



Department of Information Science and Technology

Hand-based Biometric Recognition System for Mobile Devices

Luís Miguel dos Santos Serrano

Dissertation presented in partial fulfillment of the requirements for the degree of
Master of Science in Telecommunications and Informatics Engineering

Supervisor:

Prof. Doctor Luís Eduardo de Pinho Ducla Soares, Assistant Professor,
ISCTE-IUL

Co-supervisor:

Prof. Doctor Paulo Luís Serras Lobato Correia, Assistant Professor,
Instituto Superior Técnico

June, 2011



Department of Information Science and Technology

Hand-based Biometric Recognition System for Mobile Devices

Luís Miguel dos Santos Serrano

Dissertation presented in partial fulfillment of the requirements for the degree of
Master of Science in Telecommunications and Informatics Engineering

Supervisor:

Prof. Doctor Luís Eduardo de Pinho Ducla Soares, Assistant Professor,
ISCTE-IUL

Co-supervisor:

Prof. Doctor Paulo Luís Serras Lobato Correia, Assistant Professor,
Instituto Superior Técnico

June, 2011

«You cannot open a book without learning something.»

Confucius

Acknowledgements

I want to thank my supervisor, Prof. Dr. Luís Ducla Soares from ISCTE-IUL Lisbon University Institute, and my co-supervisor, Prof. Dr. Paulo Lobato Correia from Instituto Superior Técnico, for all their support, trust, encouragement and advice. Without their help and availability, this work would not have been possible.

My thanks to all the colleagues in the Image Group from Instituto Superior Técnico, for their time and friendship. In special, to Maurício Ramalho and Sanchit Singh for their valuable help and patience, and for the good times that we spent. They made the knowledge exchange not only faster, but more immersive, interesting and interactive.

I also want to thank my parents, my two sisters and all my family and friends for all their support and cares, which have accompanied me throughout the development of this work.

Finally, I would like to thank Instituto de Telecomunicações, for the financial support that paid for the mobile device used in this dissertation.

Abstract

The usage of mobile devices such as smartphones has become a daily habit and need, as a way of managing and accessing personal data on the go. This created the emergent need to secure that data, which may contain not only contacts and messages, but also online accounts, files, bank account information and other applications or services which should be protected in case of device loss or theft. Biometrics have been an increasingly adopted solution, allowing individual recognition based on biometric traits (e.g., palmprint, voice, face, iris, signature), in a more practical and secure way, since biometrics cannot be forgotten or lost and are intrinsically associated with each individual. In this dissertation, a secure hand-based biometric recognition system is proposed and implemented for the Android 2.1 mobile platform. The system makes use of the camera present in Android devices (typically with a resolution of 5 Megapixel or higher) and performs a pre-segmentation of the captured images in the device to help the user optimize the data acquisition result. The data is then transferred over a secure connection to a server which performs further image analysis and feature extraction on the palmprint, using the Orthogonal Line Ordinal Features (OLOF) technique. A system-specific template composed of the extracted features is then securely stored in a database, with the help of a cryptographic hash function and an error correcting code (ECC).

Key-words

Android, binary templates, cryptographic hash function, error correcting code (ECC), hand-based, low-density parity-check (LDPC), mobile devices, Orthogonal Line Ordinal Features (OLOF), palmprint, secure biometric recognition system.

Resumo

A utilização de dispositivos móveis como *smartphones* tornou-se um hábito diário e uma necessidade, como forma de gerir e aceder a informação pessoal a qualquer momento. Isto criou uma necessidade emergente de tornar essa informação segura, que pode conter não apenas contactos e mensagens, mas também contas *online*, ficheiros, informação de contas bancárias, e outras aplicações ou serviços que devem estar protegidos em caso de perda ou roubo do dispositivo. As biométricas têm sido uma solução cada vez mais adoptada, permitindo reconhecimento individual baseado em características biométricas (e.g. palma da mão, voz, cara, íris, assinatura), de forma mais prática e segura, já que as biométricas não podem ser esquecidas ou perdidas e estão intrinsecamente associadas a cada indivíduo. Nesta dissertação é proposto e implementado um sistema seguro de reconhecimento biométrico baseado na mão para a plataforma móvel Android 2.1. O sistema faz uso da câmara existente em dispositivos Android (tipicamente com uma resolução de 5 Megapixel ou superior) e faz uma pre-segmentação das imagens capturadas no dispositivo para ajudar o utilizador a otimizar a aquisição de dados. Os dados são então transferidos numa ligação segura para um servidor que continua a análise de imagem e faz a extracção de *features* da palma da mão, utilizando a técnica *Orthogonal Line Ordinal Features* (OLOF). Uma *template* específica do sistema e composta das *features* extraídas é então guardada na base de dados de forma segura, com a ajuda de uma função de *hash* criptográfica e um código corrector de erros (ECC).

Palavras-chave

Android, *templates* binários, função de *hash* criptográfica, código corrector de erros (ECC), *hand-based*, código *low-density parity-check* (LDPC), dispositivos móveis, *Orthogonal Line Ordinal Features* (OLOF), palma da mão, sistema de reconhecimento biométrico seguro

Table of Contents

1. Introduction.....	1
2. Biometrics	5
2.1. Overview	5
2.2. Performance Measures	8
2.2.1. Accuracy	9
2.2.2. Time	11
2.2.3. Resources	12
2.3. Biometric Systems	12
2.4. Existing Mobile Phone Biometric Recognition Systems.....	17
3. Proposed Secure Biometric System.....	19
3.1. Target Platform, Device and Tools	19
3.1.1. Platform	19
3.1.2. Device	25
3.1.3. Development Tools.....	27
3.1.4. Libraries and Possibilities	28
3.2. System Architecture	32
3.2.1. Introduction.....	32
3.2.2. Application Scenarios	37
3.3. Implementation Details	39
3.3.1. Data acquisition and Pre-segmentation.....	41
3.3.2. Communication Interfaces and Protocol.....	45
3.3.3. Pre-processing.....	47
3.3.4. Feature Extraction.....	49
3.3.5. Error Correcting Code (ECC).....	51
3.3.6. Hash Function	54
4. Results.....	57

Hand-Based Biometric Recognition System for Mobile Devices

4.1.	Test Conditions	57
4.2.	Recognition Performance.....	59
4.3.	Application Performance	64
4.4.	Tradeoff.....	68
5.	Developed Android Application.....	73
5.1.	Introduction to the Android OS.....	73
5.2.	Developed Software Structure	76
5.2.1.	Android Application Optimization	78
6.	Conclusions and Future Work	79
7.	Bibliography	83

List of Figures

Figure 1 – Android example lock pattern.	2
Figure 2 – Types of biometrics. (Adapted individual images from the Web).	7
Figure 3 – Illustrative FAR and FRR graph.	9
Figure 4 – Interpretative graph of FAR, FRR and Equal Error Rate (EER).	10
Figure 5 – ROC curve, relating FAR and FRR at different threshold values [4].	11
Figure 6 – Main modules of a biometric recognition system.	12
Figure 7 – Attack points in a biometric system.	16
Figure 8 – Graph of the average time for platform mastery [30].	21
Figure 9 – Graph of the average debug time required for each platform [30].	21
Figure 10 – Android versions distribution graph as of 1 November 2010 [31].	23
Figure 11 – Android versions distribution graph as of 2 May 2011 [31].	24
Figure 12 – Android versions historical distribution graph from November 2011 to 2 nd May 2011 [31].	24
Figure 13 – HTC Desire device. Front side on the left, and back side on the right of the figure.	26
Figure 14 – System architecture and programming languages overview.	31
Figure 15 – Proposed system architecture (simplified).	32
Figure 16 – Secure biometric system architecture.	34
Figure 17 – Top level system architecture overview.	35
Figure 18 – Example internet access methods.	36
Figure 19 – Combined system architecture.	36
Figure 20 – NFC usage scenarios.	38
Figure 21 – Human palm.	40
Figure 22 – Palmprint features [42].	40
Figure 23 – Developed system's palmprint capture and pre-segmentation screen.	41
Figure 24 – Communication interfaces and type.	45
Figure 25 – Request message structure.	46
Figure 26 – Hand contour and reference points. Image was taken from [34].	47
Figure 27 – Image preprocessing phases. (Images adapted from [4]).	48
Figure 28 – Identification and normalization of the ROI [4].	49
Figure 29 – Block diagram of a typical ECC scenario.	51
Figure 30 – Block diagram of ECC contextualization in biometrics scenario.	52

Figure 31 – Example H matrix of a LDPC code [4].	53
Figure 32 – Interpretation of non-sparse matrix H [4] (Adapted).	53
Figure 33 – Behavior of 7 LDPC codes when correcting 8128-bit messages with bit error rates of around 27% [4].	54
Figure 34 – Genuine and Impostor Distributions, and ROC curves for the three tested databases, using 128×128 templates and the OLOF technique, with no shifts considered during template matching.	60
Figure 35 – Genuine and Impostor Distributions, and ROC curves for the three tested databases, using 128×128 templates and the OLOF technique, considering the original template position and 4 shifts during template matching.	62
Figure 36 – Genuine and Impostor Distributions for the HTC database, using 128×128 templates with the OLOF technique, and considering downsampling factors of 1:2, 1:3, 1:4 and 1:5, without considering template shifts at matching stage.	69
Figure 37 – Android software stack [31].	74
Figure 38 – Activity lifecycle [31].	75
Figure 39 – Developed Android application use case UML diagram.	76
Figure 40 – Android application Graphical User Interface (GUI) screens.	77

List of Tables

Table 1 – Comparison of the human and technical factors of seven popular biometric modalities (Adapted from [9]).	7
Table 2 – Message Digest algorithm 5 (MD5) hash function example.	13
Table 3 – Android versions distribution table as of 1 November 2010 [31].	23
Table 4 – Android versions distribution table as of 2 May 2011 [31].	23
Table 5 – HTC Desire device specifications summary.	26
Table 6 – Development tools and versions.	28
Table 7 – Message types and codes.	46
Table 8 – Possible error IDs and their meanings.	46
Table 9 – OLOF filter parameters.	50
Table 10 – Databases' specifications and consequent binary templates and comparisons.	58
Table 11 – Intra- and Inter-class comparisons interpretation.	58
Table 12 – Decidability indexes for the three tested databases, considering no shifts and considering 4 shifts in the template matching.	63
Table 13 – Application memory usage in starting activity. The units are expressed in 1024 bytes. N/A: Does not apply.	64
Table 14 – Application memory usage in image acquisition activity. The units are expressed in 1024 bytes. N/A: Does not apply.	65
Table 15 – CPU and memory usages for each of the considered downsampling ratios at data acquisition stage, when the hand image is captured and sent to the server. The units are expressed in 1024 bytes.	66
Table 16 – Processing times for different stages of the verification process, for the different considered downsampling ratios. The times are expressed in milliseconds.	67
Table 17 – Decidability indexes and considered image sizes for the various downsampling ratios tested for the HTC database, considering no shifts in the template matching.	70
Table 18 – Tradeoff between biometric recognition and computational performances. 1 KB = 1024 bytes.	71

List of Acronyms

- AAPT – Android Asset Packaging Tool
- ADT – Android Development Tools
- API – Application Programming Interface
- ARE – Asymptotic Relative Efficiency
- AVD – Android Virtual Device
- CAGR – Compound Annual Growth Rate
- CER – Crossover Error Rate
- CPU – Central Processing Unit
- CTS – Compatibility Test Suite
- DDMS – Dalvik Debug Monitor Server
- DNA – Deoxyribonucleic acid
- DOM – Document Object Model
- DVM – Dalvik Virtual Machine
- ECC – Error Correcting Code
- EER – Equal Error Rate
- FAR – False Accept Rate
- FEC – Forward Error Correction
- FRR – False Reject Rate
- FVB – Feature Vector Binarization
- GPS – Global Positioning System
- GSM – Global System for Mobile Communications
- GUI – Graphical User Interface
- I/O – Input/Output
- ID – Identifier
- IDE – Integrated Development Environment
- ISO – International Standards Organization
- JAI – Java Advanced Imaging
- JDK – Java Development Kit

JJIL – Jon’s Java Imaging Library

JNI – Java Native Interface

JVM – Java Virtual Machine

Java ME – Java Micro Edition

Java SE – Java Standard Edition

LAN – Local Area Network

LDA – Linear Discriminant Analysis

LDPC – Low-Density Parity-Check

LED – Light-Emitting Diode

MCC – MATLAB Code Compiler

MD – Message Digest

NDK – Native Development Kit

NFC – Near Field Communications

OLOF – Orthogonal Line Ordinal Features

OS – Operating System

OpenCV – Open Computer Vision Library

PCA – Principal Component Analysis

PIN – Personal Identification Number

RAM – Random-Access Memory

RFID – Radio-Frequency Identification

RIM – Research in Motion

ROC – Receiver Operating Characteristic

ROI – Region of Interest

ROM – Read-Only Memory

SDK – Standard Development Kit

SHA – Secure Hash Algorithm

SMS – Short Message Service

SSL – Secure Socket Layer

UML – Unified Modeling Language

Hand-Based Biometric Recognition System for Mobile Devices

URL – Uniform Resource Locator

XML – Extensible Markup Language

XOR – Bitwise Exclusive Disjunction

1. Introduction

In the modern world of 2011, with the increase in the density and complexity of information networks, users need to deal with increasing numbers of accounts, logins and other private data to interact with new products and services. With the evolution of the mobile phones industry, these services have become reachable on the go, anywhere, anytime, and with increasing demand, powdered by the increasing pace at which people must access and deal with information in today's society.

According to [1] it is estimated that the market of mobile phones will continue to increase in the years to come, notably from 2011 to 2015, in both emerging and developed markets around the world, sometimes with even higher penetration rates for middle income countries. This increase will lead to a continuous boost in the amount of available services and access to information and reveal emergent security and privacy needs. Issues such as private data protection, online banking, access management, identity authentication, and combinations of these, need to be addressed efficiently in order to protect the user from the higher risk of theft and loss inherent to accessing such application features through smaller devices such as mobile phones or tablet PCs.

Additionally, with the escalating computational power and inherent security threats, the need for more efficient and secure authentication methods is necessary, and noticeable in market studies such as "Global Biometric Forecast to 2012" [2] which states the global biometrics market is anticipated to grow at a Compound Annual Growth Rate (CAGR) of around 22% between 2011 and 2013.

Recent Nielsen studies in [3] dating from 3 March 2011, state that Android bears a 29% market share in U.S., being ahead of Research in Motion (RIM) Blackberry (27%) and Apple iOS (27%). The tendency is for this increase to continue, since the same study refers that youth are giving preference to the Android OS over other mobile platforms.

Typically, mobile phones have a Personal Identification Number (PIN) or password to protect their contents. In most Android versions a locking pattern is used (see Figure 1), which basically consists of a sequence of movements through 9 dots that works as the password but can easily be spotted because of the required finger movements.

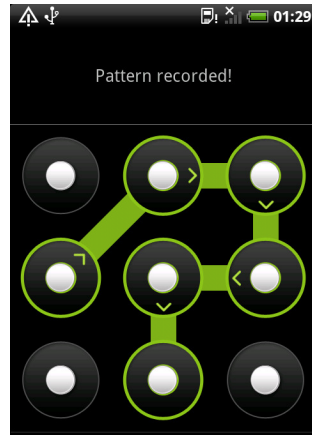


Figure 1 – Android example lock pattern.

What is proposed in this dissertation is the usage of biometrics as an alternative, safer method to protect mobile devices' information.

In fact the developed application aims to fill a gap in the market by presenting a new physical biometric alternative within the Android domain, which is the palmprint, and which is more practical to use than the face, in devices that only possess one camera, and much more practical than behavioral biometrics such as the signature, which is difficult to express in Android devices since most are meant to be used with just the fingers' touch.

The main contributions of this dissertation are:

- The explorations on the possibilities and limitations regarding the usage of biometric recognition software in the Android platform, and the reutilization of code across different platforms.
- A new biometric recognition system adapted from [4] into the mobile devices scenario and using a new technique for template creation.
- A system architecture friendly to mobile commerce applications and adaptable to different devices' specifications.
- A communication interface between Android Java and MATLAB, developed in 2 parts, one for each end of the connection used in the system.
- To the best of the author's knowledge, the first hand-based palmprint recognition system developed for Android, as of June 2011. The system targets a wide range of devices and works with the average hardware specifications.

Hand-Based Biometric Recognition System for Mobile Devices

- An important reference regarding performance results of hand-based palmprint recognition on mobile phones and its feasibility in the Android 2.1 platform.
- System results for known databases but also for hand images from the mobile phone. Since there is a lack for other databases with hand images acquired with similar acquisition devices, these results are important for future biometric recognition research for mobile devices.

This dissertation is structured as follows:

- Chapter 2 – State of the art regarding biometrics and biometric systems.
- Chapter 3 – The proposed secure biometric recognition system, explorations, and implementation details.
- Chapter 4 – Results are presented, explored and discussed regarding the system's performance.
- Chapter 5 – The developed software is presented in terms of graphical user interface and functionality.
- Chapter 6 – Conclusions and future work suggestions are discussed.
- Chapter 7 – The references used throughout the dissertation are listed.

2. Biometrics

2.1. Overview

A biometric trait is a measurable physiological or behavioral trait of an individual, useable for recognition purposes through mathematical and statistical analysis methods or image processing and pattern recognition. In order to be a biometric trait, the human characteristic being considered must satisfy the following requirements [5] [6]:

- **Universality:** each person should have the characteristic.
- **Distinctiveness/Uniqueness:** any two individuals should be sufficiently different in terms of the characteristic, for the metrics used.
- **Permanence:** the characteristic should be sufficiently invariant over a period of time (e.g., it should resist aging, regenerate fast and to the original form if injured).
- **Collectability:** the characteristic should be acquired and measured with simplicity.

One of the many advantages of using biometrics is that the user will not have to remember a password or a locking pattern, for he will be the key himself. This also makes the system much more resistant to attacks because there is a much broader spectrum of large possible inputs that can be the key and those complex inputs are always the ones in use. Although in a textual password system a 512 characters password can be used, typically a much smaller one is chosen for practical reasons, but with biometrics, the complexity of the authentication input is always high. Additionally, those inputs cannot be often perceived with bare eyes by the user's surroundings, unless recorded by a surrounding device, in which case the inputs can be replicated whether text-based or biometric. The difference is in the fact that multi-biometric systems require more inputs to be captured, and for a single biometric trait there is the need that the acquisition fulfils some quality criteria harder to fulfill than to capture text-based passwords or movement patterns such as the ones used in Android's pattern lock.

In order for a biometric system to be successful after its deployment it is also important that some additional issues are considered, both in terms of the biometric traits used and the system's architecture itself [5]:

- **Performance:** which depends on the accuracy of the system, on its speed, and on the efficient usage of the available resources;
- **Acceptability:** which indicates the approval rate the target audience of the system has;
- **Circumvention:** which reflects how easily the system can be fooled using fraudulent methods.

In a mobile device application, with limited sensors, memory and processing capability, it is difficult to achieve a good performance, which makes the optimization of the system's resources usage a vital need.

The lack of a controlled environment and the limitations of the average and generic acquisition hardware of the device are also important barriers that need to be overcome in order to achieve a good accuracy.

A vast number of characteristics have been used in biometric recognition systems (see Figure 2), using different human traits [7] such as:

- **Fingerprint** – the pattern of ridges and valleys on the surface of a fingertip.
- **Palmprint** – the palms of the human hands contain the same type of tissue that fingertips do, but in a larger area.
- **Hand and finger geometry** – shape and size of the hand, including the length and width of the fingers.
- **Iris** – the texture of the colored membrane in the eye, responsible for controlling the diameter and size of the pupil.
- **Face** – shape and position of facial attributes.
- **Ear** – the shape of the ear and the structure of the cartilaginous tissue.
- **Deoxyribonucleic acid (DNA)** – genetic data.
- **Gait** – the way one walks.
- **Signature** – the way a person signs their name.
- **Keystroke** – the way one types on a keyboard.
- **Voice** – acoustic spectrum of the voice.
- **Mouse** – acceleration and speed of mouse pointer movement, clicks frequency and idle time [8].

Hand-Based Biometric Recognition System for Mobile Devices

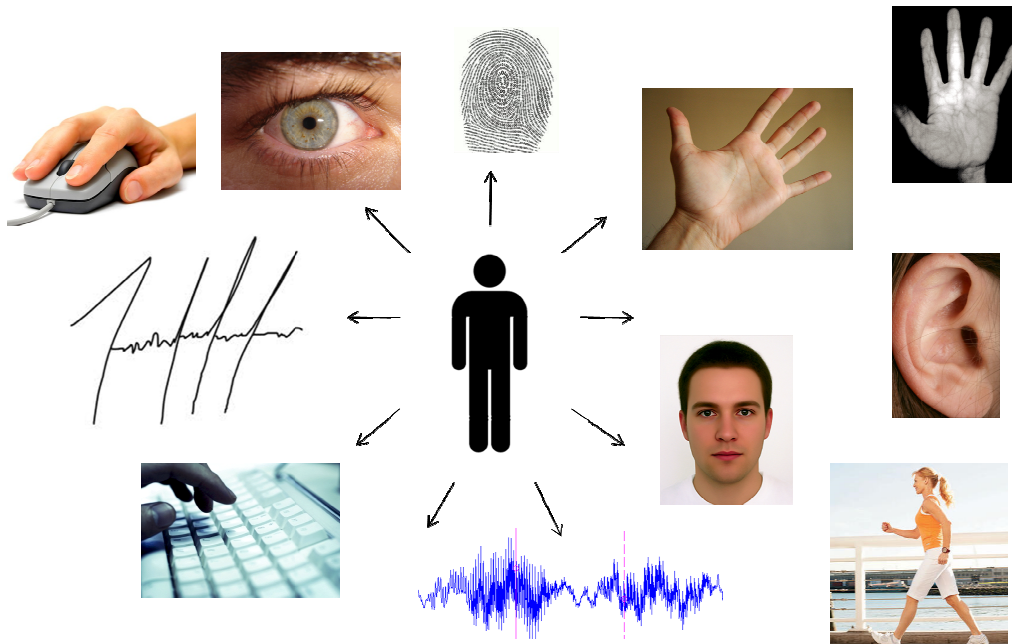


Figure 2 – Types of biometrics. (Adapted individual images from the Web).

Nowadays, most mobile phones come with a camera and with a gravity sensor, in addition to the microphone, so the usage of biometric recognition systems that rely on images of average resolution can be attempted (e.g., palmprint, face, ear, iris), as well as voice recognition or gait.

In the following table, multiple biometrics are compared according to the colored prioritization of the requirements mentioned earlier, highlighting the qualities that make palmprint a good biometric trait for mobile biometric systems [9].

Biometrics	Fingerprint	Face	Hand Geo.	Palmprint	Iris	Voice	Signature
Universality	M	H	M	M	H	M	L
Uniqueness	H	L	M	H	H	L	L
Permanence	H	M	M	M	H	L	L
Collectability	M	H	H	H	M	M	H
Performance	H	L	M	H	H	L	L
Acceptability	M	H	M	M	L	H	H
Circumvention	M	H	M	L	L	H	H
Scalability	H	M	L	H	H	L	H
Maturity	H	M	H	L	M	M	M
Cost	M	L	H	M	H	L	M

Table 1 – Comparison of the human and technical factors of seven popular biometric modalities (Adapted from [9]).

