIS recognition system. In chapter 4, we will discuss about the unwrapping technology .In chapter 5, about the wavelet analysis. In chapter 6, about the software used for the implementation.

CHAPTER - 2
IRIS RECOGNITION

2.1 INTRODUCTION:

The iris is an externally visible, yet protected organ whose unique epigenetic pattern remains stable throughout adult life. These characteristics make it very attractive for use as a biometric for identifying individuals. The purpose of 'Iris Recognition', a biometrical based technology for personal identification and verification, is to recognize a person from his/her iris prints. In fact, iris patterns are characterized by high level of stability and distinctiveness. Each individual has a unique iris. The difference even exists between identical twins and between the left and right eye of the same person.

The probability of finding two people with identical iris patterns is considered to be approximately 1 in 10^{52}(population of the earth is of the order 10^{10}). Not even one-egged twins or a future clone of a person will have the same iris patterns. The iris is considered to be an internal organ because it is so well protected by the eyelid and the cornea from environmental damage. Iris recognition is the most precise and fastest of the biometric authentication method.

Biometrics	**Universal**	**Unique**	**Permanence**	**Collectable**	**Perfor-mance**	**Accepta-bility**	**Potential to fraud**
Face	High	Low	Medium	High	Low	Low	Low
Fingerprint	Medium	High	High	Medium	High	Medium	Low
Iris	**High**	**High**	**High**	**Medium**	**High**	**Low**	**Low**
Signature	Low	Low	Low	High	Low	High	High
Voice	Medium	Low	Low	Medium	Low	High	High
Vein	Medium	Medium	Medium	Medium	Medium	Medium	Low
DNA	High	High	High	Low	High	Low	Low

TABLE 2.1 COMPARISION OF MAJOR BIOMETRIC TECHNIQUES

Method	Coded pattern	Misidentification rate	Security	Applications
Iris recognition	**Iris Pattern**	**1/1,200,000**	**High**	**High-security facilities**
Fingerprinting	Fingerprints	1/1,000	Medium	Universal
Hand shape	Size, length, and thickness of hands	1/700	Low	Low-security facilities
Facial recognition	Outline, shape and distribution of eyes and nose	1/100	Low	Low-security facilities
Signature	Shape of letters, writing order, pen pressure	1/100	Low	Low-security facilities
Voice printing	Voice characteristics	1/30	Low	Telephone service

TABLE 2.2 IRIS RECOGNITION SYSTEM ACCURACY

The comparison of various biometrics techniques is given in tables 2.1 and 2.2. The comparison shows that IRIS recognition system is the most stable, precise and the fastest biometric authentication method.

2.2 HUMAN IRIS:

The iris is a thin circular diaphragm, which lies between the cornea and the lens of the human eye. A front-on view of the iris is shown in Figure 1.1. The iris is perforated close to its centre by a circular aperture known as the pupil. The function of the iris is to control the amount of light entering through the pupil, and this is done by the sphincter and the dilator muscles, which adjust the size of the pupil. The average diameter of the iris is 12 mm, and the pupil size can vary from 10% to 80% of the iris diameter.

The iris consists of a number of layers; the lowest is the epithelium layer, which contains dense pigmentation cells. The stromal layer lies above the epithelium layer, and contains blood vessels, pigment cells and the two iris muscles. The density of stromal pigmentation determines the color of the iris.

The externally visible surface of the multi-layered iris contains two zones, which often differ in color. An outer ciliary zone and an inner pupillary zone, and these two zones are divided by the collarette – which appears as a zigzag pattern

2.3 WORKING OF IRIS RECOGNITION SYSTEM:

Image processing techniques can be employed to extract the unique iris pattern from a digitized image of the eye, and encode it into a biometric template, which can be stored in a database. This biometric template contains an objective mathematical representation of the unique information stored in the iris, and allows comparisons to be made between templates. When a subject wishes to be identified by iris recognition system, their eye is first photographed, and then a template created for their iris region. This template is then compared with the other templates stored in a database until either a matching template is found and the subject is identified, or no match is found and the subject remains unidentified.

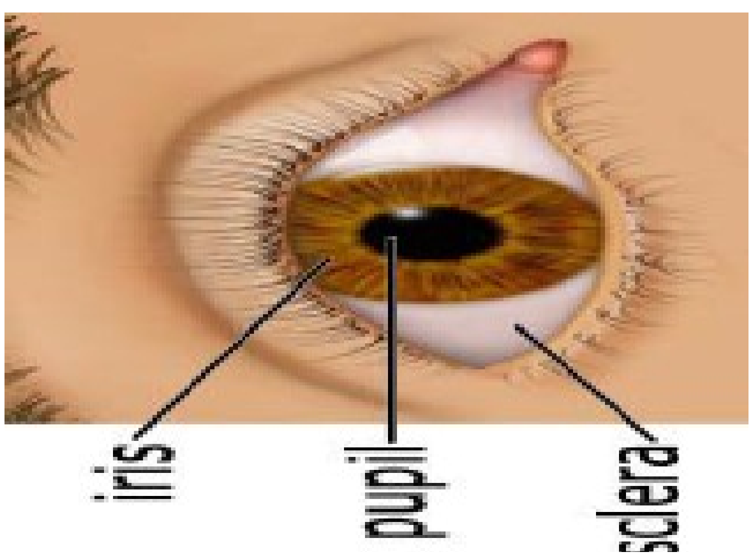


Fig 2.1 HUMAN EYE

2.4 FEATURES OF IRIS RECOGNITION:

- **Measurable Physical Features:** 250 degrees of freedom,250 non-related unique features of a person's iris
- **Unique:** Every iris is absolutely unique. No two iris are the same
- **Stable:** iris remains stable from 1styear till death.
- **Accurate:** Iris recognition is the most accurate of the commonly used biometric technologies.
- **Fast:** Iris recognition takes less than 2 seconds. 20 times more matches per minute than its closest competitor.
- **Non-Invasive:** No bright lights or lasers are used in the imaging and iris authentication process**.**

2.5 IRIS RECOGNITION SYSTEM STRENGTHS:

- **Highly Protected:** Internal organ of the eye. Externally visible; patterns imaged from a distance
- **Measurable Features of iris pattern:** Iris patterns possess a high degree of randomness
  - Variability: 244 degrees-of-freedom
  - Entropy: 3.2 bits per square-millimeter
- **Uniqueness:** Set by combinatorial complexity
- **Stable:** Patterns apparently stable throughout life
- **Quick and accurate**: Encoding and decision-making are tractable
  - Image analysis and encoding time: 1 second

2.6 IRIS RECOGNITION SYSTEM WEAKNESS:

- Some difficulty in usage as individuals doesn't know exactly where they should position themselves.
- And if the person to be identified is not cooperating by holding the head still and looking into the camera.

