





## **Abstract**

Iris Recognition is a rapidly expanding method of biometric authentication that uses pattern-recognition techniques on images of irides to uniquely identify an individual. Algorithms produced by Professor John Daugman [1] have proven to be increasingly accurate and reliable after over 200 billion comparisons [2]. The aim of this group project is to implement a working prototype of the techniques and methods used for iris recognition, and to test these methods on a database of irides provided by the Chinese Academy of Sciences' Institute of Automation (CASIA) [3], which consists of 756 images of irides from 108 individuals.



## Acknowledgements

The authors would like to take this opportunity to thank their supervisor, Professor Duncan Gillies, for his constant encouragement and regular feedback on a project that was enjoyable, challenging and intriguing from start to finish.

This report was typeset using L<sup>A</sup>T<sub>E</sub>X.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Specification</b>	<b>2</b>
2.1	Minimum Requirements . . . . .	2
2.2	Extensions . . . . .	2
<b>3</b>	<b>Methodology</b>	<b>3</b>
3.1	Iris Location . . . . .	3
3.2	Encoding the Iris . . . . .	9
3.3	Comparisons . . . . .	11
3.4	Graphical User Interface . . . . .	12
<b>4</b>	<b>Final Results</b>	<b>15</b>
<b>5</b>	<b>Conclusions</b>	<b>17</b>
5.1	Final Product . . . . .	17
5.2	Auto Detection . . . . .	17
5.3	Source Code . . . . .	18
<b>6</b>	<b>Future Work</b>	<b>19</b>
6.1	Otsu Thresholding for Iris/Sclera Separation . . . . .	19
6.2	Eyelashes . . . . .	20
6.3	Eyelids . . . . .	20
6.4	Application Format . . . . .	20
<b>A</b>	<b>Division of Work</b>	<b>22</b>
<b>B</b>	<b>Logbook</b>	<b>25</b>

# Chapter 1

## Introduction

Security and the authentication of individuals is necessary for many different areas of our lives, with most people having to authenticate their identity on a daily basis; examples include ATMs, secure access to buildings, and international travel. Biometric identification provides a valid alternative to traditional authentication mechanisms such as ID cards and passwords, whilst overcoming many of the shortfalls of these methods; it is possible to identify an individual based on “who they are” rather than “what they possess” or “what they remember” [4].

Iris recognition is a particular type of biometric system that can be used to reliably identify a person by analysing the patterns found in the iris. The iris is so reliable as a form of identification because of the uniqueness of its pattern. Although there is a genetic influence, particularly on the iris’ colour, the iris develops through folding of the tissue membrane and then degeneration (to create the pupil opening) which results in a random and unique iris [5].

In comparison to other visual recognition techniques, the iris has a great advantage in that there is huge variability of the pattern between individuals, meaning that large databases can be searched without finding any false matches [1]. This means that irides can be used to identify individuals rather than just confirm their given identity; a property that would be useful in a situation such as border control, where it might be important to not just show that an individual is not who they say they are but also to show exactly who they are.

The objective of this project was to produce a working prototype program that functions as an iris recognition tool using the algorithms described by Professor John Daugman [1] and other techniques in order to implement this in an accurate and useful way that is also user-friendly. Commercial iris recognition systems are available that implement similar algorithms to these; however, there does seem to be an absence of open source implementations. This is the hole that this project sets out to fill: providing a fast, usable program that can easily be extended.



# Chapter 2

## Specification

### 2.1 Minimum Requirements

At the outset of this project, some key tasks were identified that needed to be carried out to fulfil the aim of creating a working prototype of an iris recognition system. The first task was to read in an image of an eye and display this on screen. From this stage the iris then needs to be isolated from the rest of the image; to do this accurately the pupil, iris, and eyelids all need to be identified. This isolation was originally specified to be carried out manually by the user by clicking points on the image.

The next task for the program was to calculate features of the iris pattern using the algorithms described by Professor John Daugman[1]. The iris bitcodes that are obtained from this can then be used to compare against a database of other such bitcodes to find the identity of the individual.

### 2.2 Extensions

A number of extensions to these requirements were also proposed in order to increase the effectiveness of the application if time permitted. The most desirable of these was the implementation of a robust method for the automatic detection and isolation of the iris; this would have the effect of reducing the need for user input and hence reduce the margin for error resulting from this to potentially improve the application's ability to successfully identify an individual.

Other features that would also potentially improve the match rate include locating and appropriately dealing with specular highlights in the image that can obviously alter the iris pattern that is visible in the image. This would allow the application to mask these areas of the image and so produce a more accurate encoding pattern. Along the lines of increased efficiency in the application, it was noted that methods would need to be investigated to ensure that finding a bitcode match in increasingly large databases remains fast and does not become a bottleneck for the application. Other potential extensions include an investigation into other possible methods for iris recognition. In particular, analysing whether the Gaussian wave and its differential are sufficient for encoding a unique bitcode, so that the sinusoidal waves are no longer required.

# Chapter 3

## Methodology

To achieve automated iris recognition, there are three main tasks: first we must locate the iris in a given image. Secondly, it is necessary to encode the iris information into a format which is amenable to calculation and computation, for example a binary string. Finally, the data must be storable, to load and compare these encodings.

The decision was made to use C++ with the Qt framework for this project. C++ is extremely fast and flexible for the large amount of calculations that need to be performed for each iris. In the same vein, Qt is a comprehensive and well-established C++ toolkit which provides a large amount of classes for producing elegant user interfaces. In addition to this, it has several classes such as QPixmap and QImage for manipulating and loading images easily. Qt is also a cross-platform toolkit, so the application will run on Linux, Windows, and Mac desktops, adopting the operating system's native toolkit and giving it an integrated feel on all platforms.

### 3.1 Iris Location

When locating the iris there are two potential options. The software could require the user to select points on the image, which is both reliable and fairly accurate, however it is also time consuming and implausible for any real-world application. The other option is for the software to auto-detect the iris within the image. This process is computationally complex and introduces a source of error due to the inherent complexities of computer vision. However, as the software will then require less user interaction it is a major step towards producing a system which is suitable for real-world deployment, and thus became a priority for extending the program specification.

Locating the iris is not a trivial task since its intensity is close to that of the sclera<sup>1</sup> and is often obscured by eyelashes and eyelids. However the pupil, due to its regular size and uniform dark shade, is relatively easy to locate. The pupil and iris can be approximated as concentric and this provides a reliable entry point for autodetection.

---

<sup>1</sup>The white part of the eye.

### 3.1.1 Pupil

The pupil's intensity and location are fairly consistent in most images and so it lends itself well to auto-detection. Detection of the pupil can be carried out by: removing noise by applying a median blur, thresholding the image to obtain the pupil, performing edge detection to obtain the pupil boundary and then identifying circles.

