Concurrent Image Processing with emphasis on Fingerprint Anlaysis

by Edoardo Foco

Concurrent Image Processing with emphasis on Fingerprint Enhancement

Edoardo Foco

Supervisor: Sean Tohill

Date: 29th April 2013

Keywords: Image Processing, Parallel Image Processing, Multithreading, Fingerprint Analysis, Fingerprint Enhancement, Edge Detection, Python, Spatial Filters, Numpy, SciPy, OpenCv

Abstract: This report contains the design and implementation of three systems targeting image analysis algorithms and high performance computing applied to fingerprint enhancement. The systems are executed from a command line and display the algorithms results. Designed using UML and developed using Python 2.7 it operates on Mac OS-X.

This report is submitted in partial fulfilment of the requirements for the BEng (Hons) Software Engineering Degree at the University of Westminster

Acknowledgements

The author's gratitude goes to his supervisor Sean Tohill who introduced him to this thesis and guided him through the project. Very special thanks also to Simone Rossi for his contributions and advice. Finally the author would like to thank Maurizion Foco, Simona Rossi, and Roberto Fiormonte for their support throughout these years of University.

Figures

Tables

Table of Contents

Chapter 1: Introduction

Automated biometrics applications have been developed since the early 90's and are used for personal identification worldwide. Fingerprint analysis is the most used biometric application, it relies on extracting features from a fingerprint. As of today, it is the most studied field among the forensic sciences because of the variety of implementations it can count, the aim of this chapter is to introduce the reader to the project and to the art of fingerprint analysis explaining the evolution and the basics of such science.

1.1 Expected Reader Level

This thesis does not require the reader to have any knowledge of fingerprint analysis as all arguments related to this science are explained throughly in the next chapters. Although this guide also explains some fundamentals on image processing, the ideal reader will already have some experience in this field and should also have some knowledge of software engineering, focusing especially on concurrent programming, programming languages, unified modelling language (UML).

1.2 Assumptions

Fingerprints analysis is used every day in airports, forensics laboratories, and security systems. Fingerprints are, in fact, one of the most reliable methods of verifying someone's identity, they are unique to each person, they remain unchanged during an individual's lifetime and the patterns formed from the ridges can be systematically classified. The National Institute of Science and Technology (NIST) has developed a Biometric Image Software (NBIS) which complies to the FBI standards. The algorithms used in the NBIS though are not concurrent and therefore can be enhanced to meet higher performance proving fingerprint recognition software are still a subject of study.

1.3 Biometrics

Biometrics identification systems are automated computerized systems used to identify a person from his unique characteristics or traits. Such systems rely on biometric identifiers which can be classified as physiological and behavioural characteristics. Physiological biometric identifiers include voice, DNA, face, iris, retina, and fingerprint recognition while behavioural biometrics identifiers are related to the behaviour of an individual and therefore focus on recognizing, for example, the typing rhythm, the gait, and in some cases the voice. Biometric technologies provide secure identification and personal verification solutions, a number of these applications are used today in many fields ranging from access control to airport security to forensics."Although biometrics emerged from its extensive use in law enforcement to identify criminals (e.g., illegal aliens, security clearance for employees for sensitive jobs, fatherhood determination, forensics, and positive identification of convicts and prisoners), it is being increasingly used today to establish person recognition in a large number of civilian applications"[1].

1.4 Fingerprint Analysis

Human fingertips are fully developed after the seventh month of fetal development. The ridges form unique patterns that do not change throughout the life of an individual unless they are altered by scars caused by cuts or bruises on the fingertips. Fingerprints are prints of a person's fingertips and therefore are a record of the unique patterns formed by the ridges. Today, fingerprint analysis out stands all other forensic sciences due to the following reasons [1]:

- It has been used for more than 100 years by governments all over the world as the main method to identify criminals.
- No two fingerprints have been found alike and therefore is an optimal solution for accurately identifying a person.
- Is the most used forensic evidence worldwide.

- The database continues to expand with thousands of new entries added to the repository each day only from US borders.

Fingerprint analysis relies on extracting the unique features formed by ridges and compare them against a database to find a match. In order to do this there are two main challenges that have to be considered: classifying the fingerprint, enables the search to consider only a few candidates excluding the fingerprints that do not belong to the same type, and extracting the minutiae, which have to be extracted meticulously avoiding false results.

1.4.1 History of Fingerprinting

The study of fingerprints as a mean of personal identification finds its origins in 1880 with the publication of an article in the scientific journal *"Nature"* written by Henry Faulds in which the author introduces fingerprints as a method to identify criminals. Faulds's work was not accurate enough to be considered as a study, but it did however open the waters to Sir Francis Galton, a British anthropologist and cousin to Charles Darwin that in 1892 published *"Finger Prints"* the first book that identifies the unique characteristics of fingerprints described as minutiae, also known as Galton's Details. Galton's work was then examined more in depth by Sir Edward Henry, a Police Inspector in Bengal, India, who developed the first system of classifying fingerprints in 1901, his system was immediately adopted as the official system in England and later spread throughout the whole world. Fingerprinting finally became the main method of personal identification in 1903 when the current identification system based on Bertillion's measurements failed to identify two American inmates which were later correctly identified using the modern technique. Fingerprint identification began its transition to automation in the late 1960. The first automated identification system was in fact developed by NIST for the Federeal Bureau of Investigation (FBI) Automated Fingerprint Identification System (AFIS) in 1969. In the early 90's NIST scientists evolved the AFIS project to enable the electronic exchange of fingerprints records and images by law enforcement agencies, the Integrated Automated Fingerprint Identification System (IAFIS) is currently operational in West Virgina [2].

Similar systems have been developed and are currently used in a variety of fields all around the world.

1.4.2 Fingerprint Classification

Due to the increasing number of fingerprint records in the filesystems it soon became necessary to develop a system to classify fingerprints. In 1897 Sir Richard Henry introduced a classification method that is still widely used today. Henry's method involved identifying the general ridge patterns which he classified in three major categories [3]:

- *Loops:* Constitute 60% to 65% of all fingerprints. Their characteristic resides in having one or more ridges entering from on side, curving and exiting from the same side. This category includes two sub-categories: the ulnar loop, which opens towards the little finger, and the radial loop, which opens towards the thumb.
- *Whorls:* Account for 30% to 35% of all fingerprints. Their characteristic is having two deltas and a ridge forming a complete circuit. This category may be divided into four sub-categories: plain whorl, central pocket, double loop, and accidental.
- *Arches:* Constitute 5% of all fingerprints. Arch ridges enter from one side of the print and exit on the opposite side. This category includes: plain arches, and tented arches.

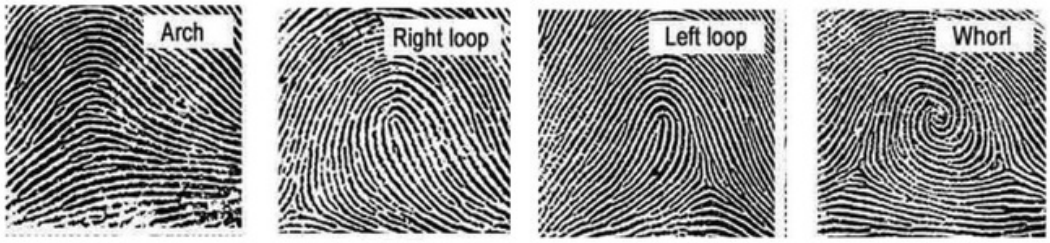


Figure 1.1. Fingerprint Classification: Arch, Right Loop, Left Loop, Whorl

1.4.3 Minutiae Extraction

Minutiae are defined as the unique patterns formed from an individual's fingertips ridges. Such characteristics are so unique that no two fingerprints have ever been found alike and that's a considerable record since fingerprint databases count billions of entries. A single fingerprint can count more than 100 minutiae and although they appear in various forms they can be classified in two main types: ridge ending (point where ridge terminates) and ridge bifurcation (point where the ridge forks or branches into more ridges). The rest of the minutiae result as combinations of these two types. Once the minutia has been identified, it is recorded as a combination of its location (in form of x, y coordinates on the fingerprint image) and its orientation (θ) [4]. The process of extracting minutiae must be very accurate as these are going to be later used to match the fingerprint against other candidates, therefore this process requires a further trimming action to delete the false results.

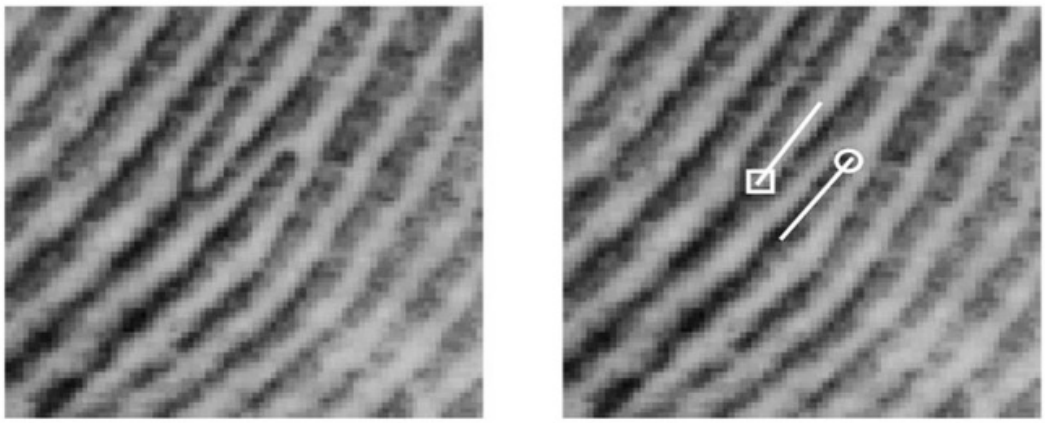


Figure 1.2. Minutiae: bifurcation (square marker) and ridge ending (circle marker)

The process of extracting minutiae is highly affected by the quality of the image, it is necessary to enhance the image as much as possible in order to reliably extract minutiae, in most cases minutiae are extracted from a binary representation of the original fingerprint.

This project will focus on the fingerprint enhancement phase as this is the step that effectively allows minutiae extraction algorithms to be efficient and reliable.

1.5 Conclusion

Automated fingerprint identification systems are key software products and fingerprint analysis is a science which potential has not yet been fully unlocked. The most accurate fingerprint recognition software was developed by NIST scientists without using concurrency, therefore fingerprint recognition solutions are still a subject of research, a new programming approach could greatly enhance the performance of such software. The aim of this project is to develop parallel image processing skills and apply them to fingerprint enhancement, the project will cover general image processing algorithms applied to fingerprints.

Chapter 2: Further Research

When developing a project it is essential to make some decisions and establish a project plan. The aim of this chapter is explain the decisions that have been taken during the planning phase.

2.1 The System Architecture

Analysing an image is a task which requires steps to take place before and after the actual feature extraction occurs. Generally speaking the entire process takes the form described in Figure 2.1. In the particular case of fingerprint analysis the pre-processing stage consists in enhancing the image to refine, and in some cases reconstruct, the ridges. This process is fundamental to increase the reliability of the feature extraction. The objective of this phase is to return a binary image where all ridges are well defined. The systems that were created all target the enhancement process focusing on the analysis of some algorithms that can accomplish this task.

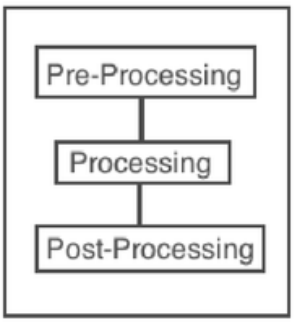


Figure 2.1: Analysis Process

2.2 User Experience

The user experience is one criteria used to assess the quality of a system and often affects how attractive the product is on the market. Most people prefer using a GUI rather than a CLI, this is because a GUI is much more intuitive and sometimes faster to work with. Lets take as example how a normal user browses a system, using a GUI the user will usually clicks on the folders he wants to browse until he finds the file he's looking for, double click and open it with the default application. Using a CLI the user would have to execute a

command each time he has to change folder and list the files executing another command, apart from the fact that these operations are time consuming and not at all intuitive the displayed results are showed in a list which makes them hard to identify if the directory contains many files. Furthermore the files are listed in alphabetical order and it is sometimes more useful to display them in another order such as the date they have last been modified, this operation would be at one click distance using a GUI making the user experience less stressful. Although a GUI would be preferable also in a Fingerprint identification System, the author decided to use a CLI because he retains that a high performance software should not be slowed down by an elaborated interface, furthermore a graphic user interface would be useless to a further objective of the project of detecting minutiae.

2.3 Choice of Language

Choosing a programming language for a High Performance Computing (HPC) software is a very delicate and critically important phase. An HPC developer will usually choose the programming language based on the specific requirements of the software to be developed. This particular project is based on applying biometrics algorithms to enhance a fingerprint image in the least possible time, hence the need to find a language which supports tools (libraries) for image processing whilst delivering a high computational performance.
When it comes to High Performance Computing for Biometric Software the debate usually comes down to a few options: C/C++, C#, Java, Fortran, Python, and Matlab.
The first comparison to be made was between C++ and C#/Java [5][8]. While C++ and C are low-level languages and therefore generate binary-code executables, C# and Java are high-level languages that produce byte-code which needs to be translated by a virtual machine before being executed on the physical machine. This intermediate process is said to slower down Java's and C# performances resulting in C and C++ being able to do faster computations. This is true to a certain extent. In fact, interpreted languages have the ability

to *'learn'* what the code is going to do before executing it and this allows the virtual machine to greatly improve its performance. Java supporters predict that Java will soon become faster than C and C++ because of the following reasons:

- **Managing Pointers –** C and C++ are known to generate errors when dealing with pointers while Java, thanks to the JIT compiler, is able to handle the pointers in a much better way.
- **Garbage Collection** – This utility is provided by the virtual machine in Java and C#. The garbage collector provides memory allocation information so that during run time the software does not have to look for memory because it already has all the information of the available memory slots. This utility also rearranges the memory to provide better resource management.
- **Run-time Compilation** – Because of the Java and C# virtual machine ability to retrieve useful information before the code is executed, the JIT compiler is able to optimize the code to work on the specific hardware. For example if the machine is running a PIII the compiler will optimize the code specifically for that processor.

Finally C and C++ prove themselves to be faster in terms of computation time but still cannot achieve a definitive superiority against Java and C# because of their inability to optimize code execution at runtime. High performance computing though relies on fast computation times and this led the author to consider other programming languages such as Python and Fortran.

Fortran's strength is that being a compiled programming language it produces binary-code which makes it faster compared to Python. Both these languages share some features such as the ability to easily call C functions and have a long history in HPC. Fortran in particular was designed to support scientists in numerical computing, in fact its design makes it the fastest for this type of computations. Python, on the other hand, proves itself to be slower than C, C++ and Fortran. Then why consider this option? Python is able to improve its speed

by calling both C functions and Fortran functions [6], yet it benefits from all the scripting languages features and this makes it extremely flexible. Pythons' major strength is in providing a hybrid solution to interpreted and compiled languages. On top of this, Python supports Object Oriented programming and is usually preferable to Fortran because of its portability, scalability, ease of testing, and faster coding. In fact, the code generated in Fortran is generally more complicated and hard to read. Furthermore Python has a larger amount of libraries to support the software requirements.

Matlab is still the most used language for biometrics software development [9][12] because it is provided with an extensive library that includes the best tool set to work with multi-dimensional arrays and plots and the software also comes with an IDE and Simulink which is a tool that simply cannot be found elsewhere. So what's wrong with Matlab?

- **Algorithms are Proprietary** which means that the code underneath them cannot be seen and therefore the developer finds himself in a position in which he has to trust Matlab and cannot verify the algorithms himself.
- **Matlab is Expensive**. This not only imposes cost constraints but also affects its portability. In fact, since a new version of Matlab is released every 6 months and new features must be implemented in order to render the product commercially appealing, the software's portability is compromised in the way that the application must be exactly the same version of the installed MCR (Matlab's portability solution). This forces the developers to check if their application is supported every time a new version is released.
- **No Third Party Tools**. Due to its proprietary nature third party developers have difficulties writing tools that support Matlab's functions.

Even though the Python development environment requires more time to set up due to the fact that it does not come with any IDE and each library must be installed before being able to use it, Python provides a very powerful alternative to Matlab. Using libraries such as

Numpy, SciPy and MatplotLib allows Python developers to use similar functions as Matlab, furthermore Python is able to use Matlab functions using a wrapper that enables Matlab code to be run in Python.

Finally the author decided to use Python not only because its' features appear to suit the requirements the most, but also because Python seems to be conquering the High Performance Computing environment [7].

2.4 Selecting a Software Development Lifecycle

Common software development life cycles such as Waterfall and the V-Model were retained to be unsuitable for this project due to the following reasons:

1. The author had no experience in image processing and concurrency, therefore a significant amount of prototyping was expected throughout the project. It was foreseen that a fair amount of techniques and approaches were to be experimented prior to attempt the final deliverable.
2. The algorithms analysed to be either incorporated in the final deliverable or discarded had to be tested thoroughly, it was foreseen that it would have been necessary to develop other systems to test the components and decide which ones to use.
3. The research phase highlighted the fact that the project would have been highly affected by time constraints. It was therefore decided to produce prototypes of the system so to have working sub-systems in case the project didn't reach conclusion.

It was finally decided to use a Rapid Application Development (RAD) approach in order to be able to implement various algorithms and prove their functionality. RAD allows the developer to create sub-systems and easily discard them if they cannot be implemented in the final solution.

2.5 The Building Blocks

Tackling an image processing software to improve its performance requires an exhaustive research in the areas of image processing, concurrent programming, and the fingerprint analysis algorithms. The aim of this phase is to develop expertise in the topics to understand what affects the processing speed and how this can be improved. This chapter is going to discuss the main topics that where encountered throughout the research, explaining why they have been considered relevant and what conclusions the author derived from them.

2.5.1 Concurrency

Image processing may sometimes be computationally very expensive depending on the image size and on the operations performed on such image. To achieve high performance a developer must consider to what extent modern hardware can enhance the speed and how to optimize the code to make the best use of such hardware. Multi-threading is a technique used to distribute the workload over the multiple CPU cores, once a thread is initiated the operating system will automatically take care of distributing the thread to the next available processor. A parallel image processing approach is preferred when computing a spatial analysis on the image which is so divided in blocks of size *w* x *w*, each block is then assigned to a thread which computes the appropriate algorithms and returns the modified image. The blocks are then recomposed to display the full modified image. In parallel image processing it often happens that the image is divided in small blocks usually of size 16 x 16 [9][13][14], this means that a medium size image of size 640 x 480 will be divided into 1,200 blocks, each one of these blocks is going to be assigned to a thread and executed on a processor. It is clear that a single multi-core CPU is not enough to process this amount of data and high performance may be reached by joining more processors, it is therefore recommended to experiment on a cluster of CPUs or other type of hardware such as GPUs. It is to be noted that when processing images in parallel the developer should also consider boundary conditions.

Boundary conditions arise when the blocks being analysed requires information from adjacent pixels which have been assigned to a different block.

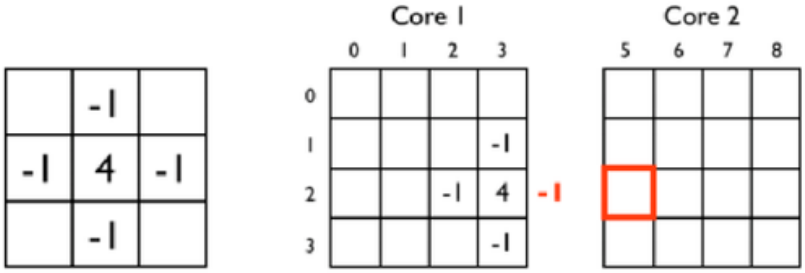


Figure 2.2: Boundary Conditions

A solution to this problem is to create the so called ghost cells to form a *'halo'* around the block and use Message Passing such as MPI to allow threads to communicate and set values of the ghost cells to the actual pixel values retrieved through thread communication. Due to the fact that the author had no experience in parallel image processing, boundary conditions have not been considered when developing concurrency in this project although the algorithms used are slightly affected from them.

2.5.2 Fingerprint Image Processing

As mentioned in the previous chapters the ultimate objective of fingerprint analysis is to detect minutiae. Minutiae, though, are often hidden or not clear enough to be detected, this depends on how the print has been acquired, on the pressure the person applied on the various parts of the fingertip and on the material used to take the print. All these factors affect the *'readibility'* of the print by adding noise or imperfections on the fingerprint image, this results in the detection of false minutiae or the non-detection of the real ones, it is therefore essential to meticulously remove such imperfections and enhance the image as much as possible. Various approaches have been developed to enhance a fingerprint image, from the NBIS software which greatest strength relies in producing quality maps and then comparing the algorithms results to such maps to obtain more reliable results [4], to Hong et al's methodology [10][11] which computes similar algorithms and attempts to reconstruct

the undefined ridges, to other techniques that analyse the image in frequency domains [15]. This project bases its implementation on Hong et al's algorithms because the NBIS software is retained to be too big to be analysed due to time constraints while analysing the image in frequency domains makes the parallelisation useless on some tasks and impossible or extremely hard to implement on others tasks.

