Jagiellonian University
Faculty of Mathematics and Computer Science
INSTITUTE OF COMPUTER SCIENCE AND COMPUTATIONAL MATHEMATICS
Full time studies

Identification number: 1090589

Magdalena Marta Malczyk

Iris Recognition Algorithm Implementation

Master's thesis supervisor:
dr Krzysztof Misztal

Cracow 2017

Index

List of abbreviations

CASIA	Center for Biometrics and Security Research; localised in the Institute of Automation, Chinese Academy of Sciences
HD	Hamming Distance
IDE	Integrated development environment
IRIS	United Kingdom's Iris Recognition Immigration System
OpenCV	Open Source Computer Vision Library; available in C++, C, Python and Java

Abstract

This paper outlines the iris recognition problem, some of the currently deployed iris recognition systems and biometric databases accessible to researchers. The goal of this project was to build a Java implementation of an iris recognition algorithm. The focus was on modularity in design, which would enable further expansion, unit test application and potential replacement of module elements. Functions from an open source vision library (OpenCV) were used to process images. This paper describes the project's structure, implementation and modifications. Experiments were performed in order to simulate both iris verification and identification. The results show how the number of erroneous matches can be modified through adjusting the Hamming Distance ranges used for iris code comparison. Finally the project's areas for improvement are discussed.

Keywords: *iris recognition, biometrics, image analysis, 2-D Gabor filters.*

Praca opisuje problem rozpoznawania tęczówki oka, niektóre z funkcjonujących obecnie systemów rozpoznawania tęczówki oraz bazy biometryczne, do których można uzyskać dostęp. Celem projektu było zaimplementowanie algorytmu rozpoznawania tęczówki przy użyciu języka programowania Java. W projekcie postawiono nacisk na modularną konstrukcję oprogramowania, co pozwoliłoby na jego dalszy rozwój, stosowanie testów jednostkowych oraz potencjalne zastępowanie elementów modułów. Do przetwarzania obrazów zostały użyte funkcje z biblioteki widzenia komputerowego open source (OpenCV). W pracy opisano strukturę projektu, zagadnienia implementacji i użyte modyfikacje. Eksperymenty zostały przeprowadzone w celu symulowania zarówno potwierdzenia tożsamości, jak i pełnej identyfikacji. Wyniki pokazują, jak zmiana przedziałów odległości Hamminga używanych do porównywania kodów tęczówek wpływa na ilość błędnych rezultatów. Na zakończenie przedstawiono obszary projektu, które można dalej rozwijać.

Słowa kluczowe: *rozpoznawanie tęczówki, biometria, analiza obrazów, filtry Gabora 2D.*

Introduction

Iris recognition is an increasingly widely used biometric method. It is a very broad topic which could easily fit the pages of an entire book. At the same time, it is a relatively new subject in the field of security and authentication, as the first deployment of an iris recognition system began in 2002 with the trial of UK's IRIS [6], and there is still room for improvement and advances in this area.

Authorship or identity commonly needs to be proven or somehow marked. Many methods have been used thorough history, namely presenting a token of sorts, such as a seal or signature, or relying on a testimony of an external, trusted authority. These combined give us a modern identity card. These kinds of identification methods carry a following problem: the proof of identity is either immaterial or separable from the person who needs to have their identity confirmed.

Biometrics are a solution for this problem, a metric derived directly from a person's physical characteristics. Into this category falls, among others, voice, human gait, finger- and palm prints, iris and retinal pattern. Biometrics are not a recent concept. Fingerprints, for example, have been found on ancient Mesopotamian artifacts. Forensic science has been using fingerprints since the XIXth century.

What makes iris patterns a particular focus in the field of biometrics is an exceptionally good false match rate and stability. The iris forms in gestation and, barring injury or a major illness, remains unchanged throughout one's lifetime. Each eye has a statistically unique pattern. The acquisition of an image of an iris is noninvasive and more socially acceptable than, for example, a DNA sample.

The goal of this project was to build a Java implementation of an iris recognition. The focus was on modularity in design, which opens potential for further expansion, unit test application and potential replacement of module elements. The

emphasis was on using readily available, open-source solutions such as the OpenCV library.

The first chapter of this paper introduces the iris recognition problem and discusses the advantages and disadvantages of this biometric. It outlines the applications of some of the currently deployed iris recognition systems and lists several biometric databases accessible to researchers.

The second chapter discusses the methods used in this implementation. Each stage of Daugman's iris recognition algorithm is described. Next, several methods from the OpenCV library used in this implementation were chosen to be described.

The third chapter demonstrates how these methods have been modified and applied to each module. The adjustments are partially influenced by the format of the iris images in the CASIA-IRIS-Thousand database.

The fourth chapter describes how the joined modules were used to analyse a set of iris images. The accuracy of this implementation of an iris recognition algorithm is estimated. The fact that iris edges cannot be localised on all images has to taken into account, but most of the comparisons return a conclusive result. It can be demonstrated that changing the HD ranges affects the percentage of correctly accepted and rejected iris code matches.

In the last part of this work, the results of the assessment prompt a discussion of areas where the project could be expanded and improved.

1. Iris recognition

Iris recognition is a method of biometric recognition which aims to identify individuals through analysing the pattern of their irises. In comparison with other biometric methods, the iris pattern remains stable throughout the lifetime, which makes it a valuable and dependable metric. It has a low likelihood to produce false matches, as was shown in multiple comparison tests performed in commercially deployed iris comparison systems. It's reliable at distances of up to 1 meter [2]. To avoid false matches, the images should be adequately focused and show at last 50% [2]. The iris and the iris itself should have a radius of at least 70 pixels [2].

Iris codes for different eyes have been demonstrated to be consistently different. That includes in particular the cases where the encoded irises belonged to the same person or twins. These eyes have identical or nearly identical genetic material, but the circumstances of epigenetics lead to the development of differing patterns. Although the process of gaining pigmentation continues through the early childhood, the iris pattern is formed by the eighth month of gestation [2].

The purpose of biometric systems, including iris recognition systems, is to provide means of identification and access control to places, information or other resources. They are meant to replace or complement already existing, more traditional authorisation and security schemes. Typically, though not necessarily, there is an emphasis placed on a single characteristic, such as security or efficiency [6].

1.1. Classification

Iris recognition systems can be classified by whether they operate in *identification* or *verification* mode. During identity verification we are assuming and confirming somebody's identity. That requires a person to present an additional token as proof. Assuming there's a match for this token in the database, an iris code can be then computed out of a current eye image and verified against the existing record [6].

A verification system requires less computing time and space than a system primarily meant for identification. A single run of an iris identification algorithm calls for an exhaustive search, which potentially encompasses the entire iris database. The search ends either when a match is found or when there are no more iris code records to compare against [6].

Systems can also be differentiated by whether a potential attacker would conduct an impersonation attack or a concealment attack. In the first case, the aim would be to impersonate somebody in particular. In the second case, to impersonate without distinction anybody who is already enrolled in the database. It is essential not to allow any false matches. The probability of false matches grows as the iris code database expands. Designing the system to enforce an additional token reduces the risk of a false match, as only one comparison is necessary to confirm or reject the identity match [6].

1.2. Advantages

The iris pattern is unique, forms in gestation and remains stable throughout the human lifetime, especially in comparison with other biometrics, for example

fingerprints or face features. It is notable that iris patterns differ for genetically identical eyes, by which are also meant the eyes of identical twins [3].

Another trait which makes the iris suitable as a biometric is that the entire iris doesn't need to be shown on the image to make a match. The iris pattern can be to a certain point partially concealed with eyelids.

The iris code is a biometric that is accessible for collection in a non-invasive, safe way, making it acceptable for the use in general populace. An example of a more invasive biometric is DNA or the retinal pattern.

An important characteristic of the iris code is that it does not provide information about a person's race, religion or economic status, preventing possible bias against them.

Because of the low false match probability, a deployed biometric system leads to the improvement of security and efficiency where the ability to identify a person is necessary [6].

1.3. Shortcomings

Like other biometrics, iris recognition systems face notable challenges in real-world deployment. Eye images acquisition in less controlled environments means gathering also images with poor resolution, at greater distance and in motion. This creates distortion of the images and may present problems with localising pupil and iris boundaries [3].

There is also a potential of encountering falsified images. An impersonator may try to present a static image or wear colored contact lenses. For some systems, it is critical to recognise impersonation attempts [3].

Another concern is that local law may enforce restrictions upon collection and storage of biometric data. For example, Dutch law used to forbid the State to store personal biometric data. By extension this prevented watch-list type systems from

being deployed and required biometric systems in Netherlands to work in verification rather in identification mode [6].

1.4. Deployed applications

1.4.1. Access to restricted areas

Iris recognition systems can provide controlled access to high security areas that need to screen the entrances and exits for intruders. This type of a system can be found in Google datacenters and areas of airports restricted to qualified personnel [6].

1.4.2. Airports and border-crossings

Iris pattern scanning can be used as means of identification. A passenger who, preceding a flight, registers their identity in a suitable iris recognition system is able to confirm their identity without a passport both on departure and arrival. One of the examples is Iris Recognition Immigration System, which used to function in the Great Britain but is currently decommissioned [6].

United Arab Emirates has been screening arrivals to the country since 2003. This is done through a watch-list system called *Expellee Tracking and Border Security Iris System*. It uses a database of biometrics of known *personae non gratae*. At the time this system was introduced, the percent of foreign immigrants on UAE grounds was about 85% [6]. The country is saturated with immigrant workers and new arrivals.

In 2010, all existing iris recognition systems deployed at airports were using the Daugman algorithm [6].

1.4.3. Smartphones

In 2015, the first smartphones containing an iris scanner were introduced to the consumers. The first was a Fujitsu product, followed within the span of a few months by Microsoft Lumia 950.

1.4.4. Tracking attendance

Some schools have been tracking student attendance through iris identification systems. A notable example is the system deployed in Kenya in 2015, which is applied both to school buses and roll-calls in classes.

1.4.5. Adhaar, India

People living in India may lack a proper proof of their identity due to their living and economic situation. Physical proof typically used in First World countries, in the form of identity documents, may be lost during a natural disaster. The region unfortunately floods easily and on a large scale.

The Adhaar system is designed to work with several biometrics, including iris scans, to provide a reliable identity proof for the citizens of India. Iris pattern has an additional advantage of not delivering information on caste or religion, therefore avoiding unnecessary bias.

1.5. Biometric databases

There are several publicly available digital repositories of biometric data. Typically their creators require the credentials of persons who seek to use their databases. Official ties to universities and research facilities might be necessary.

Some of more noteworthy databases are listed below.

1.5.1. CASIA-IrisV4

This database was created by the Institute of Automation of Chinese Academy of Sciences. It consists of six subsets, each addressing a different problem in iris recognition, and contains over 50,000 of iris images in total. The images were gathered from over 1,800 persons. The photographs were taken in infrared light and saved as 8 bit grayscale JPEG files [3].

Of the listed subsets, CASIA-Iris-Thousand and CASIA-Iris-Interval were the ones used in the process of writing of this paper.

CASIA-Iris-Thousand– A total of 20,000 images. The iris data was acquired from a 1,000 people; each eye was recorded as 10 images. Iris pattern may be obscured by eyeglasses frames and the reflections that form on the surface of lenses and eyes [3].

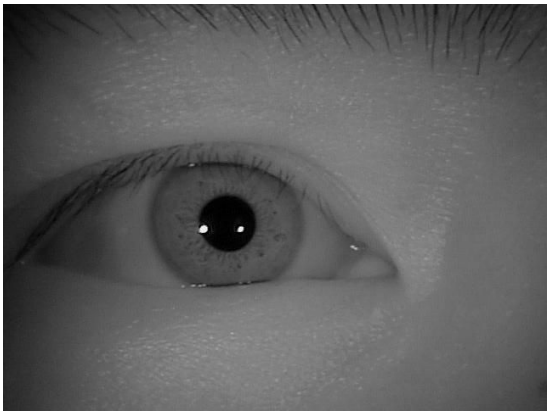

Figure 1: An example image from CASIA-Iris-Thousand database[7].

CASIA-Iris-Interval - The details of the iris pattern are meant to be especially visible in this dataset [3].

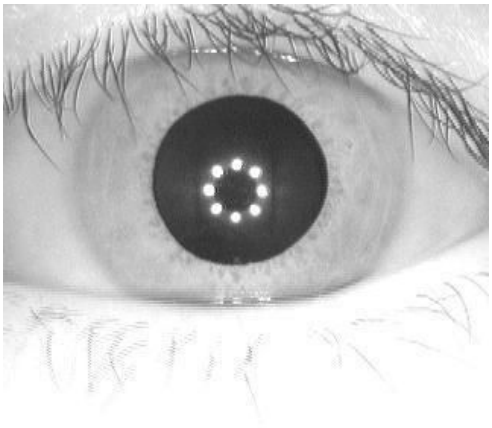

Figure 2: An example image from CASIA-Iris-Interval database[7].

CASIA-Iris-Twins - Eye images collected from 100 pairs of twins. It is notable that iris patterns differ for genetically identical eyes [3].