A median filter is a kernel based, convolution filter which blurs an image by setting a pixel value to the median of itself with its neighbours. A naïve approach to applying a median filter would be to simply find the median for a given area around each pixel by sorting an array and finding the median to replace the current pixel with. This could be implemented with a sorting algorithm which would have runtime  $\mathcal{O}(n \log n)$  on average. However, since the software requires several median blurs on different images a more efficient solution is required.

For an improved algorithm we consult Perreault's paper [6], which describes an algorithm to create a median filter in linear time. The process (see algorithm 1) involves constructing individual column histograms and combining them to form histograms centred around a pixel, known as a kernel histogram.

The significant speed increase comes from the way in which the column histograms are updated and combined. For each pixel we remove an old column histogram from the kernel, shift a new column histogram down one pixel so it is centred on the required row and then add this new histogram to the kernel histogram. While this radically reduces the number of operations which need to be performed for each pixel, there is an initialisation step for each row which has runtime linear in the size of the kernel histogram  $\mathcal{O}(r)$ . This enabled the entire median filter to be applied in a matter of milliseconds.

---

**Algorithm 1** Median filtering algorithm as proposed in [6]

---

**Input:** Image  $X$  of size  $m \times n$ , kernel radius  $r$ .

**Output:** Image  $Y$  of size  $m \times n$ .

Initialise each column histogram  $h_0, \dots, h_{n-1}$  as if centred on row  $-1$ .

**for**  $i = 1$  to  $m$  **do**

    Shift the first  $r$  column histograms  $h_0, \dots, h_{r-1}$  down 1 pixel.

    Combine these  $r$  column histograms to form the kernel histogram  $H$ .

**for**  $j = 1$  to  $n$  **do**

        Set pixel  $Y_{i,j}$  equal to the median of  $H$ .

        Shift column histogram  $h_{j+r}$  down 1 pixel.

        Remove column histogram  $h_{j-r-1}$ .

        Add column histogram  $h_{j+r}$ .

**end for**

**end for**

---

The overall effect of the median blur is to reduce the noise and pixel intensity complexity of the iris image without perturbing the edge fidelity of the original image. This results in a stronger clustering of pixel values in the resultant pixel data histogram; this permits a largely noise-free, dynamic analysis of features which occupy discrete pixel ranges, such as the pupil (see figure 3.2).

Upon application of a dynamic threshold to our median blurred image the leftmost image in figure 3.1.1 is obtained. The location of the pupil is well defined, and edge

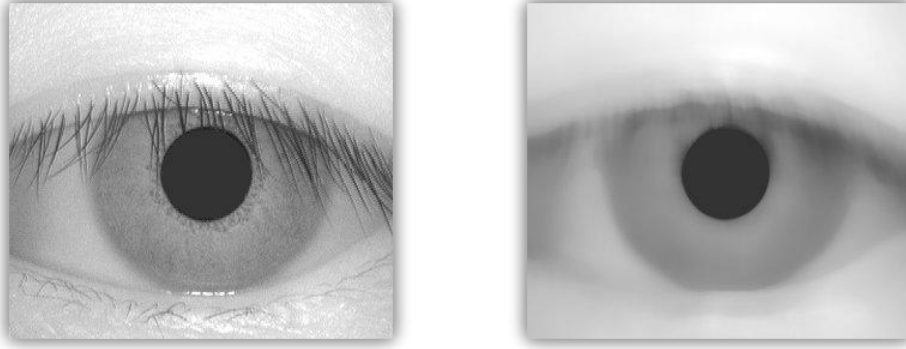


Figure 3.1: Image before and after a median filter.

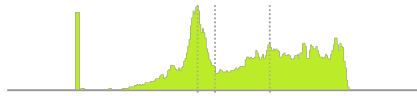


Figure 3.2: Histogram generated in the application following a median blur of a database image. Note the distinct, left-most (darkest pixel values) peak as a typical pupil's pixel signature on the image.

detection can extract this information to form a single bit depth image which can be searched for circles.

The Sobel filter is an edge detection technique which calculates the gradients (in both the  $x$  and  $y$  direction) of the image intensity for each pixel and then combines them to give a gradient magnitude. This indicates how sharp the boundary is, which is itself an indication of whether an edge is present.

As can be seen in 3.3 the result of this process of filtering is a circular artifact denoting the extremities of the pupil in the image. This data requires further analysis for the presence of circular artifacts to acquire a "best-fit" representation of the potentially elliptical pupil.

The Hough Transform is implemented to locate the pupil (and subsequently the iris). Given a regular curve – in this case a circle – and a suitable range of parameters –  $x$

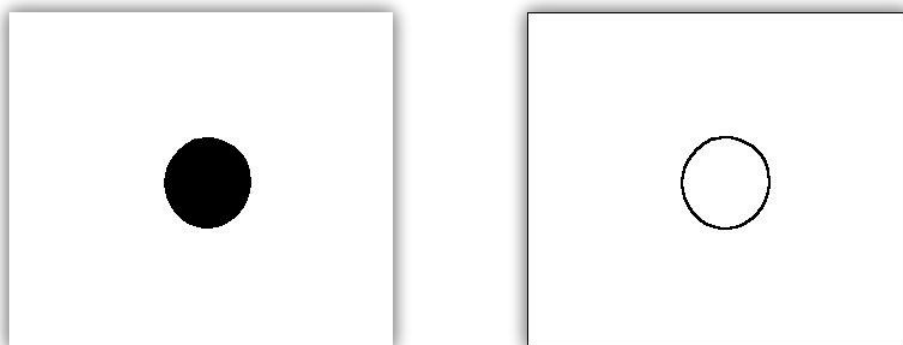


Figure 3.3: Thresholded image before and after edge detection with a 2D Sobel convolution filter.

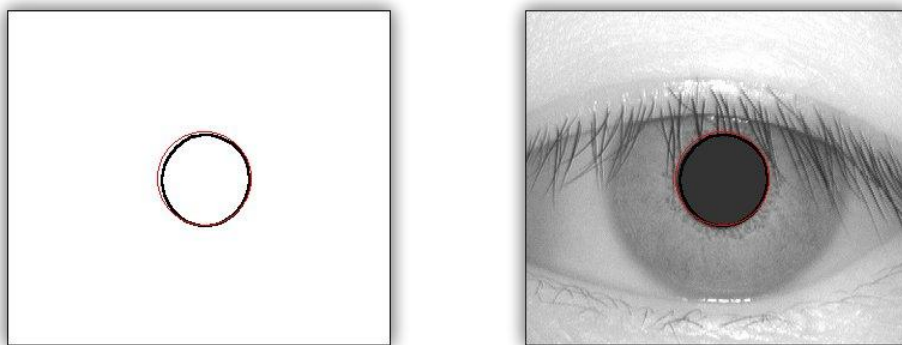


Figure 3.4: Hough transform solution for a circle (red), given edge detected pupil image.

centre,  $y$  centre and radius – the generalised Hough transform can query individual pixels in an image for their contribution to a globally consistent solution for any parameters within the valid range.