2.5.3 Image Processing

Due to the scope of the project the author is going to concentrate only on the pre-processing stage. To effectively enhance a fingerprint image and reach the final objective of a binarized image where all ridges are well defined it is necessary to develop image processing skills. As a first approach to this practice the author focused on basic image manipulation techniques on grayscale images, grayscale images have been chosen because fingerprints are analysed using this colour scheme. A grayscale image can be studied as a function in which the resulting image *G* is defined by [9]:

$$G(i,j) = F(i,j) \tag{2.2}$$

Where *i, j* are the coordinates of the pixel values and *F* is the function applied to the image. There are three main operations at the basis of image processing:

- *Point operations* – Are defined as functions of pixel intensities. They include image enhancement operations, image scaling, image negative and control over the image brightness.
- *Arithmetic Operations* – Are operations performed on two images comparing them on a pixel basis. These kind of operations are mostly used in motion detection algorithms.
- *Geometric Operations* – Are operations used to change the image appearance. Such operations include image translation, rotation, and zoom.

Such operations constitute only the basics of image processing, in fact, they only allow to enhance the quality, morph, and compare images. The author decided to mention these operations as the algorithms analysed in the following chapters are based on these concepts.

Chapter 3: The Algorithms

This chapter is going to guide the reader through the algorithms that have been analysed throughout the project explaining their purpose and their applications.

3.1 Contrast Enhancement

The aim of the contrast enhancement is to adjust the image graylevel range so to create a sharper image for further processing. Common techniques used to achieve this objective are: Histogram Equalization, and Normalization.

3.1.1 Histogram Equalization

A useful tool to analyse an image is the image histogram [9]. This tool gives a representation of the frequency of occurrences of each colour in the image in the form of a plot or graph. Considering Figure 3.1 which is a greyscale image which grey levels vary from {0, … , K-1} and a range of intensity extending from 0 to the total number of pixels MN in the image, the correspondent image histogram results in Figure 3.2.

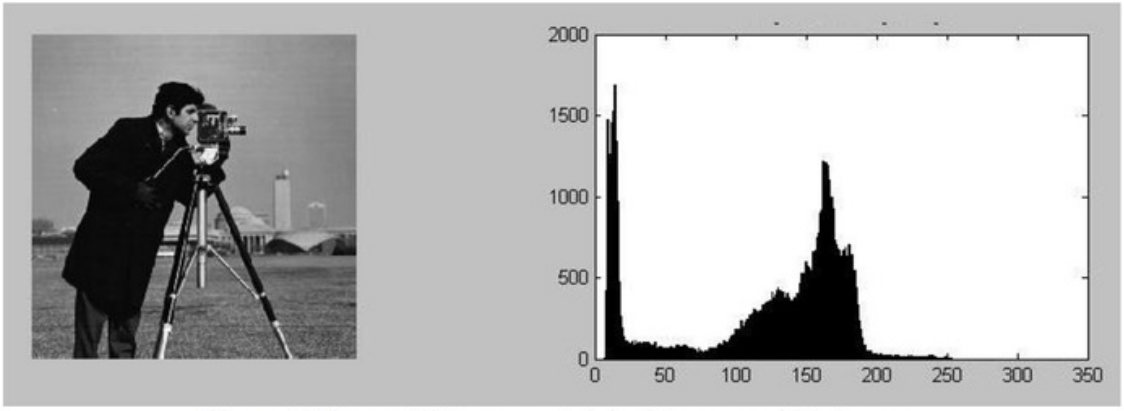


Figure 3.1 Image Histogram: Original Image and Histogram

A histogram so becomes very useful to extract information on the image e.g. threshold values for the binarization process. The histogram equalization process essentially re-distributes the pixel intensity occurrences on the histogram, this technique is also known as histogram stretch. An example of the histogram equalization process can be seen in Figure 3.2.

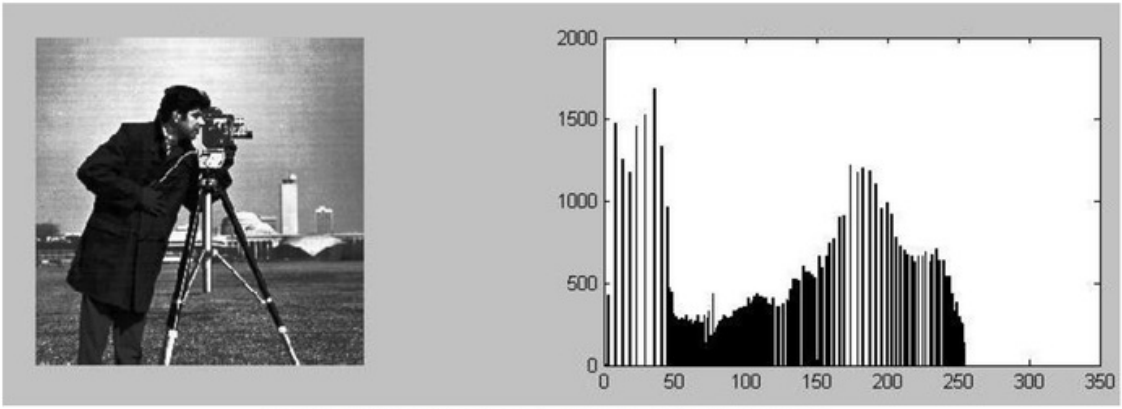


Figure 3.2: Enhanced Image and Histogram

3.1.2 Normalization

Normalisation, similarly to the histogram equalization, is a process that changes the range of pixel intensities to reach a sharpening effect without changing the ridges structure. This process is also known as contrast stretching and consists of bringing the image into a more familiar range to facilitate the processing hence the name *'normalisation'* [10]. There are various algorithms that compute this calculation, the author analysed three of them: IplImage normalisation, Wikipedia's normalisation, and a normalisation approach based on the mean and standard deviation of the image. All normalisation techniques return better results if applied on blocks rather than on the entire image, this is because the approximations calculated in from the algorithms are much more accurate if applied on local areas.

3.1.2.1 IplImage Normalization

An IplImage is a data-structure contained in the OpenCV module that represents images, the strength of this algorithm resides in IplImage's functions that perform an automatic normalization [17]. This approach uses the *'energy normalization'* formula shown in Figure 3.3 to bring the image pixel intensities to a very low range. The resulting image is expected to have a pixel intensity range from 0 to a maximum of 3.

$$N(i,j) = I(i,j) / M \tag{3.3}$$

3.1.2.2 Wikipedia's Normalization

Wikipedia's algorithm essentially reduces the entire pixel intensity range to the desired min and max values. In this particular case the author decided to normalize the image to a scale 0 to 1 where the lower values represent darker regions. Due to the resource quoted in Wikipedia's article [16] the formula is retained to be reliable. The author of the article suggests to apply the following formula:

$$I_N = (I - \text{Min})\frac{\text{newMax} - \text{newMin}}{\text{Max} - \text{Min}} + \text{newMin} \tag{3.4}$$

3.1.2.3 Mean and Standard Deviation Normalization

This algorithm was suggested in various fingerprint enhancement algorithms [10][18][19][20][21], it consists of reducing the pixel intensity range by calculating the mean and variance of the image and then applying a thresholding process using the mean as a threshold. This algorithm performs an image segmentation by separating the foreground (values below the threshold) from the background (values above the threshold) and then computes different formulas so to reduce the values of the foreground and increase the values of the background, this results in a severe contrast enhancement and a reduction in

gray level variations. The resulting image is expected to have a larger range when compared to the algorithms mentioned above, but its contrast is going to be highly enhanced. Below is the formula used in this algorithm.

$$N(i,j) = \begin{cases} M_0 + \sqrt{\frac{V_0(I(i,j)-M)^2}{V}} & \text{if } I(i,j) > M, \\ M_0 - \sqrt{\frac{V_0(I(i,j)-M)^2}{V}} & \text{otherwise,} \end{cases}$$

(3.5)

3.2 Spatial Filters

Spatial filters are based on the concept of convolving kernels with the image so to calculate new pixel values based on the surrounding pixels.

3.2.1 Gaussian Smoothing Filter

A gaussian smoothing filter is a two-dimensional convolution filter used to remove noise from an image (blurring). The filter works by convolving an image using a kernel created using a Gaussian function which looks as in Figure 3.6 [22]. For each pixel in the image the filter is going to compute a new value based on the surrounding pixel intensities.

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

(3.6)

Where G is the image, x are the coordinates, and σ is the sigma.

3.2.2 Sobel Filter

The Sobel filter is also a two-dimensional convolution filter based on Sobel kernels that look as in Figure 3.7 [23]. This filter is widely used in edge detection algorithms because it computes the derivatives along the x and y directions of the image which can be later used to calculate the gradient intensities at each pixel. This filter is known to generate noise, therefore it is recommended to blur the image using a gaussian before the execution of this operation and after on the derivative matrixes.

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

Gx

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

Gy

(3.7)

3.3 Edge Detection

When analysing a fingerprint it is essential to correctly identify the ridges and separate them from the background. Ridges correspond to the darker segments in the image and are characterized by strong variations in the pixels brightness hence they can be analysed as edges. The aim of edge detection algorithms is to identify discontinuities in image brightness and return a set of connected curves that represent the boundaries of the objects in the image, in our case the ridges. There are two main approaches to edge detection algorithms: search-based approaches and zero crossing. The search-based approach will be discussed in the following paragraphs followed by the Canny edge detection algorithm. These algorithms are known to generate noise which is dealt with a gaussian low-pass filter, such filter is a pre-requisite and is described ***Chapter 3.2.1***.

3.3.1 Sobel Edge Detection Algorithm

The Sobel edge detection algorithm is a search-based approach which computes the first derivative on the x and y directions using a Sobel filter [23], then it calculates the gradient intensities at each pixel using the formula in Figure 3.8. The gradient intensities represent the changes in brightness where the highest values are going to be found on the edges of the ridges, therefore applying a thresholding process will return a binary image which enhances the edges.

$$G(i,j) = \sqrt{(G_x)^2 + (G_y)^2}$$

(3.8)

3.3.2 Canny's Edge Detection Algorithm

Canny's edge detection algorithm returns the most accurate results. It's strength resides in suppressing the less-strong edges and applying a hysteresis thresholding (two thresholding values) resulting in a binary image which edges are not as thick as the ones detected using the Sobel edge detection algorithm. Canny's algorithm can be divided in four steps [24]:

1. Apply a gaussian filter to smooth the image
2. Apply Sobel's filter to find the first derivatives along the x and y axis
3. Perform a non maximal suppression
4. Finally binarize the image using hysteresis thresholding

The last two steps are what differs Canny's approach from Sobel's. The non -maximal suppression identifies the local maximums by first approximating the detected angles to: 0°, 45°, 90°,135° then, considering the angle direction, it will compare the neighbouring gradient intensities and set the pixel intensities that are not maximums to zero. The final binarization process uses two threshold values to first identify strong edges and then identify weaker edges connected to them.

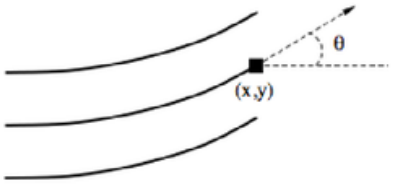
Edge detection algorithms appeared to be essential when approaching fingerprint analysis but their implementations led the author to consider another approach. The reason is that detecting the edges of a ridge results in displaying the ridges contours which can be mistaken for other ridges. On the other hand the algorithm chosen for the fingerprint enhancement uses an orientation field estimation which computes the gradients like in the Sobel edge detection algorithm.

3.4 Fingerprint Enhancement

After developing skills in image analysis the author moved on to more complex algorithms and approached the full fingerprint enhancement system. A commonly used fingerprint enhancement approach [10][18][19][20][21]uses a Gabor filtering technique to enhance the image and then a thinning algorithm to remove the imperfections along the ridges. The first step of this algorithm consists in normalizing the original image with the mean and standard deviation approach described in ***Chapter 3.1.2.3*** where the author decided to differ from Hong et al's approach by implementing the normalization in parallel, then build an orientation image and a frequency image that will be used to construct the Gabor filter.

3.4.1 Building the Orientation Map

The orientation map is a matrix of the same dimensions of the original image, its values represent the local orientations of the pixels at each block see Figure 3.9. This step is fundamental as the Gabor filter relies on an accurate local orientation estimation to perform correctly.



(3.9)

The algorithm operates on blocks of size 16×16, it first applies a Gaussian filter to smooth the sub-image and then computes the first derivatives along each axis G_x *and* G_y using a Sobel filter. The Sobel filter is known to generate noise therefore another Gaussian filter is applied along each derivative before calculating the gradient intensity G and the orientation angle θ given by the formula in Figure 3.10.

$$G(i,j) = \sqrt{(G_x)^2 + (G_y)^2}$$

$$\theta(i,j) = \frac{1}{2} tan^{-1} \frac{G_y(i,j)}{G_x(i,j)}$$

(3.10)

It is then necessary to calculate the average orientation angle of the block to obtain a more precise result. This is done by calculating the least square estimate of the angles, the formula is expressed in Figure 3.11.

$$G(i,j)^2 * (\cos^2\theta(i,j) - \sin^2\theta(i,j)) = G_x(i,j)^2 - G_y(i,j)^2$$

$$G(i,j)^2 * (2 \sin\theta(i,j) \cos\theta(i,j)) = 2G_x G_y$$

(3.11)

These operations return the variance and covariance of the angles in the blocks of 16×16 and are used to calculate the orientation field components in the *x* and *y* directions using the formula shown in Figure 3.12.

$$\theta x(i,j) = \frac{Gx(i,j)^2 - Gy(i,j)^2}{\sqrt{((2Gx\,Gy)^2 + (Gx(i,j)^2 - Gy(i,j)^2)^2)}}$$

$$\theta y(i,j) = \frac{2GxGy^2}{\sqrt{((2Gx\,Gy)^2 + (Gx(i,j)^2 - Gy(i,j)^2)^2)}}$$

(3.12)

Once again it is necessary to remove the noise generated by the calculations with a gaussian filter. The final orientation field is given from the formula in Figure 3.13

$$\theta(i,j) = \frac{\pi + \arctan\left(\frac{\theta y(i,j)}{\theta x(i,j)}\right)}{2}$$

(3.13)

3.4.2 Local Ridge Frequency Estimation

The ridge frequency estimation is the second component of the Gabor filter, it represents the frequency of the ridges occurrences in a given block of size *w x w* typically *32 x 32*. Hong et al's method calculates the frequency by creating an almost sinusoidal signal which extends on an axis orthogonal to the local orientation field. The signal's values are going to be the projections of the gray level variations within the block so that the minimums of the signal are determined by the ridges while the maximums from the valleys. The local ridge frequency *F* can now be computed with the formula shown in Figure 3.14 where *S(i,j)* is the wavelength, in pixels, computed by averaging the space between each projected minimum.

$$F(i,j) = \frac{1}{S(i,j)}. \tag{3.14}$$

It is to be noted that this calculation might prove inaccurate in case of poor quality regions or in the presence of minutiae. In these cases the frequency should be calculated from neighbouring blocks.

3.4.3 Gabor Filter

A Gabor filter is a spatial convolution filter which objective is to enhance the ridges while preserving their structure, it can be defined as a Gaussian kernel modulated by a sinusoidal wave along a given direction. Due to its orientation and frequency components this filter is able to maximize its effect along the ridges removing excessive noise. An even symmetric Gabor filter so assumes the following form:

$$G(x,y;\theta,f) = \exp\left\{-\frac{1}{2}\left[\frac{x_\theta^2}{\sigma_x^2} + \frac{y_\theta^2}{\sigma_y^2}\right]\right\}\cos(2\pi f x_\theta)$$

$$x_\theta = x\cos\theta + y\sin\theta.$$
$$y_\theta = -x\sin\theta + y\cos\theta.$$

(3.15)

Where θ is the orientation of the Gabor filter obtained from the orientation map, f is the local frequency estimation obtained through the frequency map, *x* and *y* are the coordinates of the pixels used to point also to the values on the orientation map, and σ_x and σ_y denote the sigma values of the filter.

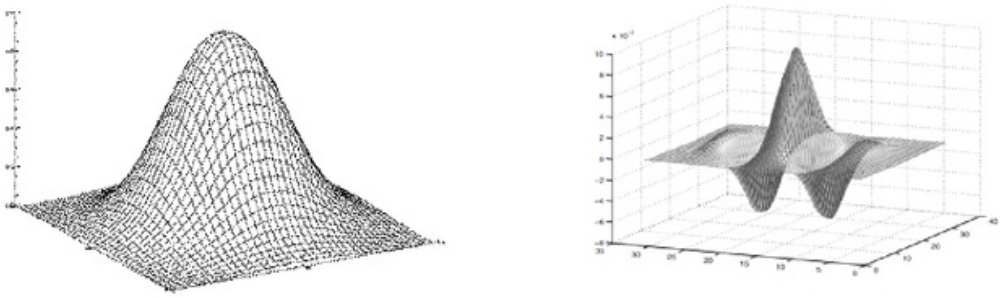


Figure 3.16: Comparison between Gauss Filter (left) and Gabor's Filter (right)

It is to be noted that Hong et al's [11] approach uses fixed values for the σ_x and σ_y while Raymond Thai [10] suggests a more flexible approach by calculating new values based on the frequency map. His assumption consists in describing σ_x and σ_y as the standard deviations along the *x* and *y* axis and therefore he deducts that these values can be calculated using the following formulas:

$$\sigma_x = k_x F(i,j)$$
$$\sigma_y = k_y F(i,j) \qquad \textbf{(3.17)}$$

Where k_x and k_y are constants and *F(i,j)* denotes the frequency. This approach finds its strength in providing adaptive sigma values hence tuning Gabor's filter to return more accurate results.

The resulting enhanced pixel *E(i,j)* finally takes the form of:

$$E(i,j) = \sum_{u=-\frac{w_x}{2}}^{\frac{w_x}{2}} \sum_{v=-\frac{w_y}{2}}^{\frac{w_y}{2}} G(u,v;O(i,j),F(i,j))N(i-u,j-v) \qquad \textbf{(3.18)}$$

Where *G* defines the Gabor filter constructed using the pixel's orientation *O(i,j)*, the pixel's frequency value *F(i,j)* convolved on a kernel of size w_x and w_y, applied on the normalised image *N*.

3.4.4 Binarization and Thinning

The final steps in the Fingerprint Enhancement System are the binarization and the thinning operations [18]. The binarization is going to use a threshold value to better define the image foreground from the background, the values that fall below the threshold are going to be set to 0 resembling darker regions hence the ridges while the values that are greater than the given threshold will be assigned 1 to resemble brighter regions hence the background. A final thinning algorithm should be applied to the binarized image, the algorithm should erode the ridges in order to deliver a skeletonized version of the fingerprint ridges. This final step is necessary as some minutiae extraction algorithms use combinations of pixels to determine the type of minutiae being extracted.

Chapter 4: Requirements

Requirement gathering is a crucial phase when developing a software solution. Requirements help the analyst understand how the stakeholders expect the system to work. Based on the analysts deductions it becomes easier to structure the system, fix the goals and foresee eventual problems.

Due to the fact that the author had no experience in image processing and parallel computing and he had to test various algorithms it was decided to implement the following systems before attempting to develop a full fingerprint enhancement system, in this case the developer has become a user and the main stakeholder of his own systems:

- Normalization Comparison System
- Sobel Edge Detection System

and finally the Fingerprint Enhancement System.

This chapter is going to guide the reader through the requirements elicitation techniques used throughout the project and the actual requirements of each system that was developed.

4.1 Elicitation of Requirements

- *Document Analysis:* Document Analysis consists in gathering information from existing sources and understanding how similar systems have been developed by experts. This particular technique is very useful when the project involves evolving an existing product because existing documentations are a valuable resource. The NIST Biometric Image Software documentation introduced the author to the enhancement algorithms, these were later researched and studied more in depth analysing the NBIS source code and other similar software products.

- *Testing Existing Versions:* This technique is self-explanatory, it involves using the previous versions to understand the systems strengths and weaknesses. The testing usually starts by simply using the existing systems as a normal user, this will help

gather information on the user experience. It will then follow black-box testing to understand the system's capabilities.

- *Interviewing:* Interviewing experts is a valuable requirement elicitation technique. Learning from someone else's experience allows to foresee problems and better understand the system's requirements. Throughout the project interviewing experts always lead to a deeper understanding of the algorithms and how these could be improved in performance and accuracy.
- *Prototyping:* This technique involves producing prototypes of the system or parts of the system. By prototyping the author was able to gain a deeper knowledge of how the algorithms were performing their tasks.
- *Use-Cases:* Modelling the system using use cases helped to better define the use of the system.

4.2 Elicited Requirements Overview

Scientists, forensics inspectors, or any other user of image processing software are generally interested in two main aspects of the software being used: performance and accuracy. Users of these systems are generally not interested in user interfaces because their input to the system is going to be minimum, they expect to input an image and get back results in the least time possible. In the case of fingerprint analysis the users final objective is to retrieve a set of minutiae that he will later check against a database. The user in this case, is not interested in knowing what the system does nor why it does it, he does not know the parameters being used by the algorithms and often is not even aware of what the algorithms do. The user expects the system to be fully automated and optimized for the image he has to process, he does not want to provide any other input except from the original image.

On the other hand, a developer at the first arms with image processing algorithms becomes the user of his own prototypes. His objective is to experiment the various algorithms he took in consideration for the final deliverable retrieving information on which parameters are most suited for the tasks he has to accomplish and which algorithms perform better, therefore the need to create more systems which requirements include setting different parameters to experiment with. These systems are bound to be either discarded or integrated in the final solution.