Legend: H – High, M – Moderate, L – Low. The biometric characteristics were prioritized for the mobile palmprint domain through the colors green, yellow and red, which were assigned to the characteristics of higher, moderate and lower importance respectively.

Alternative behavioral biometrics can be used, such as signature recognition or keystroke patterns, but those are particularly ineffective in accuracy and acceptability in Android mobile devices, due to the fact that such devices typically only support finger touch interaction, which makes it difficult for proper signature or usage of virtual keyboards.

When biometric systems use more than one biometric trait they are called multimodal biometric systems and the acquired biometrics can be merged at different levels through the usage of fusion techniques [7].

For systems performing identification on large databases, a linear search is necessary since biometrics have no inherent natural order. For this reason, other techniques can also be used, such as indexation of the database through biometric hash generation techniques [10]. As discussed in [11], soft biometrics can be used to reduce search times in large databases, and when using hand images, biometric traits such as hand geometry can be used as soft biometrics for this purpose, without need for any additional acquisition sensors [12].

Additionally, biometric systems may use continuous biometrics in order to keep the user authenticated throughout his session, by continuously collecting biometric data of the user in a passive way [13].

This is most useful with mobile phones since they are highly portable and very prone to theft. It is possible that a genuine user could be authenticated in his mobile device when the theft occurs, leaving the impostor user with both the device and full access to the system.

2.2. Performance Measures

Whenever a measurement or capture is performed by a biometric system, it is processed and converted into a **feature vector** or **template**, which is a representation of the readings in a format supported by the system. This representation is typically different for multiple measurements of the same user because of the noise introduced by capture conditions, physical and physiological factors, so in order to compare them, a similarity score must be computed between two measurements and a decision must be taken. A threshold t is used to define the maximum differences between templates that is considered to result in a successful authentication decision, or the equivalent correcting

power applied to different templates, in an attempt to correct acquisition noise and making them equal.

Performance measures universally accepted are important to achieve system results with significance, and to allow for different biometric systems to be compared.

2.2.1. Accuracy

Typically, a recognition system's accuracy is measured through the **False Accept Rate** (FAR) and the **False Reject Rate** (FRR) and there is a trade-off between both, due to their inverse dependence with the threshold t , as we can see through their definition (also see Figure 3):

- **FAR** – Corresponds to the probability with which an impostor successfully accesses the system. This is the probability of the similarity score between an impostor's template and a genuine user's template being greater than t .
- **FRR** – Corresponds to the probability with which a genuine user fails to access the system. This is the probability of the similarity score between a captured genuine template and the genuine template used for enrolment being smaller than t .

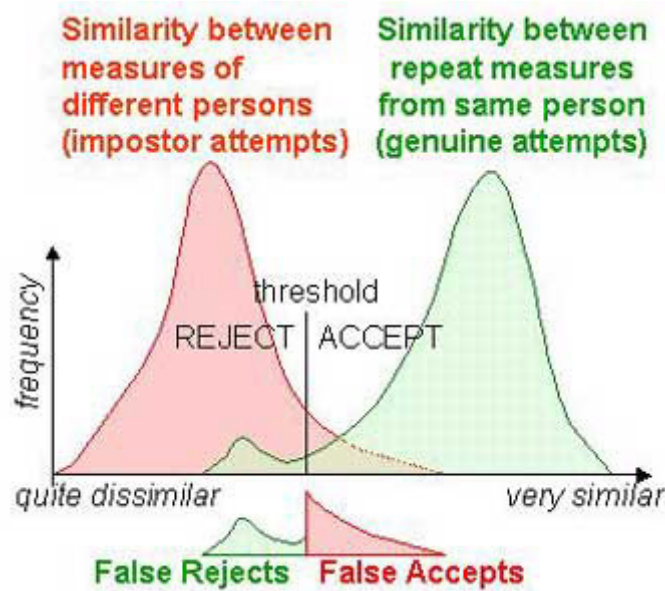


Figure 3 – Illustrative FAR and FRR graph.

Source: http://www.tiresias.org/phoneability/accessible_biometrics_proceedings/images/mansfield_slide_9.jpg

Hand-Based Biometric Recognition System for Mobile Devices

The threshold t must be adjusted for each system, to ensure enough security with a low FAR, but enough flexibility with a FRR not high enough that could make the acquisition noise and variations become a problem for a genuine user.

The balance between FAR and FRR is illustrated in Figure 4, along with the Equal Error Rate (EER) (or Crossover Error Rate (CER)), which is the rate at which FAR and FRR are equal:

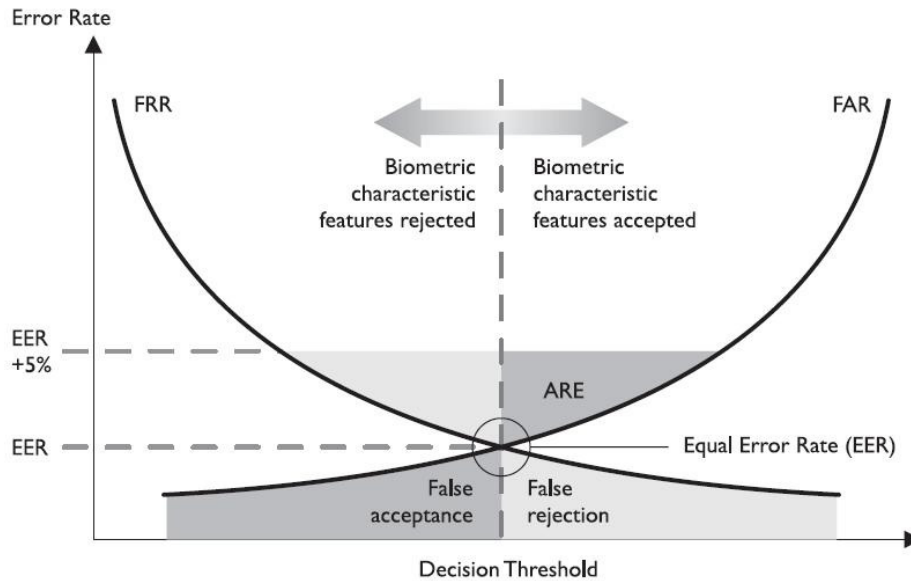


Figure 4 – Interpretative graph of FAR, FRR and Equal Error Rate (EER).

Source: http://2.bp.blogspot.com/_WovOZk-HYJw/TE0UALf-mbI/AAAAAAAAAAU/JvjkwDXG0Wo/s1600/31.JPG

The Asymptotic Relative Efficiency (ARE) region is marked and represents the “best possible” region of operation for a given loss function. This varies for each system, since some systems need high FARs and lower FRRs for practical reasons, while others require low FARs, to ensure that no impostor user enters the system. In Figure 4, the ARE was defined as the region between the EER and EER+5%, above the threshold value. The EER+5% was used as an example, and the considered region may change to consider different EER limits and/or different thresholds, depending on the system’s application domain and its security requisites.

The EER gives an idea about the sensitivity of the system to threshold adjustments, and corresponds to the threshold of equilibrium between FAR and FRR. This point is not necessarily the best operating point for the system, because some systems may require high security, prioritizing FRR over FAR, and others may have speed and practical needs, prioritizing FAR over FRR.

Given the above discussion, it becomes useful to plot FAR against FRR in order to see how the system performs at all the possible threshold values. This plot is called the Receiver Operating Characteristic (ROC) curve (see Figure 5):

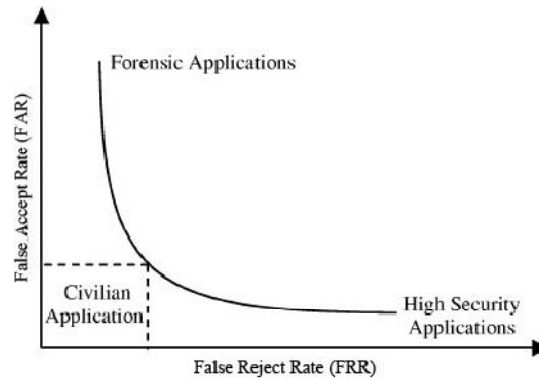


Figure 5 – ROC curve, relating FAR and FRR at different threshold values [4].

Another rate that is also useful and of important significance, especially in mobile systems, is the Failure to Enroll Rate (FER or FTE), which is the rate at which attempts to create a template are unsuccessful. This is mostly caused by insufficient quality of the input data, and is more significant in less constrained capture environments, such as the one considered for a mobile device usage scenario.

2.2.2. Time

The amount of time the system takes to perform its set of operations using a set of techniques is also a performance measure of the system, but not necessarily of the techniques themselves, for that depends on the way they are implemented and used within the system.

This is most important in mobile applications where the computational power is limited and the user should not be left waiting for the application to respond. In order to guarantee the success and satisfaction of the users, and even to commercialize an application in some markets, some requirements of responsiveness must be taken into account.

Ideally, the fastest a system performs, the better, as long as it continues to satisfy the other criteria such as security.

2.2.3. Resources

The resources the system uses to perform its set of operations is also a measure of the system's performance.

In mobile phone applications this is even more important because memory and computational power are limited, and the application will also have to share those scarce resources with other applications that run in the mobile device. By being more efficient, the application will be faster and also ensure a longer lifespan of the battery's charge of the mobile device.

Ideally, the less the memory and Central Processing Unit (CPU) cycles used by the application, the better, as long as it also does not interfere with the other criteria.

2.3. Biometric Systems

Biometric recognition systems perform the biometric recognition of individuals through a set of 5 essential modules (see Figure 6):

- **Data Acquisition** – User interface module where the biometric trait is presented and acquired by the system through sensors.
- **Signal Processing** – Signal processing module, including pre-processing tasks, feature extraction and template creation.
- **Database** – Database where the templates or the results of other functions applied to them, are stored for future comparisons.
- **Matching** – Module responsible for computing a similarity score based on the post-processed acquired data and the data stored in database.
- **Decision** – Decision module which decides if access is granted or not, based on a similarity score and a threshold value.

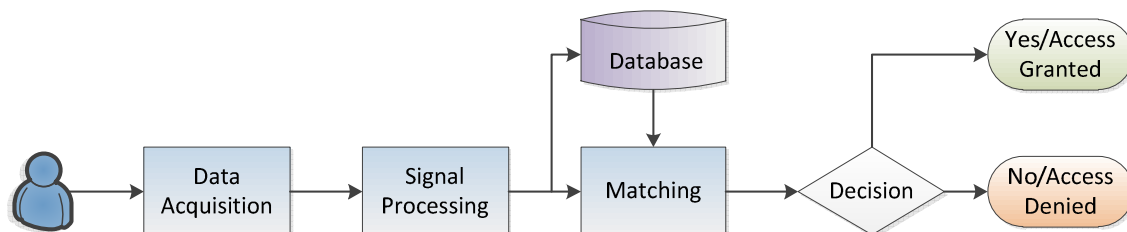


Figure 6 – Main modules of a biometric recognition system.

Based on the application scenario, a biometric system may operate in two modes:

- **Verification Mode** – The system validates a person’s identity, by performing a one-to-one comparison between the stored data and the presented data for a given identity, claimed by the user at authentication time. This claimed identity may be, for example, a username, an email, a PIN, or a smartcard.
- **Identification Mode** – The system recognizes an individual without additional information other than the biometric data, by finding the best match between the presented data, and the data stored in database for each user. This way the system performs a one-to-many comparison.

In either case, there is the need to store recognition-critical data in the database, but this poses some serious security concerns. If the biometric template is stored, whether encrypted or not, it may be stolen and eventually deciphered, resulting in identity theft. To make this worse, the users have a limited set of biometrics, and are unable to generate new ones. The need for data that must be stored for the system to work is typically the weakest link in security for biometric authentication systems.

In the typical password based systems, this problem is addressed by using one-way hash functions. Hash functions have the particularity of providing very different output values (although with the same length) in most circumstances, even when the input values are very similar, as shown below using MD5 as an example hash function, with two similar input strings:

MD5	
Input - P	Output - H(P)
password	5f4dcc3b5aa765d61d8327deb882cf99
passwArd	769b6259c5c2ee6090b945826979e049

Table 2 – Message Digest algorithm 5 (MD5) hash function example.

Since they are one-way functions, those systems can allow the user to provide a password P , hash it with a function H , resulting in $H(P)$ which is then stored for future comparisons, and cannot be reverted back into the original P . Upon authentication the user provides P' , and $H(P')$ is calculated for comparison with the stored $H(P)$. If they are equal, there is a high probability that $P' = P$ and the user is successfully authenticated. Even though two different inputs may result in the same output of a hash

function due to the limited number of outputs, that typically never happens, because the inputs for a given domain are also limited (e.g., passwords may not contain more than x characters in a given system, or generated templates may not be larger than a specified size).

However, in biometric systems, hash functions cannot be used as easily. The acquisition noise makes two templates t and t' from the same user be typically slightly different, resulting in $H(t) \neq H(t')$ most of the time.

In order to solve the secure biometrics template storage problem, biometric systems take one of two approaches: usage of an Error Correcting Code (ECC); or the usage of Encryption.

For ECC-based biometric systems, an Error Correcting Code (ECC) is used to calculate and store the parity bits for a given template along with its hash, instead of storing the template itself. In this way, when authentication is necessary, the stored parity bits are used to “correct” the presented template. In this approach, acquisition noise is a concept equivalent to user variability [14]. The desired correction power depends on each application scenario, but the ideal is that templates from different users are not corrected enough that they become the same, and templates from the same user are corrected enough that the acquisition noise is eliminated, resulting in the same template. Since the stored information consists of only the parity bits and the hash result of the template, it is considered very difficult to recover the biometric based on the stored data. The security of this approach is not provided by the difficulty of inverting the hash function (since it is considered non-invertible), but by the number of the equally likely biometrics which may match the stored data.

For encryption-based biometric systems, homomorphic encryption is applied to the biometric, and authentication uses a protocol based on the homomorphic properties of the used encryption, providing security and data privacy [15]. The logic behind this is that portions of transferred messages reveal no important biometric information by themselves, but when the other end of the connection receives the message, based on those properties and in information the receiving entity knows, it can extract biometric information from the received message. The computational security offered by this approach is that of the cryptographic algorithms used and the used protocol.

Hybrid approaches can be taken, especially when the system works with a client-server architecture, in which an encryption approach is necessary for the communication, and the error correcting code approach can still be used as a means of storing biometric data with transforms considered to be irreversible.

Regardless the approaches that are used, it is important to understand that biometric systems can be the target of multiple security threats at various modules of their architecture and processing stages [16] [17] [18]. In general, the attacks which can be performed on a biometric recognition system can be summarized in four major types [18]:

- **Attacks at the User Interface** – A spoof biometric trait is presented to the system, allowing the attacker to intrude the system in the event it cannot distinguish between the fake biometric trait, and a genuine one. Liveliness detection is a possible solution to deal with this problem.
- **Attacks at the Interface Between Modules** – The information exchanged between modules is intercepted by an attacker and manipulated with the goal of triggering the system's behavior desired by the attacker. Secure communication channels and cryptography are ways to minimize the risk associated with these attacks.
- **Attacks on the Software Modules** – The attacker changes and corrupts the system's behavior by changing the software's code statically using for example Trojan horses, or dynamically, by providing inputs for which the system's algorithms are not prepared, triggering different and unexpected behaviors in the system.
- **Attacks on the Template Database** – The templates database is attacked, compromising to some extent the users' identities. Some problems associated with this attack are the fact that: legitimate access can be denied by the deletion of templates; templates can be stolen and worked outside of the system for the creation of physical spoofs that originate equal templates; the templates can be injected directly at the matching module; the templates can be used for tracking of the user's access to other systems; and in the event the template is not encrypted or can be decrypted and provide a good regeneration of the original biometric traits, the biometric identity of all the system's users can be stolen and compromised.

When looking at a biometric system's architecture, more specific attack points can be looked into, as shown in Figure 7:

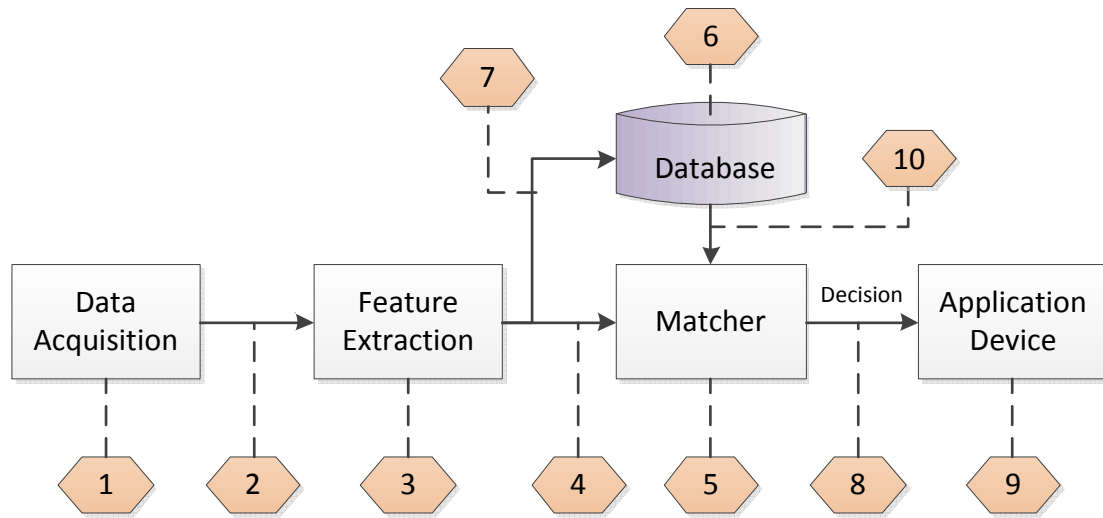


Figure 7 – Attack points in a biometric system.

The red hexagons mark the attack points, which can be described as:

- **Type 1 – Attack on the Data Acquisition Module** – A denial of service can be caused by destruction or corruption of the system's sensors and/or data acquisition components.
- **Types 2, 4, 7, 8, 10 – Attack on the Communication Channel between Modules** – The attacker intercepts the communication channel between two modules, being able to steal, alter, and/or inject channel information. In the event data previously captured is sent again by the attacker, it is also known as a **Replay Attack**.
- **Type 3 – Attack on the Feature Extraction Module** – A Trojan horse can be used by the attacker to remotely control the module and send any desired combination of values as input to the matching module.
- **Type 5 – Attack on the Matcher Module** – The module can be controlled remotely with a Trojan horse, providing the possibility for the attacker to cause a denial of service by forcing the matcher to produce poor scores always, or providing access to the system to himself or to all, by producing high scores.
- **Type 6 – Attack on the System's Database** – The attacker can access the system's database by either cracking an account or exploiting vulnerabilities in

the database software and its provided management interfaces. Once inside, the attacker can steal and compromise the templates of all the system's users.

- **Type 9 – Attack on the Application** – The vulnerabilities in the final application to which the biometric recognition system is attached to are explored and exploited by the attacker. The fact that all software inevitably has bugs and flaws provides room for an attacker to find them and use them to achieve his impostor goals. Additionally, in the event that the application provides other authentication methods as alternatives to biometrics, they can be used for this attack instead.

Every module and each channel between modules is a possible point of attack, and as such, security is an issue that must be addressed at its various levels, from the top view of the application to each of the modules that compose it and the channels between them. The weakest link in this path will typically be what compromises the system's security.

2.4. Existing Mobile Phone Biometric Recognition Systems

Many mobile biometric recognition systems work through the usage of a specific acquisition device, which is connected to the mobile device in question, and provides a way to capture the necessary biometrics within a favorable environment, and with higher standards of quality in the captured data and consequent results, when compared with systems that do not use specific attachable capturing devices.

In the context of mobile devices such as mobile phones, the usage of external biometrics acquisition devices is not practical, and in some cases, not possible because if used for user authentication to third party systems, these systems might not load enough connectivity system settings and/or drivers necessary for the acquisition module and/or the biometric recognition system as a whole, prior to authentication. For example, before logging in to the mobile device, the operating system might not allow the execution of applications which are not part of the OS's core, or load the drivers and ports necessary for the recognition of the external biometrics acquisition device.

The big challenge for biometric recognition systems in mobile phone devices is for them to be capable of capturing biometric data with the typical acquisition sources native to the devices, and managing to optimize the results and security obtained through them.

Hand-Based Biometric Recognition System for Mobile Devices

There are some mobile phone biometric solutions available for most of the mobile phone platforms available, but still many issues to address and evolution to undertake. Those solutions typically make use of the iris [19] [20], the face [21], the voice [22] or a combination of any of those.

For the chosen platform, Android, there are essentially three solutions known within the community:

- **BioWallet from Mobbeel** [23] – which originally makes use of the user's signature for authentication and to allow access to private encrypted files stored within the device.
- **BioLock** – which allows the user to access private data through iris scan, face recognition, or the usage of a traditional text password. This software has been in development for more than six months and is still in beta, being only available to partners and/or press, upon request [24].
- **CredentialME AppLock** [25] – which performs applications protection (lock/unlock access) based on face recognition.

Mobbeel also has software for other devices, which performs biometric recognition through the iris, but according to forum discussions [26] on the possibility of porting it for the Android mobile phones, Mobbeel staff mentions that the image acquisition from the phones does not provide enough overall quality for an accurate iris scan.

The alternative used by Mobbeel is a good solution, but the fact that most Android devices are operated with the user's fingers, authenticating through a signature becomes a rather impractical, inaccurate and cumbersome task.

BioLock's software sounds promising, but the fact that the beta versions have remained unreleased to the public, suggests that many issues are still arising and needing to be addressed.

No systems using the palmprint were found for Android phones at the time of this dissertation's proposal.

For those reasons, and for the promising insight on the usage of the palmprint as a recognition biometric with characteristics well-suited for a mobile phone environment, the usage of this biometric trait was chosen and attempted.

3. Proposed Secure Biometric System

In this section the proposed secure biometric system will be presented, starting with the planning of the target platform, device and tools, and proceeding to the internal system architecture and implementation details.

3.1. Target Platform, Device and Tools

In order to develop a system for a platform, it is important to study and research the available alternatives, their advantages and disadvantages and how they connect with the goal of the work and the application.

Each mobile platform has got its specificities just like any OS, and each device has got its own set of hardware, which will dictate its performance limitations but also the range of sensors that will limit the biometric traits useable by the system. Additionally, the tools to use will influence the productivity and the depth of detail and complexity that can be faced at development time.

3.1.1. Platform

In order to create a biometric recognition system for a mobile device, the first choice to make is to decide on the target platform, bearing in mind that this choice will have a huge impact in the implementation, potential, adaptability and limitations of the application to develop.

There are many mobile platforms available for development, such as Symbian, Blackberry, Android, iOS (for iPhone), Windows Mobile, Palm, and various others, including cross-platform choices such as Java Micro Edition (Java ME). Each platform with their own set of tools and communities of developers, each with their pros and cons.