Due to the high computational complexity of the Hough transform technique (the parameter space for a simple circle is in three dimensions and depending on valid ranges this can quickly grow to millions of computations) and previous considerations for application performance, a number of elements were streamlined to accelerate the computation.

Thresholding the input image has the inherent benefit of changing the virtual bit-depth of the image to 1 (black or white). Furthermore, the implemented algorithm saves time by only operating on black pixels – a greyscale implementation would significantly add to the processing time. Additionally a search on the threshold result (figure 3.1.1) gives confident parameter estimations for centre and radius which can be used to constrain and guide the Hough algorithm.

As can be seen in figure 3.4, this process gives fast, accurate and reliable solutions for circular artefacts in the thresholded and edge detected pupil image. The basis of iris auto-detection relies on this process.

### 3.1.2 Iris

Once we have the location of the pupil clearly defined, the complexity of locating the iris is somewhat reduced due to the relative concentricity of the pupil and iris.

In contrast to pupil detection, efficiently locating the iris is somewhat more complicated due to (i) the obstruction of the iris by the eyelids for most eyes, (ii) its irregular pattern and (iii) because of the relative similarity with the iris boundary and the sclera.

Nevertheless, following a similar method to that of locating the pupil generally provides good results. Again, a median filter is applied to the original image in order to remove noise from the image and to strengthen the clustering in the pixel histogram. The resultant histogram is used to identify a point that can be used to threshold the image to produce an image similar to figure 3.5.

The method involves finding the peaks of the histogram (excluding the left-most peak that represents the pupil) and choosing a threshold value between these points. Although not always optimum, this method generally creates an image that the Hough Transform can use to find an approximate match for the iris; other, potentially better, methods for

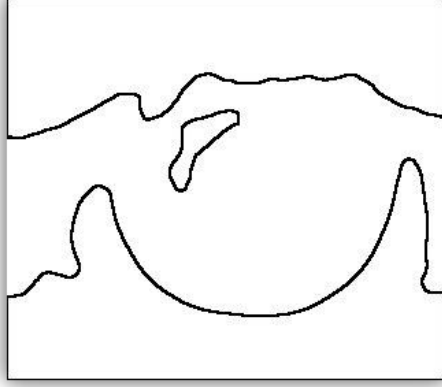


Figure 3.5: An example of an image that is obtained by intelligently thresholding the original input image and then using Sobel to edge detect. This image represents the data that is used to locate the iris; the data is used by the Hough Transform to decide on a best-fit circle to represent the location of the iris. A rough semi-circle can be made out by the human eye; this is what allows the Hough Transform to accurately locate the iris.

achieving this threshold are discussed in the future work section (6).

The Hough transform is very efficient for the task of finding the iris from an image such as this because it is resilient to noise and performs well even when a large amount of the circle is hidden. For example, when two eyelids are covering a large portion of the iris, such as in figure 3.6. This provides us with a circle that defines the iris, although portions of this are still obscured by the eyelids, meaning that these also need to be defined.

### 3.1.3 Eyelids

We note that an image of an eye roughly has three intensities, from darkest to lightest: pupil, iris and sclera, and eyelids. Hence a suitable thresholding method should be able to separate the eyelids from the rest of the image. Otsu gives a method in [7] for appropriately thresholding grey level images; the algorithm works as follows.

Assume we have an image with  $L$  possible intensities for each pixel and  $n_i$  is the number of pixels of intensity  $i$ . We first normalize the histogram into a probability distribution, that is  $p_i = n_i / \sum_{i=1}^L n_i$ . We assume that we threshold the image at level  $k$  into two classes,  $C_0$  containing levels less than or equal to  $k$  and  $C_1$  containing the rest. We then define

$$\omega(k) := \sum_{i=1}^k p_i \text{ and } \mu(k) := \sum_{i=1}^k i p_i,$$

which are the probability of a pixel picked at random being in  $C_0$  and the expected (mean) intensity for  $C_0$  respectively.

It is clear to see that that  $1 - \omega(k)$  is the probability of a pixel being in  $C_1$  and  $\mu(L)$  is the mean intensity of the entire image. We then calculate

$$\sigma^2 = \frac{[\mu(L)\omega(k) - \mu(k)]^2}{\omega(k)[1 - \omega(k)]},$$

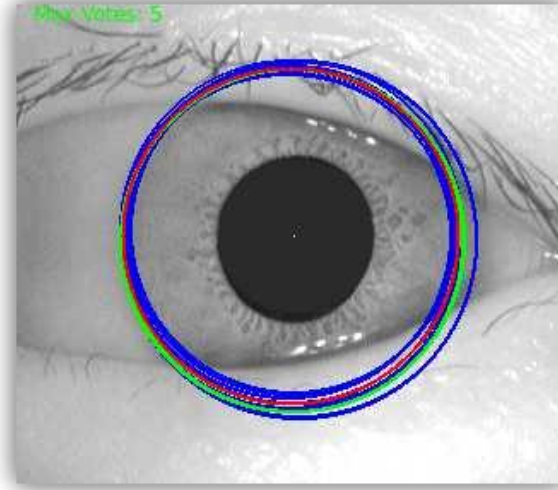


Figure 3.6: A demonstration of the results obtained by applying the Hough Transform to an image such as that in figure 3.5. The chosen circle is shown in red, which is an average of all of the circles with the maximum number of votes (in green). The blue circles represent those with the second most votes.

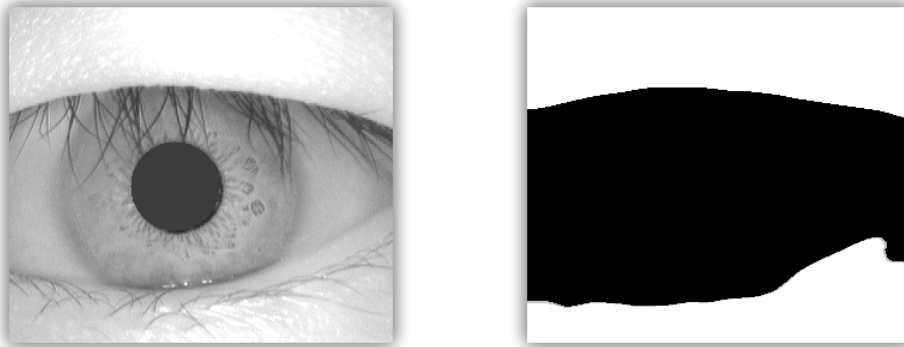


Figure 3.7: Image before and after a median blur and Otsu threshold.

which is shown to be measure of the variance between the two classes  $C_0$  and  $C_1$  by Otsu in [7]. We calculate this between-class variance for each  $k$  and finally threshold on the  $k$  where  $\sigma^2$  maximized.