4.3 Normalization System Requirements

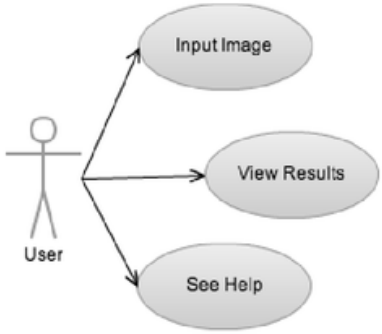


Figure 4.1: Normalization System Use Case Diagram

The aim of this system is to generate an output from which the user is able to compare the results of three different normalization algorithms. The objective here is to decide which algorithm to use in the final image enhancement system by comparing their results.

4.3.1 Functional Requirements

FR1. The system shall output the result of Wikipedia's normalization algorithm applied on the entire image.

FR2. The system shall output the result of the mean and standard deviation normalization on the full image.

FR3. The system shall output the result of Python's IplImage normalization on the full image.

FR4. The system shall output the result of a parallel Wikipedia's algorithm implementation on image blocks of size 16×16.

FR5. The system shall output the result of the mean and standard deviation normalization implemented in parallel on blocks of size 16×16.

FR6. The system shall output the result of the IplImage algorithm executed in parallel on blocks of size 16×16.

FR7. The system shall process 640X480 images.

4.3.2 Non Functional Requirements

NFR1. The system shall not have any user interface.

NFR2. The system shall ask for user input.

NFR3. The system shall check for a valid input.

NFR4. The algorithms shall use point-operations.

NFR5. The system shall process only 640×480 size images.

NFR6. The system shall process only .png files

NFR7. The system shall be limited to displaying the results without saving them.

NFR8. The system's source code shall be written in Python 2.7

NFR9. The system shall use the following modules: Numpy, OpenCV, PIL.

NFR10. All the output images shall be the same size of the original image.

NFR11. The system shall use Wikipedia's normalization algorithm.

NFR12. The system shall use mean and standard deviation algorithm.

NFR13. The system shall use openCV's IplImage normalization.

NFR14. The system shall execute all the previously mentioned algorithms on a local basis in parallel.

NFR15. The system shall use threads.

NFR16. The system shall use a queue from which the threads will retrieve the data.

NFR17. The system shall decompose the original image, process it, and re-compose the processed image.

NFR18. The system shall use a dictionary data-structure to control the thread's output.

NFR19. The system shall provide a *'help'* page for the user.

4.4 Sobel Edge Detection Requirements

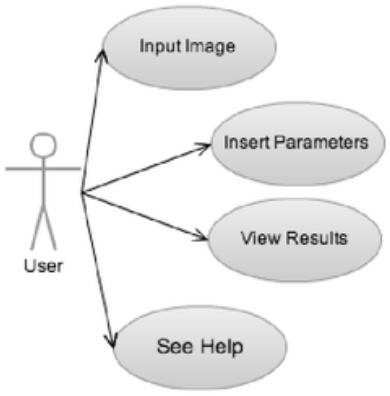


Figure 4.2: Sobel Edge Detection System Use Case Diagram

This system is developed to analyse Sobel's Edge Detection algorithm. It's purpose is to be able to compare the results obtained throughout Sobel's algorithm with the original image and experiment the effect of different values parameters on the image.

4.4.1 Functional Requirements

FR1. The system shall output the original image.

FR2. The system shall output a blurred image.

FR3. The system shall output an image which represents the first derivative on the x-axis.

FR4. The system shall output an image which represents the first derivative on the y-axis.

FR5. The system shall output a binarized image where the edges of the ridges are well defined.

4.4.2 Non Functional Requirements

NFR1. The system shall not have any user interface.

NFR2. The system shall ask for user input.

NFR3. The system shall check for a valid input.

NFR4. The algorithms shall use point-operations.

NFR5. The system shall process only 640×480 size images.

NFR6. The system shall process only .png files

NFR7. The system shall be limited to displaying the results without saving them.

NFR8. The system's source code shall be written in Python 2.7

NFR9. The system shall use the following modules: Numpy, OpenCV, PIL.

NFR10. The parameters for each operation shall be specified by the user.

NFR11. All output images shall be the same size of the original image.

NFR12. The system shall apply a Gaussian blurring filter of size and sigma specified by the user.

NFR13. The system shall apply a spatial Sobel filter to calculate the gradients in both the x and y directions.

NFR14. The system shall apply Sobel's edge detection algorithm.

NFR15. The threshold for binarization shall be specified by the user.

NFR16. The sigma for the Gaussian filter shall be specified from the user.

NFR17. The system shall provide a *'help'* page for the user.

4.5 Fingerprint Enhancement System Requirements

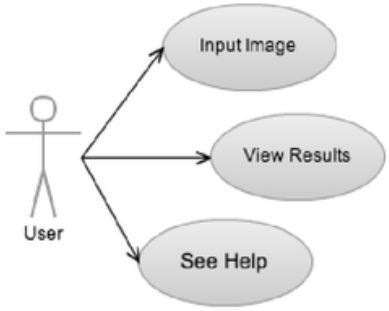


Figure 4.3: Fingerprint Enhancement System Use Case Diagram

The fingerprint enhancement system is developed as an attempt to produce a complete fingerprint enhancement system.

4.5.1 Functional Requirements

FR1. The system shall output a binarized image where all edges are well defined.

FR2. The system shall output an orientation image where lines represent local directions.

FR3. The system shall output a ridge frequency image.

FR4. The system shall output a thinned binary image.

FR5. The system shall output a normalized image on which orientation lines are drawn.

4.5.2 Non Functional Requirements

NFR1. The system shall not have any user interface.

NFR2. The system shall ask for user input.

NFR3. The system shall check for a valid input.

NFR4. The algorithms shall use point-operations.

NFR5. The system shall process only 480×640 size images.

NFR6. The system shall process only .png files

NFR7. The system shall be limited to displaying the results without saving them.

NFR8. The system's source code shall be written in Python 2.7

NFR9. The system shall use the following modules: Numpy, OpenCV, PIL, SciPy.

NFR10. The system shall apply a Normalization algorithm.

NFR11. The system shall apply a Gaussian blurring filter.

NFR12. The system shall apply a Sobel filter to calculate the gradient derivatives on the x and y axis.

NFR13. The system shall produce an orientation map.

NFR14. The system shall produce a ridge frequency map.

NFR15. The system shall use a spatial Gabor filter to enhance the ridges.

NFR16. The system shall use a thinning algorithm to refine the ridges.

NFR17. The system shall use threads.

NFR18. The system shall use a queue to control the input data of the threads.

NFR19. The system shall decompose and re-construct the image.

NFR20. The system shall provide a *'help'* page for the user.

Chapter 5: Design

Due to the iterative nature of the software development lifecycle the design of the systems kept changing throughout the development. This chapter is going to illustrate the main design choices of each developed system and the design of how the parallelisation of tasks occurred.

5.1 UML as modelling language

UML is a standard modelling language established in the early 90's. It is implemented with the aim of producing and visualizing a system's architectural blueprints such as its actors, activities, components and programming language statements. UML proves to be very efficient when analysing a system which architecture has been represented using this technique because it allows the analyst to easily detect the components and understand how they have been constructed and how they interact with the other components. Although UML is used as a modelling standard it does present some drawbacks, since the code is represented in diagrams elements of the code loose much information and therefore the represented code is sometimes inaccurate. Furthermore developers are sometimes obliged to write more lines of code because they have to follow the design.
Despite its flaws UML still stands as the most used modelling language and is best practice to represent the system being developed using this standard.

5.2 Sobel Edge Detection Architecture

Sobel's algorithm uses two convolution filters to smooth the image and determine the gradients at each pixel. The design choice for this system is to isolate the filters so to be able to easily test them and eventually re-use them in other implementations.

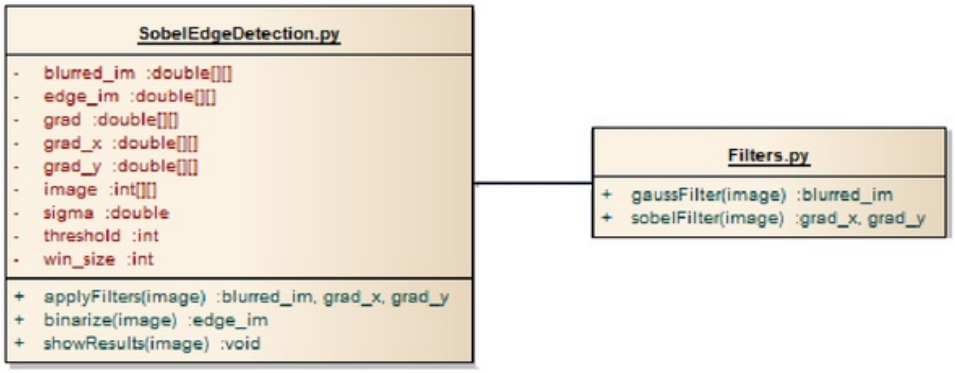


Figure 5.1: Sobel Edge Detection System Class Diagram

5.3 Parallel Image Processing Architecture

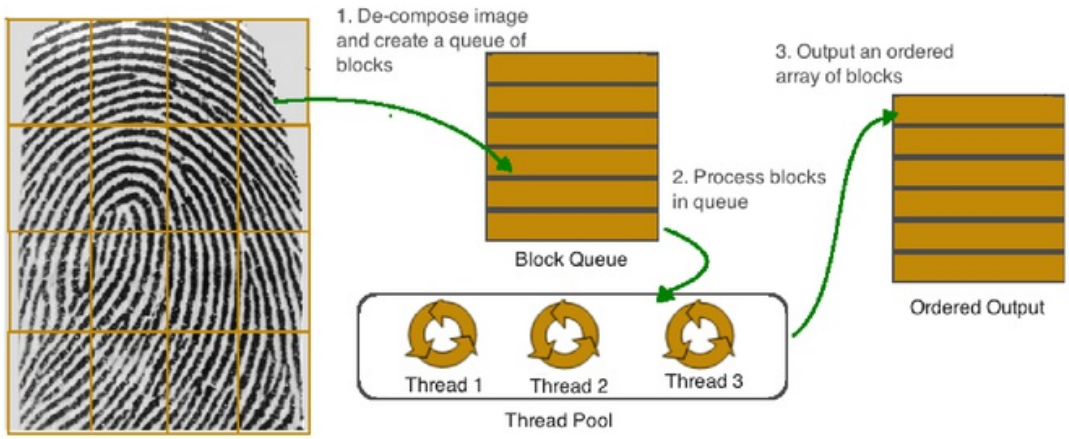


Figure 5.2: Parallelization Architecture

One approach to analyse an image in parallel is to de-compose the image in blocks. Each block is then assigned to a thread which will process it using the desired algorithms and return the processed result. Once the threads have processed all the blocks and the results organised in an ordered array, then the image can be re-composed to form the final processed image. Although this approach does not require thread synchronization or mutual exclusion of the resource there where some problems that had to be dealt with:

Problems	Solutions
Threads had to process each block only once.	Create a queue of blocks.
No matter how many threads are created, when a thread is done processing the given block it should load other blocks recursively.	Create a queue of blocks and a pool of threads.
Control the threads output so to construct an array of processed blocks in which each block is in the correct position.	Use a dictionary data-structure (also known as associative arrays) and load the entire data-structure in the queue.

Table 5.1: Parallelization Problems and Solutions

The proposed approach uses a thread controller to de-compose the image, create a pool of threads, start the threads and re-compose the image while using the actual threads to execute the operations on the blocks. The class diagram for such design is as follows:

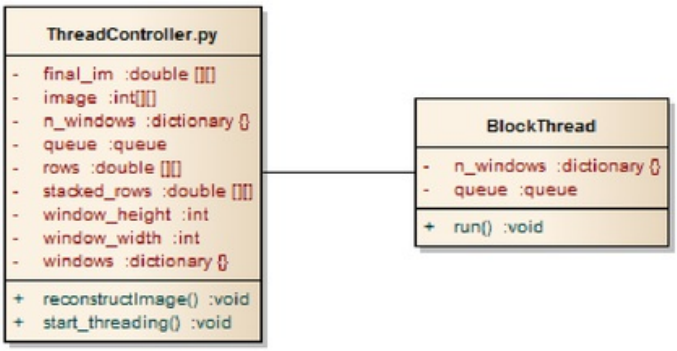


Figure 5.4: Parallelization Class Diagram

The image reconstruction occurs by first creating the rows by stacking horizontally the ordered blocks. Then, the rows are stacked vertically to form the final image.
As in all design decisions there are strengths and weaknesses: the great strength of this design is that the number of threads can easily be changed to suit the number of processors and that thanks to the use of a queue and the dictionary data-structure there is no need to

synchronize the threads to be able to control the input and output. As a weakness it must be noted that this design does not support boundary conditions and therefore it is not suited for certain algorithms.

5.4 Normalization System Architecture

As mentioned in the previous chapters the normalization system was built to test three normalization algorithms. It was decided to create a system that would apply the algorithms first on the entire image and then on a local basis using a parallel approach so to compare the results.

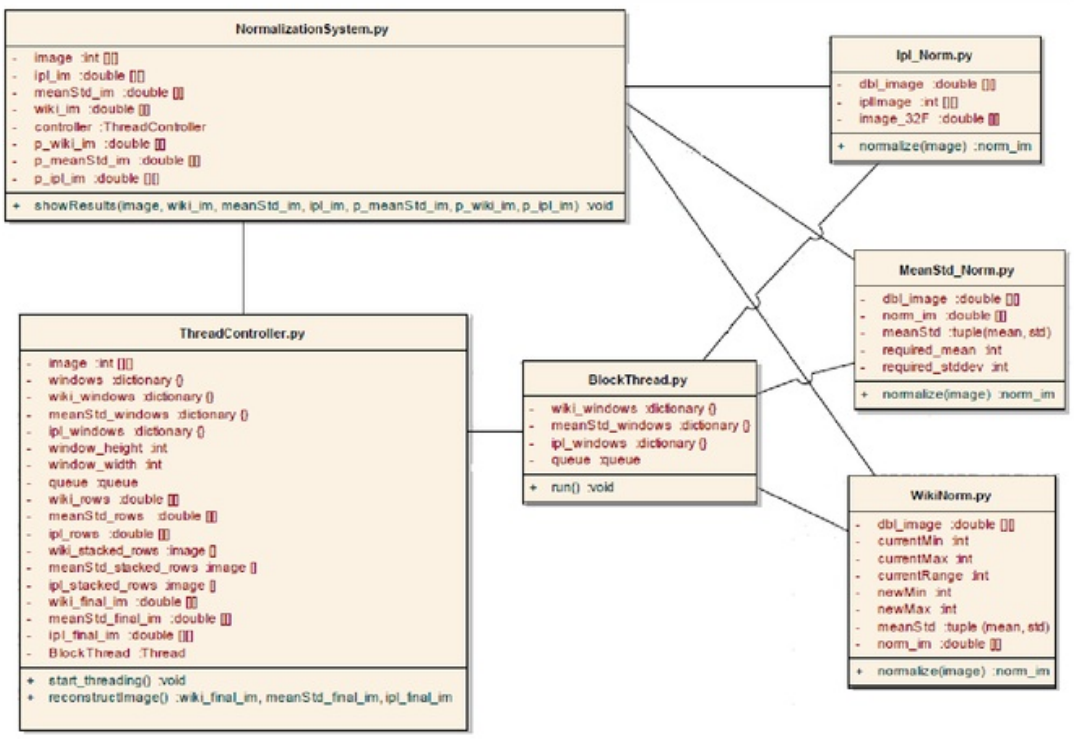


Figure 5.4: Fingerprint Enhancement Class Diagram

The design decisions for this system where based on the following considerations:

- *Re-usability* – The system must re-use the parallel code used in other applications. The algorithms must be designed to allow an easy integration in other systems.
- *Maintanability* – The aim of this system is to compare the results of the algorithms hence the need to isolate each algorithm in order to be able to modify them without compromising the system.

The normalisation system initially computes the algorithms on the entire image by initiating the algorithms components and passing the original image as an argument to their functions saving the results in the wiki_im, meanStd_im and ipl_im. It then applies the same algorithms on blocks of size 16×16 making use of a slightly modified version of the parallel system explained in the previous paragraph. This time, the ThreadController will produce three images each one representing the original image after the respective normalization algorithms have been executed saving their output to p_wiki_im, p_meanStd_im, p_ipl_im. The normalised images are returned to the main NormalisationSystem class that will display them together with their respective min and max values.

The strength of this design relies in using of the previously written parallelisation code and the re-usability of the algorithms components throughout the software execution.

5.5 Fingerprint Enhancement Architecture

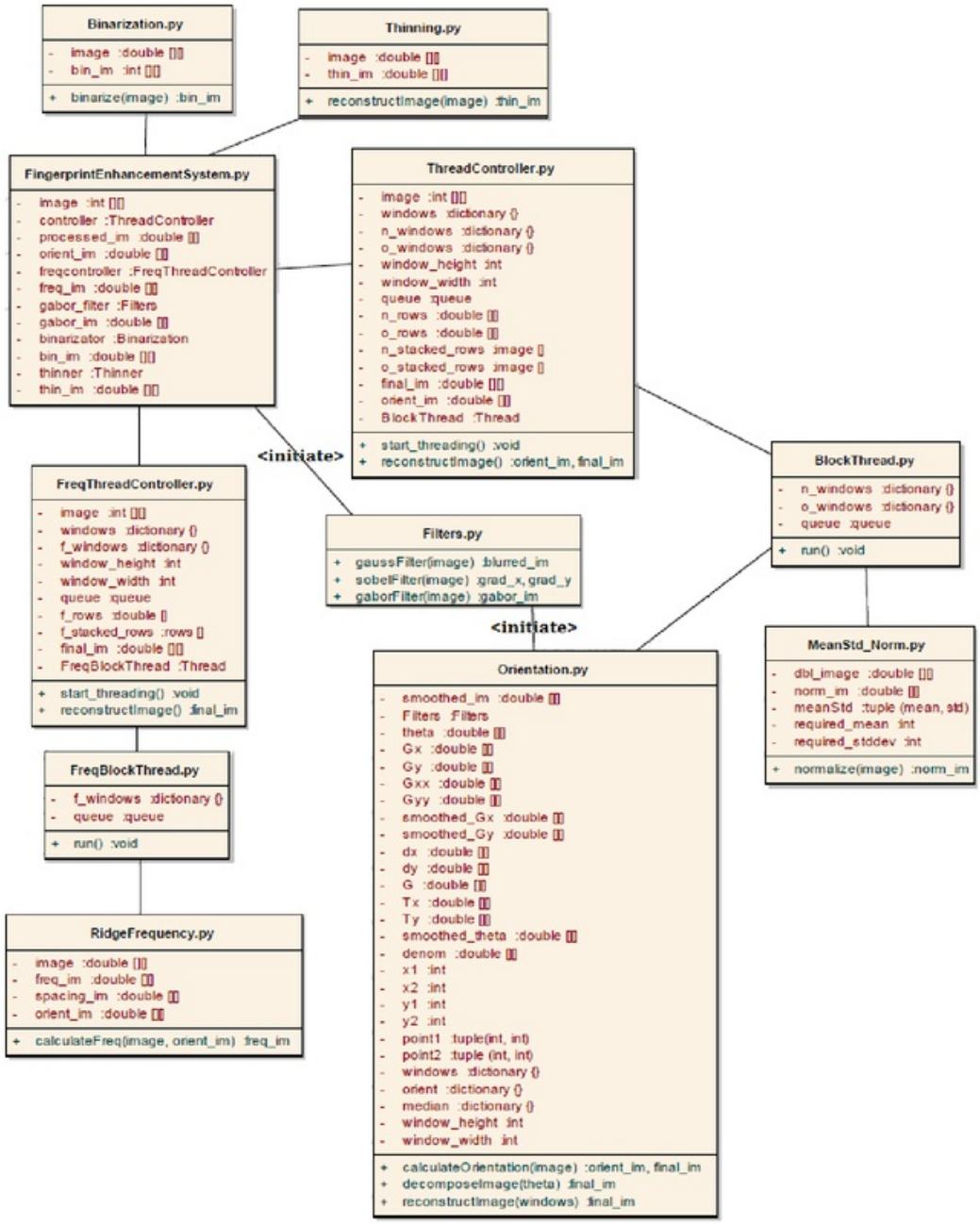


Figure 5.5: Fingerprint Enhancement System Class Diagram

The Fingerprint Enhancement System makes use of eleven components:

1. *FingerprintEnhancementSystem.py* – is the main executable.
2. *ThreadController.py* – which creates the thread-pool, divides the image blocks and then re-composes the image returning an orientation map as well as the processed image.
3. *BlockThread.py* – which schedules the normalization and the orientation estimation.
4. *MeanStd_Norm.py* – which is a re-used component from the normalization system.
5. *Orientation.py* – which is the component that calculates the estimated orientation using the the Gauss and Sobel filters from the Filters component orientation and draws lines to represent it.
6. *FreqThreadController.py* – operates the same way as the ThreadController but divides the image in blocks of size 32X32
7. *FreqBlockThread* – initiates the frequency estimation algorithm component and passes the given block as an argument.
8. *Filters* – this class contains all the filters needed from the other components: Sobel Filter, Gauss Filter, Gabor Filter.
9. *Binarization.py* – this class will take as an input the enhanced image and return a binary version of it.
10. *Thinning.py* – Is the last component to be initiated and returns a skeletonised version of the binary image.

