Face Authentication on Mobile Devices: Optimization Techniques and Applications

PUN Kwok Ho

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

July 2005

Abstract

Mobile devices, such as cellphones and PDAs, have become an essential part of our daily lives. Mobile applications such as phone banking and e-commerce are becoming increasingly popular. To protect the large amount of sensitive information stored in and handled by these devices, more secure authentication methods such as face recognition and fingerprint verification have been proposed to replace traditional passwords. Unfortunately, the weak computation power of mobile devices remains the biggest obstacle to the adoption of the less intrusive face recognition methods. In this thesis, two representative face recognition algorithms, Principal Component Analysis (PCA) and Elastic Bunch Graph Matching (EBGM), are implemented and optimized for mobile devices. An optimization model and various optimization techniques are proposed to fully exploit the computation power of mobile devices. Experimental results show a significant improvement in execution time. For PCA, the time for one authentication session is reduced from 30 seconds to 1.3 seconds; for EBGM, a 420 times reduction is achieved – from 553 seconds down to 1.3 seconds. Further verification experiments show that the real time performance is achieved without any significant loss in accuracy. Our results enable the practical use of face authentication on mobile devices. Also, the optimization model and techniques developed in this thesis can be easily adapted to other applications which require real time performance on constrained platforms.

論文摘要

流動設備(Mobile Devices)諸如流動電話和個人數位助理(PDA)已經成爲我們日常生活的一部份，像電話理財和電子商務之類的應用變得越來越普及。由於這些流動設備儲存並處理大量的個人資料，一種安全可靠的認證方法更形重要。傳統的密碼認證方法，容易產生用戶忘記密碼和密碼被盜等問題，未能提供足夠的保護。與之相比，生物特徵辨識技術(Biometrics)如人臉識別(Face Recognition)和指紋核實(Fingerprint Verification)，具有通用性、唯一性和持久性等優點，提供了更方便和可靠的認證方法。不過由於流動設備有限的運算能力，這些技術一直未被廣泛地應用在其上。有見及此，我們實現並優化了兩種具代表性的人臉識別算法: 主要成分分析 (PCA) 及彈性束圖匹配(EBGM)。我們提出了一個針對優化的軟件工程模型以及多種優化技巧。優化後的 PCA 速度加快了六倍，認證時間從原來的 30 秒減少至 1.3 秒，而 EBGM 的認證速度更加快了 420 倍 — 從 553 秒下降至 1.3 秒。實驗結果顯示，在無損認證準確性的前提下，兩種算法在流動設備上的執行時間大幅減少，達到了實時的要求。我們的研究令流動設備的運算能力得到充份的發揮，使到人臉識別技術在流動裝備上的廣泛應用變得可能。我們提出的軟件優化模型和技巧，同時適用於其他系統資源有限的平臺上的開發。

To my dearest family and friends

Acknowledgment

I wish to thank my supervisor, Prof. Moon Yiu Sang, for his continuous guidance and supervision, for supporting me to attend academic conferences and for all his teachings in research, learning and living. I wish to express my gratitude to Prof. Heng Pheng Ann and Prof. Wu Yu Liang for being members of my thesis committee. I would also like to thank Prof. G. Fairweather for being my external marker.

Special thanks are due to the face recognition team at the Colorado State University for making their Face Identification Evaluation System publicly available. Without their contribution, this research work would not have such a good start. I gratefully acknowledge helpful discussions with Mr. Chen Jiansheng and Dr. Gary Leung for sharing their research insight and ideas with me. My sincere thanks to Chiu, Gary and for doing such a wonderful job in implementation of the door access control system. I must thank Tony, Kim, Terence, Oscar, Simon and other technical staff for their patience with me and excellent work in maintaining a stable computing environment. I would like to thank the Chinese University of Hong Kong for providing such a beautiful campus, a resourceful library and excellent recreational facilities (especially the swimming pool).

Then there is the list of friends and colleagues that deserve my deepest gratitude. Many thanks to Walty and Leo for making our corner a happy one; Szeto, Josh, Bud,

Gigi, Li Gang and Zhang Kun, for creating such an enjoyable atmosphere in room 1013; Gordon and Lu Yang, for bringing us laughter via emails even after their departure; Alan, Ken and CJ, for inviting me to join their board games while I was writing this thesis; Hackker and Raymond, for letting me know what is meant by efficient and hardworking; Friends in room 1026, for all the wonderful snacks and pizzas.

And, my heartfelt thanks to those who have helped me one way or the other, even though both you and I may not be aware of it. I wish to express my deepest gratitude to Simon, Fai and Amanda for giving me humble advice during troubled times. My special thanks goes to Simon, who gave me motivation to finish this thesis by telling me that he had already finished his in April.

I would like to thank the world for being such a wonderful place to live in, to appreciate and to explore.

And most of all, thanks to my mum and sister for giving me love and support.

Table of Contents

List of Tables

List of Figures

List of Algorithms

1. Introduction

1.1 Background

1.1.1 Introduction to Biometrics

Biometrics are measurable physiological characteristics such as face, iris and fingerprint, or behavioural traits such as gait and handwriting, which can be used to verify the identity of an individual[1]. Figure 1 shows some typical examples of biometrics.

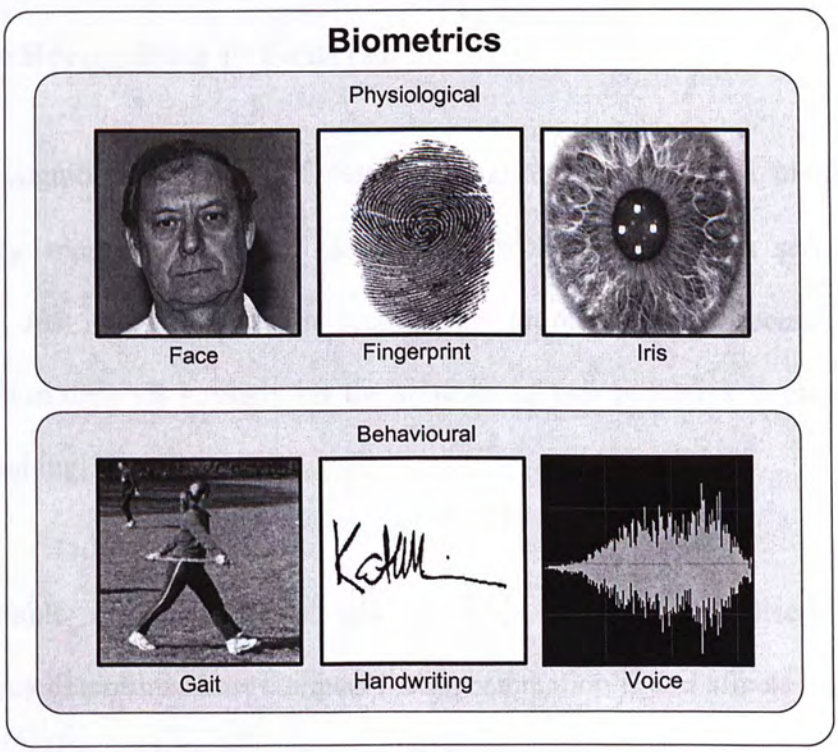

Figure 1 Examples of biometrics

Compared with traditional authentication methods such as password, a biometrics-based authentication technique presents a more robust alternative because it depends on physiological or behavioral traits which are unique, immutable and cannot be stolen[2]. Among all forms of biometrics, the face has an added advantage in that it can be captured easily and non-intrusively with a low cost, off-the-shelf camera instead of expensive sensors. This is also the reason why face authentication becomes increasingly popular, with military applications such as border control and security access and civilian ones such as personal computer login.

1.1.2 Face Recognition in General

Face recognition is a class of pattern recognition problems. It involves the automatic matching of novel face images with previous ones seen by the system. Just like any pattern recognition problems, the success of face recognition depends strongly on the solution of two problems: representation and matching[3].

A desirable representation should be both efficient and discriminating. Efficiency determines how compact the representation is and affects the storage and computational requirement. Discriminating power determines how far apart the faces are under the representation. A raw face image is transformed into the selected representation via a process called feature extraction, in which only the

most discriminating features are extracted, preserved and stored as a face template. A clearly and easily separable feature space is considered crucial to the final matching stage.

After feature extraction, matching is carried out in the feature space. A majority of face recognition makes use of minimum distance matching, which computes the separation (e.g., Euclidean distance) between the live and registered face template.

Face recognition systems are often classified by the representation scheme they adopt. Local feature (component) based systems record and represent faces using useful features extracted from different locations of the face. Features ranging from more prominent ones like eyes, nose and mouth, to subtle ones like skin texture are used, either alone or in combination. Global feature based (holistic) systems, on the other hand, treat a face as a whole and extract statistical information from the entire face. Local information such as the relationship between different facial components is ignored. Examples of both classes of systems will be given in Section 2.1.

In the following sections, the general components and tasks of a face recognition system are discussed. General terminologies and evaluation methodologies will also be introduced.

1.1.3 Typical Face Recognition Systems

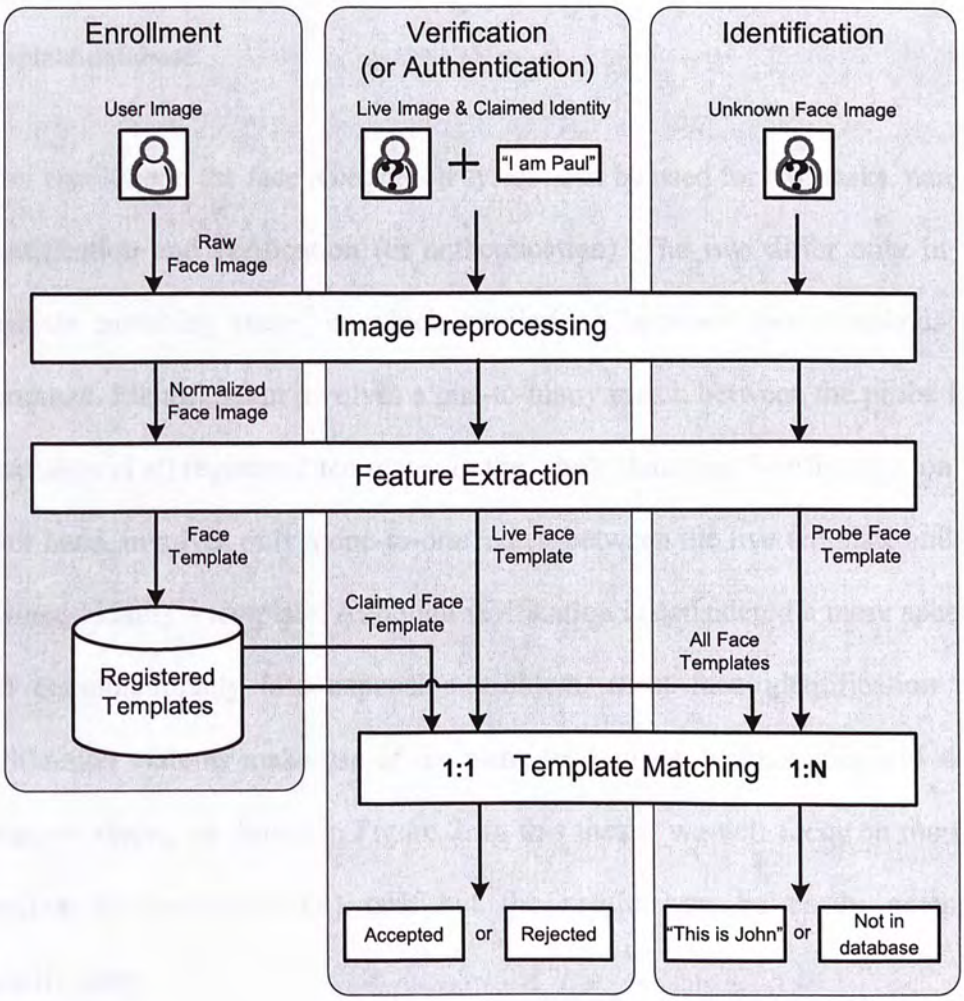

Figure 2 Stages and tasks of a typical face recognition system

As shown in Figure 2, a typical face recognition system consists of three stages: 1) Image Preprocessing, 2) Feature Extraction and 3) Template Matching. Before a face recognition system can be put to use, authorized users must first be enrolled. During enrollment, a user's face image is first acquired. Then this raw image is normalized in terms of size, rotation, intensity and contrast in the

image preprocessing stage. Distinguishing features are then extracted from the normalized face image to form a face template, which is then stored into the template database.

After enrollment, the face recognition system can be used for two tasks, namely identification and verification (or authentication). The two differ only in the template matching stage, in which similarities between face templates are computed. Identification involves a one-to-many match between the probe face template and all registered templates in the whole database. Verification, on the other hand, involves only a one-to-one match between the live template and the claimed identity's template. Although verification is considered a more specific and computationally less expensive problem, most face identification and verification systems make use of the same recognition technologies and share common stages, as shown in Figure 2. In this thesis, we will focus on the face verification (authentication) task but the results can be easily extended identification.

1.1.4 Face Database and Evaluation Protocol

To evaluate a biometrics verification system, large database testing is often done to determine the accuracy. In all our experiments, the FERET verification testing protocol for face recognition [4] is used. FERET is the de facto standard evaluation methodology in the face recognition domain. It is used in

conjunction with the Facial Recognition Technology (FERET) Database[5]. The FERET image corpus was collected and assembled under the sponsorship of the Department of Defense of the United States. The two are widely used by the face recognition community for the testing and evaluation of face recognition algorithms.

For the FERET database, images of an individual were acquired in sets of 5 to 11 images, collected under relatively unconstrained conditions. Two frontal views with different facial expressions were taken (**fa** and **fb**). For 200 sets of images, a third frontal image was taken with a different camera and different lighting (**fc**). The duplicate I (**dup1**) images were obtained anywhere between one minute and 1031 days after their respective gallery matches. The duplicate II (**dup2**) images are a strict subset of the duplicate I images; they are those taken at least 18 months after their gallery entries. Figure 3 shows an example of the different categories of images.

Figure 3 Examples of images in the FERET database[5]

For evaluation, the FERET images are split into two sets: gallery and probe set. The gallery is the set of known individuals. An image of an unknown face presented to an algorithm is called a probe, and the collection of probes is called the probe set. For each set of images, one of the frontal images (**fa** or **fb**) was randomly placed in the gallery, and the other images were placed in the **FB** probe set. The **dup1**, **fc** and **dup2** images form the corresponding probe sets. The same gallery is used for all probe sets. Table 1 shows a summary of the probe categories.

Probe category	Evaluation Task	Gallery size	Probe set size
FB	Facial expression	1196	1195
dup1	Aging of subjects	1196	722
fc	Illumination	1196	194
dup2	Aging of subjects	1196	234

Table 1 Summary of probe categories used in the FERET test [4]

1.1.5 Evaluation Metrics

When given two face templates, a biometrics verification system will output a binary decision – either 'accept' (the two templates come from the same person) or 'reject' (the templates do not match). Depending on the correctness of this decision, there are a total of four possible outcomes:

1. Genuine Accept: a genuine identity is accepted.
2. True Reject: an impostor is correctly rejected.
3. False Accept: an impostor is accepted as a genuine.
4. False Reject: a genuine identity is rejected as a fake.

For an ideal system, only cases 1 and 2 will occur, accepting all registered users while rejecting all impostors. For a real system, however, case 3 and 4 do occur and the probabilities of occurrence for these two errors are measured to determine how accurate a system is. False Acceptance Rate (FAR) refers to the probability that the system identifies an impostor as a genuine, while False Rejection Rate (FRR) refers to the probability that a genuine is identified as an imposter. As shown in Figure 4, FAR and FRR are closely related with the system threshold. The higher the threshold, the harder for a face template, be it genuine or not, to be accepted by the system. Hence when the threshold increases, FAR decreases and FRR increases and vice versa. The error rate at which the FAR equals FRR is called the Equal Error Rate (EER). It is useful for describing the accuracy of a system when the costs for false rejects and false accepts are equal. The lower the EER, the more accurate a system is.

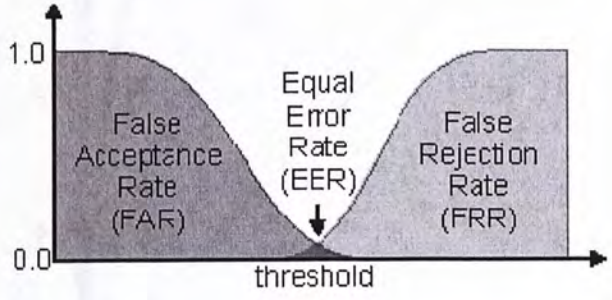

Figure 4 Relationship between FAR, FRR and EER

Since biometrics verification system are used in a variety of situations, each requiring different levels of security, it is common to tune the system threshold

so that a specific operating point – a particular FAR and FRR – can be obtained. The Receiver Operating Characteristics (ROC) curve is introduced for this purpose. As shown in Figure 5, the ROC curve describes the relationship between the Genuine Acceptance Rate (GAR or 1-FRR) and the False Acceptance Rate (FAR). An operating point can be chosen judging from the requirement of a particular application. The higher an operating point is, the higher will be the GAR; the more an operating point is to the left, the lower will be the FAR. Hence, a specific operating point can be picked from the ROC curve to suit ones needs. ROC curves also provide a way for easy comparison between different biometric systems. For the curves illustrated, system A is better than B which is in turn better than C.