To get any useful information from Otsu thresholding we must first ignore the pupil to ensure that the result does not only consist of two classes (the pupil and then everything else). Once we have ignored the pupil in class calculations we find that Otsu thresholding gives us two classes: one containing the iris, sclera and the pupil (since it has lowest intensity), and the other class containing the eyelids, see figure 3.7. With the iris located within the image we must now begin the task of extracting information from it.

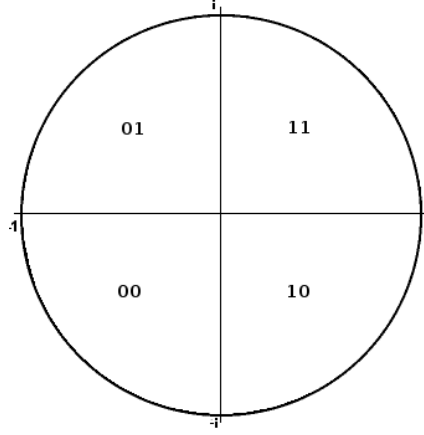


Figure 3.8: The four phase quadrants and the corresponding bit sequences they represent

## 3.2 Encoding the Iris

The iris is encoded to a unique set of 2048 bits which serve as the fundamental identification of that person's particular iris. These iris bit codes can be stored in a database and then compared to uniquely identify a person. The size of 2048 is sufficiently large to store the data of several particular filters on most angles of the iris, while also being sufficiently small to be easily stored in a database and manipulated quickly. We wish to extract phase information from the iris as opposed to amplitude information since phase information is not skewed by pupil deformation. We use Gabor filters to extract this phase information as suggested by Daugman [1].

The Gabor filter is an application of the Gabor wavelet (see equation 3.1) [1]. This will return two bits depending on which quadrant (see figure 3.8) the normalised resulting imaginary number from this lies in. This equation can be simplified for computation by considering it as a combination of two filters: one filter representing the real part of the integral and the other representing the imaginary part. Each of these filters consist of a combination of a Gaussian and a sinusoidal filter (see figure 3.9). The parameters  $\alpha$ ,  $\beta$  and  $\omega$  are then tuned by constraining the Gaussian filter to range over one standard deviation and the sinusoidal filter to range from  $-\pi$  to  $\pi$ .

$$h_{\{Re,Im\}} = sgn_{\{Re,Im\}} \int_{\rho} \int_{\phi} I(\rho, \phi) e^{-i\omega(\theta_0 - \phi)} e^{-(r_0 - \rho)^2 / \alpha^2} e^{-(\theta_0 - \phi)^2 / \beta^2} \rho d\rho d\phi \quad (3.1)$$

### 3.2.1 Filter Placement and Size

The Gabor filters need to be placed so that they take account for a large enough range of data without missing out the fine detail within the iris. As we require 2 bits to uniquely identify each quadrature, we have the additional constraint that 1024 filters must be used, allowing a manageable bit code length of 256B for each iris which can be split into sections of 4B. To simplify the calculations involved in applying the filters to the unwrapped iris they can also be considered to all be placed at a uniform rotation of 0. Since it can be observed that there is consistent amounts of information at points varying



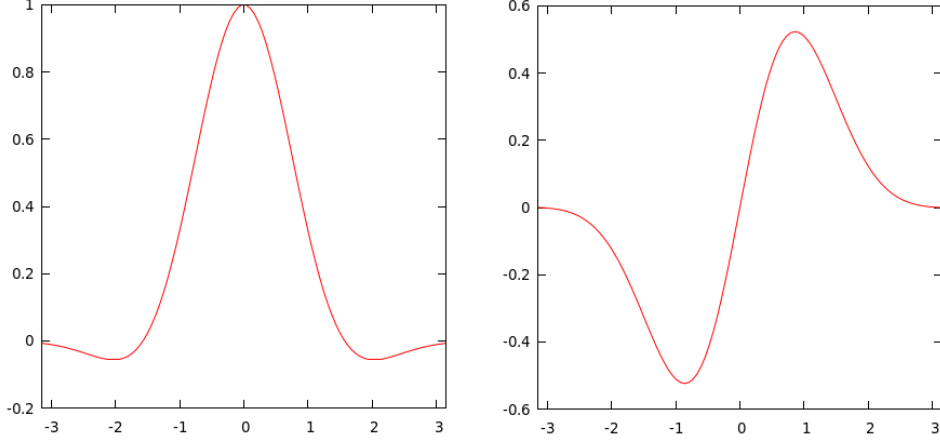


Figure 3.9: The real (left) and imaginary (right) parts of the Gabor wavelets

in the angular direction in the iris, it is also a good idea to place the filters uniformly in this direction. In this implementation 256 angular directions were considered because it divides our required number of filters evenly and includes a large spread of data across the iris. However, in many cases, in the radial direction there is more distinct information observed closer to the pupil.

To include the more important data close to the pupil the filters can be positioned so that a larger proportion of them have their centres in this range. With the 256 angular directions there are 4 radial filter locations left to position. These can be placed uniformly between the bottom of the iris and half-way across the iris to allow the data around the pupil to be explored. Finally, to make sure that filters are always large enough to represent enough data, the constraint that a filter centre cannot be placed on the 3 pixels nearest the top and bottom of the image is added. This means that useless filters of sizes 1 and 3 will never be included when encoding the iris, and we avoid including additional pupil debris that may have been incorporated into the iris section. The final set of filter locations across an unwrapped iris is represented in figure 3.10.

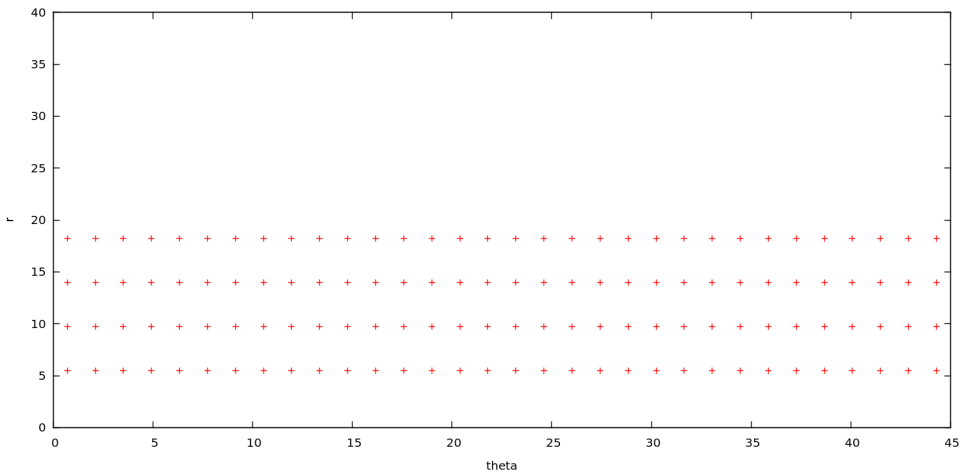


Figure 3.10: Filter positions for an iris of radius 40 between  $\theta=0$  and  $\theta=45$