An interesting design choice was the construction of the thread controllers: both the TreadController.py and the FreqThreadController.py use the same concept explained in ***Chapter 5.4.*** The need to create two controllers was given by the fact that the ridge frequency estimation is to be computed on blocks of size 32X32 while the orientation estimation requires blocks of size 16X16, the author could have designed the system in such a way to have one controller for both algorithms but he preferred this choice of design as he retains it to be logically easier to understand and even if this approach results in more lines

of codes it also facilitates maintainability and re-usability. In fact the strength of this architecture resides in the fact that the components are isolated and can easily be modified.

Chapter 6: Implementation

This chapter is going to illustrate the implementation of the systems describing why some decisions have been taken over others. It includes discussions and comparisons of the algorithms being used, benchmarks, and the results of such systems.

6.1 Implementation Environment

Although the delivered systems where developed and implemented using Python 2.7 programming language on PyDev which is an IDE for Eclipse, some results where analysed and compared to pre-written functions using MATLAB R2012a. This is because Matlab provides a view of the data being manipulated hence making it easier to analyse. The Operating System used throughout the implementation and testing was a Mac OSX Version 10.8.3 with a processor Intel Core i7 2.9GHz and 8Gb of RAM. The experiments were conducted on fingerprint images from the 2002 Fingerprint Verification Competition (FVC2002) database [27], these images have been converted from Tagged Image File Format (Tiff) to Portable Network Graphics (PNG) format for ease of testing.

6.2 Threading

Threading was used throughout the project when developing the normalization system and the fingerprint enhancement system. Both applications make use of a ThreadController to divide the image in blocks of size 16X16, process the blocks using the BlockThread, and then reconstruct the full processed image. Both systems use a queue of blocks to control the thread input and dictionary data-structures to control the output, this approach does not require thread synchronization because the threads do not change the state of the data being invoked and also because, thanks to the queuing system, the object being processed is only used once within the current thread. The code in Figure 6.1 illustrates how the image is divided in blocks of size 16X16, organized in a queue, and how the thread-pool initiates the threads [25]. Line 39 uses an interesting Numpy feature, the so called "*slicing*" allows to

extract a sub-image by specifying the coordinates, the 'count' variable is used to specify the index in the 'windows' dictionary saving the sub-image to a specified destination. The thread pool is created in line 47, it spawns 4 threads initialising them with the queue and the output data-structure. It is to be noted that the queue is a thread-safe data-structure hence it automatically controls the input.

```
23    def start_threading(self):
24        '''Segments the image in windows of size 16x16 then sends
25            each windows to a thread which processes the window
26        '''
27
28        # initialize queue
29        queue = Queue.Queue()
30
31        currentx = 0
32        currenty = 0
33
34        # create windows
35        count = 0
36        for y in range (0,self.image.shape[0]/self.window_height):
37            currentx = 0
38            for x in range(0,self.image.shape[1]/self.window_width):
39                self.windows[count]=self.image[currenty:currenty+self.window_height,currentx:currentx+self.window_width]
40                currentx += self.window_width
41                count += 1
42
43            currenty += self.window_height
44
45        # spawn pool of threads passing the queue object, this allows for threads to
46        # recurrently analyze queue objects.
47        for i in range(4): # change range to change number of threads
48            t = BlockThread.BlockThread(queue,self.n_windows)
49            t.setDaemon(True)
50            t.start()
51
52        # insert windows in the queue
53        new_windows = self.windows.items()
54        for window in new_windows:
55            queue.put(window)
56
57
58        queue.join()
```

Figure 6.1: Code Snippet ThreadController.py (Fingerprint Enhancement System)

Depending on the system being used the BlockThread class invokes the algorithms to be computed, in the fingerprint enhancement system the thread invokes the normalization and the local ridge orientation algorithm and executes them on the sub-image saving their results in a single dictionary *'n_windows'* the advantage of using this data-structure is that it allows to save the result in a specific position no matter which thread finishes first. The code illustrated in Figure 6.2 shows how the data-structure is used: 'win' is the object from the

queue, it is an associative array where *'win[0]'* is the index and *'win[1]'* is the actual sub-image; hence the code reads *"normalize win[1] and save the result to n_windows at position win[0] then calculate the orientation".*

```
class BlockThread(threading.Thread):


    def __init__(self, queue, n_windows):
        threading.Thread.__init__(self)
        self.queue = queue
        self.n_windows = n_windows

    def run(self):
        while True:

            print threading.currentThread().getName()

            # grabs window from queue
            # Note: win is a tuple [key, value], using dictionary data-structure
            win = self.queue.get()

            # Normalization
            self.n_windows[win[0]] = WikiNorm.normalise(win[1])

            # Orientation
            self.n_windows[win[0]] = Orientation.calculateOrientation(self.n_windows[win[0]])
            # signals queue job is done
            self.queue.task_done()
```

Figure 6.2: Code Snippet of BlockThread.py (Fingerprint Enhancement System)

The last function of the ThreadController is to re-construct the images, this occurs by first creating the rows by stacking each sub-image horizontally, when the process hits the last sub-image of the row it appends the row to an array of rows. Finally the process stacks the rows vertically to form the final image.

6.3 Normalization System

The normalization system was built to compare three different algorithms. These were first computed on the entire image, and then in parallel.

6.3.1 Wikipedia's Normalization Algorithm

Figure 6.3: Wikipedia's Algorithm Normalization Results

Image	Min	Max
Original Image	0	255
Wikipedia's Normalization	0	1
Parallel Wikipedia's Normalizatio	0	1

Table 6.1: Wikipedia's Normalization Algorithm Results

As shown in Figure 6.1 applying Wikipedia's normalization algorithm on the entire image slightly enhances the contrast while reducing the range from 0-256 to 0-1. Computing this algorithm on blocks of size 16X16 clearly enhances the ridges more than the normal approach, this is due to the fact the normalization algorithm computes the values on a local basis and so calculates more accurate results.

```
16  def normalise(image):
17
18      dbl_image = image.astype(float)
19
20      # computing normalization
21      currentMin = np.min(dbl_image)
22      currentMax = np.max(dbl_image)
23      currentRange = currentMax - currentMin
24      newMin = 0
25      newMax = 1
26      newRange = newMax - newMin
27
28      # calculating mean and standard deviation
29      meanStd = cv2.meanStdDev(dbl_image)
30
31      norm_im = dbl_image
32      # regions with a low standard deviation are assumed to NOT be regions of interest and
33      # have values close to currentMax therefore their value is set to the brightest possible -> 1
34      if meanStd[1]>20:
35
36          norm_im = (dbl_image - currentMin)*(newRange / currentRange)
37
38      elif meanStd[1]<=20:
39
40          norm_im = norm_im / currentMax
41
42      return norm_im
```

Figure 6.4: Code Snippet of WikiNorm.py (Normalization System)

Before Wikipedia's formula is applied the image is converted from *'uint8'* to *'float'* in line 18, this is because the new values are going to be decimal values. Once calculated the mean and standard deviation in the tuple meanStd the algorithm performs a segmentation: since the ridges are characterized by high standard deviation values, this property is exploited to isolate the background (who's standard deviation values are assumed to be very low) using a threshold. The background regions are assigned a bright colour close to 1. The segmentation only makes sense on a parallel approach as the standard deviation calculated on the full image would return the full range of the pixels hence the *'elif'* statement in line 38 will never be reached.

6.3.2 Mean and Standard Deviation Normalization

Figure 6.5: Mean and Standard Deviation Normalization Algorithm Results

Image	Min	Max
Original Image	0	255
MeanStd Normalization	-36.51535	2.2685941
Parallel MeanStd Normalization	-36.33924	17.697812

Table 6.2: Mean and Standard Deviation Normalization Algorithm Results

Figure 6.5 illustrates the results of this normalization approach on the full image, the ridges result greatly enhanced to the point where they appear to lose their structure. This is due to the fact that the algorithm applies a threshold approach based on the Mean of the pixels intensities greatly enhancing the values that fall below that threshold, some of these represent the noise in between the ridges which is enhanced as well resulting in damaging the ridges structure. This effect is greatly reduced when applying the algorithm in parallel on blocks of size 16X16. The ridges result more separated from the background noise but appear to be broken in some points, also in this case the effect depends on the threshold value. The parallel approach sometimes generates some gray blocks, this happens when the block being analysed contains a very low level of standard deviation. This effect usually occurs on the borders of the fingerprint or on areas that are not well defined.

```
def normalise(image):

    dbl_image = image.astype(float)
    norm_im = dbl_image

    # finding mean and standard deviation
    # Note: mean_stddev is a tuple where: mean = mean_stddev[0], std = mean_stddev[1]
    meanStd = cv2.meanStdDev(dbl_image)
    required_mean = 0
    required_stddev = 1

    # mapping coordinates where pixel is more or less than mean
    x0,y0 = np.where(norm_im >= meanStd[0])
    x1,y1 = np.where(norm_im < meanStd[0])

    # computing normalization
    norm_im = dbl_image - meanStd[0]
    norm_im = norm_im**2
    norm_im = norm_im/meanStd[1]

    # separating foreground from background
    if meanStd[1] > 20:
        norm_im[x0,y0] = required_mean + np.sqrt(required_stddev*norm_im[x0,y0])
        norm_im[x1,y1] = required_mean - np.sqrt(required_stddev*norm_im[x1,y1])

    else:
        norm_im[x1,y1] = 1 # 1 is the maximum desired value
        norm_im[x0,y0] = 1

    return norm_im
```

Figure 6.6: Code Snippet of MeanStd_Norm.py (Normalization System)

Also in this case the author is applying a sort of segmentation where all blocks who's standard deviation is below a given threshold of 20 are assigned a bright value, the author is assigning a value of 1 which is the desired maximum value. Modifying the threshold value will result in displaying the aforementioned gray blocks.

The strength of this algorithm resides in producing very high and very low values respectively for the valleys and the ridges hence greatly enhancing the contrast, these values are then used to calculate the local orientation at each pixel. The calculations made in the orientation algorithm produce better results when there is a high level of variation between the ridges and the valleys so this algorithm proves to be very efficient when calculating the gradient angle. The drawbacks, as mentioned before, are that this algorithm performs a very strong contrast enhancement that in some cases results in hiding the ridges structure, this could affect the ridge frequency calculation and therefore the construction of Gabor's filter.

6.3.3 IplImage Normalization

Figure 6.7: IplImage Normalization Results

Image	Min	Max
Original Image	0	255
IplImage Normalization	0.00416419	1.06187
Parallale IplImage Normalization	0.008764	2.23477

Table 6.3: IplImage Normalization Results

The IplImage approach proved to be the less effective. Although it does enhance the ridges the final contrast enhancement is retained to be too low to use this algorithm on the final fingerprint enhancement system. Especially when applying this algorithm in parallel on blocks of size 16X16 the resulting image is not well defined while the final system will produce better results if the ridges are well separated from the background. Figure 6.8 shows the code used for this algorithm, this was taken from [17] and adapted to suit the author's needs.

```
14 def normalise(image):
15
16     dbl_image = image.astype(float)
17     # calculate the mean of the image.
18     mean = np.mean(dbl_image)
19
20     # converting numpy 8-bit image to 8- bit cv2.iplimage
21     iplImage = cv2.cv.CreateImageHeader((image.shape[1], image.shape[0]), cv2.cv.IPL_DEPTH_8U, 1)
22     cv2.cv.SetData(iplImage, image.tostring(), image.dtype.itemsize * 1 * image.shape[1])
23
24     # initializing 32-bit floating point iplimage
25     image_32F = cv2.cv.CreateImage(cv2.cv.GetSize(iplImage), cv2.cv.IPL_DEPTH_32F,1)
26
27     # converting 8-bit unsigned integer image to 32-bit floating point image
28     cv2.cv.CvtScale(iplImage,image_32F)
29
30     # energy Normalization. Formula: image = image/mean(image)
31     cv2.cv.ConvertScale(image_32F, image_32F, (1/mean), 0);
32
33     # re-converting to numpy image
34     norm_im = np.asarray(image_32F)
35
36     return norm_im
```

Figure 6.8: Code Snippet of Ipl_Norm.py

6.3.4 Matlab's Histogram Equalization Experiment

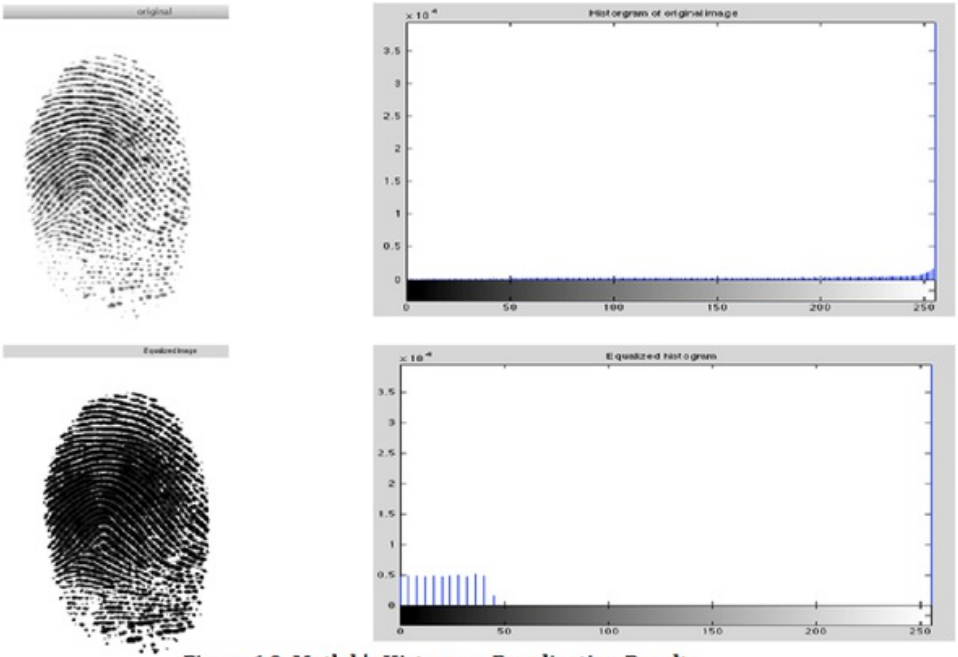


Figure 6.9: Matlab's Histogram Equalization Results

Matlab's experiment was conducted because the programming language allows to easily create plots. The author used a pre-written function **histeq()** to execute the equalization, a similar function is contained in OpenCv called **equalizeHist().**

```
10 -    I = imread(file);
11 -    figure,imhist(I),title('Historgram of original image');
12 -    qual_img = histeq(I);
13 -    figure, imshow(qual_img),title('Equalized Image');
14 -    figure,imhist(qual_img),title('Equalized histogram');
```

Figure 6.10: Code Snippet of Matlab Histogram Equalization

The aim of Matlab's experiment was to graph the image pixel occurrences and see the effect of the histogram equalization on a fingerprint image. Histogram Equalization was not considered for the final fingerprint enhancement system because it returned integer values ranging from 0 to 255, when the author was looking to process the image on a more "normalized" scale.

6.3.5 Conclusions

The first normalization algorithm to be excluded was the IplImage normalization because of its ineffectiveness. It was later necessary to compare Wikipedia's normalization to the mean and standard deviation approach suggested by Hong et al [11]. The decision was made taking into account how the normalization would impact the other algorithms:

1. The local orientation algorithm performs better when there is a very high difference between the ridges (minimum values) and the valleys (maximum values), therefore a mean and standard deviation approach would definitely return better results.
2. The local ridge frequency, on the other hand, computes its calculations on a wave which is constructed by analysing the ridges occurrences in a block of 32X32 pixels. Therefore these calculations are heavily impacted by the ability to recognize the ridge regions which, in the mean and standard deviation approach are sometimes hidden.

After these considerations the author came to the conclusion to use a parallel mean and standard deviation approach even though it meant losing some details on the ridges because it returned the strongest contrast and a better distinction between ridges and valleys.

6.4 Sobel's Edge Detection System

Sobel's edge detection algorithm required the creation of a Gaussian blurring filter and a Sobel filter. Although these filters are included in Python's module Scipy the author insisted in creating them to gain a better understanding of their functionalities and how they are affected by the parameters. The implementation results are explained in the following paragraphs followed by a comparison between Sobel's and Canny's edge detection algorithms. It is to be noted that this edge detection approach was not included in the final fingerprint enhancement system because it didn't return useful results to the final objective although some of its components such as the Gaussian and Sobel filters were re-used in some algorithms.

6.4.1 Gaussian Blurring Filter

Figure 6.11: Gaussian Blurring Results (a) Original Image (b) Low Blurring effect (c) Severe Blurring effect

Figure 6.11 illustrates the effect of a Gaussian filter applied on a non-normalized fingerprint image. The filter's effect is controlled by two parameters: the *sigma* and the *window size*. Increasing the sigma will produce a heavier blurring effect while increasing the window size will convolve more pixels also resulting in a heavier blurring. An example of the Gaussian kernel created for Figure 6.1 (b) using a sigma of 1 and a window size of 5 is shown in

Table 6.4. The kernel was created using the formula explained in ***Chapter 3.4.1,*** the code used for convolution is a Scipy function.

0.00296902	0.01330621	0.02193823	0.01330621	0.00296902
0.01330621	0.0596343	0.09832033	0.0596343	0.01330621
0.02193823	0.09832033	0.16210282	0.09832033	0.02193823
0.01330621	0.0596343	0.09832033	0.0596343	0.01330621
0.00296902	0.01330621	0.02193823	0.01330621	0.00296902

Table 6.4: Gaussian Kernel

```
# creating an empty kernel
kernel = np.zeros((window,window))
# centre of the kernel
c0 = window // 2

# computing kernel using Gaussian function
for x in range(window):
        for y in range(window):
            # calculating magnitude of the centre pixel. x^2 + y^2
            r = np.hypot((x-c0),(y-c0))
            # computes gaussian filter
            val = (1.0 / 2 * np.math.pi * sigma) * np.math.exp(- (r * r) / ( 2 * sigma * sigma ))
            kernel[x,y] = val

kernel = kernel / kernel.sum()
print kernel
# executes convolution
blurred_im = conv(image,kernel)[1:-1,1:-1]

return blurred_im
```

Figure 6.12: Code Snippet of Filters.py

6.4.2 Sobel Filter

Sobel's filter is used to calculate the first derivatives along the *x* and *y* axis of the image. Figure 6.12 shows how the filter affects the image, the horizontal filter returned an image which seems almost corrupted on the horizontal direction, while the vertical filter returned an image which appears corrupted along the vertical direction. This result was expected because the gradients are later going to be used

Figure 6.13: Sobel Gradients Results

as x and y components to calculate the intensity gradient and the orientation angle of the pixels.

Figure 6.14 illustrates the creation of the filters and their application on the image. The author created the filters manually according to the specifications explained in ***Chapter 3.4.2*** and then used Scipy's convolution function to convolve Sobel's kernel and the image. It was decided to use a 3X3 kernel because the same kernel is going to be used later on blocks of size 16X16, therefore a kernel of size 5X5 is retained to be too big for such blocks.

```
65      # creating sobel kernels
66      fx = np.array([1, 2, 1,
67                     0, 0, 0,
68                     -1,-2,-1])
69      fy = np.array([-1,0,1,
70                     -2,0,2,
71                     -1,0,1])
72
73      fx = fx.reshape(3,3)
74      fy = fy.reshape(3,3)
75
76      # convolving kernels
77      grad_x = conv(image,fx)
78      grad_y = conv(image,fy)
```

Figure 6.14: Code Snippet of Filters.py

6.4.3 Thresholding

The thresholding process is in charge of selecting the edges and enhancing them. The results of using two thresholds of values 170 and 200 on an image blurred with a sigma of 1 and a window of 5X5 can be seen in Figure 6.15 The edges detected using a higher threshold are not as defined as the ones found using a lower threshold.

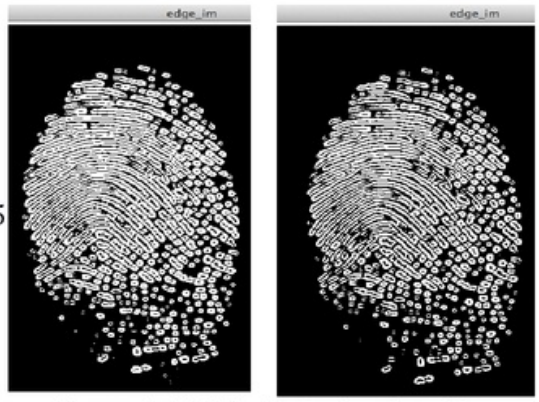


Figure 6.15: Edge Detection Results

6.4.4 Conclusion

Figure 6.16: Comparison between (a)Canny's and (b)Sobel's Edge Detection Algorithms

Figure 6.16 (a) illustrates the results of Canny's edge detection algorithm implemented in Matlab using the code shown in Figure 6.17. The **edge(I, 'Canny', [0.2, 0.4], 1)** function takes as an input the image ***I*** , the threshold values 0.2 and 0.4 and the sigma for the 5X5 gaussian window set as 1. It can be seen that Canny's algorithm detects thinner edges hence it proves to be more accurate and less influenced by the image's noise. As a conclusion the author does not recommend edge detection algorithms for fingerprint enhancement as they are not very useful to the final objective but, if the reader finds himself in the need to trace the edges, Canny's algorithm is definitely recommended over Sobel's one.

```
clc
clear
close all

path ='C:\*.png';
[name,p]=uigetfile(path); %select path

file = [p, name];
I = imread(file);

BW = edge(I,'canny',[0.2,0.4],1);

figure, imshow(BW),title('edge_im Matlab');
```

Figure 6.17: Code Snippet of Canny's algorithm in Matlab

6.5 Fingerprint Enhancement System

The Fingerprint Enhancement System follows Hong et al's [11] approach starting from the normalization of the image to enhance its contrast, then calculating the local orientation at each pixel, computing the local frequency estimation, binarizing, and finally thinning the fingerprint image. The author did not complete the system due to time constraints but implemented part of it. Since the choice for the normalization algorithm has been explained in ***Chapter 6.2.2***, the following paragraphs are going to go through all the other choices made when implementing Hong's algorithms in parallel explaining why certain parameters were chosen over others.

