Chapter 1

Introduction
Biometrics is the science of automated recognition of persons based on one physiological or behavioral characteristic. It would be mainly for the purposes of either identification or verification. [1]

Biometric feature extraction for iris scans will be referred to as iris recognition in this text. Iris recognition entails use of the distinct iris patterns to identify an individual from another as no two sets of iris can be the same. Of note is that even left and right set of iris from the same individual would differ. It is thus a reliable biometric method that would be appropriate in wide range of application.

The best of Biometric system are those capable of using distinct unique characteristics in their identification to distinguish the samples used. It ensures no chance is given two different individual to possess the same unique characteristics. The system developed here extracts salient features of an iris and compares it with those from another to establish if they came from same individual. The mathematical abstraction that does this is well covered in length under the software implementation section.

The original proposal of using retina as the biometric tool is replaced by iris due to unavailability of database with scanned retinas. The Chinese Academy of Sciences Institute of Automation (CASIA) which holds huge databases that they provide strictly for research had only irises images; which they gladly offered for my research. [3]

Problem definition
The problem requires one to first familiarize self with Biometrics identification techniques with bias in use of iris scans. Those salient features that contribute to iris uniqueness and distinguish an individual from another would need to be identified.

Software is then to be developed and used for extracting the features in question for mainly identification purposes.

Problem justification
Issues to do with security will forever be prioritized in organizations and companies. Forgery and imposters never cease and most have gone notch higher in breaching and defeating most installed systems.

This project would be quite instrumental in helping stop this problem where other measures have failed. Iris comparisons to provide verification would be reliable due to efficiency and uniqueness as their patterns differ among individuals.

Other Biometrics technique like Fingerprint recognition would be helpful in authentication though as compared to iris, they could be easily forged.

The industry of application mainly desires quick and accurate results in verification of subject’s identity. Iris scan procedure offers this option as it takes only 10-15 seconds to extract and do comparison.

With advance in technology nothing can be left to chance especially in fighting fraud. Iris scans if used properly would serve as a good alternative to most systems mainly those...
biometric in nature. Organizations only incur huge costs at installation and setting up of the equipment but would thereafter save a lot in terms of security investment. The project area has room for intensive research and exploration as most in Biometrics field have mainly focused on fingerprint recognition method for identification at expense of iris scans.

**Project objectives**

Main:
- To develop understanding of use of iris for biometric identification systems and develop software for extracting salient features of an iris scan for purpose of identification.

Specific:
- To identify salient features of the iris and how to use them to distinguish a variety of scans.
- To develop software for extraction and display of the salient features of given iris scans.
- To compare sets of irises and check for matching.

**Scope of the project**

The project generally seeks to demonstrate understanding and use of iris scans for biometric identification. Salient features of iris are identified, their extracts analyzed critically by use of a software. The software for extraction of the features is developed on MATLAB platform. Already prepared iris scans are used due to lack of scanners and/or very high resolution cameras that would have otherwise assisted to obtain the required distinct features for comparison. The program developed would compare two iris images a time and determine if they came from same individual; what we are referring to as matching.

A computer vision algorithm based on The Hough Transform is made use of in determining circles present in the image which are essentially the iris and pupil. The region of interest from which the unique features are obtained would be located between papillary and limbic boundaries. Papillary (Between iris and pupil) while limbic boundary is the one demarcating the iris and sclera.

The comparison entails first determining codes of the two iris images through generation of separate biometric template for them then using encoding procedure. Hamming distance calculation between the iris codes determines if they match.
Hardware and software support
The software would be developed on a MATLAB platform. Proposed version is Matlab 2008a which has additional functions. A basic window O/S is essential and if running on Linux is desired then an executable file of the software can be easily developed and used. Computer that comfortably runs the above software would suffice.

Uses
- The main application would be in authentication and identification purposes.
- Prisons and ATM identity verification.
- Passport could employ such to assist immigration offices in people’s identification.
- Medical - Communicable illness such as Aids, Leukemia that impact eyes. Iris patterns would help doctors diagnose these diseases with ease.
Chapter 2

Literature review

The Human Iris
Iris is located in the human eye at region between the cornea and the lens [5]. It controls the amount of light entering the eye through the pupil. This it does through the use of specialized muscles unique to it called sphincter and dilator. It is an internal organ visible from outside and of importance to note is it remains unchanged from birth to death once formed making it a suitable biometric tool.
Unique pattern on the surface of iris is formed during the first year of life. The unique patterns are formed randomly with no tie to genetic factors. This epigenetic nature explains why even left and right irises of the same person never resemble. Identical twins are also no an exception. Their iris pattern would differ too.

Iris recognition technology
Iris recognition technique is appropriate as even identical twins cannot have same iris patterns [1]. The left and right eye of an individual also has distinct iris texture patterns. It is the only internal organ visible from outside and also does not change much after youth.

Pros & Cons of Iris recognition

Pros
✓ Low occurrence of false positives.
✓ Extremely low (almost 0%) false negative tests.
✓ Highly reliable even twins differ in iris patterns.
✓ Speedy results: identity of the subject is verified very quickly.
✓ Iris remains unchanged from birth till death thus suitable in wide applications.

Cons
• Measurement accuracy can be affected by diseases like cataracts, and astigmatism.
• Scanning procedure perceived by some as invasive.
• High equipment cost.

Examples of other biometric techniques.
Related work in the Biometric field would be use of facial recognition, fingerprint and retinal scans.

Retinal scans
Retinal scans use infrared light to survey the unique pattern of blood vessels of the retina, which is the nerve tissue that lines the back of the eye. There is little chance that retinal patterns can be replicated or forged. They are thus considered to be the least violable of biometric security measures (Fingerprints by comparison are relatively easy to forge).
**Fingerprint recognition**
This involves use of unique pattern of fingerprints that differ among people in attempt to distinguish them in identification or verification process.

**Facial Recognition**
This Biometric technique would involve capturing the entire face and obtaining the salient features that would help identify one image from another.

**The Iris Recognition System**

A Complete Iris recognition system can be split into 4 stages: [1]
- Data acquisition
- Segmentation
- Encoding
- Matching

![Fig 2.1 Block diagram of iris recognition procedure.](image)

*Data acquisition*
This involves capturing iris images. Infra red illumination is used in most iris image acquisition. Cameras such as one developed by ‘OKI’ Japan would help acquire clear images. Imaging done under natural light would on the contrary lead to specula reflections. [3]
Segmentation

The techniques to be used here are explained.

- **Hough Transform** [6]
  This is a standard computer vision algorithm used to determine parameters of simple geometric objects such as lines and circles in an image. One can employ the circular Hough transform to determine centre and radius of iris and pupil regions. You first generate an edge map by calculating the first derivative of intensity value in an eye image and then thresholding the result. From the edge map, votes are cast in Hough space for the parameters of circle passing through each edge point. These parameters are the centre coordinates and radius of the circle. They would define any circle using the usual equation

\[
X_e^2 + Y_e^2 - r_e^2 = 0
\]

Equation 2.1

A maximum point in the Hough space will correspond to the centre and radius coordinates of the circle best defined by the edge points.

- **Canny edge detection**
  The method is used for generation of edge maps. Gradients are biased in the vertical direction for the outer iris/sclera boundary. Vertical and horizontal boundaries are weighted equally for the inner iris/pupil boundary.