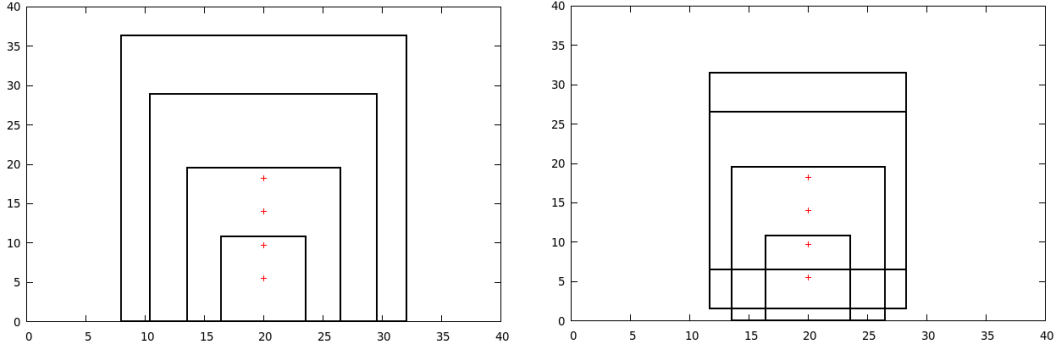


Figure 3.11: Two schemes for filter sizes and placements at a fixed point in the angular direction in an iris of radius 40. Left: maximum size filter placed. Right: filter sizes capped at 25

The final consideration with the filters is their size. Two schemes for doing this are demonstrated in figure 3.11. The first (left) aims to position the largest possible filter at every filter location. This means that the filters fulfil one of the original requirements; that the filters should capture as much information as they can. The issue which this scheme suffers from is that the filters with their centres in the middle of the iris can be so large that they aren't really able to represent some of the finer detail in the image. The second scheme (right) tackles this problem by capping all filters in the image at a maximum size. This maximum size can be tuned in the implementation to get more distinct bitcodes, which is why it is favourable and consequently the one we use.

### 3.3 Comparisons

Iris encodings can be compared by computing a Hamming distance between them. The Hamming distance between two iris codes is calculated as given in [1]. Assume we have two iris codes, *codeA* and *codeB*, with corresponding bit masks, *maskA* and *maskB* (where a 0 in the bit mask corresponds to a bad bit in that position in the iris code). Then we calculate the hamming distance *H* as:

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

Where  $\otimes$  is the bitwise XOR operator and  $\cap$  is the bitwise AND operator.

We can see that the numerator will be the number of differences between the mutually non-bad bits of *codeA* and *codeB* and that the denominator will be the number of mutually non-bad bits. However an astute reader may observe that this measure can fail if there is an exceedingly low number of mutually non-bad bits causing the numerator to be zero even though there may actually be large differences between *codeA* and *codeB*. However, this is a simple case to handle and will rarely occur when comparing iris images.

This comparison is performed for various shifts in the codes to check for rotation of the image, as it is very possible that a subject's eye in the image is rotated if the camera or their head is tilted slightly.

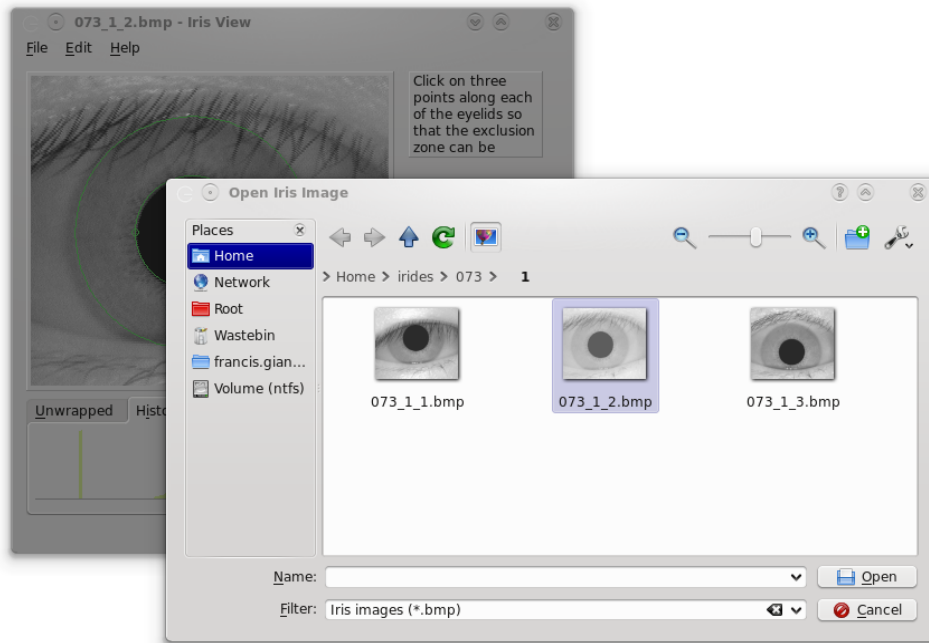


Figure 3.12: The application in Linux with the Open File view.

This statistical independence test is the key to iris recognition. It provides sufficiently many degrees-of-freedom to fail in every case where the iris bitcodes of two different eyes are compared, but uniquely passed when the same iris is compared with itself.

### 3.3.1 Iris Database

Since we are able to generate the code and the iris mask directly, these are the only two items that are required to be stored in a database, the image bitmaps themselves are no longer required.

We are currently using a basic text file as our database as this was simple to implement and for the small number of eyes we have as test subjects this is perfectly adequate and proves to be able to compare iris bitcodes extremely fast.

## 3.4 Graphical User Interface

Since the software was built in Qt, a cross-platform toolkit, the final product runs on Linux, Windows, and Mac desktops, adopting the operating system's native toolkit and giving it an integrated feel on all platforms.

Usability is an important consideration when creating an application, and throughout development the aim was to provide a clean and intuitive interface whilst still providing full usability and advanced features. The key intellectual challenges for the user interface have been primarily to provide a straight-forward and informative access point for the user, but also to provide relevant extra data when required with a fall-back mechanism for difficulties in auto-detection.