6.5.1 Threading

It was decided to implement a parallel approach by dividing the image in blocks of size 16X16, these dimensions were chosen based upon the image size 640X480. The sub-images being so created are 1,200. The number of threads should be chosen based on the hardware capacity, the more processors the more threads can be created.

6.5.2 Filter Components

The two filters being used from the orientation estimation algorithm are the Gaussian blurring filter and Sobel's filter. These will work on sub-images of size 16X16 hence the kernels for the filters must be relatively small, the author chose to use kernels of size 3X3 for both filters. The Gaussian filter also needed a sigma parameter which was set to 0.5, the author was looking for a very light blurring effect.

6.5.3 Local Orientation Estimation

The local orientation estimation algorithm followed Hong's et al specifications explained in ***Chapter 3.3.1.*** The code in Figure 6.18 shows the operations performed on each block to determine the local orientation, the filters construction and effects have been explained in ***Chapter 6.3.1*** and ***Chapter 6.3.2*** . It is to be noted that when calculating the double angles *Tx*

and *Ty* the author added a 0.001 offset to the gradient magnitude *G*, this is to avoid a value of 0 which would impact negatively when calculating the denominator.

```
66  def calculateOrientation(image):
67
68      # smooth image with gaussian blur
69      smoothed_im = Filters.gaussFilter(image, 0.5, 3)
70
71      # calculate gradients with sobel filter
72      dx,dy = Filters.sobelFilter(smoothed_im)
73
74      # smooth gradients
75      Gx = Filters.gaussFilter(dx,0.5,3)
76      Gy = Filters.gaussFilter(dy,0.5,3)
77
78      # compute gradient magnitude
79      Gxx = Gx **2
80      Gyy = Gy **2
81      G = np.sqrt(Gxx + Gyy)
82
83      # calculate theta
84      theta = np.arctan2(Gy,Gx)
85
86      # smooth theta
87      smoothed_theta = Filters.gaussFilter(theta, 0.5, 3)
88
89      # calculate double sine and cosine on theta --> increases precision
90      Tx = (G**2 + 0.001) * (np.cos(smoothed_theta)**2 - np.sin(smoothed_theta)**2)
91      Ty = (G**2 + 0.001) * (2 * np.sin(smoothed_theta) * np.cos(smoothed_theta))
92
93      denom = np.sqrt(Ty**2 + Tx**2)
94
95      Tx = Tx / denom
96      Ty = Ty / denom
97
98      # smooth theta x and y
99      smoothed_Tx = Filters.gaussFilter(Tx, 0.5, 3)
100     smoothed_Ty = Filters.gaussFilter(Ty, 0.5, 3)
101
102     # calculate new value for theta
103     theta = np.pi + np.arctan2(smoothed_Ty,smoothed_Tx)/2
```

Figure 6.18: Code Snippet of Orientation.py

After the orientation angle has been computed the program will draw lines that follow the ridges directions. This last step has not been well coded but does not impact further algorithms because it is used just to display the results, its inaccuracy lies in two points:

1. The angle with which the line is drawn is not computed properly. Approximating an average angle by making a simple sum and dividing by the number of pixels is not a good method to use as their might be too much variation between angles (e.g, how can you determine the average orientation when you have an angle of 0° and another of 90°?).

2. The author tried to mitigate this problem by dividing the 16X16 sub-image in four sub-images of size 8X8 but this resulted in drawing four lines for each block, which is not a proper solution because it affects the results visibility.

After some attempts the author decided to move on in the project as this function is needed just to show the results and will not have any effect on the other algorithms. Furthermore, even if not completely accurate, the function returns an approximation which can still be used to display the results of the orientation calculation. It is to be noted that the performance is severely impacted and in a final solution, where there is no need to draw the lines the lines, this function should be removed.

Figure 6.19 illustrates the results of applying the orientation estimation algorithm in parallel on a normalized image.

Figure 6.19: Orientation Estimation Results

Figure 6.19 shows how the ridges change directions in the image, looking at this fingerprint from left to right the ridges clearly follow an angle of approximately 45° and change direction at the centre of the print to an angle of about -45°.

6.5.4 Benchmarks

Figure 6.20 illustrates the time of execution of the software when computing the normalization and orientation algorithm. As can be seen the speed of execution depends on the number of threads. Due to hardware restrictions it was not possible to experiment on multi CPU systems. It is visible that when increasing the number of threads the computation time increases as well, this is due to the overhead of managing threads on a single processor, the results should improve when testing on multi CPU systems. It is also possible to speed up the computations by initially segmenting the image and having only the regions of interest be computed instead of the entire image, this would result in less data being computed hence better performance.

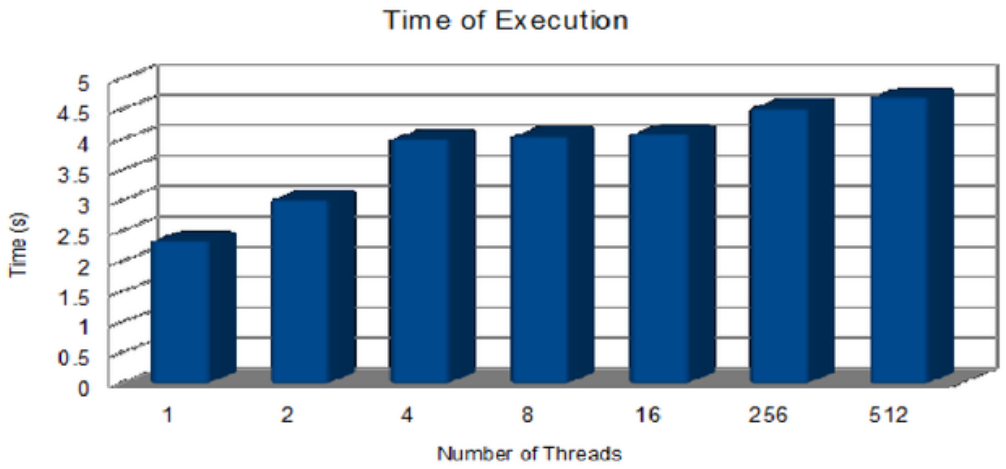


Figure 6.20: Benchmarks

Benchmarks							
Number of Threads	1	2	4	8	16	256	512
Time (s)	2.34113	3.017806	4.01512	4.05938	4.095505	4.521856	4.70088

Table 6.5: Benchmarks

6.5.5 Conclusions

The implementation of the Fingerprint Enhancement System has not been completed due to time constraints, the missing components are: the ridge frequency estimation, the Gabor filter, the binarization and the thinning algorithms. The ridge frequency estimation algorithm is the only one that can be executed in parallel, it is suggested to use blocks of size 32X32. Furthermore it is suggested to experiment segmenting the image and computing only the regions of interest as this would greatly benefit performance.

Chapter 7: Testing

This chapter explains how the systems testing was carried out. It will go through white-box testing, testing the algorithms, and black-box testing presenting some test cases.

7.1 White-Box Testing

White box testing was carried on through the entire development process using Python's standard unit testing framework called PyUnit, it is based on Java's JUnit hence it uses a proven testing architecture. Throughout the project white-box testing has been carried on mostly using code inspection, each component has been inspected using the checklist shown in Table 7.1 [28].

Test Case	Pass/Fail
1 Data Fault Prevention	
Check that all variables are declared and initialized or assigned before their values are used	
Check that all constants are named	
Check that array upper bounds are not exceeded	
Check that division by zero errors cannot occur	
Only compare variables of compatible types	
2 Control Fault Prevention	
Check that branches don't occur too soon or too late	
Check that each if..else condition can be reached	
Check that each loop can terminate	
3 Interface Fault Prevention	
Check all function/method parameters are used in the right sequence	
Check all function/method parameters have the intended type	
Check all shared variables are properly managed	
Check that all external functions have been used correctly	
4 Memory and File Fault Prevention	
Check that literal strings have enough space allocated	
Check that dynamic memory has been allocated correctly	
Check that files have been opened/modified/closed correctly	

Table 7.1: White Box Testing Checklist

7.2 Testing the Algorithms

Testing the algorithms functionalities was probably the most important phase in this project as it heavily relies on their results. In image analysis these kind of tests look for specific computed values that can either be displayed as a resulting image, or graphed (e.g. using a histogram). Since part of the project involved experimenting different algorithms some systems such as the normalization system and the edge detection system became actual test cases to determine the algorithms functionalities, their results are explained in the Implementation in ***Chapter 6.*** Testing the orientation estimation algorithm proved to be an extremely hard task due to the fact the fingerprint images used for testing are of medium-low quality hence it became hard to interpret its results. It was therefore necessary to test this algorithm on a synthetic image.

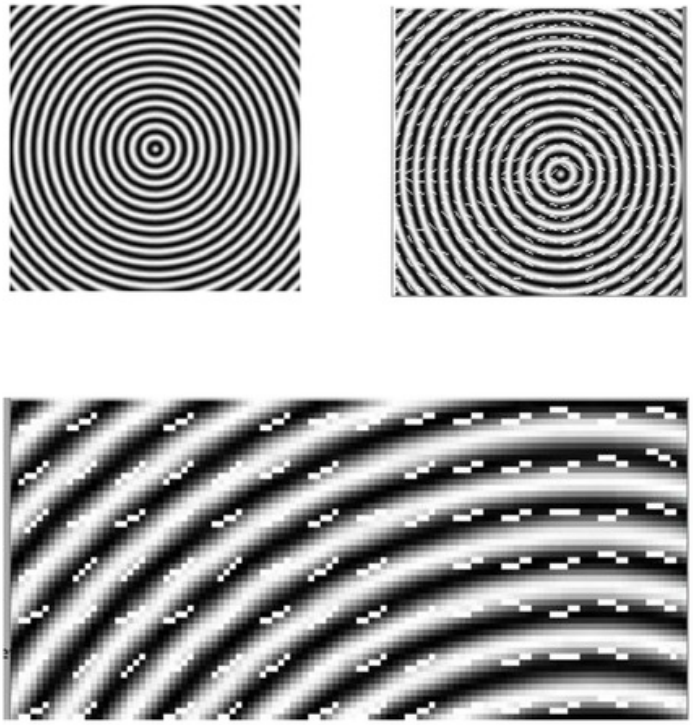


Figure 7.1: Orientation Estimation Algorithm Test

As shown in Figure 7.1 the algorithm definitely detects the orientation angle, even though the lines display some imperfections, it can be assumed that the algorithm is computing the appropriate resultts. As explained in ***Chapter 6.4.3*** the lines are not drawn accurately so minor orientation variations were expected especially when drawing in the vertical direction as the approximations computed are not very accurate.

7.3 Black-Box Testing

Since the algorithms are optimized for 640X480 images and their functionality was tested using white-box testing, Black-Box testing came down to test that all the functional requirements were met and that the input was validated before the software could execute the algorithms. The test-cases used for Black-Box testing on each system are explained in this paragraph together with some results.

7.3.1 Testing the Normalization System

Name: Test Functional Requirements (Normalization System)	
Purpose:	To test if the system was performing the requested operations and displaying the correct output
Testing Actions:	Provide an ideal input: a fingerprint image of size 640X480
Notes:	This test was performed to check if there were any problems during an ideal execution and to see if the system met all the functional requirements.

Table 7.2: Normalization System Functional Requirement Test

The system passed the test meeting all functional requirements.

Name: Test Input Validation (Normalization System)		
Purpose:	To test if the system detects if someone is trying to submit an invalid input	
Testing Actions:		
Input Data:	**Expected Result:**	**Actual Result:**
Fingerprint image of size 256X338 .png format	Error Message	**Error: Size of the image is not supported. Supported size: 640x480
Fingerprint image of size 640X480 .tiff format	Error Message	**Error: Invalid file type. The software supports only .png files
Non existing file	Error Message	**Error: File not found
Invalid number of arguments: 2 > arguments > 1	Error Messag	**Error: Invalid arguments. Usage: python NormalizationSystem.py <filename>
Notes:	This test was performed to check if the system was validating the user's input before applying the algorithms.	

Table 7.3: Test Input Validation Normalization System

7.3.2 Testing Sobel's Edge Detection Algorithm

Name: Test Functional Requirements	
Purpose:	To test if the system was performing the requested operations and displaying the correct output
Testing Actions:	Provide an ideal input: a fingerprint image of size 640X480, a gaussian sigma of 1, a gaussian window size of 3, and a threshold value of 100
Notes:	This test was performed to check if there were any problems during an ideal execution and to see if the system met all the functional requirements.

Table 7.4: Functional Requirements Test Sobel's Edge Detection System

The test was successful, the system complies to all functional requirements.

Name: Test Input Validation		
Purpose:	To test if the system detects if someone is trying to submit an invalid input	
Testing Actions:		
Input Data:	**Expected Result:**	**Actual Result:**
Fingerprint image of size 256X338 .png format	Error Message	**Error: Size of the image is not supported. Supported size: 640x480
Fingerprint image of size 640X480 .tiff format	Error Message	**Error: Invalid file type. The software supports only .png files
Non existing file	Error Message	**Error: File not found
Invalid number of arguments: arguments more or less than 5	Error Message	**Error: Invalid arguments. Usage: python NormalizationSystem.py <filename>
Specified sigma is not a float or int	Error Message	**Error: Parameter. Specified sigma is not a float or an int
Specified window size is not an int	Error Message	**Error: Parameter. Specified threshold is not an int
Specified threshold is not an int	Error Message	**Error: Parameter. Specified threshold is not an int
Specified window size is not 3 or 5 or 7	Error Message	**Error: Parameter Error. The dimensions of the gaussian window are not supported. Supported values for the windows are: 3, 5, 7
Notes:	This test was performed to check if the system was validating the user's input before applying the algorithms.	

Table 7.5: Test Input Validation Sobel Edge Detection System

7.3.3 Testing the Fingerprint Enhancement System

Name: Test Functional Requirements (Fingerprint Enhancement System)	
Purpose:	**To test if the system was performing the requested operations and displaying the correct output**
Testing Actions:	**Provide an ideal input: a fingerprint image of size 640X480**
Notes:	**This test was performed to check if there were any problems during an ideal execution and to see if the system met all the functional requirements.**

Table 7.6: Test functional requirements Fingerprint Enhancement System

The test was not successful as FR1, FR3, and FR4 were not met.

Name: Test Input Validation (Fingerprint Enhancement System)		
Purpose:	To test if the system detects if someone is trying to submit an invalid input	
Testing Actions:		
Input Data:	**Expected Result:**	**Actual Result:**
Fingerprint image of size 256X338 .png format	Error Message	**Error: Size of the image is not supported. Supported size: 640x480
Fingerprint image of size 640X480 .tiff format	Error Message	**Error: Invalid file type. The software supports only .png files
Non existing file	Error Message	**Error: File not found
Invalid number of arguments: 2 > arguments > 1	Error Messag	**Error: Invalid arguments. Usage: python NormalizationSystem.py <filename>
Notes:	This test was performed to check if the system was validating the user's input before applying the algorithms.	

Table 7.7: Test Input Validation Fingerprint Enhancement System

Chapter 8: Known Problems

8.1 The Convolution Bug

When running Sobel's Edge Detection system and the Fingerprint Enhancement system the output will show a ComplexWarning which looks as follows:

```
scipy/signal/signaltools.py:422: ComplexWarning: Casting complex values to real discards the imaginary part

  return sigtools._convolve2d(in1, in2, 1, val, bval, fillvalue)
```

This is a known bug that occurs when applying convolution filters on the image using Python's module Scipy v0.11.0. It reports a warning without interfering with the software's functionality, This bug was fixed in v0.12.0.

8.2 Drawing the Lines

As explained in ***Chapter 6.4.3*** a problem encountered when developing the full Fingerprint Enhancement System was drawing the lines that represented the local orientations. To the author's embarrassment this problem does not occur because the algorithm is being computed wrong, it occurs because the calculations being computed just for the aim of drawing the lines are very roughly approximated. This block of code should be replaced entirely, the author suggests the following approach which he did not have time to implement:

Given that the orientation fields have been calculated correctly and considering the ridges contained in a block of size 16X16, it is be possible to segment the background (valleys) from the foreground (ridges) with a simple thresholding process. Once the developer knows information on the ridges, he is able to spot the ridges boundaries hence, with some calculations he will find the centre of the ridge. Note that there could be more ridges, so detect the centre of all the ridges. Now draw the lines making them pass through the centre pixel of the ridge and giving them the previously calculated orientation field as an angle. It is

also suggested to draw the lines on a brightened version of the image so to increase the line visibility.

It is to be noted that drawing the lines will not affect in any way the software's functionality as they are drawn just to display the orientation field results. Due to this reason the author preferred focusing on other aspects of the system regardless of the lines.

Chapter 9: Related Work

Automated Fingerprint Analysis tools have been developed since the 70's, various algorithms have been suggested and implemented in a number of systems, furthermore parallel image analysis is a broad subject which sees a number of applications in different fields making this project related to a very wide range of other projects. From the algorithms point of view it is directly related to the NIST software, Hong et al's Fingerprint Enhancement Project [11], RaymondThai's project [10], and a GPU cluster fingerprint enhancement project [20]. The similarities between these systems are the algorithms being computed, even though the NBIS software computes additional components and is by far the most accurate among these systems NIST approach is based on the same concepts as the other ones. Other projects to be mentioned that belong to this category are all the projects presented at the Fingerprint Verification Competition (FVC) which is an international competition based on the science of fingerprint analysis and a number of Biometric applications which use some of the concepts used in this project. It is to be noted that from a programming language point of view this project does not relate very well to the aforementioned implementations because most of them have been developed either in C/C++ or Matlab.

Chapter 10: Further Work

Due to time constraints the implementation of the Fingerprint Enhancement system was not completed hence the first step to work on would be the completion of the system. Furthermore the author has a number of ideas to enhance this project that lead in various directions:

1. Add boundary conditions
2. Add supported file formats such as JPEG, Tiff, and the ANSI/NIST-ITL 1-2007 format.
3. Add support for variable image dimensions.
4. Extend to a full fingerprint recognitions software by adding various components such as a minutiae extracting algorithm, a categorizing algorithm such as 'pcasys' from the NBIS software and a fingerprint matching algorithm such as 'bozorth3'.
5. Increase the software's reliability by assessing the algorithms output against quality maps in a similar way to the NBIS software.
6. Increase its performance by analysing only the regions of interest excluding the background. This could be accomplished by calculating the actual fingerprint's boundaries in the image and isolate it from the background, this would result in processing less pixels hence a high increase in performance. It is to be noted that the new size of the fingerprint is to accommodate the 16X16 bocks.
7. Further increase its performance by extending the system to run on GPU's and then on clusters of GPU's. Python's module PyCuda is a good point to start from.
8. Replace computationally expensive code with C/C++ code and integrate the new functions in Python's environment. The author suggests creating Python interfaces using SWIG for this purpose. This would increase the performance even further.

9. Create a distributed system which analyses entire databases instead of single images.
10. Evolve the project by creating a mobile app which could send a fingerprint image to a server that would process it and send back the results, this would result in an extremely portable and valuable product.

Chapter 11: Critical Evaluation

The critical evaluation of this project is explained in this chapter. Accomplishments and issues are discussed together with an evaluation and lessons learnt throughout the project.

11.1 Achievements

The project topic was decided in early October 2012, at the time the author had no experience in image processing or parallel computing but accepted the challenge seizing the opportunity to develop knowledge in the aforementioned fields of study. On one hand image processing opens the way to a number of applications including biometric applications, on the other hand a good understanding of high performance computing allows developers to increase the quality of their products furthermore this field can be applied to an extremely broad range of applications and the author retains that this is an essential skill to develop especially nowadays, when hardwares are reaching speed limits it is up to developers to make the best use of the available resources. This said, the project started with a research phase that led the author to discover new technologies and understand how they can be exploited for different needs. A good example of these findings is Amazon Cloud Computing, this platform allows to rent equipment that suits the developers needs at reasonable costs and allows to easily build distributed systems, its technology can be exploited to develop very scalable systems while experimenting with hardware that would else be very hard or expensive to find. It is to be noted that more and more companies are migrating to a cloud environment and developing understanding in this technology is an achievement for future works. The research then brought to image analysis, a topic never encountered during the course of study which the author is proud of having undertaken as this is the perfect topic to apply parallel computing to as it requires an extensive amount of data to be processed. The biggest achievement in this field lies in the fact that the author developed strong fundamentals in the subject demonstrating to be able to understand complex algorithms that, apart from fingerprint analysis, can be implemented in a number of applications.