At first thought, cross-platform choices would be the best way to go for a flexible application, capable of running on multiple Operating Systems and consequently targeting a wider range of mobile devices. However, Java ME is harder to understand, less responsive and provides less Java Standard Edition (Java SE) support when compared to Android [27].

Android was the taken choice for multiple reasons:

- **Open source Operating System (OS)** – Code available online, based on Linux, and can be changed to make specific versions if necessary, extending possible limitations and also making it adaptable to different types of hardware, including customized hardware designed with specific purposes in mind. Additionally, continuous updates are made available, providing a steady evolution of the OS;
- **Java Programming Language** – The applications that run on the OS use the Java Dalvik Virtual Machine (DVM), which runs applications written using the Java Programming language and making use of a specific set of android libraries. This also reduces the learning curve for people experienced with the Java Programming Language, which is the case for this dissertation;
- **Computational efficiency** – Java is a slow language by nature, because it is interpreted, meaning that the code is compiled into class files which are then read by virtual machines specific to each platform, which in turn “translates” them into byte code to execute. However, it is also possible to optimize code within the application by the usage of the Native Development Kit (NDK), and Java Native Interface (JNI), which allow for the execution of C and C++ code from within the Java application. Additionally, the Dalvik Virtual Machine used within Android has got lots of optimizations to improve performance and optimize the energy spent to translate and process the code. In [28], it is shown that for simple algorithmic tasks, the implementations are almost as fast in Java as they are in C/C++, as long as they are iterative. For recursive implementations big improvements are achieved through the C/C++ language.
- **Growing share of mobile phones** – According to Gartner’s forecast from April 2011 in [29], Android will have a worldwide market share of around 38.5% in 2011, and is forecast to continue growing, reaching 49.2% in 2012.
- **Supported by the Open Handset Alliance** – Supported by a group of over 60 international device manufacturers and service carriers such as Telefonica, Vodafone, T Mobile, and various others.
- **Efficient Tools and Development** – According to a study from Mobile Developer Economics in 2010 [30], done with a universe containing both beginners and experts from the various platforms, Android is the faster platform to master, and the faster to debug, making it one of the best choices to develop

applications at a steady pace, and with time constraints regarding their delivery (see Figure 8 and Figure 9).

- **“Easily” Cross-Platform** – Since applications developed for Android are mostly written in the Java Programming Language, converting the application to Java ME to make it cross-platform becomes easier, for most of the functional and technical code remains the same. What changes is the set of core libraries used, which imply an architecture change in the way the application works, to respect the different lifecycles imposed by each platform.

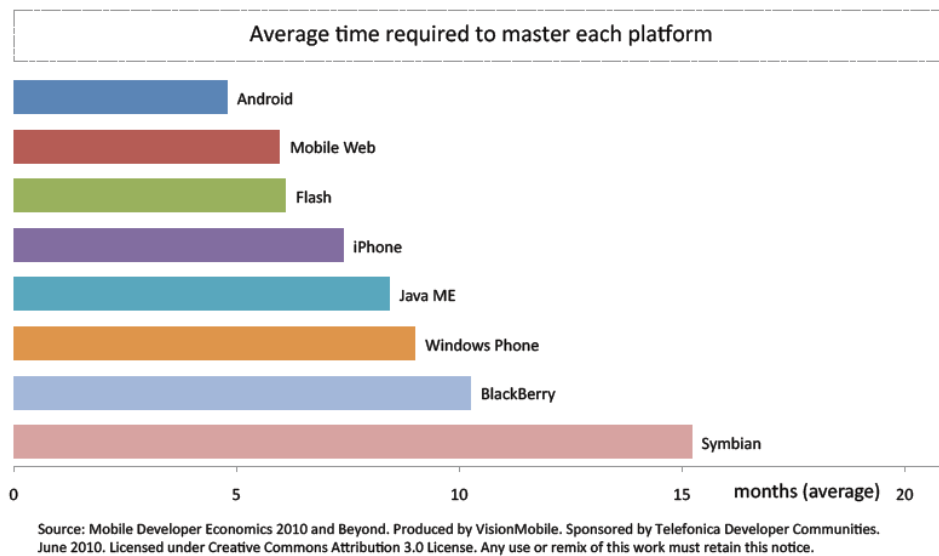


Figure 8 – Graph of the average time for platform mastery [30].

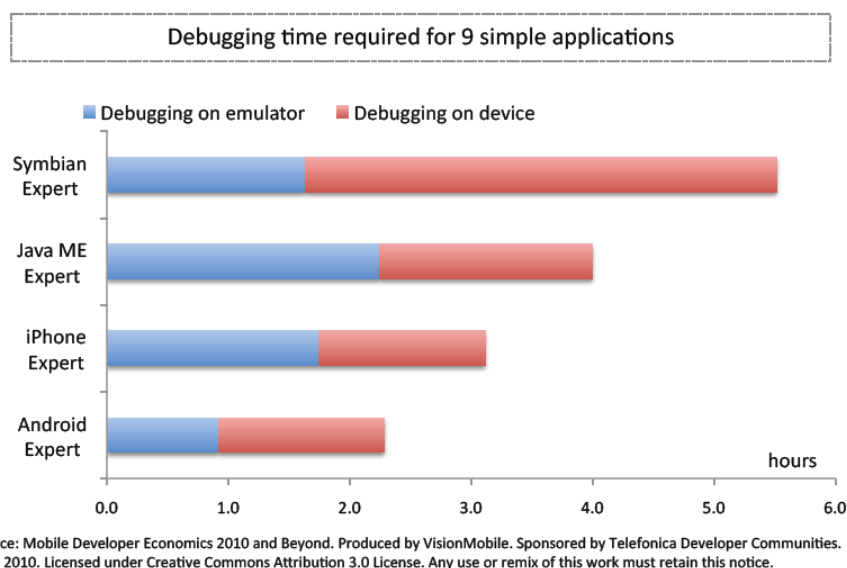


Figure 9 – Graph of the average debug time required for each platform [30].

Upon deciding for the Android platform, it is important to choose which version of the platform the application will target, bearing in mind that the versions have been progressively improved and optimized with each release and that at the time of choosing there were mostly devices with Android 1.6, 2.1 and 2.2 available in the market.

There is a difference between the hardware specifications of mobile phones that come with the Android 1.6 (and previous versions) by default, and those that come with 2.1 or newer versions. Typically, devices running 1.6 or older versions have slower CPUs, in the range of 0.5 GHz and about 256 MB Random-Access Memory (RAM). There are however some exceptions (such as Gigabyte GSmart G1305 Boston, also known as Codfish), where the devices run Android 1.6 and have a CPU of 600 MHz, also allowing the user to update to version 2.1. This means that the mobile devices are evolving towards meeting the optimal specifications for running Android 2.1 and beyond.

Another big difference between Android 1.6 and 2.1 is the amount of heap memory each application is allowed to allocate in an instance of the Dalvik Java Virtual Machine. In Android 1.6 and older versions, this limit is set to 16 MB, but in Android 2.1 and forward this limit is 24 MB. This difference, along with the optimizations from version 2.1, could be vital for biometric recognition systems, where image processing algorithms may require allocating much memory to process acquired images and signals. This makes version 2.1 a wiser choice than 1.6 because it will allow more flexibility to deal with the limitations of developing for a mobile device with limited resources.

As for version 2.2, it has got further speed improvements provided by the platform, and also the possibility of allowing users to install applications to the expandable memory. However, these advantages are not significant enough to make it a better choice at the time of decision, because at the time of this decision there was a higher share of users running version 2.1 than 2.2 according to the data published on Android developer's official website, with data collected during two weeks ending on 1 November, 2010, as shown below in Table 3 and Figure 10:

Platform	API Level	Distribution
Android 2.2 (Froyo)	8	36.2%
Android 2.1 (Eclair)	7	40.8%
Android 1.6 (Donut)	4	15.0%
Android 1.5 (Cupcake)	3	7.9%

Table 3 – Android versions distribution table as of 1 November 2010 [31].

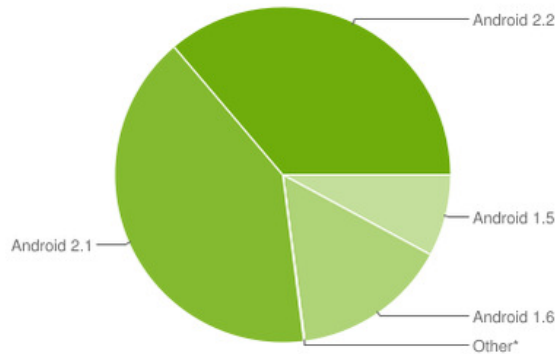


Figure 10 – Android versions distribution graph as of 1 November 2010 [31].

Another note of importance is the fact that a single Android application can target a whole range of Android versions as long as it uses calls from an Application Programming Interface (API) present in the targeted Android versions. Each version has an integer number that identifies it, called API Level. The Android APIs are typically incremental in their releases, meaning that the API available in Android 2.2 contains the calls provided by version 2.1. Through this, it is possible to develop an application that will target 2.1 and consequently 2.2, allowing the application to target a total of 77% of the Android users according to Table 3 previously shown, bearing in mind that that percentage will grow as users using older versions update their platform versions.

Taking a second look at the versions in use, with the data published by Android Developers website collected for two weeks, ending on May 2, 2011 (see Table 4 and Figure 11), it is possible to see that, as initially predicted, the tendency is for users to update to the newest versions, and the versions previous to 2.1 start becoming obsolete.

Platform	API Level	Distribution
Android 1.5	3	2.3%
Android 1.6	4	3.0%
Android 2.1	7	24.5%
Android 2.2	8	65.9%
Android 2.3	9	1.0%
Android 2.3.3	10	3.0%
Android 3.0	11	0.3%

Table 4 – Android versions distribution table as of 2 May 2011 [31].

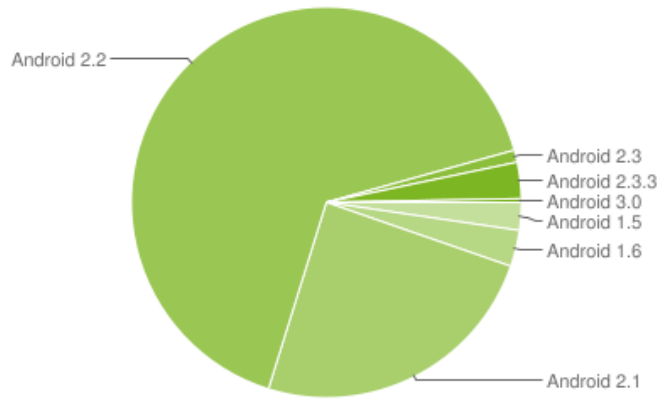


Figure 11 – Android versions distribution graph as of 2 May 2011 [31].

With this, it becomes clear that the choice taken was the most appropriate because an application targeting Android 2.1 will also be targeting all versions after it, which means that the developed application will be targeting 94.7% of the market’s mobile phones. Since new devices keep coming with new versions of the Android OS and previous ones keep getting updated, according to the graph of the evolution (see Figure 12), it is to expect that the number of devices running 1.6 and previous versions continues to decrease, increasing the percentage of targeted devices even further.

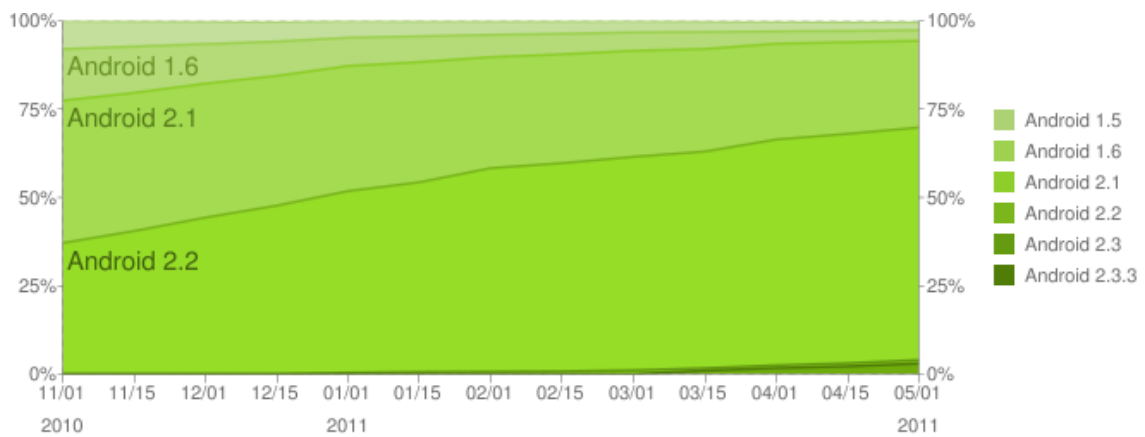


Figure 12 – Android versions historical distribution graph from November 2010 to 2nd May 2011 [31].

There are tools available in the community such as the Compatibility Test Suite (CTS) Framework for Android [32], provided by the Open Handset Alliance, and which allows for testing and tuning Android applications towards compatibility through multiple platform versions.

The possibility of adapting the system to make a new implementation for other mobile platforms in the future, such as Java ME is also not completely limited by the specificities of Android’s Java, because regardless the fact the Android platform

imposes a specific architecture in an application and its Graphical User Interface (GUI), the core part of the application, regarding the image processing operations and security functionality is isolated from the rest in an abstract layer of its own, using Java libraries and operations that are mutually supported by Android's Java and standard Java. Below, a set of the most relevant libraries supported by both platforms is shown:

- **java.io** – File and stream Input/Output (I/O)
- **java.lang (except java.lang.management)** – Language and exception support
- **java.math** – Big numbers, rounding, precision
- **java.net** – Network I/O, Uniform Resource Locators (URLs), sockets
- **java.nio** – File and channel I/O
- **java.security** – Authorization, certificates, public keys
- **java.sql** – Database interfaces
- **java.text** – Formatting, natural language, collation
- **java.util (also java.util.concurrent)** – Lists, maps, sets, arrays, collections
- **javax.crypto** – Ciphers, public keys
- **javax.net** – Socket factories, Secure Socket Layer (SSL)
- **javax.security (except javax.security.auth.kerberos, javax.security.auth.spi, and javax.security.sasl)** – General security frameworks
- **javax.sound** – Music and sound effects
- **javax.sql (except javax.sql.rowset)** – More database interfaces
- **javax.xml.parsers** – Extensible Markup Language (XML) parsing
- **org.w3c.dom (but not sub-packages)** – Document Object Model (DOM) nodes and elements
- **org.xml.sax** – Simple API for XML

3.1.2. Device

The device choice was planned in July 2010. At this point, the official Google development phone known as Nexus One was becoming discontinued, and the new version of that phone was not yet available. After a technical specifications analysis of the mobile phones available on the market at that time and available for purchase, HTC Desire was the chosen device (see Figure 13).



Figure 13 – HTC Desire device. Front side on the left, and back side on the right of the figure.

The device’s specifications are shown in Table 5 below:

Attribute	Specifications
Default OS	Android 2.1
CPU	Qualcomm Snapdragon QSD8250 1 GHz processor
Memory	576 MB RAM; 512 MB ROM
Camera	5 MP, 2592 x 1944 pixels, autofocus, LED flash
Card Slot	4 GB (up to 32 GB)
Extras	Multi-touch input method; Accelerometer sensor

Table 5 – HTC Desire device specifications summary.

Although other devices could have been chosen, the choice taken and the relevance of the specifications presented in the previous table fulfill the following logic:

- **Heap limit** – The fact that the device comes with Android 2.1 will allow the developed application to use a 24 MB heap instead of the 16 MB allocated for Android 1.6 applications and before, as well as to take advantage of improved performance optimizations that the Android OS has had from version 1.6 to 2.1.
- **Camera** – This phone also possesses a 5 MP camera, which is typical for Android devices built for Android 2.1, and puts this phone in the average-phone situation for what concerns the camera.
- **CPU** – The 1GHz CPU, 576 MB RAM and 512 MB Read-Only Memory (ROM) make it one of the best phones for processing experimentation in comparison to all the other considered devices available at the time of the planning and purchase. This way the development process was optimized for both speed and research.

- **Extra Sensors** – Multi-touch input and the accelerometer sensor are two extras which also allow the possibility of further experimentation at a later stage of the work, regarding the usage of additional biometrics, to further improve the system and its performance rates.
- **External Memory Card** – Less important, but still worth mentioning, is the fact that the 4GB card included will allow for the testing of the system when installed both directly on the device, or in its memory card, in case the OS would be updated to Android 2.2, since that version allows for those two installation locations.

3.1.3. Development Tools

In order to develop the proposed application, along with its documentation and diagrams, a set of tools will be used for development, testing and optimization purposes:

- **Eclipse Integrated Development Environment (IDE)** – Eclipse is a powerful tool developed in Java, with the goal of aiding the programmer in the task of developing the application. It has many plug-ins that can be integrated with it to add additional functionalities, and it also links together compilers and settings for the used languages and tools, making it easier to use them altogether, in a more efficient way during the development task.
- **Java Development Kit (JDK)** – It includes a set of utilities that make it possible to develop software systems for the Java Platform, used by Android. It includes the compiler and a set of libraries.
- **Android Standard Development Kit (SDK)** – Consists of the managing tool for the installed Android APIs, as well the Android Virtual Device (AVD) Manager tool, to manage virtual devices to use with the emulator and their hardware specifications. Additionally, it comes with a set of tools for the testing of application packages, optimization of code and other features. Each Android API also includes packages with classes specific for instrumentation and testing.
- **Android Development Tools (ADT)** – It is a plugin for Eclipse, provided by Google and Android developers, that comes with a set of tools and functionality used in Android development, such as editors for specific file formats, special XML parsers using special templates specific to Android, an Android emulator for testing and debugging applications within a virtual environment, Dalvik

Debug Monitor Server (DDMS) where applications can be deeply traced and analyzed in terms of threads, calls, and heap memory used, and further tested by allowing the possibility to send signals to the virtual device, such as virtual phone calls, position coordinates, and various other useful functionality.

The set of tools and corresponding versions used during development are summarized in the following table:

Tool	Version
Eclipse IDE	3.5.2
JDK	JDK 6 Update 22 (Java SE)
Android SDK	revision 7
Android API	Android 2.1-update1, API Level 7, revision 2
ADT & DDMS	0.9.9.v201009221407-60953

Table 6 – Development tools and versions.

3.1.4. Libraries and Possibilities

The application being developed requires plenty of image processing algorithms already commonly used in other platforms, and as such, multiple alternatives were considered:

- Use an existing image processing library for Android
- Use an existing image processing library for Java, and use it within the Android environment
- Use a C/C++ library within Android, by using the JNI and NDK
- Use MATLAB Code Compiler (MCC) to compile MATLAB files into a C/C++ library
- Use MATLAB Builder JA to generate Java wrapper classes with the MATLAB code to be used
- Use virtualization tools to execute a MATLAB runtime within the Android environment
- Create a new Android image processing library from scratch

Regarding the first option, there is mostly one known image processing library specifically designed for Android, called Jon’s Java Imaging Library (JJIL), and which implements a vast amount of image processing algorithms. However, there are some problems with this library:

- The fact that it defines its own data structures based on a fairly redundant representation of the original matrix data from the images leads to huge matrices being passed in the constructor every time an image needs to be processed.
- Since the image processing algorithms are defined in chain sequences, a whole set of objects needs to be created in order to achieve a specific image operation of a more complex nature.
- The API for the image operations provided by the library does not offer the desired information about the operation and the way it is performed. There are also cases in which no descriptive information is provided by the API at all.
- It is not clear how new image processing algorithms can be defined to extend the library, and in the event those are created they will be deeply linked to the library's architecture. This will make them look less intuitive and less reusable in other libraries for other mobile platforms which could possibly have use for similar algorithms.

For those reasons, the JJIL library was not used, as it could have been limiting in the long run, and it hides too many details about the way images are processed, while at the same time enforcing its own structure.

The second alternative would be to use a Java image processing library such as Java Advanced Imaging (JAI). However, at the time this project was started, the source code for some of the most powerful libraries such as JAI was not available, and plenty of other projects hosted in Java websites were taking on hosting changes, which made some resources unavailable or at inconsistent states. The java.net website for community source projects has taken structural changes and not all the source code was available at the time it was considered. As seen in the project properties section of the java.net website for that project [33], "Jai-core is a subproject of Jai -- Placeholder, was started in January 2011 ...".

As for the third option, C/C++ libraries could be converted and adapted, at the cost of losing the straight forward compatibility that Java would provide to all the Android 2.1 and forward devices, since those libraries would have to be compiled for different and specific mobile phone processor models. Additionally, the performance boost is not guaranteed in the sense that each time there is a JNI call, much overhead is necessary, and long arguments need to be converted and passed by the Java Virtual Machine into

the C/C++ libraries. Another important thing to keep in mind is that debugging, which is already a difficult task in mobile environments becomes further difficult when mixing C/C++ code with the Java code used as the core of the environment's applications. This would greatly reduce productivity. Also of notice is the fact that access to Bitmap buffers with some NDK versions is limited to Android 2.2 versions and beyond.

The usage of libraries such as Open Computer Vision Library (OpenCV) were also considered for this purpose, but its conversion for use in Android is not direct since recent versions of OpenCV hold much functionality from C++ which is not compatible or convertible to the Java language. This would imply using old outdated OpenCV versions. A new version of OpenCV specific for use within Android started being developed in the beginning of January 2011, but it is still at a very early stage of development, with few examples available, limited functionality and little guarantees of reliability.

The fourth option is similar, but would consist of converting MATLAB files into C/C++ files through MCC. However, the problem with this is that it can only be done easily to create executable files, assuming the target platform will hold a runtime version of MATLAB, which is not the case for the used mobile phone. The C/C++ compiler option produces code difficult to read and with plenty of gaps due to toolbox dependencies which would have to be introduced by hand. In addition, this solution poses the same problems than the one before since it would produce C/C++ code to be called from within a Java environment through JNI.

The fifth option would be a solution if Android was a supported platform for MATLAB, which is not the case at the point of this decision. Although this option could be used to produce code for a web context, it was not used because this would result in less efficiency due to the additional interaction steps and protocol layers, as well as the necessary Java objects to handle the requests. If a web solution is adopted, it should be implemented directly in MATLAB, for a better performance.

The sixth option would require the usage of virtualization tools, running a virtual machine with Windows or another platform MATLAB-compatible, in order to have its runtime running within, hence making it capable of executing MATLAB code. At the moment of this dissertation virtualization within the Android platform is still quite young and taking its first steps, which would make this a poor and limited choice, since

virtualization has still many issues to address before it matures in this platform and mobile platforms in general.

The seventh and last considered alternative would be to develop an image processing library from scratch. This choice was taken temporarily in an attempt to gain insight both about the task's complexity and also about image processing within Android, in order to better realize how available libraries achieve it in this platform. A simple image processing library was created, but the complexity required to implement the desired system would imply more time than what was available to complete the dissertation as a whole.