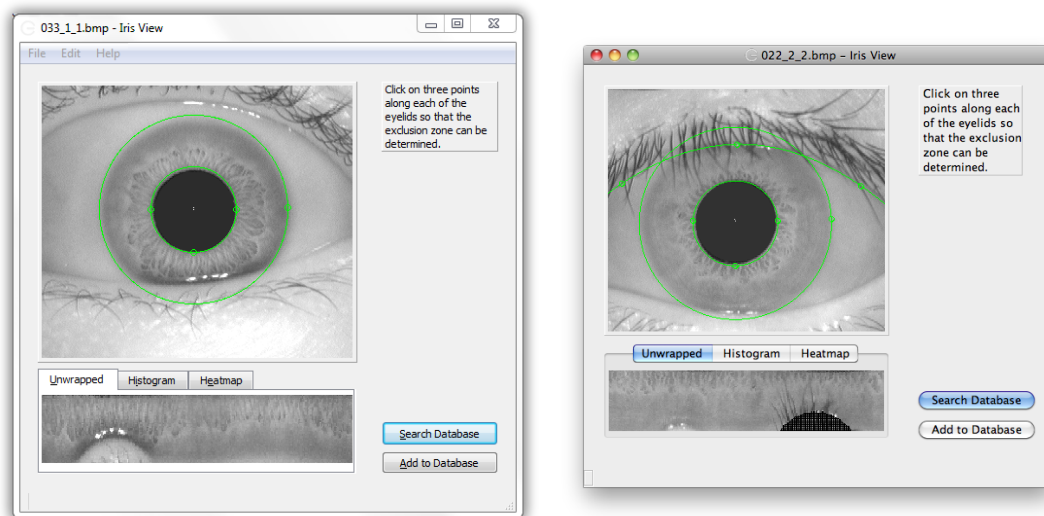


Figure 3.13: Windows and Mac OS X.

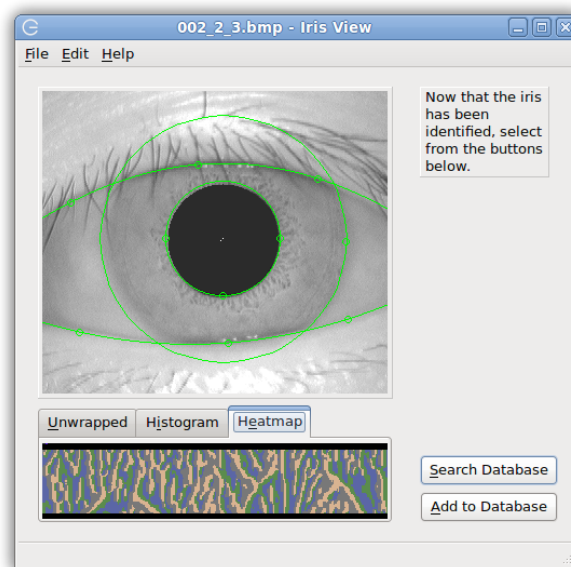


Figure 3.14: Application with heatmap tab.

The final product includes relevant data and feedback behaviour such as histogram information from the image to allow the advanced user to track-down and diagnose auto-detection failures. This information is placed into a tabbed interface below the primary image so that it is easily accessible without encroaching on our clean interface which allows the average user point-and-click refinement (if necessary) of the iris parameters with real-time visual feedback without overloading them with this more advanced information.

# Chapter 4

## Final Results

With the batch command line option of the application, it is possible to process a large amount of images of eyes. Using this option, 108 images of eyes from the CASIA database were loaded into the application; the pupil, iris and eyelids were auto-detected to generate a bitcode and store it in the database. These were then compared to an alternative image of each eye to check for matches.

To further test primarily for false matches, the first three images of each eye were loaded into the database, and then compared to a fourth image for each eye. As there were originally three images for each eye, and then a comparison was performed on all of the extra 108 images, roughly 35,000 iris comparisons took place  $((108 * 3) * 108)$ .

Our results of these tests are as follows:

- 0 false matches in  $\sim 35,000$  comparisons
- 70% (75 out of 108) match rate with Professor Daugman's suggested 0.32 Hamming distance.

The complete lack of any false matches and the incredibly high match rate are a testament to the robustness and feasibility of an iris recognition system. The tests have suggested that in virtually any case where the iris boundary is correctly located, the iris should be correctly identified.

As a roughly 70% match rate was achieved for the iris location, this percentage has, as predicted, been reflected in the overall match rate in the database of images. Similarly to Daugman, we can also observe that the Hamming distances produced by comparing different irides tends to follow a binomial distribution, with a mean around 0.45; see figure 4.1.

If time had permitted, it would have been beneficial to decrease the reliance on the batch mode for the testing phase and process the images manually, in order to ensure that each iris was located correctly in the image. This would enable a separation in our results of the success of the auto-detection and of the matching process.

Another appealing extension to the testing phase would have involved setting up a high resolution (and preferably infra-red) camera to appropriately test the application's ability to identify an individual outside of the CASIA database. This would also have had the benefit of being able to emulate a real-world use of the application far more accurately.

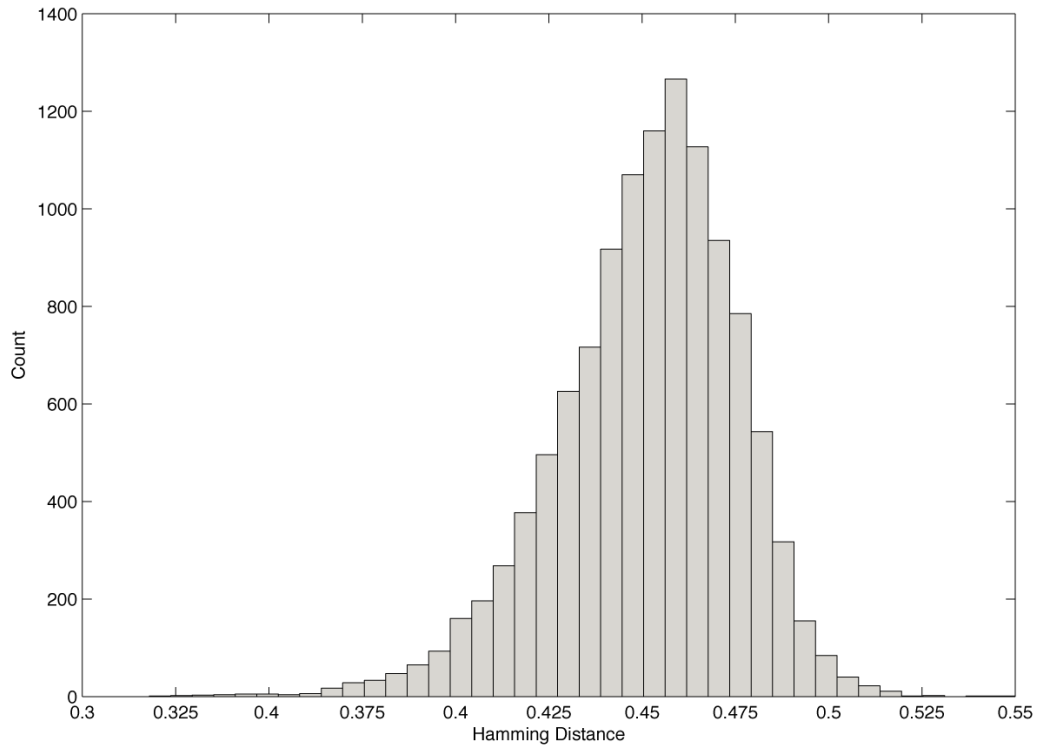


Figure 4.1: The distribution of Hamming distances for non-matching irides with one run of all the irides in the database. The results resemble a binomial distribution with  $\mu = 0.45$  and  $\sigma = 0.024$ .