Throughout the project it was also decided to use Python as a programming language, a language which the author had never used before and that is suitable for many branches of development: from web development to high-performance computing and imaging software. It's interesting properties include the ability to import pre-written C/C++ functions and to integrate Matlab's functions in the working environment allowing the developer to minimize the language's weaknesses. Other than Python, the author experienced using Matlab to analyse some algorithms throughout the project, this too is an achievement worth mentioning as Matlab is today the most used programming language for imaging software, especially for biometric applications. Finally the author is proud to hand in his research which code is highly extensible, maintainable and re-usable thanks to the Object Oriented approach he adopted throughout the design.

11.2 Criticism

From a developer's point of view a system is never perfect and never satisfies entirely the developers expectations. By the time that the project comes to an end the developer has gained more knowledge on the topics and experience in how to carry out all the tasks from the conception to the implementation, looking back at the initial stages the criticism goes to the research phase, the author spent too much time focusing on analysing existing source code such as the NBIS solution when instead he should have focused on researching other algorithms or best practices. Another critic goes to the testing phase which wasn't carried out properly due to hardware restrictions, these restrictions should have been foreseen and the software should have been tested on distributed systems and on different OS's. On the implementation side the Fingerprint Enhancement System is not complete but to the point the author got to the software performs correctly. Finally the last critic goes to the portability factor, currently the software runs only on Mac OSX with python and all the required modules installed. Python though provides ways to build the software and create

executables that work independently from the modules, this can be accomplished by using components such as PyInstaller which creates executables for Windows, Linux and Mac's.

11.3 Implementation Evaluation

The implementation has been evaluated according to McCall's software qualities triangle. Hence the next paragraphs are going to illustrate the quality of the product's operations, revision and transition.

11.3.1 Correctness

The developed prototypes meet the requirement specifications and were used to test the algorithms hence the correctness is proved in the implementation ***Chapter 6*** and in the testing ***Chapter 7***. The fingerprint Enhancement System does not meet all the requirements but this is due to the fact that the system was not complete in time. To the point the author got to it does however compute the appropriate algorithms returning the correct results, the tests for the algorithms performance are listed in ***Chapter 7.***

11.3.2 Reliability

The reliability was tested using Black-Box testing and White Box testing. White Box testing proved to be especially useful in cases that presented division by zero computations. The complete testing review can be found in ***Chapter 7.***

11.3.3 Efficiency

Thanks to the use of multi-threading the system is highly efficient. Its efficiency can still be enhanced in various ways partly described ***in Chapter 6*** and ***Chapter 10*** . The main point to consider in the fingerprint enhancement system is the image segmentation before the algorithms are applied; by considering just the regions of interest and excluding the background the system would compute much less data resulting in a better performance. Another way to increase efficiency is to remove the "drawing lines" code as this is useless to the final objective and computationally very expensive.

11.3.4 Usability

The systems were not designed to meet usability needs, there are no GUI's therefore their usability is very limited. It was the author's intention not to provide users with GUI's as he retained that in this case a GUI would have only affected performance and would have been useless to a final objective of enhancing the image.

11.3.5 Maintainability

Is a quality factor that the author is proud to say he has accomplished. The code is well commented throughout all the systems, the UML diagrams are clear and understandable, and the components are well separated so the system can undergo changes at any time. Furthermore this document provides all the information needed to understand the systems thoroughly.

11.3.6 Testability

Testing the systems components results in an easy task as the Normalization system and the Sobel Edge Detection system display all the results of the algorithms. The fingerprint enhancement system makes use of some of the components of the other systems therefore it is possible to test its components directly on the other systems while it displays results of the algorithms that belong just to it. For further information the complete testing can be found in ***Chapter 7.***

11.3.7 Re-Usability

All components of the systems were designed to be highly re-usable. All the algorithms can be re-used in completely different systems as they are without any changes apart from the thread controllers that are optimized for images of size 640 X 480.

11.3.8 Portability

Is a factor that was not considered until the end of the project when it was too late to rectify this feature. At the moment the systems can only be executed on systems that have a Python 2.7 installation with the following modules installed: Numpy, PIL, SciPy, and OpenCv. Portability can be reached through PyInstaller which is a utility that is able to build the systems to Windows executables, Linux binaries, and Mac OS X applications.

11.3.9 Interoperability

Interoperability was not considered throughout the project as the developed systems are not supposed to interact with any other system. The only point in common that would enable an

external system to communicate with the fingerprint enhancement system would be the output of the enhanced fingerprint that could be processed by another system to extract minutiae.

11.4 Lessons Learnt

Throughout the project development the author learnt a lot of lessons, so many that they can't even be listed. Perhaps the most important lesson learnt is to always be ambitious and never give up.

Edoardo Foco

29 April 2013

Chapter 12: Appendix

Use Case Specification: Input Image (Normalization)	
Description:	The user inputs an image to normalize it according to the algorithms.
Pre-Conditions:	The user already has an image file to process.
Flow of Events:	
Basic Flow:	
	1. User inputs image.
	2. System validates input.
	3. System applies Wikipedia's algorithm to the image
	4. System applies the Mean and Standard Deviation algorithm to image.
	5. System applies the IplImage normalization algorithm.
	6. System applies a parallel implementation of the Wikipedia, Mean and Standard Deviation, and IplImage normalization algorithms
	7. System displays results
	8. User views results
	9. System terminates
Alternative Flow:	
	3. Systems detects a non-valid input.
	4. System displays Error message
	5. System terminates
Execeptions:	
	3. System displays the help page
	4. System terminates
Post-Conditions:	
	The system has successfully computed the algorithms and the user is able to compare the results

Table 12.1: Use Case Specification – Input Image (Normalization System)

Use Case Specification: See Help (Normalization)	
Description:	The user wants to see the help
Pre-Conditions:	User inputs a '-h' as an argument
Flow of Events:	
Basic Flow:	
	1. System validates input.
	2. Help is displayed
	3. System terminates
Alternative Flow:	
	3. Systems detects a non-valid input.
	4. System displays Error message
	5. System terminates
Execeptions:	
	None
Post-Conditions:	
	The user views the Help

Table 12.2: Use Case Specifcation- See help (Normalization System)

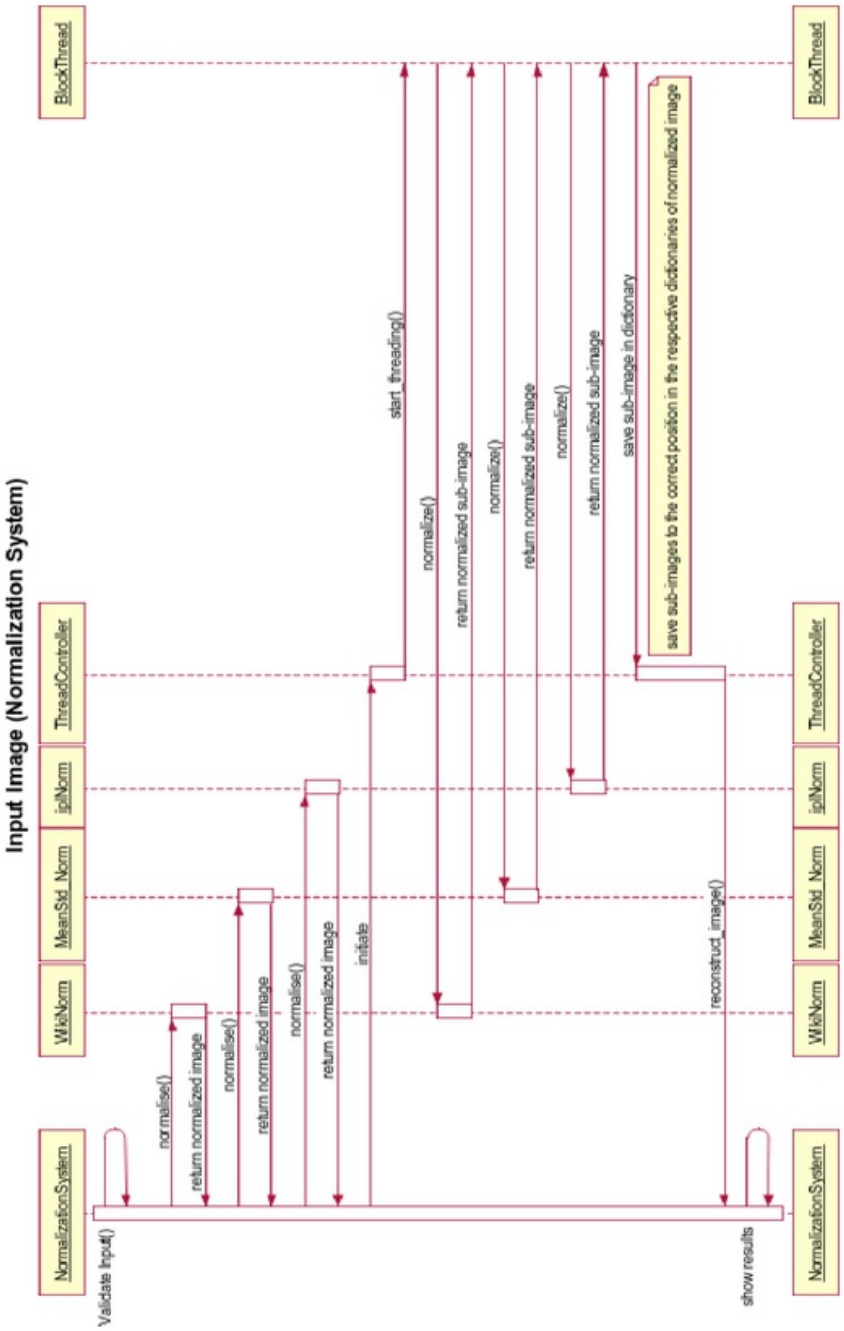


Figure 12.1: Normalization System Sequence Diagram

Use Case Specification: Input Image (Sobel Edge Detection)	
Description:	The user wants to apply Sobel's Edge Detection algorithm to an image
Pre-Conditions:	User has an input image he wants to analyse
Flow of Events:	
Basic Flow:	
	1. User inputs image
	2. System validates input
	3. System asks to input parameters [see use case: Insert Parameters]
Alternative Flow:	
	3. Systems detects a non-valid input.
	4. System displays Error message
	5. System terminates
Execeptions:	
	3. System displays the help page
	4. System terminates
Post-Conditions:	
	The system has successfully validated the input and asks the user to input parameters

Table 12.3: Use Case Specification – Input Image (Sobel Edge Detection System)

Use Case Specification: Input Parameters (Sobel Edge Detection)	
Description:	The user is asked to insert the parameters for the calculations
Pre-Conditions:	User has inputted the image [see Use Case: Input Image]
Flow of Events:	
Basic Flow:	
	1. System asks to input the parameters
	2. User inputs the parameters
	4. Systems applies Sobel Edge Detection algorithm
	5. System displays results
	6. System terminates
Alternative Flow:	
	3. Systems detects a non-valid input
	4. System displays Error message
	5. System terminates
Execeptions:	
	None
Post-Conditions:	
	The user views the results

Table 12.4: Use Case Specification – Input parameters (Sobel Edge Detection)

Use Case Specification: See Help (Sobel Edge Detection)	
Description:	The user wants to see the help
Pre-Conditions:	User inputs a '-h' as an argument
Flow of Events:	
Basic Flow:	
	1. System validates input.
	2. Help is displayed
	3. System terminates
Alternative Flow:	
	3. Systems detects a non-valid input.
	4. System displays Error message
	5. System terminates
Execeptions:	
	None
Post-Conditions:	

Table 12.5: Use Case Specifiaction – See Help (Sobel Edge Detection)

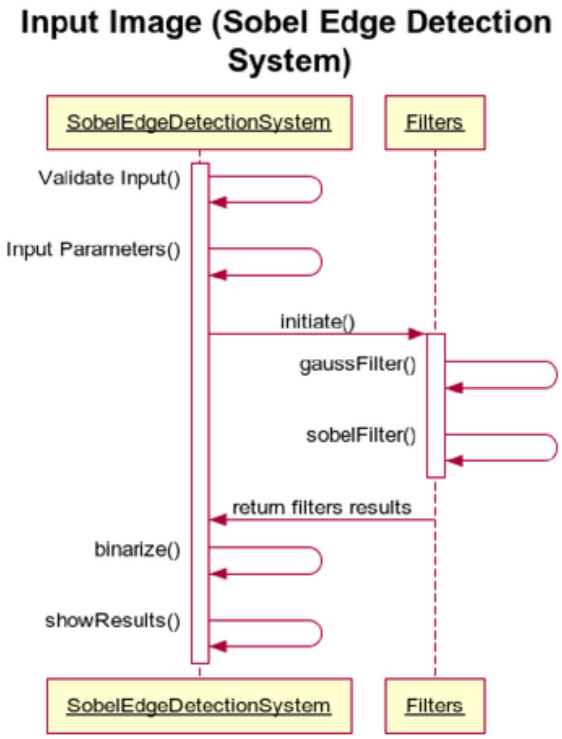


Figure 12.2: Sobel Edge Detection System Sequence Diagram

Use Case Specification: Input Image (Fingerprint Enhancement System)	
Description:	The user wants to enhance a fingerprint image
Pre-Conditions:	The user has a fingerprint image and wants to enhance it
Flow of Events:	
Basic Flow:	
	1. User inputs the image
	2. System validates user input
	3. System normalizes image using a parallel Mean and Standard Deviation normalization algorithm
	4. System computes the orientation estimation algorithm
	5. System computes the ridge frequency estimation algorit
	6. System applies Gabor's filter
	7. System binarises the image
	8. System applies a thinning algorithm
	9. System displays results
	10. System terminates
Alternative Flow:	
	3. Systems detects a non-valid input
	4. System displays Error message
	5. System terminates
Execeptions:	
	3. System displays help
	4. System terminates
Post-Conditions:	
	The user views the results and and is ready to apply a minutiae extraction algorithm

Table 12.6: Use Case Specification-Input Image (Fingerprint Enhancement System)

Use Case Specification: See Help (Fingerprint Enhancement System)	
Description:	The user wants to see the help
Pre-Conditions:	User inputs a '-h' as an argument
Flow of Events:	
Basic Flow:	
	1. System validates input.
	2. Help is displayed
	3. System terminates
Alternative Flow:	
	3. Systems detects a non-valid input.
	4. System displays Error message
	5. System terminates
Execeptions:	
	None
Post-Conditions:	
	The user views the Help

Table 12.7: Use Case Specification – Input Image (Fingerprint Enhancement System)

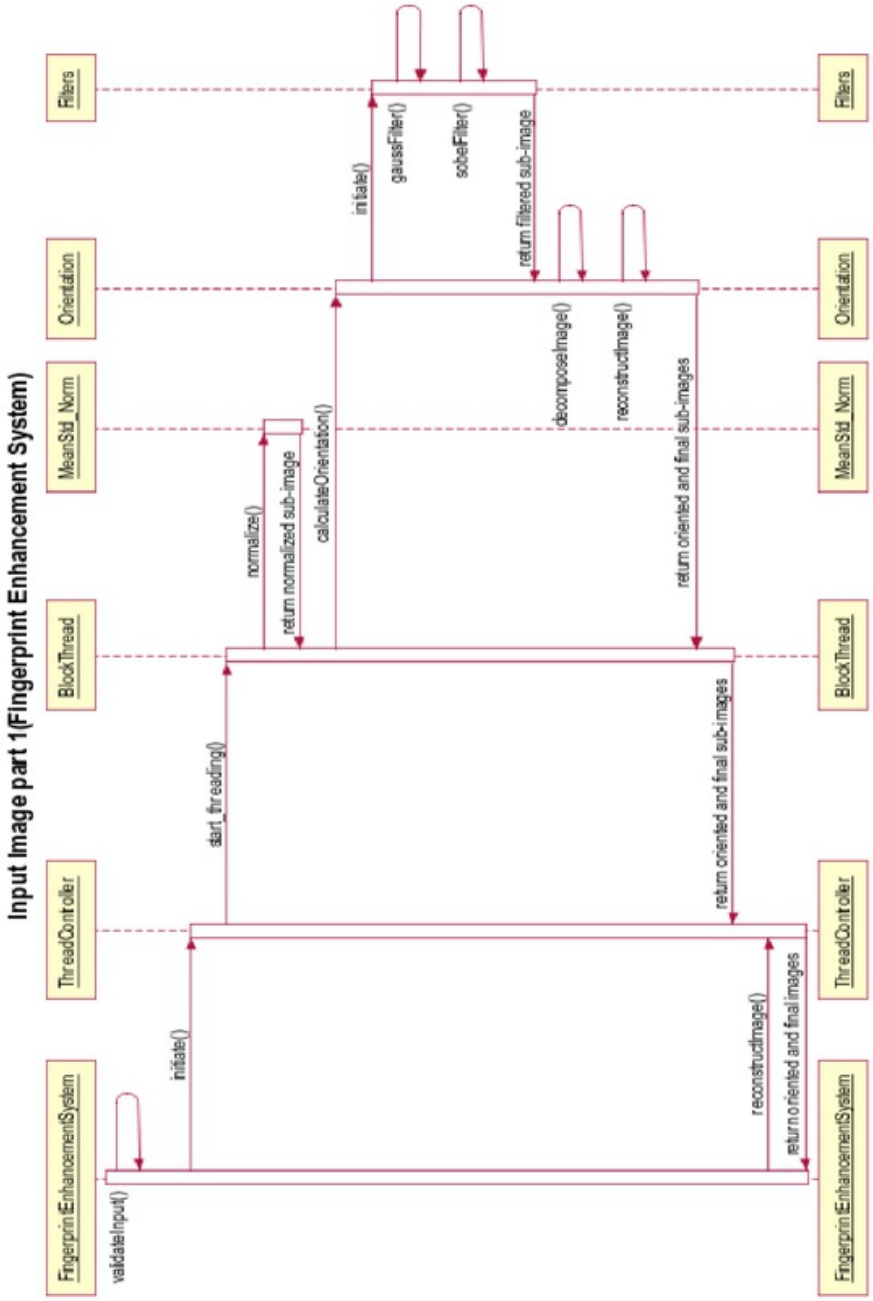


Figure 12.3: Fingerprint Enhancement System Sequence Diagram Part 1

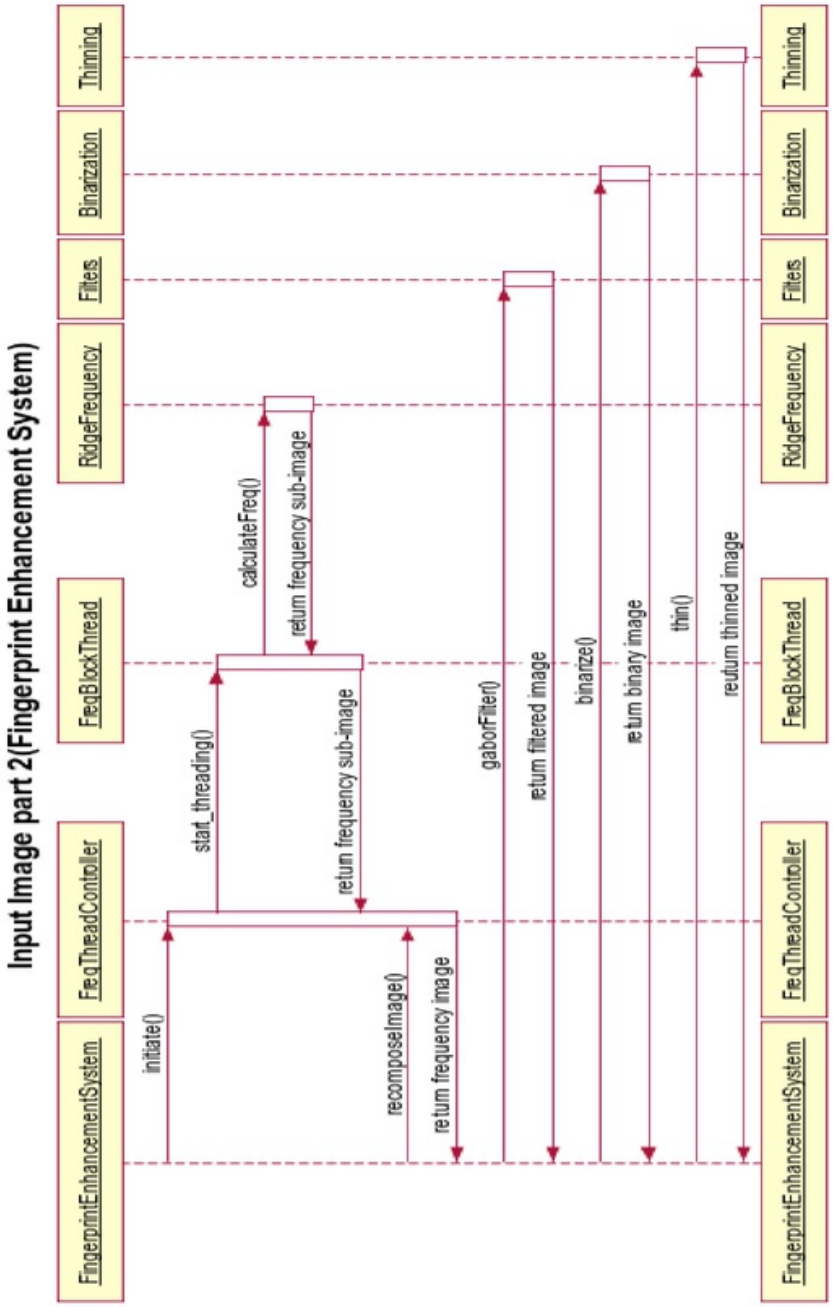


Figure 12.4: Fingerprint Enhancement System Sequence Diagram Part 2

Chapter 13: Manuals

13.1 Normalization System Manual

Prerequisites:

The software needs the following components to be installed on the system:

1. Python
2. OpenCv
3. Numpy
4. PIL

How to run the Normalization System:

The system can be run by executing the NormalizationSystem.py file from the command line and giving it the system path as an argument. The command would look as follows:

python NormalizationSystem.py <filename>

The system will compute the algorithms automatically and will display the results as images and as text in the command line.