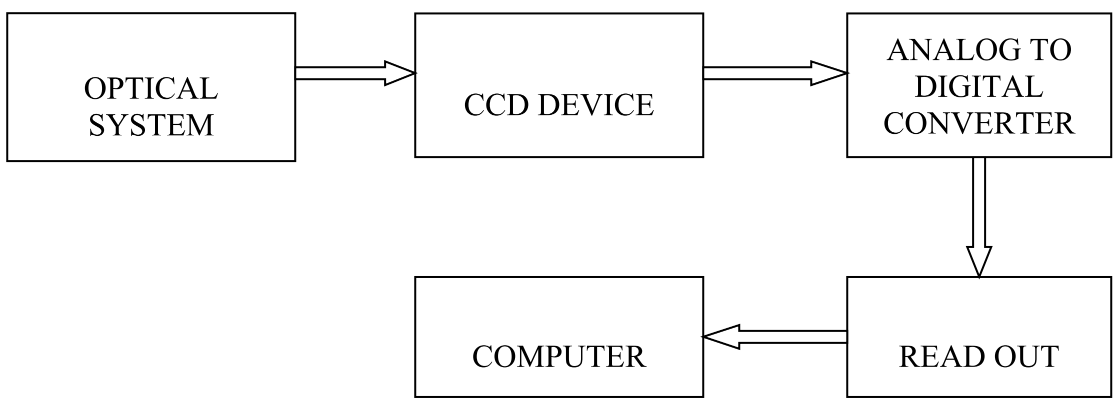
CHAPTER - 3

IRIS SYSTEM IMPLEMENTATION

3.1 IMAGE ACQUISITION:

Image acquisition is considered the most critical step in the project since all subsequent stages depend highly on the image quality. In order to accomplish this, we use a CCD camera. We set the resolution to 640x480, the type of the image to jpeg, and the mode to white and black for greater details. The camera is situated normally between half a meter to one meter from the subject. (3 to 10 inches)

The CCD-cameras job is to take the image from the optical system and convert it into electronic data. Find the iris image by a monochrome CCD (Charged couple Device) camera transfer the value of the different pixels out of the CCD chip. Read out the voltages from the CCD-chip. Thereafter the signals of each data are amplified and sent to an ADC (Analog to Digital Converter).



Figs 3.1 BLOCK DIAGRAM OF IMAGE ACQUISITION USING CCD CAMERA

3.2 IMAGE MANIPULATION:

In the preprocessing stage, we transformed the images from RGB to gray level and from eight-bit to double precision thus facilitating the manipulation of the images in subsequent steps.

3.3 IRIS LOCALIZATION:

Before performing iris pattern matching, the boundaries of the iris should be located. In other words, we are supposed to detect the part of the image that extends from inside the limbus (the border between the sclera and the iris) to the outside of the pupil. We start by determining the outer edge by first down sampling the images by a factor of 4 then use the canny operator with the default threshold value given by Matlab, to obtain the gradient image.

Since the picture was acquired using an infrared camera the pupil is a very distinct black circle. The pupil is in fact so black relative to everything else in the picture simple edge detection should be able to find its outside edge very easily. Furthermore, the thresholding on the edge detection can be set very high as to ignore smaller less contrasting edges while still being able to retrieve the entire perimeter of the pupil. The best edge detection algorithm for outlining the pupil is canny edge detection. This algorithm uses horizontal and vertical gradients in order to deduce edges in the image. After running the canny edge detection on the image a circle is clearly present along the pupil boundary.

Canny Edge Detector finds edges by looking for the local maxima of the gradient of the input image. It calculates the gradient using the derivative of the Gaussian filter. The Canny method uses two thresholds to detect strong and weak edges. It includes the weak edges in the output only if they are connected to strong edges. As a result, the method is more robust to noise, and more likely to detect true weak edges.

3.4 EDGE DETECTION:

Edges often occur at points where there is a large variation in the luminance values in an image, and consequently they often indicate the edges, or occluding boundaries, of the objects in a scene. However, large luminance changes can also correspond to surface markings on objects. Points of tangent discontinuity in the luminance signal can also signal an object boundary in the scene.

So the first problem encountered with modeling this biological process is that of defining, precisely, what an *edge* might be. The usual approach is to simply define edges as step discontinuities in the image signal. The method of localizing these discontinuities often then becomes one of finding local maxima in the derivative of the signal, or zero-crossings in the second derivative of the signal.

In computer vision, edge detection is traditionally implemented by convolving the signal with some form of linear filter, usually a filter that approximates a first or second derivative operator. An odd symmetric filter will approximate a first derivative, and peaks in the convolution output will correspond to edges (luminance discontinuities) in the image.

An even symmetric filter will approximate a second derivative operator. Zero-crossings in the output of convolution with an even symmetric filter will correspond to edges; maxima in the output of this operator will correspond to tangent discontinuities, often referred to as bars or lines.

3.5 CANNY EDGE DETECTOR:

Edges characterize boundaries and are therefore a problem of fundamental importance in image processing. Edges in images are areas with strong intensity contrasts – a jump in intensity from one pixel to the next. Edge detecting an image significantly reduces the amount of data and filters out useless information, while preserving the important structural properties in an image.

The Canny edge detection algorithm is known to many as the optimal edge detector. A list of criteria to improve current methods of edge detection is the first and most obvious is low error rate. It is important that edges occurring in images should not be missed and that there be NO responses to non-edges. The second criterion is that the edge points be well localized. In other words, the distance between the edge pixels as found by the detector and the actual edge is to be at a minimum. A third criterion is to have only one response to a single edge. This was implemented because the first 2 were not substantial enough to completely eliminate the possibility of multiple responses to an edge.

Step1:

In order to implement the canny edge detector algorithm, a series of steps must be followed. The first step is to filter out any noise in the original image before trying to locate and detect any edges. And because the Gaussian filter can be computed using a simple mask, it is used exclusively in the Canny algorithm. Once a suitable mask has been calculated, the Gaussian smoothing can be performed using standard convolution methods. The Gaussian mask used is shown below.

$$\frac{1}{115}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Step2:

After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. The Sobel operator performs a 2-D spatial gradient measurement on an image. Then, the approximate absolute gradient magnitude (edge strength) at each point can be found. The Sobel operator uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows).

They are shown below:

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

Gx

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

Gy

The magnitude, or EDGE STRENGTH, of the gradient is then approximated using the formula:

$$|G| = |Gx| + |Gy|$$

3.6 HOUGH TRANSFORM:

The Hough transform is a technique which can be used to isolate features of a particular shape within an image. Because it requires that the desired features be specified in some parametric form, the classical Hough transform is most commonly used for the detection of regular curves such as lines, circles, ellipses, *etc.* A generalized Hough transform can be employed in applications where a simple analytic description of a feature(s) is not possible. Due to the computational complexity of the generalized Hough algorithm, we restrict the main focus of this discussion to the classical Hough transform.

Despite its domain restrictions, the classical Hough transform (hereafter referred to without the *classical* prefix) retains many applications; as most manufactured parts contain feature boundaries which can be described by regular curves. The main advantage of the Hough transform technique is that it is tolerant of gaps in feature boundary descriptions and is relatively unaffected by image noise.

The Hough transform is computed by taking the gradient of the original image and accumulating each non-zero point from the gradient image into every point that is one radius distance away from it.

That way, edge points that lie along the outline of a circle of the given radius all contribute to the transform at the center of the circle, and so peaks in the transformed image correspond to the centers of circular features of the given size in the original image. Once a peak is detected and a circle 'found' at a particular point, nearby points (within one-half of the original radius) are excluded as possible circle centers to avoid detecting the same circular feature repeatedly.

