IMPERIAL COLLEGE DEPARTMENT OF COMPUTING

IRIS RECOGNITION

MSc Group Project Spring 2009

Edmund Noon
Mark Howe
Arnar Jonsson
Andrew Durnin
Sebastian Smith

3/20/2009

An Iris Recognition package forming the focus of the group project module as a syllabus component for the Computing Science MSc program at Imperial College within the DoC
Produced with the very kind assistance of Dr. **Ashok Argent-Katwala** and Professor **Duncan Gillies**.

TABLE OF CONTENTS

1. Introduction

The broad aim of our project was to allow a user to upload an image of an iris and compare it to a pre-existing database to validate identity. Our software extracts a code unique to every iris and as such allows a range of functionality. Firstly an administrator can upload an image of an individual's iris and commit its code to a centralised database along with their name and access privileges. Secondly a user can upload an image of an individual's iris and validate their identity against the contents of the central database. Thirdly a user can upload two separate images and obtain a measure of similarity, thus allowing the system to be tested and calibrated. The software is implemented in Java with a graphical user interface created using the Swing Library.

2. Background

The rationale for using the iris as the basis for biometric recognition rests on the fact that the iris is cited as being the most robust feature for such a system, due to its stability over time and its high degree of complexity, uniqueness and inter-subject differentiation.

The National Laboratory of Pattern Recognition in Beijing made the CASIA database available for this project, which consists of iris images from 108 unique subjects (see Figure 1 on page 2).

Based predominantly on the research by John Daugman[1], whose iris recognition algorithms are now widely used in industry, we discuss in this report our success or otherwise in achieving a functional application that can match identities from a given image of an iris.

Daugman's strategy for iris recognition is broadly based on two functionally distinct elements; firstly the extraction of an iris bit-code which satisfies the intrinsically unique nature of each iris, and secondly a robust measure of bit-code similarity which corresponds to a statistically acceptable threshold of false match/false acceptance.

The bit-code extraction itself relies on having a useable bounded image of an iris section, and transforming this image from XY co-ordinated to doubly dimensionless co-ordinates. Daugman outlines proven mathematical algorithms for these steps together with the bit-code extraction and comparison, so our task in this respect was to interpret these algorithms for effective use and translate them into functional Java modules specific to our overall scheme. The mathematics behind this will be presented and assessed later in this report.

The next section will describe the delivered specification and engineering methods incorporated during development, and the various technologies used throughout the life-cycle of the project. Further sections will describe in more detail the architecture of the program and the various components that comprise it. Finally the report will evaluate and cross-reference the end solution with the initial specification, discuss the success of our approach to project management and how well the group worked together as a team.

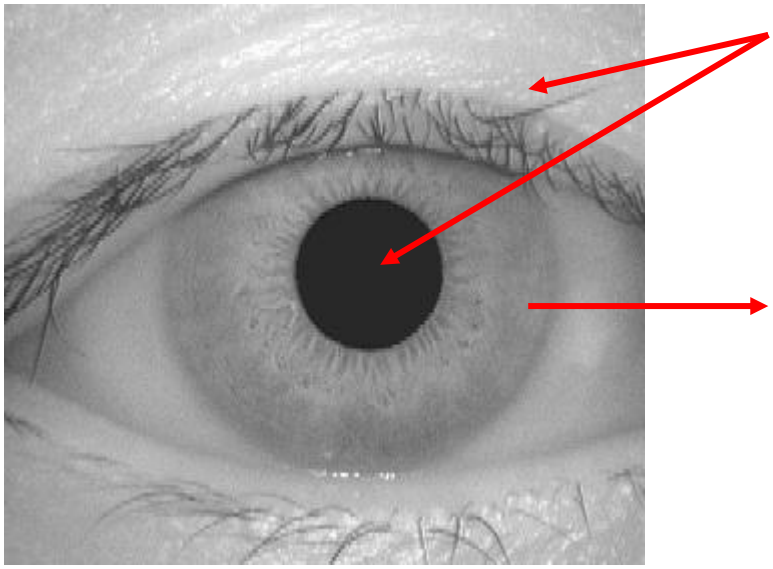
[1] For a detailed account see John Daugman's homepage: http://www.cambs.ac.uk/~jgd1000/

See also ("High confidence visual recognition of persons by a test of statistical independence." , 1993) ('How Iris Recognition Works', 2004)

The document will conclude by evaluating the success of the project and highlight areas of development that would have been followed if time had allowed.

Figure 1

An annotated image taken from the CASIA database



The eyelid/eyelash area and pupil are not accurate measures of biometric recognition, so needed to be removed
from the image before processing.

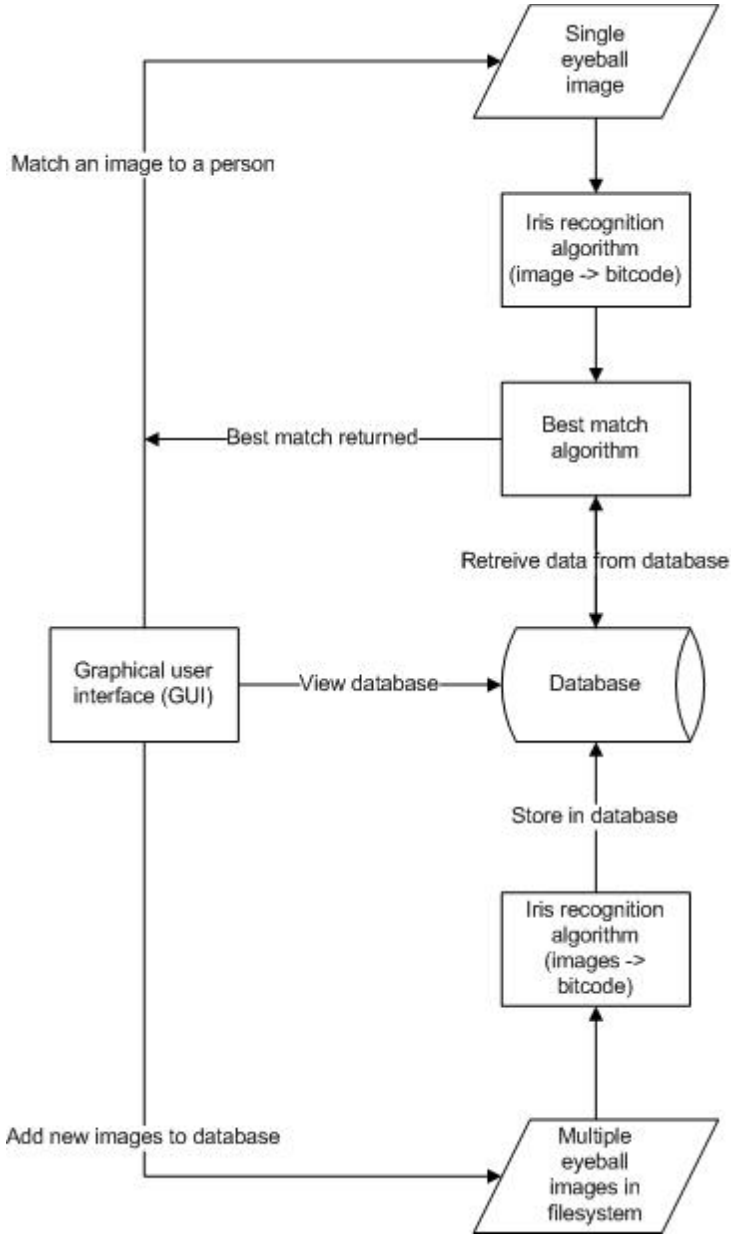
The iris; Cited as being the most robust feature for biometric recognition.

3. Delivered Specification

The design of a system flow diagram early on in the project gave us a good 'high-level' understanding of how the software would be designed (see figure 2 below). The final piece of software adheres extremely closely to this.

Figure 2

Initial system flow-diagram



The submitted software has the following broad functionality:

a. A GUI enabling basic user interaction.
b. The ability for a user to upload a bitmap of an Iris from an accessible file source.
c. Manipulation of the image such that the iris can be unwrapped, analysed and a unique bit-code extracted (Daugman 2007).
d. Indication of any positive identity match or the absence of such based on bit-code comparison (implemented using a hamming distance algorithm) with all records in the database, and the display of the corresponding ID and access privileges where applicable (Daugman 1993).
e. A utility that allows an administrator to upload and add new ID entries to the pre-existing database from the GUI.
f. A utility that allows an administrator to suspend or restore an existing users access privileges.
g. Automation of iris location such that the software can recognise the location of the pupil centre and boundaries of the iris itself.
h. A visual representation of the bit-code and unwrapped image for an uploaded iris image.
i. An indication of the level of certainty when a positive identification occurs.
j. A graphical representation of the contents of the database which can be updated as changes are made.
k. The ability to upload two iris images, compare them using our developed algorithms and view the corresponding hamming distance (measure of independence)
l. A diagnostic function enabling a user to obtain measures of system efficacy and statistics in terms of functionality.

4. Implementation and Architecture

4.1 Graphical User Interface (GUI)

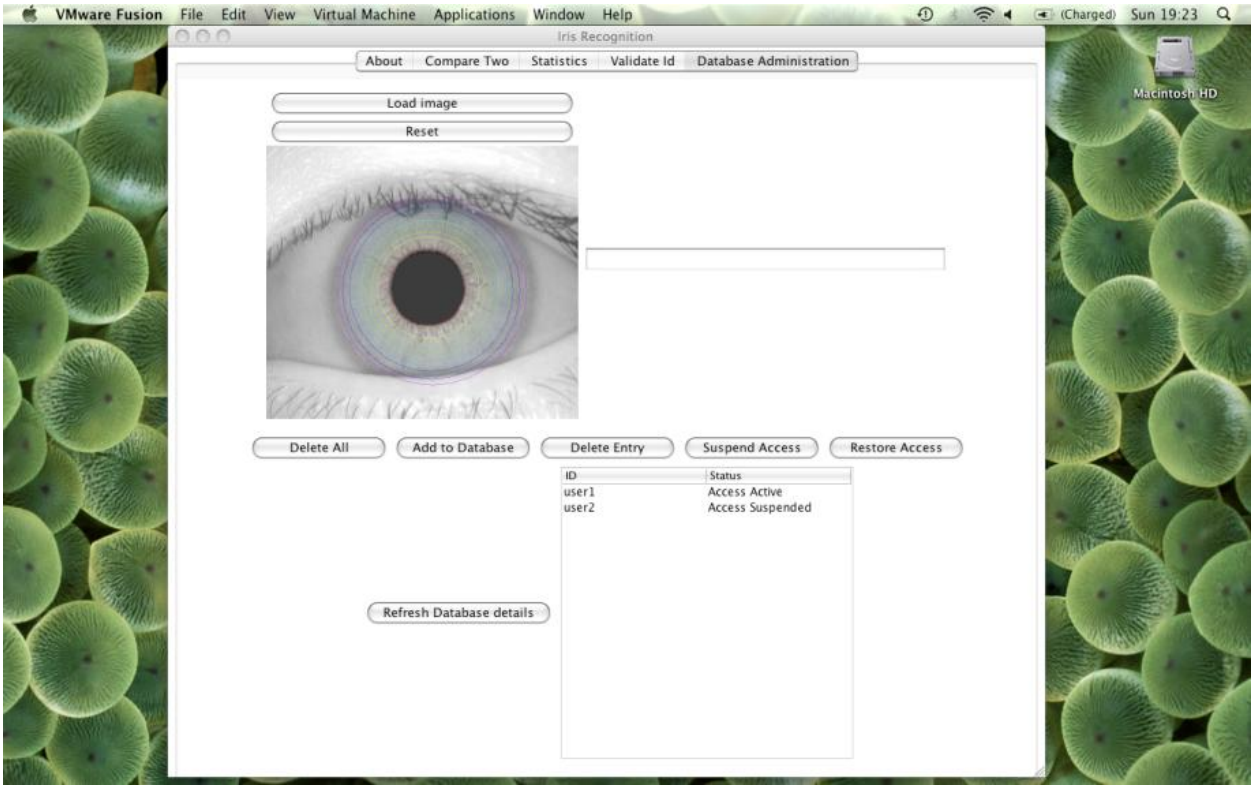
The graphical user interface (GUI) grew throughout the development of the project as the code for various functional components was added. To begin with, the GUI was created by writing lines of code alone. This was fine in terms of creating a functional interface; however there was limited scope in terms of addressing the aesthetical and usability aspects of the design. As a result, we investigated using various 'drag and drop' applications that are supposed to help in designing graphic based applications. The first of these was 'Netbeans', a stand alone program that will write the Java in correspondence with the graphics (buttons, labels, frames etc) that the user places on the screen in a 'what you see is what you get' fashion. Unfortunately we found that Netbeans was frustrating to use and offered little advantage over manually writing the code ourselves. Our next port of call was to use a plug-in for Eclipse called 'Jigloo'. This provided more useful functionality to us and had the added benefit of being directly incorporated with

the Eclipse environment. Jigloo made trying out various layouts and GUI ideas much easier in abstract, though we still had to make changes to our GUI by editing the code ourselves. We also designed and incorporated an image to display in the 'About' tab, that is shown on opening the program. This was created using 'GIMP', an open source picture editing program.

The Swing Libraries provided the basis for our GUI implementation. Broadly speaking we have a single JFrame which contains a JTabbedPane forming the main body. Each of the sections below are implemented separately and extend the JPanel classFigure 3 below shows the database administration pane in the GUI.

Figure 3

A screenshot of the GUI



The GUI was split into different sections by using tabs that the user can select depending on the operation they wish to perform. The tabs are:

About

As described above, this tab gives the user a brief description about the program, including where and when it was designed as well as displaying the program logo. There are no interactive options.

Compare Two

This tab allows the user to upload two images and to compare the two to determine whether or not the irises are from the same person. The results are displayed with their 'hamming result'. When an image is uploaded, the tab automatically displays the extracted and transformed image of the iris and a visual representation of its bit-code. The tab can be reset to return it to its original state.

Validate

Allows the user to upload an image and click the 'validate' button, thus checking whether or not there is a matching identity stored in the database. The results are returned and displayed in a text box, together with the appropriate hamming distance. This panel includes a visual representation of the result of a search. There is a colour change to indicate a match failure (red), a match found with access rights (green) and a different indication for a match found but without access rights (yellow with a warning). The colour change is implemented as a background colour change to one of the Panels included. The entire tab can be reset, returning each element to its original state. The panel includes the same details for each uploaded image as the 'compare two' tab.

Statistics

This tab allows the user to view current statistics about the program, including the success rate in correctly identifying identities by iris images.

Administrator

The administrator tab allows the user to directly manipulate the database, including buttons to 'add to the database', 'delete from the database' and 'delete all from the database'. This tab also allows an administrator to suspend and restore access rights, to simulate the effect of a user being absent for a specific amount of time. The tab includes a table of all current users and their access status. This table can be refreshed to include any recent changes. The table itself is implemented using a JTable, with its TabelModel extending AbstractTableModel, changing the functionality so the results from the database are retrieved and set as values in the tab when the GUI is initialized. When the refresh table button is pressed, the table is populated again in a similar fashion.

4.2 Database

Using the group's allocated Imperial DOC postgres database, it was decided to only store the bit-code relevant to each iris of an individual, that individuals name and their access status. Each of these is implemented as a column in a single table, with the id as primary key. ID is of type VARCHAR, bit-code of type ARRAY and access status as BOOLEAN.

The JDBC connection utility library provided the core functionality to create methods for the databaseWrapper class. As well as setting up a robust database connection, this class encapsulates all commonly needed database operations and implements them as member functions. The creation of a databaseWrapper object sets up a connection, a resultset and statement object as public attributes of the object. Common operations include adding and retrieving from the database and updating the status. The resultset is scrollable, which allows operations using the databaseWrapper class to retrieve information about the size of the database and to manipulate the results as desired. The JDBC prepared statement function is used in member functions which put and get database records. A fresh resultset can be returned if needed, to accommodate run-time updates.