Reading the output:

The output will display seven images and their respective minimum and maximum values is going to be displayed in the command line.

Terminating the System:

Once the system displays the output the user may press any key to terminate the execution.

13.2 Sobel Edge Detection System Manual

Prerequisites:

The software needs the following components to be installed on the system:

1. Python
2. OpenCv
3. Numpy
4. PIL
5. SciPy

How to run the Sobel Edge Detection System:

The system can be run by executing the SobelEdgeDetection.py file from the command line and giving it the system path as an argument. The command would look as follows:

python SobelEdgeDetection.py <filename>

The system is then going to ask for the parameters to use in the Gaussian filter and for the desired threshold value. The values to input for the gaussian filter are the sigma and the window size:

1. The sigma must be either a decimal or an integer and it is going to affect the blurring effect. Increasing the sigma will increase the blurring effect.
2. The window size must be an integer value and it is also going to affect the blurring effect. The values accepted by this parameter are 3, 5, 7 respectively creating gaussian kernels of size 3X3, 5X5, 7X7. Increasing the window size will increase the blurring effect.

The threshold value must be of type integer and is going to affect the detection of the edges. A low threshold value is going to detect weaker edges while a high threshold value will detect only the thinner ones.

Reading the Output:

The system is going to output one binary image. The edges are going to be enhanced by a white colouring.

Terminating the System:

Similarly to the Normalization system, once the system displays the output the user is allowed to press any key to terminate the execution.

13.3 Fingerprint Enhancement System Manual

Prerequisites:

The software needs the following components to be installed on the system:

6. Python
7. OpenCv
8. Numpy
9. PIL
10. SciPy

How to run the Fingerprint Enhancement System:

The system can be run by executing the FingerprintEnhancementSystem.py file from the command line and giving it the system path as an argument. The command would look as follows:

python FingerprintEnhancementSystem.py <filename>

The system is going to compute the algorithms automatically and display the results.

Reading the Output:

Currently the system is going to output two images. One image represents the normalized version of the original where the orientation of the pixels is depicted by white lines. The second image only represents the orientation of the pixels depicted as white lines on a black background.

Terminating the System:

The system can be terminated by pressing any key on the normalized image and then closing the orientation image.

Notes:

This system is not complete hence it does not compute all the algorithms. A complete system will output a normalized image, an orientation image, a binary image enhanced by gabor's filter and finally a thinned fingerprint image.

Chapter 14: References

[1] Anil K. Jain, Arun Ross and Salil Prabhakar "An Introduction to Biometric Recognition," Appeared in IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Image- and Video- Based Biometrics, Vol. 14, No. 1, January 2004.

[2] "History of Fingerprints." Crime Scenes Forensics, LLC. 2 Jan. 2013 <http://www.crimescene-forensics.com/History_of_Fingerprints.html>.

[3] Mr. Lazaroff. "Classification of Fingerprints." Classification of Fingerprints. 2 Jan. 2013 <http://shs.westport.k12.ct.us/forensics/04-fingerprints/classification.htm>.

[4] User's Guide to NIST Biometric Image Software (NBIS). Gaithersburg: National Institute of Standards and Technology. PDF.

[5]. Orion Adrian. "Re: C++ performance vs. Java/C#." Web log comment. StackOverflow. 20 Dec. 2012 <http://stackoverflow.com/questions/145110/c-performance-vs-java-c>.

[6] Bob Dowling. Interfacing Python with Fortran. University Computing Service. PDF

[7] Dr. Jeff Layton. "Scripting in HPC Using Python: Part 1 – Quick Introduction." High Performance Computing. DellTechCenter. 20 Nov. 2012 <http://en.community.dell.com/techcenter/high-performance-computing/w/wiki/2280.aspx>.

[8] Nikolaos Tountas. Secure File Share. Thesis. University of Westminster, 2007

[9] Bovik, Alan C. The essential guide to image processing. Burlington, MA: Academic P, 2009.

[10] Raymond Thai. Fingerprint Image Enhancement and Minutiae Extraction. Thesis. University of Western Australia, 2003.

[11] Hong, L., Wan, Y., and Jain, A. K. Fingerprint image enhancement: Algorithm and performance evaluation. IEEE Transactions on Pattern Anal- ysis and Machine Intelligence 20, 8 (1998)

[12] Gonzalez, Rafael C., Richard E. Woods, and Steven L. Eddins. Digital Image processing using MATLAB. Upper Saddle River, NJ: Pearson Prentice Hall, 2004. >.

[13]] Iwasokun Gabriel Babatunde, Akinyokun Oluwole Charles, and Olabode Olatubosun. A Block Processing Approach to Fingerprint Ridge-Orientation Estimation. Thesis. Department of Computer Science, Federal University of Technology, Akure, Nigeria, 2012.

[14] Mathworks. "Performing Distinct Block Operations." Weblog post. Mathworks. 25 Feb. 2013 <http://www.mathworks.co.uk/help/images/performing-distinct-block-operations.html#bs2shgg-1>.

[15] "Griaule Biometrics." FFT BASED PREPROCESSING. 22 Feb. 2013 <http://www.griaulebiometrics.com/en-us/book/understanding-biometrics/types/enhancement/based]

[16] "Normalization (image processing)." Wikipedia. 29 Mar. 2013. Wikimedia Foundation. 27 Feb. 2013 <http://en.wikipedia.org/wiki/Normalization_%28image_processing%29>.

[17] " DeveloperStation.ORG." Energy Normalize Image in OpenCV. 1 Mar. 2013 <http://www.developerstation.org/2011/04/normalize-image-in-opencv.html>.

[18] Anna Bruno, and Dario Maio. Impronte Digitali - Feature Extraction. Http://bias.csr.unibo.it. PPT.

[19] Asker M. Bazen, and Sabih H. Gerez. Directional Field Computation for Fingerprints Based on the Principal Component Analysis of Local Gradients. Thesis. University of Twente, Department of Electrical Engineering, Laboratory of Signals and Systems.

[20] N.P. Khanyile, J.-R. Tapamo, and E. Dube. Distributed Fingerprint Enhancement on a Multicore Cluster. Thesis. University of KwaZulu-Natal, South Africa.

[21] Peter Kovesi. "Example of fingerprint enhancement." Example of fingerprint enhancement. 27 Feb. 2013 <http://www.csse.uwa.edu.au/~pk/research/matlabfns/FingerPrints/Docs/index.html>.

[22] "Gaussian Smoothing." Spatial Filters. 5 Mar. 2013 <http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm>.

[23] "Sobel Edge Detector." Feature Detectors. 5 Mar. 2013 <http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm>.

[24] Canny Edge Detection." CyroForge. 5 Mar. 2013 <http://cyroforge.wordpress.com/canny-edge-detection/>.

[25] Practical threaded programming with Python." Practical threaded programming with Python. 20 Feb. 2013 <http://www.ibm.com/developerworks/aix/library/au-threadingpython/>.

[26] Warren Weckesser. "Re: ComplexWarning when calling convolve2d() in SciPy, why?" Weblog comment. StackOverflow. <http://stackoverflow.com/questions/15992800/complexwarning-when-calling-convolve2d-in-scipy-why>.

[27]NIST Biometric Image Software." NBIS. 25 Dec. 2012 <http://www.nist.gov/itl/iad/ig/nbis.cfm>.

[28] Colin Myers. "White Box Testing (Code Inspection)." White Box Testing (Code Inspection). N.p., n.d. Web. 15 Apr. 2013.

[29] Fredrik Berg Kjolstad, and Marc Snir. Ghost Cell Pattern. Thesis. University of Illionois, 2010.

Chapter 15: Source Code

Table of Contents

1 Normalization System

1.1 NormalizationSystem.py

```
'''

NormalizationSystem.py

Description:

    This system was designed to compare three different Normalization
algorithms applied on the entire image

    and then in parallel on sub-images of size 16 X 16.


Usage:

    From command line:   python NormalizationSystem.py <filenename>


Output:

    -original image

    -wiki_im : image after wikipedia's normalization algorithm on the
entire image

    -meanStd_im : image after Mean and Standard Deviation algorithm on
the entire image

    -ipl_im : image after iplImage algorithm on the entire image

    -p_wiki_im : image after a parallel wikipedia algorithm

    -p_meanStd_im : image after a parallel Mean and Standard Deviation
algorithm

    -p_ipl_im : image after a parallel iplImage algorithm

    -Min and Max values for each resulting image (visible from command
line)


Usage:

    From command line:   python NormalizationSystem.py <filename>

@author: Edoardo Foco

'''
```

```
import numpy as np
import cv2
import ThreadController
import WikiNorm
import MeanStd_Norm
import IplNorm
import sys
from PIL import Image

def main():

    image = validateInput()

    # Wiki Normalization
    wiki_im = WikiNorm.normalise(image)

    # MeanStd Normalization
    meanStd_im = MeanStd_Norm.normalise(image)

    # IplImage Normalization
    ipl_im = IplNorm.normalise(image)

    # Parallalel Normalizations
    controller = ThreadController.ThreadController(image)
    controller.startThreading()
    p_wiki_im, p_meanStd_im, p_ipl_im = controller.reconstructImage()


showResults(image,wiki_im,meanStd_im,ipl_im,p_wiki_im,p_ipl_im,p_meanS
td_im)
    sys.exit()
```

```
def validateInput():
    if not len(sys.argv) == 2:
        print "\n**Error: Invalid arguments.\nUsage: python main.py
<filename>\n"
        sys.exit()


    image_str = sys.argv[1]
    if image_str == "-h":
        print "\nUsage:\npython NormalizationSystem.py <filename>\n"
        print "Supported File Formats: .png\n"
        print "The system is going to compute Wikipedia's, Mean and
Standard Deviation, and IplImage normalization algorithms\n"
        sys.exit()


    length = len(image_str) - 4
    file_extension = image_str[length:]


    if not file_extension == ".png":
        print "\n**Error: Invalid file type.\nThe software supports
only .png files\n"
        sys.exit()


    image = cv2.imread(image_str, cv2.CV_LOAD_IMAGE_GRAYSCALE)


    if image == None:
        print "\n**Error: File not found\n"
        sys.exit()
```

```
    if not image.shape[0] == 480:
        if not image.shape[1] == 640:
            print "\n**Error: Size of the image is not
supported.\nSupported size: 640x480\n"
            sys.exit()

    return image


def
showResults(image,wiki_im,meanStd_im,ipl_im,p_wiki_im,p_ipl_im,p_meanS
td_im):
    print "The Normalization applied on the whole image using
Wikipedia's algorithm returned an image who's: \nMin =
",np.min(wiki_im)," and Max = ",np.max(wiki_im)
    print "\nThe Normalization applied on the whole image using Means
and Standard Deviation algorithm returned an image who's: \nMin =
",np.min(meanStd_im)," and Max = ",np.max(meanStd_im)
    print "\nThe Normalization applied on the whole image using the
IplImage energy normalization algorithm returned an image who's: \nMin
= ",np.min(ipl_im)," and Max = ",np.max(ipl_im)
    print "\nThe Normalization applied locally on blocks of 16 x 16
using Wikipedia's algorithm returned an image who's: \nMin =
",np.min(p_wiki_im)," and Max = ",np.max(p_wiki_im)
    print "\nThe Normalization applied locally on blocks of 16 x16
using Means and Standard Deviation algorithm returned an image who's:
\nMin = ",np.min(p_meanStd_im)," and Max = ",np.max(p_meanStd_im)
    print "\nThe Normalization applied locally on blocks of 16 x 16
using the IplImage energy normalization algorithm returned an image
who's: \nMin = ",np.min(p_ipl_im)," and Max = ",np.max(p_ipl_im)


    cv2.imshow('original', image)
    cv2.imshow('wiki_normalization', wiki_im)
    cv2.imshow('meanStd_normalization', meanStd_im)
    cv2.imshow('ipl_normalization', ipl_im)
    cv2.imshow("p_wiki_normalization",p_wiki_im)
```

```
    cv2.imshow("p_ipl_normalization", p_ipl_im)
    cv2.imshow("p_meanStd_normalization", p_meanStd_im)

    cv2.waitKey()


if __name__=="__main__":
    main()
```

1.2 ThreadController.py

```
'''

ThreadController.py

Description:

    This class creates a thread controller which will takes as an
input an image fragmenting it, then

    spawnining a pool of threads to process it, and finally
reconstructing it.


Input:

    -Image

Output:

    - wiki_final_im : image after a parallel Wikipedia's algorithm

    - meanStd_final_im : image after a parallel Mean and Standard
Deviation algorithm

    - ipl_final_im: image after a parallel IplImage algorithm

@author: Edoardo Foco


'''


import numpy as np

import Queue

import BlockThread



class ThreadController:



    def __init__(self,image):

        self.image = image

        self.windows = {}       #dictionary of windows
```

```
        self.wiki_windows = {}  #dictionary of resulting windows
        self.meanStd_windows = {}
        self.ipl_windows = {}
        self.window_height = 16
        self.window_width = 16


    def startThreading(self):
        '''Segments the image in windows of size 16x16, creates a
queue
            of windows and processes it using a pool of threads
        '''

        # initialize queue
        queue = Queue.Queue()

        currentx = 0
        currenty = 0

        # divide image in windows
        count = 0
        for y in range (0,self.image.shape[0]/self.window_height):
            currentx = 0
            for x in range(0,self.image.shape[1]/self.window_width):

self.windows[count]=self.image[currenty:currenty+self.window_height,cu
rrentx:currentx+self.window_width]
                currentx += self.window_width
                count += 1


            currenty += self.window_height
```

```
        # spawn pool of threads passing the queue object, this allows
for threads to
        # recurrently analyze queue objects.
        for i in range(4): # change range to change number of threads
            t =
BlockThread.BlockThread(queue,self.wiki_windows,self.meanStd_windows,s
elf.ipl_windows)
            t.setDaemon(True)
            t.start()



        # insert windows in the queue
        new_windows = self.windows.items()
        for window in new_windows:
            queue.put(window)



        queue.join()



    def reconstructImage(self):
        ''' Re-constructs Processed Image '''


        wiki_stacked_rows =[]
        meanStd_stacked_rows = []
        ipl_stacked_rows = []
        counter=0


        # constructing rows by stacking the windows horizontally
        for y in range (0,self.image.shape[0]/self.window_height):
```

```
            wiki_row = []
            meanStd_row = []
            ipl_row = []
            for x in range (0,self.image.shape[1]/self.window_width):
                if x == 0:
                    wiki_row = self.wiki_windows[counter]
                    meanStd_row = self.meanStd_windows[counter]
                    ipl_row = self.ipl_windows[counter]
                    counter +=1
                else:

wiki_row=np.hstack((wiki_row,self.wiki_windows[counter]))

meanStd_row=np.hstack((meanStd_row,self.meanStd_windows[counter]))

ipl_row=np.hstack((ipl_row,self.ipl_windows[counter]))
                    counter += 1
                # if it is considering the last window of the row then
append to stacked_rows
                if x == self.image.shape[1]/self.window_width - 1:
                    wiki_stacked_rows.append(wiki_row)
                    meanStd_stacked_rows.append(meanStd_row)
                    ipl_stacked_rows.append(ipl_row)


        # constructing columns by stacking the rows vertically
        counter = 0
        for x in range (0,len(wiki_stacked_rows)):  # creating final
image
                if counter == 0:
                    wiki_final_im=wiki_stacked_rows[0]
                    meanStd_final_im = meanStd_stacked_rows[0]
                    ipl_final_im = ipl_stacked_rows[0]
```

```
                    counter+=1
                else:
wiki_final_im=np.vstack((wiki_final_im,wiki_stacked_rows[x]))
                    meanStd_final_im =
np.vstack((meanStd_final_im,meanStd_stacked_rows[x]))
                    ipl_final_im =
np.vstack((ipl_final_im,ipl_stacked_rows[x]))

        return wiki_final_im,meanStd_final_im,ipl_final_im
```

1.3 BlockThread.py

```
'''

Description:

    This is the creation of the actual thread which will process the
given sub-image applying

    first a Normalization algorithm then a orientation estimation
algorithm.

Input:

    - queue : Thread-safe data structure which controls the input

    - wiki_windows : Dictionary which controls the output

    - meanStd_windows : Dictionary which controls the output

    - ipl_windows : Dictionary which controls the output

@author: Edoardo Foco

'''


import threading

import WikiNorm

import MeanStd_Norm

import IplNorm


class BlockThread(threading.Thread):



    def __init__(self, queue,
wiki_windows,meanStd_windows,ipl_windows):

        threading.Thread.__init__(self)

        self.queue = queue

        self.wiki_windows = wiki_windows

        self.meanStd_windows = meanStd_windows

        self.ipl_windows = ipl_windows
```

```
    def run(self):
        while True:
            # grabs window from queue
            # Note: win is a tuple [key, value], using dictionary
data-structure
            win = self.queue.get()

            # Normalization
            self.wiki_windows[win[0]] = WikiNorm.normalise(win[1])
self.meanStd_windows[win[0]]=MeanStd_Norm.normalise(win[1])
            self.ipl_windows[win[0]] = IplNorm.normalise(win[1])
            self.queue.task_done()
```

1.4 WikiNorm.py

```
'''
WikiNorm.py
Description:
    Normalizing 0 - 255 initial fingerprint image to a 0 - 1 value
image using
    normalization formula suggested by Wikipedia.

    Input:
        -image

    Output:
        -norm_im
@author: Edoardo Foco
'''

import cv2
import numpy as np


def normalise(image):

    dbl_image = image.astype(float)

    # computing normalization
    currentMin = np.min(dbl_image)
    currentMax = np.max(dbl_image)
    currentRange = currentMax - currentMin
    newMin = 0
    newMax = 1
```

```
    newRange = newMax - newMin


    # calculating mean and standard deviation
    meanStd = cv2.meanStdDev(dbl_image)


    norm_im = dbl_image
    # regions with a low standard deviation are assumed to NOT be
regions of interest and
    # have values close to currentMax therefore their value is set to
the brightest possible -> 1
    if meanStd[1]>20:


        norm_im = (dbl_image - currentMin)*(newRange / currentRange)


    elif meanStd[1]<=20:


        norm_im = norm_im / currentMax


    return norm_im
```

1.5 MeanStd_Norm.py

```
'''
MeanStd_Norm.py
Description:
    Normalizing 0 - 255 initial fingerprint using Hong et al approach.
Input:
        -image
 Output:
        -norm_im

@author: Edoardo Foco
'''
import cv2
import numpy as np


def normalise(image):

    dbl_image = image.astype(float)
    norm_im = dbl_image

    # finding mean and standard deviation
    # Note: mean_stddev is a tuple where: mean = mean_stddev[0], std =
mean_stddev[1]
    meanStd = cv2.meanStdDev(dbl_image)
    required_mean = 0
    required_stddev = 1

    # mapping coordinates where pixel is more or less than mean
    x0,y0 = np.where(norm_im >= meanStd[0])
```

```
    x1,y1 = np.where(norm_im < meanStd[0])


    # computing normalization
    norm_im = dbl_image - meanStd[0]
    norm_im = norm_im**2
    norm_im = norm_im/meanStd[1]


    # separating foreground from background
    if meanStd[1] > 20:
        norm_im[x0,y0] = required_mean +
np.sqrt(required_stddev*norm_im[x0,y0])
        norm_im[x1,y1] = required_mean -
np.sqrt(required_stddev*norm_im[x1,y1])


    else:
        norm_im[x1,y1] = 1 # 1 is the maximum desired value
        norm_im[x0,y0] = 1


    return norm_im
```

1.6 IplNorm.py

```
'''
IplNorm.py
Description:
    Normalizing 0 - 255 initial fingerprint to a normalized image.
    Using energy normalization.

    Input:
        -image

    Output:
        -norm_im
@author: Edoardo Foco
'''

import cv2
import numpy as np

def normalise(image):

    dbl_image = image.astype(float)
    # calculate the mean of the image.
    mean = np.mean(dbl_image)

    # converting numpy 8-bit image to 8- bit cv2.iplimage
    iplImage = cv2.cv.CreateImageHeader((image.shape[1],
image.shape[0]), cv2.cv.IPL_DEPTH_8U, 1)
    cv2.cv.SetData(iplImage, image.tostring(), image.dtype.itemsize *
1 * image.shape[1])
```

```
    # initializing 32-bit floating point iplimage
    image_32F = cv2.cv.CreateImage(cv2.cv.GetSize(iplImage),
cv2.cv.IPL_DEPTH_32F,1)

    # converting 8-bit unsigned integer image to 32-bit floating point
image
    cv2.cv.CvtScale(iplImage,image_32F)

    # energy Normalization. Formula: image = image/mean(image)
    cv2.cv.ConvertScale(image_32F, image_32F, (1/mean), 0);

    # re-converting to numpy image
    norm_im = np.asarray(image_32F[:,:])

    return norm_im
```