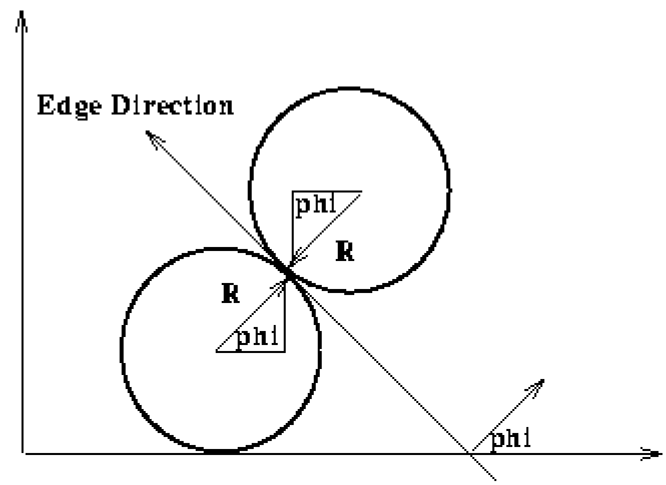
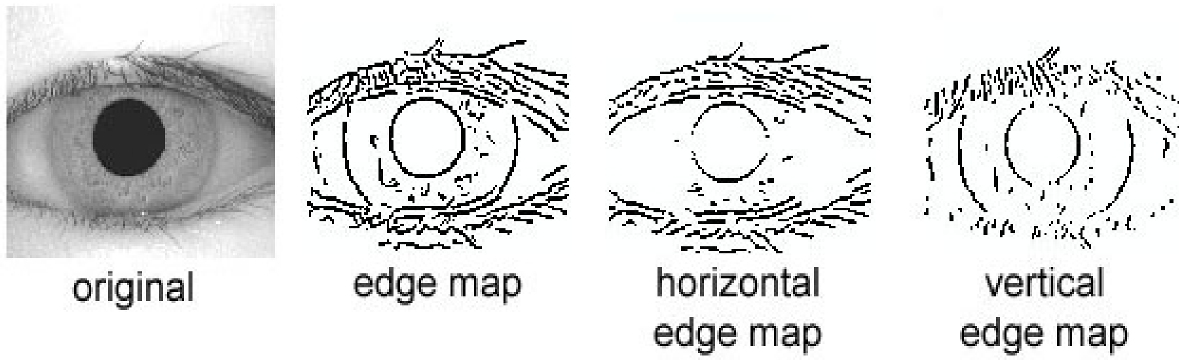


Fig 3.2: Hough circle detection with gradient information.

One way of reducing the computation required to perform the Hough transform is to make use of gradient information which is often available as output from an edge detector. In the case of the Hough circle detector, the edge gradient tells us in which direction a circle must lie from a given edge coordinate point.

The Hough transform can be seen as an efficient implementation of a generalized matched filter strategy. In other words, if we created a template composed of a circle of 1's (at a fixed r) and 0's everywhere else in the image, then we could convolve it with the gradient image to yield an accumulator array-like description of all the circles of radius r in the image. Show formally that the basic Hough transforms (*i.e.* the algorithm with no use of gradient direction information) is equivalent to template matching.



(3.3.1) (3.3.2) (3.3.3) (3.3.4)

Fig 3.3 EDGE MAPS:

3.3.1) an eye image 3.3.2) corresponding edge map 3.3.3) edge map with only horizontal gradients 3.3.4) edge map with only vertical gradients.

3.7 NORMALISATION:

Once the iris region is successfully segmented from an eye image, the next stage is to transform the iris region so that it has fixed dimensions in order to allow comparisons. The dimensional inconsistencies between eye images are mainly due to the stretching of the iris caused by pupil dilation from varying levels of illumination. Other sources of inconsistency include, varying imaging distance, rotation of the camera, head tilt, and rotation of the eye within the eye socket. The normalisation process will produce iris regions, which have the same constant dimensions, so that two photographs of the same iris under different conditions will have characteristic features at the same spatial location. Another point of note is that the pupil region is not always concentric within the iris region, and is usually slightly nasal. This must be taken into account if trying to normalize the 'doughnut' shaped iris region to have constant radius.

CHAPTER- 4

UNWRAPPING

4.1 INTRODUCTION:

Image processing of the iris region is computationally expensive. In addition the area of interest in the image is a 'donut' shape, and grabbing pixels in this region requires repeated rectangular-to-polar conversions. To make things easier, the iris region is first unwrapped into a rectangular region using simple trigonometry. This allows the iris decoding algorithm to address pixels in simple (row, column) format.

4.2 ASYMMETRY OF THE EYE:

Although the pupil and iris circles appear to be perfectly concentric, they rarely are. In fact, the pupil and iris regions each have their own bounding circle radius and center location. This means that the unwrapped region between the pupil and iris bounding does not map perfectly to a rectangle. This is easily taken care of with a little trigonometry. There is also the matter of the pupil, which grows and contracts its area to control the amount of light entering the eye. Between any two images of the same person's eye, the pupil will likely have a different radius. When the pupil radius changes, the iris stretches with it like a rubber sheet. Luckily, this stretching is almost linear, and can be compensated back to a standard dimension before further processing.

4.3 THE UNWRAPPING ALGORITHM:

In fig 4.1, points Cp and C_i are the detected centers of the pupil and iris respectively. We extend a wedge of angle dθ starting at an angle θ, from both points Cp and Ci, with radii Rp and Ri, respectively. The intersection points of these wedges with the pupil and iris circles form a skewed wedge polygon P1P2P3P4. The skewed wedge is subdivided radially into N blocks and the image pixel values in each block are averaged to form a pixel (j,k) in the unwrapped iris image, where j is the current angle number and k is the current radius number.

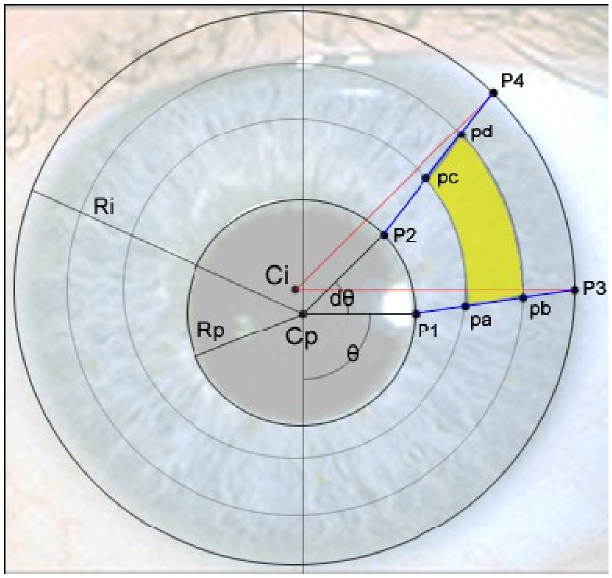


Fig 4.1: Algorithm for unwrapping the iris region.

For this project, the standard dimensions of the extracted iris rectangle are 128 rows and 8 columns (see Figure 4.1). This corresponds to N=128 wedges, each of angle $\frac{2\pi}{128}$, with each wedge divided radially into 8 sections. The equations below define the important points marked in Figure 1. Points P_a through P_d are interpolated along line segments P1-P3 and P2-P4.

$$P_1 = \mathrm{Cp} + \mathrm{Rp}\left(\cos\left(\theta\right) - \sin\left(\theta\right)\right)$$
$$P_2 = C_p + R_p\left(\cos\left(\theta + \mathrm{d}\theta\right) - \sin\left(\theta + \mathrm{d}\theta\right)\right)$$
$$P_3 = \mathrm{Ci} + \mathrm{Ri}\left(\cos\left(\theta\right) - \sin\left(\theta\right)\right)$$
$$P_4 = C_i + R_i\left(\cos\left(\theta + \mathrm{d}\theta\right) - \sin\left(\theta + \mathrm{d}\theta\right)\right)$$

$$P_a = P_1\left(1 - \frac{k}{N}\right) + \frac{P_3 k}{N}$$
$$P_b = P_1\left(1 - \frac{k+1}{N}\right) + \frac{P_3\left(k+1\right)}{N}$$
$$P_c = P_2\left(1 - \frac{k}{N}\right) + \frac{P_4 k}{N}$$
$$P_d = P_2\left(1 - \frac{k+1}{N}\right) + \frac{P_4\left(k+1\right)}{N}$$