The database administration tab in the GUI forms an effective front-end for the database, and implements a JTable that displays the current contents of the database. A databaseWrapper object is used to return the results for each position in the table. A new resultset is created when the JTable is updated at run-time. See the appendix for samples of databaseWrapper code.

As detailed elsewhere, the bit-code class is an extension of the bit-set class. A key responsibility of the databaseWrapper class is to convert the bit-code class into a format which can be stored in the database. A number of possibilities exist in this regards, but the methods toBitCode() and toByteArray() convert to and from a bit-code object a byte array representing the value of each 8-bit sequence in the bit-code as one byte. Each byte's value is then stored in the database as part of a string sequence. The member functions which retrieve bit-codes from the database convert the retrieved array of integers to a byte array and then calls the toBitCode() method and returns a bit-code object. The returned bit-code object is identical to the original bit-code and has all the same attributes and functionality.

The toBitCode() and toByteArray() methods manipulate byte-values for each 8-bit sequence in the bit-code. Relative to the position of each set bit, corresponding values are added to the byte such that they can represent each possible combination of 8 bits. When converting back, each 8 bit-sequence is re-created so that each byte-value is represented by its 8-bit sequence in binary, implemented as 8 bits in the bit-code. See Appendix 3 for code examples.

4.3 Automated Iris Location

Before any iris unwrapping, or conversion to a numeric code can take place the iris has to be located within the image. Fortunately, access to the high quality images from CASIA Iris Image database (kindly made available by the National laboratory of Pattern Recognition in Beijing), means that many of the problems of acquiring an image don't apply in this project. All of the images are in focus and there are no peculiarities on any iris.

Daugman suggests ("High confidence visual recognition of persons by a test of statistical independence." , 1993) a method for locating the iris is to make use of its circular properties. The idea is to search over (x,y) in the image (x,y) being the centre of a circle of radius r such for a maximum of:

$$\max(r, xo, yo) \left| \frac{\partial}{\partial r} \oint \frac{Image\ (x,y)}{2\pi r} d\theta \right| \qquad (\mathbf{1})$$

Initial tests with implementing this algorithm were very successful at locating the pupil but accuracy on the iris/sclera boundary proved elusive. Careful choice of parameters improved this to about 45% of iris's correctly located. (Though a higher proportion than this might be usable results as, it will be shown later most of the calculation is performed on the pupil edge of the image.)

By using an edge detection filter (A 3x3 Isotropic Gradient Operator for Image Processing, 1968)(see figure 4 below) this proportion was increased to above 80%. Further work is to be done in this area to improve this further.

The Sobel edge detection filter is a simple rule which subtracts the value of the pixel on one side from the value of the pixel on the other side, and if this is above a certain threshold turns the pixel on.

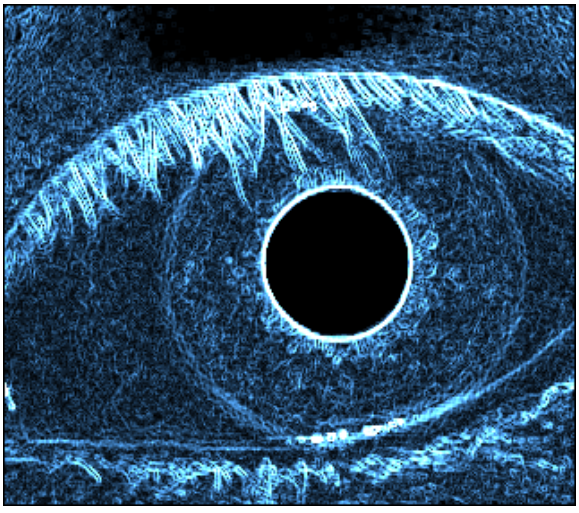


Figure 4

Visual representation of an edge detection filter

In the implementation presented here, the suggestion of Daugman ("High confidence visual recognition of persons by a test of statistical independence." , 1993) is followed in regards to the loop integral. When locating the pupil the loop excludes the lower quarter of the circle as this is the region in which specula artifacts often occur. This is not strictly necessary when using images from CASIA as these images are largely free of such artifacts, but it was implemented as it seemed like good practice. In attempting to locate the outer iris boundary the upper quarter is also excluded as this together with the lower quarter is often obscured by the eyelid. These are drawn in on the image (Figure 5).

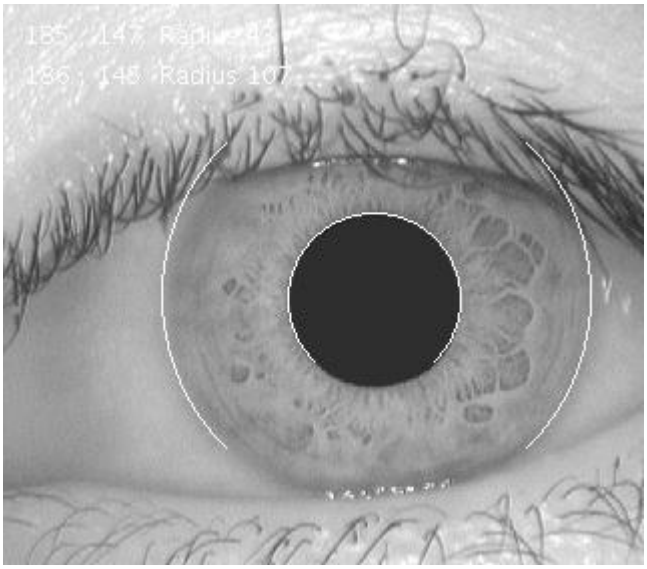


Figure 5

Automated location of the iris boundary

The circular path followed here is generated by the algorithm due to Bresenham (A Linear Algorithm for Incremental Digital Display of Circular Arcs, 1977). The values along this path are summed and averaged (to allow for the number of pixels in the path – the integer equivalent of dividing by 2*PI*r). And this is compared with the path of r+1.

Given the higher accuracy of locating the pupil in these images the centre of the pupil was used as a guide to locate the iris outer boundary.

4.4 Bitcode Generation

4.4.1 The Unwrapper Class

When the boundaries of an iris have been successfully located, the next step is to unwrap the iris into a two dimensional image. This is done because computationally it's much cheaper to work with the iris that way. This may sound like a trivial task at first, but the fact that the inner and outer boundaries of an iris are not concentric makes it a bit harder to achieve. It must also be kept in mind that a human iris is elastic and not static. To eliminate these obstructions, the following formulas are applied to map the iris from its circular form to a two dimensional image (see Figure 6 below)

$$x(r,\sigma) = (1-r)\left(x_p + r_p cos(\sigma)\right) + r\left(x_i + r_i cos(\sigma)\right) \quad \textbf{(2)}$$

$$y(r,\sigma) = (1-r)\left(y_p + r_p sin(\sigma)\right) + r\left(y_i + r_i sin(\sigma)\right)$$

$$r: radius\ [0,1], \qquad \sigma: angle\ [0,360],$$

$$(x_p, y_p): Center\ of\ pupil, \qquad (x_i, y_i): Center\ of\ iris$$

$$r_p: radius\ of\ pupil, \qquad r_i: \text{radius of iris}$$

Figure 6

An Unwrapped Iris

Unwrapper Class methods

The main feature of the UnWrapper class is a method called unWrap which accepts an eye image along with the parameters constraining the iris in that image. The constraining parameters can be passed to the method as separate integers or all in one via the EyeDataType class (which is a simple class to contain the constraining parameters). The unWrap method then returns an unwrapped image. The unWrap method is also overloaded to allow a feature called „overwrapping“, which generates an unwrapped image for an additional number of degrees as specified by an input parameter (see Figure 7 below). This feature is used by the BitcodeGenerator for computational efficiency.

Figure 7

An Unwrapped Iris with 60 degree overlap

There are also several other methods contained in the UnWrapper class. The unWrapByteArr method utilises the unWrap method and then converts the output image to an array of bytes. This is used for optimisation purposes in the BitcodeGenerator class. Finally there are two methods, unWrapWithGuides and originalWithGudes, which output the original and unwrapped images with coloured guides for visualisation purposes (see Figures 8 and 9).

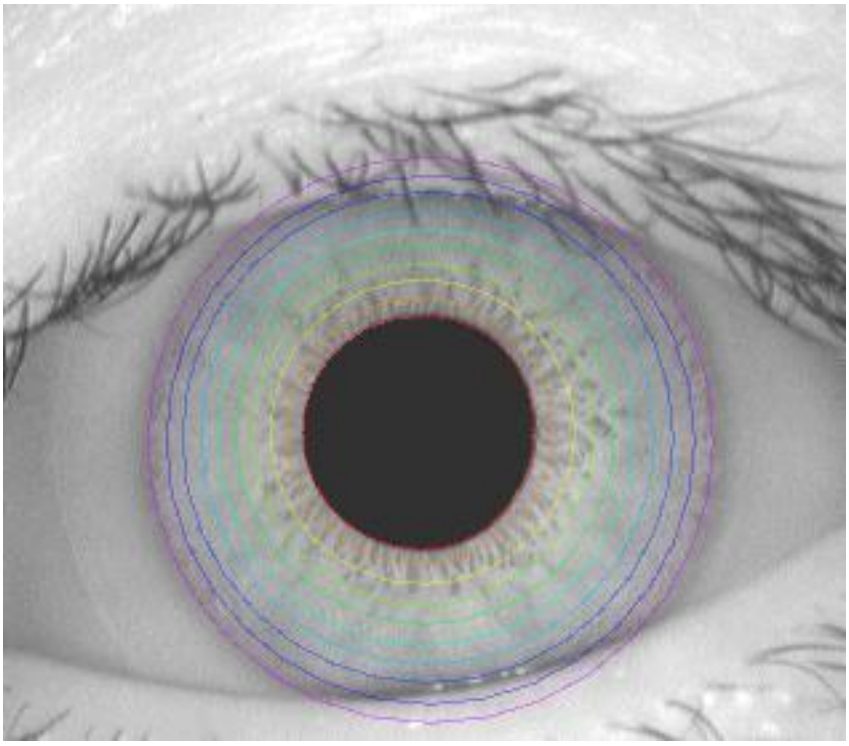


Figure 8

Original eye image with colored guides

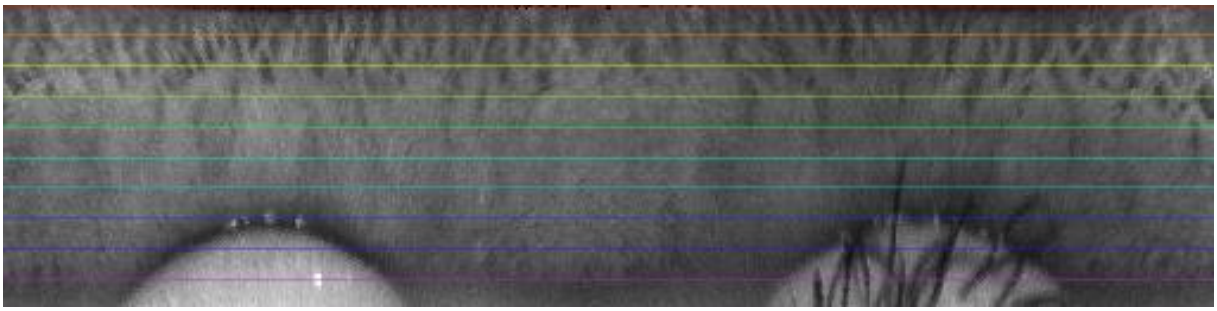


Figure 9

Unwrapped Image with colored guides

4.4.2 *BitCode Creation*

The actual encoding of an unwrapped iris to a bitcode is obviously very important for the whole solution to work. The encoding method used in this project was invented by John Daugman and is based on the use of so called Gabor wavelets (see image 5). A wavelet is defined by the equation:

$$G(x,y) = e^{-\pi((x-x_0)^2/\alpha^2+(y-y_0)^2/\beta^2)}e^{(-2\pi i(\omega_1(x-x_0)+\omega_2(y-y_0))} \quad \textbf{(3)}$$

Where

$$Re\big(G(x,y)\big) = e^{-\pi((x-x_0)^2/\alpha^2+(y-y_0)^2/\beta^2)}\cos(-2\pi\big(\omega_1(x-x_0)+\omega_2(y-y_0)\big)) \quad \textbf{(4)}$$

And

$$Im\big(G(x,y)\big) = e^{-\pi\left((x-x_0)^2/\alpha^2+\frac{(y-y_0)^2}{\beta^2}\right)}\sin\left(-2\pi\big(\omega_1(x-x_0)+\omega_2(y-y_0)\big)\right) \quad \textbf{(5)}$$

The equations are essentially just trigonometric functions in a Gaussian envelope. By changing the six constants (α,β,ω1, ω2, x0, y0), different wavelets can be created. The α and β constants control the width and height of the Gaussian envelope, x0 and y0 it's centre and ω1 and ω2 the frequency of the trigonometric functions.

To construct one bit of the bitcode, first the constants must be chosen to define a Gabor wavelet. We then convolve this wavelet (having a z-axis value range of [-1,1]) with an unwrapped iris image (which is thought of as an array of numbers with range[0,255]). Finally, we sum up all the values of the convolution result (range [-255,255]). If that sum adds up to a value greater than zero we add a bit value of 1 to the bitcode, otherwise, a zero. It's possible to use either the real or imaginary part of the Gabor wavelet, or both, in which case two bits are generated for each convolution. The size of a given bitcode is then determined by the number of different Gabor wavelets convolved with the unwrapped image.

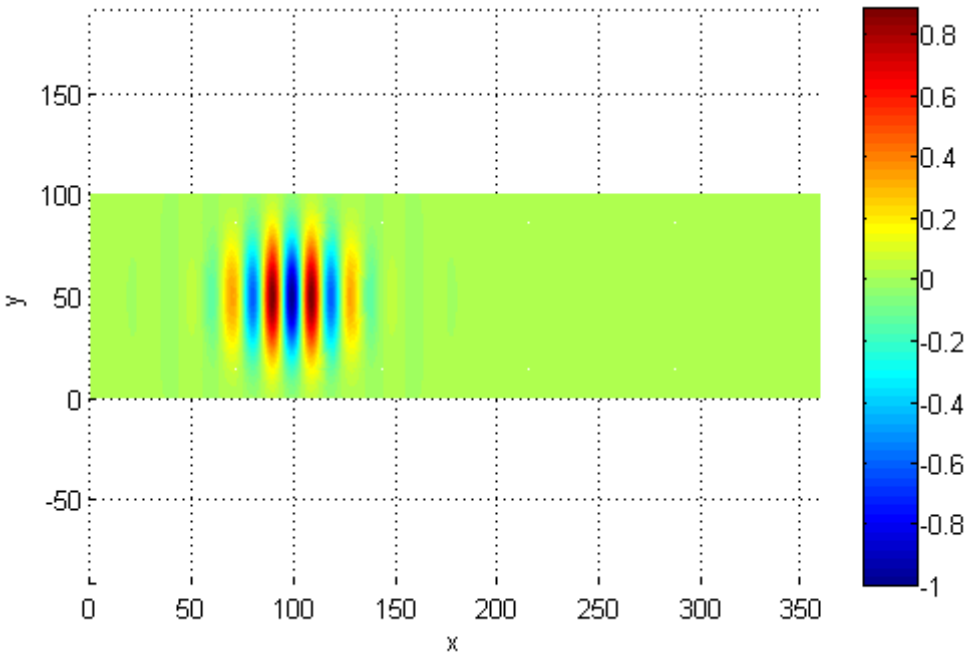


Figure 10

Gabor wavelet (top down view): α=50,β=50,ω1=0.05, ω2=0, x0=50, y0=100

Bitcode

The BitcodeGenerator has an overloaded constructor which enables the caller to either use the default Gabor wavelet parameter ranges, or pass those parameter ranges to the constructor. Those ranges can then be changed later on by calling the initialiseParams method. To make it easier to define and work with a parameter range a special GaborParameters class was written.