Iris segmentation stage localizes the iris region in the image [2]. It involves isolation of the actual iris region in the digital image. For most algorithms, the iris boundaries are modeled as two circles which are not necessarily concentric. The inner circle is the papillary boundary (between pupil and iris). The outer is the limbic boundary (between the iris and the sclera). Most segmentation algorithms are gradient based; that is they involve finding the edges between iris and pupil, iris and sclera.

We model the papillary boundary as an ellipse and the limbic as a circle. A canny style edge detector is used to generate the edge map. A circular Hough transform is performed and the maximum value in Hough space corresponds to the centre and radius of the circle. Algorithm usually first detects the limbic boundary from the entire iris then the papillary is obtained from within the limbic.

Noise processing is included in this stage. The main sources of noise being eyelid occlusion, eyelashes occlusion, specula highlight and shadows.

For noise processing, the eyelids are modeled as lines. When detecting the top and bottom eyelids we use canny edge detector to generate the edge map. The line is then located using a linear Radon Transform. Pixels with value greater than 240 constitute the highlight noise. After detection we eliminate them by marking off with NaN (Not a number) values.
Encoding
This entails encoding the iris image texture into a bit vector code. In most algorithms, filters are utilized to obtain information about the iris texture. Then the outputs of the filters are encoded into a bit vector code. Gabor and log-Gabor filters are examples of such that could be used for the process. Log-Gabor filter for instance is introduced to the features extracted and normalized. It will provide phase data that will be quantized and used to generate the bit-wise biometric template. Only the most discriminating features of the iris are used.

Matching
Matching stage calculates the distance between two sets of iris codes and decides whether it is a match (in the verification context) or recognizes the submitted probe iris from the subject in the gallery set (Identification context). Calculation of a ratio called hamming distance is essential as it’s appropriate for comparison of two irises and determines if they are from same individual (a match).

While there are a number of algorithms used for iris recognition (example: Boles, Boas hash, Wildes, Daugman), the common ones are Daugman and Wildes schemes of implementation.

Daugman Technique.
Daugman algorithm is the best known algorithm for iris recognition. The iris is modeled as two circles which are not necessarily concentric [2]. Each circle is defined by three parameters \((x_0, y_0, r)\), where \((x_0, y_0)\) locates the centre of a circle with radius \(r\). It utilizes an integrodifferential operator to estimate the three parameter values for each circular boundary. It searches the whole image with respect to an increasing radius \(r\) to maximize

\[
\left| G(r) \ast \frac{\partial}{\partial r} \frac{1}{\sqrt{2\pi G(r^2)}} I_{\text{arc}} ds \right| \quad \text{Equation 2.2}
\]

Where \(I(x, y)\) is the intensity value in the image at location \((x, y)\), \(ds\) means the circular arc, \(\frac{2\pi r}{2\pi r}\) is used to normalize the integral, \(G(r)\) is a Gaussian filter used as a smoothing function and \(*\) represents the convolution operation.

The eyelids are modeled as parabolic arcs. An integrodifferential operator as described in equation 1.1 is also used to locate the upper and lower eyelids. In that case the integral is computed over a parabolic arc instead of a circular arc. The regions detected for the eyelids are excluded from the iris image.

The segmented iris image is normalized and converted from Cartesian image coordinates to polar image coordinates. Then the 2D Gabor filter is used to encode the iris image to a binary code of 256 bytes in length. In the matching part, the hamming distance is used to indicate the similarity of two iris codes. A smaller distance means a better match. A threshold is used to determine if two iris codes match well enough to be considered to come from the same person, or not.
Daugman [J. Daugman. Biometric personal identification system based on iris analysis. U.S. Patent No. 5,291,560, 1994.] reported obtaining a false accept rate (FAR) of 1 in 4 million along with a false reject rate (FRR) of zero.

Note:
If an accepted probe sample and the gallery sample are from the same subject, it is called a true accept; otherwise, it is called a false accept. The percentage of false accepts is called the false accept rate (FAR). If a rejected probe sample and the gallery sample are not from the same subject is called a true reject. Otherwise it is called a false reject. The percentage of false rejects is called the false reject rate (FRR).

Wildes Technique
Another well-known and thoroughly tested algorithm is Wildes. The Wildes algorithm locates the iris boundaries by creating a binary edge map using gradient-based edge detection, and finds the centers and radii of these circles via a Hough Transform [6]. The upper and lower eyelids are located similarly using parabolic arcs. Rather than map every iris image to a common system of polar coordinates, the Wildes algorithm compares two images by geometrically warping one image, via shifting and rotating, until it is a best fit with the other image, in the sense of minimizing mean square distance. A Laplacian Pyramid is constructed at four different resolution levels to encode the image data. Matching is achieved via an application of normalized correlation and Fisher’s linear discriminant.

Daugman technique will be applied in the normalization stage as is explained in the next chapter (Daugman rubber sheet model) while Hough Transform employed by Wildes would be used in determining iris and pupil in the images at the segmentation level.

Daugman rubber sheet model is very accurate in conversion of Cartesian into polar thus preferred. Hough Transform being selected over integro-differential proposed by Daugman is due to the fact that the latter uses algorithm that work only on a local scale. This means it can fail where noise such as from reflections is evident.
Chapter 3
The Software

This chapter describes the process of system designing. It outlines the many different modules developed to perform various functions. They are then integrated based on what each of the blocks is supposed to achieve. All of them would point to the main module that controls the entire program.

System functionality requirement

The Project entails development of software that would help distinguish variety of scanned iris images. Images will be in a central database from which the program is expected to access on user’s request for iris identification. The system is expected to achieve a number of things and take into account the need for end user to easily understand its operation. Some of main requirements are as below.

- The program should generate a biometric template of iris image fed into it.
- It is important that wide range of scanned iris images formats be accepted (Jpeg, gif).
- The software needs to be interactive enough especially when pointing out to user errors; example: due to image format not being supported by system.
- System should prompt user for only one image at time and reject multiple request at the same time for easier comparison and matching.
- Program should eliminate errors due to occlusions of eyelid and eyelashes in extracting the desired salient features for identification.
- It is desired that the end user comfortably uses the system without need to fully understand logic behind.

The overall design would be modeled based on the concept; user interacting through a GUI with the biometric system that links to a database containing the iris images. This means the architecture of the project’s program can be divided into 3 main layers:

- User interface (GUI). This is responsible for display of dialogues, prompting user to perform number of things and relays information to models based on data from user. It responds to user request through linking with logic layer.
- Model layer will basically keep track of requests from user and respond accordingly based on logic of the program.
- Program Logic layer contains the system implementing functions.
The layers and their relationships are depicted in figure 3.1

![Diagram of system layers](image)

**Fig. 3.1 System layers**

The software is developed on MATLAB platform. This high level language is selected as its image processing toolbar is appropriate and powerful enough to handle the scanned iris images effectively. Chinese Academy of sciences- Institute of Automation (CASIA) huge Data set provides the scanned images for comparison.

**Program logic**

An iris image is accepted by the system then compared with those in the database to establish existence of a match. The iris images will undergo automatic segmentation to identify the iris and pupil positions within the eye. After identifying the two, we isolate through addition of concentric circles to demarcate the boundary. The result will be isolation of the region of interest. This is what we refer to as localization. We then normalize this region prior to encoding. Two iris codes are as a consequence generated. Calculation of Hamming Distance between two iris codes determines whether the iris image captured is a better match with any of the ones in the database. We compare two irises a time.