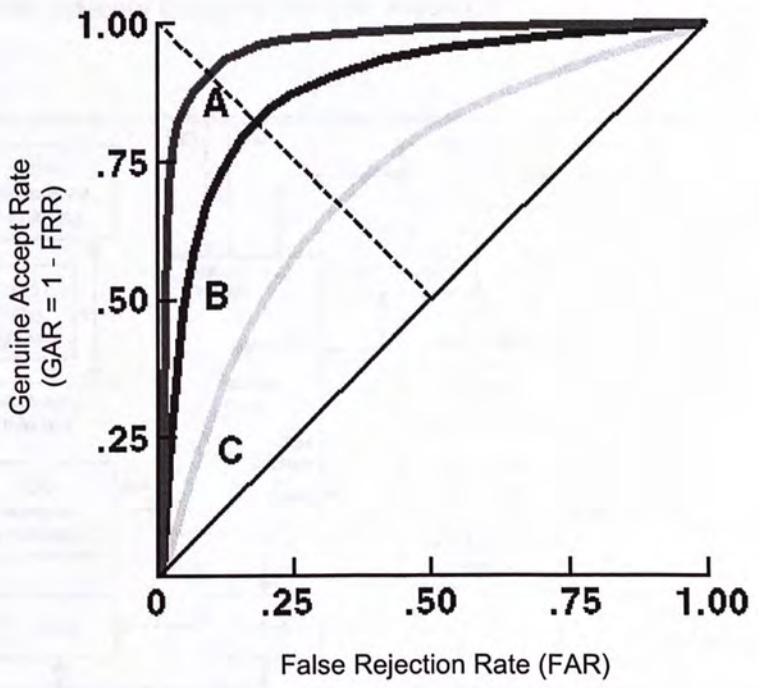

Figure 5 A Receiver Operating Characteristics (ROC) Curve

1.1.6 Characteristics of Mobile Devices

Typical mobile devices include mobile phones and Personal Digital Assistants (PDAs). Due to their portability and connectivity, mobile devices have become an essential part of our daily lives. Thanks to the advancement in technology, mobile devices have become more and more powerful. Unfortunately, inherent limitations such as size, cost and power consumption still persists for all mobile devices. To conserve battery power and chip size, typical processors for mobile devices, such as the Intel XScale [6], have a comparatively small cache size (Figure 6) and do not come with a Floating Point Unit (FPU). A small cache size means data and instructions are swapped out of cache more frequently and much slower off-chip memory must be accessed.

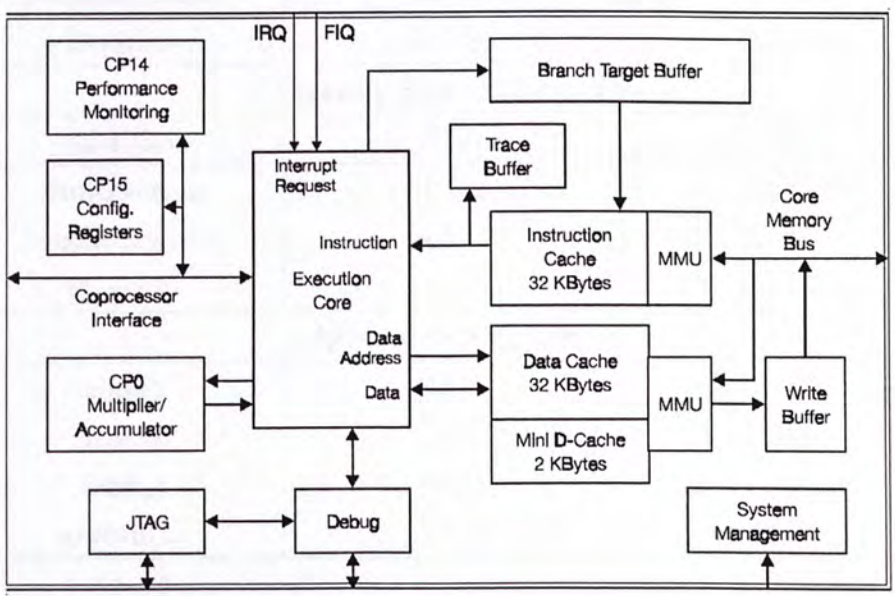

Figure 6 Intel XScale PXA255 Processor Block Diagram [6]

The lack of an FPU, on the other hand, cause floating point operations to be unacceptably slow since they are emulated by a software library. Table 2 shows a comparison between the Intel XScale PXA255 400MHz and Pentium III 450MHz. Although the two processors have close clock rates, the performance for floating point arithmetic is dramatically different - the XScale is about 70 - 120 times slower than the Pentium III. The gap grows even wider for trigonometric functions (160 - 240 times). From these results, it is obvious that for any serious, floating point intensive applications to be practical on mobile devices, dedicated efforts must be made to optimize their performance.

Processors	Intel XScale 400MHz	Pentium III 450MHz
Integer Arithmetic		
Addition	0.138s	0.032s
Subtraction	0.138s	0.047s
Multiplication	0.145s	0.047s
Division	0.698s	0.234s
Floating Point Arithmetic		
Addition	3.87s	0.031s
Subtraction	4.50s	0.047s
Multiplication	4.60s	0.047s
Division	14.0s	0.188s
Trigonometric Functions		
sin(x)	237s	0.984s
cos(x)	253s	1.016s
tan(x)	197s	1.203s
arctan(x)	251s	1.453s

Table 2 Excecution time of 3 million instructions on different processors

1.2 Motivation and Objectives

In recent years, we have seen a proliferation of mobile devices such as PDAs and cell phones as well as applications on them. Individuals use them for electronic transactions such as phone banking and stock trading, while enterprises issue them as a means of access to the corporate network. Security concerns with mobile devices have become an imminent issue as a poorly protected mobile device may expose sensitive data such as passwords and credit card information, or may become a security hole of the entire corporate network.

Unfortunately, mobile devices are traditionally guarded by simple passwords which can be easy copied, stolen or forgotten. Clearly, a more sophisticated authentication method is needed. As pointed out in Section 1.1.1, biometrics-based authentication presents a more robust alternative because it depends on physiological or behavioral traits which are unique, immutable and cannot be stolen[2]. Among the better known forms of biometrics, face authentication is ideal for mobile devices not only because it is non-intrusive, but also because it requires no extra sensors like a fingerprint does. Nowadays, most cell phones and many PDAs already come with built-in cameras[7], making face authentication a cost-effective solution to the security problem.

Traditional face recognition systems, however, involves heavy image

processing and are designed for powerful desktop PCs or faster machines. The large number of floating point calculations and memory accesses involved become a great obstacle to the adoption of face authentication on mobile devices. As discussed in Section 1.1.6, even state-of-the-art mobile devices cannot satisfy these demands. Any direct porting of existing systems will only result in slow and unacceptable performance. Clearly, optimization is the key to real time face authentication on mobile devices.

In this thesis, we will study and optimize two widely adopted face recognition algorithms and compare their performances in a mobile context. Our goal is to show that real time face authentication can be achieved on mobile devices without loss of accuracy.

1.3 Major Contributions

1.3.1 Optimization Framework

During the optimization of PCA and EBGM, it becomes clear that a software engineering model dedicated to the general optimization problem is needed. In light of this, we propose a high level view of optimization techniques and a feedback oriented workflow. The high level view gives insights on how general optimization problems should be approached and how these techniques can be categorized; the workflow keeps us sensitive to changes in program behaviour and measures the effectiveness of a technique to a particular problem. The

simplicity and flexibility of the two makes them easily adaptable to other optimization problems. More details can be found in Chapter 3.

1.3.2 Real Time Principal Component Analysis

Principal Component Analysis (PCA)[8] is a widely adopted, global feature based face recognition technique. In this study, we investigate the feasibility of a real-time implementation of PCA on mobile devices. The performance of the un-optimized code is far from satisfactory, requiring 30 seconds for just one authentication session. After extensive profiling, we found that the bottleneck is the large amount of floating point multiplications. By using optimization techniques such as fixed-point arithmetic and pre-computation, the execution time for an authentication session reduced by 22 times – to 1.3 seconds only. Large database testing shows that there is no significant loss in verification accuracy[9]. More details can be found in Chapter 4.

1.3.3 Real Time Elastic Bunch Graph Matching

In contrast to PCA, Elastic Bunch Graph Matching (EBGM) [10]based on local features of a face and is less susceptible to lighting, face position and expression variations. In this study, we attempt to build a real-time implementation of EBGM on mobile devices. An un-optimized implementation takes 550 seconds (Section 4.3.1) to complete a single face authentication

session and is unacceptable for any practical use. To improve the execution time, profiling is done to pinpoint the bottlenecks. Various optimization techniques such as fixed-point arithmetic, table lookup, pre-computation and memory optimization are developed and employed. The result is a 420 times improvement - the optimized code now only takes 1.3s for one authentication. More details can be found in Chapter 5.

1.4 Thesis Organization

This chapter provided an overview of face recognition and characteristics of mobile devices. A summary of the objectives and contributions of our work were also given. The remaining chapters of this thesis are organized as follows: Chapter 2 briefly reviews related work in face recognition and in particular those iset in a mobile context. Chapter 3 presents a high level view of different levels of optimization and a proposed workflow. Chapter 4 and 5 present an experimental study of the optimizations of PCA and EBGM respectively. Finally, Chapter 6 summarizes the research performed, and describes some challenges encountered and possible directions for future research.

2. Related Work

2.1 Face Recognition for Desktop Computers

As discussed in Section 1.1.2, face recognition systems are often classified by the representation scheme they adopt. The two major classes of system are global feature based and local feature based systems. In this section, a brief description and representative examples will be given for both classes.

2.1.1 Global Feature Based Systems

In this class of systems, face images are treated as a whole and statistical information from the entire face is extracted. Most systems in this class involve finding an easily separable subspace. By projecting the high dimensional face image to a well-selected subspace of lower dimension, an efficient and possibly discriminating representation is acquired. Various subspace selection methods, including Evolution Pursuit(EP)[11], Independent Component Analysis(ICA)[12], Principal Component Analysis (PCA) [13] and Linear Discriminant Analysis (LDA)[14] are commonly used methods, with the latter two being the most popular ones due to their simplicity and reasonable performance.

Principal Component Analysis (PCA) [13]

Principal Component Analysis (or eigenface) is a general and widely-adopted statistical method for dimension reduction. The covariance matrix of the face images is first found and then the eigenvectors for this matrix are in turn computed. The top eigenvector, proven to be the most efficient (but not necessarily discriminating) representations for the faces, is preserved for the construction of a transformation matrix. All face images are then transformed and projected to the PCA subspace, in which matching will be done.

Figure 7 Principal Components (eigenfaces) [13]

Linear Discriminant Analysis (LDA) [14]

Despite the fact that PCA derives an efficient subspace, it treats all face images – even those from the same person – as different classes. The variations between face images of the same person is not addressed and results in

non-optimal discriminating power among persons. LDA (or fisherface) tackles this problem by including class specific information in determining the subspace. The criterion is to maximize the ratio of the interpersonal variance to the intrapersonal variance, so that the resultant subspace is easily separable between classes (persons). Usually several images of the same person are needed during enrollment to capture as much intra-class variation as possible.

2.1.2 Local Feature Based Systems

Earlier methods in this class uses purely geometry based approach. The relative size of and distance between various facial components like eyes, nose and mouth are measured and used as features. Unfortunately, this kind of methods is susceptible to deformation. Updated methods cater for the deformation problem by allowing a reasonable amount of displacement to exist between feature points. Local features are often extracted by means of filter convolution, such as Gabor filters. The most representative methods in this class include Elastic Bunch Graph Matching (EBGM) and Local Feature Analysis (LFA).

Elastic Bunch Graph Matching (EBGM) [10]

This method makes use of deformable templates which allow approximation rather than exact matching between the feature points in terms of relative location. Faces are represented as a set of local features located on an elastic (deformable) graph (see Figure 8). Local features are first computed by

convoluting the face image with 2D Gabor filters with various centre frequencies, bandwidths and orientations. The filter outputs are then sampled at different locations on the graph. During matching, the novel face graph is matched with the registered one. The best match is one that preserves features and local geometry.

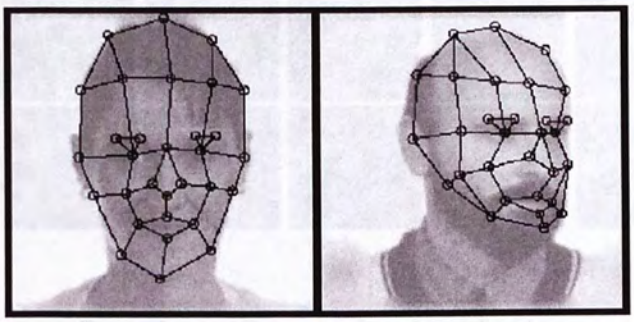

Figure 8 Elastic graphs overlaid on face images [10]

Local Feature Analysis (LFA) [15]

Local feature analysis is a derivative of the eigenface method[16]. Instead of the entire representation of a face, LFA utilizes specific areas of a face such as eyes and areas of definite bone curvature differences, such as the cheeks. The responses of multi-scale filters are used as local features and encoded using PCA to obtain a compact description.

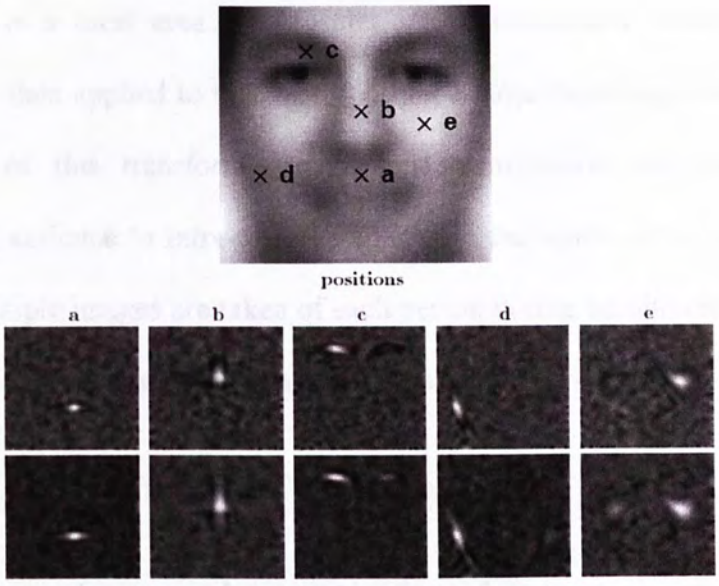

Figure 9 Local features selected by LFA [15]

2.1.3 Commercial Systems

Technology details of commercial systems are either not disclosed or a combination of local/global methods is used; hence it is hard to classify them into any single category. The face recognition products discussed below are ranked among the top 3 in the Face Recognition Vendor Test 2002 (FVRT2002)[17], a public evaluation contest of face recognition algorithms.

FaceVACs. Cognitec Systems GmbH. [18]

In this product, feature extraction starts with local image transforms that are applied at fixed image locations. These transforms capture local information relevant for distinguishing people, for example, the amplitudes at certain spatial

frequencies in a local area. The results are collected in a vector. A global transform is then applied to this vector. Using a large face-image database, the parameters of this transform are chosen to maximize the ratio of the inter-person variance to intra-person variance in the space of the transformed vectors. Multiple images are taken of each person during enrollment in order to better cover the range of possible appearances of that person's face.

fR. Neven Vision, Inc. [19]

This product makes use of a combination of Gabor wavelet and neural networks. The face detection modules employ a general face model to localize a face in the image. A 3D representation of the head copes with pose variations. The general face models are learned from a large database of face images and cover a wide variety of environmental conditions such as illumination and expressions. Local features such as eye-position, nose-position and mouth-position are used.

FaceIt. Indentix, Inc. [20]

This product uses a combination of three technologies namely Vector Feature Analysis (VFA), Local Feature Analysis (LFA) and Surface Texture Analysis (STA). VFA is optimized for low-resolution images and runs at a very high speed. LFA is based on facial geometry information and is optimized for low to medium resolution images. STA is based on skin texture micro-features and is

frequencies in a local area. The results are collected in a vector. A global transform is then applied to this vector. Using a large face-image database, the parameters of this transform are chosen to maximize the ratio of the inter-person variance to intra-person variance in the space of the transformed vectors. Multiple images are taken of each person during enrollment in order to better cover the range of possible appearances of that person's face.

fR. Neven Vision, Inc. [19]

This product makes use of a combination of Gabor wavelet and neural networks. The face detection modules employ a general face model to localize a face in the image. A 3D representation of the head copes with pose variations. The general face models are learned from a large database of face images and cover a wide variety of environmental conditions such as illumination and expressions. Local features such as eye-position, nose-position and mouth-position are used.

FaceIt. Indentix, Inc. [20]

This product uses a combination of three technologies namely Vector Feature Analysis (VFA), Local Feature Analysis (LFA) and Surface Texture Analysis (STA). VFA is optimized for low-resolution images and runs at a very high speed. LFA is based on facial geometry information and is optimized for low to medium resolution images. STA is based on skin texture micro-features and is

optimized for higher resolution images.

In one-to-one applications, VFA, LFA and STA algorithms can be used alone or in combination. In one-to-many applications, the three algorithms are used in a three-pass pipeline, where only the top percentage results of the previous pass are searched again in the next pass. For maximum speed and accuracy, VFA is used for the first pass, LFA for the second, and STA for the third. At the conclusion of the passes, the top scores of all three passes are fused together into a final set of scores.

2.2 Biometrics on Mobile Devices

In recent years, there has been growing interest in biometrics on mobile devices. Previous studies on face recognition for mobile devices mainly focus on using the device as a capturing tool, with most expensive computations done on PC servers [21, 22, 23, 24]. Some require specialized hardware which makes use of a multi-processor architecture to achieve real time performance[25]. Other attempts to implement a face recognition system on mobile devices have been reported, but no detailed account of the execution time was given [26, 27, 28]. Recently, commercial face authentication systems for mobile phones have also been introduced, including OKAO Vision from Omron [29, 30] and Genelock-light from Earth Beat [31].

Figure 10 OKAO Vision from Omron [27, 28]

Due to the high demands for more secure authentication methods, researchers around the world have also investigated other modes of biometrics authentication. Real-time voice [32] and fingerprint [33, 34, 35, 36] verification systems have been successfully implemented on mobile devices. When combined with the real-time face recognition systems developed in this thesis, it is now possible to further enhance security on mobile devices by using multimodal schemes such as the one outlined in [23].

3. Optimization Framework

3.1 Introduction

During the course of optimizing the PCA and EBGM, we have developed a common set of techniques and strategies which is re-applicable to other optimization problems. Under this unified framework, the problems of optimizing PCA and EBGM, though inherently different, can be described and approached in a similar way.