At first, a method called getBitcode was used to generate a bitcode. This method basically looped through all different parameter ranges as defined at construction. For every parameter set the method calls the gaborFilter2D method, which is responsible for actually generating the Gabor wavelet and convolving it with the unwrapped image.

Although the former setup was well written, easily understandable and very useful for verifying that the algorithm was working, it was far too slow. For this reason a new method called getFastBitcode was written. This method uses integer code, and utilises the fact that the Gabor wavelets used in our approach are merely the same few wavelets moving in the direction of the x-axis of the unwrapped image. This optimisation proved to be very successful and reduced the runtime by a factor of about 200.

4.4.3 The BitCode class

To hold the information returned by the BitcodeGenerator a specialised BitCode class was written. This class extends the Java built in BitSet class which has some very useful methods for working with sets of bits, but still lacks essential functionality required by our application.

BitCode Class methods

The class can be instantiated with or without passing the constructor a maximum size for the bitcode. An overloaded method to add bits to the bitcode was written to simplify bitcode generation. A flexible method for visualising a bitcode called getBitCodeImage was also contained in this class (see Figure 11). The class accepts a width and height for the output image as well as the number of bits per each column of the image.

Figure 11

A bitcode as visualised by the getBitCodeImage

The final method of interest in the BitCode class is called hammingDistance. This static method is used to calculate the simularity between two bitcodes, returning 0.0 for identical bitcodes, and 1.0 for two exactly opposite bitcodes. It is in this method that the rotation of an eye between two images is accounted for, using a parameter called shiftNum contained in the BitCode class. A normal hamming distance is then performed for the two bitcodes, shifting the first one by N*shiftNum where N is an integer number ranging from [-R,R] where R is a constant integer number of degrees. The method then returns the lowest of the 2*R hamming distances calculated. See section 4.5 for more detail

4.5 Hamming Distance

The Hamming distance measures the differences between two bit patterns and may be used to determine whether or not the two patterns being considered belong to the same iris. More precisely, given two bit patterns *X* and *Y*, of equal length ***N***, the Hamming distance *HD* is:

$$HD = \frac{1}{N}\sum_{n=1}^{N}(X_n \text{ XOR } Y_n) \quad \textbf{(6)}$$

The XOR operation ensures that in the sum we only count corresponding bits which disagree so that what we have is a fractional measure of the dissimilarity between the two codes. In theory, since each iris pattern is unique, the corresponding bitcode should also be unique and the Hamming distance between two code samples from the same iris should be zero. However, this is never the case in practice. The normalization process used to unwrap the iris in the first place into a rectangle onto which the Gabor convolution is done is not perfect and between two different images of the same iris may produce a slightly different result so that the convolution is actually working with an image which is not perfectly identical to the one it will be compared to. Also, the presence of noise in any image will mean that there are points in each image that are not suitable for comparison. In practice then, in trying to establish a match we would seek a Hamming distance close to zero, whereas codes from different irises should be completely independent leading to a Hamming distance of around 0.5 (equal probabilities of getting a 1 or 0) or higher. In fact, as a result of his original research on a dataset of 4258 images, which included 10 each of one subset of 70 eyes, leading to (4258×4257 - 700×9)/2 = 9,060,003 unique pairings, Daugman concluded that the probability of obtaining a false positive with a Hamming distance of between 0 and 0.33 was about 1 in 16 million.

The problem of noise may be overcome by supplying along with each iriscode a corresponding mask, that is to say, a binary array of equal dimension to the array representing the iriscode indicating whether or not the corresponding cell in the iriscode is valid for comparison or invalid corresponding to the presence of noise at that point in the image. With this modification taken into account, the Hamming distance formula now becomes:

$$HD = \frac{1}{N - \sum_{i=1}^{N} X_i \text{ OR } Y_i} \sum_{n=1}^{N} (X_n \text{ XOR } Y_n \text{ AND } X'_n \text{ AND } Y_n{}') \quad (7)$$

The prime indicates the mask arrays, and the modified denominator means that we are really only interested in counting valid bits, so that the measure is now the number of valid disagreeing bits divided by the total number of valid bits.

Another issue encountered when comparing iriscodes is that of rotational inconsistencies between the two different being compared. If one iris is rotated slightly relative to the other, then the corresponding iriscodes will be misaligned. To overcome this, Daugman, suggested employing a bitpattern shift whereby one iris pattern is shifted left/right while the other is held in place. Each time a Hamming distance is calculated and the minimum such distance taken to be the final answer, since it represents the best possible match.

Our algorithm allows for the bit shifting suggested by Daugman. From a sequence of both left and right shifts (compensating as described earlier for possible rotational inconsistencies present in the images) we obtain the minimum distance between the two bitcodes and report this result as the Hamming distance for our matching procedure.

5. Software Engineering

5.1 Methods and Technologies

It was decided during our first team meeting on the overall way in which to approach the project. This included the discussion of which software engineering methods to use, the various technologies to utilise and how to organise ourselves to ensure that we would be working closely as a team. This section will discuss these decisions and comment on how effective each of these decisions proved to be in facilitating the successful completion of the project.

The first collective concern was to ensure that all team members had a clear view of the problem at hand. Once this had been established, the entire team was in a position to make informed choices about the technologies and methodologies that would best suit the needs of the project. It was decided that our software would be written in the Java language due to its ease of portability and expandability. This was a language that all team members had had experience of to some extent and so it seemed a natural choice in selecting this for our project. In addition to this, SVN software was used in order to maintain reliable and up to date code and to ensure that new additions to the project were written in coherence with the current state of the software. On reviewing the various possibilities, we settled on using 'GoogleCode' as our main repository due to its stability and ease of use. This proved to be more than sufficient in meeting our needs.

With the fundamental technological decisions made, the discussion progressed to include which software engineering approach would be most suitable given the nature and scale of the project. It was decided that we would adopt the agile methodology, using in particular the spiral model of development. We planned to have regular builds and to develop the software with basic functionality first, progressing towards more complex tasks once these had been adequately implemented. This seemed to be the most suitable method of development as it would require regular team meet-ups and reviewing of our design. Given that all members of the group were relative 'fledgling developers', this approach proved to facilitate good inter-team support and ensured that the project continued to develop in a positive way (addressing the requirements as laid out by the specification). Tasks were rotated amongst group members as much as possible, allowing for different perspectives and ensuring mutual satisfaction for each component. This has given the project a 'well-rounded' feel that all members are happy with.

In addition to the above, there were a few other technological components to the project. For example, the central database was to be stored on the IC DOC postgres server. Utilising this meant familiarising ourselves with SQL, as well as learning how to remotely connect to the server via our Java program. For the mathematical underpinning, Matlab was used at times to ease the transition from 'algorithm' to code. We also discussed using LaTex to produce our progression reports and commenting our code in such a way that we could incorporate the JavaDoc facility provided by the Eclipse development environment. It was decided that the learning curve needed to use LaTex in comparison with the benefits it would bring were too great and so we dedicated this 'saved time' in further refining our code. There was consensus amongst group members that given the time scale of the project, that this was a necessary sacrifice to make as it allowed us to address some of the more fundamental issues of the project.

5.2 Optimization

As discussed elsewhere, a decision was made to use many of the built in Java objects, to encapsulate our own data and functions within objects and to keep the programming style as simple as possible. This proved very successful early on. Team members were able to understand each others code and make extensions and even changes when needed. However this clarity comes with a cost and that is performance.

Built in objects can be slow

The iris location algorithm sums the value of pixels in a circular path (A Linear Algorithm for Incremental Digital Display of Circular Arcs, 1977)for each value of x,y, and r in the search space. Initially this process acted on a Java BufferedImage object directly, and this approach did provide good clarity. However by copying this BufferedImage into a two dimensional integer array at the start of the process and then using this reduced the search time to locate a pupil and iris from 16 seconds to less than 2 seconds.

Generating the Bit-code

The initial implementation of the bit-code generation algorithm was clear and flexible, and taking about 5.7 seconds per image it was very acceptable for the initial work. However when attempts were made to optimize the parameters used in the bit-code generation this proved too slow. The original code is seen in Appendix 1.

When calculating the bit-code there are a number of nested loops, there is one along the theta axis of the iris image, and nesting inside this are a pair of loops for each axis of the convolution box for the Gabor filters. Calculations inside this nesting can be done order 10^6 times. To achieve the 5.7 seconds mentioned above a BufferedImage object had already been replaced by an array.

However the for each of the convolution boxes (see how the bitcode is generated in section 4.4.2) the Gaussian is the same on each pass, as are the cosine and sine terms. The first attempt at optimizing the code moved these outside the main loop. These are also the same for any given set of parameters, so their calculation was moved into the parameter initialization part of the class (which is normally called by the constructor).

Only the sign of these calculations is required and so it was straightforward to change these to integer arrays (multiplied by 65,536 – no need to divide later). A final improvement (in speed terms) was to remove the call the GaborFilter2D method and move the code inside the loop.

This reduced the time taken for the conversion to bit-code down to about 0.02 seconds (on the same hardware as the previous 5.7s time). Some tests were then done to compare the bit-codes of the original routine and the new faster one and the codes whilst not exact had a hamming distance of less than 2%.

5.3 Unit Testing

One of the policies agreed on by our team was to write unit tests for our code. Unit testing is a great way to test code, one method at a time. By writing test code in an organized way, debugging time later on can be reduced significantly, and overall quality of code will almost certainly be much higher. Our policy was to follow a commonly accepted methodology of unit testing; write unit tests for every public method of a class, excluding GUI code and trivial methods (such as simple set and get methods). Once written, unit tests also provide a fast and reliable way of verifying that a change made to some code will not scew the output of that code, or break other parts of the program relying on the modified code. Since the Eclipse IDE (which was used by our team) has built in jUnit support it was in fact very easy to run the tests as needed.

Our conclusion of the use of unit testing is very positive. Since tests were written alongside classes and their methods, we spent way less time debugging than expected. Rather than doing informal testing at a later point, it was verified straight away if a method worked correctly or not. Even though the writing of test code took some time, it was an investment that paid off extremely well.

6. User Guide

**See Appendix 1 for complete user guide*

7. Evaluation

7.1 Software

Our original specification is as follows:

7.1.1 Features

Essential

a. A GUI enabling basic user interaction.
b. The ability for a user to upload a bitmap of an Iris from an accessible source.
c. A utility for the user to identify and record the centre of the pupil, edge of iris and eyelid for their uploaded image.
d. Manipulation of the image such that the iris can be analysed and a unique bit-code extracted (Daugman 2007).
e. Indication of any positive identity match or the absence of such based on bit-code comparison with all records in the database, and the display of a corresponding ID where applicable ("High confidence visual recognition of persons by a test of statistical independence." , 1993).

Desirable

f. A utility that allows an administrator to upload and add new ID entries to the pre-existing database, from the GUI and in bulk.
g. Automation of the image manipulation such that the software can recognise the location of the pupil centre and boundaries of the iris itself.
h. An indication of probability of a false match when a positive identification occurs.

Luxury

i. An automated ability to eliminate the impact of blemishes such as eyelashes or glare on an uploaded image, whether it's being added to the database or identity matched.
j. A facility to allow users to capture images of their iris and use them directly in the application.

Undesirable

k. The application being OS dependent.

We will assess each item independently and then detail any aspects of the submitted software which were not detailed in the original specification

7.1.2 GUI

By cross-referring our GUI against the original specification, we can conclude that this area of the project has satisfied and extended our original requirements. The database integration and extra visual features have been added as a matter of course as necessary capabilities. The GUI is easy to use and abstracts away the internal workings of the program. It allows the user to access all the functionality of the code. Whilst it is not necessarily associated with the GUI, it is worth noting at this point that the program is not 'platform specific', in that it will run in a Windows machine, a Mac machine and so on. With that said, we built the GUI in such a way that it will adopt the aesthetics of the native platform, so that windows, buttons and so on will be in keeping with the operating system on the computer that the program is being executed on. Areas that would have been extended include making the administrative tab password protected, automatic refreshing of the JTable displaying database contents and a more elaborate database interface.

7.1.3 Database

The database satisfies all the features initially specified, and the use of JDBC maintains platform independence. However, the efficiency of the database may not be optimal. Given more time we would explore ways of pushing the hamming distance calculations down into the database to improve this, in addition to exploring ways of using a bit-stream format in the database. We did explore using streaming, but found that bytes weren't always accurately encoded in the database itself, and on return the bit-codes sporadically differed from their original state. In this context where accuracy is hugely important, we abandoned this approach. It may well prove to be the more efficient method of transferring data to the database however, using a field of type BYTEA in postgres. The database wrapper class allows the user to manipulate the database contents in more ways than was initially specified, with a visual representation of the data and the possibility of direct real-time access rights adjustment. Facilities which could have been extended include using images of both irises for each individual, more elaborate access rights differentiation between individuals in the database, and further information held about individuals in the database as a whole to facilitate a particular access orientated implementation of the system.

7.1.4 Bit-Code/Unwrapping /Hamming distance Algorithms

The functionality described in the original specification in points a-g and k have all been satisfied and are underpinned by the algorithms as previously described. Points h,I, and j have not been fully realized. Point h was not considered due to time constraints, and due to the fact that refining the hamming distance took priority with the time we had. Point I will be elaborated upon in more detail subsequently in the 'further work' section.

Parameter Optimization

Daugman reports ("High confidence visual recognition of persons by a test of statistical independence." , 1993) such good results with his algorithm for example under ideal conditions with 482,600 comparisons a threshold could be chosen with no false positives or false negatives.

Initial results of the code presented here were therefore somewhat disappointing. The hamming distance (HD) between images of two unrelated eyes was about 0.30 though between related eyes it was often lower than this.

There was some confidence that the algorithms were correctly implemented but there were a large number of parameters which had been somewhat arbitrarily chosen, often by what seemed to look nice in graphs produced by Matlab.

The first attempt to improve the results saw a coarse search over a number of these parameters on each occasion generating new bitcodes from the images of three eyes and calculating the HD. Two of these images were of the same eye and one was different. By testing several thousand combinations it was hoped that a better set of parameters might be indicated.

Biassed cosine

An initially surprising finding was that as the frequency of the Gabor filter was increased, the HD also increased, and for the non matching pair this number approached 0.5. However the images of the Gabor filters in ('How Iris Recognition Works', 2004), and others seem to show about 1.5 wavelengths. The contradiction was quickly traced to the real part of the Gabor filter. This is a cosine multiplied by a Gaussian. With the lower frequencies the real part is nearly always positive because the negative parts of the cosine are multiplied by a much lower and this was affecting the statistical independence of the results.

At this point a new parameter was added to our implementation such that the real part could be ignored and a bit code using only the unbiased sine used. This immediately gave better results, though it was unsatisfactory to ignore half of the data.

It was later suggested to the team that another solution would be to bias the real term so that the sum of the term did become zero. This was achieved by subtracting the average from each term. A better solution was thought to be to multiply the negative part by a scaling factor to bring it into balance with the positive part. However initial testing suggested it wasn't better in average and the best case achieved was not as good as the simple biasing of the cosine. Though, much more time has been spent on this method.

Whilst this improved the HD of non matching eyes, and it improved the numbers for the three images being considered, when other images were tested the results were still disappointing. At this time a bitcode was taking over 5 seconds to generate. To do further parameter optimization this needed to be improved. See elsewhere in this report for details of how this was achieved, but after this process the bitcode could be calculated in under 0.1 second.

With the faster code a larger test set was created. This consisted of three pairs of images each of the same eye and another six images of different eyes. Again in an attempt to find the best fit for the data a coarse grid was employed simply a series of nested loops. Whilst this too produced parameters which gave good results again when these were tested on a larger set of data the results were not as good as hoped. Interestingly when trying a number of the better solutions, it was different eyes in the larger set which caused problems.