CASIA-Iris-Lamp - The database shows the appearance of the human eye in varying illumination conditions, ranging from dim to bright light [3].

CASIA-IRIS-Distance - Data was gathered from a distance of 3 m, longer than is typically ideal [2].The images show the entire face [3].

CASIA-Iris-Syn - Iris images were computer generated, based on images which already exist in the database. The resulting constructs were then inserted into real eye images [3].

Link: http://biometrics.idealtest.org/ [7]

1.5.2. Biometrics Datasets – University of Notre Dame

The Computer Vision Research Laboratory of University of Notre Dame, USA, has compiled a collection of over 30 different biometric datasets, including iris recognition ones. Some of the noteworthy ones include *The Gender from Iris Dataset* and several contact lenses datasets. There is also a time-lapse dataset which shows eye images collected from the same subjects 4 years apart [8].

Link: https://sites.google.com/a/nd.edu/public-cvrl/data-sets [8]

1.5.3. UBIRIS – Noisy Visible Wavelength Iris Image Databases

The database authorship belongs to the Department of Computer Science of the University of Beira Interior, Portugal. The focus of this dataset is aiding the development of iris recognition algorithms that don't assume the cooperation of subjects during the image acquisition phase. Images were captured mid-motion and at larger than recommended distances [9].

Link: http://iris.di.ubi.pt/ [9]

2. Applied solutions

2.1. Biometric databases

Portions of the research in this paper use the CASIA-IrisV4 database collected by the Institute of Automation from the Chinese Academy of Sciences. In particular, some of the images in the CASIA-Iris-Thousands and CASIA-Iris-Interval were used to test the iris recognition algorithm implementation [3].

2.2. Daugman's algorithm

Iris recognition is valued for its capability to produce no false matches, shown in multiple comparison tests performed in commercially deployed iris comparison systems [2]. It's reliable at distances of up to 1 meter. To avoid false matches, the images should be adequately focused, show at last 50% of the iris and the iris itself should have a radius of at least 70 pixels.

Following paragraphs describe stages which generally preclude iris code comparison.

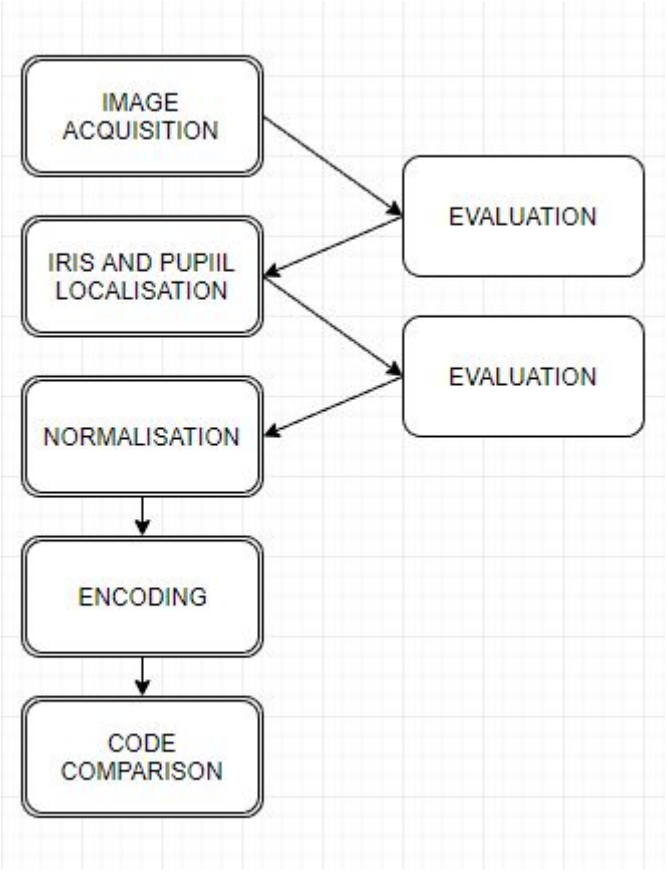

Figure 3: Major modules of a typical iris recognition algorithm.

2.2.1. Image acquisition

Ideally images should be grayscale, captured by an infrared camera which operates in the range 700-900 nm [2]. If visible light were used, it could irritate human eyes. Images taken mid-blink could conceal too much of the iris to be useful.

2.2.2. Image evaluation

The radius of iris should be at least 70 pixels, but the acceptable range is also 80-130 pixels. To make iris pattern analysis possible, an image should pass a

minimum focus criterion. Daugman suggests it should be calculated through a 2-D Fourier transform [2].

The CASIA images are held to a consistent standard, which was expanded upon in the CASIA-Irisv4 section. Therefore the focus evaluation step in this project was unnecessary and was not undertaken.

2.2.3. Localisation of iris and pupil

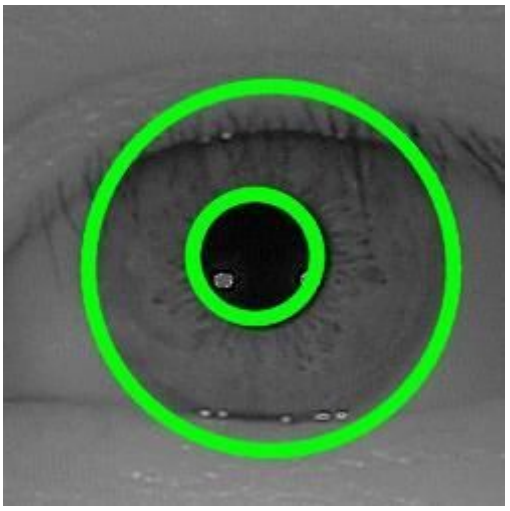

Figure 4: Edges of the pupil and iris localised by a Hugh Circles function. Original image taken from Casia-Iris-Thousand database [7].

The iris and pupil don't necessarily form concentric circles, especially if the eye is photographed at an angle or with its gaze not directed at the camera. Even when the eye is facing the camera, the pupil centre is often located closer to the nose than the iris centre [2].

2.2.4. Normalisation of iris area

Once the centres and diameters of iris and pupil are known, the eye area can be projected onto a polar coordinate system, which Daugman calls a homogenous rubber sheet model [2].

$$x(r,\theta) = (1-r)x_p(\theta) + rx_s(\theta)$$
$$y(r,\theta) = (1-r)y_p(\theta) + ry_s(\theta)$$

Each point of the iris area is assigned coordinates corresponding to its distance to the centre of the iris (r) and the angle (θ) on the circle it belongs to.

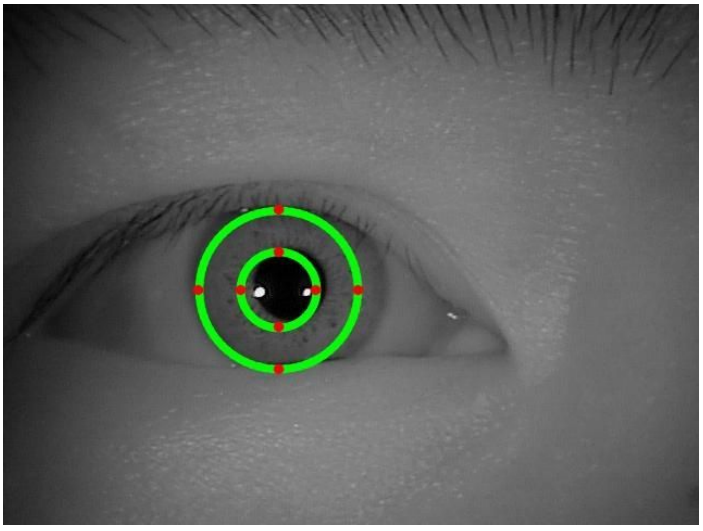

Figure 5: On the rubber sheet model, points on the iris that lay on the same radius of pupil circle are translated to lay on the same column if the circles are concentric. The original image is from CASIA-Iris-Thousand database.

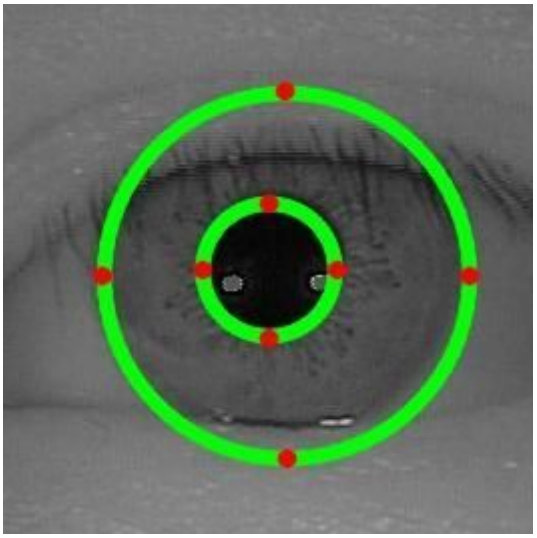

Figure 6: Distortion happens when iris and pupil circles aren't concentric.

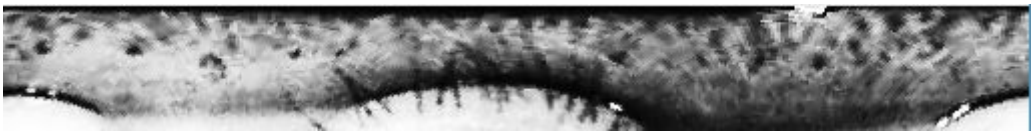

Figure 7: Normalised iris image.

2.2.5. Encoding

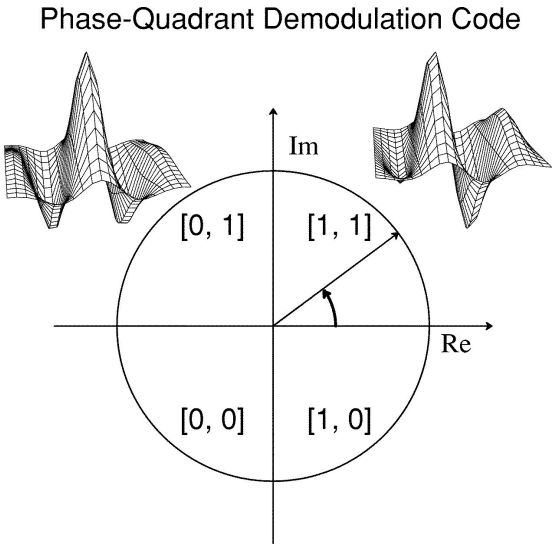

Figure 8: Illustration of phase quantification of the results of the Gabor filter. Daugman, How Iris Recognition Works *[2]*

$$h_{\{Re,\,Im\}} = sgn_{\{Re,\,Im\}} \int_{\varphi}\int_{\varphi} I(\rho,\varphi) e^{-i\omega(\theta_0-\varphi)} \cdot e^{-\frac{(r_0-\rho)^2}{\alpha^2}} \cdot e^{-\frac{(\theta_0-\varphi)^2}{\beta^2}} \rho d\rho d\varphi$$

The Gabor filter equation as dictated by Daugman [2].

The normalised iris pattern is divided into 8x124 areas, each of which is filtered by a Gabor filter. The result of a single filter pass is mapped onto the complex plane, depending on the sign of its real and imaginary parts. Each quadrant of the coordination system is assigned a 2-bit code. The total size of the standard code is 2048 bits [2].

This means the amplitude information is dismissed and the function phase information determines the code bits assigned to a given area. It is a cyclic code, which means that the 2-bit codes assigned to adjacent phase quadrants differ only in one bit. Discarding the amplitude information leads also to the ability to distinguish between badly focused iris images [2].

Similarly, a mask of equal size can be computed to note to areas of the code that don't carry meaningful information, such as those corresponding to eyelids or eyelashes in the original image [2].

2.2.6. Iris code comparison

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

Hamming Distance (HD) denotes the number of positions on which two codes differ, divided by total code size. The version used in Daugman's algorithm accounts for the inclusion of masks. Using the logical AND operator allows data from irrelevant areas of the image to be discarded [2].

Every bit in the code is assumed to be uncorrelated to the corresponding bit in any other iris code. That means HD=0.500, which Daugman confirmed with his observation of mean HD p=0.499 with standard deviation 0.0317 [2].

Daugman gives as follows the probability of encountering a false match (P_N) during an exhaustive search of a database containing *N* records [2] [6]:

$$P_N = 1 - (1 - P_1)^N$$

P_1 in this case is the probability of an erroneous result of a single comparison. This shows that the error rate of an iris recognition algorithm should grow with database size, which in turn means that particular attention has to paid to ensuring that codes generated from different irises are sufficiently distinct and codes generated from similar iris images show enough similarities. [6]

2.3. OpenCV Library

2.3.1. *Mat*

The Mat class provides an n-dimensional dense array of several different types. It supports common matrix operations such as transposition or scalar and matrix addition, subtraction and multiplication. Several different formats are available, which enables Mat class objects to be treated as pixel matrices of images. Single-channel Mat objects can represent grayscale images and multi-channel objects can handle representation of color images [12].