For those reasons, a new architecture was considered for the system, with a client and the server, where the server runs in MATLAB and does the processing directly in this language, making use of Java within, for the communication interface (see Figure 14).

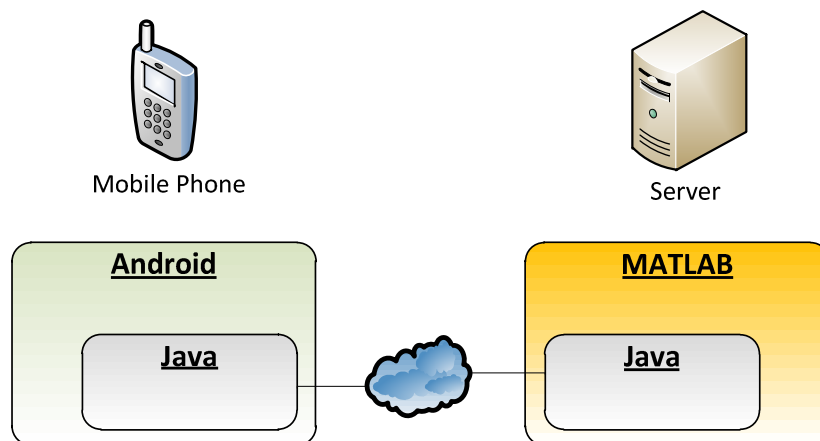


Figure 14 – System architecture and programming languages overview.

This way, the system was made possible, using an approach similar in concept to the fifth option previously explained, but with a higher efficiency and control, since instead of using computer-generated Java wrapper classes targeting a plain Java environment, Android Java classes were specifically developed.

The existing MATLAB system architecture was changed, an interface layer for the communication was implemented, and an Android client was developed from scratch. A new feature extraction technique was used by the system in order to better adapt it to the mobile devices' domain.

In this chapter's next section this architecture will be fully described in detail.

3.2. System Architecture

3.2.1. Introduction

The proposed system's architecture is adapted from the biometric recognition system proposed by M. Ramalho in [4] and his developments over the system proposed by T. Sanches in [34]. The initial system architecture for this work is illustrated in Figure 15. The Feature Vector Binarization (FVB) module was removed, and a Pre-segmentation module was added to the system.

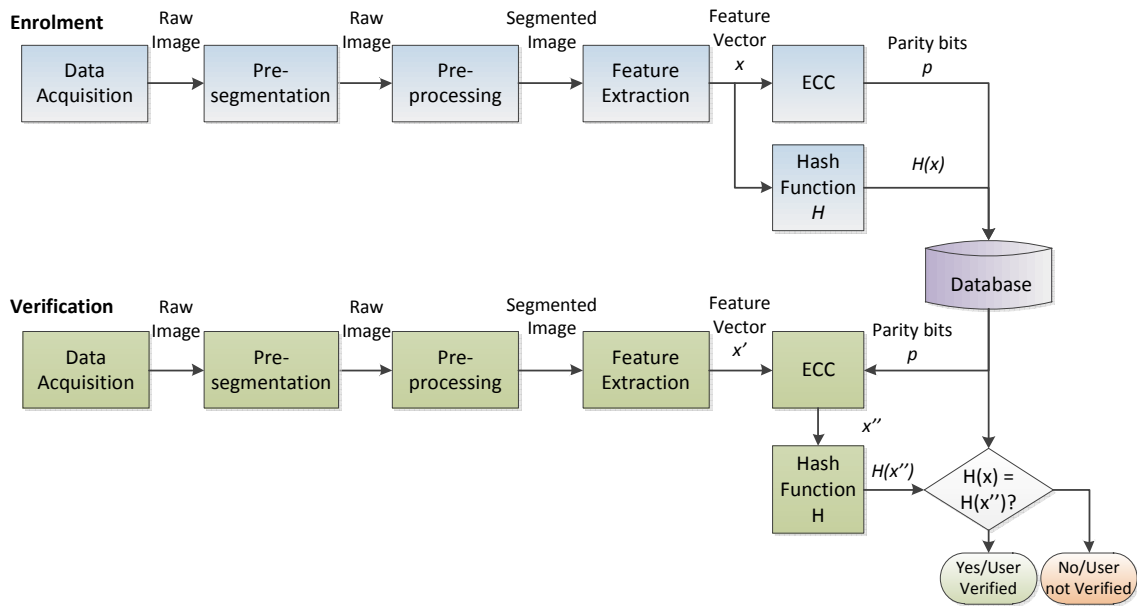


Figure 15 – Proposed system architecture (simplified).

The goal of the new pre-segmentation module is to provide feedback to the user of what the application perceives to be the user's hand in each frame captured by the device's camera, aiding in the data acquisition task and improving usability. This module is described in detail in section 3.3.1 Data acquisition and Pre-segmentation.

In the **enrolment** stage, the input biometric data is acquired from the user, pre-processed, and then the features are extracted. An ECC is then applied to the resulting template x holding the features or their representation, and the produced parity-check bits p are stored in the database, along with the result $H(x)$ of a hash function H applied to that template x . The parity-check bits p will allow the correction of templates from future verification attempts, within correction boundaries that depend on each application's scenario.

In the **verification** stage, the input biometric data presented to the system will result in a template x' which will typically be different from x even for the same user, due to the acquisition noise and eventual natural changes in the user's biometric trait itself. If the template x' differs from the original template x in a value smaller than the correction power of the ECC, x' will be corrected into the original template, and $x'' = x$, consequently resulting in $H(x) = H(x'')$.

The template storage provided by this system is secure because the only information stored for each user is the pair $(p, H(x))$ containing the parity-check bits p which reveal little information about the original template x , and the hash function result $H(x)$ which is computationally hard to invert even when knowing the hash function used.

Since the ECC and hash functions use binary inputs, in [4] there is the need for the FVB module to be integrated in the system, to convert the real-valued templates returned by the feature extraction algorithm, either Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA), into fixed-length statistically independent binary values. However, in the proposed system such module is not necessary because the used Orthogonal Line Ordinal Features (OLOF) feature extraction algorithm originates templates already in binary, by extracting ordinal features from the palm, thus not requiring the post-quantization of the feature space with a total of N equiprobable intervals, and association of each quantization interval with a binary code. Each feature has its correspondent binary code concatenated with the previously coded features directly from OLOF, resulting in the binary feature vector output b .

Additionally, as mentioned in [4], to meet the diversity and revocability requirements desirable in a biometric recognition system, a Bitwise Exclusive Disjunction (XOR) module is added which computes the result $z = w \oplus b$, where w is a random set of bits, different for each user, and b is the binary feature vector that resulted from the feature extraction module.

The complete system architecture is shown in Figure 16.

Hand-Based Biometric Recognition System for Mobile Devices

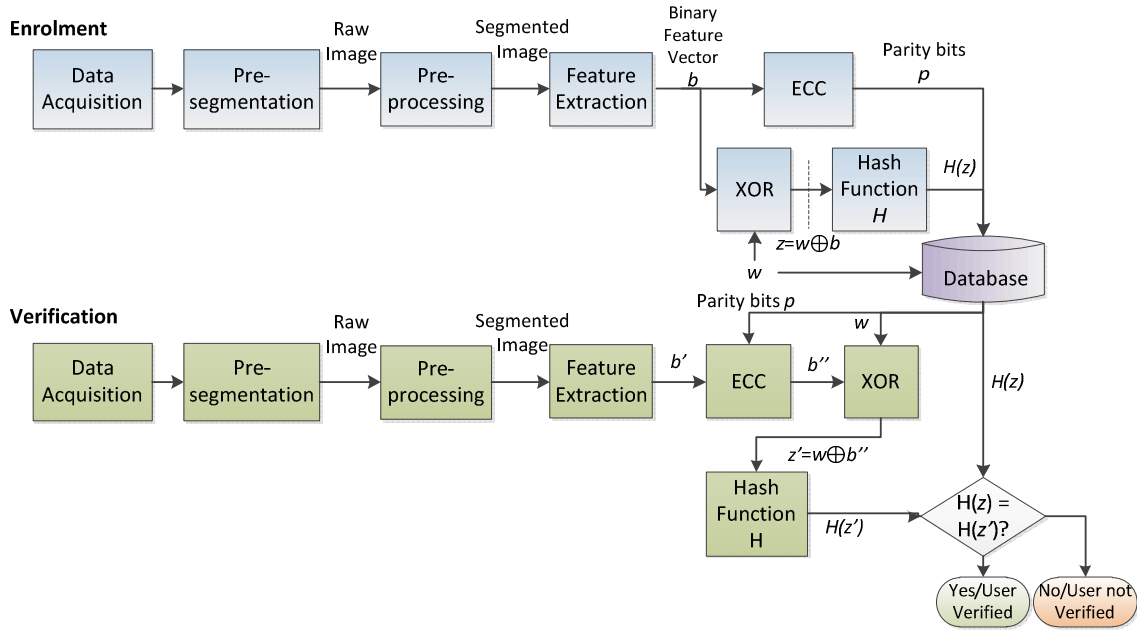


Figure 16 – Secure biometric system architecture.

For this system, which is considered secure because w is randomly generated independently from the user's biometric data, the set $(w, H(z), p)$ is stored for each user.

The system's main modules are described below:

- **Data Acquisition** – acquisition module where sensors capture the biometric characteristics relevant for the system.
- **Pre-segmentation** – pre-segmentation module where the camera frames are processed and the application displays real-time detection of the hand.
- **Pre-processing** – pre-processing module where the acquired image is aligned, transformed and improved, and the Region of Interest (ROI) is identified.
- **Feature Extraction** – module where the features are extracted from the segmented image, and put together into a feature vector or template.
- **Error Correcting Code (ECC)** – in the enrolment stage, parity-check bits p are extracted from the binary string b . In the verification stage, a correction of the b' (noisy version of b) is attempted, with the previously stored parity-check bits.
- **XOR** – provides the system with revocable templates. Performs the result $z = w \oplus b$, which is the bitwise exclusive-or between a randomly generated binary value w and a binary feature vector b .

- **Hash Function** – Ensures the privacy of the stored biometric authentication information due to its irreversible properties. In the enrolment stage the hash result $H(z)$ is stored in the database. In the verification stage, the result $H(z')$ is compared to the hash value stored at enrolment stage.
- **Decision** – Decides if the user is successfully verified based on the equation $H(z) = H(z')$.

As a consequence of the research and exploration mentioned in section 3.1.4 Libraries and Possibilities, the proposed top-level implementation architecture for the presented biometric recognition system as a whole assumes a Web architecture. The mobile phone device runs a client application in its Android platform, and communicates to a server implemented in MATLAB, using the TCP/IP protocol, as shown in the figure below:

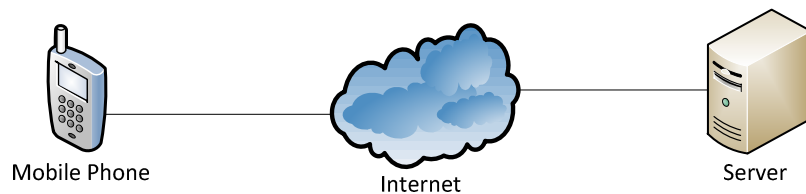


Figure 17 – Top level system architecture overview.

This makes the need for a communication interface which is described in section 3.3.2 Communication Interfaces and Protocol. Both the client and the server communicate through a software interface programmed in Java, making use of the `java.net` package, common in both Java SE and Android. The server code is developed in MATLAB but Java code is used directly within the MATLAB program, for the communication interface.

The TCP/IP protocol ensures that the information shared between both parties is delivered in the correct order, allowing the implemented connection interfaces to follow a “protocol” of their own for the communication that will always be respected for what concerns the TCP connection used beneath.

Additionally, security protocols such as SSL can be used over TCP/IP, to ensure the security of the communication and the system. If no certificates are to be used in the transactions, the usage of simple ciphered streams is equally possible over TCP/IP.

Hand-Based Biometric Recognition System for Mobile Devices

Since the Internet is used for the connection, the access method is transparent, meaning the device can use a nearby Wi-fi network, 3G, 4G or any other access method supported to connect to the Internet, to be able to use this system (see Figure 18).

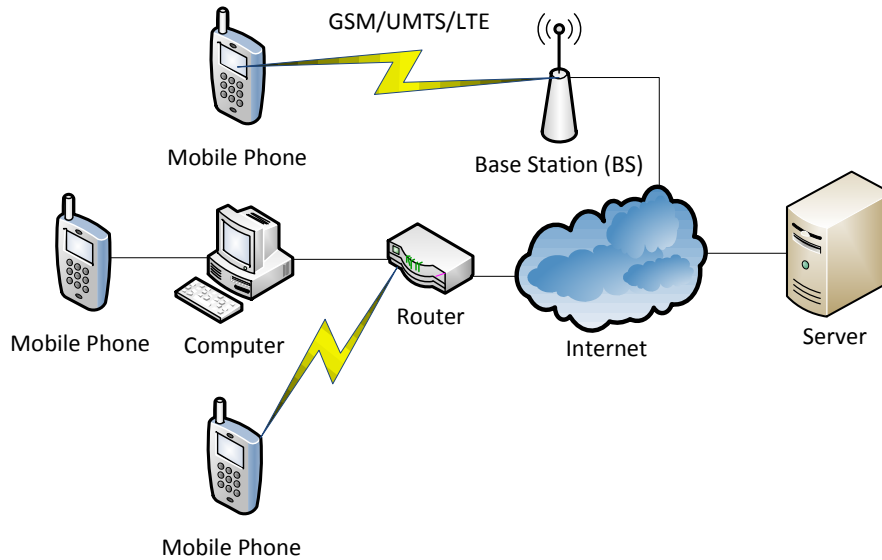


Figure 18 – Example internet access methods.

Combining the system architecture with the top level architecture that was chosen, we obtain Figure 19, shown below, where the role of the mobile device client and the server are explicit, along with the need for a communication interface between them. The pre-processing module in the mobile device client side is highly adaptable to the computational power and resources of the device.

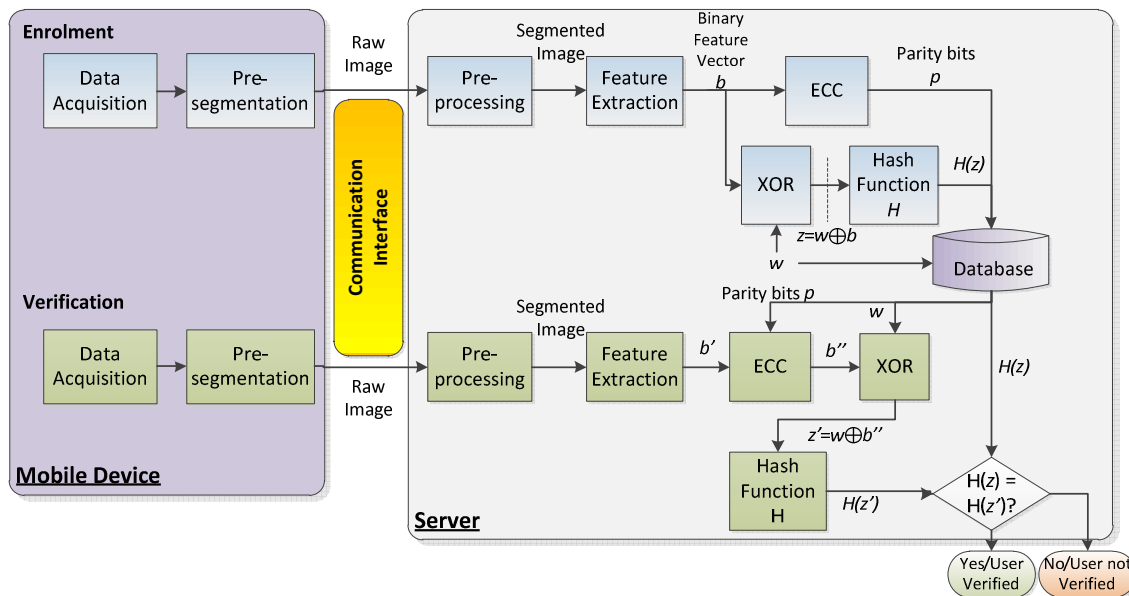


Figure 19 – Combined system architecture.

Since the platforms used by the mobile device and the server are different, two different communication interfaces were built, one for each end of the connection, and a communication protocol was defined.

3.2.2. Application Scenarios

The developed system implements biometric recognition authentication based on the palmprint, for mobile devices, in the web context.

The system is generic enough that it works as a login system, as a means to get access and manage secure items, or to perform any arbitrary operations in a secure way.

For these reasons, the system can be connected to a vast range of applications and services. Some possible examples are as follows:

- **Near Field Communications (NFC) Commerce** – The users could buy items with their mobile phone and use the developed biometric recognition system as a way to authenticate themselves and ensure that the holder of the mobile phone is genuine, validating the purchase. An example of this is a project by the name of Isis [35], whose goal is to build a mobile commerce network capable of allowing users to make purchases through smartphones. The NFC chip, which is an integrated circuit capable of transmitting and reading data for short distances (up to some meters), has three major uses: can work as a payment card; as an interface for a peer-to-peer connection; or as a reader of NFC-compatible tags [36], such as Radio-Frequency Identification (RFID) tags, which are thin integrated circuits capable of storing and transmitting information, and that can be easily attached or embedded to other objects.
- **Passwords management** – Many password management services exist, that keep multiple accounts and passwords, having the user remember one single password to access them. The palmprint could be used as that master password, providing additional security in such an application, especially for the mobile phones domain. Online password managers such as Clipperz could be used [37], but with the possibility of using palmprint as authentication method, with mobile devices.
- **File storage** – In the same way, palmprint in such a system can be used as the means of securing access to an online file storage service such as Dropbox [38].

- **Phone security application** – The system could be used in a similar way as to those of security applications available for mobile phones, which lock applications, trigger alarms and perform other actions in case of a detected security breach. This system could perform such but through the usage of the palmprint for authentication. An example of an application like this but which works with textual passwords is Lookout Mobile Security [39].

In Figure 20, a wide range of NFC technology usage examples in daily life situations are illustrated. Biometrics, and the proposed palmprint system could be present as part of any or all of them. Additional examples of NFC potential are mentioned in the Official NFC Forum in [40].

A Day in the Life of an NFC Mobile Phone User

Area	STATION AIRPORT	VEHICLE	OFFICE	STORE RESTAURANT	THEATER STADIUM	ANYWHERE
Usage of NFC Mobile Phone	Pass gate Get Information from smart poster Get Information from Information kiosk Pay bus/taxi fare	Personalize seat position Use to represent driver's license Pay parking fee	Enter/exit office Exchange business cards Log in to PC; Print using copier machine	Pay by credit card Get loyalty points Get and use coupon Share Information and coupon among users	Pass entrance Get event information	Download and personalize application Check usage history Download ticket Lock phone remotely
Service Industries	Mass and Public Transport Advertising	Drivers and Vehicle Services	Security	Banking Retail Credit Card	Entertainment	Any

Figure 20 – NFC usage scenarios.

Source: <http://theresultspeople.com/wp-content/uploads/trp-english/dayinthelife-mobile.jpg>

There are innumerable other applications possible with which the developed system could be used. Basically, everything with a log-in prompt, or module related with user authentication can be empowered by the usage of the proposed system.

The detailed description of the system's components can be found in the next section and the corresponding subsections. To isolate the graphical aspects and software view of the application from the system's architecture, aspects regarding the Android developed application's graphical user interface and functionality are described later on, in chapter 6 of this dissertation.

3.3. Implementation Details

The system was adapted from the secure biometric recognition system proposed in [4] but with the mobile phone devices scenario in mind.

In the newly developed system, a Web architecture is proposed, with an Android application developed for the data acquisition and pre-segmentation on the client's end, and the rest of the biometric system's logic (pre-processing, feature extraction, database and matching) on the server side. Two communication interfaces were built for both ends of the connection involved in the proposed architecture.

The system uses the pre-processing algorithm that is described in detail in [4]. The system proposed in this dissertation is meant for mobile devices and has algorithmic changes and consequent internal architecture changes that rely on that fact, and the fact that a different technique called Orthogonal Line Ordinal Features (OLOF), proposed in [41], was used for the template creation. The resulting templates are better adapted to the mobile domain and come already in a binary format, removing the need for the Feature Vector Binarization module.

Another novelty about this system, besides the different feature extraction algorithm, is the pre-segmentation module that runs as part of the Android application in the client side. This module's goal is to allow the application to provide feedback to the user of what it perceives to be the hand to be acquired. In this way, only good candidate images for the authentication are sent to the server, consequently reducing the number of failed tries of authentication for a genuine user, and reducing the data volume sent to the server. If less energy is necessary for the pre-segmentation than for the authentication messages that would otherwise need to be sent to the server, then this also results in a longer lifespan of the device's battery charge. Also of importance is that, since the segmentation is still performed at the server later on, the pre-segmentation performed by the device can be easily adapted to fit its computational power and resources.

For the mobile phones scenario, the quality of the captured images is often limited and highly noisy for some usage environments, resulting in very inconstant and unpredictable backgrounds, high illumination differences, and other issues that may affect the system. For these reasons, and in order to simplify the issues to be dealt with, the initial version of the proposed system is unimodal and solely makes use of the palmprint (see Figure 21 and Figure 22).

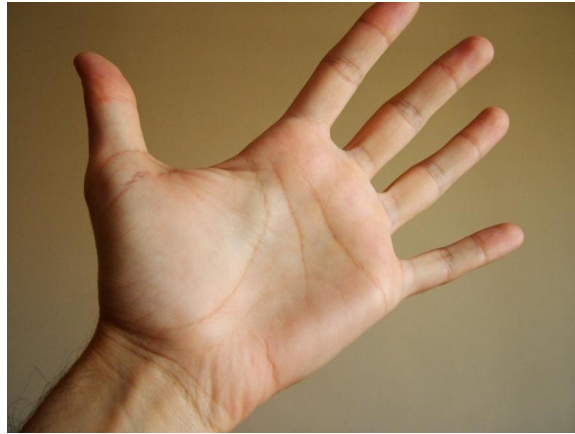


Figure 21 – Human palm.