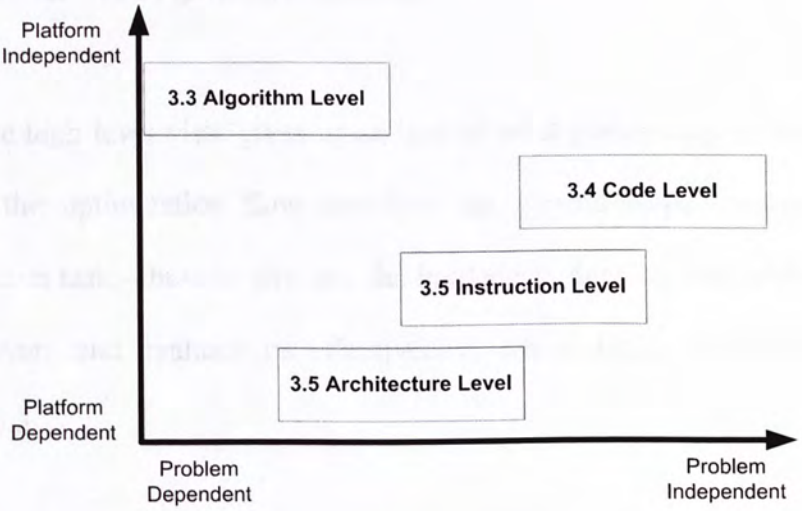

Figure 11 Different levels of optimization

Our framework consists of a high level view and a general workflow of the optimization problem. Figure 11 shows the four levels of optimization (algorithm, code, instruction and architecture), with varying degrees of platform and problem dependence. Platform dependence describes how much

knowledge of the target platform is needed for an optimization level. For instance, the architecture level is highly platform dependent, suggesting that a thorough understanding of the platform specification, such as cache configuration, is needed. Problem dependence, on the other hand, describes the importance of the problem nature – whether it is in the image processing domain or pattern recognition domain. For example, the algorithm level is highly problem dependent meaning that the nature of the problem, such as its computational complexity, is crucial at this level. A detailed account of each of the four levels will be given in Section 3.2.

While the high level view gives us an idea of what preliminary information is needed, the optimization flow describes the general steps involved in an optimization task – how to pinpoint the bottleneck, develop and implement an optimization, and evaluate its effectiveness. More details can be found in Section 3.3.

3.2 Levels of Optimization

3.2.1 Algorithm Level

Algorithm level optimizations involve the selection of an effective algorithm and a suitable set of input parameters for a problem. The algorithm level is purely problem dependent, in that optimizations often aim at improving the

intrinsic qualities of an algorithm, such as computational complexity. On the other hand, it is completely platform independent in that the same order of improvement can be achieved regardless of the target platform. The Algorithm level is often the first to consider during optimization, as a well-optimized algorithm provides a stable framework under which techniques of the other levels can be applied.

A significant part of the EBGM optimization belongs to the algorithm level. For instance, the invariant Gabor masks and bunch graphs are pre-computed and removed from the authentication routine (Section 5.4.2); Gabor mask convolutions are modified to exploit their parallel nature to reduce number of memory access (Section 5.4.6). Finally, an efficient set of Gabor filters is derived in an attempt to strike a balance between speed and accuracy (Section 5.4.4).

3.2.2 Code Level

Algorithmic optimizations can only be realized through efficient implementation. The aim of code level optimization is to improve on a correct but suboptimal implementation. At this level, a significant amount of effort is often dedicated to optimizing loops, in which most execution time is spent. Common code optimization techniques includes loop unrolling, which reduces overhead per iteration and code motion, which saves unnecessary computations

by moving loop invariant code out of the loop body. These techniques are widely employed in both the optimization of PCA and EBGM.

Code level optimization is problem independent since the control structures being optimized are common to all problems. However, as the actual implementation depends on the programming languages available on a given platform, code level optimizations can be slightly platform dependent.

3.2.3 Instruction Level

At the instruction level, the difference between basic operations such as integer and floating point arithmetic must be accounted for. This is especially true when the target platform is a mobile device, in which hardware Floating Point Units (FPU) are absent and floating point instructions are unavailable (Section1.1.6). Due to the fact that software floating point emulators are hundreds of times slower than their hardware counterpart, floating point avoidance is often regarded as a rule of thumb for mobile device optimizations.

To circumvent slow floating point operations, fixed point arithmetic can be used. By representing real numbers using integers and replacing floating point arithmetic with integer ones, the performance of real number operations can be significantly improved. A significant portion of speedup of PCA and EBGM is achieved by the adoption of fixed point arithmetic. The implementation details

and speedup of fixed point arithmetic can be found in Section 4.3 and 5.4.1.

Besides basic floating point operations, a class of commonly used routines deserves special attention. Trigonometric functions provided by the standard math library are evaluated by floating point polynomial expansion and is extremely slow. The use of table lookup is a fast and viable alternative, as will be proven in the optimization of EBGM (Section 5.4.8).

The instruction level is platform dependent, as the problems tackled at this level are often shared by the same family of processors or devices. And since the problem nature indirectly determines the amount of floating point or trigonometric operations used, instruction level optimizations are also moderately problem dependent.

3.2.4 Architecture Level

Architecture level optimizations mostly deal with cache and memory optimization. By fine tuning factors such as system cache policy and memory page attributes, the overall throughput and efficiency of the memory hierarchy can be maximized. Detail specification of the target platform must be known so that a specific set of parameters can be derived for each target.

Besides being highly platform dependent, the architecture level also relies

heavily on a thorough understanding of the problem at hand. Only if knowledge of both worlds is combined can a feasible optimization strategy be derived. For instance, while the processor is totally oblivious to the memory access pattern of a program, it is predictable and known a priori by the programmer. By utilizing information about the low level cache configuration and the high level memory access behaviour, a programmer can derive efficient memory access schemes such as those employed in EBGM optimization (see Section 5.4.5 and 5.4.7) – a hardware/software co-design effort.

3.3 General Optimization Workflow

Before an optimization strategy can be developed and tested, a feedback mechanism must be in place for easy evaluation and refinement of a strategy. Here we describe a general optimization flow which is used throughout the PCA and EBGM optimizations.

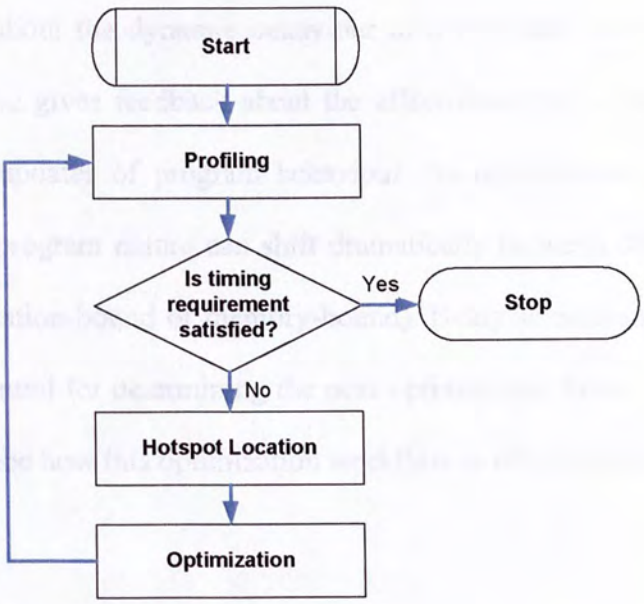

Figure 12 General optimization flowchart

As shown in Figure 12, the optimization workflow consists of four stages. In the profiling stage, program runtime statistics such as overall execution time, function counts and page faults are measured. Then the overall execution time is checked against the timing requirement of the application at hand, and if it is not satisfied, hotspot location is done to pinpoint the area which deserves the most attention. Targeting the located hotspots, an optimization strategy is derived and implemented in the optimization stage. Finally, the optimized program is subjected to profiling again, and the whole process goes on until the timing requirement is satisfied.

Here we notice a dual role assumed by the profiling stage: before each optimization, profiling acts as a preparatory stage and provides the necessary

information about the dynamic behaviour of a program; after optimization is done, profiling gives feedback about the effectiveness of a strategy and more importantly, updates of program behaviour. As optimization is a continuing process, the program nature can shift dramatically between different extremes (e.g., computation-bound or memory-bound). Being sensitive to these changes is hence essential for determining the next optimization focus. Chapter 4 and 5 further describe how this optimization workflow works in practice.

3.4 Summary

Here we categorize the optimization strategies used in PCA and EBGM optimization into different levels, as shown in Table 3:

Level	Strategies	Section
Algorithm	- Efficient Gabor Filter Selection	5.4.4
	- Gabor Masks and Bunch Graphs Precomputation	5.4.2
	- Reducing Redundant Memory Access	5.4.6
Code	- Improving Array Access	5.4.3
Instruction	- Fixed-point Arithmetic	5.4.1, 4.3
	- Optimization of Trigonometric Functions	5.4.8
Architecture	- Fine Tuning System Cache Policy	5.4.5
	- Maximizing Cache Reuse by Array Merging	5.4.7

Table 3 Optimization strategies employed

4. Real Time Principal Component Analysis

4.1 Introduction

The Principal Component Analysis (PCA) Algorithm, also known as eigenface, is a tested and widely adopted face recognition method and was first proposed in[13]. In our experiments, the PCA implementation of the CSU Face Identification Evaluation System 5.0 [37] is used as the baseline system. As shown in Figure 13, PCA authentication consists of four main stages: 1) Image Preprocessing, 2) PCA Subspace Training, 3) PCA Subspace Projection and 4) Template Matching. Each of these stages will be discussed in detail in Section 4.2.

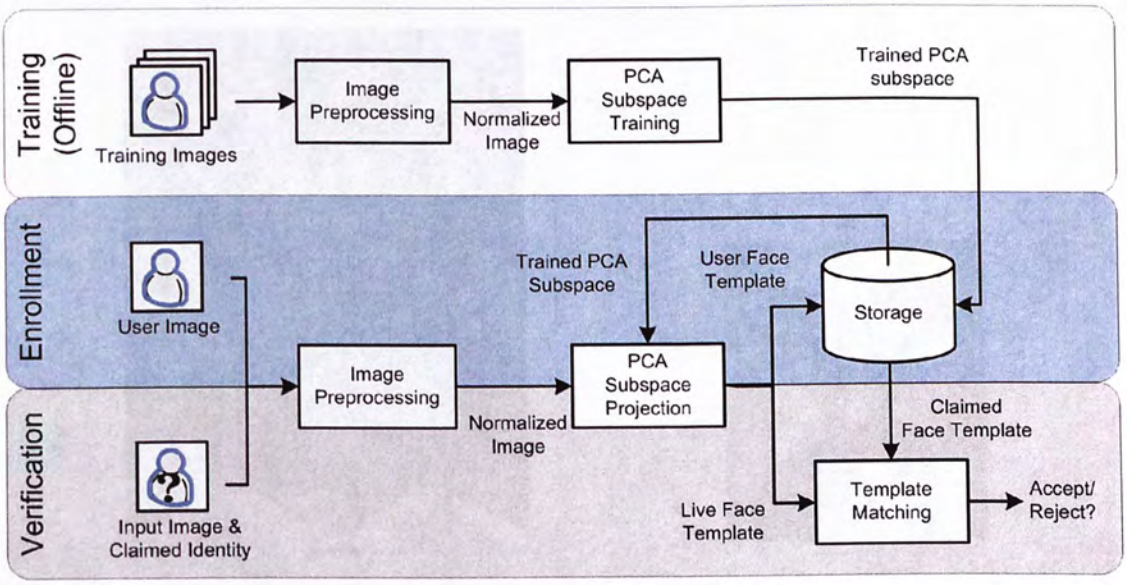

Figure 13 Major stages of PCA face authentication

4.2 System Overview

4.2.1 Image Preprocessing

In the image preprocessing stage, all input face images are normalized to reduce the variation among them. The normalization routine performs geometric normalization, masking, histogram equalization and pixel normalization [38, 39, 40] on the face image. Figure 14 shows a face image before and after normalization. Dimensions of the face images are reduced from 256 x 384 to 130 x 150 in this stage. After the preprocessing stage, the normalized face image is then passed on to the main PCA algorithm for training, enrollment or verification depending on the particular scenario.

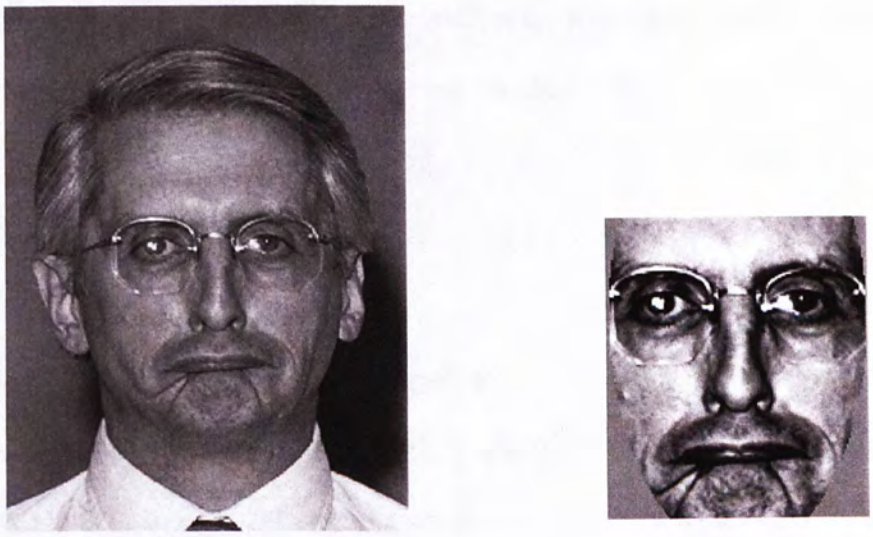

Figure 14 Original (left) and normalized (right) face image

4.2.2 PCA Subspace Training

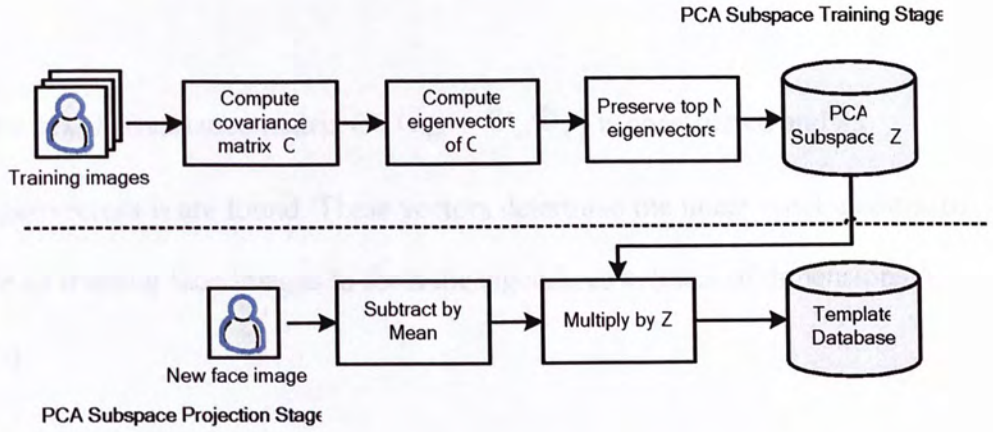

Figure 15 PCA subspace training and projection

As shown in Figure 15, PCA consists of two stages – Subspace Training and Projection. Training must be done before the system can be used. In a training scenario, training face images are used to build a subspace which efficiently preserves distinguishing features of face images. Then in an enrollment or verification scenario, new face images are projected to this trained subspace and form face templates which will be used for comparison.

During Subspace training, the rows of an $N_1 \times N_2$ training image are first concatenated into a one dimensional image vector. Let the training face image vectors be $F_1, F_2, F_3 \ldots F_M$. Each face vector differs from the mean by the vector Φ_i:

$$\Phi_i = F_i - \Psi \quad \text{where} \quad \Psi = \frac{1}{M}\sum_{n=1}^{M} F_n \tag{1}$$

The MxM covariance matrix C ($C_{mn} = \Phi_m \Phi_n$) is constructed and its eigenvectors v_l are found. These vectors determine the linear combinations of the M training face images to form the eigenfaces u_l (each of dimensions N_1 x N_2):

$$u_l = \sum_{k=1}^{M} v_{lk} \Phi_k \qquad \text{where} \quad l = 1,\ldots,M \tag{2}$$

In practice, only a subset of these eigenfaces ($k = 1,\ldots,N$ where $N << M$) is retained to form a transformation matrix Z which is used in the PCA projection stage. Only those eigenfaces which account for the most significant variations (principal components) are used in the construction. Figure 16 shows some examples of eigenfaces. The PCA subspace need only be trained once.

Figure 16 Eigenfaces (Principal Components)

4.2.3 PCA Subspace Projection

During PCA projection, a new face image vector F is first subtracted by the mean (Ψ) found in training and then multiplied by the transformation matrix Z. F is essentially projected to a point in the PCA subspace:

$$\omega_k = Z(F - \Psi) \quad \text{where} \quad Z = u_k^T \tag{3}$$

In this PCA subspace, the correlations among the projected images are minimized in order to facilitate easier classification[13]. The projected image (ω_k) is then saved as the face template of the corresponding user for future matching.

4.2.4 Template Matching

Template matching is done in the last stage of a verification scenario. After a live face image is preprocessed and projected, the live face template is then compared with the claimed user's face template. The comparison results in a similarity score which is in turn compared with the system threshold. If the score is higher than the threshold, the user is accepted; otherwise he or she will be rejected by the system. Mahalinobis Cosine distance [38] is used as the similarity measure in our face authentication system. Mahalinobis Cosine is the

cosine of the angle between two face templates after they are normalized by the variance estimates. For image vectors u and v with corresponding projections m and n in Mahalinobis space, the Mahalinobis distance is defined as:

$$D_{MahCosine}(u,v) = \frac{|m||n|\cos(\theta_{mn})}{|m||n|} \quad (4)$$

4.3 Optimization using Fixed-point Arithmetic

4.3.1 Profiling Analysis

As discussed in Section 3.3, profiling analysis must first be carried out to locate the bottlenecks in the face authentication system. Since the PCA subspace can be trained on a PC in an offline fashion, only the image preprocessing, PCA subspace projection and template matching stages are analyzed.

In our experiments, an open-source tool called GNU profiler (*gprof*) [41] is used for profiling. *gprof* can monitor program statistics such as total execution time and percentage of execution time taken by each function, which are useful for identifying and pinpointing the bottlenecks of a system. To analyze the execution time, 170 images selected from the FERET face image database (Section 1.1.4) are preprocessed, projected and matched. The execution time of these 170 authentication sessions are averaged and shown in Table 4. A detailed

description of the hardware testing platform will be given in Section 4.4.1.