2.3.2. *Canny*

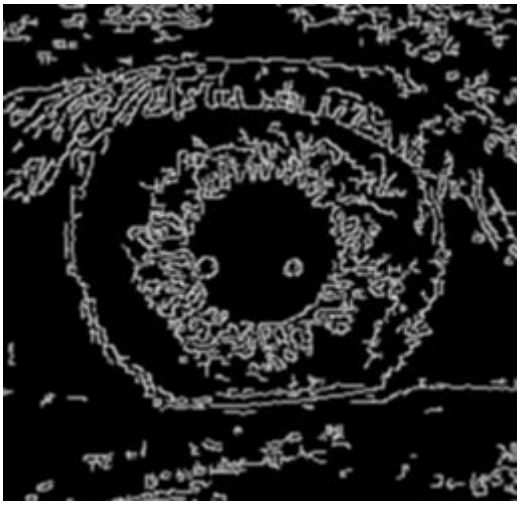

Figure 9: Eye image analysed by the Canny function.

OpenCV's Canny edge detector is used to enhance the edges of the iris and pupil. It is an implementation of the Canny algorithm [1].

The aim of this action is to remove some noise from the image, such as eyelashes and iris structure. At the iris localisation phase, this data is not necessary and can interfere with finding the edges.

2.3.3. Hugh Circle Transform

The Hugh Circles function finds centres and radii of circles in images. Minimum and maximum radius can be specified. The OpenCV algorithm is an implementation of the Hugh gradient method, which is a modification of the Hugh transform. This makes it sensitive to changes of color in the image [1].

2.3.4. Gabor Filter

The Gabor Filter can be used for the extraction of pattern features in images. The OpenCV implementation function returns wavelets which can be used as arguments for filtering 2D matrices. However, this particular implementation generates only filters with real values, so it does not provide access to imaginary values if a complex Gabor Filter is needed. [11]

3. Project description

The project is divided into several modules, corresponding to subproblems of the iris recognition algorithm. The structure is intended to enable further growth of the project, unit test application and using new versions of module classes. Implementation is based on the solutions supplied by the OpenCV library. This chapter describes the consecutive modules in more detail.

3.1. Technical requirements

The IDE used was IntelliJ IDEA 2017.2.4. Plugins GitHub, Maven, JUnit and .ignore should be installed. Java 8 JDK and OpenCV 3.2 were used. The OpenCV module needs to be manually set as a dependency in the project structure menu.

GitHub repository of project:
https://github.com/mmalczyk/Iris.git

3.2. Structure

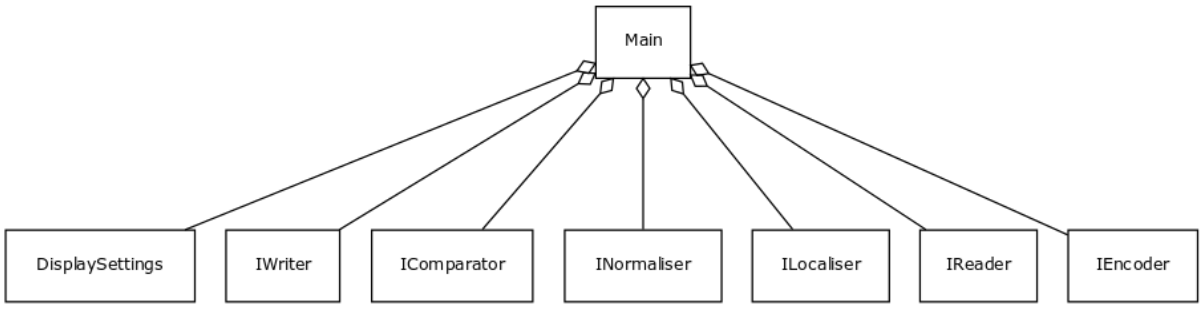

Figure 10: Diagram illustrating the basic modules used by the program.

The project contains following modules:

- Display, enables results and partial results to be viewed.
- Settings, provides runtime settings for each module.
- Reader, loads images from database.
- Localiser, finds the area of the picture containing the pupil and iris.
- Normaliser, transforms localised iris into uniform, rectangular format.
- Encoder, transforms normalised input into iris code.
- Comparator, responsible for iris code comparison.
- Writer, iris code database access (not yet implemented).

Module properties can be set by entries in the *plugin.properties* file, seen below. Afterwards the correct module version is automatically provided by the *PluginFactory* class. The properties file governs both the module version to be used and whether the results should be displayed on the screen when the program is running.

The plugins.properties file

```
#main.interfaces.IComparator=main.comparator.ComparatorStub
#main.interfaces.IComparator=main.comparator.ByteArrayComparator
main.interfaces.IComparator=main.comparator.MatComparator
main.interfaces.IComparator_DISPLAY=false

#main.interfaces.IEncoder=main.encoder.EncoderStub
main.interfaces.IEncoder=main.encoder.OpenCVEncoder
main.interfaces.IEncoder_DISPLAY=true

#main.interfaces.ILocaliser=main.localiser.LocaliserStub
main.interfaces.ILocaliser=main.localiser.OpenCVLocaliser
main.interfaces.ILocaliser_DISPLAY=true

#main.interfaces.IComparator=main.comparator.NormaliserStub
main.interfaces.INormaliser=main.normaliser.OpenCVNormaliser
main.interfaces.INormaliser_DISPLAY=true

#main.interfaces.IComparator=main.comparator.ReaderStub
main.interfaces.IReader=main.reader.OpenCVReader
main.interfaces.IReader_DISPLAY=false

main.interfaces.IWriter=main.writer.WriterStub
main.interfaces.IWriter_DISPLAY=false
```

As the iris data is analysed, it is saved into variables in the *ImageData* class. Objects of this class serve as the vector for data exchange between used modules. Some of the modules are designed to be used consecutively, not unlike a pipeline. The order is as follows: Reader, Localiser, Normaliser, Encoder, Comparator and Writer.

3.3. Display module

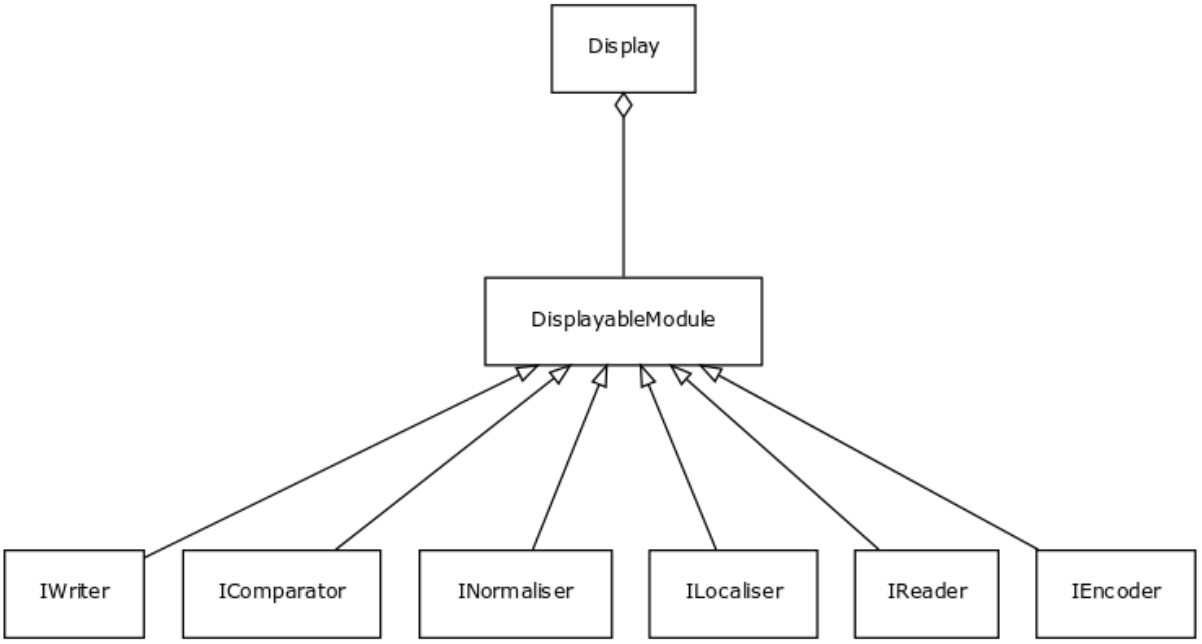

Figure 11: Diagram illustrating the Display module.

The Display module makes it possible to show the intermediate stages of the algorithm. It can be useful to restrict the output to focus only on certain modules. The display functionality can be turned off and on using the properties file.

3.4. Reader module

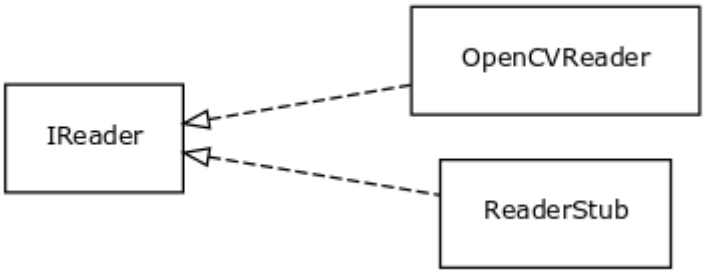

Figure 12: Diagram illustrating the Reader module.

```
public interface IReader {
   ModuleName moduleName = ModuleName.Reader;
   IReader INSTANCE =
           (IReader) PluginFactory.getPlugin(moduleName);

   ImageData read(Path filePath);
}
```

The implementation class *OpenCVReader* loads grayscale (single-channel) images into a *Mat* object and subsequently into an *ImageData* object via function *read*. It can be easily substituted in order to implement access to a specific database or provide additional functionality.

3.5. Localisation module

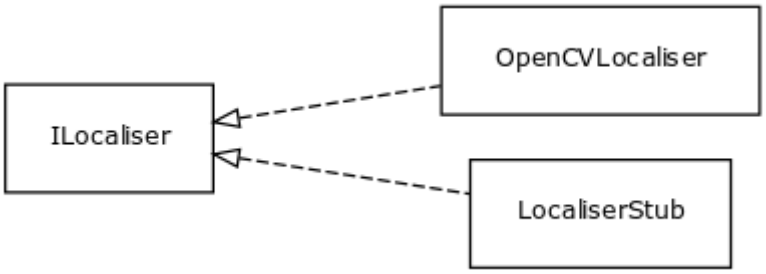

Figure 13: Diagram illustrating the localisation module.

```
public interface ILocaliser {
   ModuleName moduleName = ModuleName.Localiser;
   ILocaliser INSTANCE =
           (ILocaliser) PluginFactory.getPlugin(moduleName);

   ImageData localise(ImageData imageData);
}
```

Interface ILocaliser and its derived classes are the main components of the iris localisation module. *ILocaliser* is meant to be used by calling function *localise* with the constraint that a valid *Mat* object is assigned to the field *imageMat* in the function argument *imageData*.

Fragment of the iris localisation algorithm

```
src = removeReflections(src);
boolean foundPupil = findPupil(src);
if (foundPupil) {
    Mat roi = rebaseToROI(imageData, src);
    boolean foundIris = findIris(roi);
}
```

The first problem encountered when writing OpenCVLocaliser was a distinct shadowed area below the eyebrow, present in the images from the CASIA database. The shape of the shadow caused the *hughcircles* function to consider the eyebrow area as a part of a large circle encompassing the entire eye. This is partially dependent on the minimum and maximum radii used as arguments in *hughcircles*.

To go over this obstacle, pupil edge search was performed before iris search, which allowed the minimum and maximum iris radius to be estimated. The iris to pupil ratio has a value between 0.1 and 0.8 [2]. Using this data, a square area of the original image, concentric with the iris circle, could be separated and analysed to find iris edges.

In an effort to simplify the already complex project, the Fourier transform suggested by Daugman [2] wasn't used. Instead, Hugh Circles transform [1] was applied to find the edges of iris and pupil in images.

The *removeReflections* function is an additional but not mandatory step that was taken to improve iris localisation in images where the subject was wearing glasses. Areas of white were filled with background colour that was generated from the average of pixel values in the image. The presence of bright reflections on the lenses interfered with localising the iris, because they were considered by the *hughcircles* function as possible circles that matched the parameters specified for the iris. Additionally, replacing the area with a neutral colour improved the outcome of histogram equalisation when one was performed.

```
private boolean findPupil(Mat orgSrc) {
    Mat src = orgSrc.clone();

    int threshold = 70;
    Canny(src, src, threshold, threshold * 2);
    Imgproc.GaussianBlur(src, src, new Size(3, 3), 0, 0);

    //radius depends on whether findPupil is used before or after findIris
    int minRadius, maxRadius, minDistance;
    if (src.width() == src.height()) {
```

```
            minRadius = (int) (0.1 * src.width());
            maxRadius = (int) (0.8 * src.width());
            minDistance = src.width();
        } else {
            minRadius = 10;
            maxRadius = min(src.width(), src.height()) / 2;
            minDistance = min(src.width(), src.height());
        }
        Mat circles = new Mat();
        Imgproc.HoughCircles(
                src, //Input image
                circles, //Memory Storage
                Imgproc.HOUGH_GRADIENT, //Detection method
                2, //Inverse ratio
                minDistance,
                100, //Higher threshold for canny edge detector
                50, //Threshold at the center detection stage
                minRadius,
                maxRadius
        );

    //example use of hughcircles
//https://github.com/badlogic/opencv-fun/blob/master/src/pool/tests/HoughCircles.java
        int length = circles.cols();
        for (int i = 0; i < length; i++) {
            Circle circle = new Circle(circles.get(0, i));
            imageData.addPupilCircle(circle);
        }

        display.displayIf(src, circles, displayTitle("pupil circle"));

        return circles.cols() > 0;
    }
```

The original iris images were analysed by OpenCV's *Canny* function followed by a *GaussianBlur* to make the pupil edges more distinct. The upper threshold of

pixel values (for acceptance as an edge) was set as 140, and the lower (for rejection) as 70.