The flowchart in figure 3.2 indicates the logic operation of system developed.
Input iris image

Iris Localization

Iris normalization

Eyelid Masking

Effective region extraction

Calculation of Hamming Distance (HD)

Is HD close to threshold value?

Database (CASIA)

Iris image synthesis

yes

It’s a Match

no

No Match

Fig.3.2 Flowchart of iris recognition algorithm.
The Procedure

Input iris
Simply accepts an iris image already scanned for comparison. For lack of data acquisition machine, images were obtained from the Chinese Academy of Sciences (www.sinobiometrics.com) huge data set. The site was able to provide a number of already clearly scanned iris images taken at close proximity. This ensures absence of reflections thus eliminates computational errors.

Iris localization
Localization process is performed by a number of algorithms in different modules. We have one which is a circle generator that adds weights into a hough accumulator array. Hough transform is used for finding circles in the image. The edge of iris image undergoes canny edge detection method. A module is then developed for returning the coordinates of a circle in an image by use of the Hough transform and canny edge detection to create the edge image.
Line and circle coordinates are determined to be used for returning pixel coordinates of a circle defined by a given radius and x, y coordinates. Image gamma adjustment is performed to enhance contrast of bright region (for image gamma value range 0-1) and also enhance contrast in dark region (for values>1).
Hysteresis thresholding of an image is performed. Here, all pixels with values above a given preset threshold are marked as edges. Module for actual iris segmentation and also used for noise elimination such as ones arising from occluding eyelids and eyelashes is also included.

Iris Normalization and Encoding
Normalization of the iris image is performed by unwrapping the circular region into a rectangular block of constant dimensions. Effect of scale difference in iris images could be evident and to prevent this we normalize prior to encoding. Normalization helps compensate for iris deformation. We unwrap the image to a rectangular block of a fixed size. The eyelid region is then masked by darkening it.

The iris image is transferred from Cartesian co-ordinates to polar image coordinates using Daugman rubber sheet model as shown in the equation 3.1 and fig.3.3 [2]
\[ I(r, \theta) = I(x(r, \theta), y(r, \theta)) \]
\[ x(r, \theta) = (R_{pupil} + \frac{r \times |AB|}{20}) \times \cos \theta \]
\[ y(r, \theta) = (R_{pupil} + \frac{r \times |AB|}{20}) \times \sin \theta \]
\[ |AB| = \sqrt{d \times K \pm \sqrt{d \times K^2 - d + R_{\text{limb}}^2 - R_{\text{pupil}}^2}} \]
\[ d = (x_o - x_o')^2 + (y_o - y_o')^2 \]
\[ K = \cos(\pi - \arctan(\frac{y_o - y_o'}{x_o - x_o'}) - \theta) \]

Equation 3.2. Conversion of Cartesian to polar coordinates.

In the equation above, \( I \) is the intensity value of a pixel in the image. As shown in the figure 3.4, the reference of the Cartesian image coordinates is at the papillary boundary centre \( O \). \( O' \) is the centre of the limbic boundary. \( A \) and \( B \) are points on the papillary and limbic boundary respectively at an angle \( \theta \). The conversion is important since the papillary and limbic boundaries were modeled as ellipse and circle respectively thus polar coordinates would be appropriate to define them.

**Cartesian Coordinates**

**Polar Coordinates**

Fig.3.4 Conversion of Cartesian to polar image coordinates.
Each row of an image is convolved by use of 1D Log-Gabor filters. A biometric template is then to be generated from the normalized iris image. We illustrate how to encode a given iris image row. The encoding will generate bit vector codes for the iris that would be used for comparisons in the matching stage.

The row of the image is \( X = [P(1), P(2), \ldots, P(N)] \). We first do a discrete Fourier transform (DFT) on vector \( X \) to get vector \( Y \). Then we multiply vector \( Y \) by 1-D log-Gabor wavelet to get vector \( Z \). The 1-D log-Gabor function is defined by the function \( G(f) \), where \( f_0 \) denotes the central frequency and \( \sigma \) denotes the bandwidth. Finally we perform an inverse DFT on the vector \( Z \) to get vector \( D \). The equations are illustrated.

\[
Y = \sum_{n=1}^{N} x_n e^{- \frac{2\pi i (n-1) \log (f/f_0)}{N}} \quad \text{Equation 3.3}
\]

\[
Z = Y \times G \quad \text{Equation 3.4}
\]

\[
G(f) = e^{- \frac{\log (f/f_0)^2}{2\log (f_0/t)^2}} \quad \text{Equation 3.5}
\]

\[
D = \frac{1}{N} \sum_{n=1}^{N} Z_n e^{- j2\pi (n-1)(n-1)/N} \quad \text{Equation 3.6}
\]

**Encoding Equations illustrated**

The output of the Gabor filter has both real and imaginary part. After decomposition by 1D log Gabor filter, the phase angle of each output is quantized to 2 bits.

**Hamming Distance (Matching).**
This determines if the iris images are a match. Small hamming distance between the codes is an indication of a better match. Hamming Distance (HD) is defined as the number of different bits in the two codes divided by the total number of valid bits (equation 2.6).
\[ HD(A, B) = \frac{1}{\sum_{i,j} \text{valid}(i, j)} \sum_{i,j} ((A(i, j) \oplus B(i, j)) \text{and valid}(i, j)) \]  

Equation 3.7

Where valid \((i, j)\) is defined by

\[ \text{valid}(i, j) = 1 \text{ if } \text{mask}(i, j) = 0 \text{ otherwise } \text{valid}(i, j) = 0 \]

Another possible source of error would be rotations of iris which we overcome by use of circular shifting in the HD calculation. For instance when calculating HD between A and B, we fix code A and shift the code B from -15 degrees to +15 degrees with increment of 1.5 degrees. The minimum HD from these 20 shift positions is the desired one and used as the reported HD.
Samples used
The software was used to compare a total of 30 iris images obtained as mentioned earlier from The CASIA. One image at a time was accepted and automatic segmentation for it performed. After the extraction of the salient features, two iris images at a time were compared to establish if the samples came from same individual. An output in form of message box popped to indicate whether or not they match. 28 out of the 30 images processed gave perfect results as illustrated by one of the iris image in figures 4.1 and 4.2 at the segmentation level.

Fig.4.1 Original image  
Fig.4.2 segmented image

Notice the near perfect concentric circles in the segmented iris. This is an Indication that the process was a success for the particular image.

The two images that failed the test resulted in displaced circles so that during extraction of features from segregated region, some distinct patterns with information are not included. We illustrate this in figures 4.3 and 4.4.

Fig4.3 Original image  
Fig4.4 Segmented image.
The automatic segmentation was not as perfect for this image as the previous one. It is evident from the location of circles for iris and pupil identification. This is due to little contrast in papillary and limbic boundaries that causes canny edge detection to fail.

**Noise Elimination**
Noise that results from eyelids and eyelashes was eliminated by marking them with NaN (Not-a-Number) values which is seen here on the images as blackened regions.

![Fig4.5 Image-with noise elimination](image)

After segmentation which simply localized the desired iris, normalization was applied prior to encoding. This ensured any images inconsistencies are taken into account as the extracted iris region was converted into a rectangular of constant dimensions for all iris images.