Source: http://www.pollsb.com/photos/o/325808-hand_palm.jpg

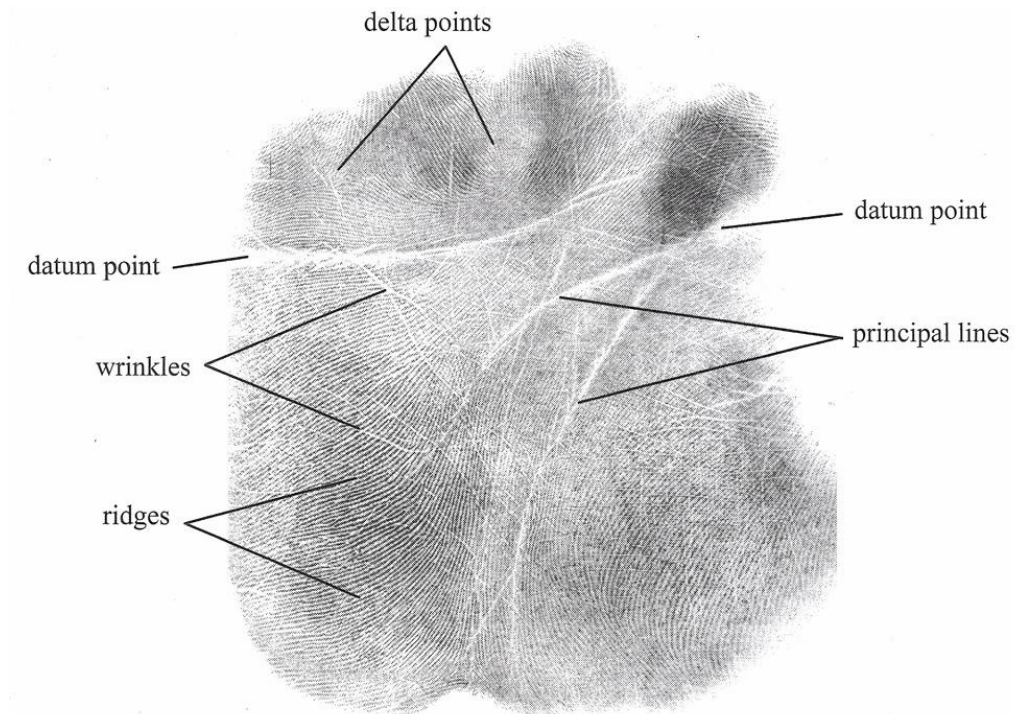


Figure 22 – Palmprint features [42].

The palmprint is feature-rich (see Figure 22) and has got a set of qualities that make it preferential for the developed system and in particular, for mobile device environments:

- Fairly large area, which means more features can be acquired.
- Stable throughout life, and less likely to be damaged than a fingerprint [42].
- Low resolution images still allow achieving a good system performance with less resources usage, and reducing the noise introduced at data acquisition by small amounts of dirt or grease [4].
- More difficult to trick than hand or finger geometry, according to A. Kumar et al. in [43].

Furthermore, the fact that the base system used for this dissertation's developments achieves good recognition results with a ROI of 16×16 pixels, makes it an ideal starting point for the mobile phones scenario, as stated earlier in this dissertation.

3.3.1. Data acquisition and Pre-segmentation

The data acquisition is performed by the mobile device, using the Android Client application that was developed.

The application provides a palmprint capture screen (see Figure 23), where the user's palmprint is acquired and then transferred to the server over a secure connection.

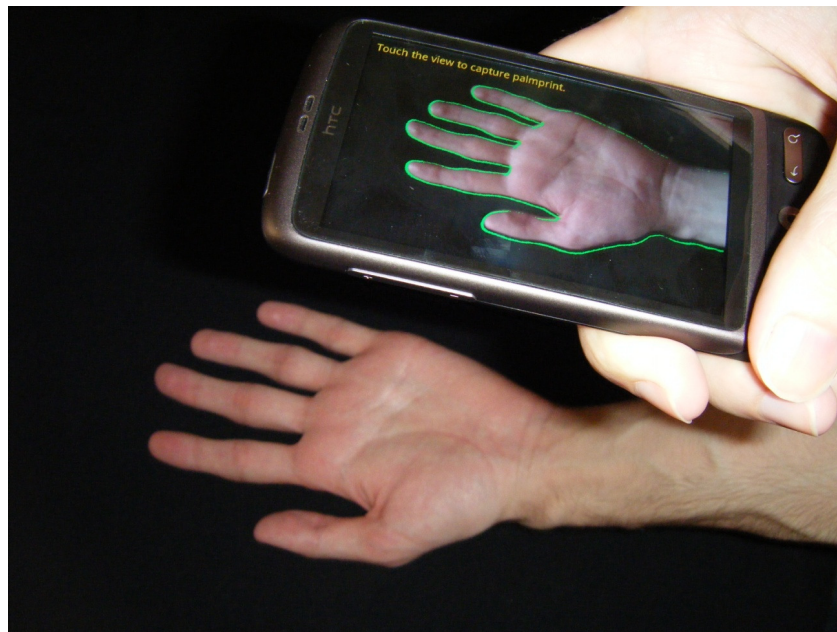


Figure 23 – Developed system's palmprint capture and pre-segmentation screen.

In the specific case for this work, the HTC Desire mobile phone was used, so the image is acquired through its 5 MP camera, and converted internally to the JPEG format, using a factor of quality 100. The autofocus and Light-Emitting Diode (LED) flash also automatically adjust to improve the photo result, as they would in a casual photo scenario.

This client guides the user through the enrolment and verification processes as well as the management of secure items. The application itself will be further explained in chapter 5 Developed Android Application, in terms of graphical user interface and functionality.

In order to give the user an idea of what the application perceives to be the user's hand, a pre-segmentation is performed in the device, in the same screen, as shown in Figure 23. This pre-segmentation can make use of simple algorithms, adaptable to the device's processing complexity, and its purpose is to provide some feedback from the application towards to user in order to prevent the task of messaging the server with an input image that could initially be considered ineffective for authentication.

The pre-segmentation performed in the device, is independent from the segmentation performed on the server, which is where complex algorithms can be used and where their results will have an impact on the obtained data and its accuracy.

Upon entering this window, the images continuously captured by the camera (frames) are processed by an algorithm adequate to the used device. For the HTC Desire device which was used, the proposed algorithm consists of performing a basic contour detection based on a Sobel filter.

The orientation of the application window for this screen was set as landscape, which means that this activity works in a horizontal mode and that any output messages are displayed in a horizontal mode, unlike for the rest of the screens of the developed Android application. This had to be developed this way because Android has had multiple different ways of changing camera orientation through the various API versions, as seen in [44] and specific devices can impose limitations to this based on their version of the Android OS source code [45]. Although the official Android documentation does not make it explicit that video objects will only display the video and coordinates system in an appropriate way, this issue has been mentioned in websites

such as Adobe in [46] where it is said: “On devices that can change the screen orientation, such as mobile phones, a Video object attached to the camera will only show upright video in a landscape-aspect orientation. Thus, mobile apps should use a landscape orientation when displaying video and should not auto-rotate”. Even though this sentence was mentioned in the Adobe Integrated Runtime (Adobe AIR) and Action Script 3.0 domain, the experimental code explorations performed proved that this is correct even for plain Java Android applications running in the Android 2.1 platform.

Another factor that improves the quality of the image frames being captured by the camera, and consequently the quality of the pre-segmentation and feedback of the data acquisition to the user is the lighting. For this reason, when there is insufficient environment light, the usage of the mobile phone’s LED light should be turned ON while the camera resource is on use by the application. However, the Android OS source code from specific devices also imposes limitations to this, which can only be surpassed by changing the OS source code. Also important of notice is the fact that the `getSupportedFlashModes()` from the camera API relies on the OS version, and does not always return values actually supported, on all devices. Equally relevant is the fact that for different devices, the same modes can work in different ways, as discussed by users in various forums such as in [47].

For the HTC Desire device which was used in this dissertation, the `getSupportedFlashModes()` method provides the correct supported flash modes, which are “on”, “off” and “auto”. In “on” mode, the LED is only activated for a brief instant, when taking a snapshot, and this behavior cannot be altered. In order to keep it constantly ON while the camera resource is on use, the “torch” mode would be necessary. For this reason, even when using the LED for the image acquisition, this resource cannot be used to improve the feedback from the application to the user during the data acquisition’s pre-segmentation. However, this does not limit the usability and utility of this resource, because it can be used to improve the image quality of the actual captured images which are in turn sent to the server for the authentication goal.

The Camera API for Android has also proved to have some issues as of API 2.1. For example, the original Camera app that comes with the device’s Android OS is capable of saving images with some additional information, such as the International Standards Organization (ISO) speed and camera model, and a similar way of saving an image

could not be achieved by an application using the Camera API, even upon inspection of the code used for the default Camera application (available in [48]), because for its parameters, it makes use of native routines which are linked to the Android OS. Also, the default Camera application that comes with the device provides interfaces that only allow to launch it and use photos acquired by it, but not to check which parameters it uses nor copy them so that they can be used directly in another application from the mobile phone. Moreover, since the Android OS parameters and the Camera API seem to be partially connected in ways that sometimes go beyond what is considered by the API, sometimes changing the camera parameters leads to unexpected results such as for example the distortion of the acquired images [49], or other types of unexpected behavior. The behavior of the default Android Camera application and API also depends on the device and its Android OS specific tweaks for other types of simple tasks, such as the actions taken when a photo is acquired. In some devices, it is added to the phone's photo gallery while in others it is not [50], and there is no way to change this at the applications layer, because the root of the behavior is in the source code of the system itself.

Another important and problematic issue related with the Android Camera code is the fact that the native Android code inherent to the Android 2.1 OS version comes with a memory management inefficiency, as described in [51]. This inefficiency is due to the fact that a new memory buffer is allocated each time a new frame is captured by the camera. This results in huge chunks of memory being freed and allocated, which consequently forces the Java garbage collector to act very often. The Java garbage's collector slows down applications when it executes and within the Android platform the slowdowns are even worse due to the platform's architecture and specificities, and makes applications stop responding periodically in this case. In [51], a solution for this problem is presented, which consists of recompiling the OS's source code by performing a change in it. The advantage of this solution is that it works for any Android version, including not only 2.1 but also previous versions of Android. The problem is that it requires the customization of the OS, and the impact of the changes will not affect users that have the default versions of the OS.

In the Android Google Code Group an issue was open for this buffer allocation problem (Issue 2794) in [52]. Apparently the problem is fixed in version 2.2, and there is a way to access the new 2.2 APIs in 2.1 but which was not intended for use, since it requires

the usage of code reflection (this is, code that browses over other code at runtime and executes it) and in some cases might make the device slower or lead to unexpected behaviors. This alternative way is described in [53].

In Android (and specifically in Android 2.1), the correct way to perform drawing over the camera's preview image is to have a View object over the SurfaceView object where the camera preview is shown. The View object to draw on has access to the data perceived by the camera in each frame, and can draw appropriately over the camera's preview. However, this means that the internally stored frame image must be replaced by a new one with each frame, which results in a cycle of a huge memory allocation followed by memory freed. Since Android's Java garbage collector stops the application briefly, every time it performs actions, this means that the frames processing will inevitably make the application "gulp" between frames, even though this does not affect the overall functionality. As previously mentioned, this problem cannot be solved at the application layer because its root is the source code for the camera hardware from the OS itself. Hopefully in future versions of Android, this issue will be addressed, providing much more effective processing related with camera frames and greatly improving user experience for applications like the one developed.

3.3.2. Communication Interfaces and Protocol

The communication between the client and the server is done using TCP/IP.

In order for this communication to be possible, two communication interfaces were developed for each of the communicating applications (the client, and the server). The client sends request messages to the server, who answers with a message type specific to the request sent (see Figure 24).

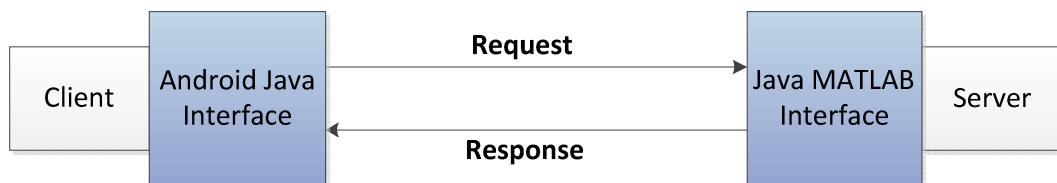


Figure 24 – Communication interfaces and type.

The need for an interface is necessary not only to define and enforce specific message types and sequences, but also because the data transferred individually must be read and interpreted in the same way, common to both parties involved. In this case, Java objects and primitive types are used in the communication, and both interfaces developed make

use of the java.net package for a common starting point for the communication, since those resources are available and compatible in both Java from Android, and plain Java available from within the MATLAB environment. The request messages consist of an integer with the message type code, an integer with the length of the data portion of the message, and the data itself, as shown in Figure 25 below. In Java, each integer uses 32 bits.

Message Type	Data length	Data
--------------	-------------	------

Figure 25 – Request message structure.

For each request type, a response message is replied by the server, with relevant data for the request type sent. In Table 7 below, the different message types is shown.

Message code	Message Type
1	Enrollment Request
2	Authentication Request
3	Delete Request (Remove user registration)

Table 7 – Message types and codes.

For each request, the integer with the size of the acquired palmprint image, and the image itself, are sent to the server, in order to authenticate the user individually for the operation being performed. The response includes an error identifier (ID) which has a value and meaning as shown in Table 8:

Error ID	Meaning
-2	Authentication Failure (but no problems with the input image)
-1	Generic program error
0	No error (Success)
1	Image size is too small
2	No objects found in image
3	Arm region is not contiguous
4	Arm region too big
5	Invalid peaks
6	Invalid valleys
7	Finger points not in correct order
8	Fingers too close together
9	Finger set does not correspond to left nor right hand

Table 8 – Possible error IDs and their meanings.

Any error ID received by the client will trigger an appropriate informative message to the user, informing about the success of the operation (if error ID = 0), or explaining the problem with the acquired image, in an attempt to help the user correct it in the next log-in attempt. In the used convention, negative error IDs are related with top-level errors, while positive error IDs are related with detailed image processing errors.

In the event the communication itself fails and the server cannot be reached, the application provides the appropriate warnings to the user.

The fact that Java streams are used as part of the implementation of the communication channel allows for the use of ciphered streams and/or the SSL protocol as means of making the data transferred private and secure.

3.3.3. Pre-processing

After the image data has been acquired and sent to the authentication server, it is pre-processed in the same way as in the initial system developed by M. Ramalho in [4], in order to achieve the segmented hand from the background along with the **hand contour** and **reference points** that will be used in the following processing stage (see Figure 26).

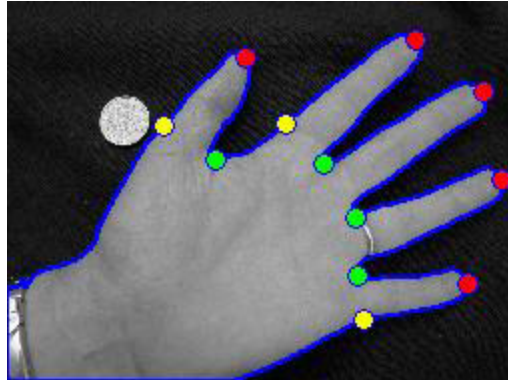


Figure 26 – Hand contour and reference points. Image was taken from [34].

The pre-processing stage consists of **image adjustment**, **segmentation**, **hand contour tracing**, and **reference points** extraction.

These steps are illustrated and summarized in Figure 27, shown below.

Hand-Based Biometric Recognition System for Mobile Devices

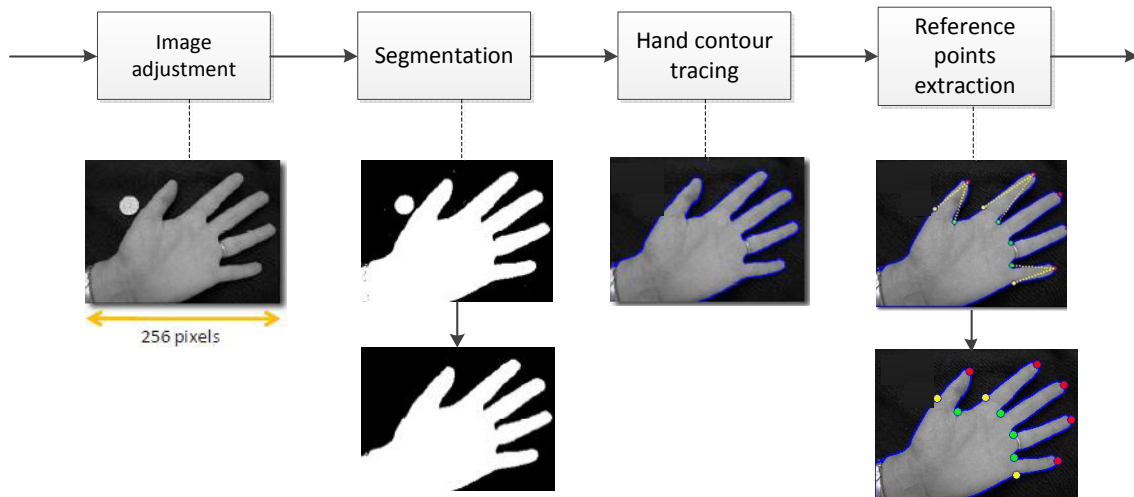


Figure 27 – Image preprocessing phases. (Images adapted from [4]).

In the image adjustment step the image is converted to grayscale, resized so that no dimension exceeds 256 pixels, and filtered for the noise, smoothness of small variance areas and preservation of the edges.

After the previous step is complete, the image is segmented into foreground and background, and the Otsu's method [54] is applied, choosing a threshold value that minimizes the intra-class variance of the output binary image. The hand region is selected using this automatic global histogram thresholding technique, and the background objects are eliminated through an algorithm based on morphological reconstruction [55], leaving a single big-length object in the image, which is considered to be the hand.

The next step is the hand contour tracing, which is performed using a popular algorithm [56]. A random starting point is chosen in the input image's hand boundary and all subsequent adjoining boundary points are searched for in a clockwise or counter-clockwise direction.

Finally, the reference points are obtained through the usage of two combined techniques which identify the fingertips and finger-webs (tissue between fingers):

- **Radial distance to a fixed point** [57], [58] – Computes the Euclidean distance between the hand contour pixels and the fixed middle point of the region where the wrist crosses the image's edge.

- **Countour curvegram** [57] – Analyzes the profile of the contour’s curvature (contour curvegram), which can be constructed using the Difference-of-slopes technique [58].

Additional reference points are obtained by discovering for the thumb, index and pinkie fingers, which contour point has an Euclidean distance to the fingertip equal to the Euclidean distance between the fingertip and the finger-web.

The region of interest will be identified in the next module, based on the reference points extracted.

3.3.4. Feature Extraction

In the feature extraction module, the region of interest is identified in the pre-processed image through the previously obtained reference points, by drawing a line segment between the finger-web and the additional reference points in the pinkie and index fingers (see Figure 28). The middle points of those line segments are used as vertices for the definition of the square that marks the ROI.

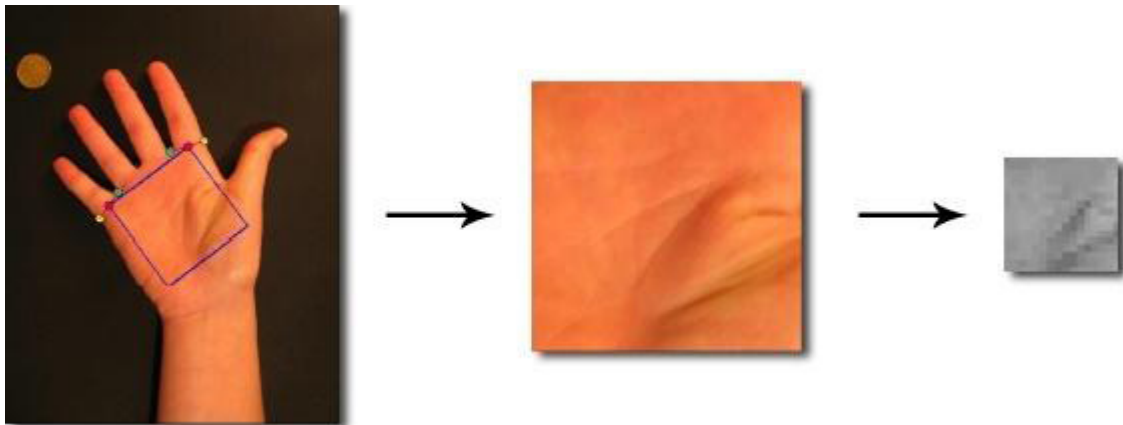


Figure 28 – Identification and normalization of the ROI [4].

As seen in Figure 28, the ROI square then needs to be rotated to a vertical position, resized, and converted to grayscale so that extracted features can be accurately compared to other samples.

The supported ROI size is 128×128 . Although the system proposed in [4] is capable of handling smaller sizes, down to 16×16 meaning smaller computation effort and there are results in [34] that claim the system performs better at 16×16 , the change to a mobile phones scenario makes need for more accuracy, given the reduction in environment constrains at the moment of image acquisition. For these reasons, the

128×128 size is used along with a different technique called Orthogonal Line Ordinal Features suggested in [41], instead of the Principal Component Analysis or Linear Discriminant Analysis used in the non-mobile existing system in [4].

While PCA and LDA are statistical analysis methods, OLOF performs the template generation based on ordinal features, which consist on a set of characteristics from the palm. OLOF compares two elongated line-like image regions orthogonal in orientation and generates a bit feature code which is concatenated with the code of the previous region. A palmprint pattern is represented by a set of thousands of ordinal feature codes relative to the multiple regions considered.

OLOF consists of three Gaussian filters which can be described by the following equations:

$$OF(\theta) = f(x, y, \theta) - f(x, y, \theta + \pi/2) \quad (1)$$

$$f(x, y, \theta) = \exp \left[- \left(\frac{(x-x_0)\cos\theta + (y-y_0)\sin\theta}{\delta_x} \right)^2 - \left(\frac{-(x-x_0)\sin\theta + (y-y_0)\cos\theta}{\delta_y} \right)^2 \right] \quad (2)$$

where, for each 2D Gaussian filter, θ is the orientation, δ_x and δ_y are the horizontal and vertical scales respectively. For each pixel in the palmprint's ROI, filtering is performed with three orientations, $OF(0)$, $OF(\pi/6)$, $OF(\pi/3)$, to obtain three bit sets of ordinal codes, based on the sign of the filtering results. The filter parameters are shown in Table 9.

	Filter Values
Filter Size (pixels)	35x35
Centre (x_0, y_0)	(17, 17)
Horizontal Scale (δ_x)	9
Vertical Scale (δ_y)	3

Table 9 – OLOF filter parameters.

In order to reduce the complexity of the decoding process in the LDPC ECC block, the filtered 128×128 images relative to each filter are reduced to a 32×32 image. At the end of the process, the results from the three filters are concatenated resulting in a binary template of size 3×32×32.

PCA and LDA make use of an average palmprint in order to project each user's palmprint in a template globally efficient for the system, but which becomes less

efficient as new palmprints are added, and makes the update of the average hand a computationally intensive process since it will require the recomputation of all the generated templates. This recomputation might not even be possible in secure systems since they don't save the initial feature vector that was used as one of PCA/LDA's inputs. The fact that the proposed system uses OLOF eliminates this problem and makes each template independent from the system's average palm for a given time, while at the same time benefiting from the number of users registered in the database.

The normalized ROI is turned into a vector of luminance values, which is used as the input for OLOF. OLOF codes the ordinal features into a **binary feature vector** or **template**, which can be stored on the templates database of a non-secure system, or, in this case, be applied an Error Correcting Code (ECC) and be discarded for security reasons, after its hash has been stored along with the parity-check bits.