When the *findPupil* function was used before calling the *findIris* function, the size of the pupil could be expected to stay within size constraints of 10 pixels to half the smaller size measurement of the image. If it was used after, the minimum and maximum radius could be set as respectively 0.1 and 0.8 [2] of the image size, as the image measurements were adjusted to the iris size. These assumptions could be made as the images in the CASIA databases follow a consistent format. In both cases, the minimum distance between the centers of localised circles was set as equal to image height.

```
    private Mat focusOnArea(Mat src, Circle circle) {
        final int r = (int) circle.getRadius();
        final int x = (int) circle.getX();
        final int y = (int) circle.getY();

        int height = src.height();
        int width = src.width();

        int right = (x + r >= width) ? x + r - width + 1 : 0;
        //right and bottom boundaries are NOT inclusive
        int bottom = (y + r >= height) ? y + r - height + 1 : 0;
        int top = 0, left = 0; //origin point; boundaries are inclusive
        if (r > x) {
            left = r - x;
            //adjust coordinates
            circle.setX(x + left);
        }
        if (r > y) {
            top = r - y;
            circle.setY(y + top);
        }

          //prevent out of bounds error
        Mat dst = new Mat(src.height() + top + bottom, src.width() + left + right,
src.type());
```

```
            Scalar color = generateBackgroundColor(src);
               copyMakeBorder(src, dst, top, bottom, left, right, BORDER_CONSTANT,
color);

            int adjX = x + left - r; //-r because Rect is initialised with coordinates
of its top left corner
            int adjY = y + top - r;
            int size = 2 * r;
            return new Mat(dst, new Rect(adjX, adjY, size, size));
        }
```

After localising the pupil center, it was possible to extract a square submatrix which would contain the entirety of iris, but exclude the eyebrow. The length of the square's side was set to 4 times the pupil radius. The pupil to iris ratio can be as little as 0.1 [2], but in the CASIA-Iris-Thousand subset the light conditions were consistent enough that a ratio of 0.4 was adopted.

Partially removing the shadowed eyebrow area prevented *Canny* and *HughCircles* from interpreting its semicircular shape as part of a larger circle, therefore the following algorithm could consider the iris to be the most likely circle in the image.

```
    private boolean findIris(Mat org_src) {
        Mat src = org_src.clone();

        equalizeHist(src, src);
        int threshold = 70;
        Canny(src, src, threshold, threshold * 2);
        Imgproc.GaussianBlur(src, src, new Size(3, 3), 0, 0);

        display.displayIf(src, displayTitle("canny iris"));

        Mat circles = new Mat();

        Imgproc.HoughCircles(
                src, //Input image
```

```
            circles, //Memory Storage
            Imgproc.HOUGH_GRADIENT, //Detection method
            2, //Inverse ratio
            100, //Minimum distance between the centers
            150, //Higher threshold for canny edge detector
            120, //Threshold at the center detection stage
            70, //min radius
            min(src.width(), src.height()) //max radius
    );

    for (int i = 0; i < circles.cols(); i++)
        imageData.addIrisCircle(new Circle(circles.get(0, i)));

    display.displayIf(org_src, circles, displayTitle("iris circle"));

    return circles.cols() > 0;
}
```

The *findIris* function begins by calling the *equalizeHist* from the OpenCV library. The goal was to improve the contrast between the sclera and the iris. Next Canny edge detection was performed with the threshold of 70 and 140, then a gaussian blur and a Hugh circles transform, analogous to the *findPupil* function. The minimum iris radius was set as 70, maximum as equal to the image's width and height.

The result of *HughCircles* is an array of Mat objects which lists the radii and centres of all the found circles. The first result in the array is supposed to have been assigned the highest certainty value [1].

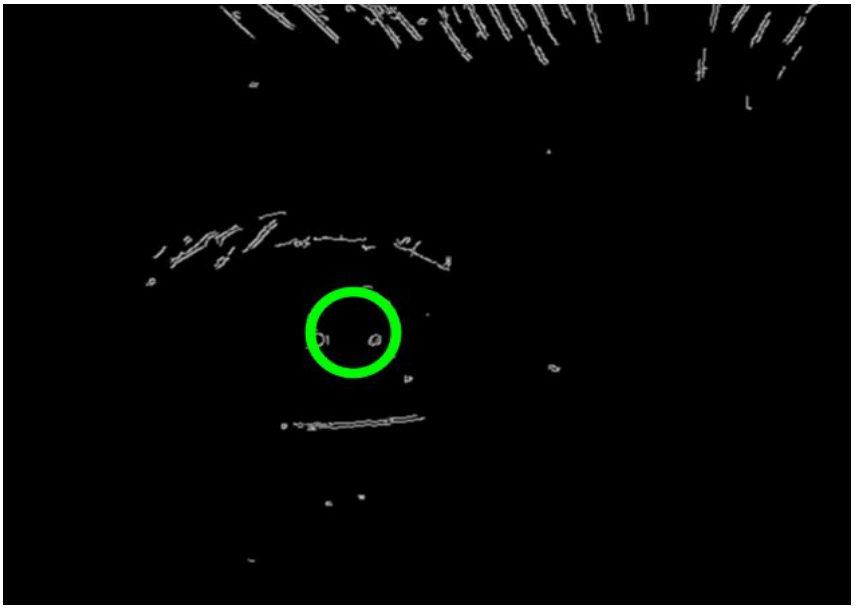

Figure 14: Localisation of the pupil after Canny function analysis.

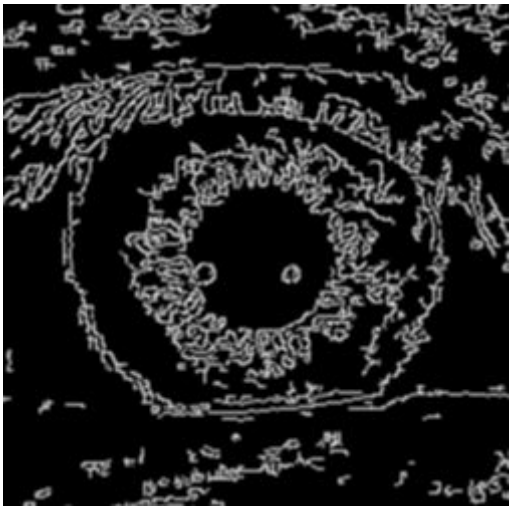

Figure 15: Iris area after calling the Canny function.

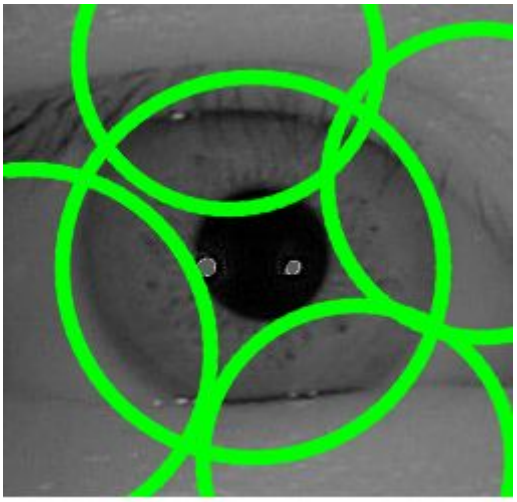

Figure 16: The Hugh circles function returns an array of possible circles. The mostly likely circle is the first element.

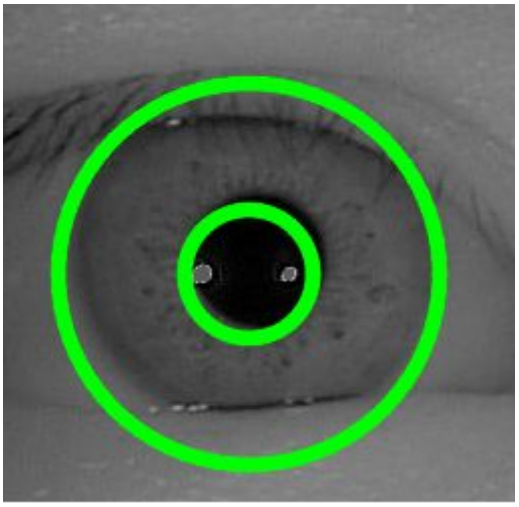

Figure 17: Both iris and pupil found by the Hugh Circles function.

3.6. Normalisation module

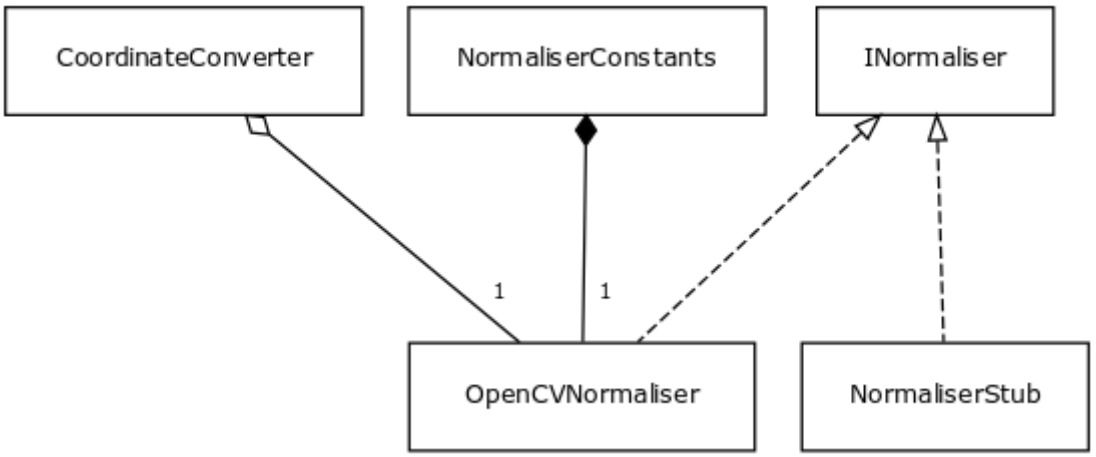

Figure 18: Diagram of the normalisation module.

```
public interface INormaliser {
   ModuleName moduleName = ModuleName.Normaliser;
   INormaliser INSTANCE =
           (INormaliser) PluginFactory.getPlugin(moduleName);

   ImageData normalize(ImageData imageData);

}
```

The normalisation module was designed to work under the assumption that at least one potential iris and pupil had been found. The *OpenCVNormaliser* class makes use only of the most likely iris and pupil coordinates.

Iris normalisation was based on Daugman's rubber sheet model [2]. A bilinear interpolation was performed in order to transform the iris image from Cartesian to polar coordinates. The end result should be a rectangle of size 512 x 64 pixels. As a test mentioned in the results chapter shows, a smaller, standard resolution

mentioned by Daugman (124 x 8 pixels) didn't result in a notable difference between the HD belonging to identical eyes and the HD belonging to different eyes. It might be connected to the resolution of the images used in the first place for iris localisation.

Function *normalise* assigns a pair of coordinates to every point of the circular iris according to the equation [2]:

$$x(r,\theta) = (1-r)x_p(\theta) + rx_s(\theta)$$
$$y(r,\theta) = (1-r)y_p(\theta) + ry_s(\theta)$$

The coordination reassignment is realised by class *CoordinateConverter*.

```
class CoordinateConverter {
     //https://www.ripublication.com/gjbmit/gjbmitv1n2_01.pdf -> publication with equations for normalisation

   private static double adjustR(double r, double height) {
         //normalisation because r is in range [0;rows*size] but the equation uses range [0;1]
       r = r / height;
       assert r >= 0 && r <= 1;
       return r;
   }

   private static double adjustTh(double th, double width) {
        //because th is in range [0;cols*size] but the equation uses range [0;2*pi] (radians)
       th = 2. * Math.PI * th / width;
       assert th >= 0 && th <= 2 * Math.PI;
       return th;
   }

   static Point toXY(double r, double th, Circle pupil, Circle iris, int width, int height) {
       r = adjustR(r, height);
```

```
            th = adjustTh(th, width);

            Point pupilPoint = pupil.pointAtAngle(th);
            Point irisPoint = iris.pointAtAngle(th);
            double x = (1 - r) * pupilPoint.x + r * irisPoint.x;
            double y = (1 - r) * pupilPoint.y + r * irisPoint.y;
            return new Point(x, y);
        }
}
```

Figure 19: This mask covers the areas where there should be eyelids and a pupil fragment.