**Segmented iris**
Figures 4.6 and 4.7 are samples from an output of segmented iris that undergoes normalization after possible noise sources have been eliminated.

![Fig 4.6 Segmented iris showing area for normalization](image)
Normalized template

Fig.4.7 Normalized template

The normalized template is standard for all images thus unique features for all irises are able to be extracted without inconsistencies so that encoding and matching would be successful.

The final stage was the encoding of the two iris images provided after which their codes are compared with through calculation of hamming distance. This determines whether the samples provided are a match.

The matching was also perfect for the 28 out of the 30 iris images used. Images taken from the same subject returned matching results just as anticipated while those from different sources did not match. The failed images would be due to canny edge detection method unable to spot the edges clearly due to little contrast in the boundaries to be detected.

The figures 4.8-5.2 display the output which the system gave when performing the extraction procedure.

Sampled output images

Fig 4.8 Image (noise eliminated)  Fig 4.9 image (shows extracted area)
The figures show sampled output that the Biometric system produced for the automatic segmentation stage.

**Conclusion and Future Work**

The Biometric system developed has been able to successfully extract the unique patterns of iris template generated. It is these salient features that were essential to be used for distinguishing an iris image from another. Already scanned iris images were input and automatic segmentation performed on them. This process involved localization of the iris and pupil regions. The Hough transform was the computer vision algorithm used to identify the circles in the image and edges were detected using the canny edge detection technique. Once the region was clearly demarcated, it was normalized into a rectangular block of constant dimension. Gabor filters were then used to perform the encoding process after which pair of iris at a time was compared against.

The author proposes that real time processing be performed for the system instead of using already scanned images. Cameras such as ‘OKI’ developed by the OKI- Japan could be acquired and used for capturing the iris images.
REFERENCE


Appendix

The Matlab Codes

CreateIrisTemplate

% createiristemplate - generates a biometric template from an iris in
% an eye image.
%
% Usage:
% [template, mask] = createiristemplate(eyeimage_filename)
%
% Arguments:
% eyeimage_filename   - the file name of the eye image
%
% Output:
% template      - the binary iris biometric template
% mask       - the binary iris noise mask

function [template, mask] = createiristemplate(eyeimage_filename)

% path for writing diagnostic images
global DIAGPATH
DIAGPATH = 'diagnostics';

%normalisation parameters
radial_res = 20;
angular_res = 240;
% with these settings a 9600 bit iris template is
% created

%feature encoding parameters
nscales=1;
minWaveLength=18;
mult=1; % not applicable if using nscales = 1
sigmaOnf=0.5;

eyeimage = imread(eyeimage_filename);

savefile = [eyeimage_filename,'-houghpara.mat'];
[stat,mess]=fileattrib(savefile);

if stat == 1
    % if this file has been processed before
    % then load the circle parameters and
    % noise information for that file.
    load(savefile);
else
    % if this file has not been processed before
    % create diagnostic images.

% then perform automatic segmentation and
% save the results to a file

[circleiris circlepupil imagewithnoise] = segmentiris(eyeimage);
save(savefile,'circleiris','circlepupil','imagewithnoise');

end

% WRITE NOISE IMAGE
%
imagewithnoise2 = uint8(imagewithnoise);
imagewithcircles = uint8(eyeimage);

% get pixel coords for circle around iris
[x,y] =
circlecoords([circleiris(2),circleiris(1)],circleiris(3),size(eyeimage));
ind2 = sub2ind(size(eyeimage),double(y),double(x));

% get pixel coords for circle around pupil
[xp,yp] =
circlecoords([circlepupil(2),circlepupil(1)],circlepupil(3),size(eyeimage));
ind1 = sub2ind(size(eyeimage),double(yp),double(xp));

% Write noise regions
imagewithnoise2(ind2) = 255;
imagewithnoise2(ind1) = 255;
% Write circles overlayed
imagewithcircles(ind2) = 255;
imagewithcircles(ind1) = 255;
w = cd;
cd(DIAGPATH);
imwrite(imagewithnoise2,[eyeimage_filename,'-noise.jpg'],'jpg');
imwrite(imagewithcircles,[eyeimage_filename,'-segmented.jpg'],'jpg');
cd(w);

% perform normalisation
[polar_array noise_array] = normaliseiris(imagewithnoise,
circleiris(2),... 
circleiris(1), circleiris(3), circlepupil(2), circlepupil(1),
circlepupil(3),eyeimage_filename, radial_res, angular_res);

% WRITE NORMALISED PATTERN, AND NOISE PATTERN
w = cd;
cd(DIAGPATH);
imwrite(polar_array,[eyeimage_filename,'-polar.jpg'],'jpg');
imwrite(noise_array,[eyeimage_filename,'-polarnoise.jpg'],'jpg');
cd(w);

% perform feature encoding
[template mask] = encode(polar_array, noise_array, nscales, 
minWaveLength, mult, sigma0nf);
circlecoords
% circlecoords - returns the pixel coordinates of a circle defined by
% the
% radius and x, y coordinates of its centre.
% Usage:
% [x,y] = circlecoords(c, r, imgsize, nsides)
% Arguments:
% c           - an array containing the centre coordinates of the
% circle
% r           - the radius of the circle
% imgsize     - size of the image array to plot coordinates onto
% nsides      - the circle is actually approximated by a polygon, this
%              argument gives the number of sides used in this
%              approximation. Default is 600.
% Output:
% x      - an array containing x coordinates of circle boundary
% y      - an array containing y coordinates of circle boundary

function [x,y] = circlecoords(c, r, imgsize, nsides)

    if nargin == 3
        nsides = 600;
    end

    nsides = round(nsides);
    a = [0:pi/nsides:2*pi];
    xd = (double(r)*cos(a)+ double(c(1)) ) ;
    yd = (double(r)*sin(a)+ double(c(2)) ) ;
    xd = round(xd);
    yd = round(yd);
    % get rid of -ves
    % get rid of values larger than image
    xd2 = xd;
    coords = find(xd>imgsize(2));
    xd2(coords) = imgsize(2);
    coords = find(xd<=0);
    xd2(coords) = 1;
    yd2 = yd;
    coords = find(yd>imgsize(1));
    yd2(coords) = imgsize(1);
    coords = find(yd<=0);
    yd2(coords) = 1;
x = int32(xd2);
y = int32(yd2);

Segmentiris Module

% segmentiris - performs automatic segmentation of the iris region
% from an eye image. Also isolates noise areas such as occluding
% eyelids and eyelashes.
%
% Usage:
% [circleiris, circlepupil, imagewithnoise] = segmentiris(image)
%
% Arguments:
% eyeimage         - the input eye image
%
% Output:
% circleiris      - centre coordinates and radius
%                  of the detected iris boundary
% circlepupil     - centre coordinates and radius
%                  of the detected pupil boundary
% imagewithnoise  - original eye image, but with
%                  location of noise marked with
%                  NaN values

function [circleiris, circlepupil, imagewithnoise] = segmentiris(image)

% define range of pupil & iris radii

% CASIA
lpupilradius = 28;
upupilradius = 75;
lirisradius = 80;
uirisradius = 150;

% LIONS
% lpupilradius = 32;
% upupilradius = 85;
% lirisradius = 145;
% uirisradius = 169;