	Execution Time / Percentage
Image Preprocessing	6.877 sec (22.7%)
PCA Projection	22.933 sec (75.8%)
Template Matching	0.425 sec (1.5%)
Total	30.235 sec (100%)

Table 4 Breakdown of execution time of one authentication session

As shown in Table 4, the PCA Projection stage alone takes up over 75% of the execution time of a typical authentication session; hence optimization effort should be focused on this stage. Experimental results show that matrix multiplication is the bottleneck of PCA projection and accounts for over 92% of the execution time. The primitive operations of matrix multiplications are floating point multiplications, which are extremely slow on mobile processors (Section 1.1.6). To circumvent the slow floating point multiplication bottleneck, fixed point arithmetic[42], using only integer operations, is used to optimize the face authentication system. All floating point variables and operations in the system are replaced with their fixed point counterparts.

4.3.2 Fixed-point Representation

A fixed point variable is implemented using a 32-bit integer (built-in C type "int"). The fixed point representation consists of three parts: sign bit, integer bits and fraction bits, as illustrated in Figure 17. The number of bits assigned to

the integer part is called Integer Word Length (IWL). Similarly, the number of fraction bits is called Fractional Word Length (FWL). IWL determines the largest possible range that can be represented by a fixed-point number, while FWL determines the precision. For instance, a real number, -1.875 in decimal (-1.111 in binary) is represented as -122880 in decimal (-11110000000000000 in binary) in fixed point format.

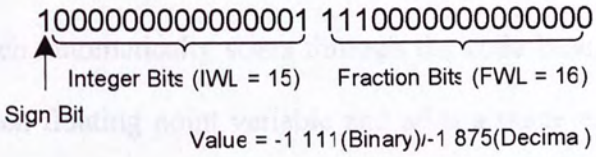

Figure 17 Data representation of a fixed-point number

4.3.3 Range Estimation

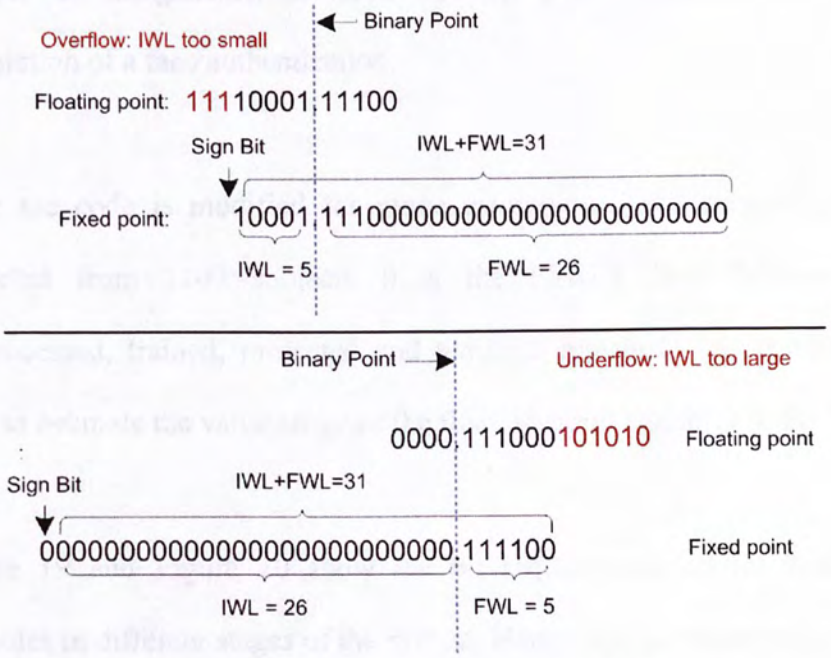

Figure 18 Examples of overflow and underflow

As shown in Figure 18, the fixed-point representation may suffer from overflow or underflow if the IWL is inappropriate. To prevent overflow or underflow, care must be taken when selecting the location of the fixed point.

Range estimation is done to determine the range and precision required by different stages of the face authentication system. A set of Perl scripts is developed which automatically scans through the code base, assigns a unique identifier to each floating point variable and adds a range estimation function call after each value assignment. The range estimation function receives the identifier and the updated value of the floating point variable, and then logs the assignment value for further analysis. The maximum absolute value and the number of assignments of each floating point variables are found on completion of a face authentication.

After the code is modified for range estimation, 3368 frontal face images collected from 1209 subjects from the FERET face database [5] are preprocessed, trained, projected and template matched. The results are then used to estimate the value range of the floating-point variables in the code.

Figure 19 and Figure 20 show the bit requirements of all floating point variables in different stages of the system. Here, bit requirement of a variable is defined as the minimum number of bits required to represent the maximum

absolute value ever assigned to the variable. Range estimation results show that the bit requirement of the image preprocessing stage is much larger than the PCA projection and template matching stages. To safely accommodate the largest floating point value while retaining sufficient accuracy, two IWLs are chosen. For Image preprocessing, IWL1 = 15 is chosen and for PCA projection and template matching, IWL2 = 9. The FWLs are chosen accordingly (i.e. FWL1=32 - IWL1-1=16, FWL2=32-IWL2-1=22). By using two stage-specific IWL instead of a system-wide, fixed IWL[32], the largely different value ranges of different stages can be accommodated.

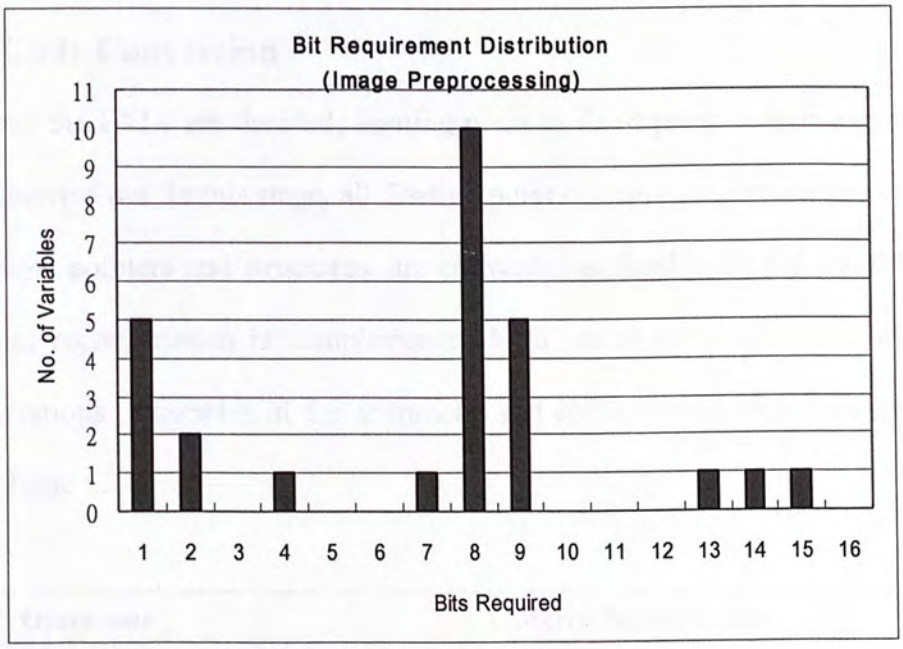

Figure 19 Bit requirement of Image Preprocessing Stage

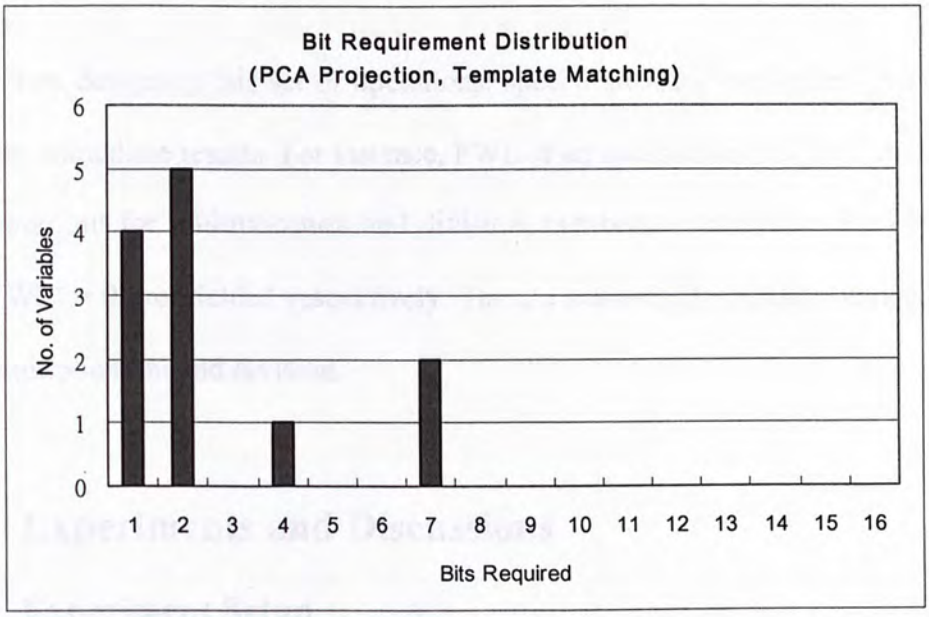

Figure 20 Bit requirement of PCA Projection and Template Matching Stage

4.3.4 Code Conversion

Once the IWLs are decided, floating point to fixed point code conversion can be carried out. In this stage, all floating point constants and variables, including arrays, pointers and structures, are converted to fixed point format. The fixed point representation is complemented by a set of conversion and arithmetic operations. Examples of the arithmetic and conversion routines are illustrated in Table 5.

Operations	C Macro Implementation	
Multiplication	#define fixMul(x, y)	((int) ((((long long)(x)) * (y)) >>FWL))
Division	#define fixDiv(x, y)	((int) ((((long long)(x)) << FWL) / y)))
Fixed point to Floating Point	#define fix2Double(x)	(((double)(x)) *2^{-FWL})
Floating Point to Fixed Point	#define double2Fix(x)	((fixed) ((x)* 2^{FWL}))

Table 5 C macro definitions of fixed point operations

When designing this set of operations, special attention was taken to cater for the immediate results. For instance, FWL of an addition/subtraction result is the same but for multiplication and division, numbers with FWL'= 2 · FWL and FWL' = 0 are yielded respectively. Thus, a scale-shift operation is needed for multiplication and division.

4.4 Experiments and Discussions

4.4.1 Experiment Setup

Experiments were carried out to measure the effect of optimization using fixed-point arithmetic. The execution time, space requirement and verification accuracy of the baseline and optimized system are compared. A development board for embedded system is used for evaluation. Table 6 shows its specification. From simplicity, we will refer to this platform as *Mobile*.

Processor	Intel XScale PXA255 400Mhz
Memory	64MB 100Mhz SDRAM
Storage	32MB flash ROM
OS	Embedded Linux (kernel version 2.4.19)
Compiler	arm-linux-gcc [43]
Compile Options	-O2 –mtune=xscale

Table 6 Specification of the evaluation system

All face images used in the experiments were selected from the FERET face database (Section 1.1.4). In all of the following experiments, the PCA subspace used is trained using 1194 face images. The first 358 principal eigenvectors are preserved.

4.4.2 Execution Time

We measured the difference in execution speed between the baseline and optimized systems. Measurements were taken by averaging the execution time for 120 authentication sessions.

	Baseline	Optimized	Reduction
I. Image Preprocessing	6.88 s	0.832 s	6.05s
II. PCA Projection	22.9 s	0.492 s	22.4s
III. Template Matching	0.425 s	0.00833 s	0.417s
Total	30.2 s	1.33s	28.9s

Table 7 Execution time of the baseline system and the optimized code

As seen from Table 7, a reduction of 22 seconds is observed for the PCA projection stage, 46 times faster than the baseline system. The image preprocessing speed is also improved by six times. Although the reduction in execution time for the template matching stage is comparatively insignificant to a verification scenario, it can be crucial to identification scenarios in which the number of matching (and hence projection) performed is directly proportional to the number of registered users in the database. Overall, the authentication

time for PCA now takes slightly more than one second, meeting the real time requirement.

4.4.3 Space Requirement

An added benefit of using fixed point representation is the reduction in storage. Table 8 shows the storage requirement of the PCA subspace and face templates used in our system. Since fixed variables (32bit integers) are used instead of floating point ones (64bit doubles), the storage requirement is halved. For mobile devices which have limited amounts of memory and storage space, this reduction can be beneficial in reducing the system runtime footprint as well as the storage requirement. As the loading time of the PCA subspace dominates the initialization time of the face authentication system, the decrease in size of the PCA subspace means much shorter setup time, as shown in Table 8.

	Baseline	Optimized	Reduction
Size of PCA Subspace	45Mb	23Mb	95.6%
Size of Face Template	2.3Kb	1.2Kb	91.6%
PCA Subspace loading time	32secs	16secs	100%

Table 8 Reduction in space requirement and its impact on loading time

4.4.4 Verification Accuracy

To investigate the effect of optimization on the verification accuracy, the FERET face database and evaluation protocol were used. (Section 1.1.4). A

total of 3307 frontal face images collected from 1196 subjects were used in the experiments. 1196 images were selected to form the gallery and the remaining images are separated into four probe sets (**FB**, **dup1**, **fc** and **dup2**). For each test, the probe images were matched against the gallery images in a round robin fashion. Images from the same subject were matched to calculate the False Rejection Rate (FRR). Match results of images from different subjects were matched to calculate the False Acceptance Rate (FAR). More details about the face images and probe sets can be found in Section 1.1.4.

Figure 21 to Figure 24 show the Receiver Operating Characteristic (ROC) curves of the optimized code and the baseline system for different probe sets. The overlapping curves indicate that verification accuracies of both systems are essentially identical.

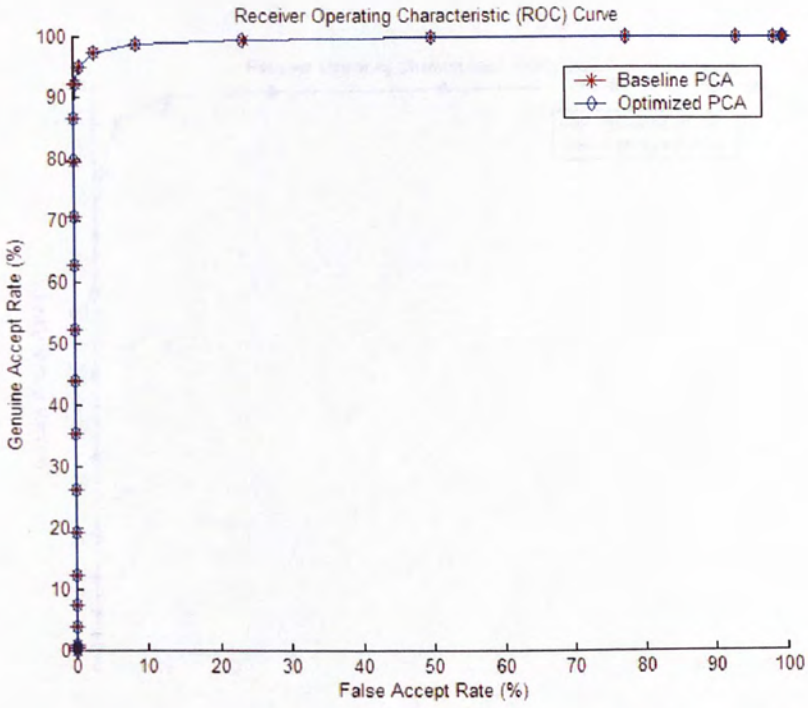

Figure 21 ROC Curves of PCA (**FB** probe set)

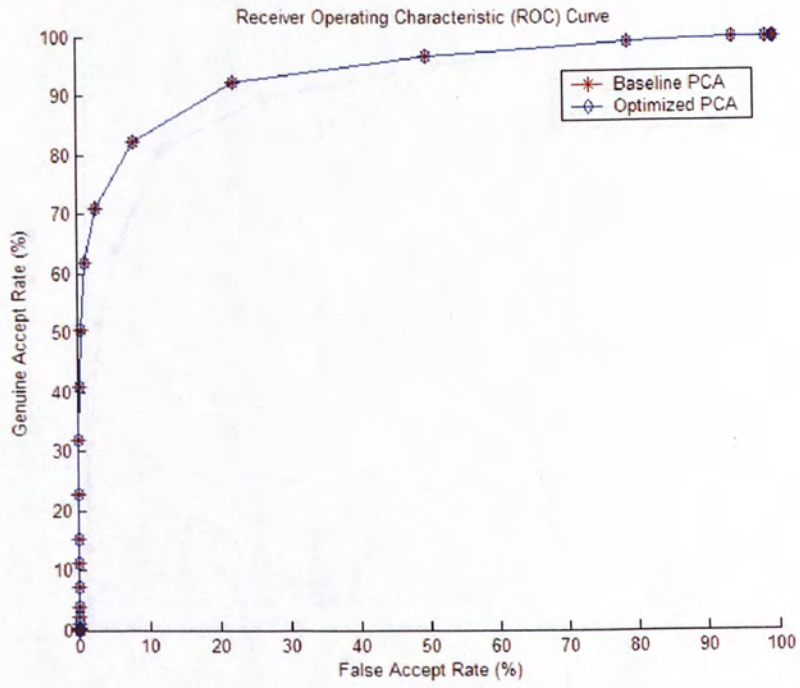

Figure 22 ROC Curves of PCA (**dup1** probe set)

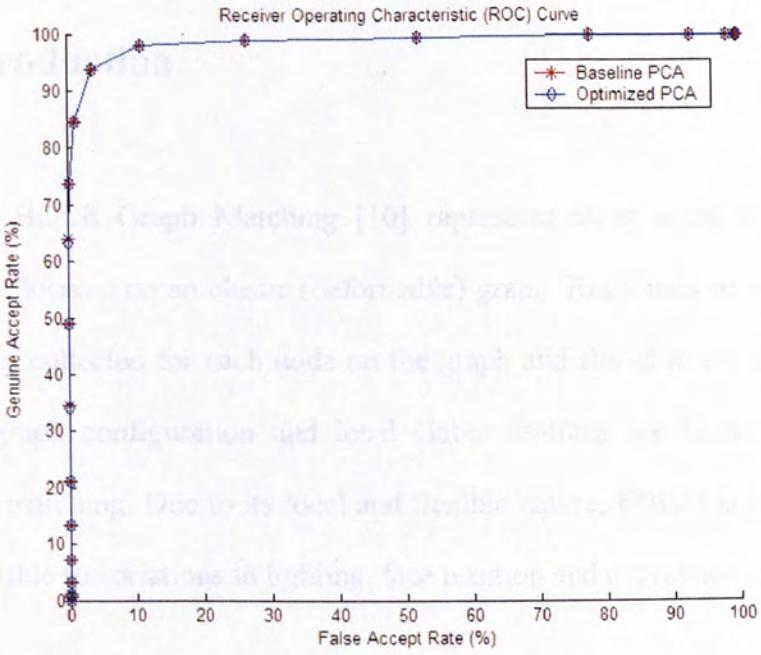

Figure 23 ROC Curves of PCA (**fc** probe set)

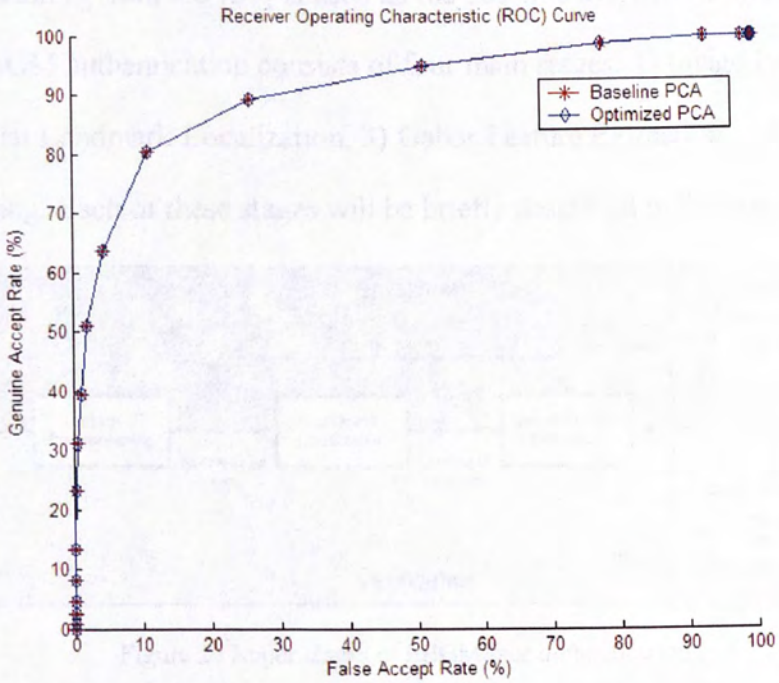

Figure 24 ROC Curves of PCA (**dup2** probe set)

5. Real Time Elastic Bunch Graph Matching

5.1 Introduction

Elastic Bunch Graph Matching [10] represents faces using a set of local features located on an elastic (deformable) graph. Responses to a set of Gabor filters is collected for each node on the graph and stored in the face template. Both graph configuration and local Gabor features are taken into account during matching. Due to its local and flexible nature, EBGM is in general less susceptible to variations in lighting, face position and expression.