# Chapter 5

## Conclusions

### 5.1 Final Product

The final prototype is a functional iris detection application, which provides the user with visual prompts and feedback and also some supplementary information about the loaded image for debugging purposes.

The final product can read in an 8-bit indexed, greyscale image and will attempt to automatically best-fit parameters for the location of the pupil, iris and eyelids with consideration for specular highlights and padding for eyelash interference. The entire auto-detection process takes a matter of milliseconds. The user may choose to manually enter points to specify all of these locations; if the user does, he or she will receive appropriate visual feedback cues for repositioning elements in a manner intuitive to any casual computer user.

From this point, the user input is complete and the application will process the image data, unwrap the iris using polar co-ordinates and mask appropriate regions of the unwrapped iris based on the auto-detection sequence or user input. This information is then extracted using Gabor wavelet convolution filters and the final output is a 2048-bit iris code containing phase information of the valid iris pixel data.

The iris code is stored in a local, new line delimited text database if unique or the prototype reports that the code has already been located in the database and a successful match dialog is given.

### 5.2 Auto Detection

Pupil detection in the application works to a very high degree of accuracy in every seen case. The pupil has a fairly unique shade in comparison to the rest of the eye and its surrounding area; this enables an intelligent threshold to be carried out with information from the image histogram to isolate the pupil. This, unfortunately, is not a property shared by the iris, making it significantly more difficult to isolate than the pupil.

Currently the prototype makes an estimation of the location of the iris based on concentricity (a potential source of error) with the pupil, and assumes a detectable gradient ramp between the iris and the sclera. These complications hamper the accuracy of our current estimates and produce a successful detection rate on the CASIA database of



around 70%, though potential improvements to this method are suggested in future work (6).

The prototype manages to detect the top edge of the eyelids in the vast majority of cases. With the small amount of padding added to accommodate the thickness of the top eyelid, it can be seen that, although not perfect, a very reasonable estimate of the eyelid boundary is found automatically.

## 5.3 Source Code

The source code is freely available under the GNU General Public Licence (GPL) from <http://projectiris.co.uk>.

# Chapter 6

## Future Work

Throughout development the focus was on producing a functioning piece of software capable of fast, consistent and reliable iris recognition. There are other possible routes for extension and exploration within this project. There are alternative ideas for the format of the application and unexplored avenues within auto-detection techniques.

### 6.1 Otsu Thresholding for Iris/Sclera Separation

As we have seen before, on average the iris has a lower intensity than the sclera. Hence it could potentially be beneficial to explore the use of Otsu thresholding for separation of the iris and sclera. Similar to the methods implemented for eyelid detection, if we remove eyelids and pupil intensities from the Otsu thresholding process then we obtain the image shown in the right-hand image of figure 6.1. Although to the human eye this image looks to hold more information than the implemented technique (see figure 3.5) the image contains a large amount of noise and would require further filtering and refinement to be useful.

This would be the first recommended route for extension since, during the testing phase, the software tended to fail recognition on images where it could not adequately locate the iris.



Figure 6.1: Image before and after an Otsu threshold.

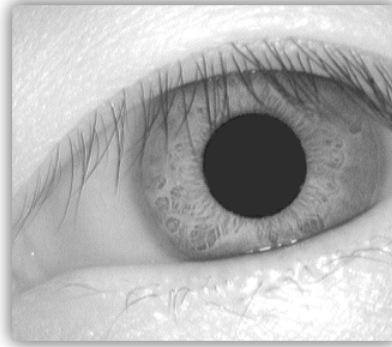


Figure 6.2: The eyelashes have a relatively low intensity and the eyelash region has high variation in intensity.

## 6.2 Eyelashes

In some images eyelashes can obscure parts of the iris. Detection and removal (marking as bad bits) of eyelashes could potentially improve the detection rate. A method is given in [8] for eyelash detection. The method relies on two characteristics of eyelashes: firstly, that they have a relatively low intensity compared to surrounding features, and secondly that the eyelash regions have a large variation in intensity due to the thin shape of the eyelashes (see figure 6.2). The method uses these characteristics by combining a standard-deviation filter and a thresholding technique.

## 6.3 Eyelids

The implementation of eyelid detection is based on the region obtained from the thresholding technique, as we saw in figure 3.7. The system currently constructs a parabola from three points on the boundary of this region. Accuracy in eyelid detection could potentially be improved by using the Hough technique for parabolas. This again would allow more accurate marking of non-iris regions and hence improve overall recognition rate.

## 6.4 Application Format

Currently the system has a text file for storing iris encodings. This could be improved to mirror a real-world application deployment by interfacing with a database. Although due to the one-to-many searches involved with iris recognition database querying could potentially become a bottle neck.

Another area worth exploring is implementing the software with a client/server architecture whereby the iris is encoded client side, and compared server side. Again, this more accurately mirrors real-world deployment of an iris recognition software. This would then require further investigation into security and encryption methods for the system.

# Bibliography

- [1] J. Daugman. How iris recognition works. *IEEE Transactions on circuits and systems for video technology*, 14(1):21–30, 2004.
- [2] J. Daugman. New methods in iris recognition. *IEEE Trans. Systems, Man, Cybernetics B*, 37(1):1167–1175, 2007.
- [3] Center for biometrics and security research. <http://www.cbsr.ia.ac.cn/IrisDatabase.htm>.
- [4] A.K. Jain, A. Ross, and S. Prabhakar. An Introduction to Biometric Recognition. *Biometrics*, 14(1), 2004.
- [5] NSTC Subcommittee on Biometrics. Iris recognition. <http://biometrics.gov/documents/irisrec.pdf/>, 2006.
- [6] S. Perreault and P. Hebert. Median Filtering in Constant Time. *IEEE Transactions on Image Processing*, 16(9):2389–2394, 2007.
- [7] N. Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11:285–296, 1975.
- [8] T. Min and R. Park. Eyelid and eyelash detection method in the normalized iris image using the parabolic hough model and otsu’s thresholding method. *Pattern Recognition Letters*, 30(12):1138 – 1143, 2009.
- [9] J. Blanchette and M. Summerfield. *C++ GUI Programming with Qt 4*. Prentice Hall PTR, Upper Saddle River, NJ, USA, second edition, 2008.
- [10] Bryan Lipinski. Iris recognition: Detecting the pupil. <http://cnx.org/content/m12487/1.4/>, 2004.
- [11] Bryan Lipinski. Iris recognition: Detecting the iris. <http://cnx.org/content/m12489/1.3/>, 2004.
- [12] Bryan Lipinski. Iris recognition: Unwrapping the iris. <http://cnx.org/content/m12492/1.3/>, 2004.
- [13] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis and Machine Vision*. Thomson-Engineering, second edition, 1998.
- [14] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. Hough transform. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm>.