4.4 CONTRAST ADJUSTMENT:

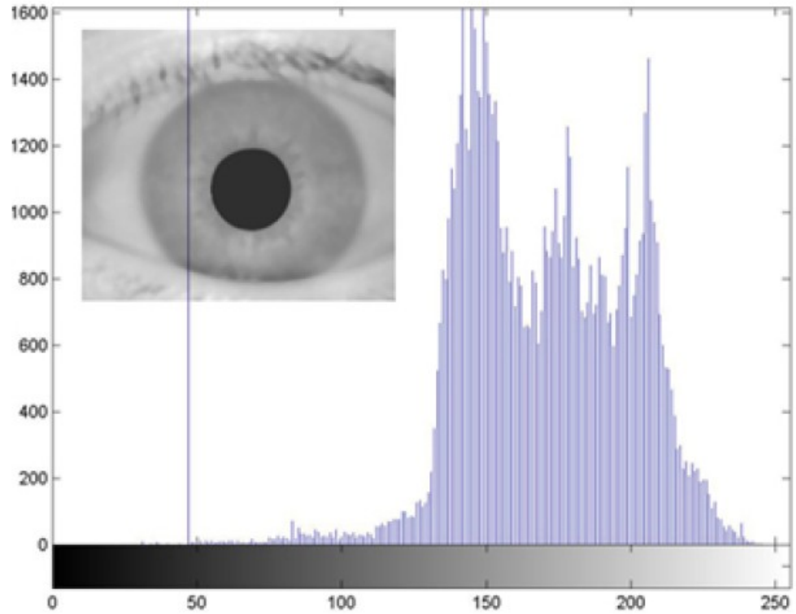


Fig 4.2: Image before Contrast Adjustment

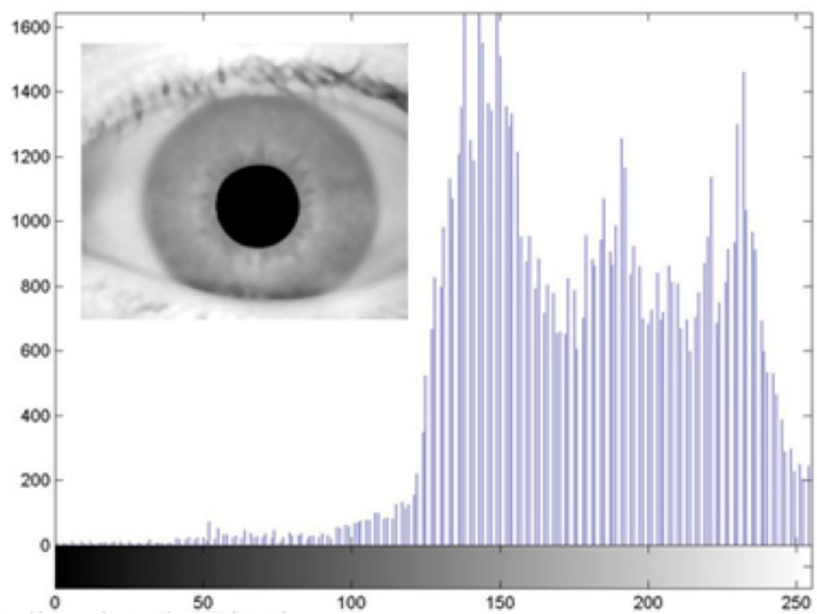


Fig 4.3: Image after Contrast Adjustment

Notice that figures 4.2 and 4.3 appear to be better contrasted than figure 4.1 These images have been equalized in contrast to maximize the range of luminance values in the iris image. This makes it numerically easier to encode the iris data. The equalizing is done by acquiring a luminance histogram of the image and stretching the upper and lower boundaries of the histogram to span the entire range of luminance values 0-255. Figure 4.1 demonstrates an example of this process.

4.5 FEATURE EXTRACTION:

"One of the most interesting aspects of the world is that it can be considered to be made up of patterns. A pattern is essentially an arrangement. It is characterized by the order of the elements of which it is made, rather than by the intrinsic nature of these elements" (Nobert Wiener). This definition summarizes our purpose in this part. In fact, this step is responsible of extracting the patterns of the iris taking into account the correlation between adjacent pixels. For this we use wavelets transform, and more specifically the "Gabor Filtering".

CHAPTER – 5

WAVELET ANALYSIS

5.1 GABOR FILTERING:

To understand the concept of Gabor filtering, we must first start with Gabor wavelets. Gabor wavelets are formed from two components, a complex sinusoidal carrier and a Gaussian envelope.

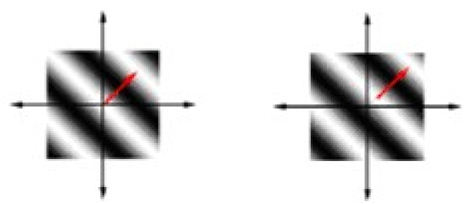
$$g\ (x, y) = s\ (x, y)\ w_r\ (x, y)$$

The complex carrier takes the form:

$$s\ (x, y) = e^{\ j(2\pi(u_0 x+v_0 y)+P)}$$

We can visualize the real and imaginary parts of this function separately as shown in this figure. The real part of the function is given by:

$$Re\ (s\ (x, y)) = \cos\ (2\pi\ (u_0 x + v_0 y) + P)$$



(5.5.1) (5.5.2)

Fig 5.1 IMAGE SOURCE

5.5.1)Real Part 5.5.2)Imaginary Part

and the imaginary:

$$Im\ (s\ (x, y)) = \sin\ (2\_\ (u0x + v0y) + P)$$

The parameters u_0 and v_0 represent the frequency of the horizontal and vertical sinusoids respectively. P represents an arbitrary phase shift. The second component of a Gabor wavelet is its envelope. The resulting wavelet is the product of the sinusoidal carrier and this envelope. The envelope has a Gaussian profile and is described by the following equation:

$$g\ (x, y) = Ke^{\ -\pi(a2(x-x_0)_r 2+b2(y-y_0)_r 2)}$$

Where: $(x - x_0)_r = (x - x_0) \cos(\theta) + (y - y_0) \sin(\theta)$

$(y - y0)_r = -(x - x_0) \sin(\theta) + (y - y_0) \cos(\theta)$

The parameters used above are:

K - a scaling constant

(a, b) - Envelope axis scaling constants,

θ - envelope rotation constant,

(x_0, y_0) - Gaussian envelope peak.

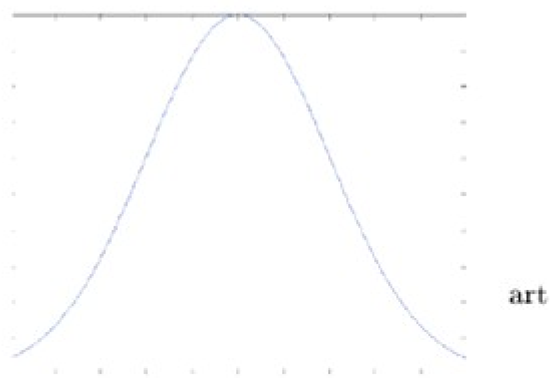


Fig 5.2 GAUSSIAN ENVELOPE

To put it all together, we multiply s (x, y) by w_r (x, y). This produces a wavelet like this one:

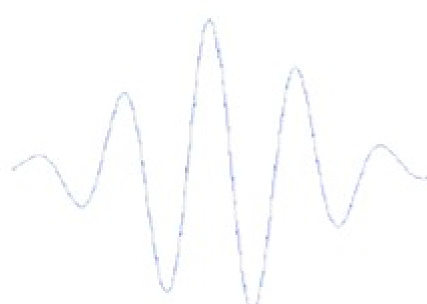


Fig 5.3 GABOR WAVELET

5.2 GENERATING AN IRIS CODE:

After performing unwrapping algorithm it maps the image to Cartesian coordinates so we have something like the following:

Fig 5.4 UNROLLED IRIS

What we want to do is somehow extract a set of unique features from this iris and then store them. That way if we are presented with an unknown iris, we can compare the stored features to the features in the unknown iris to see if they are the same. We'll call this set of features an "Iris Code."