In our experiments, the EBGM implementation of the CSU Face Identification Evaluation System 5.0 [37] is used as the baseline system. As shown in Figure 25, EBGM authentication consists of four main stages: 1) Image Preprocessing, 2) Facial Landmark Localization, 3) Gabor Feature Extraction and 4) Template Matching. Each of these stages will be briefly described in Section 5.2.

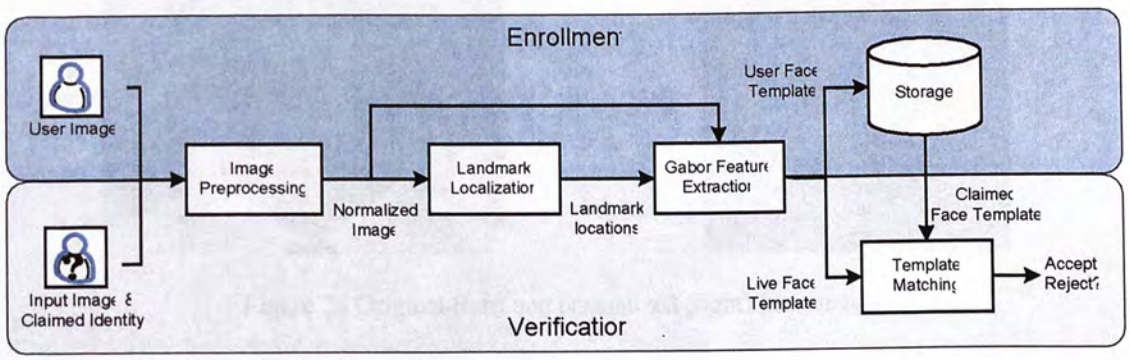

Figure 25 Major stages of EBGM face authentication

5.2 System Overview

5.2.1 Image Preprocessing

In this stage, face images are normalized to reduce the variations among them. The normalization routine performs mean centering, edge smoothing, geometric normalization, masking and pixel normalization on the face image[44]. Figure 26 shows a face image before and after normalization. Dimensions of the face images are reduced from 256 x 384 to 128 x 128 in this stage. After the preprocessing stage, the normalized face image is then passed on to the main EBGM algorithm for landmark localization and feature extraction.

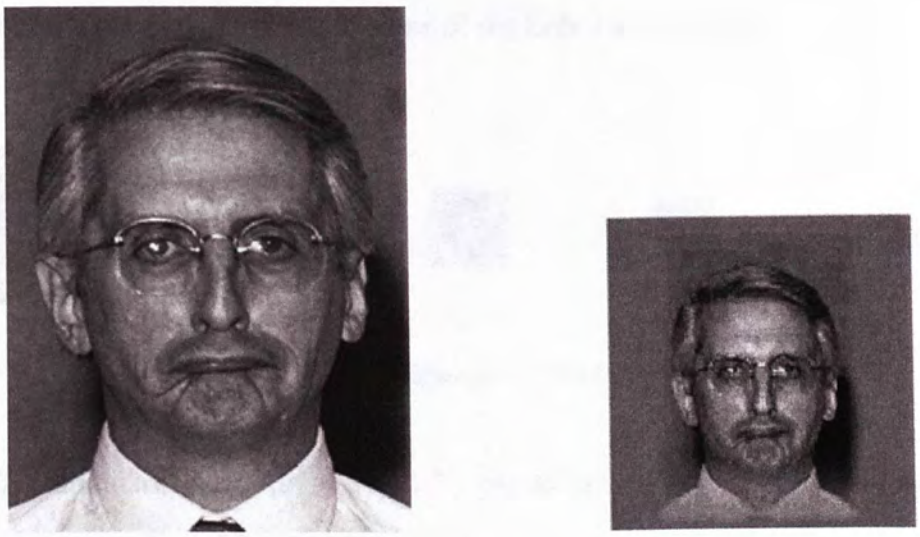

Figure 26 Original (left) and normalized (right) face image

5.2.2 Landmark Localization

In this stage, locations of facial landmarks such as eyes, nose and mouth are found. The landmark localization stage contains two steps. First, rough estimates of the landmark locations are obtained based on known landmarks, such as the eyes. Then this estimate is refined by Gabor jet comparisons.

A Gabor jet refers to a set of Gabor wavelet convolutions values obtained at a specific point. By varying the wavelet function parameters, frequency characteristics about the local image region around the extraction point can be captured. For the EBGM algorithm, 40 complex Gabor wavelets (or 80 real/imaginary pairs) of different sizes, wavelengths and orientations are used. Figure 27 shows examples of some of the Gabor filters used.

Figure 27 Gabor filters of different wavelengths and orientations

The landmark location is refined by extracting a novel jet from the estimated location of the landmark in the novel image. Then a model jet (the most similar one) is selected from a data structure called the bunch graph. Figure 28 shows a graphical illustration of the bunch graph, containing collections of sample jets at each landmark location, all extracted from training images. Using the phase

information stored in the novel and model jets, the displacement of the novel jet from the true landmark location can be calculated. After all the landmark locations are found, they are passed on to the feature extraction stage.

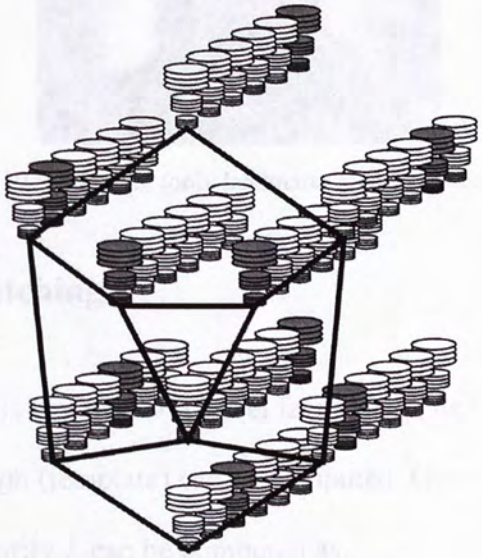

Figure 28 The bunch graph[10]

5.2.3 Feature Extraction

In the feature extraction stage, Gabor jets are extracted from the normalized face image at the landmark locations found in the previous stage. These Gabor features, together with the locations, are stored in a structure called face graph. This face graph is stored in the template database for future recognition use and the original face image is then discarded. Figure 29 shows a sample face graph.

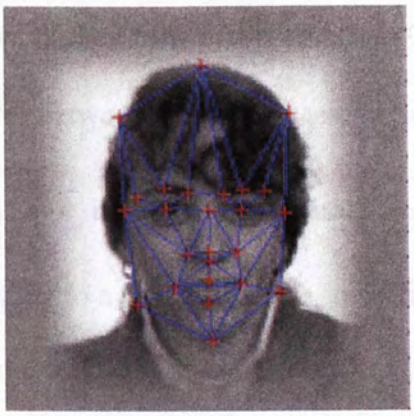

Figure 29 Face graph (only landmarks are shown here)[44]

5.2.4 Template Matching

After a face graph is created for a novel face image, its similarity with another registered face graph (template) can be computed. Given two face graphs, G and G', their similarity L can be computed as:

$$L(G,G') = \frac{1}{M}\sum_{i=0}^{M} S_D(J_i, J_i') \tag{5}$$

where M is the number of landmarks, and J_i and J_i' are jets from the i^{th} landmarks of graphs G and G'. S_D is a similarity measure for Gabor jets defined by:

$$S_D(J,J') = \frac{\sum_{j=0}^{N} a_j a_j' \cos(\Phi_j - (\Phi'_j + \vec{d}\cdot\vec{k}_j))}{\sqrt{\sum_{j=0}^{N} a_j^{\,2} \sum_{j=0}^{N} a_j'^{\,2}}} \tag{6}$$

where N is the number of Gabor filters, and a_j (a_j') and Φ_j (Φ_j') are the

magnitudes and phases of the j^{th} filter response from Gabor jets $J(J')$. $\vec{d}$ is the estimated displacement between the two jets and $\vec{k}_j$ is the spatial frequency of the j^{th} filter. The term $\vec{d} \cdot \vec{k}_j$ is used to compensate the phase shifts caused by the displacement, leading to a phase sensitive similarity function. A more detailed discussion on the EBGM algorithm can be found in[3].

5.3 Optimization Overview

The optimization process of EBGM is much more complicated than PCA. For PCA, fixed-point arithmetic (Section 4.3) alone is sufficient to bring real time performance but for EBGM, other techniques such as memory optimization are needed. For clarity, all optimization techniques adopted and the order they are implemented are summarized in Figure 30. As one can see clearly from the diagram, optimization efforts start out with fixed-point arithmetic and pre-computation, but soon turn to memory optimization techniques. This shift in focus is due to the fact that optimization is a dynamic process; program behaviour can switch from computation-bound to memory-bound and vice versa. Optimization strategies must be sensitive to these changes and adjust accordingly, otherwise efforts may be wasted in premature optimization. This is exactly the same reason why optimization techniques are applied in the specific order shown in the diagram. To cater for the changing nature of the program, various computation optimization and memory optimization techniques have

been developed and will be discussed in Section 5.3.1 and 5.2.2 respectively. Details of the eight optimization techniques will be given in Section 5.4.1-5.4.8.

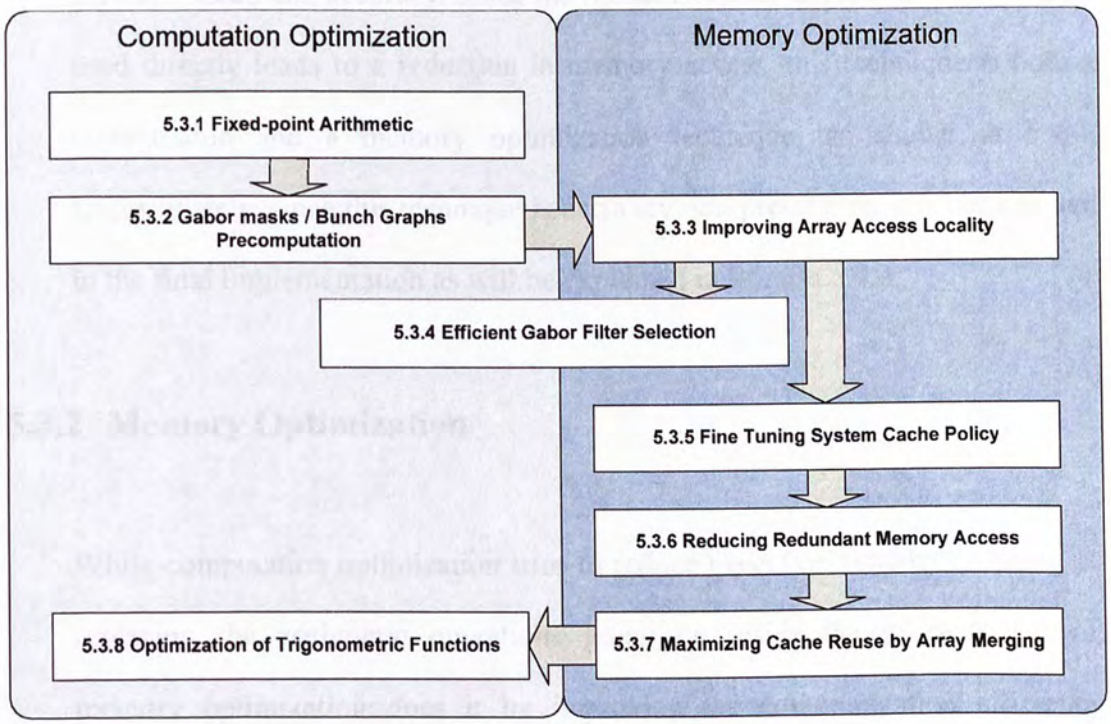

Figure 30 EBGM optimization flowchart

5.3.1 Computation Optimization

Besides the fixed-point arithmetic and pre-computation techniques already introduced in PCA implementation, an important addition is the optimization of trigonometric functions. As discussed in Section 1.1.6, trigonometric functions can be several hundred times slower on mobile devices than on PCs. In Section 5.4.8, we use table lookup to improve their performance.

In Section 5.4.4, we attempt to reduce the number of computations by using only a subset of the original Gabor filter sets and investigate the tradeoff between speed and accuracy. Since the reduction in the number of Gabor filters used directly leads to a reduction in memory access, this technique is both a computation and a memory optimization technique, as shown in 5.4.4. Unfortunately, since this technique is accuracy non-preserving, it is not adopted in the final implementation as will be explained in Section 5.4.4.

5.3.2 Memory Optimization

While computation optimization tries to reduce execution time by reducing or replacing the arithmetic operations going on inside the processing core, memory optimization does it by improving the communication efficiency between the core, on-chip cache and off-chip memory. This requires a thorough understanding of the memory hierarchy of the target processor as well as the memory access behaviour of the application to be optimized.

Here we will give a brief description of the Intel XScale core[45], a high performance and low power processor specifically designed for mobile applications. It will be used in all our testing and experiments. Figure 31 shows its internal architecture. XScale uses separate caches for instruction and data, both 32-way set associative and of size 32Kbytes. These on-chip caches, while

relatively small in size, can be accessed in the same clock speed (i.e. 400 MHz) as the execution core. Larger but slower main memory (100 MHz SDRAM) can be accessed via the Memory Management Unit (MMU) and the core memory bus. The large difference in clock rate means that a cache miss will result in undesirably long memory access latency. Hence improving cache performance is the main goal of memory optimization.

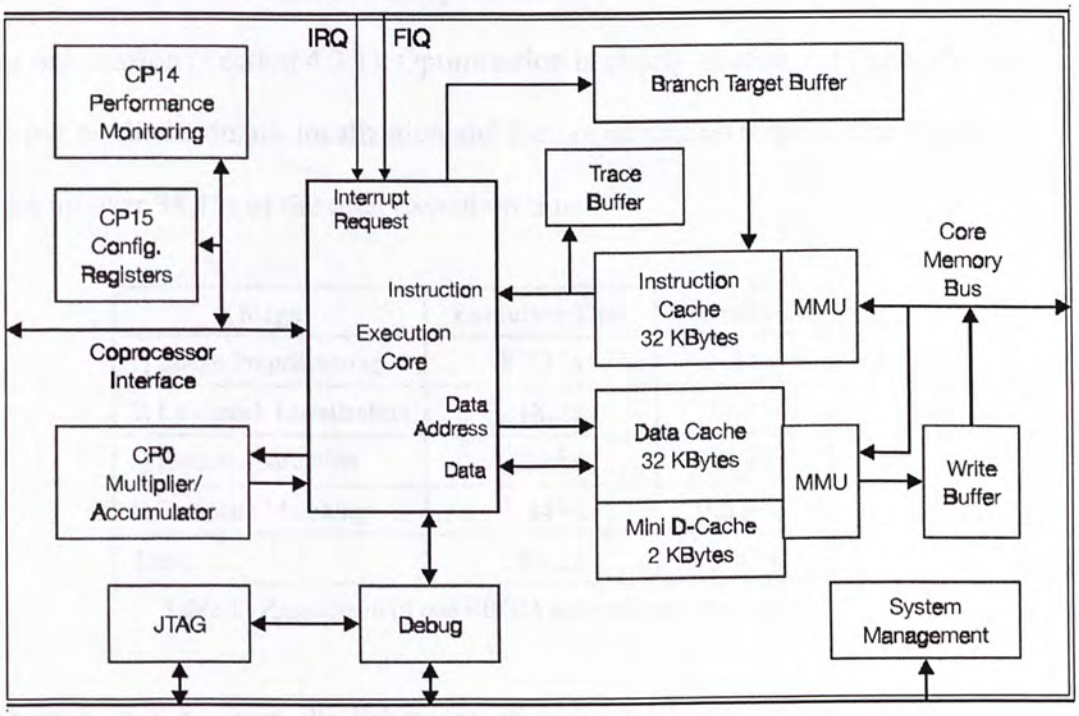

Figure 31 Intel XScale PXA255 Processor Block Diagram [6]

Cache performance is determined by factors such as cache size and associativity, block size, block replacement policy and cache write policy[46].