2 Sobel Edge Detection System

2.1 SobelEdgeDetection.py

```
'''
SobelEdgeDetection.py
Description:
    The software will apply Sobel's Edge detection algorithm to a
given image.
    The algorithm includes:
        - Gaussian Blurring
        - Sobel Filter
        - Thresholding

Input:
    - gS <gaussSigma> (float)
    - gW <gaussWindow> (int)
    - T  <threshold>   (int)

Output:
    - blurred_im (np.array)
    - grad_x (np.array)
    - grad_y (np.array)
    - edge_im (np.array)

Usage:
    From command line:   python SobelEdgeDetection.py <image>

@author: Edoardo Foco
'''
```

```
import numpy as np
import cv2
import sys
import Filters

def main():
        image = validateInput()

    t = True
    while(t):
        try:
            sigma = float(raw_input("\nInsert Gauss Sigma:\n"))
            t = False
        except ValueError:
            print "\n**Error: Parameter.\n Specified sigma is not a
float or an int\n"

    t = True
    while(t):
        try:
            win_size = int(raw_input("\nInsert Window size:\n"))
            t = False
        except ValueError:
            print "\n**Error: Parameter.\nSpecified window size is not
an int\n"

        if not win_size == 3:
            if not win_size == 5:
                if not win_size == 7:
                    print "\n**Error: Parameter Error.\nThe dimensions
of the gaussian window are not supported.\nSupported values for the
windows are: 3, 5, 7\n"
```

```
                    t = True
    t = True
    while(t):
        try:
            threshold = int(raw_input("\nInsert Threshold Value:\n"))
            t = False
        except ValueError:
            print "\n**Error: Parameter.\nSpecified threshold is not
an int\n"

    # compute filters
    blurred_im, grad_x, grad_y = applyFilters(image, sigma, win_size)

    # calculate gradient
    grad = np.hypot(grad_y,grad_x)

    # binarise image
    edge_im = binarise(grad,threshold)

    # show results
    showResults(image, blurred_im,grad_x,grad_y,edge_im)

    sys.exit()

def validateInput():
    if not len(sys.argv) == 2:
        print "\n**Error: Invalid arguments.\nUsage: \npython
SobelEdgeDetection.py <filename>\n"
        sys.exit()

    # getting input
```

```
    image_str = sys.argv[1]


    if image_str == "-h":
        print "\nUsage:\npython SobelEdgeDetection.py <filename>\n"
        print "Supported File Formats: .png\n"
        print "The system is going to compute the Sobel Edge Detection
algorithm on the image\n"
        sys.exit()


    length = len(image_str) - 4
    file_extension = image_str[length:]


    if not file_extension == ".png":
        print "\n**Error: Invalid file type.\nThe software supports
only .png files"
        sys.exit()


    image = cv2.imread(image_str, cv2.CV_LOAD_IMAGE_GRAYSCALE)
    if image == None:
        print "\n**Error: File not found\n"
        sys.exit()


    if not image.shape[0] == 480:
        if not image.shape[1] == 640:
            print "\n**Error: Size of the image is not
supported.\nSupported size: 640x480\n"
            sys.exit()


    return image


def applyFilters(image, sigma, win_size):
```

```
    '''
    Apply Gaussian Blur and Sobel filter
    '''
    blurred_im = Filters.gaussFilter(image, sigma, win_size)
    grad_x,grad_y = Filters.sobelFilter(blurred_im)
    return blurred_im, grad_x, grad_y


def binarise(grad,threshold):
    x0,y0 = np.where(grad > threshold)
    x1,y1 = np.where(grad < threshold)
    grad[x0,y0] = 1
    grad[x1,y1] = 0
    edge_im = grad

    return edge_im

def showResults(image, blurred_im, grad_x, grad_y, edge_im):
    cv2.imshow('original_im', image)
    blurred_im = blurred_im.astype(np.uint8)
    cv2.imshow('blurred_im',blurred_im)
    cv2.imshow('grad_x',grad_x)
    cv2.imshow('grad_y',grad_y)
    cv2.imshow('edge_im',edge_im)
    cv2.waitKey()



if __name__ == '__main__':
    main()
```

2.2 Filters.py

```
'''
Filters.py
Description:
    This class is made to hold three filters:
        -Gaussian Blurring Filter
        -Sobel Filter

@author: Edoardo Foco
'''

import numpy as np
from scipy.signal import convolve2d as conv


def gaussFilter(image,sigma,window):
    '''
    Description:
        This function will execute a Gaussian Blurring Filter on the
given image.

    Input:
        -image (np.array)
        -sigma (int)
        -window size (int)

    Output:
        - blurred_im (np.array)
    '''
    # creating an empty kernel
```

```
    kernel = np.zeros((window,window))
    # centre of the kernel
    c0 = window // 2

    # computing kernel using Gaussian function
    for x in range(window):
            for y in range(window):
                # calculating magnitude of the centre pixel. x^2 + y^2
                r = np.hypot((x-c0),(y-c0))
                # computes gaussian filter
                val = (1.0 / 2 * np.math.pi * sigma) * np.math.exp(-
(r * r) / ( 2 * sigma * sigma ))
                kernel[x,y] = val

    kernel = kernel / kernel.sum()
    print kernel
    # executes convolution
    blurred_im = conv(image,kernel)[1:-1,1:-1]

    return blurred_im



def sobelFilter(image):
    '''
    Description:
    This function will execute a Sobel Filter on the given image to
find its horizontal and vertical gradients.
    The default filter is set to 3x3.
    Input:
        -image (np.array)
```

```
    Output:
        - grad_x (np.array)
        - grad_y (np.array)
    '''

    # creating sobel kernels
    fx = np.array([1, 2, 1,
                   0, 0, 0,
                   -1,-2,-1])
    fy = np.array([-1,0,1,
                   -2,0,2,
                   -1,0,1])

    fx = fx.reshape(3,3)
    fy = fy.reshape(3,3)

    # convolving kernels
    grad_x = conv(image,fx)
    grad_y = conv(image,fy)

    return grad_x,grad_y
```

3 Fingerprint Enhancement System

3.1 FingerprintEnhancmentSystem

```
'''
FingerprintEnhancementSystem.py
Description:
    The software processes the image in parallel. It currently
supports .png images
    of size 640x480. It will apply a Normalization algorithm and a
Ridge Orientation
    algorithm to each 16x16 block of the fingerprint image.

    This system is not complete and is missing:
        - Frequency Estimation Algorithm
        - Gabor Filter
        - Binarization
        - Thinning

Usage:
    From command line:   python FingerprintEnhancementSystem.py
<filename>

@author: Edoardo Foco
'''

import sys
import cv2
import ThreadController
from PIL import Image
```

```
import time


def main():

    image = validateInput()

    # process image
    start_time = time.time()
    controller = ThreadController.ThreadController(image)
    controller.start_threading()
    # orient_im is the orientation image neede for the Gabor Filter.
It is not used because the software is not complete
    orient_im, processed_image = controller.reconstruct_image()
    elapsed_time = time.time() - start_time
    print elapsed_time

    cv2.imshow('original image',image)
    Image.fromarray(processed_image).show()
    cv2.imshow('proc_image', processed_image)
    cv2.waitKey()

    sys.exit()

def validateInput():
    if len(sys.argv) > 2:
        print "**Error: Invalid arguments.\nUsage: python main.py
<filename>"
        sys.exit()

    image_str = sys.argv[1]
```

```
    if image_str == "-h":
        print "\nUsage:\npython FingerprintEnhacementSystem.py
<filename>\n"
        print "Supported File Formats: .png\n"
        print "The system is going to compute an Enhancement algorithm
on the image\n"
        sys.exit()



    length = len(image_str) - 4
    file_extension = image_str[length:]


    if not file_extension == ".png":
        print "**Error: Invalid file type.\nThe software supports only
.png files"
        sys.exit()



    image = cv2.imread(image_str, cv2.CV_LOAD_IMAGE_GRAYSCALE)
    if image == None:
        print "**Error: File not found"
        sys.exit()


    if not image.shape[0] == 480:
        if not image.shape[1] == 640:
            print "\n**Error: Size of the image is not
supported.\nSupported size: 640x480\n"
            sys.exit()


    return image
    if __name__=="__main__":
    main()
```

3.2 ThreadController.py

```
'''

ThreadController.py

Description:

    This class creates a thread controller which will takes as an
input an image fragmenting it, then

    spawnining a pool of threads to process it, and finally
reconstructing it.


Input:

    -Image

Output:

    -orient_im : The Orientation Image needed for the Gabor filter

    -final_im : An normalized image representing the orientation
through the use of lines


@author: Edoardo Foco

'''


import numpy as np

import Queue

import BlockThread


class ThreadController:



    def __init__(self,image):

        self.image = image

        self.windows = {}        #dictionary of windows controls input

        self.n_windows = {}      #dictionary of resulting windows
controls output
```

```
        self.o_windows = {}      #dictionary of resulting oriented
windows controls output

        self.window_height = 16

        self.window_width = 16



    def start_threading(self):

        '''Segments the image in windows of size 16x16, saves the
windows in a queue then sends

            each element to a thread which processes it

        '''


        # initialize queue

        queue = Queue.Queue()


        currentx = 0

        currenty = 0


        # create windows

        count = 0

        for y in range (0,self.image.shape[0]/self.window_height):

            currentx = 0

            for x in range(0,self.image.shape[1]/self.window_width):

self.windows[count]=self.image[currenty:currenty+self.window_height,cu
rrentx:currentx+self.window_width]

                currentx += self.window_width

                count += 1


            currenty += self.window_height
```

```
        # spawn pool of threads passing the queue object, this allows
for threads to
        # recurrently analyze queue objects.
        for i in range(2): # change range to change number of threads
            t =
BlockThread.BlockThread(queue,self.n_windows,self.o_windows)
            t.setDaemon(True)
            t.start()

        # insert windows in the queue
        new_windows = self.windows.items()
        for window in new_windows:
            queue.put(window)



        queue.join()



    def reconstruct_image(self):
        ''' Re-constructs Processed Images '''

        stacked_rows =[]
        o_stacked_rows = []

        counter=0

        # constructing rows by stacking the windows horizontally
        for y in range (0,self.image.shape[0]/self.window_height):
            row = []
            o_row = []
            for x in range (0,self.image.shape[1]/self.window_width):
```

```
                if x == 0:
                    row = self.n_windows[counter]
                    o_row = self.o_windows[counter]
                    counter +=1
                else:
                    row=np.hstack((row,self.n_windows[counter]))
                    o_row = np.hstack((o_row,
self.o_windows[counter]))
                    counter += 1

                if x == self.image.shape[1]/self.window_width - 1:
                    stacked_rows.append(row)
                    o_stacked_rows.append(o_row)

        # constructing columns by stacking the rows vertically
        counter = 0
        for x in range (len(stacked_rows)):
                if counter == 0:
                    final_im=stacked_rows[0] # creating final image
                    orient_im = o_stacked_rows[0]
                    counter+=1
                else:
                    final_im=np.vstack((final_im,stacked_rows[x]))
                    orient_im =
np.vstack((orient_im,o_stacked_rows[x]))

        return orient_im, final_im
```

3.3 BlockThread.py

```
'''

Description:

    This is the creation of the actual thread which will process the
given sub-image applying

    first a Normalization algorithm then a orientation estimation
algorithm.

Input:

    - queue : Thread-safe data structure which controls the input

    - n_windows : Dictionary which controls the output

    - o_windows : Dictionary which controls the output


@author: Edoardo Foco
'''


import threading
import MeanStd_Norm
import Orientation



class BlockThread(threading.Thread):



    def __init__(self, queue, n_windows, o_windows):
        threading.Thread.__init__(self)
        self.queue = queue
        self.n_windows = n_windows
        self.o_windows = o_windows


    def run(self):
```

```
        while True:

            # print threading.currentThread().getName()

            # grabs window from queue
            # Note: win is a tuple [key, value], using dictionary
data-structure
            win = self.queue.get()

            # Normalization
            self.n_windows[win[0]] = MeanStd_Norm.normalise(win[1])

            # Orientation
            self.o_windows[win[0]],self.n_windows[win[0]] =
Orientation.Orientation(self.n_windows[win[0]]).calculateOrientation()

            # signals queue job is done
            self.queue.task_done()
```

3.4 MeanStd_Norm.py

```
'''
MeanStd_Norm.py
Description:
    Normalizing 0 - 255 initial fingerprint using Hong et al approach.

 Input:
    -image
 Output:
    -norm_im
@author: Edoardo Foco
'''
import cv2
import numpy as np


def normalise(image):

    dbl_image = image.astype(float)
    norm_im = dbl_image

    # finding mean and standard deviation
    # Note: mean_stddev is a tuple where: mean = mean_stddev[0], std =
mean_stddev[1]
    mean_stddev = cv2.meanStdDev(dbl_image)
    required_mean = 0
    required_stddev = 1

    # mapping coordinates where pixel is more or less than mean
    x0,y0 = np.where(norm_im >= mean_stddev[0])
```

```
    x1,y1 = np.where(norm_im < mean_stddev[0])


    # computing normalization
    norm_im = dbl_image - mean_stddev[0]
    norm_im = norm_im ** 2
    norm_im = norm_im/mean_stddev[1]


    # separating foreground form background
    if mean_stddev[1] > 5:
        norm_im[x0,y0] = required_mean +
np.sqrt((required_stddev*norm_im[x0,y0]))
        norm_im[x1,y1] =  required_mean -
np.sqrt((required_stddev*norm_im[x1,y1]))


    else:
        norm_im[x1,y1] = 1 # 1 is the maximum desired value
        norm_im[x0,y0] = 1


    return norm_im
```

3.5 Orientation.py

```
'''

Orientation.py

Description:

    This function computes the orientation angles.

    The angles are then used to determine the angle of the line that
passes in each 8x8 window in the image.


Input:

    image - 16x16 image


Output:

    orient_im - orientation image

    final_im - image representing the orientation angles


Note:

    The author is not happy with the output of the drawing function.
Although the angles

    at each pixel are calculated correctly, when it comes to drawing
the lines the algorithm

    approximates an angle for each 8x8 window by making an average of
the

    angles computed at each pixel in the window. This results in
drawing inaccurate lines. It is to be

    noted that this does not affect the enhancement algorithm because
the lines will not be considered (they are

    drawn just for resemblance)



@author: Edoardo Foco

'''
```

```
import cv2
import numpy as np
from scipy import ndimage
import math
import Filters
from PIL import Image

class Orientation():

    def __init__(self,image):
        self.image = image



    def calculateOrientation(self):

        self.image = self.image

        # smooth image with gaussian blur
        smoothed_im = Filters.gaussFilter(self.image, 0.5, 3)

        # calculate gradients with sobel filter
        dx,dy = Filters.sobelFilter(smoothed_im)

        # smooth gradients
        Gx = Filters.gaussFilter(dx,0.5,3)
        Gy = Filters.gaussFilter(dy,0.5,3)

        # compute gradient magnitude
        Gxx = Gx **2
```

```
        Gyy = Gy **2
        G = np.sqrt(Gxx + Gyy)

        # calculate theta
        theta = np.arctan2(Gy,Gx)

        # smooth theta
        smoothed_theta = Filters.gaussFilter(theta, 0.5, 3)

        # calculate double sine and cosine on theta --> increases
precision
        Tx = (G**2 + 0.001) * (np.cos(smoothed_theta)**2 -
np.sin(smoothed_theta)**2)
        Ty = (G**2 + 0.001) * (2 * np.sin(smoothed_theta) *
np.cos(smoothed_theta))

        denom = np.sqrt(Ty**2 + Tx**2)

        Tx = Tx / denom
        Ty = Ty / denom

        # smooth theta x and y
        smoothed_Tx = Filters.gaussFilter(Tx, 0.5, 3)
        smoothed_Ty = Filters.gaussFilter(Ty, 0.5, 3)

        # calculate new value for theta
        theta = np.pi + np.arctan2(smoothed_Ty,smoothed_Tx)/2

        #draw lines
        final_im = self.decomposeImage(theta)
        return theta, final_im
```

```
    def decomposeImage(self,theta):
        # The following code will calculate the average angle and draw
lines --> it should be replaced
        # the reason is that this approximation is very inaccurate.
        # Note: this piece of code will not impact the precision of
the Gabor filter. It is used
        # just to draw the lines on the orientation image.
        windows = {}
        orient = {}
        median = {}
        row_count = 0
        window_height = 8
        window_width = 8
        currentx= 0
        currenty = 0

        # decomposing the window in 8 x 8 windows
        for y in range (0,2):
                currentx = 0
                for x in range(0,2):

windows[row_count]=self.image[currenty:currenty+window_height,currentx
:currentx+window_width]

orient[row_count]=theta[currenty:currenty+window_height,currentx:curre
ntx+window_width]

                    currentx += window_width
                    row_count += 1

                currenty += window_height
```

```
# drawing lines on each 8 x 8 window
for wins in range (0,4):
    sum = 0
    for i in range (0,orient[wins].shape[0]):
        for j in range(0,orient[wins].shape[1]):
            sum += orient[wins][i][j]

    len = 1
    if not orient[wins].shape[0] == 0:
        median[wins] = sum/orient[wins].shape[0]**2
        x1 = 4 - len/2*math.cos(median[wins]) * 10
        x1 = np.around(x1)
        x1 = x1.astype(int)
        y1 = 4 - len/2*math.sin(median[wins]) * 10
        y1 = np.around(y1)
        y1 = y1.astype(int)

        x2 = 4 + math.cos(median[wins]) * 10
        x2 = np.around(x2)
        x2 = x2.astype(int)
        y2 = 4 + math.sin(median[wins]) * 10
        y2 = np.around(y2)
        y2 = y2.astype(int)

        point1 = (x1,y1)
        point2 = (x2,y2)

        x0,y0 = np.where(windows[wins]<=0.5) # values below or
equal to 0 are darker regions, hence ridges
```

```
                if np.any(x0):
                    cv2.line(windows[wins], point1, point2,
cv2.cv.CV_RGB(255, 255, 255))


        final_image = self.reconstructImage(windows)
        return final_image


    def reconstructImage(self,windows):
        stacked_rows =[]
        counter=0

        orient_im = self.image

        #print n_windows
        for y in range (0,2):  # creating array of rows
            row = []
            for x in range (0,2):
                if x == 0:
                    row = windows[counter]
                    counter +=1
                else:
                    row=np.hstack((row,windows[counter]))
                    counter += 1

                if x == 15:
                    stacked_rows.append(row)
        #print stacked_rows
```

```
counter = 0
for x in stacked_rows:  # creating final image
        if counter == 0:
            orient_im=stacked_rows[0]
            counter+=1
        else:
            orient_im=np.vstack((orient_im,x))

return orient_im
```

3.6 Filters.py

```
'''
Filters.py
Description:
    This class is made to hold three filters:
        -Gaussian Blurring Filter
        -Sobel Filter
        -Gabor Filter
Note:
    The Gabor Filter has not been developed yet
'''


import numpy as np
from scipy.signal import convolve2d as conv



def gaussFilter(image,sigma,window):
    '''
    Description:
        This function will execute a Gaussian Blurring Filter on the
given image. The sigma and window size
        are set to default sigma = 2 and window = 5x5.


    Input:
        -image (np.array)
        -sigma (int)
        -window size (int)


    Output:
        - blurred_im (np.array)
```

```
    '''
    # creating an empty kernel
    kernel = np.zeros((window,window))
    # centre of the kernel
    c0 = window // 2

    # computing kernel using Gaussian function
    for x in range(window):
            for y in range(window):
                r = np.hypot((x-c0),(y-c0)) #calculating magnitude of
the centre pixel. x^2 + y^2
                val = (1.0/2*np.math.pi*sigma)*np.math.exp(-
(r*r)/(2*sigma*sigma)) # computes gaussian filter
                kernel[x,y] = val

    kernel = kernel / kernel.sum()

    # executes convolution
    blurred_im = conv(image,kernel)[1:-1,1:-1]

    return blurred_im



def sobelFilter(image):
    '''
    Description:
    This function will execute a Sobel Filter on the given image to
find its horizontal and vertical gradients.
    The default filter is set to 3x3.
    Input:
```

```
    -image (np.array)

Output:
    - grad_x (np.array)
    - grad_y (np.array)
'''

# creating sobel kernels
fx = np.array([1, 2, 1,
               0, 0, 0,
               -1,-2,-1])
fy = np.array([-1,0,1,
               -2,0,2,
               -1,0,1])

fx = fx.reshape(3,3)
fy = fy.reshape(3,3)

grad_x = conv(image,fx)
grad_y = conv(image,fy)

return grad_x,grad_y
```

Matlab

1 HistogramEqualization.m

```
clc
clear
close all
path ='C:\*.png';

[name,p]=uigetfile(path); %select path
file = [p, name];
I = imread(file);

figure,imhist(I),title('Historgram of original image');
qual_img = histeq(I);

figure, imshow(qual_img),title('Equalized Image');
figure,imhist(qual_img),title('Equalized histogram');
```

2 CannyEdgeDetection.m

```
clc
clear
close all

path ='C:\*.png';
[name,p]=uigetfile(path); %select path

file = [p, name];
I = imread(file);
```

```
BW = edge(I,'canny',[0.2,0.4],1);
figure, imshow(BW),title('edge_im Matlab');
```