Any given iris has a unique texture that is generated through a random process before birth. Filters based on Gabor wavelets turn out to be very good at detecting patterns in images. We'll use a fixed frequency 1D Gabor filter to look for patterns in our unrolled image.

First, we'll take a one pixel wide column from our unrolled image and convolve it with a 1D Gabor wavelet. Because the Gabor filter is complex, the result will have real and imaginary parts which are treated separately. We only want to store a small number of bits for each iris code, so the real and imaginary parts are each quantized. If a given value in the result vector is greater than zero, a one is stored; otherwise zero is stored. Once all the columns of the image have been filtered and quantized, we can form a new black and white image by putting all of the columns side by side. The real and imaginary parts of this image (a matrix), the iris code, are shown here:

(5.5.1)

(5.5.2)

Fig 5.5 IMAGE OF IRIS CODE

5.5.1) REAL PART 5.5.2) IMAGINARY PART

5.3 TEST OF STATISTICAL INDEPENDANCE:

This test enables the comparison of two iris patterns. This test is based on the idea that the greater the Hamming distance between two feature vectors, the greater the difference between them. Two similar irises will fail this test since the distance between them will be small. In fact, any two different irises are statistically "guaranteed" to pass this test as already proven. The Hamming distance (HD) between two Boolean vectors is defined as follows:

$$HD = \frac{1}{N}\sum_{j=1}^{N} C_A(j) \oplus C_B(j)$$

Where, C_A and C_B are the coefficients of two iris images and N is the size of the feature vector (in our case N = 702). The $\oplus$ is the known Boolean operator that gives a binary 1 if the bits at position j in C_A and C_B are different and 0 if they are similar.

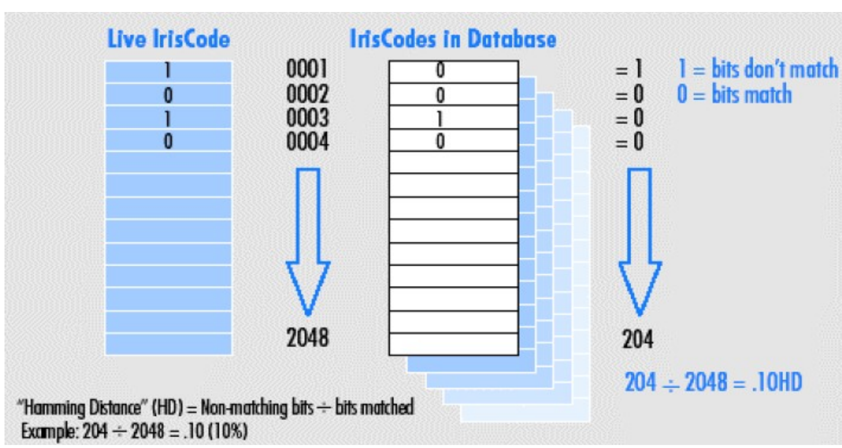


Fig 5.5 HAMMING DISTANCES CALCULATION:

CHAPTER - 6

MATLAB

6.1 INTRODUCTION:

MATLAB® is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building.

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non interactive language such as C or FORTRAN.

The name MATLAB stands for *matrix laboratory*. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

6.2 THE MATLAB SYSTEM:

The MATLAB system consists of five main parts:

a) DEVELOPMENT ENVIRONMENT:

This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, an editor and debugger, and browsers for viewing help, the workspace, files, and the search path.

b) THE MATLAB MATHEMATICAL FUNCTION LIBRARY:

This is a vast collection of computational algorithms ranging from elementary functions, like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix Eigen values, Bessel functions, and fast Fourier transforms.

c) THE MATLAB LANGUAGE:

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create large and complex application programs.

d) GRAPHICS:

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

e) THE MATLAB APPLICATION PROGRAM INTERFACE:

This is a library that allows you to write C and FORTRAN programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