5.4.1 Fixed-point Arithmetic

Pre-Profiling and Hotspot Location

Table 11 shows the execution time of one EBGM authentication session on *Mobile*. It is obvious that the execution time is far from satisfactory – over nine minutes on average. Note that the un-optimized implementation of EBGM is also 18 times slower than the un-optimized PCA, which takes only 30 seconds for one session (Section 4.3.1). Optimization is clearly needed and focus should be put on the landmark localization and feature extraction stage, which together take up over 98.1% of the total execution time.

Stage	Execution Time	Percentage
1) Image Preprocessing	8.335s	1.51%
2) Landmark Localization	218.2s	39.4%
3) Feature Extraction	324.8s	58.7%
4) Template Matching	1.849s	0.334%
Total	553.2s	100%

Table 11 Breakdown of one EBGM authentication session

To study and measure the behaviour of each EBGM stage, they are profiled separately. Table 12 shows the *gprof* profiling results of the image preprocessing stage. On *Mobile*, functions that involve a large amount of floating point operations (1, 2, 3, and 5) take up over 70% of execution time. Similar results were collected on *Desktop*, except that the matrix multiplication function takes up only one third of the execution time as in *Mobile*. As matrix

multiplication involves a large amount of memory access, the differences may be due to a difference in cache size or memory access efficiency.

Function Name	Description	Mobile	Desktop
1. ZeroMeanOneStdDevMasked	Pixel normalization	38.96%	37.22%
2. multiplyMatrix	Matrix multiplication	**17.77%**	**5.65%**
3. interpLinear	Linear interpolation	15.16%	18.85%
4. writePGMImage	Write output image	9.00%	7.43%
5. transformImage	Geometric transformation	6.88%	4.97%
6. smoothImageEdge	Edge smoothing	4.25%	5.04%

Table 12 Function Profile (Image preprocessing)

Table 13 shows the result for the landmark localization stage. Here we see a large difference between the results collected from *Mobile* and *Desktop*. Distance estimation, which occupies no more than 4% of execution time on *Desktop*, takes up over 43% of time on *Mobile*. It turns out that this function contains trigonometric functions besides simple floating point arithmetic. As mentioned in Section 1.1.6, trigonometric functions can be a serious bottleneck for mobile devices.

The image element access takes up more than 15% of the time on *Mobile*, while it is insignificant on *Desktop*. This may be due to slower memory access on *Mobile*. Similar observations can be found in the feature extraction stage. (See Table 14).

Function Name	Description	Mobile	Desktop
1. DEPredictiveIter	Distance estimation	**43.06%**	**3.27%**
2. convolvePoint	Filter convolution	40.43%	95.91%
3. ie	Image element access	**15.38%**	**~0.0%**

Table 13 Function Profile (Landmark localization)

Function Name	Description	Mobile	Desktop
1. convolvePoint	Filter convolution	68.03%	99.54%
2. ie	Image element access	**30.50%**	**~0.0%**

Table 14 Function Profile (Feature extraction)

Table 15 shows that distance estimation dominates the execution time of the template matching stage. Given that the only task in template matching is to calculate the distance between pairs of input images, the result is reasonable.

Function Name	Description	Mobile	Desktop
1. DEPredictiveIter	Distance estimation	99.73%	99.62%

Table 15 Function Profile (Template matching)

To summarize, the hotspots of the EBGM algorithm are floating point arithmetic, memory access and trigonometric functions. Taking into account the dominance of the landmark localization and feature extraction stage, the common bottleneck is floating point arithmetic. Hence in this section, we will reuse the fixed-point arithmetic technique as in PCA optimization (Section 4.3).

Optimization

As discussed in Section 4.3, range estimation must first be carried out to estimate the Integer Word Length (IWL) requirement. Figure 32 shows the minimum IWL for all four EBGM stages. It is found that the range of all four stages can be safely accommodated using an IWL of 13. After the IWL is chosen, code conversion is done following the steps outlined in Section 4.3.4.

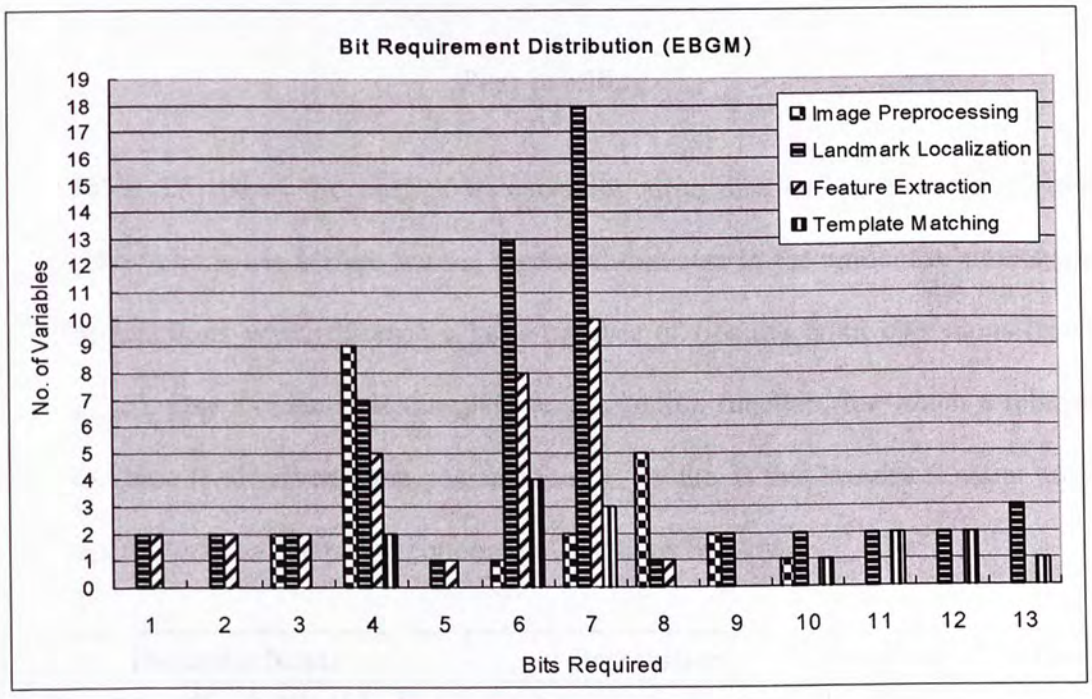

Figure 32 Bit requirements of different EBGM stages.

Table 16 shows the improvement in execution time after fixed-point arithmetic is used. As predicted, stages 2 and 3, which depend heavily on floating point calculations, show dramatic improvement as their execution times are now reduced by around 14 and 11 times respectively. Stage 4, on the other hand,

shows a moderate improvement. Overall, the execution time reduces by a factor of 11.

Stage	Baseline*	Fixed-point
1) Image Preprocessing	8.335s	1.93s
2) Landmark Localization	218.2s	15.07s
3) Feature Extraction	324.8s	29.45s
4) Template Matching	1.849s	0.2812s
Total	553.2s	46.73s

Table 16 Breakdown of one EBGM authentication session (Fixed-point)

Post-profiling

Table 17 shows the change in execution time distribution after fixed-point arithmetic is used. Here we see a general decrease in the amount of time spent in functions which include a large number of floating point operations (rows 2,3,5). One exception is the pixel normalization function, for which a relative increase is observed. One possible reason for this is that besides floating point operations, it also uses trigonometric functions heavily.

Function Name	Description	Before	After
1. ZeroMeanOneStdDevMasked	Pixel normalization	38.96%	43.61%
2. multiplyMatrix	Matrix multiplication	17.77%	4.14%
3. interpLinear	Linear interpolation	15.16%	0.92%
4. writePGMImage	Write output image	9.00%	5.20%
5. transformImage	Geometric transformation	6.88%	1.44%
6. smoothImageEdge	Edge smoothing	4.25%	18.23%

Table 17 Function Profile (Image preprocessing)

For landmark localization and feature extraction, a significant portion of execution time is relocated to image element access, as shown in Table 18 and Table 19. This suggests a shift in program behaviour from computation-bound to memory-bound. Efficient memory access may be the key to further optimization.

Function Name	Description	Before	After
1. DEPredictiveIter	Distance estimation	43.06%	10.17%
2. convolvePoint	Filter convolution	40.43%	36.33%
3. ie	Image element access	15.38%	**49.44%**

Table 18 Function Profile (Landmark localization)

Function Name	Description	Before	After
1. convolvePoint	Filter convolution	68.03%	44.56%
2. ie	Image element access	30.50%	**53.35%**

Table 19 Function Profile (Feature extraction)

For template matching, distance estimation remains the only dominant time consumer, as shown in Table 20. But since template matching takes up a mere 1% of execution time, its optimization is not of much significance at this stage.

Function Name	Description	Before	After
1. DEPredictiveIter	Distance estimation	99.73%	94.81%

Table 20 Function Profile (Template matching)

To summarize, fixed-point arithmetic dramatically reduces the time spent in computation, and memory access arises as the new bottleneck of EBGM. Fewer

and more efficient memory access should be the new goal for optimization. In light of this, various memory optimization techniques will be studied in the following sections.

Besides memory optimization, some improvements can be done at a higher level. In the beginning, one single Gabor filter set is read and created during runtime for both the landmark localization and feature extraction stages. Since we use a predetermined set of filters, the filters can be pre-calculated and then preloaded at system startup. In addition, the bunch graph structure (Section 5.2.2) is extracted every time before the landmark localization stage. Assuming that the set of model images remains invariant, pre-calculation can also be applied. This will be covered in the next section.

5.4.2 Gabor Masks and Bunch Graphs Precomputation

Optimization

As suggested in the previous section, invariant input such as Gabor filters and bunch graphs can be pre-calculated and loaded at system startup. An extension was implemented for the pre-computation, storing and retrieval of the pre-calculated data. Modifications are done to the original implementation such that Gabor filters and bunch graphs were now loaded from file instead of generated on demand.

Table 21 shows the timing breakdown of one authentication session. P1 and P2 are the times required for Gabor masks/bunch graph building (before) and loading (after). If preloading is done at setup time, the execution time for one authentication session reduces to around 36s. This reduction in computation time comes at the expense of extra storage space. Table 22 shows the extra files generated for preloading. A total of 2.1Mb extra space is required, which is a reasonable tradeoff.

Stage	Before	After
P1) Build/load Gabor Masks	216.45s	1.15s
P2) Build/load Bunch Graph	631.93s	2.21s
1) Image Preprocessing	1.93s	1.93s
2) Landmark Localization	12.91s	12.91s
3) Feature Extraction	20.97s	20.97s
4) Template Matching	0.2812s	0.2812s
Total	884.5s	39.45s
Total (Preloaded)	36.09s	36.09s

Table 21 Breakdown of one EBGM authentication session (Preload)

Filename	Description	Number	Unit Size
GaborMaskBolme.params	Mask Parameters	1	4K
GaborMaskBolmeXX.fpi	Gabor masks	80	4K/8K/12K/24K
GaborMaskBolme.bunchgraph	Bunch Graph	1	4K
GaborMaskBolmeXX.jetbunch	Jet bunches	25	48K

Table 22 Extra space requirement for precomputation

Post-profiling

As the other stages are not affected by pre-computation, only the profiling results of landmark localization and feature extraction stages are discussed here. As shown in Table 23, the dominance of image element access further increases to over 60%. As for feature extraction, filter convolution and image element access remain the most time consuming parts, both involving large amount of memory accesses. In the next few sections, we will discuss the various memory optimization techniques used.

Function Name	Description	Before	After
1. DEPredictiveIter	Distance estimation	10.17%	1.82%
2. convolvePoint	Filter convolution	36.33%	32.84%
3. ie	Image element access	49.44%	61.1%

Table 23 Execution time breakdown (Landmark localization)

Function Name	Description	Before	After
1. convolvePoint	Filter convolution	44.56%	57.84%
2. ie	Image element access	53.35%	41.36%

Table 24 Execution time breakdown (Feature extraction)

5.4.3 Improving Array Access Efficiency using 1D array

As mentioned in 5.4.1 and 5.4.2, efficient memory access is the new optimization goal. In this section, three techniques are explored to improve the efficiency of image element access.

Optimization 1: Predetermining boundary conditions

The filter convolution function, *convolvePoint*, was implemented following the basic definition. The Gabor mask is first offset to a landmark on the face image, and then the sum-of-product of the overlapping pixels are calculated and returned as the filter response at the landmark position. In other words, *convolvePoint* consists mainly of additions, multiplications and access to the face image and Gabor masks. Unnecessary operations can be reduced by making use of the boundary conditions.

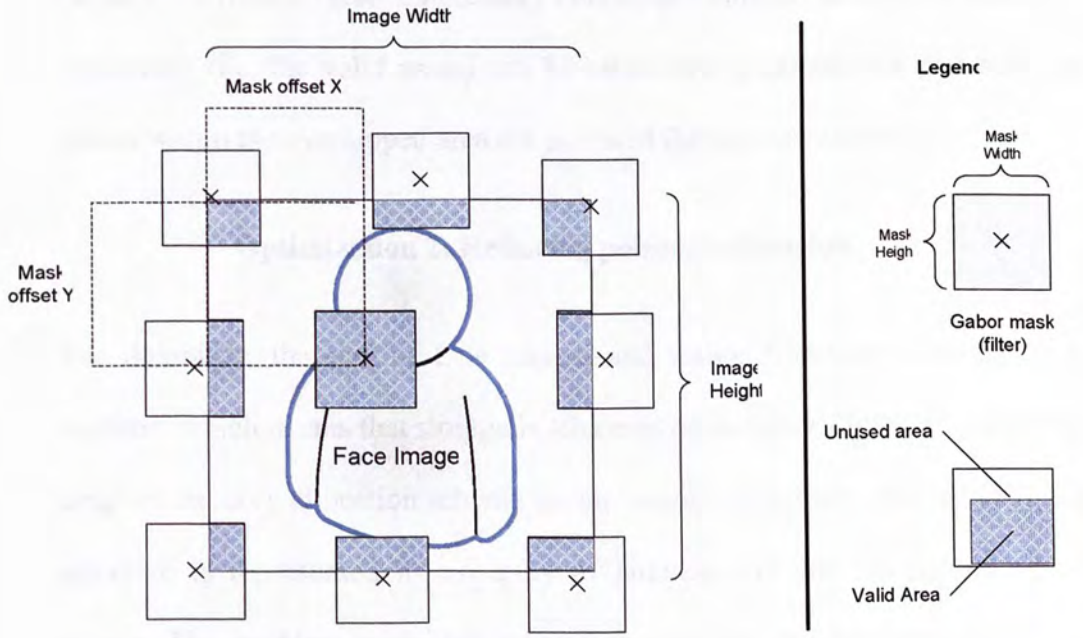

Figure 33 Boundary conditions for Gabor mask convolution

Figure 33 illustrates the boundary conditions that must be considered during Gabor filter convolution. As the filter response outside the face image is ignored and regarded as zero, only the overlapped areas between the offset

Gabor mask and the face image are valid. In the original implementation, this boundary check is carried out on a per pixel basis – each access to an image pixel is validated by checking the requested coordinates against the image dimensions. If an out-of-bounds condition occurs, a zero is returned. This approach has two problems. First, unnecessary checks are done even for pixels that lie within the boundaries. Since Gabor masks are stored and accessed in the same manner as images, the same problem exists for access to both. Secondly, multiplications for the invalid area are carried out even though the result must be zero. To reduce these unnecessary checks and multiplications, the boundary conditions (i.e. the valid areas) can be calculated in advance so that only the pixels within the overlapped area are accessed during convolution.

Optimization 2: Reducing pointer indirection

For flexibility, the size of face images and Gabor filters are determined at runtime, which means that storage is allocated on demand. Figure 34 shows the original memory allocation scheme for an image/Gabor filter. The logically 2D structure is represented by an array of pointers and one dimensional pixel arrays. The problem with this scheme is that two pointer indirections are needed to access one image pixel, creating a heavy burden on the memory system. To reduce this overhead, one 1D array is used to store the whole image (Figure 35). Rows of pixels are now packed sequentially, which means that nearby pixels can now be accessed with only one pointer indirection and one increment, effectively reducing the burden on the memory system.

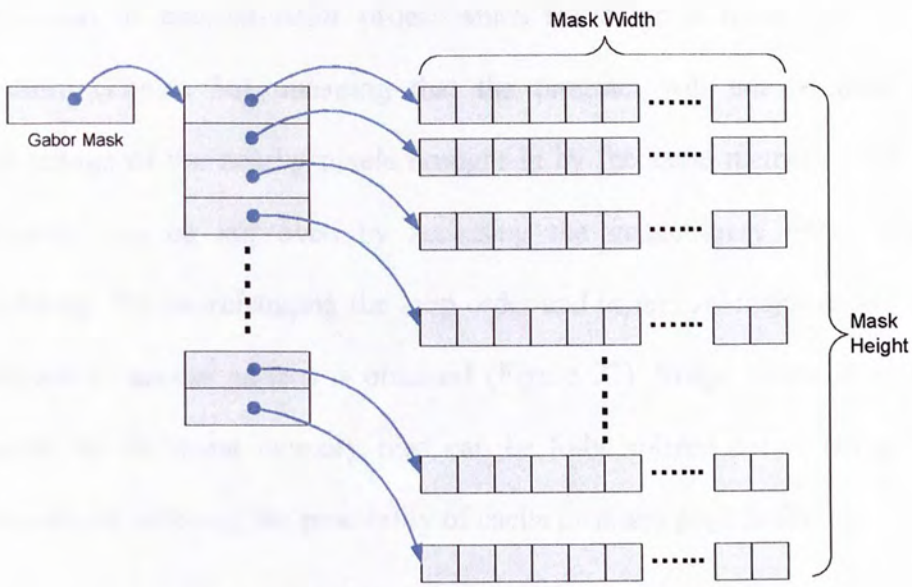

Figure 34 Image/mask stored in a 2D structure

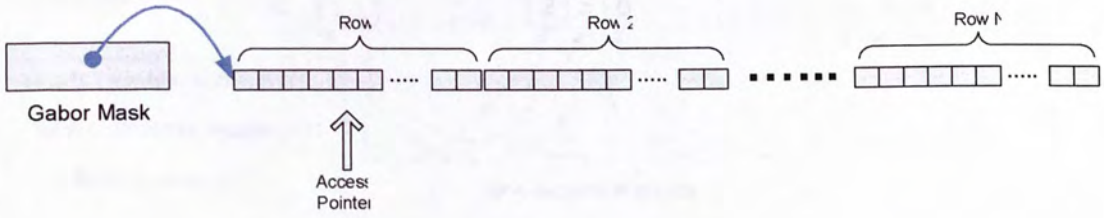

Figure 35 Image/mask stored in a 1D structure

Optimization 3: Improving access locality

With the 1D structure in place, memory access locality for image pixels can be further improved. In contrast to scientific languages such as FORTRAN, array in C language follows row-major ordering. This means that array elements adjacent to each other in memory differ in the second subscript instead of the first; 'B(5,10)' immediately follows 'B(5,9)', whereas with column-major ordering it would follow 'B(4,10)'. In the original implementation, arrays are

accessed in column-major order, which results in a non-sequential access pattern (Figure 36), meaning that the program will not be able to take advantage of the nearby pixels brought in by the same memory read. Access locality can be improved by accessing the image array using row-major ordering. By interchanging the loop order and using row-major access order, a sequential access pattern is obtained (Figure 37). Image pixels brought in to cache by the same memory read can be fully utilized before being evicted, essentially reducing the possibility of cache miss and page faults.