3.3.5. Error Correcting Code (ECC)

Error Correcting Codes are often used in network communications, where messages transferred between different parties suffer errors through the used transmission channels. In those scenarios it is often necessary to provide the possibility for the involved parties to perform the correction of the messages received in order to prevent retransmissions that might overload the network or to make the received data as useable as possible in the least amount of time.

The message recovery mechanism consists of adding some redundant information to the original message, in order to allow its correction on the receiver side, as long as the corrupted information is smaller than the redundancy can correct for. This process is called Forward Error Correction (FEC) and is illustrated in Figure 29.

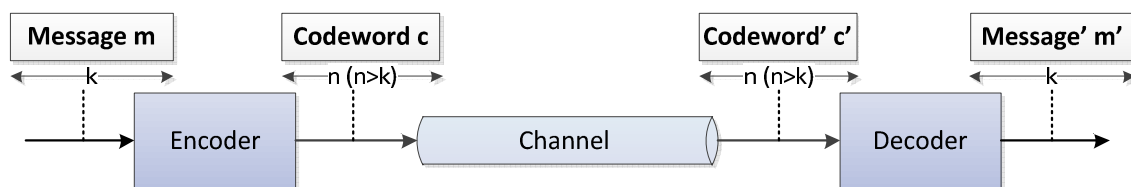


Figure 29 – Block diagram of a typical ECC scenario.

In systematic ECCs, a k -bit message m is encoded to a codeword c of n bits where $n > k$. The codeword c is composed by the original message and $n - k$ parity-check bits that will allow for the message recovery attempt on the receiving end of the

connection, by the decoder. Due to noise in the channel and transmission errors, it is possible that the codeword c suffers some changes and c' is received. The goal is that the received c' can be corrected and decoded so that $m' = m$.

In biometric systems, the channel block represents the acquisition noise that will change the data perceived by the data acquisition module. For the biometric reality, the scenario is better described as shown in Figure 30.

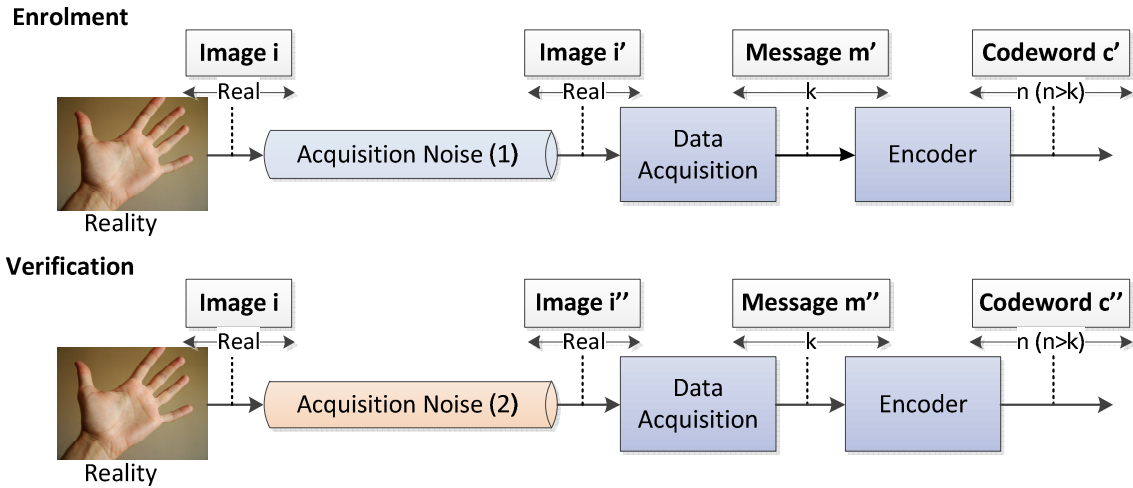


Figure 30 – Block diagram of ECC contextualization in biometrics scenario.

An image i is presented to the system, but due to the acquisition noise, which is different for each acquisition attempt, a different image i' is perceived by the system. This means that in one attempt, a message m' will be generated by the data acquisition module, resulting in a codeword c' of length $n = p + k$ where p is the number of parity-check bits generated for the message. In a second attempt, the perceived image will be different, resulting in a different message and a different codeword c'' , which will have to be corrected to c' in order for the user to be recognized. This correction must not be too powerful that a different impostor user with a different initial image i can be corrected into having the genuine user's palmprint template.

The code used by the ECC module is the **Low-Density Parity-Check (LDPC)** code.

LDPC codes (also known as Gallager codes) are characterized by a binary sparse parity-check matrix H whose rows represent parity-checks on the code's codewords. A $m \times n$ parity-check matrix has m parity-check equations that can involve n codeword bits.

In Figure 31, an example of a regular LDPC code characterized by 10 parity equations and 3 ones per column is illustrated.

$$H = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & | & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & | & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & | & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & | & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & | & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & | & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & | & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & | & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & | & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & | & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 31 – Example H matrix of a LDPC code [4].

To the left of the dashed line are the message bits, and to the right are the parity-check bits. For explanation purposes, those concepts are integrated with a non-sparse smaller matrix illustrated in Figure 32.

$$H = \begin{array}{cc} \text{Information} & \text{Protection} \\ \left[\begin{array}{cccc|ccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array} \right] \end{array}$$

Figure 32 – Interpretation of non-sparse matrix H [4] (Adapted).

A codeword of 7 bits is generated based on a message of 4, satisfying the equations of the matrix's rows. That is, the sum of the bits in the codeword for each position that has got a 1 in the matrix, must be 0. This is the principle used for the generation of the parity-check bits, and for the usage of the parity-check bits in the correction.

For a codeword $c = [c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6 \ c_7]$ and the matrix illustrated in Figure 32, the resulting parity-check equations are:

$$c.H^T = 0 \Leftrightarrow \begin{cases} c_1 \oplus c_2 \oplus c_3 \oplus c_5 = 0 \\ c_1 \oplus c_2 \oplus c_4 \oplus c_6 = 0 \\ c_1 \oplus c_3 \oplus c_4 \oplus c_7 = 0 \end{cases} \Leftrightarrow \begin{cases} c_1 \oplus c_2 \oplus c_3 = c_5 \\ c_1 \oplus c_2 \oplus c_4 = c_6 \\ c_1 \oplus c_3 \oplus c_4 = c_7 \end{cases} \quad (3)$$

A LDPC code was used for the ECC module because it is presented in the initial system proposed in [4] and LDPC possesses a set of properties that make it equally interesting in a mobile domain, namely the high granularity that can be achieved by varying the number of parity bits used (see Figure 33). This way, the correction power can be adjusted according to the desired correction performance for specific applications and contexts.

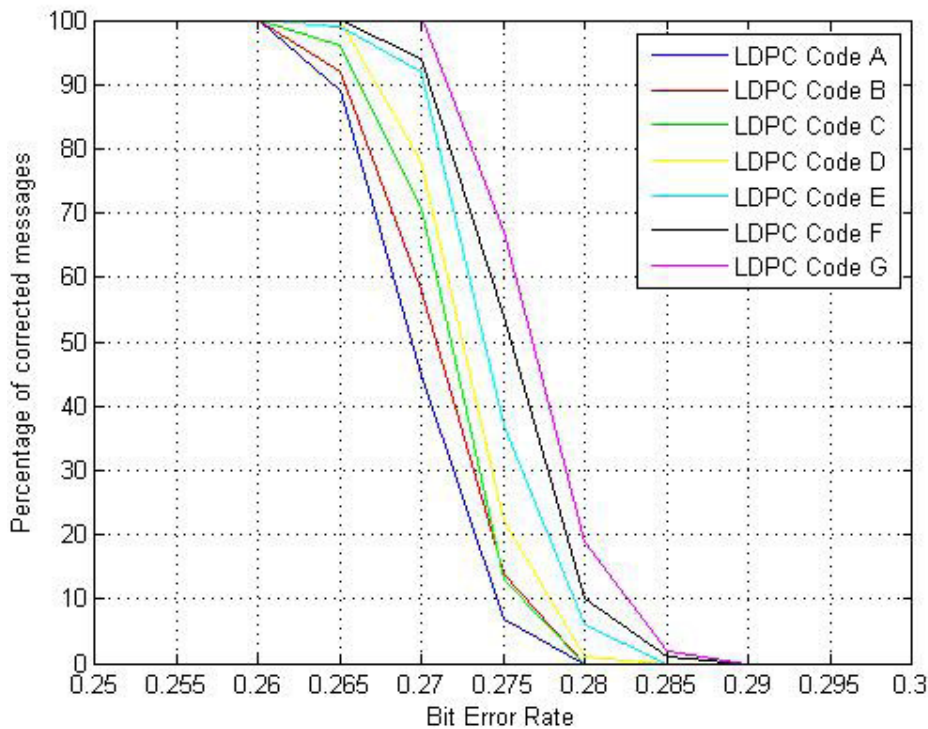


Figure 33 – Behavior of 7 LDPC codes when correcting 8128-bit messages with bit error rates of around 27% [4].

In biometric systems the goal of the LDPC code is to correct binary strings enough that the acquisition noise of a user is eliminated, allowing the authentication to the system, but not enough binary strings that an impostor’s template is “corrected” into the genuine user’s template.

3.3.6. Hash Function

Hash functions are irreversible functions that produce a fixed-length summary (also known as digest) output, based on a given input. With additional requirements, they can

be used to protect integrity of information or to provide digital signatures and certificates.

A given hash function H must satisfy the following conditions:

- H 's algorithm must be publicly known and not require additional input data than the one whose digest is sought.
- $H(x)$ must be easy to compute.
- The function must be one-way, meaning that given a hash function H and an output y , it is difficult to find a x that satisfies $H(x) = y$, and given x and $H(x)$ it is difficult to find a message $x' \neq x$ that satisfies $H(x') = H(x)$.

As previously mentioned, one-way hash functions are widely used in security for message digests and digital signatures. Some of the most commonly used hash functions are from the Message Digest (MD) family and Secure Hash Algorithm (SHA) family, namely MD2, MD5, SHA-256, SHA-384 and SHA-512 [59]. With the advances in processors and computational power, new more sophisticated hash functions have started to emerge, that seek to differentiate themselves from the others and broaden the horizons of cryptography, such as the RadioGatún hash function suggested in [60] and [61].

For the hash module, the SHA-512 hash function was chosen for its high security characteristics and for producing the largest output (512 bits) in its family.

4. Results

This chapter discusses the results of the tests performed with the proposed biometric recognition system. The system's performance was tested in terms of commonly used and widely accepted biometric performance measures, as well as in terms of computational efficiency and the tradeoff between both.

4.1. Test Conditions

The proposed system was tested with both online and offline tests:

- **Online tests** – The online tests correspond to tests in the presence of a human user, using the mobile application like a user would in a real scenario. These tests were used to obtain results concerning the computational performance of the developed Android application and the communication interface between it and the server, as well as to assure the system works as a whole.
- **Offline tests** – The offline tests correspond to tests using a database of hand images. In this case, the tests were run only on server side, with the goal of testing the system's performance in terms of standard performance measures from the Biometrics domain, such as FAR, FRR and ROC.

Three hand databases were used for the performed offline tests: the UST Hand Image Database, available on request from the Hong Kong University of Science and Technology [62]; the GPDS hand database available on request from Grupo de Procesado Digital de Señales [63]; and a HTC database that was created as part of this dissertation. Each user's hand as a separate user. The databases' specifications are summarized in Table 10, along with other information relevant for the tests and obtainable results:

Database	UST	GPDS	HTC
Number of users	217	75	5
Images per user hand	10	10	10
Total images	4340	1500	100
Considered users	434	150	10
Acquisition Device	Olympus C-3020Z digital camera (labeled as C-3100Z in Japan)	Scanner	HTC Desire Camera

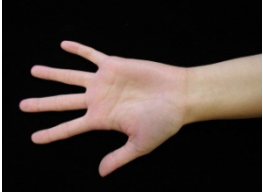


Image Resolution	1280×960	1021×1403	1552×2592
Example Image			
Training/Test images	2170	750	50
Train Binary Templates	434	150	10
Test Binary Templates	2170	750	50
Intra-class comparisons	2170	750	50
Inter-class comparisons	941780	112500	500

Table 10 – Databases' specifications and consequent binary templates and comparisons.

All the databases are divided into a training set and a test set. The training set is composed of 5 images of each user used for the enrolment, and the training set another 5, used for verification attempts. The system's performance is tested by comparing each verification binary template (that resulted from the test set) with the stored enrolment binary templates (that resulted from the training set) from the same user (intra-class variations) and from each of the other users (inter-class variations). One single binary template is created from each 5 training samples, so the number of train binary templates is the number of considered users.

After this process is repeated, comparing all the test binary templates with the training ones, the results of match or non-match can be interpreted as shown in Table 11.

	Match	Non-Match
Intra-class comparison	Correct Accept	False Reject
Inter-class comparison	False Accept	Correct Reject

Table 11 – Intra- and Inter-class comparisons interpretation.

The False Accept Rate and False Reject Rate can then be calculated as:

$$FAR = \frac{\text{Number of False Accepts}}{\text{Number of inter-class comparisons}} \times 100\% \quad (4)$$

$$FRR = \frac{\text{Number of False Rejects}}{\text{Number of intra-class comparisons}} \times 100\% \quad (5)$$

The online simulation was done with the HTC Desire connected to the server through a Local Area Network (LAN), using a wireless connection to connect to a Linksys WAG200G router.

The server side simulation environment was developed using Matlab® along with the toolboxes: Image Processing Toolbox™, Signal Processing Toolbox™, Communications Toolbox™ and Curve Fitting Toolbox™. The MATLAB server application was run in a computer with a Dual-Core 2.2GHz processor and 3 GB RAM.

4.2. Recognition Performance

The biometric recognition system's performance was evaluated through standard commonly used metrics, such as the FAR and FRR rates, using offline tests. These rates measure the number of impostor users who manage to successfully authenticate, and the number of genuine users who fail to authenticate to the system, respectively, consequently measuring the system's accuracy. Those are the two situations represented in Table 11, in the third row of the second column, and in the second row of the third column, respectively. The other situations expressed in the table represent cases in which the system performs correctly.

Additionally, the EER rate was used. This rate is the rate at which FAR and FRR are equal, and corresponds to a measure of the system's sensitivity balance. A low EER means that it is possible to get a good (small value) FAR and FRR simultaneously, which means that with a balanced approach, of no tradeoff between FAR and FRR, the system performs incorrectly in a small amount of cases. However, the EER is not necessarily the system's operating point, because depending on the application scenario, it might be reasonable to allow for higher FARs in order to get lower FRRs (such as in critical security systems, where impostor users are simply unacceptable), or decrease FARs at the cost of increased FRRs for practical and usability reasons (such as in a system for public transports, where the flow of people is more desirable than a crowd continuously attempting to get access).

The system's performance was tested for ROIs of 128×128 pixels, using the OLOF technique for the extraction of the ordinal features and creation of the templates. The genuine and impostor distributions obtained for the three tested databases are shown in Figure 34, along with the respective ROC curves. These distributions were obtained performing single comparisons between templates, without any shifts.

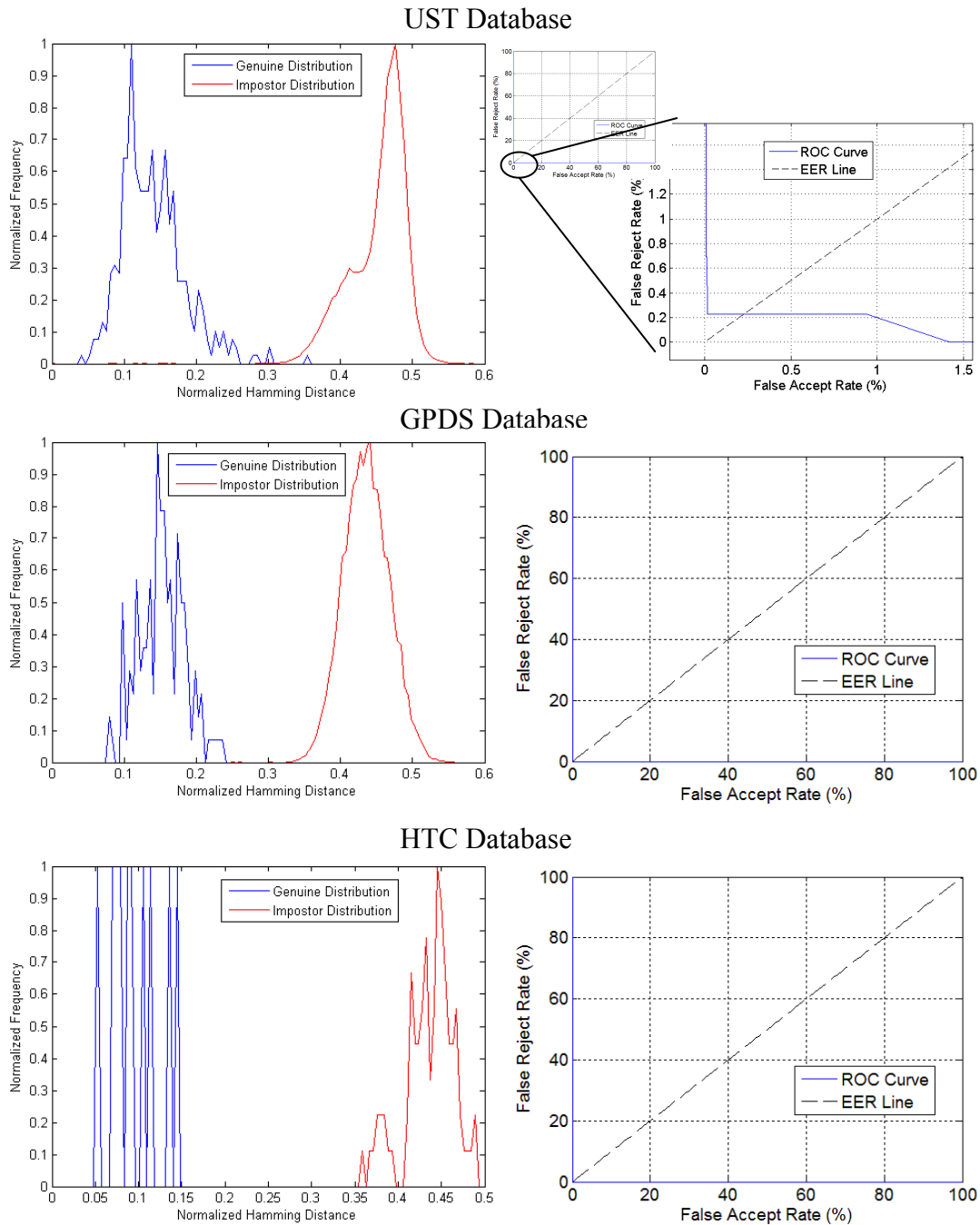


Figure 34 – Genuine and Impostor Distributions, and ROC curves for the three tested databases, using 128×128 templates and the OLOF technique, with no shifts considered during template matching.

The ROC curves are perfect corners for all the databases (which means $EER = 0$), with the exception of the UST database, in which there is a small slope resultant from the small interception present in its genuine and impostor distributions, as shown in its ROC graph zoom, where EER is roughly 0.21%. For the GPDS and HTC databases there is no overlapping of the distributions. The distributions are apart in all situations, and closer distributions were obtained for the databases with a higher number of users.

Although the HTC database has a small number of considered users (10), the distance between both the genuine and impostor distributions for this database is good and the highest among all the tested databases, which suggests the proposed system and the OLOF technique, are suited for biometric recognition with the mobile phone's camera.

The results show a good system biometric recognition performance for all the tested databases, but additional techniques can still be added and considered, such as shifting the templates in various directions as an attempt to improve matching results. This analysis makes sense because the databases used to obtain results contain a small subset of all the palmprints that exist. In a working system broadly adopted and vastly used, the amount of users processed can be bigger than the number of users considered in the used databases.

Following the interest to improve the results, the distributions were then recalculated along with the ROC curves, but this time considering both the original template position and 4 shifts (to the left, right, upwards and downwards) of the templates for the matching. The shift with the best score is the one considered, consequently improving the individual matching scores, since small differences natural to different acquisition samples are less likely to result in score differences in this situation. Since the shifts are considered when comparing an authenticating user's template with a stored template before it is possible to know if the user is a genuine or an impostor, it is probable that the system's performance results will be improved, but it is also possible that the shifts may downgrade the system's performance. The obtained results are shown in Figure 35.

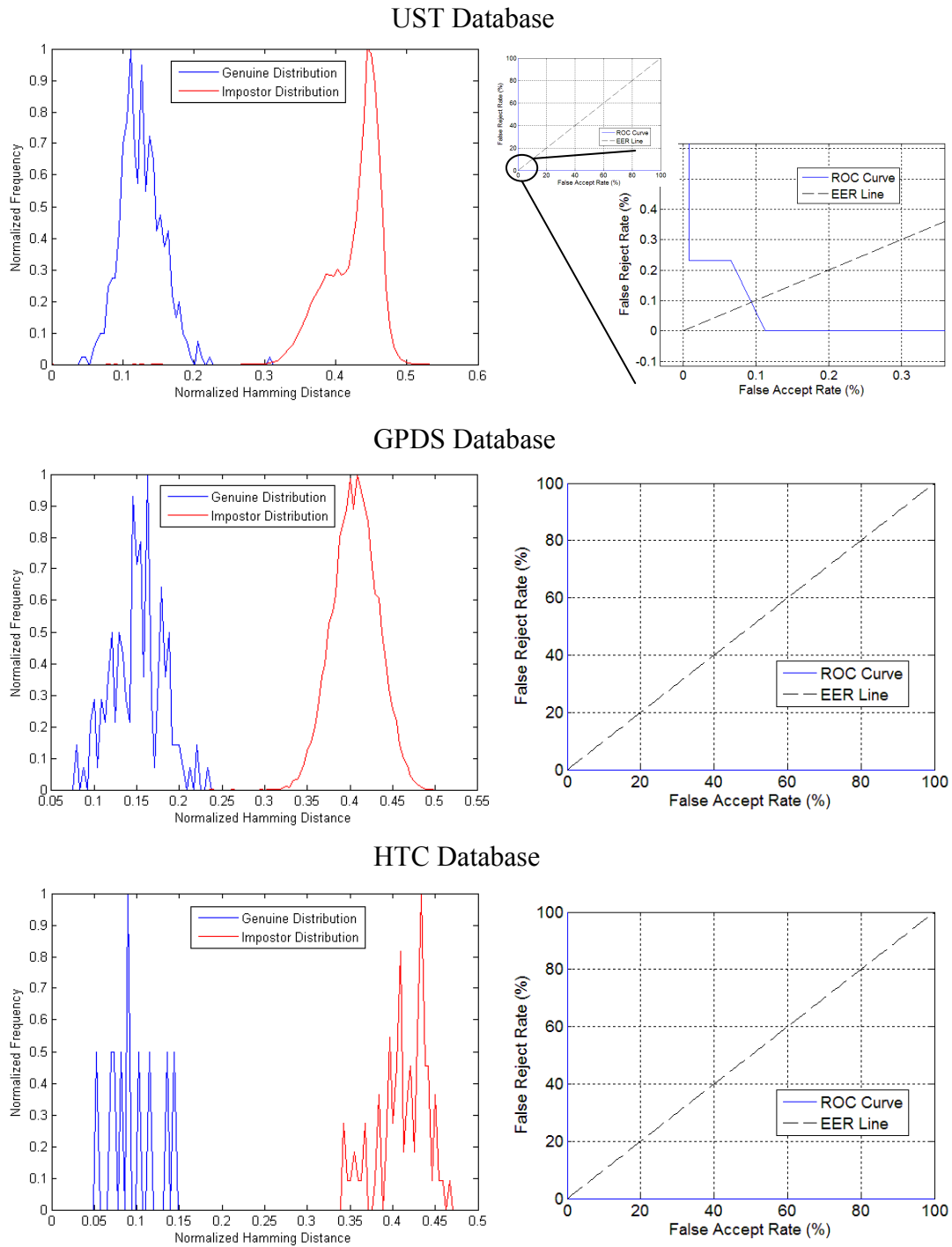


Figure 35 – Genuine and Impostor Distributions, and ROC curves for the three tested databases, using 128×128 templates and the OLOF technique, considering the original template position and 4 shifts during template matching.