% define scaling factor to speed up Hough transform
scaling = 0.4;
reflecthres = 240;

% find the iris boundary
[row, col, r] = findcircle(eyeimage, lirisradius, uirisradius, scaling, 2, 0.20, 0.19, 1.00, 0.00);
circleiris = [row col r];
rowd = double(row);
cold = double(col);
rd = double(r);
irl = round(rowd-rd);
iru = round(rowd+rd);
icl = round(cold-rd);
icu = round(cold+rd);

imgsize = size(eyeimage);

if irl < 1
    irl = 1;
end

if icl < 1
    icl = 1;
end

if iru > imgsize(1)
    iru = imgsize(1);
end

if icu > imgsize(2)
    icu = imgsize(2);
end

% to find the inner pupil, use just the region within the previously
% detected iris boundary
imagepupil = eyeimage( irl:iru,icl:icu);

% find pupil boundary
[rowp, colp, r] = findcircle(imagepupil, lpupilradius, upupilradius
    ,0.6,2,0.25,0.25,1.00,1.00);

rowp = double(rowp);
colp = double(colp);
r = double(r);

row = double(irl) + rowp;
col = double(icl) + colp;

row = round(row);
col = round(col);

circlepupil = [row col r];

% set up array for recording noise regions
% noise pixels will have NaN values
imagewithnoise = double(eyeimage);

% find top eyelid
topeyelid = imagepupil(1:(rowp-r),:);
lines = findline(topeyelid);

if size(lines,1) > 0
    [xl yl] = linecoords(lines, size(topeyelid));
yl = double(yl) + irl-1;
xl = double(xl) + icl-1;
    yla = max(yl);
y2 = 1:yla;

ind3 = sub2ind(size(eyeimage),yl,xl);
imagewithnoise(ind3) = NaN;
imagewithnoise(y2, xl) = NaN;
end

%find bottom eyelid
bottomeyelid = imagepupil((rowp+r):size(imagepupil,1),:);
lines = findline(bottomeyelid);

if size(lines,1) > 0
[xl yl] = linecoords(lines, size(bottomeyelid));
yl = double(yl) + irl+rowp+r-2;
xl = double(xl) + icl-1;

yla = min(yl);

y2 = yla:size(eyeimage,1);

ind4 = sub2ind(size(eyeimage),yl,xl);
imagewithnoise(ind4) = NaN;
imagewithnoise(y2, xl) = NaN;
end

%For CASIA, eliminate eyelashes by thresholding
ref = eyeimage < 100;
coords = find(ref==1);
imagewithnoise(coords) = NaN;

Findline

% findline - returns the coordinates of a line in an image using the
% linear Hough transform and Canny edge detection to create
% the edge map.
%
% Usage:
% lines = findline(image)
%
% Arguments:
% image - the input image
%
% Output:
% lines - parameters of the detected line in polar form
%
function lines = findline(image)
[I2 or] = canny(image, 2, 1, 0.00, 1.00);
I3 = adjgamma(I2, 1.9);
I4 = nonmaxsup(I3, or, 1.5);
edgeimage = hystresh(I4, 0.20, 0.15);

theta = (0:179)';
[R, xp] = radon(edgeimage, theta);

maxv = max(max(R));
if maxv > 25
    i = find(R == max(max(R)));
else
    lines = [];
    return;
end

[foo, ind] = sort(-R(i));
u = size(i,1);
k = i(ind(1:u));
[y, x] = ind2sub(size(R), k);
t = -theta(x)*pi/180;
r = xp(y);

lines = [cos(t) sin(t) -r];

cx = size(image,2)/2-1;
cy = size(image,1)/2-1;
lines(:,3) = lines(:,3) - lines(:,1)*cx - lines(:,2)*cy;

Linecoords

% linecoords - returns the x y coordinates of positions along a line
% Usage:
% [x,y] = linecoords(lines, imsize)
% Arguments:
% lines       - an array containing parameters of the line in
% form
% imsize      - size of the image, needed so that x y coordinates
% are within the image boundary
% Output:
% x           - x coordinates
% y           - corresponding y coordinates
%

function [x,y] = linecoords(lines, imsize)

xd = [1:imsize(2)];
yd = (-lines(3) - lines(1)*xd ) / lines(2);
coords = find(yd>imsize(1));
yd(coords) = imsize(1);
coords = find(yd<1);
yd(coords) = 1;

x = int32(xd);
y = int32(yd);

Nonmaxsup

% NONMAXSUP
% % Usage:
% % im = nonmaxsup(inimage, orient, radius);
% % Function for performing non-maxima suppression on an image using an
% % orientation image. It is assumed that the orientation image gives
% % feature normal orientation angles in degrees (0-180).
% % % input:
% % inimage  - image to be non-maxima suppressed.
% % orient   - image containing feature normal orientation angles in
% % degrees (0-180), angles positive anti-clockwise.
% % radius   - distance in pixel units to be looked at on each side of
% each pixel when determining whether it is a local maxima or not.
% (Suggested value about 1.2 - 1.5)
% % Note: This function is slow (1 - 2 mins to process a 256x256 image).
% It uses % bilinear interpolation to estimate intensity values at ideal, real-
% valued pixel % locations on each side of pixels to determine if they are local
% maxima.
% %
% function im = nonmaxsup(inimage, orient, radius)
% if size(inimage) ~= size(orient)
%   error('image and orientation image are of different sizes');
% end
% if radius < 1
%   error('radius must be >= 1');
% end
% [rows,cols] = size(inimage);
im = zeros(rows,cols); % Preallocate memory for output image for speed
iradius = ceil(radius);

% Precalculate x and y offsets relative to centre pixel for each orientation angle
angle = [0:180]*pi/180; % Array of angles in 1 degree increments (but in radians).
xoff = radius*cos(angle); % x and y offset of points at specified radius and angle
yoff = radius*sin(angle); % from each reference position.

hfrac = xoff - floor(xoff); % Fractional offset of xoff relative to integer location
vfrac = yoff - floor(yoff); % Fractional offset of yoff relative to integer location

orient = fix(orient)+1; % Orientations start at 0 degrees but arrays start % with index 1.

% Now run through the image interpolating grey values on each side % of the centre pixel to be used for the non-maximal suppression.

for row = (iradius+1):(rows - iradius)
    for col = (iradius+1):(cols - iradius)
        or = orient(row,col); % Index into precomputed arrays
        x = col + xoff(or); % x, y location on one side of the point in question
        y = row - yoff(or);
        fx = floor(x); % Get integer pixel locations that surround location x,y
cx = ceil(x);
fy = floor(y);
cy = ceil(y);
tl = inimage(fy,fx); % Value at top left integer pixel location.
tr = inimage(fy,cx); % top right
bl = inimage(cy,fx); % bottom left
br = inimage(cy,cx); % bottom right

        upperavg = tl + hfrac(or) * (tr - tl); % Now use bilinear interpolation to
        loweravg = bl + hfrac(or) * (br - bl); % estimate value at x,y
        vl = upperavg + vfrac(or) * (loweravg - upperavg);

        if inimage(row, col) > vl % We need to check the value on the other side...
            x = col - xoff(or); % x, y location on the `other side' of the point in question
            y = row + yoff(or);
            fx = floor(x);
cx = ceil(x);
fy = floor(y);
cy = ceil(y);
tl = inimage(fy,fx); % Value at top left integer pixel location.
tr = inimage(fy,cx); % top right
bl = inimage(cy,fx); % bottom left
br = inimage(cy,cx); % bottom right
upperavg = tl + hfrac(or) * (tr - tl);
loweravg = bl + hfrac(or) * (br - bl);
v2 = upperavg + vfrac(or) * (loweravg - upperavg);

if inimage(row,col) > v2            % This is a local maximum.
   im(row, col) = inimage(row, col); % Record value in the output
   end
end
end
end