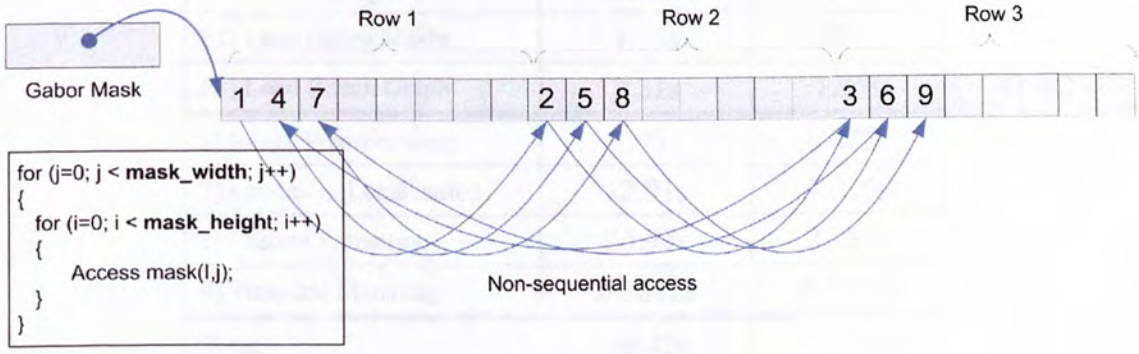

Figure 36 Original access pattern (column-major ordering)

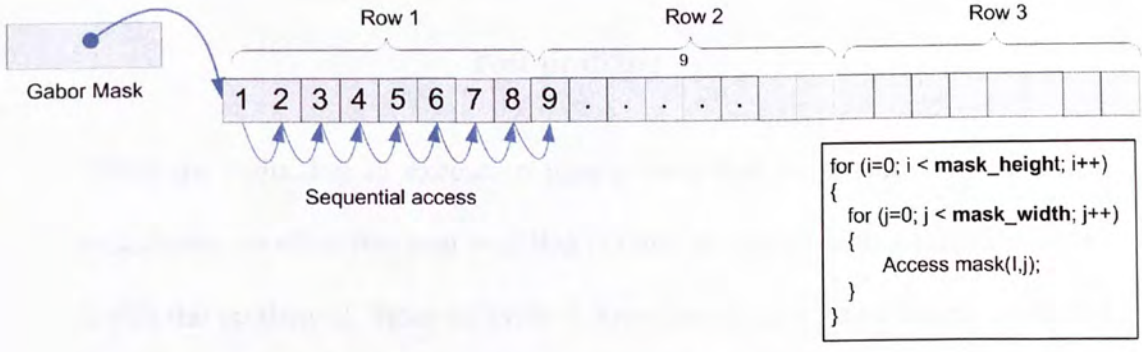

Figure 37 Optimized access pattern (row-major ordering)

Overall Results

The three optimization techniques are implemented and evaluated. Table 25 shows the improvement in execution time. Except the template matching stage, which contains no image access, there is a general reduction in execution time for all other stages. The improvement is especially significant for the feature extraction and landmark localization stages. The overall execution time for one authentication session is reduced by over three times to around 11 seconds.

Stage	Before	After
P1) Load Gabor Masks	1.15s	0.81s
P2) Load Bunch Graph	2.21s	1.95s
1) Image Preprocessing	1.93s	0.882s
2) Landmark Localization	12.91s	7.026s
3) Feature Extraction	20.97s	2.989s
4) Template Matching	0.2812s	0.2970s
Total	39.45s	13.954
Total (Preloaded)	36.09s	11.194

Table 25 Breakdown of execution time of one authentication session

Post-profiling

While the reduction in execution time proves that the memory optimization techniques are effective, post profiling is done to check whether memory access is still the bottleneck. Since individual boundary checks are no longer necessary, image access are not longer implemented as a function (ie). Hence instead of the function profiling option, the line profiling function of *gprof* is used for

analysis. The percentage of execution time spent in each statement rather than functions are recorded and output by this function.

Statement	Percentage
1. Filter element access	37.03%
2. Image element access	31.99%
3. Fixed-point multiplication	9.88%
4. Loop overhead	4.68%

Table 26 Line Profile (Landmark localization)

Statement	Percentage
1. Filter element access	26.90%
2. Image element access	23.42%
3. Fixed-point multiplication	6.95%
4. Loop overhead	3.70%

Table 27 Line Profile (Feature extraction)

Table 26 and Table 27 show the execution time breakdown for landmark localization and feature extraction. It is obvious that the two stages remain memory-bound, with over 60% of time spent in image or filter element access. Another observation is that the top four time consuming statements are identical for both stages, and they all reside in the filter convolution function (*convolvePoint*). Clearly, optimization efforts should continue to focus on memory optimization, with special emphasis put on filter convolutions by using a subset of filters.

The memory optimization techniques employed in this section mainly deal with individual pixel access. In the next few sections, focus will be put on optimizing access efficiency on a higher level by making use of the relationship between Gabor filter masks and across procedures. In the next section, we will focus on reducing the number of convolutions.

5.4.4 Efficient Gabor Filter Selection

As discussed in the previous section, filter convolution is the bottleneck of both stages. The time needed for filter convolution depends on the number of filters and filter sizes, which is in turn determined by the filter set selection. The Bolme set[44] is used in the original implementation and its configuration is shown in Table 28. There are a total of 80 real valued filters (eight orientations, five wavelengths and two phases), each generated using the general Gabor wavelet equation:

$$w(x,y;\theta,\varphi,\lambda,\sigma)=\exp\left[-\frac{1}{2\sigma^2}\left(x^2+y^2\right)\right]\cos\left(\frac{2\pi}{\lambda}x_\theta+\varphi\right) \qquad (7)$$

Parameter	Symbol	Values
Orientation	θ	{0,π/8, 2π/8, 3π/8, 4π/8, 5π/8, 6π/8, 7π/8}
Wavelength	λ	$\{4, 4\sqrt{2}, 8, 8\sqrt{2}, 16\}$
Phase	φ	{-π/4, π/4}
Gaussian Radius	σ	3*λ/4
Aspect Ratio	γ	1

Table 28 Bolme Gabor filter set

As feature extractors, Gabor filters capture the frequency-space properties of a confined area. When designing a set of Gabor filters, the orientations and wavelengths are chosen so that the resultant filters provide uniform and efficient coverage of the frequency space. The frequency coverage of a filter set is determined by the orientations and wavelengths chosen, and affects the filter number and filter size. In this section, we investigate the tradeoff between speed and accuracy when only a subset of the Bolme set is used. Table 29 shows the configuration of the four filter sets used in this study. By choosing different subsets of orientations and wavelengths, the number of filters varies from the original 80 to 24.

Set	Orientation (θ)	Wavelength (λ)	Filters
Bolme	{0,π/8, 2π/8, 3π/8, 4π/8, 5π/8, 6π/8, 7π/8}	$\{4, 4\sqrt{2}, 8, 8\sqrt{2}, 16\}$	80
Bolme_8d3f	{0,π/8, 2π/8, 3π/8, 4π/8, 5π/8, 6π/8, 7π/8}	{4, 8,16}	48
Bolme_4d5f	{0,π/8, 3π/8, 5π/8, 7π/8}	$\{4, 4\sqrt{2}, 8, 8\sqrt{2}, 16\}$	40
Bolme_4d3f	{0,π/8, 3π/8, 5π/8, 7π/8}	{4, 8,16}	24

Table 29 Filter set configurations

Results

Table 30 shows the authentication times using different filter sets. As expected, execution time reduces as number filters decreases. However, the reduction in timing must be justified by accuracy verification.

Set	2) Landmark Localization	3) Feature Extraction	4) Template Matching	Total
Bolme	7.026s	2.989s	0.2970s	10.312
Bolme_8d3f	5.003s	2.571s	0.1787s	7.753s
Bolme_4d5f	4.093s	1.949s	0.1481s	6.190s
Bolme_4d3f	1.464s	0.845s	0.0466s	2.356s

Table 30 Authentication time using different filter sets

Here, the verification accuracy is evaluated using the **FB** probe set following the FERET verification protocol[4]. Three sets of experiments were conducted. Experiment 1 investigated the effect of different filter schemes when they are used in both landmark localization and feature extraction. As the reduction in number of filters may affect the accuracy of the localization process, experiment 2 was performed to investigate the sole effect of the new filter sets on the feature extraction stage only. The localization is done using the original Bolme wavelet sets. Finally, the effect on localization was investigated in experiment 3. After the landmarks were located using the new filter schemes, the features are extracted using the original Bolme scheme.

The Equal Error Rates (EER) of different configurations are shown in Table 31. Here we see that the *Bolme_4d5f* and *Bolme_4d3f* sets cause a dramatic increase in the EER. This renders them unusable for any useful authentication. The *Bolme_8d3f* set, however, displays only a slight increase in EER, from 4.11% to 4.59%. The filter subsets in general lead to similar degradation in accuracy when they are applied to either landmark localization or feature extraction.

	Stage(s) using new filter set		
Set	**All**	**Feature Extraction + Template Matching**	**Landmark Localization only**
Bolme	4.11%		
Bolme_8d3f	4.59%	4.17%	4.52%
Bolme_4d5f	20%	10%	10.5%
Bolme_4d3f	25%	11%	10.5%

Table 31 EERs of different configurations

Conclusion

In this study, it is shown that by using a subset of Gabor filters, slight improvement in execution time (around two seconds) can be achieved at the expense of a slight drop in verification accuracy. However, this technique should be used with restraint as verification accuracy comes first for most authentication applications. As a result, this improvement is not incorporated into the final implementation. In the following sections, we will continue to investigate other accuracy preserving techniques.

5.4.5 Fine Tuning System Cache Policy

On a system wide level, cache and memory access efficiency are governed by the cache policy. Modern operating systems allow the configuration of cache policy by changing kernel settings. For Linux, one can enable/disable the Write Through (WT) and Write Allocate (WA) options. If write-through is enabled, the new data is written to both cache and main memory on every write; if it is disabled, the new data is only written to the cache only. Later, if another memory location needs to use the cache line where this data is stored, it is saved (write-back) to the system memory. On the other hand, write-allocate determines if a whole cache line worth of data is brought in (allocate) on a write miss. Depending on the nature of applications running on a system, the cache policy can be fine tuned to suit specific needs. In this section, the effects of various cache policy settings are investigated.

Results

The execution time of EBGM was measured using kernels configured with different cache policies. Table 32 shows the results. Set 0 is the default setting for *Mobile*. Results show that set 2 (write-through and write-allocate) is the best combination, resulting in total a one second reduction in authentication time. This is probably due to the fact that memory accesses in the two EBGM stages are largely sequential. By using a write-back policy, subsequent writes to nearby memory locations can be buffered and reduce slow memory writes. On

the other hand, write-allocate ensures that the first write to a memory location brings in nearby content, so that subsequent writes require no extra memory read.

Cache Policy			Timing	
Set	**WT?**	**WA?**	**Landmark Localization**	**Feature Extraction**
0	Y	Y	7.02s	2.92s
1	Y	Y	7.37s	3.00s
2	N	Y	**6.06s**	**2.78s**
3	N	N	6.22s	2.75s
4	Y	N	6.22s	2.74s

Table 32 Execution time for different cache policies

5.4.6 Reducing Redundant Memory Access by Loop Merging

In the previous sections, the efficiency of individual memory access (Section 5.4.3) and system-wide cache policy (Section 5.4.5) have been investigated. While these techniques provided considerable speed up in execution time, they are general techniques which do not make use of application specific knowledge. In this section, a detailed analysis of the EBGM algorithm will be given, which will give insight into possible optimization opportunities.

Complexity Analysis

As pointed out in Section 5.4.3, filter convolution (convolvePoint) consists mainly of addition, multiplication and image element access. For brevity, 'complexity' of an algorithm will only refer to the number of two basic

operations, namely Data Access (DACC), and Multiply and Add (MADD). Several important quantities and their values are also defined in Table 33.

Quantity	Value(s)
Mask size ($m \times m$)	m = {19, 29, 39,53,77}
Face image size ($N \times N$)	N = 128
NumOfMasks	80 (40 pairs)
NumOfNodes	25 (Landmark localization), 80 (Feature extraction)

Table 33 Quantities used in analysis

The core of the EBGM algorithms consists of two stages, namely landmark localization and feature extraction. The most time consuming operations in both stages is Gabor jet extraction, involving finding the convolution responses to a set of Gabor masks (filters) at a specific location.

The goal of landmark localization is to automatically locate the feature points of interest (nodes) in a novel face. Initial guesses are made and Gabor jets are extracted from them. The similarity between these Gabor jets and those in a bunch graph, which contains Gabor jets extracted from different training faces, are then computed. Displacement between the initial guess locations and the real ones are estimated by moving the guess location around until a point of highest similarity is found. After all nodes are found, their locations are stored as a face graph. A total of 25 nodes are extracted (Algorithm 1).

Build Face Graph

1: Load *BunchGraph*
2: Load *GaborMasks*
3: Load *FaceImage*
4: Make initial guess of *NodeLocations*
5: **for** *i* ← *1* **to** *NumOfNodes* **do**
6: *GaborJet(i)* = ***ExtractGaborJet****(NodeLocation(i), FaceImage, GaborMasks)*
7: **for** *j* ← *1* **to** *NumOfBunchJets* **do**
8: Find similarity between *BunchJet(j)* and *GaborJet(i)*
9: **end for**
10: Estimate displacement of *NodeLocation(i)* from best matching *BunchJet*
11: Update *NodeLocation(i)* with displacement
12: **end for**
13: Save *NodeLocations* to *FaceGraph*

Algorithm 1 *Build Face Graph*

For local feature extraction, the face graph built in the previous stage is loaded. Gabor jets are then extracted from the original 25 nodes and an additional 55 interpolated ones. These Gabor jets are incorporated with the face graph and forms the final face template (Algorithm 2).

Extract local features
1: Load *FaceImage*
2: Load *FaceGraph*
3: Load *GaborMasks*
4: Interpolate new *NodeLocations* from *NodeLocations* stored in face graph
5: **for** *i* ← *1* **to** *NumOfNodes* **do**
6: *GaborJet(i)* = ***ExtractGaborJet****(NodeLocation(i), FaceImage, GaborMasks)*
7: **end for**
8: Save interpolated *NodeLocations* and *GaborJets* to *FaceTemplate*

Algorithm 2 *Extract local features*

At the heart of both landmark localization and feature extraction is Gabor jet extraction. Given a face image, a landmark location and a set of Gabor filters, Gabor jet extraction computes the filter responses of all the filters and the landmark. Standard convolution is used to compute the response and is implemented in the function *convolvePoint* (the shaded lines in Algorithm 3).

ExtractGaborJet*(NodeLocation, Image, Masks)*
1: **for** *i* ← *1* **to** *NumOfMasks* **do**
2: **for** *j* ← *1* **to** *MaskHeight(i)* **do**
3: **for** *k* ← *1* **to** *MaskWidth(i)* **do**
4: Load *FaceImage(j, k)*
5: Load *GaborMask(i, j, k)*
6: *Sum(i) = Sum(i) + FaceImage (j, k)* x *GaborMask(i, j, k)*
7: **end for**
8: **end for**
9: **end for**

Algorithm 3 *ExtractGaborJet*

	General	Original
No. of DACC	$2 \times NumOfMasks \times m^2$	278240
No. of MADD	$NumOfMasks \times m^2$	139120

Table 34 Complexity for *ExtractGaborJet*

	General	Original
No. of DACC	$NumOfNodes \times DACC(ExtractGaborJet)$	6956000
No. of MADD	$NumOfNodes \times MADD(ExtractGaborJet)$	3478000

Table 35 Complexity for *Build Face Graph*

	General	Original
No. of DACC	$NumOfNodes \times DACC(ExtractGaborJet)$	22259200
No. of MADD	$NumOfNodes \times MADD(ExtractGaborJet)$	11129600

Table 36 Complexity for *Extract Local Features*

Table 34, Table 35 and Table 36 show the complexity of the three algorithms and the number of DACC and MADD for the standard configuration. It is obvious that the complexity of *Build Face Graph* and *Extract Local Features* are directly proportional to the complexity of *ExtractGaborJet*. We will focus on this algorithm for the rest of the analysis.

For each call to *ExtractGaborJet*, the function convolvePoint (Algorithm 3, lines 2-8) is called 80 times to convolve with the Gabor filter sets (40 pairs). Since the node location is identical in all 80 calls, the area of interest in the face image, and hence the image pixels loaded should be the same for filters of the same size. This high redundancy causes unnecessary image element access (Figure 38), and can be avoided by loop merging. By convolving several masks

in parallel, the same image pixel need only be accessed once only (Figure 39). This parallelism can be achieved at two levels. The first and the most straightforward one is the combination of real and imaginary part of each filter. The second level is combining multiple filters of the same size.