SOURCE CODE

```
function [BD, BWT] =bandtrace (BW)
BD = [];
BWT = BW;
ST = fnd_stP (BWT);
dir = [-1,-1];
Point = ST;
BD = [BD;ST];
NextP = Point + dir;
while (NextP(1)*NextP(2)==0) | (NextP(1)>=size(BWT,1)) |
(NextP(2)>=size(BWT,2))
    dir = dir_next(dir,0);
    NextP = Point + dir;
end
BWT(Point(1),Point(2)) = 0;
totle = sum(sum(BWT));
i = 1;j = 0;
while (i<=totle) & ((NextP(1)~=ST(1)) | (NextP(2)~=ST(2)))
   if BWT(NextP(1),NextP(2)) == 0
      if j >= 8
         NextP = Point;
         break;
      end
      j = j + 1;
      dir = dir_next(dir,0);
      NextP = Point + dir;
   else

      j = 0;
```

```
        BD = [BD;NextP];
        BWT(NextP(1),NextP(2)) = 0;
        Point = NextP;
        dir = dir_next(dir,1);
        NextP = Point + dir;
        i = i + 1;
    end
    while (NextP(1)*NextP(2)==0) | (NextP(1)>=size(BWT,1)) |
(NextP(2)>=size(BWT,2))
        dir = dir_next(dir,0);
        NextP = Point + dir;
    end
end
function [Q]=cir_quad(I,xc,yc,rc,t)
It = I;
for r = 1 : size(I,1)
    for c = 1 : size(I,2)
        Dist(r,c) = sqrt((r-yc)^2+(c-xc)^2);
        if abs(r-yc) > abs(c-xc)
            It(r,c) = 0;
        end
    end
end
Q = sum(It(find((Dist<=rc+t/2)&(Dist>=rc-t/2))));
function varargout = CirclDetect(varargin)
% CIRCLDETECT M-file for CirclDetect.fig
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...

                  'gui_OpeningFcn', @CirclDetect_OpeningFcn, ...
```

```
                 'gui_OutputFcn',  @CirclDetect_OutputFcn, ...
                 'gui_LayoutFcn',  [] , ...
                 'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
% --- Executes just before CirclDetect is made visible.
function CirclDetect_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to CirclDetect (see VARARGIN)
% Choose default command line output for CirclDetect
Image = repmat(logical(uint8(0)),200,200);
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on slider movement.
function RadSet_Callback(hObject, eventdata, handles)
% hObject    handle to RadSet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
```

```
set(handles.textR,'string',get(hObject,'Value'));
% --- Executes during object creation, after setting all properties.
function RadSet_CreateFcn(hObject, eventdata, handles)
% hObject    handle to RadSet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
C = OElocate(handles.Image,[handles.setX;handles.setY]);
set(handles.RadDet,'String',C(3));
% --- Executes on button press in pushbutton1.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(gcf);
function RadDet_Callback(hObject, eventdata, handles)
% hObject    handle to RadDet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of RadDet as text
%        str2double(get(hObject,'String')) returns contents of RadDet as a double
% --- Executes during object creation, after setting all properties.
function RadDet_CreateFcn(hObject, eventdata, handles)
% hObject    handle to RadDet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in pbSet.
function pbSet_Callback(hObject, eventdata, handles)
% hObject    handle to pbSet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.setX = str2double(get(handles.CentX,'String'));
handles.setY = str2double(get(handles.CentY,'String'));
handles.setR = get(handles.RadSet,'Value');
handles.Image = handles.Image*0;
output_coord = plot_circle(handles.setX,handles.setY,handles.setR);
for i=1:size(output_coord,1)
    handles.Image(round(output_coord(i,2)),round(output_coord(i,1))) = 1;
end
axes(handles.axes1);
imshow(handles.Image);
guidata(hObject, handles);
function [Nextdir]=dir_next(dir,step)
```

```
if step==0
    dir_N=[[-1,-1]',[-1,0]',[-1,1]',[0,1]',[1,1]',[1,0]',[1,-1]',[0,-1]'];
    for i = 1:8
        if dir' == dir_N(:,i)
            try Nextdir=dir_N(:,i+1);
            catch Nextdir=dir_N(:,1);
            end
            Nextdir=Nextdir';
        end
end
elseif step==1
    dir_Y=[[-1,-1]',[1,-1]',[1,1]',[-1,1]';[-1,0]',[0,-1]',[1,0]',[0,1]'];
D(i,j)=sqrt((dataset1(i,1)-dataset1(j,1))^2+(dataset1(i,2)-dataset1(j,2))^2);
                end
            end
        end
    case 2
        [m1,n1]=size(dataset1);
        [m2,n2]=size(dataset2);
        D=zeros(m1,m2);
        for i=1:m1
            waitbar(i/m1)
             for j=1:m2
                D(i,j)=sqrt((dataset1(i,1)- dataset2(j,1))^2+(dataset1(i,2)- dataset2(j,2))^2);
             end
        end
    otherwise
        error('only one or two input arguments')
end
close(h)
function D=Euclidian(dataset1,dataset2)
```

```
h = waitbar(0,'Distance Computation');
switch nargin
    case 1
        [m1,n1]=size(dataset1);
        m2=m1;
        D=zeros(m1,m2);
        for i=1:m1
            waitbar(i/m1)
            for j=1:m2
                if i==j
                    D(i,j)=NaN;
                else
                    D(i,j)=sqrt((dataset1(i,1)-dataset1(j,1))^2+(dataset1(i,2)-
dataset1(j,2))^2);
                end
            end
        end
    case 2
        [m1,n1]=size(dataset1);
        [m2,n2]=size(dataset2);
        D=zeros(m1,m2);
        for i=1:m1
            waitbar(i/m1)
              for j=1:m2
                  D(i,j)=sqrt((dataset1(i,1)- dataset2(j,1))^2+(dataset1(i,2)-
dataset2(j,2))^2);
              end
         end
            otherwise
        error('only one or two input arguments')
end
```

```
close(h)
function [ST] = fnd_stP(BW)
sz = size(BW);
r = 0;c = 0;
sign = 0;
for i = 1 : sz(1)
   for j = 1 : sz(2)
      if BW(i,j) == 1
         r = i;c = j;
         sign = 1;
         break;
      end
   end
   if sign == 1,break,end
end
if (r==0) | (c==0)
   error('There is no white point in "BW".');
else
   ST = [r,c];
End
function [C,HM]=Houghcircle(BW,Rp)
%HOUGHCIRCLE - detects circles in a binary image.
   I = BW;
[sy,sx]=size(I);
[y,x]=find(I);
totalpix = length(x);
HM_tmp = zeros(sy*sx,1);
b = 1:sy;
a = zeros(sy,totalpix);
if nargin == 1
   R_min = 1;
```

```
    R_max = max(max(x),max(y));
else
    R_min = Rp(1);
    R_max = Rp(2);
end
y = repmat(y',[sy,1]);
x = repmat(x',[sy,1]);
HPN = 0;
for R = R_min : R_max
    R2 = R^2;
    b1 = repmat(b',[1,totalpix]);
    b2 = b1;
    a1 = (round(x - sqrt(R2 - (y - b1).^2)));
    a2 = (round(x + sqrt(R2 - (y - b2).^2)));
    b1 = b1(imag(a1)==0 & a1>0 & a1<sx);
    a1 = a1(imag(a1)==0 & a1>0 & a1<sx);
    b2 = b2(imag(a2)==0 & a2>0 & a2<sx);
    a2 = a2(imag(a2)==0 & a2>0 & a2<sx);
    ind1 = sub2ind([sy,sx],b1,a1);
    ind2 = sub2ind([sy,sx],b2,a2);
    ind = [ind1; ind2];
    val = ones(length(ind),1);
    data=accumarray(ind,val);
    HM_tmp(1:length(data)) = data;
    HM2_tmp = reshape(HM_tmp,[sy,sx]);
%imshow(HM2,[]);
    maxval = max(max(HM2_tmp));
    if maxval>HPN
        HPN = maxval;
        HM = HM2_tmp;
        Rc = R;
```

```
    end
  end
  [B,A] = find(HM==HPN);
  C = [mean(A),mean(B),Rc];
  function varargout = houghGUI(varargin)
  % HOUGHGUI M-file for houghGUI.fig
%   HOUGHGUI, by itself, creates a new HOUGHGUI or raises the existing
  %      singleton*.
  % Begin initialization code - DO NOT EDIT
  gui_Singleton = 1;
  gui_State = struct('gui_Name',       mfilename, ...
                     'gui_Singleton',  gui_Singleton, ...
                     'gui_OpeningFcn', @houghGUI_OpeningFcn, ...
                     'gui_OutputFcn',  @houghGUI_OutputFcn, ...
                     'gui_LayoutFcn',  [] , ...
                     'gui_Callback',   []);
  if nargin && ischar(varargin{1})
     gui_State.gui_Callback = str2func(varargin{1});
  end
  if nargout
     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
  else
     gui_mainfcn(gui_State, varargin{:});
  end
  % End initialization code - DO NOT EDIT
  % --- Executes just before houghGUI is made visible.
  function houghGUI_OpeningFcn(hObject, eventdata, handles, varargin)
  % This function has no output args, see OutputFcn.
  % hObject    handle to figure
  % eventdata  reserved - to be defined in a future version of MATLAB
  % handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   command line arguments to houghGUI (see VARARGIN)
% Choose default command line output for houghGUI
handles.output = hObject;
S = repmat(logical(uint8(0)),400,400);