**Adjgamma**

% ADJGAMMA - Adjusts image gamma.
% function newim = adjgamma(im, g)
% Arguments:
% im     - image to be processed.
% g      - image gamma value.
% Values in the range 0-1 enhance contrast of bright
% regions, values > 1 enhance contrast in dark regions.

function newim = adjgamma(im, g)
    if g <= 0
        error('Gamma value must be > 0');
    end
    if isa(im,'uint8');
        newim = double(im);
    else
        newim = im;
    end

    % rescale range 0-1
    newim = newim-min(min(newim));
    newim = newim./max(max(newim));

    newim = newim.^(1/g);   % Apply gamma function

**Findcircle**

% findcircle - returns the coordinates of a circle in an image using the Hough transform
% and Canny edge detection to create the edge map.
% Usage:
% [row, col, r] = findcircle(image, lradius, uradius, scaling, sigma, hithres, lowthres, vert, horz)
%
% Arguments:
% image      - the image in which to find circles
% lradius      - lower radius to search for
% uradius      - upper radius to search for
% scaling      - scaling factor for speeding up the Hough transform
% sigma      - amount of Gaussian smoothing to apply for creating edge map.
% hithres      - threshold for creating edge map
% lowthres     - threshold for connected edges
% vert      - vertical edge contribution (0-1)
% horz      - horizontal edge contribution (0-1)
%
% Output:
% circleiris     - centre coordinates and radius of the detected iris boundary
% circlepupil     - centre coordinates and radius of the detected pupil boundary
% imagewithnoise - original eye image, but with location of noise marked with NaN values
%
function [row, col, r] = findcircle(image, lradius, uradius, scaling, sigma, hithres, lowthres, vert, horz)

lradsc = round(lradius*scaling);
uradsc = round(uradius*scaling);
rd = round(uradius*scaling - lradius*scaling);

% generate the edge image
[I2 or] = canny(image, sigma, scaling, vert, horz);
I3 = adjgamma(I2, 1.9);
I4 = nonmaxsup(I3, or, 1.5);
edgeimage = hysthresh(I4, hithres, lowthres);

% perform the circular Hough transform
h = houghcircle(edgeimage, lradsc, uradsc);

maxtotal = 0;

% find the maximum in the Hough space, and hence the parameters of the circle
for i=1:rd

    layer = h(:,:,i);
    [maxlayer] = max(max(layer));

    if maxlayer > maxtotal
        maxtotal = maxlayer;
    end

end
r = int32((lradsc+i) / scaling);

[row,col] = ( find(layer == maxlayer) );

row = int32(row(1) / scaling); % returns only first max value
col = int32(col(1) / scaling);
end
end

Hysthresh

% HYSTRESH - Hysteresis thresholding
% Usage: bw = hysthresh(im, T1, T2)
% Arguments:
%       im  - image to be thresholded (assumed to be non-negative)
%       T1  - upper threshold value
%       T2  - lower threshold value
% Returns:
%  bw  - the thresholded image (containing values 0 or 1)
% Function performs hysteresis thresholding of an image.
% All pixels with values above threshold T1 are marked as edges
% All pixels that are adjacent to points that have been marked as edges
% and with values above threshold T2 are also marked as edges.
% If it is assumed that the input image is non-negative
%
% A stack (implemented as an array) is used to keep track of all the
% indices of pixels that need to be checked.
%
function bw = hysthresh(im, T1, T2)

if (T2 > T1 || T2 < 0 || T1 < 0)  % Check thresholds are sensible
    error('T1 must be >= T2 and both must be >= 0 ');
end

[rows, cols] = size(im);  % Precompute some values for speed and convenience.
rc = rows*cols;
rcmr = rc - rows;
rpl = rows+1;

bw = im(:);  % Make image into a column vector
pix = find(bw > T1);  % Find indices of all pixels with value > T1
npix = size(pix,1); % Find the number of pixels with value > T1
stack = zeros(rows*cols,1); % Create a stack array (that should never % overflow!)

stack(1:npix) = pix; % Put all the edge points on the stack
stp = npix; % set stack pointer
for k = 1:npix
    bw(pix(k)) = -1; % mark points as edges
end

% Precompute an array, O, of index offset values that correspond to the
% surrounding pixels of any point. Note that the image was transformed
% into a column vector, so if we reshape the image back to a square the
% indices surrounding a pixel with index, n, will be:
% n-rows-1   n-1   n+rows-1
% n-rows     n     n+rows
% n-rows+1   n+1   n+rows+1
O = [-1, 1, -rows-1, -rows, -rows+1, rows-1, rows, rows+1];

while stp ~= 0 % While the stack is not empty
    v = stack(stp); % Pop next index off the stack
    stp = stp - 1;

    if v > rp1 && v < rcmr % Prevent us from generating illegal
        % Now look at surrounding pixels to see if they % Prevent us from generating illegal
        % and reshape the image
        % Calculate indices of points around this % Prevent us from generating illegal
        % Processed as well.
        index = O+v;
        for l = 1:8
            ind = index(l);
            if bw(ind) > T2 % if value > T2,
                stp = stp+1; % push index onto the stack.
                stack(stp) = ind;
                bw(ind) = -1; % mark this as an edge point
            end
        end
    end
end

bw = (bw == -1); % Finally zero out anything that was not an edge
bw = reshape(bw,rows,cols); % and reshape the image

Hough circle
% houghcircle - takes an edge map image, and performs the Hough transform
% for finding circles in the image.
% Usage:
% h = houghcircle(edgeim, rmin, rmax)
% Arguments:
% edgeim      - the edge map image to be transformed
% rmin, rmax  - the minimum and maximum radius values
% of circles to search for
% Output:
% h           - the Hough transform
%
function h = houghcircle(edgeim, rmin, rmax)

[rows,cols] = size(edgeim);
nradii = rmax-rmin+1;
h = zeros(rows,cols,nradii);

[y,x] = find(edgeim~=0);

%for each edge point, draw circles of different radii for index=1:size(y)
    cx = x(index);
cy = y(index);
    for n=1:nradii
        h(:,:,n) = addcircle(h(:,:,n),[cx,cy],n+rmin);
    end
end

Add circle

% ADDCIRCLE
% A circle generator for adding (drawing) weights into a Hough accumulator
% array.
% Usage:  h = addcircle(h, c, radius, weight)
% Arguments:
% h      - 2D accumulator array.
% c      - [x,y] coords of centre of circle.
% radius - radius of the circle
% weight - optional weight of values to be added to the accumulator array (defaults to 1)
% Returns:  h - Updated accumulator array.
function h = addcircle(h, c, radius, weight)