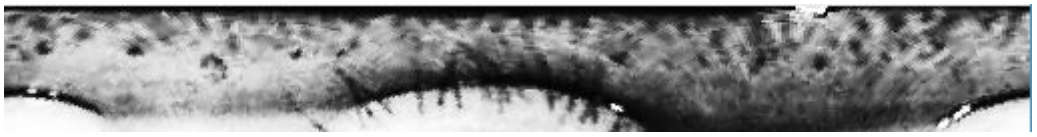

Figure 20: Iris pattern image before combining with the mask.

Figure 21: Fully normalised iris pattern image.

3.7. Encoding module

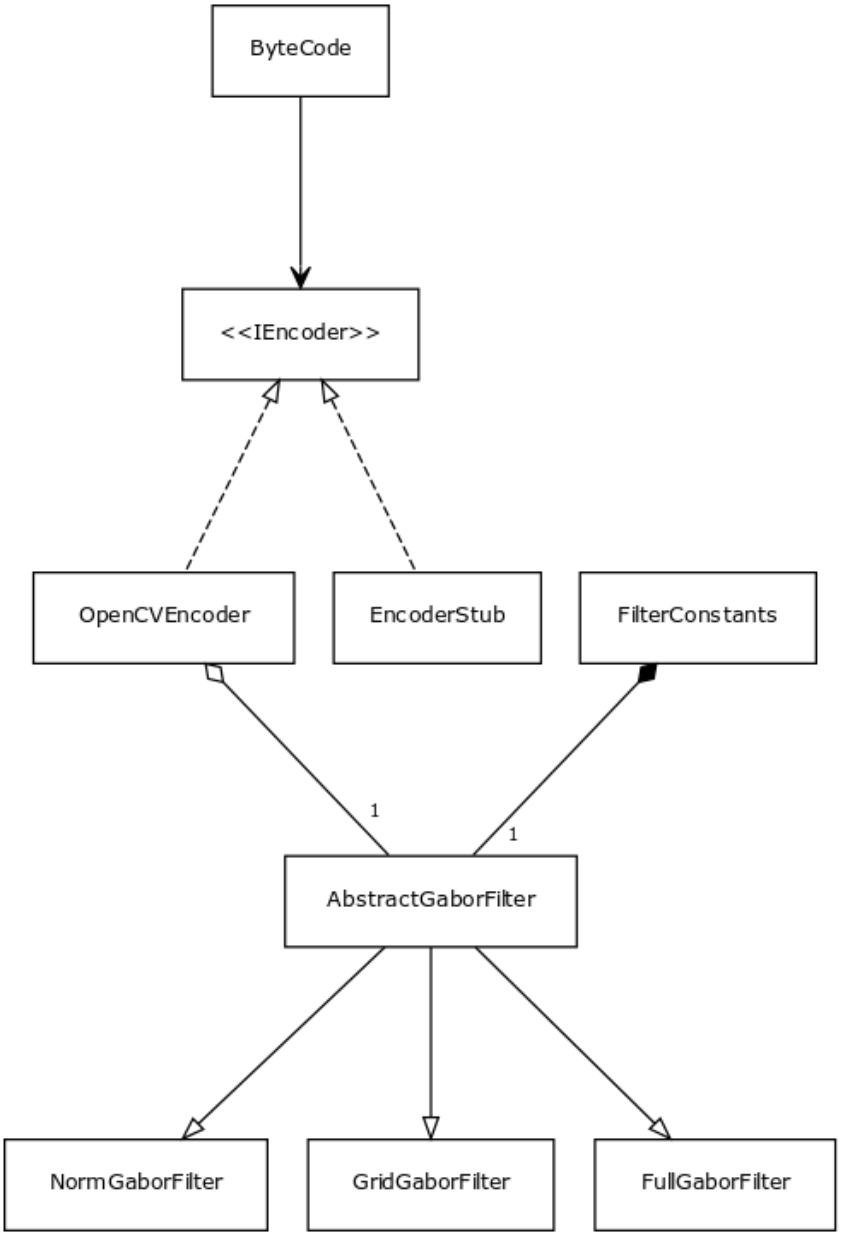

Figure 22: Diagram of the Encoding module.

The encoding module contains several versions of the encoding algorithm, all making use of the Gabor function supplied by the OpenCV library. These version provide varying degrees of success despite attempts to adjust the normalisation parameters. What these methods have in common is that the normalised image provided by the previous module is filtered by a Gabor filter and the result is then binarized with the threshold of 0 and translated into byte code (class *ByteCode*).

```
public interface IEncoder {
   ModuleName moduleName = ModuleName.Encoder;
   IEncoder INSTANCE =
           (IEncoder) PluginFactory.getPlugin(moduleName);

   ImageData encode(ImageData image);
}
```

The *Gabor* class is for the most part a translation of OpenCV's *getGaborKernel* function from C++ to Java. OpenCV's *getGaborKernel* unfortunately computes only real values, not complex. The imaginary part of the filter (function *getImaginaryComponent*) can be calculated by simply changing *cos* for *sin* in the equation.

```
public class Gabor {
   public static Mat getGaborKernel(Size ksize, double sigma, double theta,
                   double lambd, double gamma, double psi, int ktype, boolean real)
{
       double sigma_x = sigma;
       double sigma_y = sigma / gamma;
       int nstds = 3;
       int xmin, xmax, ymin, ymax;
       double c = Math.cos(theta), s = Math.sin(theta);

       if (ksize.width > 0)
```

```
            xmax = (int) ksize.width / 2;
        else
             xmax = (int) Math.round(Math.max(Math.abs((double) nstds * sigma_x * c),
Math.abs((double) nstds * sigma_y * s)));

        if (ksize.height > 0)
            ymax = (int) ksize.height / 2;
        else
             ymax = (int) Math.round(Math.max(Math.abs((double) nstds * sigma_x * s),
Math.abs((double) nstds * sigma_y * c)));

        xmin = -xmax;
        ymin = -ymax;

        assert (ktype == CV_32F || ktype == CV_64F);

        Mat kernel = new Mat(ymax - ymin + 1, xmax - xmin + 1, ktype);
        double scale = 1.;
        double ex = -0.5 / (sigma_x * sigma_x);
        double ey = -0.5 / (sigma_y * sigma_y);
        double cscale = Math.PI * 2. / lambd;

        for (int y = ymin; y <= ymax; y++)
            for (int x = xmin; x <= xmax; x++) {
                double xr = x * c + y * s;
                double yr = -x * s + y * c;
                double v;
                if (real)
                    v = getRealComponent(scale, ex, xr, ey, yr, cscale, psi);
                else
                    v = getImaginaryComponent(scale, ex, xr, ey, yr, cscale, psi);
                if (ktype == CV_32F)
                    kernel.put(ymax - y, xmax - x, (float) v);
                else
                    kernel.put(ymax - y, xmax - x, (double) v);
            }

        return kernel;
    }
```

```
   public static Mat getRealGaborKernel(Size ksize, double sigma, double theta,
                                double lambd, double gamma, double psi, int ktype)
{
       return getGaborKernel(ksize, sigma, theta, lambd, gamma, psi, ktype, true);
   }

     public static Mat getImaginaryGaborKernel(Size ksize, double sigma, double
theta, double lambd, double gamma, double psi, int ktype) {
       return getGaborKernel(ksize, sigma, theta, lambd, gamma, psi, ktype, false);
   }

   public static double getRealComponent(double scale, double ex, double xr, double
ey, double yr, double cscale, double psi) {
        return scale * Math.exp(ex * xr * xr + ey * yr * yr) * Math.cos(cscale * xr
+ psi);
   }

    public static double getImaginaryComponent(double scale, double ex, double xr,
double ey, double yr, double cscale, double psi) {
        return scale * Math.exp(ex * xr * xr + ey * yr * yr) * Math.sin(cscale * xr
+ psi);
   }
}
```

The result computed by *FullGaborFilter* is the direct output of a complex Gabor filter applied to the normalised image. In this case, the size of the normalised image and the size of the iris code are equal. More than one Gabor kernel can be generated, in which case all filter result matrices are combined by computing their element-wise mean.

GridGaborFilter is similar to the previous class except for that the normalised image is divided into segments. The mean of the filter values over the segment area is used to create the code. The iris code size is therefore smaller than the normalised (124x8, depending on the values assigned to filter constants).

Both of the previously listed filters make use of OpenCV's *getGaborKernel* function in order to produce results. *NormGaborFilter* on the other hand uses Gabor kernel values inspired by the OSIRIS project [11]. Another notable difference is that

the real and imaginary values are written into separate arrays for the sake of ease of observing results, instead of being treated as two consecutive bits of data, as Daugman recommends [2]. The size of the code in the final version of the project is 512x64.

```
filters.add( LoadFilter( new double[][] {    //[9][27]
    {0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,-0.50,-0.50,-0.50,-0.50,0.00,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,-0.50,-0.50,-0.50,-0.50,0.00,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,-0.50,-0.50,-0.50,-0.50,0.00,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,-1.00,-1.00,-1.00,-1.00,0.00,1.00,1.00,1.00,1.00,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50},
    {0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,-1.00,-1.00,-1.00,-1.00,0.00,1.00,1.00,1.00,1.00,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50},
    {0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,-1.00,-1.00,-1.00,-1.00,0.00,1.00,1.00,1.00,1.00,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50},
    {0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,-0.50,-0.50,-0.50,-0.50,0.00,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,-0.50,-0.50,-0.50,-0.50,0.00,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,-0.50,-0.50,-0.50,-0.50,0.00,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25}
}));
filters.add( LoadFilter( new double[][] {    //[9][51]
    {-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,
        0.50,0.50,0.50,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,
        0.50,0.50,0.50,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,
        0.50,0.50,0.50,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,1.00,1.00,1.00,1.00,1.00,1.00,1.00,1.00,1.00,1.00,
        1.00,1.00,1.00,1.00,1.00,1.00,1.00,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50},
    {-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,1.00,1.00,1.00,1.00,1.00,1.00,1.00,1.00,1.00,1.00,
        1.00,1.00,1.00,1.00,1.00,1.00,1.00,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50},
    {-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,1.00,1.00,1.00,1.00,1.00,1.00,1.00,1.00,1.00,1.00,
        1.00,1.00,1.00,1.00,1.00,1.00,1.00,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50},
    {-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,
        0.50,0.50,0.50,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,
        0.50,0.50,0.50,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,
        0.50,0.50,0.50,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25}
}));
filters.add( LoadFilter( new double[][] {    //[9][51]
    {0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,0.00,0.50,0.50,0.50,
        0.50,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,0.00,0.50,0.50,0.50,
        0.50,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,0.00,0.50,0.50,0.50,
        0.50,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,-1.00,-1.00,-1.00,-1.00,-1.00,-1.00,-1.00,-1.00,0.00,1.00,1.00,1.00,
        1.00,1.00,1.00,1.00,1.00,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50},
    {0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,-1.00,-1.00,-1.00,-1.00,-1.00,-1.00,-1.00,-1.00,0.00,1.00,1.00,1.00,
        1.00,1.00,1.00,1.00,1.00,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50},
    {0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,-1.00,-1.00,-1.00,-1.00,-1.00,-1.00,-1.00,-1.00,0.00,1.00,1.00,1.00,
        1.00,1.00,1.00,1.00,1.00,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50},
    {0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,0.00,0.50,0.50,0.50,
        0.50,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,0.00,0.50,0.50,0.50,
        0.50,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25},
    {0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,-0.50,0.00,0.50,0.50,0.50,
        0.50,0.50,0.50,0.50,0.50,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25,-0.25}
}));
```

Figure 23: Selected values used as Gabor wavelets [11].