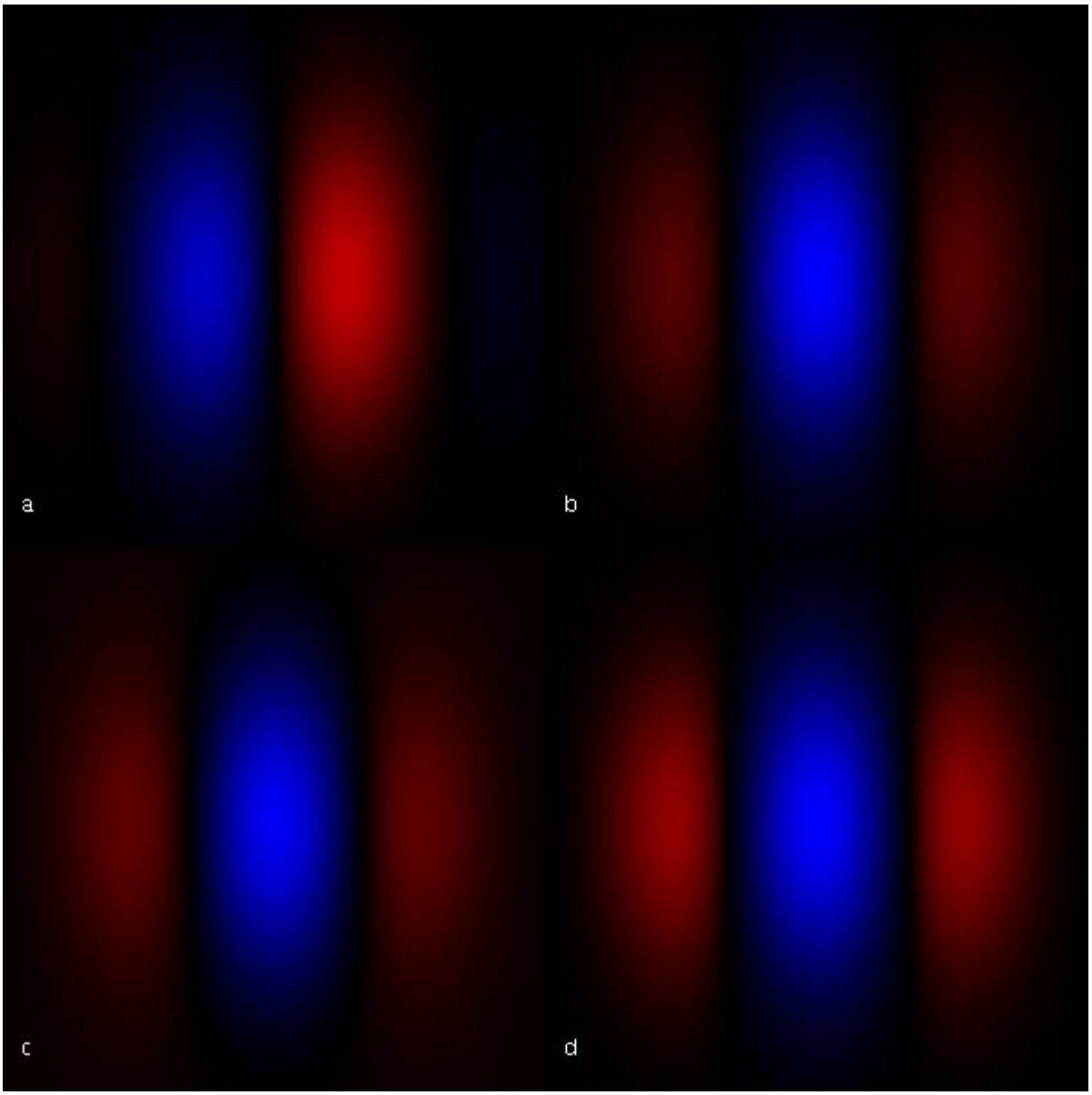


Figure 12

Due to the symmetry of the sine wave (a) there is no bias to its sign. However the equivalent cosine wave (b) might have such a problem. By subtracting the average an be reduced which produces a slightly ugly red box, so perhaps scaling the negative part is probably a better solution (d), but see text. Red is negative, blue positive.

A maximization algorithm

At this point the time taken to span the grid of test points was becoming significant. On the underpowered old PC the previous search had taken 10 hours, attempting a finer search didn't seem realistic.

At this moment the search space of parameters was four dimensional, with two being integer (for the size of the smallest convolution box, and another for the largest box – the middle box is just interpolated). The other two related to the frequencies, again one each for smallest and largest box. Other parameters did vary such as whether or not to use the cosine term, but separate runs were done for each of these. Some parameter variations have not yet been tried. For example the convolution box is always square, and the angle of the wavelet is always perpendicular to the radius.

The integer terms made many of the standard maximization routines difficult to implement. An attempt was made to use a modified Simplex algorithm (Nelder, et al., 1965) but it proved too difficult to modify in a short space of time. The aim was to fine a set of parameters which worked well, rather than necessarily the best set.

A not very good modified Simplex algorithms (One D Simplex)

With that in mind a routine was produced to try and find some improved parameters. It is somewhat inspired by Simplex:-

1) Choose a starting point (possibly randomly)
2) Choose another point within a 'radius' of the first
3) If this is an improvement extrapolate along this line until it stops improving
4) If this is not an improvement, reflect it across the original point
5) If this is not an improvement reflect it, but only take a small fraction of the distance
6) If an improvement has been made repeat the above (from 2)
7) If no improvement is made repeat a small number of times (from 2)
8) Try a small change (both positive and negative) in each direction in turn
9) Then halve the radius in 2, repeat from 2, until the radius has been halved 4 times

There is no theoretical basis for the above. In tests so far it did seem to significantly improve bad starting points, however if it is a good algorithm the space being searched must have many local maxima as most attempts at running this terminated at different solutions – though often quite reasonable ones. Remember the aim here was to find a good answer.

It's that condor moment

The test set was increased to 87 images. This was a set created by taking 216 images (two of each of the 108 eyes) and testing with the iris location routine that we had in use at the time. 87 were correctly located. (This has been subsequently improved to over 80%.) With a test set this long passing through the above routine took many hours. As a random starting point was employed, it seemed that it would be easy to run multiple sessions in parallel, and indeed it was. Using condor, installed on the computer network at Imperial College Department of Computing,

up to 500 computers are available when not in use by an individual at the keyboard. One D Simplex was tried on the enlarged set, three hundred times. This involved 2.9Bn Hamming Distance calculations. It did not find a set of parameters such that the positives and negatives could be separated by a single threshold. The best threshold found would have no false positives and 3 (out of 25) false negatives.

Possible over fitting

The test set was further extended to 600 images, with up to 7 images of a given eye. Using the parameters provided by 1D Simplex, the graphs of positives and negatives had a 35-40% overlap.

Once again a further refinement in parameter searching was begun. The range was narrowed by previous experience but was still much too large to iterate over with any confidence of success. Very small changes in some parameters cause large changes in the results. With the benefit of condor the above results were significantly improved. The overlap (were there exist false positives or negatives) was reduced to about 8%. The remarkable feature is the small size of the standard deviation of the non matching pairs. This is very consistent over a wider range of parameters. For a security application it would make sense to choose the threshold below this point and reject people who should be accepted rather than the converse. [See graph in the conclusions.]

Obviously rather than try and fit the whole set, it may be better to fit 30% of the set or there will be a risk of fitting to the test set. But given that the failure distribution doesn't change much with small changes in the parameters , there is hope that is not the case.

Image Noise

The issue described earlier in our discussion of hamming distance, namely that of accounting for image noise, was certainly considered and incorporated into the Hamming distance code without any difficulty. However, since we did not have time to construct a good noise model, that is to say, a definitive method of detecting automatically any noise present within the iris images, our results did not in the end incorporate the use of noise masks. We expect that this would certainly have some impact on the accuracy of our matching procedure, however the extent to which it informs the results would be difficult to quantify a priori.

We report later, in the section dealing with suggested improvements some suggestions as to how this might be achieved.

The iris location algorithm is still not performing as well as hoped. More work is need here to improve its accuracy (currently only 87% of irises correctly located from the test set). Effort is better spent improving this rather than speeding it up.

7.2 Further Work

Overview

If this project were to be extended the two pieces of code most in need of optimization are the hamming distance and the iris location routines. These issues are functionally distinct but affect each other in implementation.

In a recent scan to search for better parameters 2.9Bn hamming tests were done. As the test set of iris images increases the number of bitcodes to calculate increases linearly, but the number of hamming tests increases like n^2. It is easy to see than in order to perform statistical analysis on a large set of data the speed of the hamming function becomes critical.

The reason the hamming function is slow is that repeated tests are needed with one of the tested codes shifted each time to allow for a possible rotation of one of the images with respect to the other. For each given orientation of the eye, currently six bits are produced. In optimising the parameters [see] other numbers of bits are considered. With the current objects in use such a change would be easy to implement. (In fact the hamming function does so automatically, it looks up how many bits need to be shifted in between each comparison. However, at a high computational cost.

After work on the format of the bitcode is finished it would not require a large amount of work to replace the bitcode object with an array of bytes (providing less than 8 bit shifts were required for each test). The bitcode object might remain in other parts of the program, or perhaps it would be replaced everywhere.

7.2.1 Image Noise

As mentioned earlier, the unavoidable presence of noise in any image will surely affect the accuracy of the results to some degree and it is therefore preferable to incorporate into the software a method for automatically accounting for this. It is at this point that the mask array mentioned earlier in our discussion of the Hamming distance calculation would receive its information and accompany the bitcode as an input into the Hamming distance algorithm so that the invalid bits in the code can be avoided in our calculations thereby reducing the chances of erroneous results.

There appear to be three sources of noise which an automatic procedure should account for. The eyelids themselves are considered noise from the point of view they occlude part of the iris pattern from analysis, this also applies to the eyelashes, and then lastly, any bright specular reflections usually at the bottom of the image should also be captured.

7.2.2 Eyelid Detection

We report here on some of the methods used to try and detect eyelids in an image. Daugman's approach is to adapt his integro-differential operator by changing the path of integration from circular to spline curves statistically chosen so as to fit the expected shape of the eyelids in the image.

Another approach, employed by Arvacheh and Tizhoosh, combines Daugman's operator with a spherical model based on a spherical shape for the eyeball and the expected eyelid curve in different degrees of openness of the eye. The openness of the eye is defined as the angular position of the eyelid with respect to the centre of the sphere:

$$x^2 + y^2 + y^2 = R^2 \tag{8}$$

An eyelid curve can then be obtained by intersecting this sphere with a plane which passes through the *x* axis making an angle ϕ with the *z* axis as shown overleaf in figure 13.

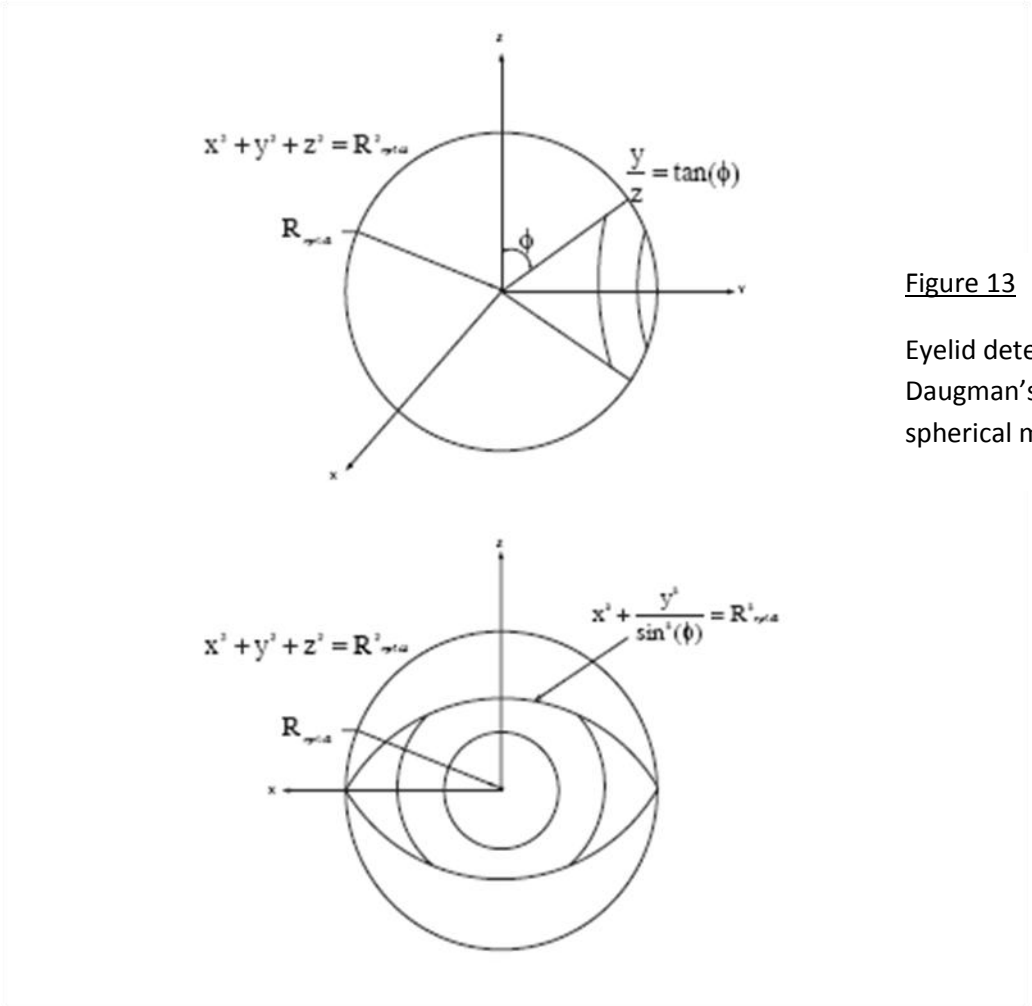


Figure 13

Eyelid detection combining Daugman's operator with a spherical model

This plane can be written as:

$$\tan\phi = y/z \tag{9}$$

The intersection curve then takes the shape of an ellipse:

$$x^2 + y^2/\sin^2\phi = R^2 \qquad (\mathbf{10})$$

This curve can then be transformed to polar coordinates so as to make it more appropriate for our image:

$$r = \frac{R}{\sqrt{\sin^2\vartheta + \frac{\cos^2\vartheta}{\sin^2\phi}}} \qquad (\mathbf{11})$$

With *R* known, having already detected the circles corresponding to the iris and limbus boundaries, the algorithm could now search through various values ϕ to locate the upper and lower eyelids.

Finally, another possible approach would be, given the two circles for the pupil and limbus boundaries, to use a linear Hough transform applied to possible edge points within this annular region to fit a line to the upper and lower eyelid. Then, for each of these two lines, a second horizontal line is drawn which intersects the Hough line at the iris edge closest to the pupil. As Figure 14 below illustrates, the region thus obtained is then taken to be the region over which the convolution is performed.

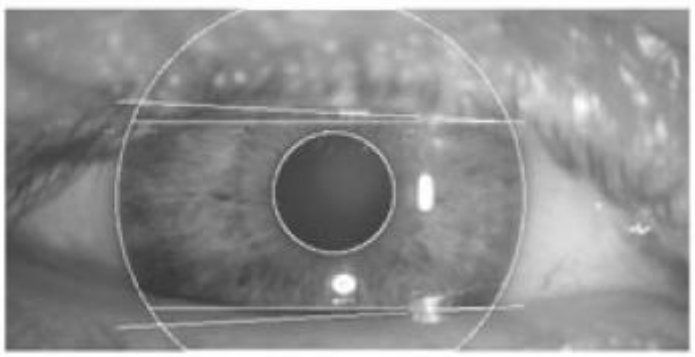


Figure 14

Showing a linear Hough transform applied to edge points with accompanying intersection

7.2.3 *Eyelash Detection*

The simplest approach to removing eyelashes would be to make use of a thresholding technique assuming that the eyelashes are quite dark compared to other features of the iris. This assumption is important and would render the method useless if applied under imaging conditions whereby certain parts of the iris have been rendered equally dark and would also be identified for removal quite erroneously, an example of such images would be those in the LEI database.