Considering template shifts during the matching process is an attempt to improve the recognition performance not just for the considered databases, which were already providing good results, but for a generality of other databases and working systems. This would desirably result in genuine and impostor distributions further away from each other. However, this did not have a positive impact on all the tested databases. For

the UST Database, this results in an improvement of the ROC curve, since the overlapping between both distributions has been diminished, resulting in a thinner ROC slope and an EER of about 0.1%. For the GPDS and HTC databases, the shifts approach resulted in slightly closer distributions, but did not result in a downgrade big enough to change the ROC curves, which already were a perfect corner in the previous approach. Since an authenticating user's template is compared with the stored templates considering multiple shifts, it will improve the scores of genuine users, but also may improve the scores of impostor users.

It is possible to quantify the distance between the genuine and impostor distributions, by using the decidability index as suggested in [64]. This index is proportional to the distance between the genuine and impostor distributions, is independent from the actual threshold used by the system, and can be calculated as:

$$d' = \frac{|\mu_{gen} - \mu_{imp}|}{\sqrt{\frac{\sigma_{gen}^2 + \sigma_{imp}^2}{2}}} \quad (6)$$

Where μ_{gen} and μ_{imp} are the mean of the genuine and impostors distributions respectively, and σ_{gen} and σ_{imp} are the standard deviations of these distributions, in same order.

For both scenarios previously described, considering solely the original template orientation, and then considering also shifts in 4 directions, the obtained decidability indexes are presented in Table 12.

	UST	GPDS	HTC
Without shifts	7.6226	7.9245	11.8169
With 4 shifts	9.0013	7.8448	10.8896

Table 12 – Decidability indexes for the three tested databases, considering no shifts and considering 4 shifts in the template matching.

The results' changes provided by the shifts approach previously seen in the distribution graphs is confirmed for all the databases. For the UST database this results in a performance boost, which is translated into an increase of the index, while for the GPDS and HTC databases, the index suffers a slight decrease, which reflects the small performance downgrade this approach introduces.

4.3. Application Performance

The developed application should be tested in terms of processing time/speed, CPU usage, and memory usage, using online tests.

In Android 2.1, each Android Java application is limited to a budget of 24 MB of heap size in the DVM instance that executes it. This means that when dealing with resources such as images, which often need to be manipulated internally, for compression, uncompression and processing, the budget of 24 MB can easily become smaller than it is for other types of applications. If we add to that the memory used by the application itself and its other resources and objects, it becomes understandable that exceeding this budget is quite easy in signal processing applications.

Downsampling acquired and manipulated images is a practice often used in Android applications to deal with this issue, and should continue to be since newer devices are possessing better cameras with higher resolutions, which will generate higher amounts of information. For this reason, it is important to understand how does the downsampling of the acquired image affects the application performance according to computational standards (discussed in this section), and how it affects the biometric recognition performance (discussed in section 4.4 Tradeoff).

The sum of the native heap size (heap of the java virtual machine running the application) with the Dalvik heap size (heap of the java application) must never exceed the maximum heap budget of 24 MB, or the application will terminate with an out of memory exception.

The used heap sizes were calculated for the first activity of the application, in order to get an idea of the memory used by the visual and interactive components. These results are presented as an average of 10 executions in Table 13, and the obtained CPU usage relative to the components' initialization was 19%.

	Native Heap	Dalvik Heap	Total	Limit
Size	5480	3783	9263	24576
Allocated	5478	3149	8627	N/A
Free	1	634	635	N/A

Table 13 – Application memory usage in starting activity. The units are expressed in 1024 bytes. N/A: Does not apply.

There is a difference of 1 unit between the free native heap and the subtraction of the allocated heap to the native heap size, due to the implementation of this heap in the Android architecture. This difference may be of even greater differences for the native heap at calls in other activities or stages of the processing, and is verifiable through Linux kernel functions. The free heap size is not too important because when more memory is necessary than the available free memory in the heap, it expands. The most important is the 24 MB limit of the heap size, which must always be respected.

Regarding the image acquisition activity, the CPU usage took an average of 71%, and the respective memory information is shown in Table 14.

	Native Heap	Dalvik Heap	Total	Limit
Size	7752	9351	17103	24576
Allocated	6163	8621	14784	N/A
Free	84	730	814	N/A

Table 14 – Application memory usage in image acquisition activity. The units are expressed in 1024 bytes. N/A: Does not apply.

For the image acquisition activity, the memory and CPU usages are superior because the captured image frames are continuously processed to provide feedback to the user. The pre-segmentation of those frames is also suboptimal due to the limitations inherent to a correct usage of the Android APIs, as previously explained in section 3.3.1 Data acquisition and Pre-segmentation. As a consequence of the multiple memory allocations necessary between the pre-segmentation between processed camera frames explained in the same section, the time it takes for a pre-segmentation to complete is of 2.418 seconds. The actual pre-segmentation perceived by the user has a significant delay due to the high CPU usage and the stacking up of pre-segmented images which need to be drawn on the screen, as well as the refreshment of the application components involved.

The computational performance was then measured for the final stage of the data acquisition module, in which the actual photo is taken and sent to the server for processing and for a resulting response (error ID), indicating the success or error cause of the operation. For this analysis, the memory and CPU usages were obtained for the different considered sampling ratios. The 1:1 ratio was not considered in this phase because the application can only work with 1:2 and superior downsampling ratios in order to respect the 24 MB heap budget. The results were obtained for verification attempts, and are presented in Table 15.

Downsampling Ratios	1:2	1:3	1:4	1:5
CPU Usage	17%	27%	43%	21%
Native Heap Size	9152	9056	7728	8028
Dalvik Heap Size	10887	11591	9351	10887
Total Heap Size	20039	20647	17079	18915
Native Heap Allocated	7738	7652	5943	6173
Dalvik Heap Allocated	4971	6640	5436	5515
Total Heap Allocated	12709	14292	11379	11688
Native Heap Free	149	147	84	126
Dalvik Heap Free	5916	4951	3915	5372
Total Heap Free	6065	5098	3999	5498

Table 15 – CPU and memory usages for each of the considered downsampling ratios at data acquisition stage, when the hand image is captured and sent to the server. The units are expressed in 1024 bytes.

The CPU usage is smaller in this phase because no camera frames are continuously being pre-segmented. All the processor has to do is execute the code relative to establishing the connection, sending the image to the server, and reading an errorID code.

The results obtained for the CPU seem inconsistent, but this is explained because the CPU usage logging needs to be done through a separate thread, so that the CPU usage of the application while it is active is considered. According to the principles of most Operating Systems, this thread can execute and calculate the CPU usage of the application at different places of the actual program, generating different results, as previously shown.

The memory usage results are equally inconsistent. The Android platform relies on a Linux kernel, and in the principles that unused memory is wasted memory. For this reason, the heap sizes and allocation sizes do not necessarily refer to memory which is absolute necessary by the program, but may also include much memory which is already ready to be freed but is not, for optimization purposes. Equally important is the fact that since huge amounts of memory are being allocated and freed (specially for what concerns the image files manipulated within the program), this provides room for huge deviations in the result of the memory inspection commands, at this stage of the application. Although in Android the Java garbage collector can be called, it adds additional delay to the application and still does not assure that all memory that can be freed is actually released. For all those reasons, it is not possible to get accurate memory

results within the Android platform. The obtained results, suggest that the downsampling ratio of 1:4 brings considerable memory gains in comparison with the 1:2 and 1:3 ratios, although the accuracy of these results is limited by the factors previously described.

The processing times were then calculated for all the considered downsampling ratios. The results are shown in Table 16.

Client side (Device)	1:2	1:3	1:4	1:5
Connection establishment	11	15	24	17
Bitmap – Allocation	454	484	84	86
Bitmap – Send	914	1058	325	486
Bitmap – Total	1368	1542	409	572
Reading response	8524	8044	7583	7856
Time to Authenticate	9901	9593	7998	8433
Server side				
Process Image time	794	876	331	344
Process Request time	9830	9500	7982	8347

Table 16 – Processing times for different stages of the verification process, for the different considered downsampling ratios. The times are expressed in milliseconds.

For the processing times, the results also show some inconsistency, at least between the 1:2 and 1:3 sampling ratios, and between the 1:4 and 1:5 sampling ratios. One of the reasons for these discrepancies is the fact that the huge chunks of memory involved in the allocation and freeing of the memory, results in multiple calls to the garbage collector, which, as previously mentioned, does not act in a completely linear and predictable way. This means that random calls to the garbage collector happen, adding random delays to the processing times, at random places in the program, which are different for each execution. To make this worse, the fact that some stacks of the pre-segmentation images are still waiting to be processed after the pre-segmentation has been stopped, the memory associated with them can apply be freed at also random times throughout the execution of the communication phase (sending the image data, and getting the response). For these reasons, the processing times between the different downsampling ratios, and between different execution attempts for a given ratio may result in very different results. According to the obtained results, there seems to be an overall gain, especially in bitmap allocation and sending times, when using the 1:4 downsampling ratio, in comparison to 1:2 or 1:3.

The time to establish the connection is the smallest, followed by the time for the bitmap memory allocation time, and finally the time to send the bitmap. The time to read the response is the biggest on client side, since it requires not only the usage of the communication channel, but also the wait for the server to process the sent input images. The time to process the images seems roughly similar with the time necessary to send the bitmap, but there is no connection between both, since the time to send depends also on the network architecture and factors external to the system, in addition to the factors that influence both, such as the image size. In all the downsampling cases, the time to authenticate is bigger than the time the server takes to process the request, since the client needs to wait for the response to arrive and be processed. The time to process the image on server side consists of the preprocessing only. The time necessary to pre-process the images (Process Image time, in the table) is relatively small when compared to the OLOF template generation time, and the time necessary to look up the appropriate user data and perform the matching. It is that time that adds up to the image processing and results in the final process request time mentioned in the table.

The results obtained in this section, regarding the computational performance of the application, suggest that the downsampling ratio of 1:4 brings considerable gains over 1:2 and 1:3. Between 1:4 and 1:5, the difference is not as clear. These results should however be considered as rough estimations, due to the inherent uncertainty and randomness associated with the Android platform and many of the Operating Systems and Java Virtual Machines' logics and optimizations which results in nonlinear system execution trees for different execution attempts. This problem is not only from Android, but takes a huge impact in this platform due to its common roots with Linux.

4.4. Tradeoff

Previously, the system's recognition performance was tested, and its computational performance was measured in terms of speed, memory, and CPU, along with the impact of image downsampling in the computational measures. In this section, the impact of the downsampling in the recognition performance will be studied, with the goal of understanding the tradeoff between the computational efficiency and the recognition performance.

In order to achieve results regarding the impact of the downsampling of the hand images used for recognition, the considered downsampling intervals will be between 1:2

(0.50 reduction factor) and the highest value that does not generate recognition errors in the contours used for the palmprint recognition with the considered database's images.

For the UST and GPDS databases, a downsampling of 1:2 (0.50 reduction factor) results in errors in contour detection. For this reason, the downsampling was not possible for those databases.

For the HTC database, downsamplings of 1:2, 1:3, 1:4, and 1:5 were tested, considering no shifts for the resulting templates, to avoid the unpredictable gains in genuine and impostor scores previously observed. For a downsampling ratio of 1:6 or higher, no recognition is achieved due to errors in contour detection. The genuine and impostor distribution results are shown in Figure 36. Since there is no overlap between distributions in any situation, the ROC curves are a perfect corner for all considered downsampling ratios.

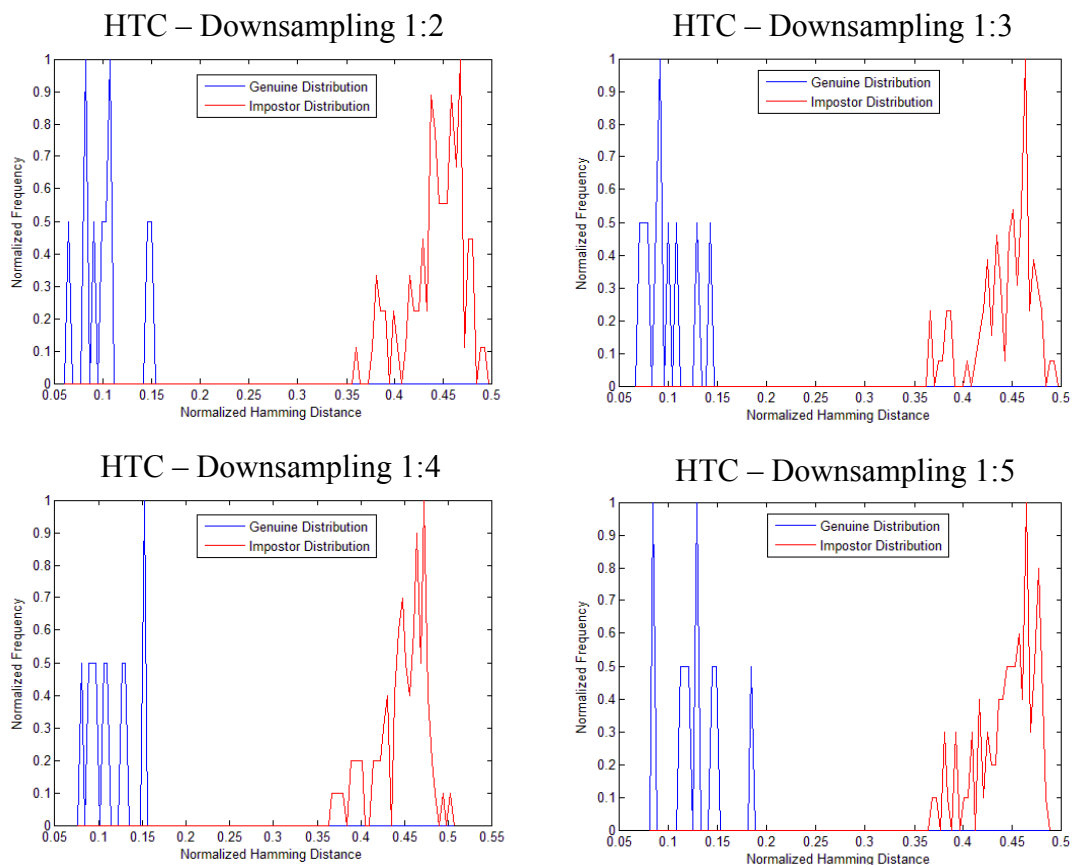


Figure 36 – Genuine and Impostor Distributions for the HTC database, using 128×128 templates with the OLOF technique, and considering downsampling factors of 1:2, 1:3, 1:4 and 1:5, without considering template shifts at matching stage.

As seen in the graphs from Figure 36, with the downsampling, the genuine and impostor distributions change their shape slightly, as well as the distances between them. In order

to better quantify and understand these distances, the decidability index d' was calculated for each downsampling case, and presented in Table 17, along with the scenario with no downsampling (1:1).

Downsampling Ratio	Hand Image Size	Decidability Index d'
1:1 = 1	1552×2592	11.8169
1:2 = 0.50	776×1296	12.0880
1:3 = 0.333...	518×864	12.7323
1:4 = 0.25	388×648	12.0359
1:5 = 0.2	311×519	10.7592

Table 17 – Decidability indexes and considered image sizes for the various downsampling ratios tested for the HTC database, considering no shifts in the template matching.

As seen in Table 17, the decidability indexes increase for downsampling ratios of 1:2 and 1:3, meaning that the genuine and impostor distributions get further apart, improving the system's recognition performance. This may happen due to the small size of the considered database. For a downsampling ratio of 1:4, the decidability index starts to decrease in comparison to the ratio of 1:3, but is still superior to the original ratio of 1:1. For a downsampling ratio of 1:5, the recognition performance starts to decrease a lot, as both distributions start to approach each other, due to lack of quality in the considered hand images. For ratios of 1:6 and superior, this decrease of quality is so intense that the image processing algorithms fail to process the input hand images correctly.

Downsampling the acquired images may become a problem in systems which work in identification mode, and with many users registered. For the proposed system, the verification should still work with recognition performance results using a downsampling ratio of 1:2, which results in an image of 776×1296, which is not too different from the 1280×960 images in the UST database or the 1021×1403 ones in the GPDS database. Since the same approach was taken in all databases, capturing images of the whole hand, it is fairly safe to assume that in a huge mobile phone images database, the downsampling ratio of 1:2 would not affect the system's recognition performance significantly.

In order to study the tradeoff between the system's recognition performance and computational performance, these results were compared with the computational measures taken in section 4.3 Application Performance, with the goal of understanding

which downsampling ratio should be used to get the best out of both performance domains, without neglecting either. The most relevant results from the different calculated performance measures are summarized and presented in Table 18. Since the achieved results possess considerable deviations, the resulting conclusions and observations should be taken with care.

Downsampling Ratio	1:2	1:3	1:4	1:5
Decidability Index d'	12.0880	12.7323	12.0359	10.7592
Time to Authenticate (ms)	9901	9593	7998	8433
Total Heap Allocated (KB)	12709	14292	11379	11688

Table 18 – Tradeoff between biometric recognition and computational performances. 1 KB = 1024 bytes.

Any of the downsampling ratios provide better decidability indexes than the original 1:1 ratio with a d' of 11.8169, except for the ratio of 1:5. The time necessary for authentication seems to get a substantial gain at the ratio of 1:4, and the total allocated heap seems to be smaller for this ratio as well, but with a smaller difference than the one obtained for the time to authenticate. Combining these results, the best downsampling ratios for the system to operate with are the 1:3 ratio, if privileging recognition performance over computational performance, or the 1:4 ratio, if giving priority to computational efficiency over recognition results. Ultimately, since worse recognition results may result in more attempts to log in, the bigger decidability index could be, in a way, considered the decisive factor, indicating in this case that the system should operate at the 1:3 downsampling ratio.

As explained throughout this analysis, the considered HTC database is fairly small, and the platform issues mentioned result in a reasonable degree of inaccuracy in the time and memory measurements taken. These conclusions should be handled with care, taking this into account.

5. Developed Android Application

5.1. Introduction to the Android OS

Android is a software stack for mobile devices that includes an operating system based on the 2.6 Linux kernel, middleware to provide additional facilities for other software, and a set of key applications such as email clients, calendar, Short Message Service (SMS) management, maps, browser, contacts and others.

In order to develop applications for the Android platform, it is necessary to use the Android Standard Development Kit (SDK), which provides a set of tools and APIs to develop Android Applications using the Java Programming Language.

This mobile platform offers the following features [31]:

- **Application framework** enabling reuse and replacement of components
- **Dalvik Virtual Machine (DVM)** optimized for mobile devices
- **Integrated browser** based on the open source WebKit engine
- **Optimized graphics** powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- **SQLite** for structured data storage
- **Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **Global System for Mobile Communications (GSM) Telephony** (hardware dependent)
- **Bluetooth, EDGE, 3G, and WiFi** (hardware dependent)
- **Camera, Global Positioning System (GPS), compass, and accelerometer** (hardware dependent)
- **Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE, which is recommended for development

The main difference between Android and other Mobile Platforms is the fact that it is Open Source, meaning that the operating system is available and can be edited, and it is also possible to create applications that use any of the available APIs, without the developer entities having to pay for those APIs or their usage.

With this open nature, this platform has become very attractive for the industry of both hardware manufacturers and software developers, both for software companies and individual software developers, and also both for commercial purposes and research or entertainment. Android is a project continuously evolving, supported by the Open Handset Alliance.

Android's Architecture is based on layers as seen in the following image:

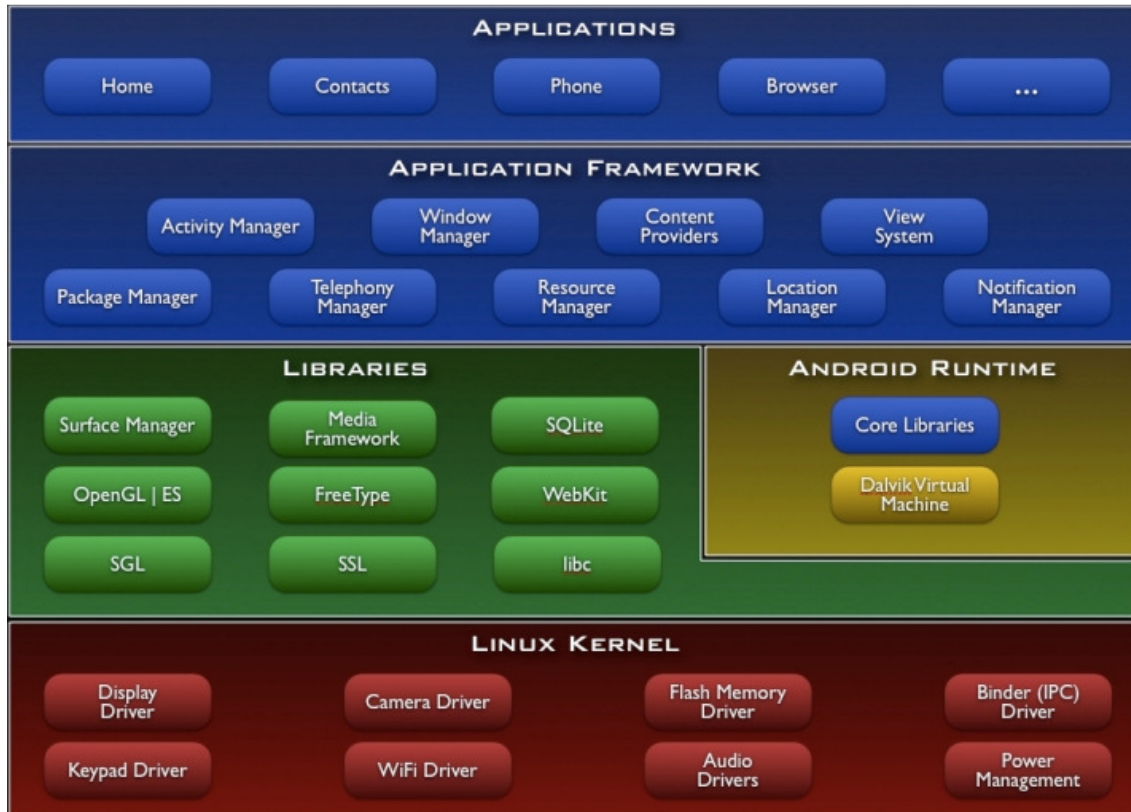


Figure 37 – Android software stack [31].