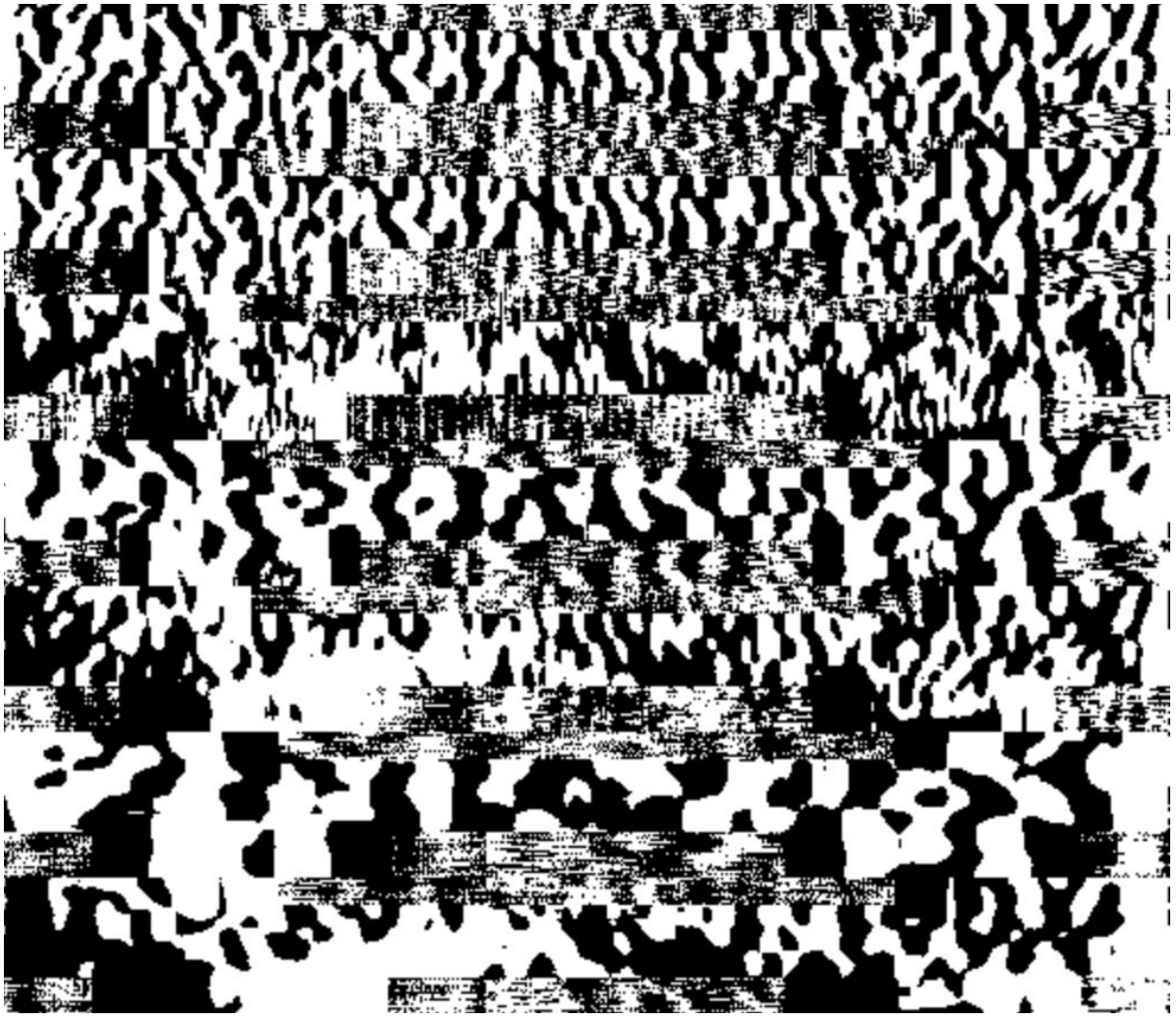

Figure 24: From top to bottom, results of encoding based on different Gabor kernels. In each pair the top half is the real part of the filter and the bottom part is imaginary.

3.8. Comparison module

```
public interface IComparator {

   ModuleName moduleName = ModuleName.Comparator;
   IComparator INSTANCE =
           (IComparator) PluginFactory.getPlugin(moduleName);

   HammingDistance compare(ImageData imageDataA, ImageData imageDataB);

   List<Mat> getPartialResults();
}
```

The *compare* method of the *IComparator* interface extracts already computed iris codes from two *ImageData* objects. The method *getPartialResults* is meant for testing purposes and is supposed to provide a view of intermediate stages of the comparison algorithm, as there is no guarantee the eyes in the image are tilted at the same angle.

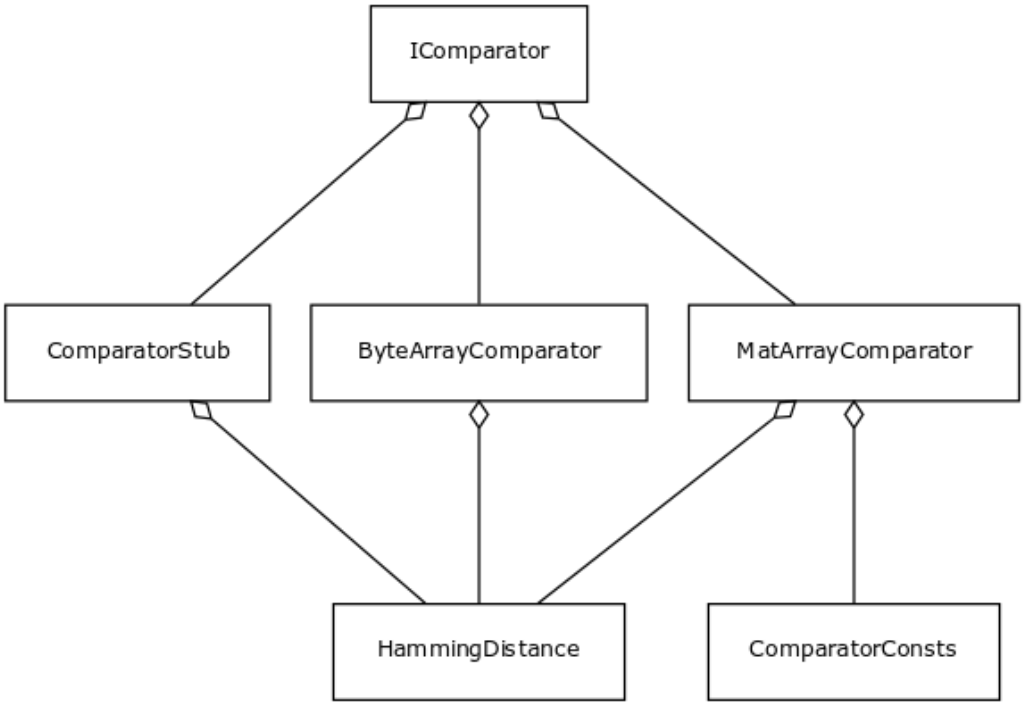

Figure 1: Diagram of the Comparator *module.*

Two classes under the *Comparator* module compute the Hamming Distance, each working on a different level of abstraction.

ByteArrayComparator works on byte array objects. This format makes it possible to save the iris codes as records in a database. Keeping the iris code in a database belongs is not yet implemented. *MatComparator* on the other hand works directly on *Mat*.

```
private double hammingDistance(Mat matA, Mat maskA, Mat matB, Mat maskB,
ComparisonType comparisonType ) {
   assert matA.size().equals(matB.size());

   double hd = 1.;
   int max_j = matA.width();
   int max_i = matA.height();

   boolean maskA_present = maskA!=null;
   boolean maskB_present = maskB!=null;

   int total = max_i*max_j;
   assert total > 0;

   int distance = 0;
   boolean a, b;
   for (int j=0; j<max_j; j++) {
       for (int i=0; i<max_i; i++) {
                   if (maskA_present && maskA.get(i,j)[0]==0 || maskB_present &&
maskB.get(i,j)[0]==0) {
               total--;
           }
           else {
               a = matA.get(i, j)[0] != 0;
               b = matB.get(i, j)[0] != 0;
               if (
                       comparisonType==ComparisonType.EQUAL && a==b ||
```

```
                    comparisonType==ComparisonType.NOT_EQUAL && a!=b ||
                    comparisonType==ComparisonType.XOR && a^b
                ) {
                distance++;
            }
        }
    }

    return (double) distance / (double) total;
}
```

The function *hammingDistance* implements the HD as given by Daugman in the equation [2]:

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

The Comparison module adds a modification to this method. The frame that encompasses the area under comparison is smaller than the actual iris code by a set of margin values: left margin = 40, right margin = 40, upper margin = 8, lower margin = 16. During the comparison the frame is subjected to both vertical and horizontal movement, making steps of one code unit.

This accounts for a possible mismatch during comparison of iris codes computed from the same areas of different codes. It is not guaranteed that rows with the same index number correspond to each other. This is dependent on how accurate was the iris edge localisation in the *Localisation* module.

4. Results

4.1. Iris verification

A **match** or an accepted comparison was defined as a comparison for which HD indicates the iris codes are sufficiently similar.

A **rejected match** or comparison was defined as a comparison for which HD indicates the iris codes are sufficiently different.

A **conclusive result** was defined as a match that could be considered either accepted or rejected.

An **inconclusive result** was defined as a match that could not be considered accepted or rejected.

4.1.1. HD range for accepting matches

In this experiment we investigated how the iris code comparison results are influenced by the Hamming Distance range. An any-to-any comparison was performed among 96 images (12 different eyes with 8 images each). Discounting the images that were rejected because no iris or pupil was found, a total of 8010 comparisons was made. The size of the iris code for this experiment was 512x64.

It was assumed that HD resulting from the comparison of iris codes generated from images of **different eyes** should fall into the HD range **0.36-0.50**. The HD resulting from the comparison of iris codes generated from different images of the

same eye should be in the HD range **0.0-N**. This made *N* the variable in this experiment.

Results: Table 1 shows the percentage of correctly found matches drastically decreased as the bound *N* was raised. The number of rejected matches was **constant** (3927); 93.48% of them were rejected correctly. The number of inconclusive matches decreased 4 times. These results show that adjusting HD bounds to match iris codes is an important factor in receiving satisfactory results.

Table 1:

HD	FOUND MATCHES	CORRECTLY FOUND MATCHES	CORRECTLY FOUND MATCHES (%)	INCONCLUSIVE
0.14	2	2	100	4081
0.15	2	2	100	4081
0.16	6	6	100	4077
0.17	10	8	80	4073
0.18	16	10	62.5	4067
0.19	28	14	50	4055
0.19	28	14	50	4055
0.21	81	22	27.16	4002
0.22	113	29	25.66	3970
0.23	176	36	20.45	3907
0.24	251	46	18.33	3832
0.25	346	53	15.32	3737
0.26	482	68	14.11	3601
0.27	660	83	12.58	3423
0.28	860	104	12.09	3223
0.29	1112	135	12.14	2971
0.3	1386	164	11.83	2697
0.31	1714	180	10.5	2369
0.31	1714	180	10.5	2369
0.33	2582	256	9.91	1501
0.34	3069	276	8.99	1014
0.35	3578	301	8.41	505

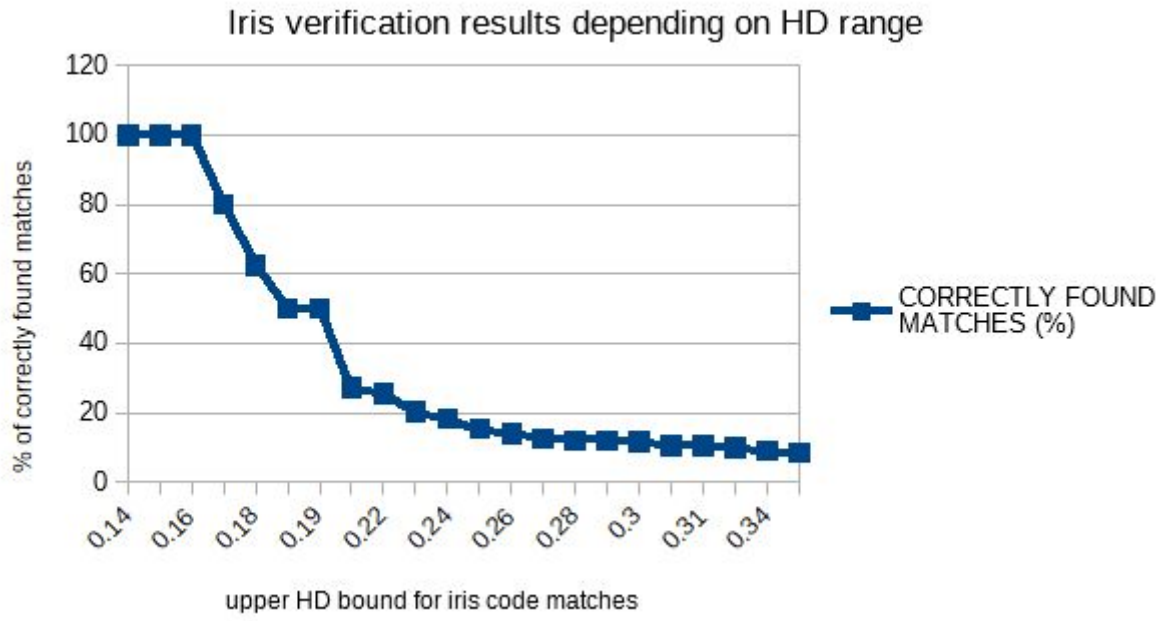

Figure 26.

4.1.2. HD range for rejecting matches

This experiment was analogous to the previous one (4.1.1). We investigated how rejection of matches during the iris code comparison is influenced by the Hamming Distance range.

The HD range for match **rejection** was set as **N-0.50**, making *N* the variable in this experiment. The HD range for match **acceptance** was **0.0-0.3**.