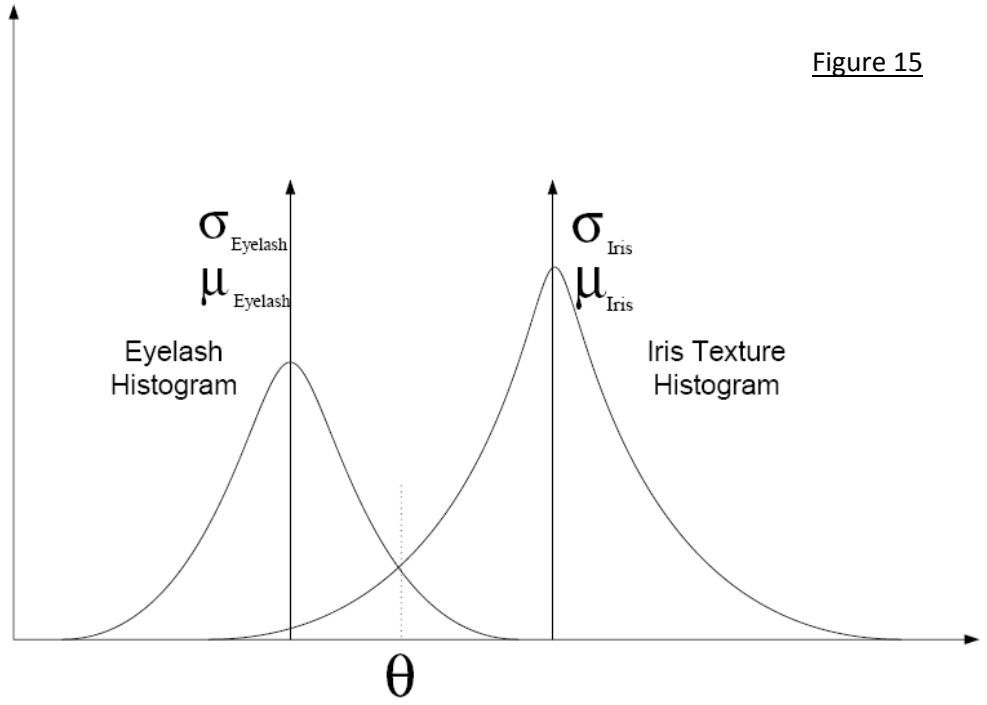
Kong and Zhang have proposed a method for eyelash detection where eyelashes are divided into two groups, separable eyelashes which are isolated and multiple eyelashes bunched together and overlapping in the image. To detect separable eyelashes are then detected using 1D Gabor filters, based on the idea that a convolution of a separable eyelash with a Gaussian smoothing function will result in a low output value, so that if the result is lower than some threshold it is taken as belonging to an eyelash.

Multiple eyelashes are detected by examining variance of intensity. If the variance of intensity values in a small window is lower than a threshold, then the centre of the window is considered as a point in an eyelash. Their model also makes use of the use of a connective criterion whereby each point in an eyelash should connect with another point or an eyelid.

Finally, we mention another method, reported by Arvacheh in which two histograms are obtained, one for the iris texture values and one for the eyelashes. For the iris histogram, in order to remove the effect of eyelashes the maximum value of the histogram is considered as the mean of the iris texture, and the variance is then estimated from only those values of the histogram greater than the mean. The mean and variance of the histogram of eyelashes is estimated in a similar way as for the iris with the difference of considering the area outside of the localized iris.

For the images in the CASIA database, making use of the observation that the pupil and eyelashes have the lowest intensity values, the mean value of the eyelashes was defined as the local maximum that occurs two standard deviations lower than the iris mean value. The author then stresses that this definition of the mean value was a process of trial and error aimed at avoiding erroneous identification of the iris texture. The variance of the eyelashes is estimated using only those values that are less than the estimated mean value assuming all the areas that are darker than the mean value refer to eyelashes in the image. Figure 15 below shows the threshold point ϑ which identification either way takes place

Figure 15



7.2.4 Specular Reflections

A thresholding technique characterized by high pixel values close to 255 may be applied to isolate reflections, what the results of this procedure together with the previously discussed Hough method applied to the eyelids might lead to, ideally, are shown below in figure 16.

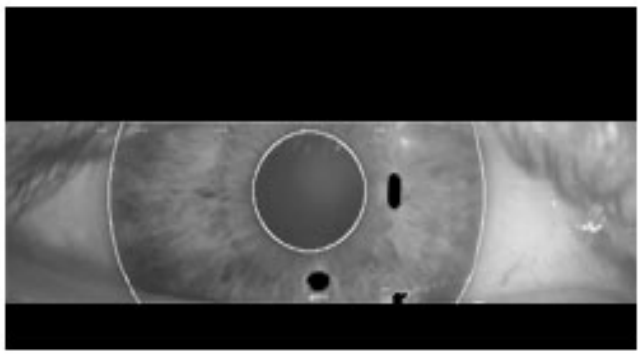


Figure 16

Possible thresholding technique with Hough method

As suggested earlier then, pixels corresponding to any form of noise, whether it is due to eyelids, eyelashes or reflections have their coordinates recorded into an array which then gets passed into whatever iris matching procedure is being used, in our case the Hamming distance, at which point only valid pixels are considered.

7.2.5 Automatic Detection of the Iris

We present here an alternative method for automatic localization of the iris based on the use of the Hough transform. As a preliminary step, the input into the transform is an edgemap, which at its simplest would consist of two-dimensional array of 1s and 0s of the exact same dimension as the original image from which we are to extract the iris. Each cell in the edgemap array would contain a 1 if the same cell in the original image corresponded to an edge. The idea then is that among these edges would be included the iris and limbus boundaries, the eyelids, eyelashes and unavoidably some noise.

There is more than one way to create such a map. The simplest method is to use Sobel filtering, which is defined by two 3×3 arrays which visit each pixel in the original image in turn taking a measure of the gradient (rate of change in intensity) in both the *x* and *y* directions, if that intensity exceeds a certain threshold then that point is considered to correspond to an edge, otherwise it is not and is ignored.

A more sophisticated approach to building an edgemap is to use the Canny detection algorithm, which proceeds by first smoothing the image using a Gaussian blur to eliminate noise. It then, like the Sobel method, finds the image intensity gradient at each point so as to highlight regions with high spatial derivatives. The algorithm then tracks along these regions and eliminates any pixel that is not at the maximum (nonmaximum suppression). The gradient array is then further reduced by hysteresis. Hysteresis is used to track along the remaining pixels that have not been suppressed. Hysteresis uses two thresholds and if the magnitude is below the first threshold, it is set to zero (made a nonedge). If the magnitude is above the high threshold, it is made an edge. And if the magnitude is between the two thresholds, then it is set to zero unless there is a path from this pixel to a pixel with a gradient above the higher threshold.

Once an edgemap has been obtained, we can then pass in this array and apply the Hough procedure as follows. Suppose we denote our set of edge points as (x_j, y_j) where j runs from 1 to N, then the idea is to consider each edge point in turn and search the edgmap for possible circles passing through the point of centre (x_C, y_C) and radius r. The Hough transform would then be defined as:

$$H(x_j, y_j, x_C, y_C, r) = \sum_{j=1}^{N} h(x_j, y_j, x_C, y_C, r) \qquad (12)$$

Where the function $h(x_j, y_j, x_C, y_C, r)$ = 1 and not zero only when $g(x_j, y_j, x_C, y_C, r)$ = 0 where g is the function describing the circle:

$$g(x_j, y_j, x_C, y_C, r) = (x_j - x_C)^2 + (y_j - y_C)^2 - r^2 \quad (13)$$

So it essentially collects votes for possible candidate circles as it moves through the set of edge points. The method does however suffer from the drawback that in practice it turns out that the algorithm does actually rely on the user providing a judiciously chosen threshold value, with the result that any possible circle coordinates (x_C, y_C, r) with votes below this number will not be considered. In fact, from this point of view, the integro-differential operator has the advantage since it works only on the raw derivative data.

7.3 Project Management

Our initial adoption of agile methods affected how the project was managed. The 'Spiral' model, mentioned Each member of the group had a specific role, with interaction between these when appropriate, which each member updated the rest of the team about regularly. Our initial plan to have regular builds was facilitated by the use of a central repository, so that each of us could check their code worked well with the overall scheme. The basic functionality was completed first, with more elaborate goals built on top of these. The spiral model highlights several key facets of the agile approach, namely an allowance for reciprocity between implementation stages and specification, allowing for an interaction between the two. This helped to shape the final software because the initial specification didn't include exhaustive details in terms of the style of user interaction or format of user interface. Once basic functionality was completed to an initial level, we were able to refine the specification in some cases and decide that extra functionality was needed in certain areas such as the database. These changes were only apparent at this stage, and highlight the benefits of a flexible software development strategy. Clearly this approach could have been extended in a commercial application, where time constraints are less intrusive.

The approach as described required regular interaction between group members to facilitate the ongoing assessment of how requirements were being met and any changes to the specification that were becoming apparent . In addition, this required stringent administrative duties, including taking notes at each meeting, and other administrative roles being adequately assigned.

7.4 Group Work

The group worked well together largely because of the range of skills and backgrounds. The project allowed each member to use their skills appropriately. Communication between group members was an area where improvements could have been made, particularly as each individual's task was large, detailed and often strongly distinct from other areas of the project.

Broadly speaking the project was split as follows:

Edmund Noon (Team Leader): GUI interface, image manipulation, report writing.
Sebastian Smith (Team Secretary): Database Implementation, report writing, GUI support & refinement, note taking in meetings.
Arnar Jonsson: Implementing Daugman's algorithms in Java.
Andrew Durnin: Database Implementation and Management, GUI support and report writing.
Mark Howe: Implementing Daugman's algorithms in Java.

8. Conclusions

The main aim of this project (taken from the initial report MSc Group Project – Iris Recognition – Report 1) is ".. to allow a user to upload an image of an iris and compare it to a pre-existing database to ascertain or verify identity.." and with some caveats this was achieved.

In addition all of the desirable additional features again with a caveat were also achieved.

Daugman in a report ('How Iris Recognition Works', 2004) summarizing his work presents a graph which shows virtually no false positives or negatives in the identification of an individual. The code we present here will correctly reject an unknown individual with certainty (in the tested set), and will correctly authenticate over 91% of those known to it. But it will also reject 8% of individuals who should not be rejected. As a security application this is the correct way round. However there is some disappointment that this false rejection rate isn't lower. See Figure 17.

This application only uses one eye. With the use of two eyes it is expected that the results stated above will improve. Whilst there has not been time to do extensive testing of this the initial findings are very positive. Maintaining the threshold so that there are no false positives, the number of false negatives would drop to about 1%. Unfortunately there has not been time to try and optimize the parameters for this test. See Figure 17.

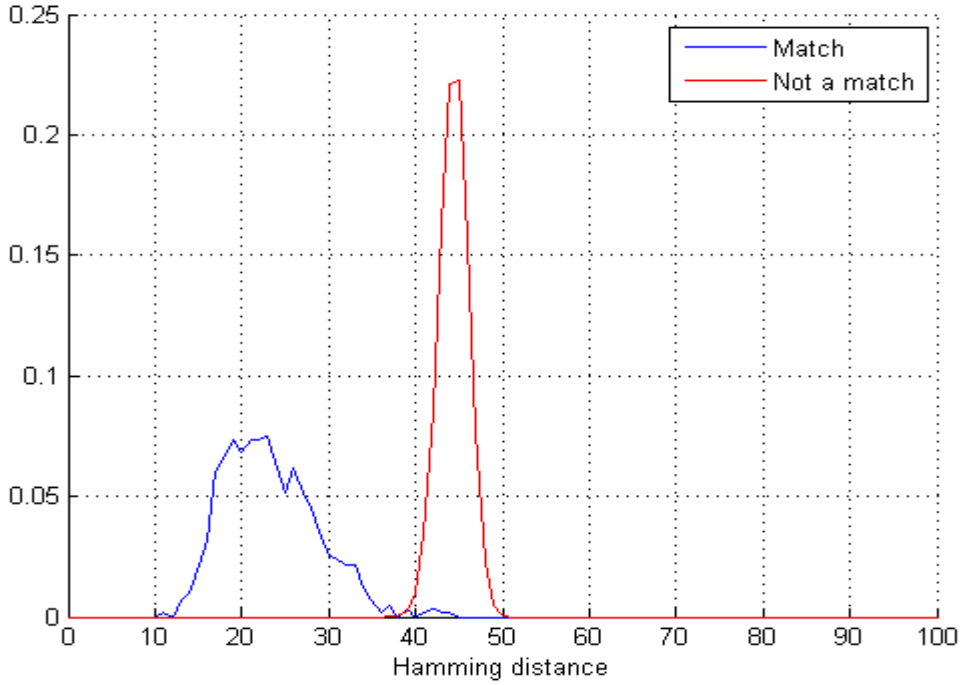
The other caveat mentioned above concerns the iris location routines. Whilst able to correctly identify and locate every pupil in the CASIA database, it correctly locates the outer edge of the iris about 80% of the time. There are a number of avenues available in attempting to improve this and they are covered elsewhere in this report. All subsequent tests data presented in this report use the subset of CASIA for which the iris location works correctly.

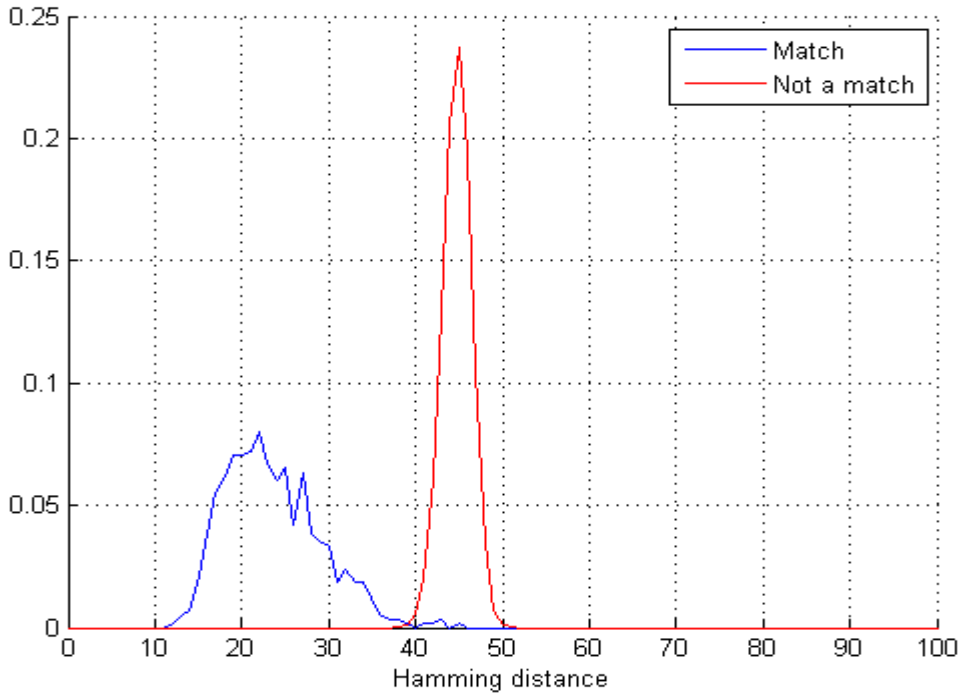
No attempt has been made to fully implement masking of eyelid or eye lash occlusions. The parameters of the Gabor wavelets have been chosen such that the most significant impact on the bitcode generated here is the part of the iris closest to the pupil. It is possible that when this is included the numbers would improve.