The **Linux Kernel** provides an abstraction layer between the hardware and the software stack and is used for core system services, the **libraries** are a set of C/C++ libraries used by the Android system, and exposed to developers through the **application framework**, which allows developers to use and reuse services and components, and to make new content also available to use by yet other applications.

In Android, the runtime of applications works in a similar way as the Java Virtual Machine (JVM) does in machines with platforms such as Linux, Windows or Mac OS. When an Android Application is launched, a new instance of Dalvik Virtual Machine (DVM) is started in a separate process, and executes the code related to that Android Application. Android uses the included “dx” tool to turn Java compiled class files into

Dalvik Executable (.dex) files, optimized for minimal memory footprint. The DVM is register-based and runs those files.

Android applications are written using the Java programming language. After the code is compiled along with other data and resources needed by the application, the Android Asset Packaging Tool (AAPT) is used to bundle it all into an Android package file (.apk). The code in an .apk file is one application, and is used to distribute and install the application in mobile devices.

Unlike typical Java applications, Android applications do not have a single entry point, because they reuse components from the Android library, which possess specific lifecycles. This becomes clear when looking at the lifecycle of one of the principal component of the Android architecture: the Activity class (see Figure 38).

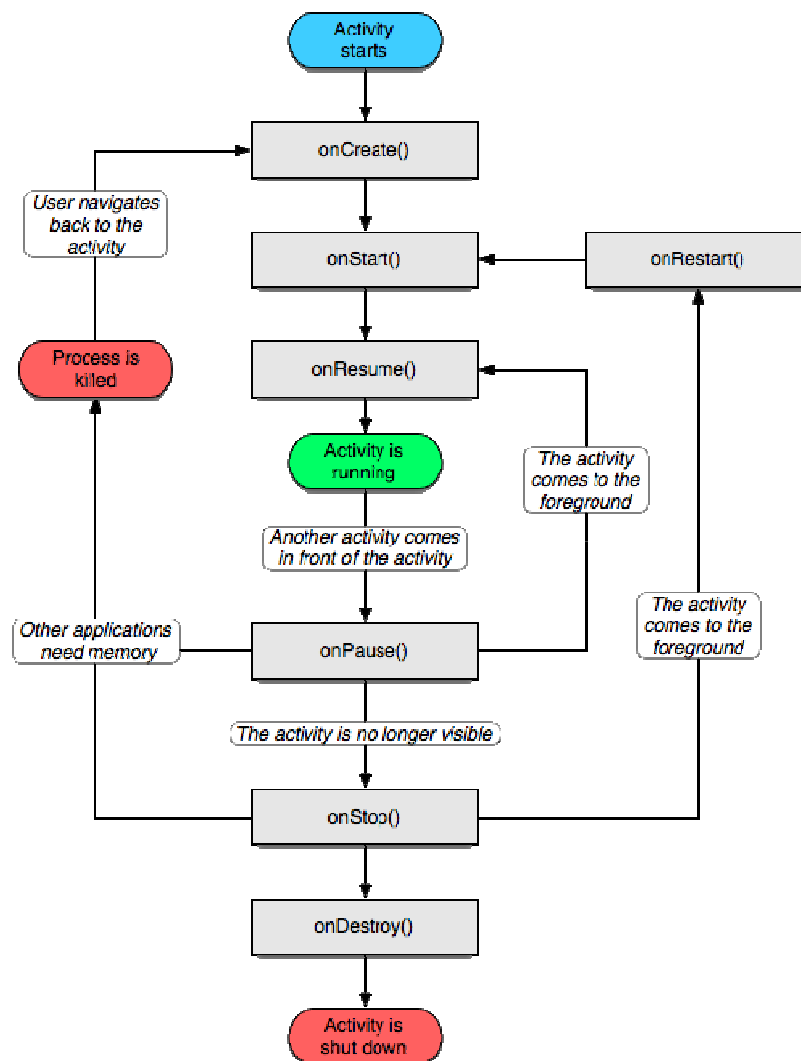


Figure 38 – Activity lifecycle [31].

One of the important tasks to perform in those methods is saving the activity's state, so that the appropriate actions are taken by the application in the event that OS calls are performed, either by the automatic choice of the Android OS, or by user actions, such as sending one activity to background, exiting an activity, or performing other actions that might relate with those methods, such as changing device orientation or opening a physical keyboard (in case the device has one).

In Android, applications may also use the XML language to define graphical user interfaces independently from the code. The use of C/C++ code libraries is equally possible through the NDK.

5.2. Developed Software Structure

The developed Android application targets the Android 2.1 version and is intended to work in all devices using that version and more recent versions of the Android OS. The software was tested using the HTC Desire device.

As previously mentioned in this work, the Android application works as the data acquisition and pre-segmentation modules of the system, but also provides functionality for the user. Below, in Figure 39 the possible user actions are presented in a use case Unified Modeling Language (UML) diagram.

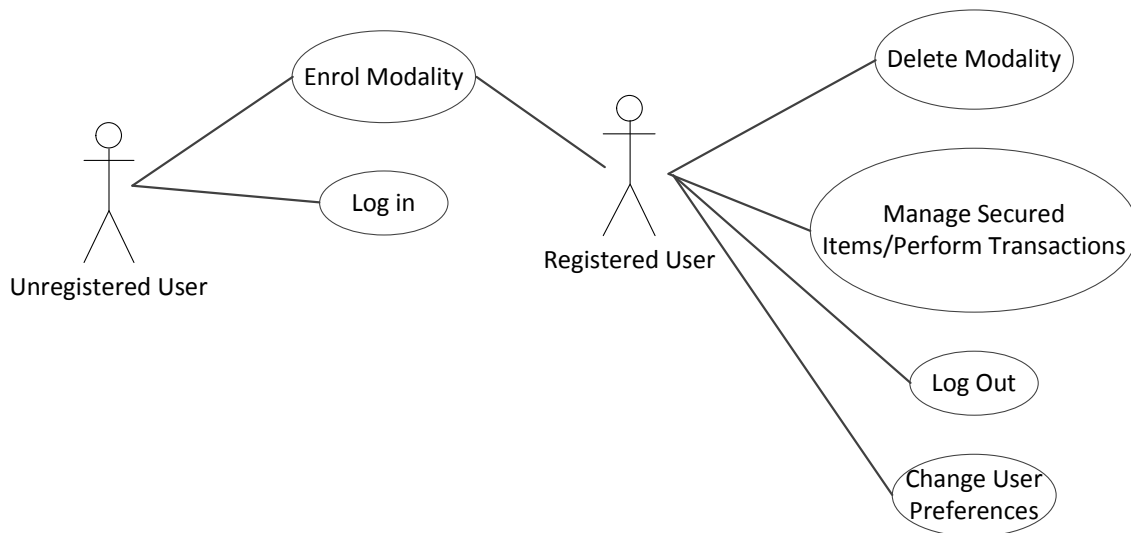


Figure 39 – Developed Android application use case UML diagram.

The application's graphical user interface screens, related with the presented use cases are shown in Figure 40.

Hand-Based Biometric Recognition System for Mobile Devices



Figure 40 – Android application Graphical User Interface (GUI) screens.

5.2.1. Android Application Optimization

The Android client developed was designed with the performance recommendations from Android Developers website (the official Android Website) in mind [31].

The developed Android application was recompiled using the zipalign tool, which optimizes Android application (.apk) files, considerably reducing the amount of RAM and allocations necessary through execution, by pre-aligning uncompressed data (in terms of the .apk file) on 4-byte boundaries.

6. Conclusions and Future Work

A secure biometric recognition system for Android mobile devices was presented and proposed in this dissertation. The Android platform and its specificities were explored and reported, along with other issues that arose throughout the planning and implementation.

The Android 2.1 platform is limited in terms of memory efficiency for camera image capture according to the official Camera API for that platform's version. Although this issue is solved in newer versions of Android, it is also possible to fix it in version 2.1 by using code reflection to iterate over the available API functions, and access hidden functions which are not documented for Android 2.1, but solve the performance limitations of the original API, allowing this platform to be used for signal processing tasks that rely in the device's camera.

Regarding the memory measures taken, the Android API provides ways for an application to have an idea of the memory that is still available, and for the application to be alerted when the system's memory budget is close to depletion. However, accurate measurements of memory become a difficult task since Android relies in the same principles as Linux, and manages memory in a different way than the typical in Windows systems. The system considers free memory as wasted memory, and does not rush to free allocated memory, since it may still be of use at a later time. Memory is only freed when necessary, or, following the principles of Java's memory management scheme, randomly, when the garbage collector is called by the system. In the event it is called, it may or may not free the memory, based on system decidability factors beyond the context of the developer's Android Java program.

The measurements regarding the CPU usage are accurate for a program with a fairly linear execution behavior through time, and with small peaks of activity, but inaccurate for high peaks of activity, since a Thread needs to be launched in parallel with the program's execution, and the measurement(s) may take place at very different situations, depending if they happen before the activity peak, during (and if during, in which stage they happen), or after. This problem is not specific to Android.

The processing times are greatly influenced by the garbage collector calls, which, for memory intensive applications such as the proposed system and its correspondent

Android client application, result in some random noise in the time results. This is not an Android issue as well, and may happen in any platform, but is aggravated in this case due to the fact that the Java Virtual Machine provides noticeable slowdowns when the garbage collector is called, and also due to the fact that the pre-segmentation delay resulting from the huge memory allocations, results in some delayed garbage collector calls right before the connection stage of the developed application.

These issues are being continuously improved and optimized in newer versions of Android, and this platform has proved fast and productive, being capable of handling research projects such as this dissertation, while providing the tools and background to make it a working application, deployable in devices and brought to a vast audience, with compatibility between multiple Android platform versions, consequently making it marketable and profitable. The proposed system was successfully implemented and works, fulfilling all those requirements and expectations.

The OLOF technique has proved very effective at providing good biometric recognition results for all the tested databases even with relatively high downsampling factors. This makes it very suitable for the mobile devices domain.

Regarding the biometric recognition performance, the HTC database built considers full hand images like the other databases considered (UST and GPDS), for comparison purposes. The downsampling ratio of 1:2 for the HTC database results in images of 776×1296 , which are relatively similar with the 1280×960 images from the UST database or the 1021×1403 images in the GPDS database. This gives some certainty that a downsampling rate of 1:2 will not decrease the system's performance in a noticeable way, and will probably give good performance boosts in both recognition performance (as seen by the decidability indexes) and overall computational performance.

Considering template shifts during template matching may greatly improve biometric recognition performance results, as seen for the UST database, but may also cause slight performance downgrades as seen for the GDPS and HTC databases. This may happen because since the comparison of the authenticating template with the stored templates considers the shift that produces the best score, the impostor shifts that generate the best score for each comparison are also considered.

In the future, a bigger database of palmprints must be created using the HTC Desire camera, in order to provide more accurate and statistically significant results. Additionally, the LDPC must be used with proper probability estimations, as suggested in [12], so that the results of the secure system can be properly studied. Other biometric traits from the hand, such as hand geometry and finger geometry, should also be taken into account to improve recognition performance.

The system needs to be tested and improved for more unconstrained environments, where the acquired images can be highly noisy in terms of light exposure, shadows and background irregularities. The usage of existing techniques of background subtraction, and adaptation to the mobile devices scenario must be taken as an important step to improve the system's usability in a real scenario, where the user should be capable of performing authentication to the system at any time and condition.

Regarding the communication between server and client, more secure approaches can be taken besides the proposed usage of a ciphered channel or SSL and certificates. The usage of security schemes based on functions with homomorphic properties should be explored and implemented as a mean to further improve the security of the communication and the privacy of the transferred data.

The evolution of the Android platform and of open source libraries that start to create branches for Android (such as OpenCV), along with the improvements in virtualization tools, and MATLAB support for mobile devices, are important factors that in a near future might be able to allow for all of the processing to be done in the client side (in the mobile device). However, the client-server architecture and its studies are equally useful, especially when considering applications that involve financial transactions, such as NFC Commerce, where the server side is necessary and might equally want to validate and perform the biometric authentications as a security policy.

7. Bibliography

- [1] Going Global Means Going Mobile in Emerging Markets - <http://blog.nielsen.com/nielsenwire/global/going-global-means-going-mobile-in-emerging-markets/>
- [2] Global Biometric Forecast to 2012 - <http://www.marketwire.com/press-release/Global-Biometric-Market-to-Grow-22-Annually-Between-2011-2013-1322339.htm>
- [3] NielsenWire - Who is Winning the U.S. Smartphone Battle? - http://blog.nielsen.com/nielsenwire/online_mobile/who-is-winning-the-u-s-smartphone-battle/
- [4] M. Ramalho, "Secure Palmprint Verification System", M.Sc. Dissertation, Instituto Superior Técnico (IST), Lisbon, 2010.
- [5] A. K. Jain, A. Ross, and S. Prabhakar, "An Introduction to Biometric Recognition". IEEE Transactions on Circuits and Systems for Video Technology, vol. 14, pp. 4-20, Jan., 2004.
- [6] A. Pocovnicu, "Biometric Security for Cell Phones". Informatica Economică, vol. 13, no. 1, pp. 57-63, 2009.
- [7] A. Mishra, "Multimodal Biometrics it is: Need for Future Systems". International Journal of Computer Applications (0975 – 8887), vol. 3, no. 4, pp. 28-29, Jun., 2010.
- [8] E. Zavadskas, A. Kaklauskas, M. Seniut, G. Dzemyda, S. Ivanikovas, V. Stankevicius, C. Simkevicius, and A. Jaruševicius, "Web-based Biometric Mouse Intelligent System for Analysis of Emotional State and Labour Productivity". Proc. of the 25th International Symposium on Automation and Robotics in Construction (ISARC), pp. 26-29, Jun., 2008.
- [9] N. Paveši, S. Ribari, and D. Ribari, "Personal authentication using hand-geometry and palmprint features – the state of the art". Proc. of Workshop on Biometrics, Cambridge, United Kingdom, Aug. 22, 2004.
- [10] C. Rathgeb and A. Uhl, "Iris-Biometric Hash Generation for Biometric Database Indexing". Proc. of the 20th International Conference on Pattern Recognition (ICPR), Istanbul, Turkey, pp. 2848-2851, Aug. 23-25, 2010.
- [11] C. Roberts, "Biometric Attack Vectors and Defences". Computers & Security, vol. 26, no. 1, pp. 14-25, Feb., 2007.
- [12] M. Ramalho, L.D. Soares and P.L. Correia, "Distributed Source Coding for Securing a Hand-based Biometric Recognition System". Proc. of the 18th International Conference on Image Processing (ICIP), Brussels, Belgium, Sep. 11-14, 2011.
- [13] K. Niinuma, P. Unsang, and A.K. Jain, "Soft Biometric Traits for Continuous User Authentication". IEEE Transactions on Information Forensics and Security, Volume 5, Issue 4, pp. 771-780, Nov. 15, 2010.

- [14] S.Z. Li and A.K. Jain, "Encyclopedia of Biometrics", Springer, Volume 2, p. 97, 2009.
- [15] S.D. Rane, W. Sun and A. Vetro, "Secure distortion computation among untrusting parties using homomorphic encryption". Proc. of the 16th IEEE International Conference on Image Processing (ICIP), Cairo, Egypt, pp. 1485-1488, Nov. 7-10, 2009.
- [16] A. Adler, "Biometric System Security". Handbook of Biometrics, Springer, ch. 19, pp. 381-402, 2008.
- [17] M. Kaur, S. Sofat and D. Saraswat, "Template and Database Security in Biometrics Systems: A Challenging Task". International Journal of Computer Applications (IJCA), vol. 4, no. 5, pp. 1-5, Jul., 2010.
- [18] A.K. Jain, K. Nandakumar, and A. Nagar, "Biometric Template Security". EURASIP Journal on Advances in Signal Processing, Jan., 2008.
- [19] B.J. Kang and K.R. Park, "A new multi-unit iris authentication based on quality assessment and score level fusion for mobile phones". Machine Vision and Applications, Springer, vol. 21, no. 4, pp. 541-553, 2009.
- [20] D.S. Jeong, H. Park, K.R. Park and J. Kim, "Iris Recognition in Mobile Phone Based on Adaptive Gabor Filter". Proc. of International Conference on Biometrics (ICB), pp. 457-463, 2006.
- [21] P. Abeni, M. Baltatu, and R. D'Alessandro, "NIS03-4: Implementing Biometrics-Based Authentication for Mobile Devices". Proc. of Global Telecommunications Conference (GLOBECOM), 2006.
- [22] L. Shen, N. Zheng, S. Zheng and W. Li, "Secure mobile services by face and speech based personal authentication". Proc. of International Conference on Intelligent Computing and Intelligent Systems (ICIS), Xiamen, China, pp. 97-100, Oct., 2010.
- [23] BioWallet Signature from Mobbeel Official Website - <http://www.mobbeel.com/products/biowallet/overview/>
- [24] BioLock App Review at Bright Hub - <http://www.brighthub.com/mobile/google-android/reviews/105400.aspx>
- [25] Anometrics - CredentialME AppLock - <http://www.anometrics.com/android/>
- [26] Biowallet Forum Discussion - <http://androidforums.com/android-applications/4063-biowallet.html>
- [27] S. Hashimi, S. Komatineni and D. MacLean, "Pro Android 2", Apress, 2010.
- [28] M. Gargenta, "Using NDK for Performance - Dalvik Versus Native", 2010 - <http://marakana.com/forums/android/examples/96.html>
- [29] Gartner's Worldwide OS Market Evolution and Forecast - <http://www.gartner.com/it/page.jsp?id=1622614>
- [30] A. Constantinou, E. Camilleri and M. Kapetanakis, "Making sense of a fragmented world: Mobile Developer Economics 2010 and Beyond", VisionMobile, pp. 32; 52, 2010.

- [31] Android Developer's Official Website - <http://developer.android.com/>
- [32] Android Compatibility Test Suite (CTS) Framework User Manual - http://static.googleusercontent.com/external_content/untrusted_dlcp/source.android.com/pt-PT//compatibility/android-cts-manual-r4.pdf
- [33] Java Advanced Imaging Core Project Website - <http://java.net/projects/jai-core>
- [34] T. Sanches, "Hand Surface Biometrics for Personal Recognition", M.Sc. Dissertation, Instituto Superior Técnico (IST), Lisbon, 2008.
- [35] Telematics News - Mobile commerce network for NFC payments - http://telematicsnews.info/2010/11/19/us-network-operators-create-mobile-commerce-network-for-nfc-payments_n112/
- [36] C. Morris, "NFC: Enabling Mobile Payments, the Internet of Things, and the Next Wave of Applications". BostInnovation, 2010. <http://bostinnovation.com/2010/12/14/nfc-enabling-mobile-payments-the-internet-of-things-and-the-next-wave-of-applications/>
- [37] Clipperz Online Password Manager - <http://www.clipperz.com/>
- [38] Dropbox - <http://www.dropbox.com/>
- [39] Android Market - Lookout Mobile Security App - https://market.android.com/details?id=com.lookout&feature=search_result
- [40] Official NFC Forum - <http://www.nfc-forum.org/aboutnfc/ecosystem/>
- [41] Z. Sun, T. Tan, Y. Wang, and S.Z. Li, "Ordinal Palmprint Representation for Personal Identification". Proc. of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, California, USA, pp. 279-284, Jun., 2005.
- [42] D.D. Zhang, "Palmprint Authentication". Kluwer Academic Publishers, 2004.
- [43] A. Kumar, D.C.M. Wong, H.C. Shen and A.K. Jain, "Personal Verification using Palmprint and Hand Geometry Biometric". Proc. of the 4th International Conference on Audio- and Video-Based Biometric Person Authentication, Guildford, United Kingdom, pp. 668-678, 2003.
- [44] Stack Overflow Android Camera Orientation - <http://stackoverflow.com/questions/4645960/how-to-set-android-camera-orientation-properly>
- [45] Android Google Group - Camera preview only works on landscape mode issue - <http://code.google.com/p/android/issues/detail?id=1193#c42>.
- [46] Adobe AIR Camera Documentation - http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/media/Camera.html
- [47] Stack Overflow - Android LED Camera behavior - <http://stackoverflow.com/questions/5017455/how-to-use-camera-flash-led-as-torch-on-a-samsung-galaxy-tab>

- [48] Android default Camera app source code -
http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/2.1_r2/android/hardware/Camera.java
- [49] Stack Overflow - Android picture distorted -
<http://stackoverflow.com/questions/5540981/picture-distorted-with-camera-and-getoptimalpreviewsize>.
- [50] Stack Overflow - Camera photos added to gallery depending on device -
<http://stackoverflow.com/questions/5221704/camera-intent-activity-avoid-saving-to-gallery>.
- [51] Things about Stuff Blog - Goodbye garbage collector
<http://nhenze.net/?p=349#comments>
- [52] Android Google Code Group Issue 2794 -
<http://code.google.com/p/android/issues/detail?id=2794>
- [53] Ivomania Mobile World Blog -
<http://superivomania.blogspot.com/2010/05/android-camera-performance-new-options.html>
- [54] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms". IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62-66, 1979.
- [55] P. Soille, "Morphological Image Analysis: Principles and Applications". Springer-Verlag, pp. 173-174, 1999.
- [56] L. Shapiro and G. Stockman, "Computer Vision". Prentice Hall, 2001.
- [57] E. Konukoğlu, E. Yörük, J. Darbon and B. Sankur, "Shape-Based Hand Recognition". IEEE Transactions on Image Processing, vol. 15, no. 7, pp. 1803-1815, Jul., 2006.
- [58] C. Lin, T. Chuang and K. Fan, "Palmprint Verification using Hierarchical Decomposition". Pattern Recognition, vol. 38, no. 12, pp. 2639-2652, Dec., 2005.
- [59] N. Ferguson and B. Schneier, "Practical Cryptography". John Wiley & Sons, 2003.
- [60] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "Sponge Functions". Ecrypt Hash Workshop, 2007.
- [61] RadioGatún Hash Website - <http://radiogatun.noekeon.org/>
- [62] "UST Hand Image Database, Department of Computer Science, The Hong Kong University of Science and Technology. (Provided by Dr. Helen Shen)".
- [63] M. A. Ferrer, A. Morales, C. M. Travieso, J. B. Alonso, "Low Cost Multimodal Biometric Identification System based on Hand Geometry, Palm and Finger Textures". Proc. of the 41st Annual IEEE International Carnahan Conference on Security Technology, ISBN: 1-4244-1129-7, Ottawa, Canada, pp. 52-58, Oct. 8-11, 2007.
- [64] J. Daugman, "How Iris Recognition Works". IEEE Transactions on Circuits and Systems for Video Technology, vol. 14, no. 1, pp. 21-30, Jan., 2004.