Results: Table 2 shows the number of rejected matches **decreased** approximately 20 times, but interestingly enough, the percentage of correctly rejected matches did not show a noticeable increasing or decreasing trend as the bound *N* was lowered. The number of inconclusive matches **increased** approximately 20 times. The number of accepted matches was constant (1386); 11.83% of them were accepted correctly. These results suggest that adjusting the value of N could help create a clear distinction between the criteria for iris code acceptance and rejection.

Table 2:

HD	REJECTED MATCHES	CORRECTLY REJECTED MATCHES	CORRECTLY REJECTED MATCHES (%)	INCONCLUSIVE
0.31	6296	5886	93.49	328
0.32	5881	5509	93.67	743
0.33	5429	5095	93.85	1195
0.34	4941	4627	93.65	1683
0.35	4432	4143	93.48	2192
0.36	3927	3671	93.48	2697
0.37	3388	3162	93.33	3236
0.38	2847	2656	93.29	3777
0.39	2320	2178	93.88	4304
0.4	1820	1716	94.29	4804
0.41	1393	1306	93.75	5231
0.42	1013	948	93.58	5611
0.43	622	574	92.28	6002
0.44	316	294	93.04	6308

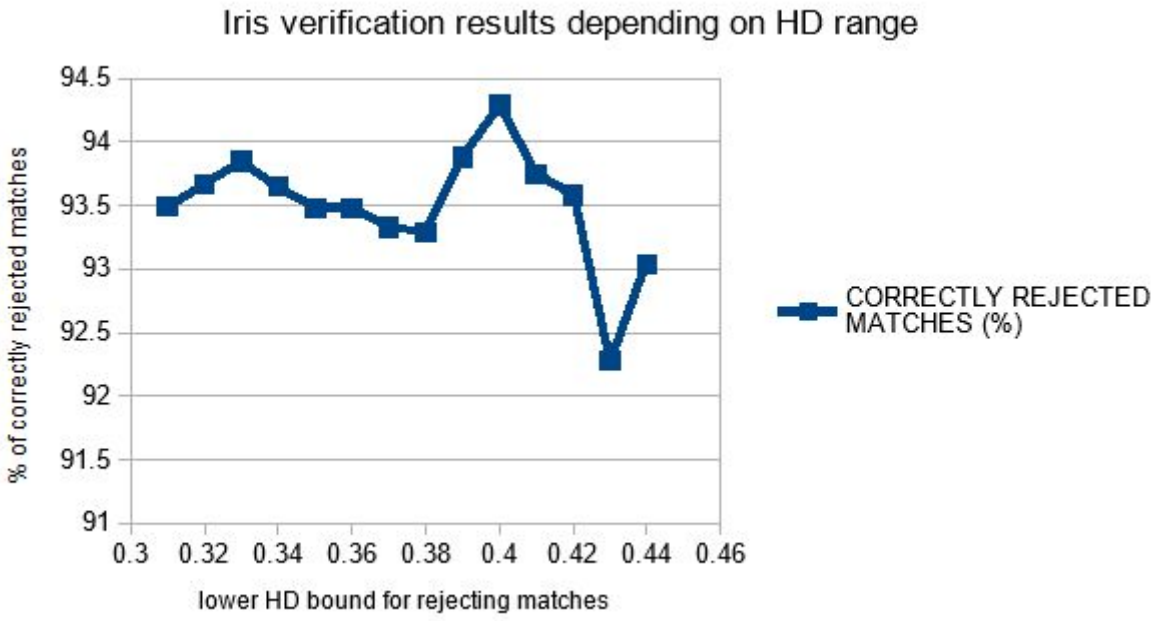

Figure 27.

4.2. Iris identification

Several experiments were run to determine the performance of the project. The tests were meant to simulate **iris identification** based on iris codes saved in a database. The scope of tests was restricted by the computation time necessary to run them. Additionally, if only a single image represented an iris, the number of tests with a definitive answer was insufficient (below 20%), so some modifications had to be applied.

To prepare data for testing, sets of Hamming Distance values alongside with relevant image data were calculated and saved into .sql format. The sets were the results of any-to-any comparisons within a given range. The files were afterwards queried using Transact-SQL.

Every person was represented by a group of iris codes generated from different images of the same iris. All iris codes were then compared against entire groups of codes.

For the following tests, a **match** was considered to be found for an iris when more than half of comparisons against a group of irises representing the same eye implicated on the basis of HD that the iris images were similar. A match was **rejected** when more than half of these comparisons were different. An **inconclusive** result happened when a match could neither be found nor rejected. The set of **correct results** encompasses both correctly found matches and correctly rejected matches.

4.2.1. HD range for accepting matches

This experiment was analogous to the one performed in section 4.1.1, however this was meant to simulate **iris identification** instead of verification.

It was assumed that HD resulting from the comparison of iris codes generated from images of **different eyes** should fall into the HD range **0.36-0.50**. The HD resulting from the comparison of iris codes generated from different images of the **same eye** should be in the HD range **0.0-N**. The number of persons in this test was 12. Per each person 8 images of the same eye were used, which meant that at least 4 comparisons within a group had to agree for a result to be declared accepted or rejected. For each tested HD value **1080 comparisons** of iris codes were performed in total. 96 of these comparisons were between images of the same iris. The size of the iris code for this experiment was 512x64.

Results: While the total percentage of correct results (C) rose slightly as N increased and the percentage of inconclusive results rapidly declined, more attention should be

paid to the percentage of errors (E). When N=0.35, E was nearly equal to C, confirming that bound N is a major factor in avoiding false matches.

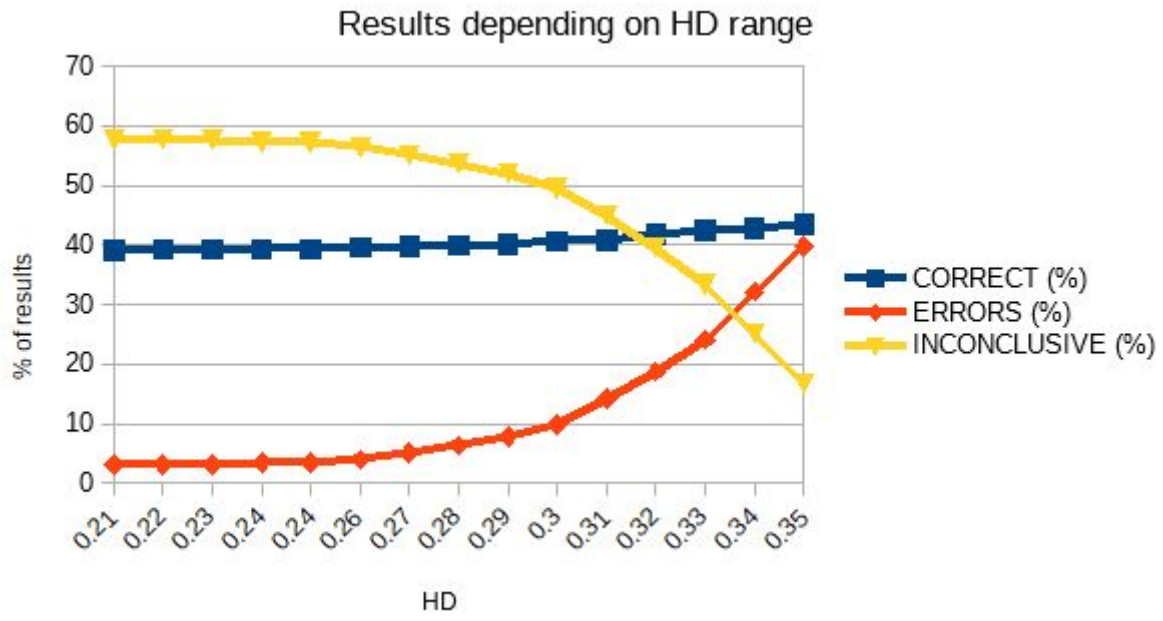

Figure 28: HD is the lower bound of the range used to confirm that iris patterns are different.

Table 3:

HD	CORRECT	ERRORS	CORRECT (%)	ERRORS (%)	INCONCLUSIVE (%)
0.21	423	33	39.17	3.06	57.78
0.22	424	33	39.26	3.06	57.69
0.23	424	33	39.26	3.06	57.69
0.24	425	37	39.35	3.43	57.22
0.24	425	37	39.35	3.43	57.22
0.26	427	43	39.54	3.98	56.48
0.27	430	55	39.81	5.09	55.09
0.28	431	69	39.91	6.39	53.7
0.29	433	85	40.09	7.87	52.04
0.3	439	106	40.65	9.81	49.54
0.31	441	154	40.83	14.26	44.91
0.32	451	202	41.76	18.7	39.54
0.33	459	260	42.5	24.07	33.43
0.34	462	347	42.78	32.13	25.09
0.35	470	430	43.52	39.81	16.67

4.2.2. HD range for rejecting matches

The focus of this experiment was to observe the effect of changing the HD range used as a criterion for rejecting matches. It is analogous to previous section (4.2.1).

The HD range for **accepting matches** was **0.0-0.3** and for **rejecting** matches **N-0.5**. The rest of the parameters were the same as in the previous test (4.2.1).

Results: As N rises, the percentage of correct results decreased. This was caused directly by simultaneous increase of inconclusive results, which happened at a similar rate. Total percentage of errors decreased only slightly.

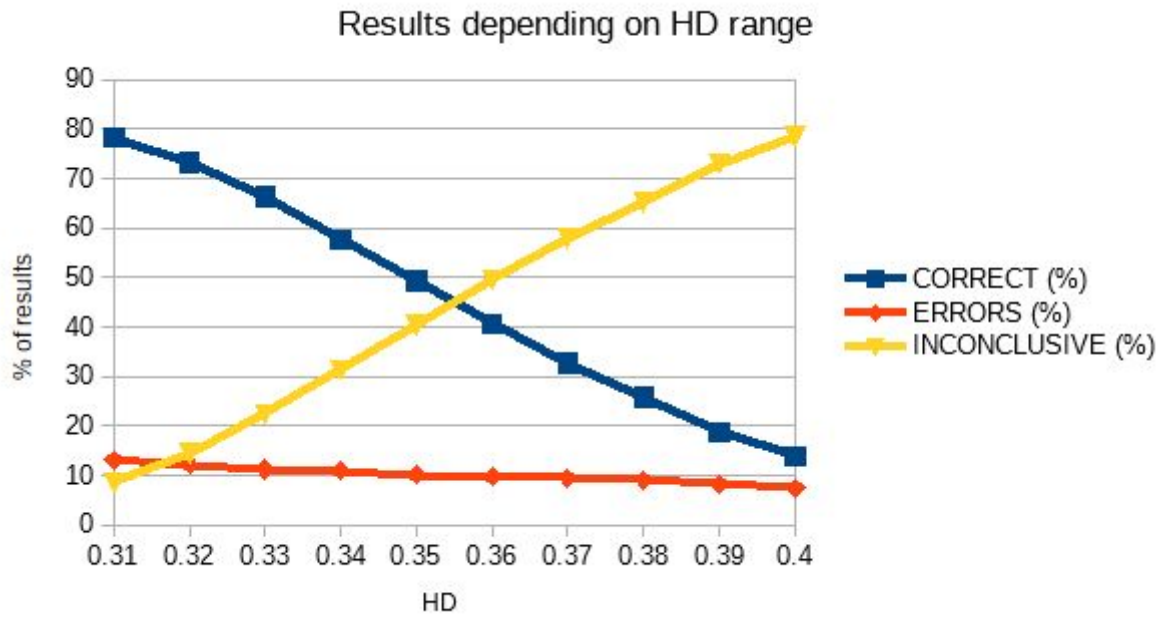

Figure 29: HD is the upper bound of the range used to confirm that iris patterns correspond to the same eye.

Table 4:

HD	CORRECT	ERRORS	CORRECT (%)	ERRORS (%)	INCONCLUSIVE (%)
0.31	845	142	78.24	13.15	8.61
0.32	791	132	73.24	12.22	14.54
0.33	716	121	66.3	11.2	22.5
0.34	623	118	57.69	10.93	31.39
0.35	533	110	49.35	10.19	40.46
0.36	439	106	40.65	9.81	49.54
0.37	354	102	32.78	9.44	57.78
0.38	279	98	25.83	9.07	65.09
0.39	203	89	18.8	8.24	72.96
0.4	151	81	13.98	7.5	78.52

4.2.3. Results by total number of comparisons

In the following experiment we investigated how the results were influenced by the number of comparisons performed and the number of persons. The rest of the parameters were similar to the previous test (4.2.2).

The variable in this case was the total number of performed comparisons (dictated by the number of persons in the database). The HD ranges used for determining different and same irises were respectively **0.36-0.50 and 0.0-0.3.**

Results: On average 44% of comparisons had an inconclusive result. This accounts also for these images where the iris boundary could not be determined. Chapter 4.2.4. shows how the percentages derived from these results relate to each other.