output_coord = plot_circle(200,200,75);
for i=1:size(output_coord,1)
   S(round(output_coord(i,2)),round(output_coord(i,1))) = 1;
end
axes(handles.axes1);
imshow(S);
handles.S = S;
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes houghGUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% --- Outputs from this function are returned to the command line.
function varargout = houghGUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
% --- Executes on button press in pbLoad.
function pbLoad_Callback(hObject, eventdata, handles)
% hObject    handle to pbLoad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[filename, pathname] = uigetfile({'*.bmp';'*.jpg';'*.tif'});
S = imread([pathname filename]);
handles.S = S;
```

```
axes(handles.axes1);
imshow(S);
handles.output = hObject;
guidata(hObject, handles);
% --- Executes on button press in pbDetect.
function pbDetect_Callback(hObject, eventdata, handles)
% hObject    handle to pbDetect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
R = get(handles.sldR,'Value');
[C,HM] = Houghcircle(handles.S,[R,R]);
[maxval,maxind] = max(max(HM));
axes(handles.axes2);
imshow(HM);
axes(handles.axes3);
imshow(handles.S);hold on;
output_coord = plot_circle(C(1),C(2),C(3));
plot(output_coord(:,1),output_coord(:,2));
plot(C(1),C(1),'r*');hold off;
set(handles.txtX,'string',C(1));
set(handles.txtY,'string',C(2));
set(handles.txtVal,'string',maxval);
% --- Executes on slider movement.
function sldR_Callback(hObject, eventdata, handles)
% hObject    handle to sldR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.txtR,'string',get(hObject,'Value'));
% --- Executes during object creation, after setting all properties.
```

```
function sldR_CreateFcn(hObject, eventdata, handles)
function txtVal_Callback(hObject, eventdata, handles)
% hObject    handle to txtVal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of txtVal as text
%        str2double(get(hObject,'String')) returns contents of txtVal as a double
% --- Executes during object creation, after setting all properties.
function txtVal_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txtVal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in pbReturn.
function pbReturn_Callback(hObject, eventdata, handles)
% hObject    handle to pbReturn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Close Current figure window.
close(gcf);
function [BW]=im2bw_t(I,t)
if nargin == 1
   t = t_autoset(I);
end
sz=size(I);
BW=repmat(logical(uint8(0)),sz(1),sz(2));
```

```
  for r = 3 : sz(1)-2
   for c = 3 : sz(2)-2
       if I(r,c)<t
           if (I(r-1,c-1)<t)&(I(r-1,c)<t)&(I(r-1,c+1)<t)&(I(r,c-
1)<t)&(I(r,c+1)<t)&(I(r+1,c-1)<t)&(I(r+1,c)<t)&(I(r+1,c+1)<t)
               BW(r,c) = 0;
           else
               BW(r,c) = 1;
           end
       else
           BW(r,c) = 0;
       end
   end
end
for r = 1 : sz(1)
   BW(r,1:2) = 0;
   BW(r,(size(BW,2)-1):size(BW,2)) = 0;
end
for c = 1 : sz(2)
   BW(1:2,c) = 0;
   BW((size(BW,1)-1):size(BW,1),c) = 0;
end
function varargout = IrisLocat(varargin)
% IRISLOCAT M-file for IrisLocat.fig
%
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @IrisLocat_OpeningFcn, ...
                   'gui_OutputFcn',  @IrisLocat_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
```

```
                 'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% --- Executes just before IrisLocat is made visible.
function IrisLocat_OpeningFcn(hObject, eventdata, handles, varargin)
guidata(hObject, handles);
% UIWAIT makes IrisLocat wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% --- Outputs from this function are returned to the command line.
function varargout = IrisLocat_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double
% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double
% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[filename, pathname] = uigetfile({'*.bmp';'*.jpg';'*.tif'});
S = imread([pathname filename]);
if isrgb(S)
   S = rgb2gray(S);
end
%S=resize_u2(S,0.8,0.8);
handles.S = S;
axes(handles.axes1);
imshow(S);
handles.output = hObject;
```

```
guidata(hObject, handles);
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
BW = im2bw_t(handles.S);
BW = N_bandtrace(BW);
tic;
C = Houghcircle(BW);
t1 = toc;
set(handles.text_T1,'string',t1);
handles.C = C;
set(handles.edit1,'string',C(1));
set(handles.edit2,'string',C(2));
set(handles.edit3,'string',C(3));
axes(handles.axes2);
imshow(handles.S); hold on;
plot(C(1),C(2),'xr');
output_coord = plot_circle(C(1),C(2),C(3));
plot(output_coord(:,1),output_coord(:,2));
hold off;
C = handles.C
BW = ~(im2bw(handles.S));
Center = [C(1);C(2)];
for i = -5 : 5
   for j = -5 : 5
      Cent = [C(1)+i;C(2)+j];
      Center = [Center,Cent];
   end
end
```

```
tic;
C1 = OElocate(BW,Center,C(3));
t2 = toc;
set(handles.text_T2,'string',t2);
handles.C1 = C1;
set(handles.edit4,'string',C1(1));
set(handles.edit5,'string',C1(2));
set(handles.edit6,'string',C1(3));
axes(handles.axes3);
imshow(handles.S); hold on;
plot(C1(1),C1(2),'*g');
output_coord = plot_circle(C1(1),C1(2),C1(3));
plot(output_coord(:,1),output_coord(:,2));
hold off;
handles.output = hObject;
guidata(hObject, handles);
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fig = houghGUI;
fig2_handles = guihandles(fig);
handles.figure2 = fig2_handles;
handles.output = hObject;
guidata(hObject, handles);
% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
axes(handles.axes1);imshow([255]);
axes(handles.axes2);imshow([255]);
axes(handles.axes3);imshow([255]);
set(handles.edit1,'string','');
set(handles.edit2,'string','');
set(handles.edit3,'string','');
set(handles.edit4,'string','');
set(handles.edit5,'string','');
set(handles.edit6,'string','');
set(handles.text_T1,'string','');
set(handles.text_T2,'string','');
if isfield(handles,'S')
   handles.S = [];
end
if isfield(handles,'C')
   handles.C = [];
end
if isfield(handles,'C1')
   handles.C1 = [];
end
handles.output = hObject;
guidata(hObject, handles);
% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)
if isfield(handles,'figure2')
   if ishandle(handles.figure2.figure1)
      close(handles.figure2.figure1);
```

```
    end
end
if isfield(handles,'figure3')
    if ishandle(handles.figure3.figure1)
        close(handles.figure3.figure1);
    end
end
close(gcf);
% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fig = CirclDetect;
fig3_handles = guihandles(fig);
handles.figure3 = fig3_handles;
handles.output = hObject;
guidata(hObject, handles);
function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double
% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
recognize
% --- Executes on button press in pushbutton12.
```

```
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
BW = im2bw_t(handles.S);
BW = N_bandtrace(BW);
tic;
C = Houghcircle(BW);
ST = fnd_stP(BW);
prompt = {'Enter file name:'};
dlg_title = 'Input for Radii export';
num_lines = 1;
def = {'David'};
answer = inputdlg(prompt,dlg_title,num_lines,def);
saveradii(answer{1},C,ST);
% --- Executes during object creation, after setting all properties.
function axes3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes3
function [BDim,BD]=N_bandtrace(BW)
BD = [];
BWT_tm = BW;
BDim=repmat(logical(uint8(0)),size(BW,1),size(BW,2));
while sum(sum(BWT_tm)) ~= 0
   [BD_tm,BWT_tm] = bandtrace(BWT_tm);
   if length(BD) < length(BD_tm)
      BD = BD_tm;
   end
```

```
end
for i = 1 : length(BD)
   BDim(BD(i,1),BD(i,2)) = 1;
End
function [C] = OElocate(BW,Center,R)
C = [];
Q = 0;
totle = sum(sum(BW));
delta_r = [1:min(size(BW))];
delta_r = min(size(BW))./delta_r+delta_r;
delta_r = find(delta_r == min(delta_r));
try
   Rmin = R(1);
catch
   Rmin = 1;
end
try
   Rmax = R(2);
catch
   Rmax = size(BW,1);

end
for i = 1 : size(Center,2)
   xc = Center(1,i);
   yc = Center(2,i);
   r = Rmin+delta_r/2;
   while (yc+r-delta_r <= size(BW,1)) & (xc+r-delta_r <= size(BW,2)) %| 1
      Qt = cir_quad(BW,xc,yc,r,delta_r);
      if Q < Qt
         Q = Qt;
         C = [xc,yc,r];
```

```
            if  Q >= 0.5*totle
                break;
            end
        end
        r = r + delta_r;