    [hr, hc] = size(h);
    if nargin == 3
        weight = 1;
    end

    % c and radius must be integers
    if any(c-fix(c))
        error('Circle centre must be in integer coordinates');
    end

    if any(radius-fix(radius))
        error('Radius must be an integer');
    end

    x = 0:fix(radius/sqrt(2));
    costheta = sqrt(1 - (x.^2 / radius^2));
    y = round(radius*costheta);

    % Now fill in the 8-way symmetric points on a circle given coords
    % [px py] of a point on the circle.
    px = c(2) + [x y y x -x -y -x];
    py = c(1) + [y x -x -y -y -x x y];

    % Cull points that are outside limits
    validx = px>=1 & px<=hr;
    validy = py>=1 & py<=hc;
    valid = find(validx & validy);
    px = px(valid);
    py = py(valid);

    ind = px+(py-1)*hr;
    h(ind) = h(ind) + weight;

Canny

% CANNY - Canny edge detection
%
% Usage: [gradient or] = canny(im, sigma)
%
% Arguments:  im       - image to be procesed
%              sigma    - standard deviation of Gaussian smoothing
%              filter
%                      (typically 1)
%              scaling - factor to reduce input image by
%              vert    - weighting for vertical gradients
%              horz    - weighting for horizontal gradients
%
% Returns:     gradient - edge stretch image (gradient amplitude)
%              or       - orientation image (in degrees 0-180, positive
%                         anti-clockwise)
% % See also:  NONMAXSUP, HYSTHRESH

function [gradient, or] = canny(im, sigma, scaling, vert, horz)

xscaling = vert;
yscaling = horz;

hsize = [6*sigma+1, 6*sigma+1]; % The filter size.
gaussian = fspecial('gaussian',hsize,sigma);
im = filter2(gaussian,im); % Smoothed image.
im = imresize(im, scaling);

[rows, cols] = size(im);

h = [ im(:,2:cols) zeros(rows,1) ] - [ zeros(rows,1) im(:,1:cols-1) ];
v = [ im(2:rows,:); zeros(1,cols) ] - [ zeros(1,cols); im(1:rows-1,:) ];
d1 = [ im(2:rows,2:cols) zeros(rows-1,1); zeros(1,cols) ] - ...
    [ zeros(1,cols); zeros(rows-1,1) im(1:rows-1,1:cols-1) ];
d2 = [ zeros(1,cols); im(1:rows-1,2:cols) zeros(rows-1,1); ] - ...
    [ zeros(rows-1,1) im(2:rows,1:cols-1); zeros(1,cols) ];

X = ( h + (d1 + d2)/2.0 ) * xscaling;
Y = ( v + (d1 - d2)/2.0 ) * yscaling;

gradient = sqrt(X.*X + Y.*Y); % Gradient amplitude.
or = atan2(-Y, X); % Angles -pi to + pi.
neg = or<0; % Map angles to 0-pi.
or = or.*~neg + (or+pi).*neg;
or = or*180/pi; % Convert to degrees.

NormaliseIris

% normaliseiris - performs normalisation of the iris region by
% unwrapping the circular region into a rectangular block of
% constant dimensions.
% % Usage:
% [polar_array, polar_noise] = normaliseiris(image, x_iris, y_iris,
r_iris,...
% x_pupil, y_pupil, r_pupil,eyeimage_filename, radpixels, angulardiv)
% % Arguments:
% image            - the input eye image to extract iris data from
% x_iris - the x coordinate of the circle defining the iris boundary
% y_iris - the y coordinate of the circle defining the iris boundary
% r_iris - the radius of the circle defining the iris boundary
% x_pupil - the x coordinate of the circle defining the pupil boundary
% y_pupil - the y coordinate of the circle defining the pupil boundary
% r_pupil - the radius of the circle defining the pupil boundary
% eyeimage_filename - original filename of the input eye image
% radpixels - radial resolution, defines vertical dimension of normalised representation
% angulardiv - angular resolution, defines horizontal dimension of normalised representation

function [polar_array, polar_noise] = normaliseiris(image, x_iris, y_iris, r_iris,...
  x_pupil, y_pupil, r_pupil, eyeimage_filename, radpixels, angulardiv)

global DIAGPATH
radiuspixels = radpixels + 2;
angledivisions = angulardiv-1;
r = 0:(radiuspixels-1);
theta = 0:2*pi/angledivisions:2*pi;
x_iris = double(x_iris);
y_iris = double(y_iris);
r_iris = double(r_iris);
x_pupil = double(x_pupil);
y_pupil = double(y_pupil);
r_pupil = double(r_pupil);

% calculate displacement of pupil center from the iris center
ox = x_pupil - x_iris;
oy = y_pupil - y_iris;
if ox <= 0
  sgn = -1;
elseif ox > 0
  sgn = 1;
else
  sgn = 0;
end

% calculate polar array
x = r.*cos(theta) + x_iris;
y = r.*sin(theta) + y_iris;
polar_array = [x, y];

% calculate polar noise
polar_noise = [ox, oy];
sgn = 1;
end

if ox==0 && oy > 0
    sgn = 1;
end

r = double(r);
theta = double(theta);

a = ones(1,angledivisions+1)* (ox^2 + oy^2);

% need to do something for ox = 0
if ox == 0
    phi = pi/2;
else
    phi = atan(oy/ox);
end

b = sgn.*cos(pi - phi - theta);

% calculate radius around the iris as a function of the angle
r = (sqrt(a).*b) + ( sqrt( a.*(b.^2) - (a - (r_iris^2))));

r = r - r_pupil;

rmat = ones(1,radiuspixels)'*r;

rmat = rmat.* (ones(angledivisions+1,1)*[0:1/(radiuspixels-1):1])';
rmat = rmat + r_pupil;

% exclude values at the boundary of the pupil iris border, and the iris
sclera border
% as these may not correspond to areas in the iris region and will
introduce noise.
%
% ie don't take the outside rings as iris data.
rmat = rmat(2:(radiuspixels-1), :);

% calculate cartesian location of each data point around the circular
iris
% region
xcosmat = ones(radiuspixels-2,1)*cos(theta);
xsinmat = ones(radiuspixels-2,1)*sin(theta);

xo = rmat.*xcosmat;
yo = rmat.*xsinmat;

xo = x_pupil+xo;
yo = y_pupil-yo;

% extract intensity values into the normalised polar representation
through
% interpolation
[x, y] = meshgrid(1:size(image,2), 1:size(image,1));
polar_array = interp2(x, y, image, xo, yo);

% create noise array with location of NaNs in polar_array
polar_noise = zeros(size(polar_array));
coords = find(isnan(polar_array));
polar_noise(coords) = 1;

polar_array = double(polar_array)/255;

% start diagnostics, writing out eye image with rings overlayed

% get rid of outlying points in order to write out the circular pattern
coords = find(xo > size(image,2));
xo(coords) = size(image,2);
coords = find(xo < 1);
xo(coords) = 1;

coords = find(yo > size(image,1));
yo(coords) = size(image,1);
coords = find(yo<1);
yo(coords) = 1;

xo = round(xo);
yo = round(yo);

xo = int32(xo);
yo = int32(yo);

ind1 = sub2ind(size(image), double(yo), double(xo));
image = uint8(image);

image(ind1) = 255;
% get pixel coords for circle around iris
[x, y] = circlecoords([x_iris, y_iris], r_iris, size(image));
ind2 = sub2ind(size(image), double(y), double(x));
% get pixel coords for circle around pupil
[xp, yp] = circlecoords([x_pupil, y_pupil], r_pupil, size(image));
indl = sub2ind(size(image), double(yp), double(xp));

image(ind2) = 255;
image(indl) = 255;