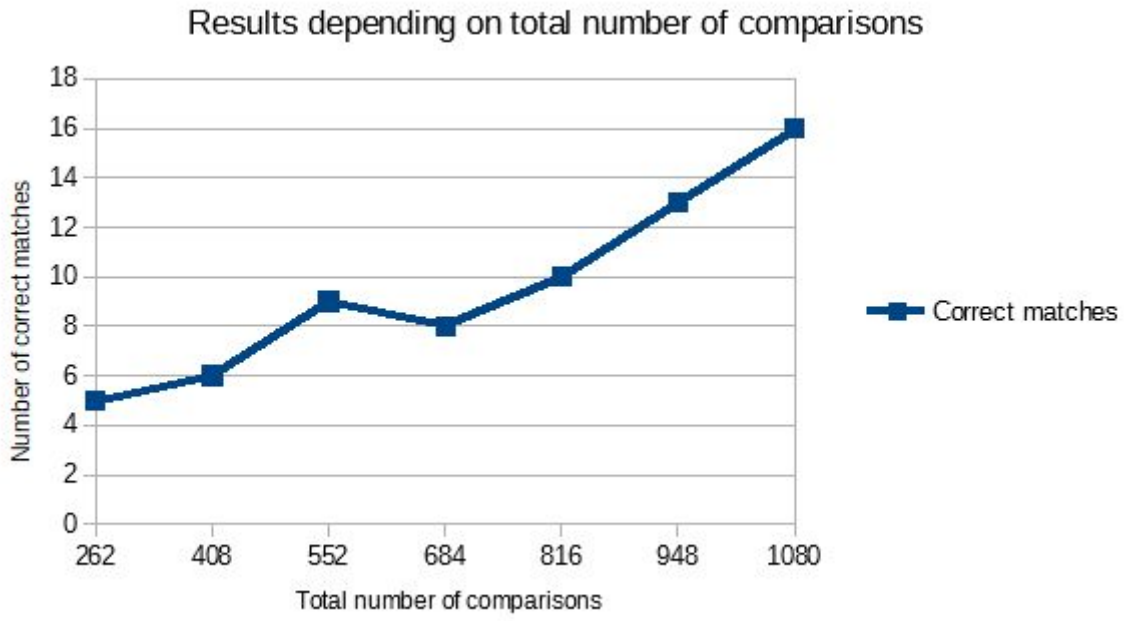

Figure 30: Number of correctly accepted matches depending on the total number of comparisons between 12 persons.

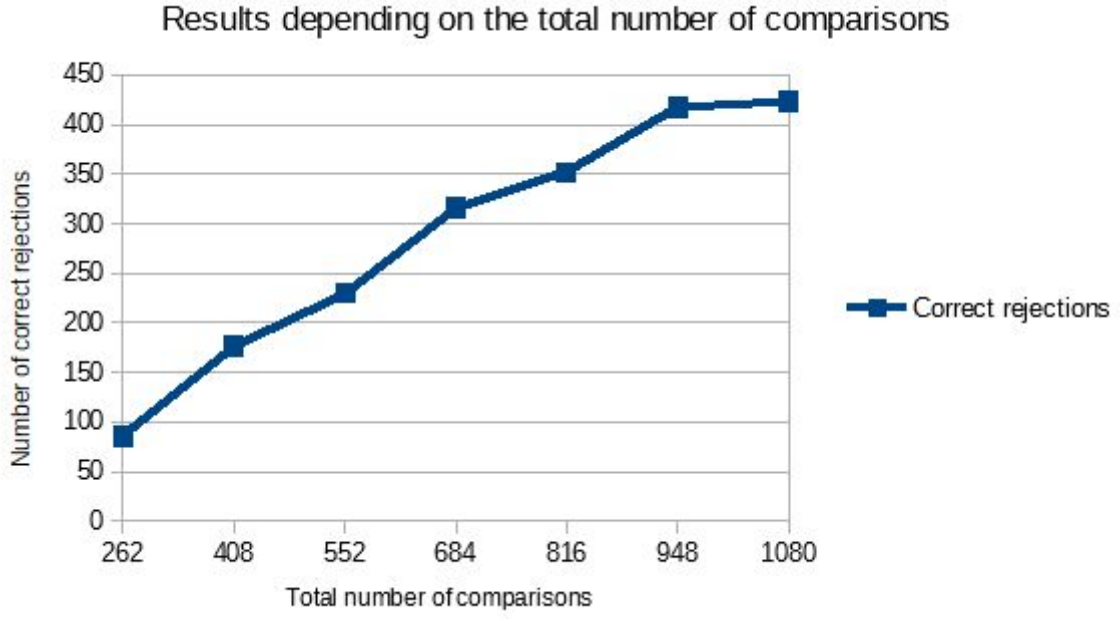

Figure 31: Number of correctly rejected matches depending on the total number of comparisons between 12 persons.

Table 5: 6 persons

Number of images in a group	Total comparisons	Correct matches	Correct rejections
2	65	4	15
3	102	6	46
4	138	7	60
5	174	5	82
6	210	7	90
7	246	8	115
8	276	9	103

Table 6: 9 persons

Images in a group	Total comparisons	Correct matches	Correct rejections
2	152	4	40
3	234	6	102
4	315	9	120
5	387	7	166
6	459	10	186
7	531	12	214
8	603	14	206

Table 7: 12 persons

Number of images in a group	Total comparisons	Correct matches	Correct rejections
2	262	5	86
3	408	6	177
4	552	9	230
5	684	8	316
6	816	10	352
7	948	13	417
8	1080	16	423

4.2.4. Results by total number of comparisons in percentages

The parameters were as in the previous test (4.2.4), but percentages were taken into account instead of bare numbers.

Results: Comparing tables 8, 9 and 10 demonstrates that so far the error rate did not show a consistent upward or downward trend. In all three tables, however, the error rate was the smallest where a single image was compared against a group of exactly 5 images. Groups with an odd number of images showed a better percentage of correct results than groups with an even number of images.

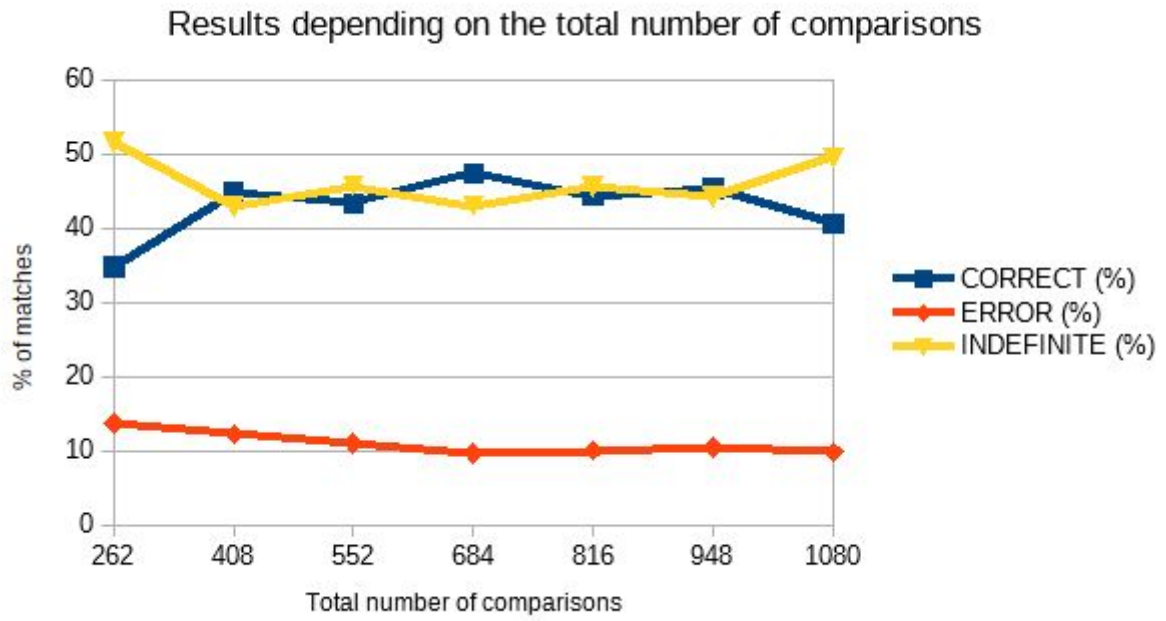

Figure 32: Percentages of correctly accepted matches depending on the total number of comparisons between 12 persons.

Table 8: 6 persons

Images in a group	Total comparisons	CORRECT (%)	ERROR (%)	INDEFINITE (%)
2	65	29.23	27.69	43.08
3	102	50.98	14.71	34.31
4	138	48.55	15.22	36.23
5	174	50	9.77	40.23
6	210	46.19	13.33	40.48
7	246	50	13.41	36.59
8	276	40.58	11.96	47.46

Table 9: 9 persons

Images in a group	Total comparisons	CORRECT (%)	ERROR (%)	INDEFINITE (%)
2	152	28.95	19.08	51.97
3	234	46.15	14.1	39.74
4	315	40.95	12.06	46.98
5	387	44.7	9.04	46.25
6	459	42.7	10.02	47.28
7	531	42.56	11.86	45.57
8	603	36.48	10.45	53.07

Table 10: 12 persons

Images in a group	Total comparisons	CORRECT (%)	ERROR (%)	INDEFINITE (%)
2	262	34.73	13.74	51.53
3	408	44.85	12.25	42.89
4	552	43.3	11.05	45.65
5	684	47.37	9.65	42.98
6	816	44.36	10.05	45.59
7	948	45.36	10.44	44.2
8	1080	40.65	9.81	49.54

4.3. Summary

The percentage of accepted and rejected matches can be manipulated through adjusting the Hamming Distance ranges which define them. Given the right parameters, about 56% of comparisons had a conclusive result. There appears to be a direct correlation between the percentage of inconclusive results and the error percentage. During the iris identification simulation, the error percentage seemed to be the smallest when exactly 5 images were assigned to each person.

5. Discussion

The primary goal of this project has been met - laying the foundation for an expandable framework for iris recognition, based on open source solutions. However, during writing the code of this project, several difficult steps were encountered, which opens the way for discussion of suggestions for further expansion of the project.

This project was designed under the assumption that database access would eventually be added, enabling the storage of iris codes and allowing comparisons to be run against them. For the time being the Writer module is a placeholder containing only a stub class, which is a class that does not actually implement any function, but was inserted to maintain structural integrity of the project's design.

During the process of creation of the project it became clear that the correct localisation of iris edges was a definite necessity. Hugh transform, whose implementation is available in the OpenCV library, was used to simplify the localisation process. To maximize the accuracy of localising the iris and pupil edges, an alternative algorithm could be implemented instead.

On a similar note, a more precise mask detecting light reflections, eyelashes and similar noise data is necessary to increase the precision of iris comparison. High resolution images would also be preferable, as normalisation causes further distortion of images. Using higher resolution images might make it possible to make the code size smaller, which would in turn greatly improve the computation time.

Apart from localisation of iris edges, the greatest influence on the comparison result belongs to the Gabor filter, in particular to the initialisation of Gabor kernels. The amount of applied Gabor filters is restricted by the need to strike a balance between comparison accuracy and computation time. Finding optimal parameters for the enhancement of the iris pattern could be a separate problem to be examined.

To summarize, the primary points of further interest are the improvement of iris edge localisation, creation of a more detailed code mask and the analysis of the

influence of Gabor filter parameters on the results. These suggestions are however beyond the scope of this project.

6. Bibliography

[1] OpenCV 3.0 documentation. *Feature Detection*. http://docs.opencv.org/3.0-beta/modules/imgproc/doc/feature_detection.html. Accessed 10 Sep 2017.

[2] Daugman, John, *How Iris Recognition Works,* 2004.

[3] Institute of Automation, Chinese Academy of Science: *Note on CASIA-IRISV4*. http://biometrics.idealtest.org/findTotalDbByMode.do?mode=Iris. Accessed 10 Sep 2017.

[4] Sunil Chawla, Aashish Oberoi, *A Robust Algorithm for Iris Segmentation and Normalization using Hough Transform*, 2011. https://www.ripublication.com/gjbmit/gjbmitv1n2_01.pdf. Accessed 10 Sep 2017.

[6] Stan Z. Li, *Encyclopedia of Biometrics*, Chapter: *Iris Recognition at Airports and Border-Crossings*, Springer Publishing Company, Incorporated, 2010

[7] Institute of Automation, Chinese Academy of Science: CASIA-IrisV4 database. http://biometrics.idealtest.org/. Accessed 10 Sep 2017.

[8] Computer Vision Research Laboratory, University of Notre Dame. https://sites.google.com/a/nd.edu/public-cvrl/data-sets. Accessed 10 Sep 2017.

[9] Department of Computer Science, University of Beira Interior: UBIRIS. http://iris.di.ubi.pt/. Accessed 10 Sep 2017.

[10] OpenCV 3.0 documentation. *Image Filtering.* http://docs.opencv.org/3.0-beta/modules/imgproc/doc/filtering.html. Accessed 10 Sep 2017.

[11] Nadia Othman, *A biometric reference system for iris OSIRIS version 4.1.*

[12] OpenCV 3.0 documentation. *Basic Structures*. http://docs.opencv.org/3.0-beta/modules/core/doc/basic_structures.html, Accessed 10 Sep 2017.