    end
end
Q = 0;
for r = C(3)-delta_r/2 : C(3)+delta_r/2
    Qt = cir_quad(BW,C(1),C(2),r,1);
    if Q < Qt
        Q = Qt;
        C(3) = r;
        %break;
    end
end
%
% Inputs :
%     X              -   The X coordinates of the center of this circle
%     Y              -   The Y coordinates of the center of this circle

%     Radius         -   The radius of this circle >= 0
%     intervals (optional) - The number of evenly spaced points along the X axis
%                            inclusive of the start and end coordinates
%                           (needed by 'interval' algorithm)
function output_coord = plot_circle(X,Y,Radius,varargin)
outputX = [];
outputY = [];
temp_mod = 1;
cal_mode = 'angle'; %default
intervals = NaN;
```

```
%
=====================================================================
% Check optional inputs
%
=====================================================================
if (length(varargin) > 0)
   for a = 1:1:length(varargin)
      if isstr(varargin{a}) == 1
         if ((strcmp(varargin{a},'angle') == 1)|...
            (strcmp(varargin{a},'interval') == 1)|...
            (strcmp(varargin{a},'bresenham') == 1)|...
            (strcmp(varargin{a},'bresenhamSolid') == 1)...
           )
            cal_mode = varargin{a};
         end
      elseif isfinite(varargin{a}) == 1
         intervals = varargin{a};
      else
         error('Invalid arguments');

      end
   end
end

%
=====================================================================
% Initialization of 'interval'
%
=====================================================================
if strcmp(cal_mode,'interval') == 1
```

```
        % Satisfies requirement for calculation by the given intervals
    if isnan(intervals) == 1
      error('No interval given');
    end
        temp_mod = mod(intervals,2);
        interval = (intervals - temp_mod) / 2;
        interval_size = (Radius+Radius)/intervals;
        valueX = 0;
        if temp_mod == 1
                outputX = [0];
      outputY = [Radius];
    else
      valueX = (-1/2)*interval_size;
        end
    cal_mode = 'interval';
end
===========================================================
% Calculate the 1st quardrant
%
===========================================================
        % *********************************************************
        %       by the angle, scaled by the given radius
        % *********************************************************
if strcmp(cal_mode,'angle') == 1
        increment_angle = pi*(1/Radius);
        curr_angle = pi/2;
        loop=0;
        while curr_angle>0
        loop=loop+1;
      outputX = [outputX(:);sqrt(Radius^2-(Radius*sin(curr_angle))^2)];
      outputY = [outputY(:);sqrt(Radius^2-(Radius*cos(curr_angle))^2)];
```

```
      curr_angle = curr_angle-increment_angle;
        end
        outputX = [outputX(:);Radius];
        outputY = [outputY(:);0];
elseif strcmp(cal_mode,'interval') == 1
        % ***********************************************************
        %       by the given x-interval
        % ***********************************************************
        valueX = valueX + interval_size;
        while valueX < Radius
      outputX = [outputX(:);valueX];
      outputY = [outputY(:);sqrt((Radius*Radius)-(valueX*valueX))];
      valueX = valueX + interval_size;
     end

   outputX = [outputX(:);Radius];
   outputY = [outputY(:);0];
elseif (strcmp(cal_mode,'bresenham') == 1) | (strcmp(cal_mode,'bresenhamSolid') ==
1)
   X = round(X);
   Y = round(Y);
   Radius = round(Radius);
        % ***********************************************************
     x_i = 0;
   y_i = Radius;
   theta_i=2*(1-Radius);
   Limit = 0;
   while y_i >= Limit
      % Set Pixel
      if (strcmp(cal_mode,'bresenham') == 1)
        % Plot the circumference
```

```
    outputX = [outputX(:);x_i];
    outputY = [outputY(:);y_i];
elseif (strcmp(cal_mode,'bresenhamSolid') == 1)
    % Create a solid circle
    tempY = [0:1:y_i]';
    tempX = tempY;
    tempX(:) = x_i;
    outputX = [outputX(:);tempX];
    outputY = [outputY(:);tempY];
else
    error('Invalid option');
end
% determine if case 1 or 2, 4 or 5, or 3
if theta_i < 0

    delta = 2*(theta_i + y_i) - 1;
    % determine whether case 1 or 2
    if delta <= 0
        % move horizontally
        x_i = x_i + 1;
        theta_i = theta_i + (2*x_i) + 1;
    else
        % move diagonally
        x_i = x_i + 1;
        y_i = y_i - 1;
        theta_i = theta_i + (2*(x_i - y_i)) + 2;
    end
elseif theta_i > 0
    end
elseif theta_i == 0
        % move diagonally
```

```
        x_i = x_i + 1;
        y_i = y_i - 1;
        theta_i = theta_i + (2*(x_i - y_i)) + 2;

end
  end
end

==================================================================
% Calculate the 2nd quardrant
%
==================================================================
length_outputX = length(outputX);
if temp_mod == 1
      % Avoids duplicate coordinates
      outputX = [outputX([length_outputX:-1:2])*(-1);outputX(:)];
      outputY = [outputY([length_outputX:-1:2]);outputY(:)];
else
  outputX = [outputX([length_outputX:-1:1])*(-1);outputX(:)];
      outputY = [outputY([length_outputX:-1:1]);outputY(:)];
end
==================================================================
% Shift the circle to the desired center
%
==================================================================
outputX = outputX+X;
outputY = outputY+Y;
output_coord = [outputX,outputY];
% If a solid is asked for make sure there are no duplicates
if (strcmp(cal_mode,'bresenhamSolid') == 1)
  output_coord = unique(output_coord,'rows');
```

```
end
function recognize
dat1=input('enter the path of text file 1');
dat2=input('enter the path of text file 2');
a = textread(dat1 , '%s', 'whitespace', ' ' );
b = textread(dat2 , '%s', 'whitespace', ' ' );
%d=Euclidian(a,b);
if (cell2mat(a(9,1))~=cell2mat(b(9,1)))
      disp('IRIS does not matches with the current person');
else if (cell2mat(a(10,1))~=cell2mat(b(10,1)))
          disp('IRIS does not  matches with the current person') ;
   else
       disp('IRIS DOES  MATCHES WITH THE CURRENT PERSON');
   end
end
function saveradii(name,C,ST)
name=strrep(name,' ','_');
date=datestr(now,29);
FileName=[name '_' date '.txt'];
file=fopen(FileName,'wt');
fprintf(file,'%2.0f \n',size(C,1));
 fprintf(file,'%s','Radius: ');
 fprintf(file,'%2.0f \n',[ST]);
fprintf(file,'%s \n','------------------------------------------------------------------');
fprintf(file,'%s \n','------------------------------------------------------------------');
fclose(file);
function [T]=t_autoset(I)
%I=imread('eye.jpg')
graycount=imhist(I);
sz=size(I);
T=0;
```

```
graycount=filter([0.5 0 0.5],1,graycount);
%figure,imhist(I);
diff_cnt=diff(graycount);
pre_max = graycount(1);
for i = 2 : (length(diff_cnt)-1)
   if (graycount(i)>graycount(i-1))&(graycount(i+1)<graycount(i))&(diff_cnt(i-
1)>0)&(diff_cnt(i+1)<0)
        %¼«´óÖµµã
        pre_max=max(graycount(i),pre_max);
        continue;
   end
if (graycount(i)<graycount(i-1))&(graycount(i+1)>graycount(i))&(diff_cnt(i-
1)<0)&(diff_cnt(i+1)>0)&(graycount(i)<0.75*pre_max)&(pre_max/(sz(1)*sz(2))>0.0
025)
  T=i;
        break;
   end
end
```

CONCLUSION

We have successfully developed a new Iris Recognition system capable of comparing two digital eye-images. This identification system is quite simple requiring few components and is effective enough to be integrated within security systems that require an identity check. The errors that occurred can be easily overcome by the use of stable equipment. Judging by the clear distinctiveness of the iris patterns we can expect iris recognition systems to become the leading technology in identity verification.

BIBLOGRAPHY

- "The Biometrics technology" Third Edition.

  by Guodong Guo.

- "IRIS Recognition" Second Edition.

  by Roy Kaushik and Prabir Bhattacharya.

- "The IRIS Recognition System"

  by kshvik Douglas.

- "Biometric systems"

  by Wayle Wolf

- www.alibris.com
- www.adaptiveoptics.org
- www.icdri.org
- www.sans.org
- www.google.co.in
- www.wikipedia.com