A great deal of work went into searching for better parameters, with some significant improvements from the starting point early in the project. Daugman is vague about his implementation in the papers we refer to. There are many possible parameters we could introduce to try and improve the false negatives. For example the convolution box here is always square – it would be worth trying another rectangle; the angle of the wavelets is parallel to the radius – other angles should be considered (perhaps changing each pass); and the code here always has three different sized boxes, perhaps two or four might be better.

Figure 17

The top graph shows the results for a single eye using a set of just under 600 images of about 100 eyes. The height of the graph is the proportion of the set at that point. 132 of 1544 image pairs that should show a match would be incorrectly rejected. There are about 180 000 correct rejections. The lower graph shows data draw from the same test set but treated as pairs of eyes. In this case only 8 of 599 matching tests would be rejected.





As an aside an attempt was made to avoid using the Gabor wavelets altogether. Instead the first and second derivative of the Gaussian was used. The method was to differentiate the Gaussian with respect to x (of the unwrapped iris, or the θ axis of the original). The first test was very hopeful, but on a larger set of data tests are slightly worse than currently achieved by using Gabor wavelets. However the results are much better than those achieved early in the search for better parameters before considerable effort (and CPU time) were expended. It would certainly be an interesting place for further research.

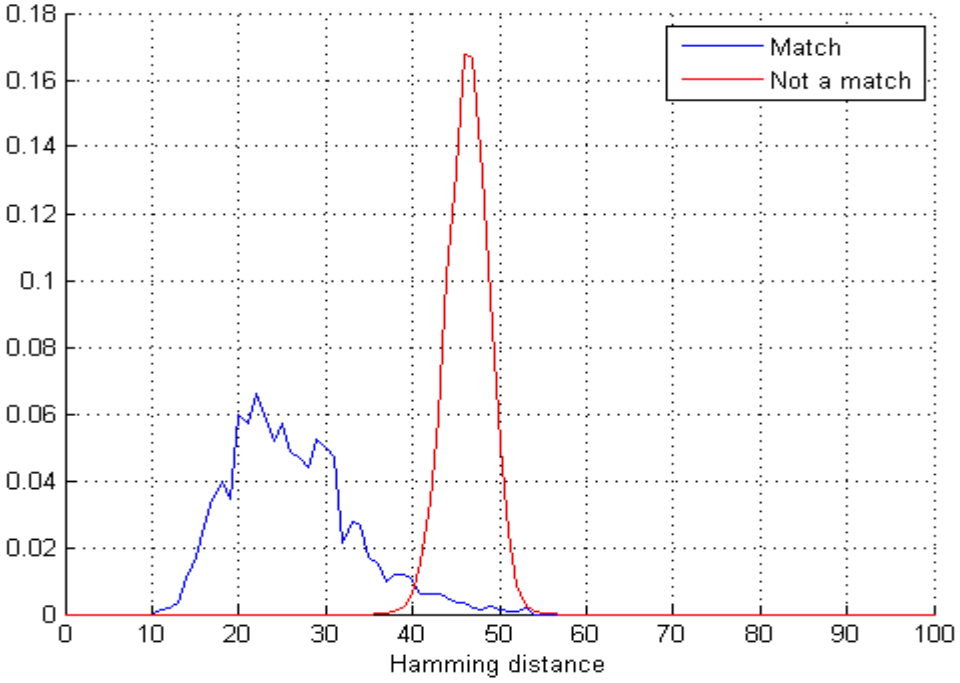


Figure 18

The best results obtained using a small iteration over a range of parameters. This is result is slightly worse (143vs132 incorrect rejections) than those using Gabor Wavelets, but this research is at a much earlier stage.

With much more still to consider the group feels it has made a good start in understanding both Iris Recognition and the complexities involved in engineering software in teams.

Bibliography

"High confidence visual recognition of persons by a test of statistical independence." . **Daugman, J. 1993.** s.l. : IEEE Transactions on Pattern Analysis and Machine Intelligence, 1993, Vols. vol. 15(11), pp. 1148-1161.

'How Iris Recognition Works'. **Daugman, J. 2004.** vol.14(1), s.l. : IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, 2004.

A 3x3 Isotropic Gradient Operator for Image Processing. **Sobel, I. and Feldman, G. 1968.** s.l. : presented at a talk at the Stanford Artificial Project ., 1968.

A Linear Algorithm for Incremental Digital Display of Circular Arcs. **Bresenham, J.E. 1977.** s.l. : Communications of the ACM, 20(2), 1977. Vols. February ,100-106.

Bresenham, J.E. (1965) "Algorithm for computer control of a digital plotter". IBM Systems Journal Vol 4(1).

Nelder, J.A. and Mead, R. 1965. s.l. : Computer Journal, 1965, Vols. vol.7, pp.308-313.

W.K Kong and D. Zhang (2001), "Accurate Iris Segmentation Based on Novel Reflection and Eyelash Detection Model," Proceedings of International Symposium on Intelligent Multimedia, Video and Speech Processing,Hong Kong.

Libor Masek, Rocognition of Human Iris Patterns for Biometric Identification, University of Western Australia, 2003.

Arvacheh, E.M.; Tizhoosh, H.R. (2006) 'IRIS Segmentation: Detecting Pupil, Limbus and Eyelids', Image Processing, 2006 IEEE International Conference.

9. Appendix

Appendix A: User Guide

(see next page)

Imperial College London

Iris Recognition ©

USER GUIDE

'Iris Recognition v1.0' was designed and developed at Imperial College, London during the Spring term 2009, by:

Arnar Jonsson
Edmund Noon
Seb Smith
Andrew Durnin
Mark Howe

Based on John Daugman's algorithms and written in Java using the Eclipse development tool.

Contents

Introduction

Thank you for purchasing 'Iris Recognition v1.0'. We hope that you enjoy using the software and that this user guide makes the program easy to use. In the unlikely even that you encounter any problems whilst using the application, please contact the Iris Recognition team at Imperial College, London at the following address:

Iris Recognition Team

Department of Computing

180 Queen's Gate

South Kensington Campus

Imperial College London

London SW7 2AZ

'Iris Recognition' uses the latest biometric identification technology to give you and your business the ultimate peace of mind. Based on the research of John Daugman (the forefather of iris recognition), we provide a robust solution to your security requirements.

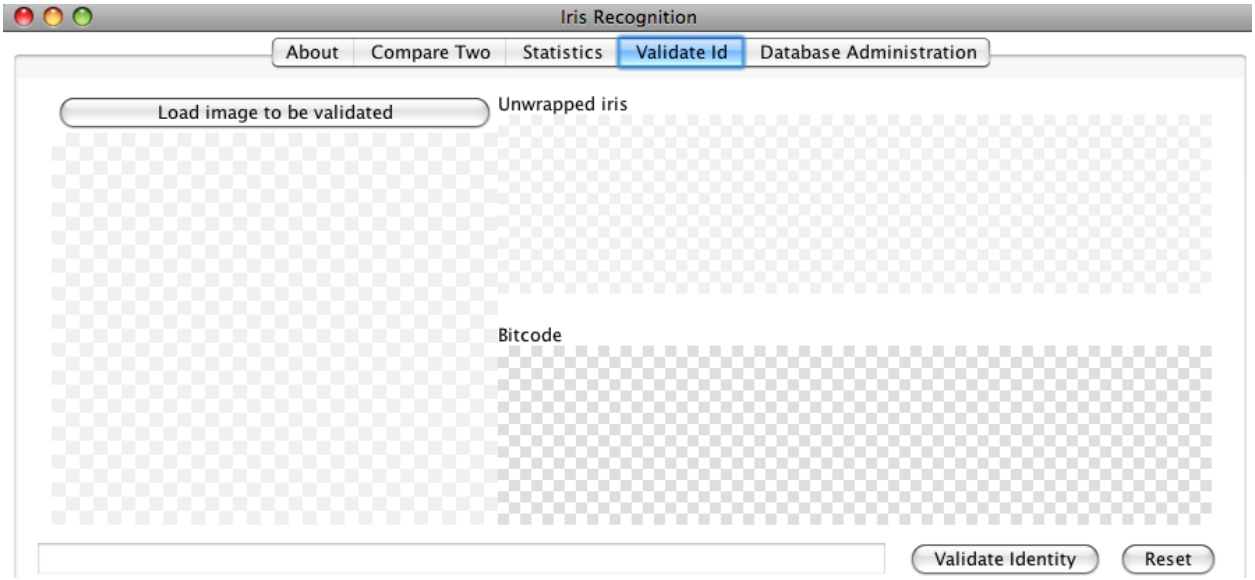
Getting Started

On opening the program you will notice 5 tabs at the top of the screen; About, Compare Two, Statistics, Validate ID and Database Administration. In the next section, we will explain how to use the functionality contained in each of these tabs in full detail.

Validating Identification

The 'Validate ID' tab allows the user to compare an uploaded image of an iris to the records stored on the database, thus determining whether or not access has previously been granted for that person. Figure 1.0 below shows the tab as you will see it on the screen.

Figure 1.0 – The 'Validate ID' tab

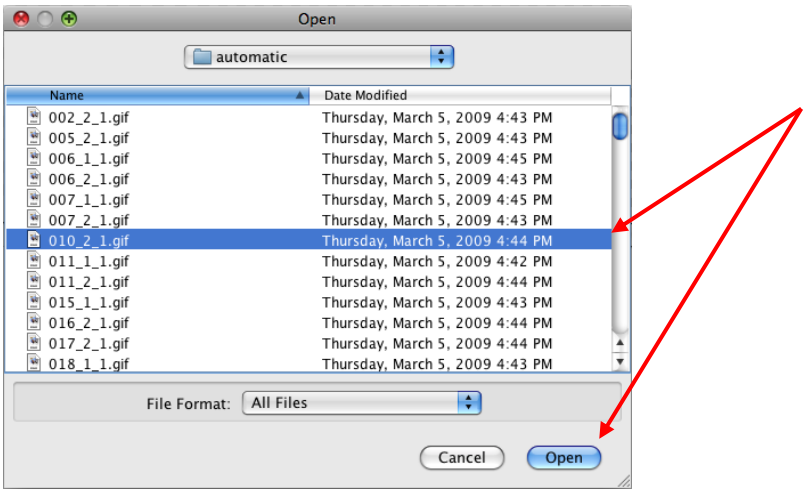


Loading an Image

To get started, you must first load an image that you want to validate. Click on the 'Load image to be validated' button and locate the image that you want in the file browser window (see figure 1.1 below).

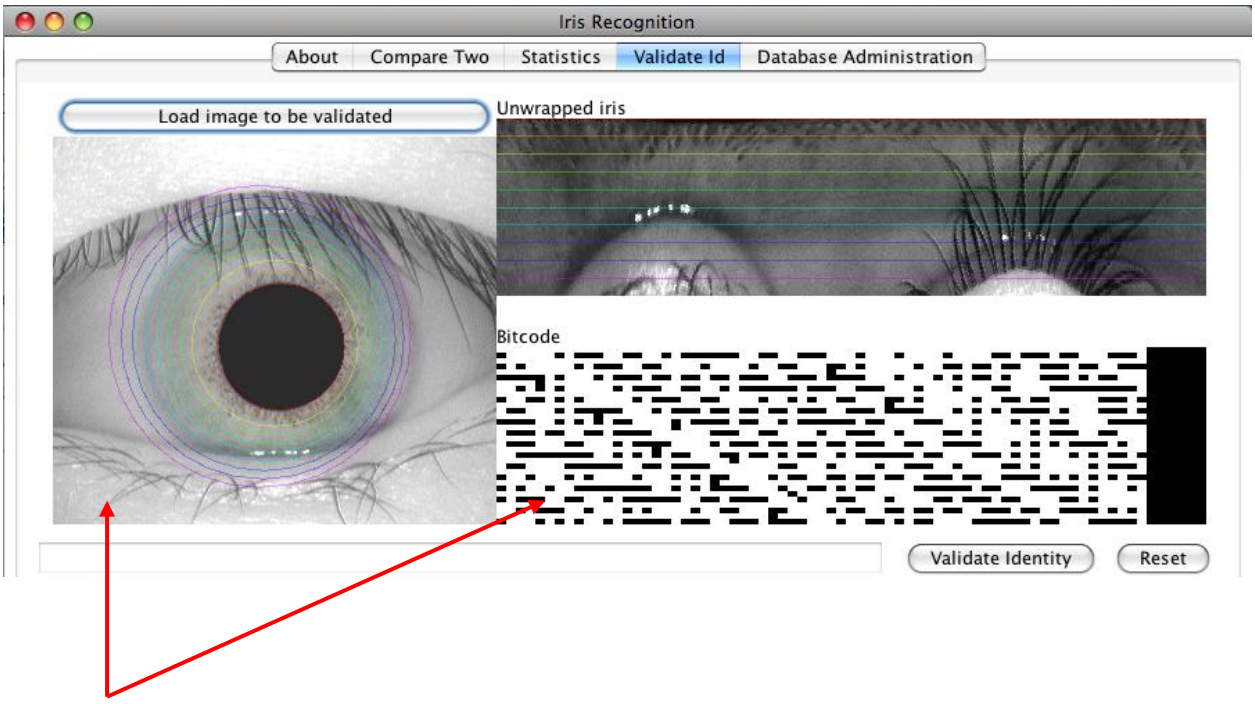
This will load your selected file into the program. The image will be displayed on the left of the screen, with the 'unwrapped' version and Bitcode for that image on the right of the screen (see figure 1.3 below).

Figure 1.1 – The file browser window



Select the file you wish to validate and click on the 'Open' button.

Figure 1.2 – The chosen image displayed on the screen



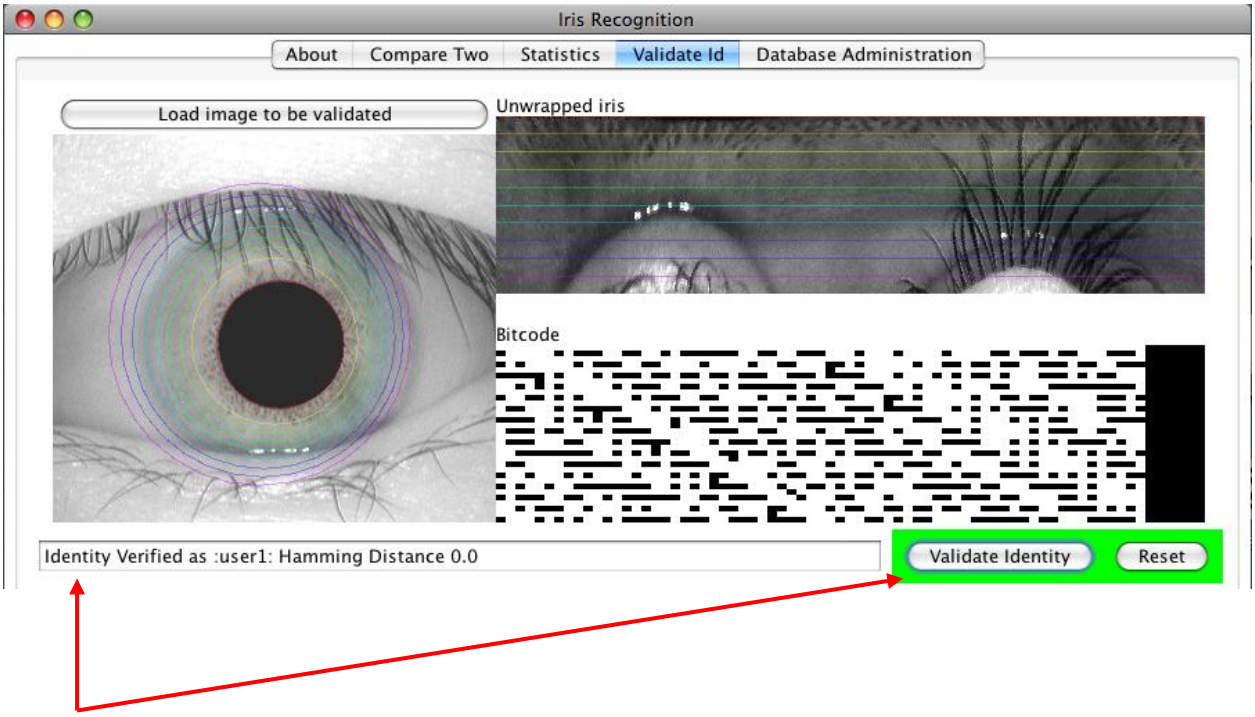
The loaded image will be displayed on the left of the screen, with the 'Unwrapped iris' and Bitcode displayed on the left.