# Appendix A

## Division of Work

The group was initially divided into two parts: Dragos Carmaciu and Thomas Payne working on HCI and application-level decisions, and William Snell and Michael Boyd working on the technical implementation of the wavelets and associated processes. Francis Giannaros aided in delegation between the groups and worked specifically on each side of the divide as required.

To improve design quality and have a more resilient programming flow dynamic, we have frequently exercised *paired programming* throughout the project. As a consequence, a couple of people are frequently responsible for one feature.

Further details are highlighted below.

### Michael Boyd

- Theoretical work on filters
- Implementing the Gaussian filters
- Producing the bitcode and mask
- Producing the heat mask
- Creating the parabolas for the eyelids
- Wrote report 2

### Dragos Carmaciu

- General user interface handling
- Threshold, median and Sobel filters
- Assigning points and generating circles
- Implementing the Hough transformation
- Pupil and iris edge detection
- Website development

- Created tabbed interface
- Wrote report 2

## **Francis Giannaros**

- General user interface
- Creating/implementing the parabolas for the eyelids
- Unwrapping the iris and generating the mask
- Assisting with the Hough transformation
- Text database adding/searching
- Making point locations adjustable
- Infrastructure (mercurial/website) creation/maintenance
- Removal of specular highlights / pupil bad-bits
- Eyelid detection implementation
- Wrote report 1, 2, 3

## **Thomas Payne**

- General user interface
- Unwrapping the iris
- Text database adding/searching
- User instruction/feedback system
- Pupil auto-detection
- Implemented threshold improvements
- Histogram smoothing
- Removal of specular highlights / pupil bad-bits
- Wrote report 2, 3

## William Snell

- Implementation of filter classes
- Implementation of Gabor wavelets
- Iris code comparison and rotation functions
- Linear time median filter
- Otsu thresholding
- Autodetection of eyelids via Otsu thresholding
- Wrote report 2, 3

# Appendix B

## Logbook

Date	Length	Summary
11/01/10	< 1 hour	Supervisor Meeting. <ul style="list-style-type: none"><li>• Outlined requirements of the implementation.</li><li>• Discussed technical implementation of filters.</li></ul>
13/01/10	4 hours	Group Meeting: Full attendance. <ul style="list-style-type: none"><li>• Split into two teams: User Interface and Qt logistics team (Francis, Dragos and Thomas) and Technical Methods and Implementation Team (Michael and William).</li><li>• Test code for Hamming distances written.</li></ul>
18/01/10	< 1 hour	Supervisor Meeting. <ul style="list-style-type: none"><li>• First working prototype of the user interface displaying the loading and I/O functionality of the basic system, including successful unwrapping of the iris to polar coordinates.</li><li>• Refined requirements.</li><li>• More technical details fleshed out. Such as how to interpret Daugman's equations.</li><li>• Discussed the possibilities of generating filters with the differentials of the Gaussian.</li></ul>



24/01/10	6 Hours	<p>Group Meeting: Dragos, William, Francis and Thomas(partial - via webcam).</p> <ul style="list-style-type: none"> <li>• Discussion regarding overall design.</li> <li>• UML diagram refined, clear targets and deadlines set.</li> <li>• Extensions such as auto-detection and full database integration considered.</li> <li>• Re-factored the code, moving to the main source directory. Version control repository overhauled.</li> <li>• Started using doxygen as a code commenting system.</li> </ul>
25/01/10	< 1 hour	<p>Supervisor Meeting.</p> <ul style="list-style-type: none"> <li>• Discussion of bit masks for eyelids and minimum requirements</li> <li>• Prof. Gillies suggested researching the Hough transform for pupil detection.</li> </ul>
28/01/10	4 Hours	<p>Group Meeting: Dragos, William, Francis and Michael (partial).</p> <ul style="list-style-type: none"> <li>• Bit code generated for full size of the unwrapped iris.</li> <li>• New UI integrated though incomplete.</li> <li>• Considered retrieval and comparison operations for stored bitcodes.</li> <li>• Micheal implemented code to produce heatmaps for the Gabor filters.</li> </ul>
01/02/10	< 1 hour	<p>Supervisor Meeting.</p> <ul style="list-style-type: none"> <li>• Demonstrated current revision, with latest UI.</li> <li>• Discussed first report and usefulness of current techniques in autodetection.</li> <li>• Proposed new extension to implement a client/server mechanism to more accurately parallel a real-world application.</li> <li>• Suggest improvements to UI (draggable points) and comparing shifted codes to compensate for rotation of the iris image.</li> </ul>

08/02/10	< 1 hour	Supervisor Meeting. <ul style="list-style-type: none"><li>• Demonstrated suggested improvements from previous week's meeting (draggable points and code shifting).</li></ul>
15/02/10	< 1 hour	Supervisor Meeting.
21/02/10	4 hours	Group Meeting: Dragos, William and Thomas. <ul style="list-style-type: none"><li>• Work on report 2.</li></ul>
01/03/10	< 1 hour	Supervisor Meeting. <ul style="list-style-type: none"><li>• Submitted report 2.</li><li>• Discussed finalisation of implementation of project.</li><li>• Discussed report 3.</li></ul>
08/03/10	< 1 hour	Supervisor Meeting. <ul style="list-style-type: none"><li>• Received feedback on report 2; discussion on the style of the report. Suggestions for structural improvement made.</li><li>• Presented Hamming distance results; discussion on the similarity to Daugman's results, as a binomial distribution and a similar mean.</li></ul>
12/03/10	2 Hours	Group Meeting: William, Thomas. <ul style="list-style-type: none"><li>• Structure of report 3 finalised.</li></ul>
15/03/10	< 1 hour	Supervisor Meeting. <ul style="list-style-type: none"><li>• Discussed presentation: formulated outline, slide designs, demo methodology.</li><li>• Submitted draft of report 3.</li></ul>
18/03/10	3 hours	Group Meeting: Dragos, Francis and Thomas. <ul style="list-style-type: none"><li>• Layout finalised for presentation.</li></ul>