% write out rings overlaying original iris image
w = cd;
cd(DIAGPATH);

imwrite(image, [eyeimage_filename, '-normal.jpg'], 'jpg');
cd(w);
% end diagnostics
% replace NaNs before performing feature encoding
coords = find(isnan(polar_array));
polar_array2 = polar_array;
polar_array2(coords) = 0.5;
avg = sum(sum(polar_array2)) /
(size(polar_array,1)*size(polar_array,2));
polar_array(coords) = avg;

Encode

% encode - generates a biometric template from the normalised iris region,
% also generates corresponding noise mask
% Usage:
% [template, mask] = encode(polar_array, noise_array, nscales, ...
% minWaveLength, mult, sigmaOnf)
% Arguments:
% polar_array - normalised iris region
% noise_array - corresponding normalised noise region map
% nscales - number of filters to use in encoding
% minWaveLength - base wavelength
% mult - multicative factor between each filter
% sigmaOnf - bandwidth parameter
% Output:
% template - the binary iris biometric template
% mask - the binary iris noise mask
%
function [template, mask] = encode(polar_array, noise_array, nscales, minWaveLength, mult, sigmaOnf)

% convolve normalised region with Gabor filters
[E0 filtersum] = gaborconvolve(polar_array, nscales, minWaveLength, mult, sigmaOnf);

length = size(polar_array,2)*2*nscales;
template = zeros(size(polar_array,1), length);

length2 = size(polar_array,2);
h = 1:size(polar_array,1);

% create the iris template
mask = zeros(size(template));

for k=1:nscales
    E1 = E0{k};
    % Phase quantisation
    H1 = real(E1) > 0;
    H2 = imag(E1) > 0;
% if amplitude is close to zero then
% phase data is not useful, so mark off
% in the noise mask
H3 = abs(E1) < 0.0001;

for i=0:(length2-1)
    ja = double(2*nscales*(i));
    %construct the biometric template
    template(h,ja+(2*k)-1) = H1(h, i+1);
    template(h,ja+(2*k)) = H2(h,i+1);

    %create noise mask
    mask(h,ja+(2*k)-1) = noise_array(h, i+1) | H3(h, i+1);
    mask(h,ja+(2*k)) = noise_array(h, i+1) | H3(h, i+1);
end
end

Gaborconvolve

% gaborconvolve - function for convolving each row of an image with 1D log-Gabor filters
%
% Usage:
% [EO, filtersum] = gaborconvolve(im, nscale, minWaveLength, mult, ...
%                  sigmaOnf)
%
% Arguments:
%   im          - the image to convolve
%   nscale      - number of filters to use
%   minWaveLength   - wavelength of the basis filter
%   mult        - multiplicative factor between each filter
%   sigmaOnf    - Ratio of the standard deviation of the Gaussian describing
%                  the log Gabor filter's transfer function in the frequency
%                  domain to the filter center frequency.
%
% Output:
%   EO          - a 1D cell array of complex valued convolution results
%
function [EO, filtersum] = gaborconvolve(im, nscale, minWaveLength, mult, ...
                                          sigmaOnf)
    [rows cols] = size(im);
    filtersum = zeros(1,size(im,2));
EO = cell(1, nscale); % Pre-allocate cell array
ndata = cols;
if mod(ndata,2) == 1 % If there is an odd No of data points
    ndata = ndata-1; % throw away the last one.
end
logGabor = zeros(1,ndata);
result = zeros(rows,ndata);

radius = [0:fix(ndata/2)]/fix(ndata/2)/2; % Frequency values 0 - 0.5
radius(1) = 1;
wavelength = minWaveLength; % Initialize filter wavelength.

for s = 1:nscale, % For each scale.
    % Construct the filter - first calculate the radial filter component.
    fo = 1.0/wavelength; % Centre frequency of filter.
    rfo = fo/0.5; % Normalised radius from centre of frequency plane
    % corresponding to fo.
    logGabor(1:ndata/2+1) = exp((-log(radius/fo).^2) / (2 * log(sigmaOnf)^2));
    logGabor(1) = 0;

    filter = logGabor;
    filtersum = filtersum+filter;

    % for each row of the input image, do the convolution, back transform
    for r = 1:rows % For each row
        signal = im(r,1:ndata);

        imagefft = fft( signal );

        result(r,:) = ifft(imagefft .* filter);
    end

    % save the output for each scale
    EO(s) = result;

    wavelength = wavelength * mult; % Finally calculate Wavelength of next filter
end % ... and process the next scale
filtersum = fftshift(filtersum);
Gethamming distance

% gethammingdistance - returns the Hamming Distance between two iris templates
% incorporates noise masks, so noise bits are not used for calculating the HD
%
% Usage:
% [template, mask] = createiristemplate(eyeimage_filename)
%
% Arguments:
% template1       - first template
% mask1           - corresponding noise mask
% template2       - second template
% mask2           - corresponding noise mask
% scales          - the number of filters used to encode the templates,
%                   needed for shifting.
%
% Output:
% hd              - the Hamming distance as a ratio

function hd = gethammingdistance(template1, mask1, template2, mask2, scales)

    template1 = logical(template1);
    mask1 = logical(mask1);

    template2 = logical(template2);
    mask2 = logical(mask2);

    hd = NaN;

    % shift template left and right, use the lowest Hamming distance for shifts=-8:8
    templatels = shiftbits(template1, shifts, scales);
    maskls = shiftbits(mask1, shifts, scales);

    mask = maskls | mask2;
    nummaskbits = sum(sum(mask == 1));
    totalbits = (size(templatels,1)*size(templatels,2)) - nummaskbits;
    C = xor(templatels,template2);
    C = C & ~mask;
    bitsdiff = sum(sum(C==1));
    if totalbits == 0
        hd = NaN;
    end
else
    hd1 = bitsdiff / totalbits;
    if  hd1 < hd || isnan(hd)
        hd = hd1;
    end
end
end
if hd<0.4
    msgbox('Its a Match!')
else
    msgbox('Its Not Match!')
end

Shiftbits

% shiftbits - function to shift the bit-wise iris patterns in order to provide the best match
% each shift is by two bit values and left to right, since one pixel value in the normalised iris pattern gives two bit values in the template
% also takes into account the number of scales used
%
% Usage:
% [template, mask] = createiristemplate(eyeimage_filename)
%
% Arguments:
% template - the template to shift
% noshifts - number of shifts to perform to the right, a negative value results in shifting to the left
% nscales - number of filters used for encoding, needed to determine how many bits to move in a shift
%
% Output:
% templatenew - the shifted template

function templatenew = shiftbits(template, noshifts,nscales)
    templatenew = zeros(size(template));
    width = size(template,2);
    s = round(2*nscales*abs(noshifts));
    p = round(width-s);
    if noshifts == 0
        templatenew = template;
% if noshifts is negative then shift towards the left
elseif noshifts < 0

x=1:p;
templatenew(:,x) = template(:,s+x);

x=(p + 1):width;
templatenew(:,x) = template(:,x-p);

else

x=(s+1):width;
templatenew(:,x) = template(:,x-s);

x=1:s;
templatenew(:,x) = template(:,p+x);

end