Validating Against Database

With an image loaded, it is then possible to validate that person's identity against the records stored in the database.

Click the 'Validate Identity' button on the bottom right of the screen. The results will be shown within a few seconds (see figure 1.3 below).

Figure 1.3 – Validating Identity



Having clicked on the 'Validate Identity' button, the validation results will be shown. If the identity has been matched to a record in the database, the area around the button will turn green and the identity of that person will be displayed in the dialogue box at the bottom left of the screen. If a match is not found, the area around the 'Validate Identity' button will turn red. If that ID has been suspended, the graphic will turn orange.

Re-Setting

Once an image has been loaded, it is possible to 'reset' the screen in order to load/validate other images. To do this, simply click on the 'Reset' button at the bottom right of the screen. This will clear any images that are currently loaded and will allow the user to load another image.

Comparing Two Images

This tab allows the user to compare two images and to determine whether or not the irises belong to the same person. The tab is shown in figure 2.0 below, as it would be seen on screen.

Figure 2.0 – The Compare Two tab

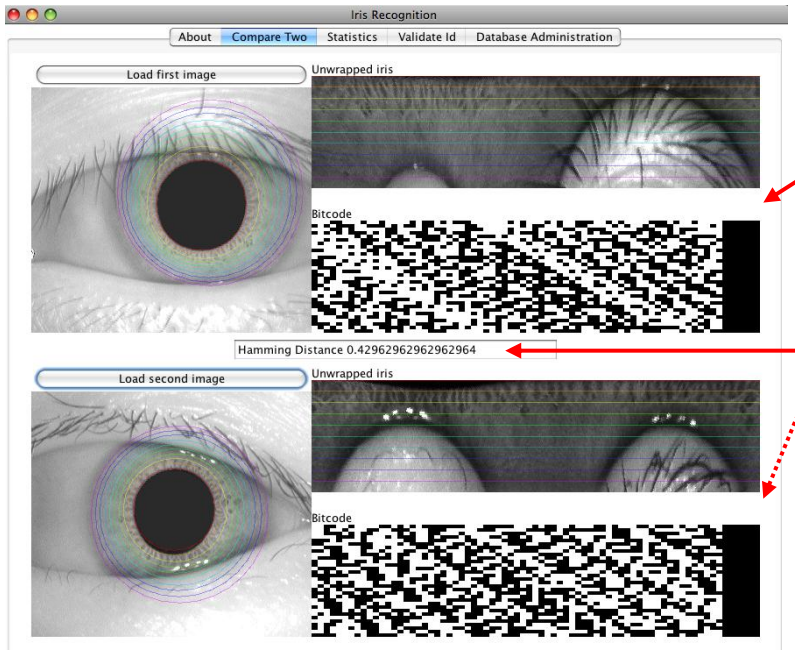
How to Load Two Images

Images are uploaded in this tab in much the same way as they are in the 'Validate ID' tab. Here there are two 'image loading' buttons, 'Load first image' and 'Load second image'. Clicking each of these will bring up the file browser window as before. Select an image in each of the browsers.

Comparing Images

Once the two images you wish to compare have been loaded, you will see a screen similar to the one shown below in figure 2.2.

Figure 2.2 – Comparing Images



The two images loaded into the program

The 'hamming result', as discussed in the 'Interpreting the results' section below

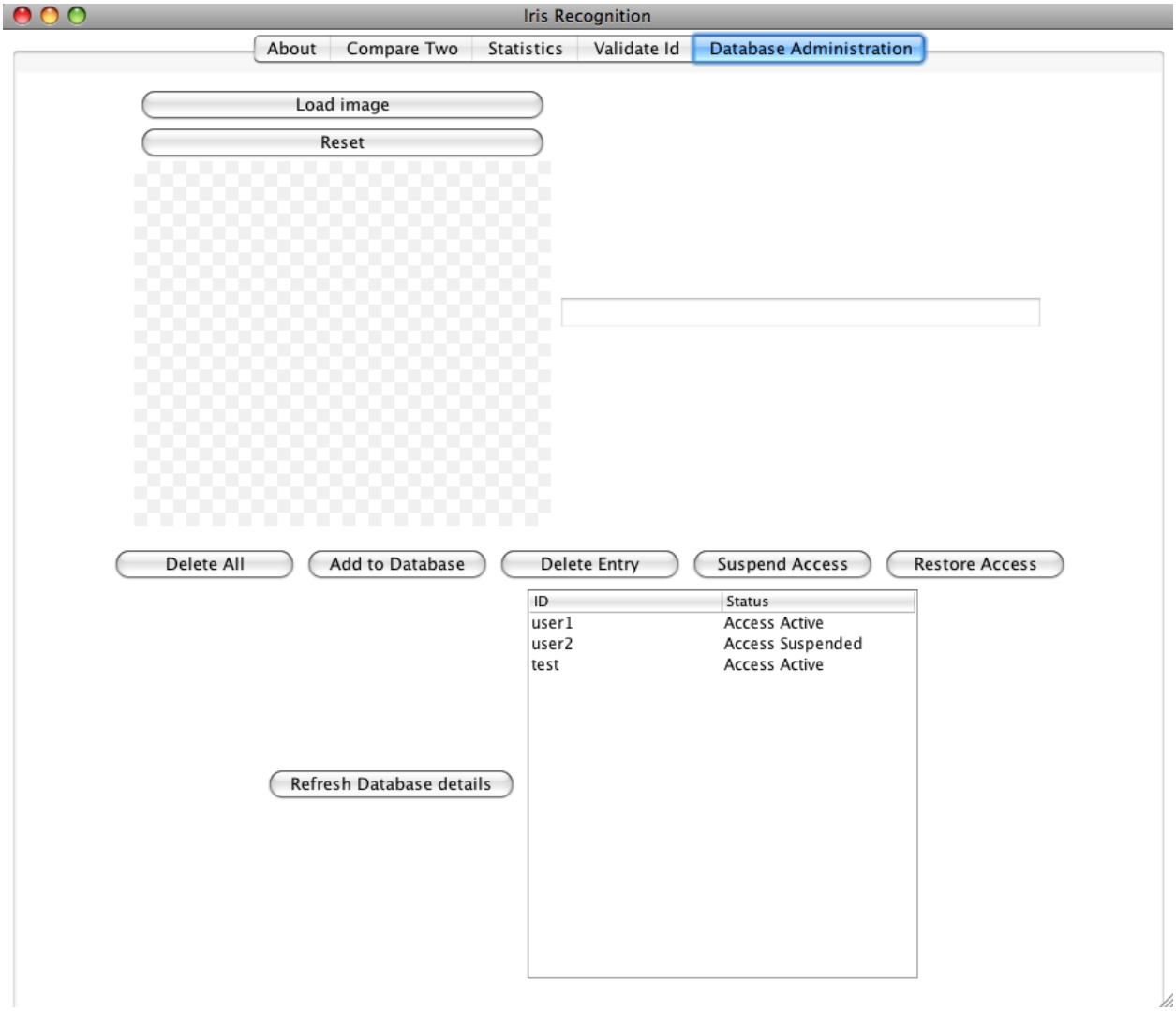
Interpreting the Results

Located between the two loaded images (as shown above) is the 'Hamming Distance' result. This is the result of mathematical calculation that determines how similar the two images are. Research has shown that a result that is less than 0.35 is almost 100% accurate in determining an identity match between the two images. As this number rises, there is less and less of a chance that the two iris images are from the same person.

Database Administration

This tab is designed for the Database Administrator so that he or she can directly view and manipulate the data stored in the database. From here, the user can add an entry to the database, delete an entry from the database, suspend access to the database for a particular person and delete all entries from the database. The tab looks as follows (see figure 3.0).

Figure 3.0 – The Database Administration tab

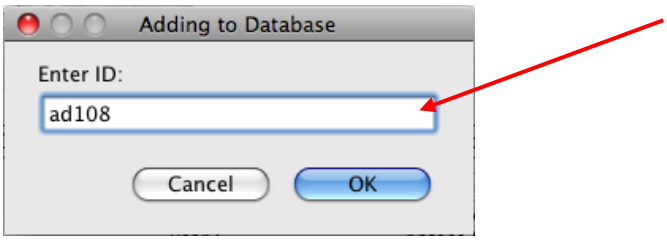


Adding an ID to the Database

In order to add a record to the database, you must first load the required image into the program. This is done in the same way as before, by clicking the 'Load image' button and selecting the required file from the file browser window.

Once the image has been loaded, click on the 'Add to Database' button. This will bring up the window shown below (figure 3.1).

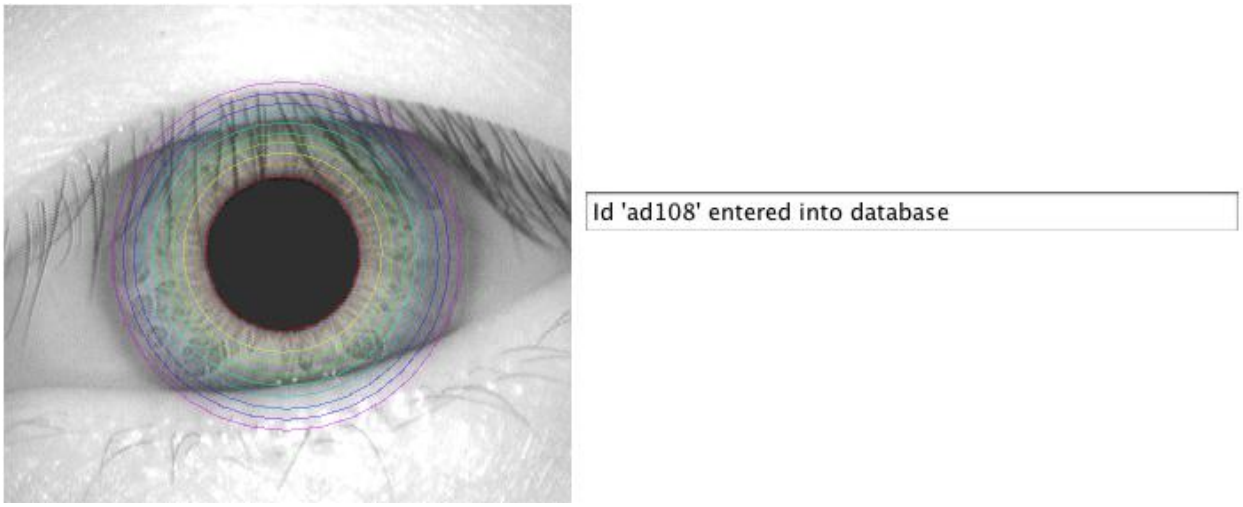
Figure 3.1 – Entering an ID



Enter the ID for the image that is to be added to the database.

Clicking 'OK' commits the action and adds the new ID and details to the database. The following notification is displayed in the dialogue box (figure 3.2).

Figure 3.2 – ID successfully added to the database

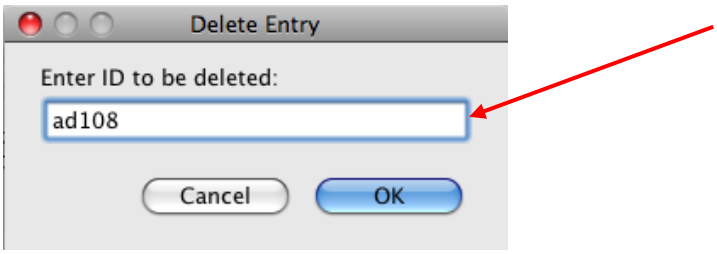


The dialogue box will alert you if there has been a problem in adding the details to the database.

Deleting an Entry from the Database

Clicking on the 'Delete Entry' button prompts you to enter an ID that they wish to be removed from the database (see figure 3.3 below).

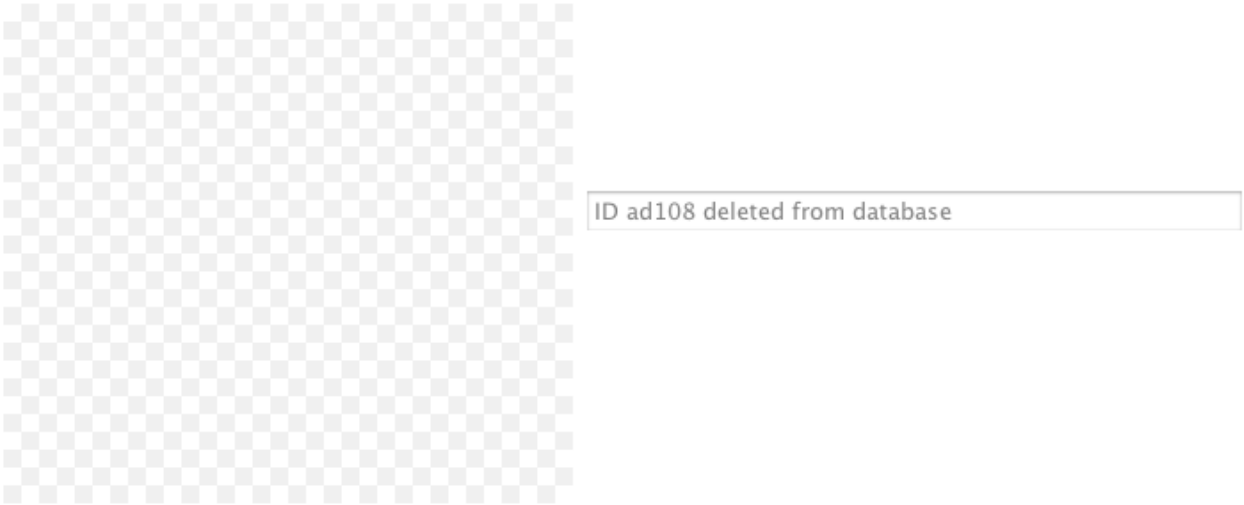
Figure 3.3 – Deleting an Entry from the Database



Enter the ID for the image that is to be deleted from the database.

Clicking on the 'OK' button commits the action to the database and the details for the ID entered are deleted. As with 'adding and entry', you will be notified of the actions success (or otherwise).

Figure 3.4 – Deleting and Entry Confirmation



There is a 'Delete All' button on the same tab. Clicking this will remove all entries from the database. You will be prompted to confirm that you really wish to commit to this action before it is carried out.

Suspending Access to the Database

There is a facility in the Administrator tab that allows you to 'suspend access' for particular entries in the database. This function could be used in instances where an ID should temporarily *not* be granted access by the program. By doing this, the entry is marked as being 'suspended', it is not actually deleted from the database.

Figure 3.5 – Suspending Access

Enter the ID for the image that is to be deleted from the database here.

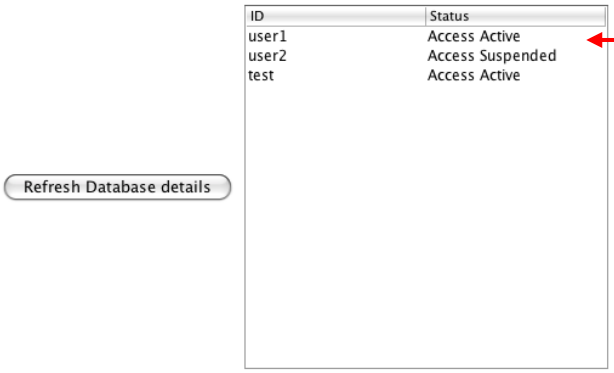
Clicking 'OK' commits the action and the result is displayed as before. Restoring access is done by clicking 'Restore Access' and entering the ID to be restored.

Additional Features

As with the 'Validate ID' tab, there is a button that allows you to 'reset' the currently loaded image.

There is also a small database viewing window that will show the current state of database at any given time. Clicking the 'Refresh Database details' button will ensure that you are viewing up to date details (see figure 3.6 below).

Figure 3.6 – Database Viewing Window

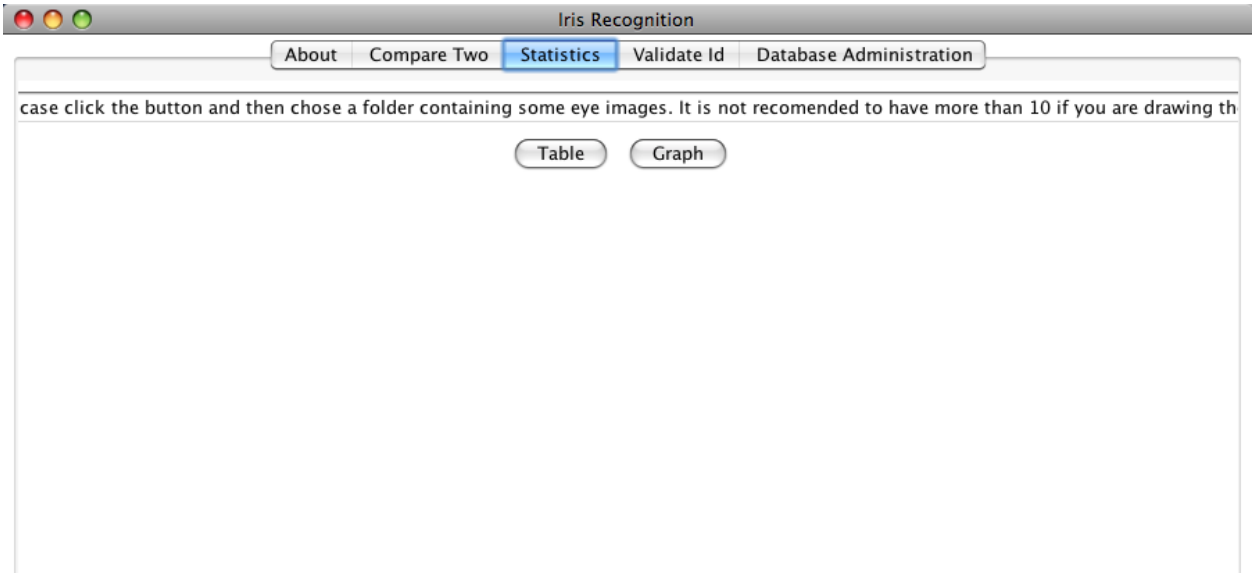


The Database Details window shows the current state of the database with the ID displayed on the left and the ID status show on the right. Clicking on the 'Refresh Database details' button ensures that this display is up to date.

Statistics

The statistics tab displays performance information to the user. The tab looks as follows (figure 4.0).

Figure 4.0 – The Statistics Tab



Appendix B: Code Optimization

Original code

```
public BitCode getBitcode(BufferedImage eyeImage, int xPup, int yPup,
int rPup, int xIris, int yIris, int rIris)
      {
            xp = xPup; yp = yPup; rp = rPup;
            xi = xIris; yi = yIris; ri = rIris;
            uw = new UnWrapper();
            bitcode = new BitCode();

            // array of intensity values (used rather than BufferedImage
for speed)
            intensityArr = uw.unWrapByteArr(eyeImage, xp, yp, rp, xi,
            yi, ri, unwrHeight, unwrWidth,(int) (2.0 * abPar.upLim
            +1.0));
            //unWrapped = uw.unWrapWithGuides(eyeImage, xp, yp, rp, xi,
yi, ri, unwrHeight, unwrWidth);


            for (int x0I=0; x0I < x0Par.numVals-1; x0I++) {
                  for (int i=0; i < bitcodeShiftNum; i++ )  {
                        // set parameters
                        a = abPar.get_StepN(i);
                        b = abPar.get_StepN(i);
                        w = wPar.get_StepN(i);
                        //w = wPar.lowLim/(2.0*a);
                        x0 = x0Par.get_StepN(x0I);
                        y0 = y0Par.get_StepN(i);

                        // apply filter for given set of parameters
                        this.gaborFilter2D();
                  }
            }
            bitcode.setShiftNum(bitcodeShiftNum*bitsPerBox);
            return bitcode;

      }


private void gaborFilter2D()
      {
            a2 = a*a;
            b2 = b*b;
            double k,imPart,rePart,tmpVal, imgVal;
            double sumRe = 0;
            double sumIm = 0;
            double xmin, xmax,ymin,ymax;
            xmin = Math.floor(x0-a);
            xmax = Math.ceil(x0+a);
            ymin = Math.floor(y0-b);
            ymax = Math.ceil(y0+b);
            //System.out.println("x "+xmin+" to "+xmax+"  y "+ymin+" to
"+ymax);
            //double lambda =2.0;
            //w = lambda /(a*2);
            for(double x = xmin; x <= xmax; x++)
```

```
            {
                  for(double y =ymin; y <=ymax; y++)
                  {

                        //imgVal = (double)intensityArr[(int) x][(int)
y];

                        //Color c = new Color(unWrapped.getRGB((int)x,
(int)y));

                        //imgVal = (c.getRed() + c.getGreen() +
c.getBlue())/3;
                        imgVal = intensityArr[(int)x][(int)y];
                        //e^(-pi((x-x0)^2/a^2 + (y-y0)^2/b^2)
                        k = Math.exp( -Math.PI * (Math.pow( x - x0, 2) /
a2 + Math.pow( y - y0, 2) / b2) );
                        //sin(-2*pi*w(x-x0 + y-y0))
                        tmpVal =  -w * 2 * Math.PI * ( x-x0 );

                        imPart = Math.sin( tmpVal );
                        //cos(-2*pi*w(x-x0 + y-y0))
                        rePart = Math.cos( tmpVal);// * wPar.upLim);

                        sumRe +=  (double) imgVal * k * rePart;
                        sumIm +=  (double) imgVal * k * imPart;
                  //System.out.println(rePart+" "+k*rePart);//+imPart);
                  }
            }

            if (bitsPerBox==2) bitcode.addBit(sumRe >= 0.0);
            bitcode.addBit(sumIm >= 0.0);
            //System.out.println(sumRe+" "+sumIm);
      }
```

Faster code

```
public BitCode getFastBitcode(BufferedImage eyeImage,EyeDataType eye)
      {
            xp = eye.inner.x; yp = eye.inner.y; rp = eye.inner.radius;
            xi = eye.outer.x; yi = eye.outer.y; ri = eye.outer.radius;
            uw = new UnWrapper();
            bitcode = new BitCode();

            // array of intensity values (used rather than BufferedImage
for speed)
            intensityArr = uw.unWrapByteArr(eyeImage, xp, yp, rp, xi,
yi, ri, unwrHeight, unwrWidth,(int) (2.0 * abPar.upLim +1.0));
            int imgVal;
            int[] ab = new int[bitcodeShiftNum];
            for(int i =0; i< bitcodeShiftNum;i++) ab[i] = (int)
abPar.get_StepN(i);
            int ab_max = (int) abPar.upLim;
            int sumRe,sumIm;
            for (int x0 =ab_max; x0 < unwrWidth+ab_max; x0++)
            {
                  for (int i=0; i < bitcodeShiftNum; i++ )
```

```
                  {
                        sumRe = 0;
                        sumIm = 0;
                        for(int x = -ab[i]; x <= ab[i]; x++)
                        {
                              for(int y =-ab[i]; y <=ab[i]; y++)
                              {
                                    imgVal = intensityArr[x0
+x][y+ab[i]];
                                    sumRe +=
imgVal*gaborReal[i][x+ab[i]][y+ab[i]];
                                    sumIm +=
imgVal*gaborImaginary[i][x+ab[i]][y+ab[i]];
                              }
                        }
                        if (bitsPerBox==2) bitcode.addBit(sumRe >= 0);
                        bitcode.addBit(sumIm >= 0);
                  }
            }
            bitcode.setShiftNum(bitcodeShiftNum*bitsPerBox);
            return bitcode;
      }


      public void initialiseParams(GaborParameters _wPar,GaborParameters
_abPar, GaborParameters _x0Par, GaborParameters _y0Par, int _unwrWidth,
int _unwrHeight,int _bitsPerBox)
      {
            unwrWidth = _unwrWidth;
            unwrHeight = _unwrHeight;
            bitcodeShiftNum = 3;
            wPar = _wPar;
            abPar = _abPar;
            x0Par = _x0Par;
            y0Par = _y0Par;
            if (_bitsPerBox==1 || _bitsPerBox==2) bitsPerBox =
_bitsPerBox;
            else bitsPerBox =2;



            gaborReal = new
int[bitcodeShiftNum][(int)abPar.upLim*2+1][(int)abPar.upLim*2+1];
            gaborImaginary= new
int[bitcodeShiftNum][(int)abPar.upLim*2+1][(int)abPar.upLim*2+1];
            double k,tmpVal;
            for(int series=0;series<bitcodeShiftNum;series++)
            {
                  int ab= (int) abPar.get_StepN(series);
                  double lw = wPar.get_StepN(series);
                  int ab2 = ab*ab;
                  for(int x = -ab; x <= ab; x++)
                        for(int y =-ab; y <=ab; y++)
                        {

                        //e^(-pi((x-x0)^2/a^2 + (y-y0)^2/b^2)
                        k = Math.exp( -Math.PI * (Math.pow( x , 2) / ab2
+ Math.pow( y , 2) / ab2) );
                        //sin(-2*pi*w(x-x0 + y-y0))
                        tmpVal =  -lw * 2 * Math.PI * ( x );
```

```
                        //cos(-2*pi*w(x-x0 + y-y0))
                        gaborReal[series][x+ab][y+ab] = (int)(65536.0 *
k * Math.cos( tmpVal));// * wPar.upLim);
                        gaborImaginary[series][x+ab][y+ab] =
(int)(65536.0 * k * Math.sin( tmpVal));
                    //these are 65536 times too big, but we only care
about the sign!
                        }
            }
```

Appendix C:

Database Functionality

```
try {
          conn = DriverManager.getConnection
          ("jdbc:postgresql://localhost:1432","g08v36205_u","6IxtbnTGoI"
          );
                stmt =
conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                        ResultSet.CONCUR_READ_ONLY);
              } catch (Exception e) {
                    System.err.println("Exception: " + e + "\n" +
e.getMessage() );
              }
```

Bitcode Conversion

```
public static byte[] toByteArray(BitCode bitcode){

                        byte[] result = new byte[bitcode.length()/8+1];
                        for(int i=0; i<bitcode.length(); i++){
                              if  (bitcode.get(i))
                                    result[result.length-i/8-1] |=
1<<(i%8);

                        }

                        return result;

                  }


public static BitCode toBitCode(byte[] stored){

                        BitCode bitcode = new BitCode(2048);

                        for(int i=0; i<stored.length*8; i++){

                        if((stored[stored.length-i/8-1]&(1<<(i%8))) > 0)
                                    bitcode.addBit(1);
                              else bitcode.addBit(0);
```

Appendix D:

JavaDocs – Class:BitCode Generator

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS
SUMMARY: NESTED | FIELD | CONSTR | METHOD

FRAMES NO FRAMES All Classes
DETAIL: FIELD | CONSTR | METHOD

iris.imageToBitcode
Class BitcodeGenerator

java.lang.Object
└ **iris.imageToBitcode.BitcodeGenerator**

public class **BitcodeGenerator**
extends java.lang.Object

A class that uses Gabor filters to generate a bitcode from an image of an iris The basic idea is to create different Gabor wavelets, defined by the constants a,b,w,x0 and y0 and filter the unwrapped iris with those wavelets. The total number of bits returned i

Version:
1.0
Author:
Arnar B. Jonsson, Mark Howe, Edmund Noon

Constructor Summary

BitcodeGenerator()
BitcodeGenerator(GaborParameters _wPar, GaborParameters _abPar, GaborParameters _x0Par, GaborParameters _y0Par, int _unwrWidth, int _unwrHeight)
BitcodeGenerator(GaborParameters _wPar, GaborParameters _abPar, GaborParameters _x0Par, GaborParameters _y0Par, int _unwrWidth, int _unwrHeight, int bits)

Method Summary

java.awt.image.BufferedImage	drawWavelet(java.awt.image.BufferedImage eyeImage, int x0, int y0, int step) Only for testing and report purposes, and for visually seeing the Gabor Wavelet
BitCode	getBitcode(java.awt.image.BufferedImage eyeImage, EyeDataType eye)
BitCode	getBitcode(java.awt.image.BufferedImage eyeImage, int xPup, int yPup, int rPup, int xIris, int yIris, int rIris) Generate bitcode
BitCode	getFastBitcode(java.awt.image.BufferedImage eyeImage, EyeDataType eye) Does the same as getBitcode, but faster.Uses integer arithmetic
void	initialiseParams(GaborParameters _wPar, GaborParameters _abPar, GaborParameters _x0Par, GaborParameters _y0Par, int _unwrWidth, int _unwrHeight) Overloading without the need to specify the number of bits which will default to 2
void	initialiseParams(GaborParameters _wPar, GaborParameters _abPar, GaborParameters _x0Par, GaborParameters _y0Par, int _unwrWidth, int _unwrHeight, int _bitsPerBox)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

BitcodeGenerator

public **BitcodeGenerator**()

BitcodeGenerator

public **BitcodeGenerator**(GaborParameters _wPar,
GaborParameters _abPar,
GaborParameters _x0Par,
GaborParameters _y0Par,
int _unwrWidth,
int _unwrHeight)

Class LocateIris:

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

iris.imageToBitcode
Class LocateIris

java.lang.Object
  └ **iris.imageToBitcode.LocateIris**

public class **LocateIris**
extends java.lang.Object

This class locates the pupil and outer iris boundary. It is currently 80% successful on the set of images it has been tested with.

Author:
en108

Constructor Summary

LocateIris()

Method Summary

static java.awt.image.BufferedImage	**draw_part_circle**(java.awt.image.BufferedImage bi, int cenx, int ceny, int r, char octant, int colour)
static java.awt.image.BufferedImage	**edgeDetection**(java.awt.image.BufferedImage bi)
static CircleType	**find_circle**(java.awt.image.BufferedImage bi, int pixel_blur, char octant, iris.imageToBitcode.Bounds bounds)
static EyeDataType	**find_iris**(java.awt.image.BufferedImage bi)
static java.awt.image.BufferedImage	**gaussian_blur**(java.awt.image.BufferedImage bi, int pixels) Blurs an images using a standard convolution method with the width being 3 SD and pixels wide
static java.awt.image.BufferedImage	**houghx**(java.awt.image.BufferedImage bi)
static java.awt.image.BufferedImage	**houghy**(java.awt.image.BufferedImage bi)
static double	**loop_integral**(int[][] array_bi, int cenx, int ceny, int r, char octant)

Class: GaborParameters

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS
SUMMARY: NESTED | FIELD | CONSTR | METHOD

iris.imageToBitcode
Class GaborParameters

java.lang.Object
  └ iris.imageToBitcode.GaborParameters

public class GaborParameters
extends java.lang.Object

Field Summary

double	lowLim
int	numVals
double	step
double	upLim

Constructor Summary

GaborParameters()
GaborParameters(double _lowLim, double _upLim, int _numVals)

Method Summary

GaborParameters	add(GaborParameters gp, double scale)
double	get_StepN(int n)
void	set(double _lowLim, double _upLim, int _numVals)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

lowLim

public double lowLim

upLim

public double upLim

step

public double step

numVals

INDEX