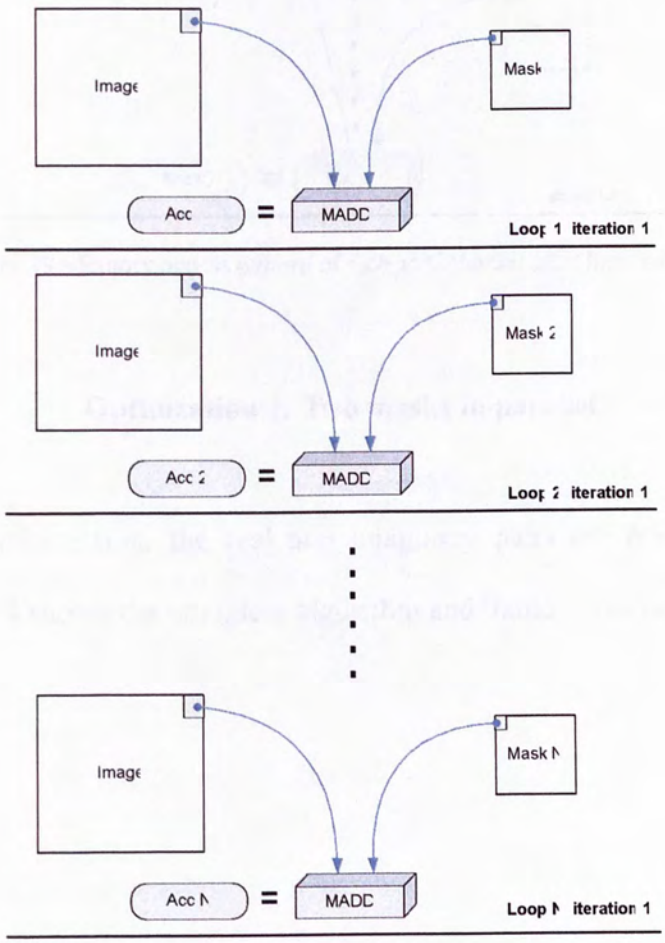

Figure 38 Memory access pattern of the original *ExtractGaborJet*

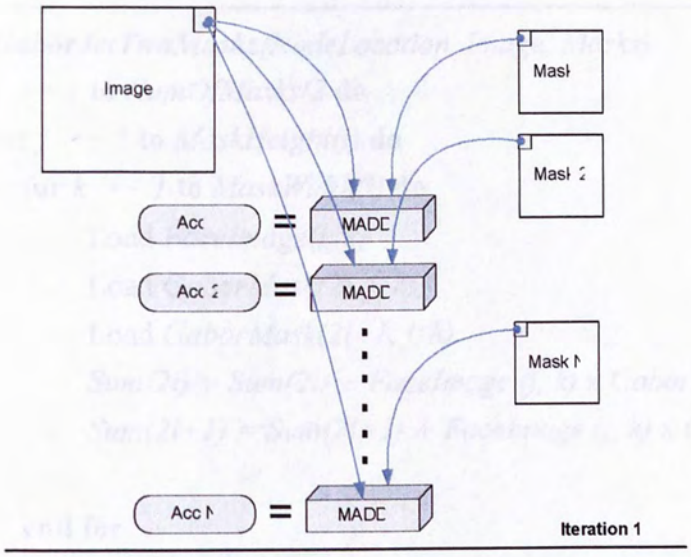

Figure 39 Memory access pattern of *ExtractGaborJet* after loop merging

Optimzation 1: Two masks in parallel

For this optimization, the real and imaginary pairs are convolved together. Algorithm 4 shows the complete algorithm and Table 37 shows the complexity analysis.

```
ExtractGaborJetTwoMasks(NodeLocation, Image, Masks)
1:  for i ← 1 to NumOfMasks/2 do
2:      for j ← 1 to MaskHeight(i) do
4:          for k ← 1 to MaskWidth(i) do
5:              Load FaceImage(j, k)
6:              Load GaborMask(2i, j, k)
7:              Load GaborMask(2i+1, j, k)
8:              Sum(2i) = Sum(2i) + FaceImage (j, k) x GaborMask(2i, j, k)
9:              Sum(2i+1) = Sum(2i+1) + FaceImage (j, k) x GaborMask(2i+1, j,
k)
10:         end for
11:     end for
12: end for
```

Algorithm 4 *ExtractGaborJetTwoMasks*

	General	Standard
No. of DACC	$\frac{3}{2} \times NumOfMasks \times m^2$	208680
No. of MADD	$NumOfMasks \times m^2$	139120

Table 37 Complexity for ExtractGaborJetTwoMasks

Optimization 2: Multiple Masks in parallel

For this optimization, all filters in a mask set are convolved at once. Here we define the quantity MaskSetSize, which means the number of masks convolved simultaneously. Note that *ExtractGaborJet* (**Algorithm 3**) and *ExtractGaborJetTwoMasks* (Algorithm 4) are in fact special case of *ExtractGaborJetMulipleMasks* (Algorithm 5), for which *MaskSetSize* = 1 and 2 respectively. Table 38 shows the complexity analysis.

```
ExtractGaborJetMultipleMasks(NodeLocation, Image, Masks)
1:  for i ← 1 to NumOfMasks/MaskSetSize do
2:      for j ← 1 to MaskHeight(i) do
4:          for k ← 1 to MaskWidth(i) do
5:              Load FaceImage(j, k)
6:              for l ← 1 to MaskSetSize do
7:                  MaskIndex = i x MaskSetSize+l
8:                  Load GaborMask(MaskIndex, j, k)
9:                  Sum(MaskIndex) += FaceImage (j, k) x
                GaborMask(MaskIndex, j, k)
10              end for
11:         end for
12:     end for
13: end for
```

Algorithm 5 *ExtractGaborJetMultipleMasks*

	General	Standard
No. of DACC	$\left(1+\frac{1}{MaskSetSize}\right)\times NumOfMasks\times m^2$	147815
No. of MADD	$NumOfMasks\times m^2$	139120

Table 38 Complexity for ExtractGaborJetMultipleMasks

Theoretically, the more masks convoluted simultaneously, the more the saving. As shown in the above analysis, the number of memory access reduces from the original $2\times NumOfMasks\times m^2$ to $\left(1+\frac{1}{MaskSetSize}\right)\times NumOfMasks\times m^2$. It is obvious that the more masks in a mask set, the more the saving. The table below shows the projected memory access for the three algorithms. Note that when *MaskSetSize* equals 16, the number of access is nearly halved (Table 39).

MaskSetSize	1	2	16
MACC (Ratio)	229,215,200 (2)	21,911,400 (1.5)	15,520,575 (1.06)

Table 39 Number of MACC for different *MaskSetSize*

Results

To validate the above finding, the proposed algorithms were implemented and timing information was collected. Table 40 shows the execution time and page faults of each algorithm. It is found that execution times of Algorithm 4 (*MaskSetSize* = 2) and Algorithm 5 (*MaskSetSize* =16) are slightly better than the original one (*MaskSetSizse* = 1). When *MaskSetSize* increases from 2 to 16, a mild decrease in performance is observed, suggesting that the more complicated implementation outweighs its benefit.

The number of page faults is largely the same for all settings, which means the memory access efficiency is not improved by the change in access pattern. Also, the large reduction in number of memory accesses (halved) does not directly lead to corresponding decrease in execution time, suggesting that the bottleneck may resides at the architecture level. Cache misses which result in slow memory access, may be the root of the problem and will be investigated in the next section.

Stage	Statistics	*MaskSetSize* 1	2	16
2)Landmark Localization	Time	6.06s	**5.87s**	6.00s
	Page faults (major)	759	760	760
	Page faults (minor)	1476	1476	1476
3) Feature Extraction	Time	2.78s	**2.12s**	2.56s
	Page faults (major)	755	755	756
	Page faults (minor)	1326	1326	1326

Table 40 Execution time of using different *MaskSetSize*

5.4.7 Maximizing Cache Reuse by Array Merging

As discussed in the last section, although the number of memory access is cut by half using loop merging, execution time shows no major improvement. This suggests that data access is still far from efficient, and a more in-depth analysis of the memory access pattern of EBGM is needed.

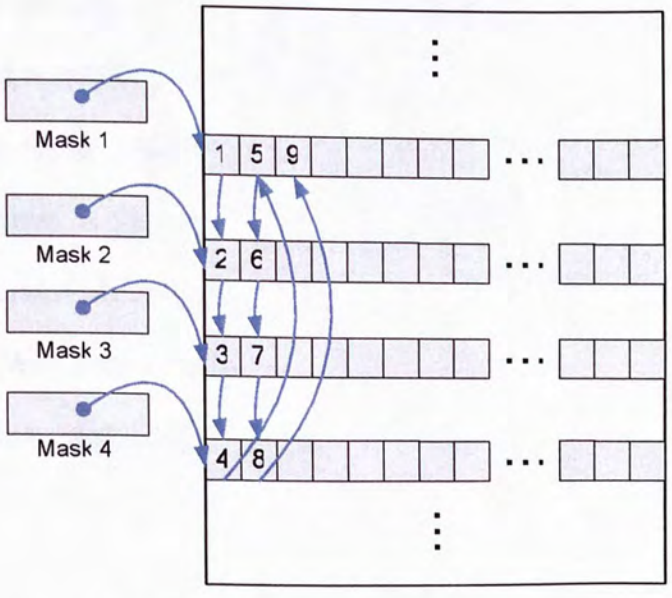

Figure 40 Mask array layout (original)

Figure 40 shows the memory access pattern for *ExtractGaborJetMultipleMask* (Algorithm 5) using a *MaskSetSize* of 4. Since the Gabor masks are allocated dynamically, there is no guarantee on their relative locality and they may occupy separate memory locations. Hence, accessing pixels from different masks will result in extremely poor locality, causing frequent 'jumps' in access locations. This explains why simply convoluting several masks together cannot bring about the predicted improvement – the long stalls between each mask pixel accesses outweigh any benefits brought about by the parallelism. In addition, the rapidly changing access address may increase the probability of a cached data being evicted before being reused.

Optimization

To alleviate this problem, a new memory layout is derived for Gabor masks using array merging to increase the access locality and cache reuse. Array merging and reordering were first proposed in [47] for compiling data intensive applications for an embedded device. During compilation, data dependencies among different arrays used in the same program are first analyzed, and then efficient storage schemes are derived to improve memory access performance at runtime. With prior knowledge about the EBGM memory access pattern, it is also possible to apply the same idea to our optimization problem.

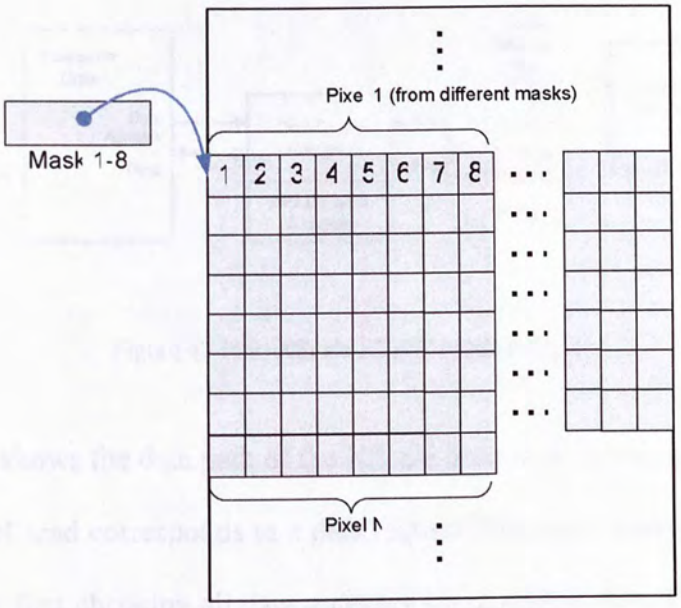

Figure 41 Mask array layout (after merging)

Figure 41 shows the proposed layout. Suppose a *MaskSetSize* of 8 is used; the eight masks in the set will be merged and stored in one single array. The pixels from each mask are stored in an alternate manner – pixel one from mask one, pixel two from mask two ... and so on. Using this scheme, mask pixels accessed within each iteration of *ExtractGaborJetMultipleMasks* now have a sequential order. In addition to the improvement in access locality, this scheme also allows maximum cache data reuse as will be explained below.

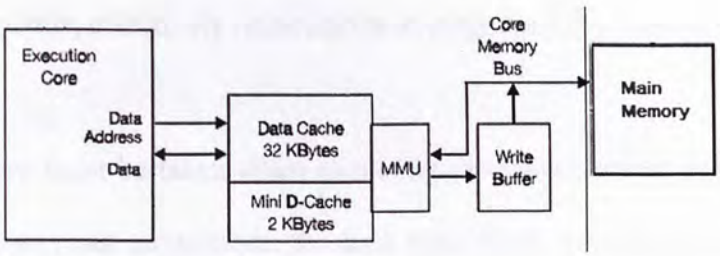

Figure 42 Intel XScale PXA255 data path[6]

Figure 42 shows the data path of the XScale processor. At the architecture level, every pixel read corresponds to a data request. The execution core handles this request by first checking all data registers for requested data. If it is not found, the fast on-chip data cache is checked. If the data is found in one of the cache entries, it is returned to the core; otherwise, the memory management unit is invoked and a request is sent to the main memory (cache miss). The memory page containing the requested data is then brought in and the data is forwarded to cache and then back to the core. In fact, a cache line size worth of data is sent to the cache, so neighbours of the requested data are brought in even if only one bit is requested.

Our proposed scheme makes use of this fact and maximizes data reuse by packing pixels accessed in the same iteration together (Figure 41). The *MaskSetSize* of 8 is chosen since the cache line size in the XScale processor is 32 bytes, or 8 fixed-point pixels. Ideally, when a request is sent for the first pixel, the following seven pixels are brought in by the same memory read and occupy one cache line. The latency of one memory access is shared among the

eight mask pixels, effectively reducing the average memory stall time.

Note that care must be taken when allocating the masks arrays. Since the data cache is 32-way set associative, the first pixel must be aligned at a 32-byte boundary so that it and the following seven pixels are allocated to the same cache line. Otherwise, useless pixels in front of the first pixel are brought in and subsequent pixels may cause an extra memory read. This memory alignment requirement can be enforced by allocating all arrays with allocation function calls that guarantee alignment. Array merging can be done offline at the same time when Gabor masks are pre-computed (Section 5.4.2), so no overhead is incurred.

Results

The array merging scheme was implemented and evaluated. Table 41 shows the execution time for one authentication session on the *Mobile* platform. For comparison, we also conducted the experiment on the *Desktop* platform, and the execution time for one hundred sessions is shown in Table 42.

For *Mobile*, the improvement for the feature extraction is dramatic – a 2.5 times reduction in time for a *MaskSetSize* of 8. However, there is no significant improvement for the landmark localization stage. Note that there is almost the same reduction in the number of minor page faults in both stages; hence further analysis is needed to find out why the discrepancy can be so large. The result

on *Desktop* is much more reasonable, in the sense that significant improvement is shown in both stages. A similar reduction in number of minor page faults is also observed. This shows that an efficient data mapping can be beneficial to both mobile and desktop performance, despite their inherent differences in memory architecture and configuration.

		MaskSetSize		
Stage	**Statistics**	**1**	**2**	**8**
Build Face Graph	Time	6.06s	5.87s	5.85s
	Page faults (major)	759	760	761
	Page faults (minor)	1476	1476	**1219**
Extract local features	Time	2.78s	2.12s	**1.08s**
	Page faults (major)	755	755	756
	Page faults (minor)	1326	1326	**1072**

Table 41 Execution time using different MaskSetSize (Mobile, 1 session)

		MaskSetSize		
Stage	**Statistics**	**1**	**2**	**8**
Build Face Graph	Time	6.87s	7.39s	**4.62s**
	Page faults (major)	0	0	0
	Page faults (minor)	1560	1543	**1323**
Extract local features	Time	11.85s	12.71s	**3.96s**
	Page faults (major)	0	0	0
	Page faults (minor)	1314	1309	**1098**

Table 42 Execution time using different MaskSetSize (Desktop, 100 sessions)

Post-profiling

As pointed out above, the landmark localization stage does not show improvement as feature extraction does. Post-profiling results reveal the cause. As shown in Table 43 and Table 44, filter convolution remains the most time consuming function in the feature extraction stage, while its significance drops to around 14% for the landmark localization stage. The distance estimation and trigonometric functions become the new bottlenecks. Array merging improved the memory access efficiency, resulting in a change of program behaviour – from memory-bound back to computation bound once again.

Function Name	Description	Before	After
1. convolvePoint	Filter convolution	71.03%	14.76%
2. DEPredictiveIter	Distance estimation	9.29%	27.63%
3. sin, cos, atan	Trigonometric functions	18.79%	25.87%

Table 43 Function Profile (Landmark localization)

Function Name	Description	Before	After
1. convolvePoint	Filter convolution	97.59%	75.75%

Table 44 Function Profile (Feature extraction)

Line level profiling further confirms this observation. As shown in Table 45 and Table 46, the effect of memory optimization becomes even more prominent and agrees with our analysis at the function level. For landmark localization, the most time consuming lines do not include any memory access operations. For feature extraction, however, the percentage of time devoted to memory access is still over 70%.

Overall, the profiling results show that memory access remains the key to optimizing the feature extraction stage. For landmark localization, however, focus should be put on distance estimation and trigonometric functions.

Statement	Percentage
1. Trignometric functions	30.06%
2. Fixed-point multiplication	7.77%

Table 45 Line Profile (Landmark localization)

Statement	Percentage
1. Filter element access	64.97%
2. Image element access	5.61%

Table 46 Line Profile (Feature extraction)

5.4.8 Optimization of Trigonometric Functions using Table Lookup

Optimization

Trigonometric functions were implemented using polynomial series. A large amount of floating point multiplications is involved in the series expansion and can result in poor execution time on mobile devices. To solve this problem, the table look up technique is used. Values of trigonometric functions are pre-computed, stored and loaded into arrays on program startup. Making use of the periodic nature of and relationship between trigonometric functions, only two tables are required to implement the sine, cosine, tangent and arctangent functions. All calls to trigonometric functions are then modified to array accesses.

Results

As shown in Table 47, we see a dramatic improvement in execution time for both landmark localization and template matching. This can be explained by the fact that both stages involve intensive use of distance estimation, in which most trigonometric functions are invoked. The feature extraction stage also shows a moderate improvement. Post-profiling results further confirm the effectiveness of our approach. As shown in Table 49, the significance of trigonometric functions virtually falls to zero. The significant performance gain strongly justifies the 1.5Kb extra storage introduced by table lookup (Table 48). Note that the time for one face authentication session now requires only 1.3s, meeting our real-time requirement. In the next section, the accuracy of the optimized system will be verified.

Stage		**Before**	**After**
2) Landmark Localization	Time	5.85s	**0.52s**
	Page faults (major)	761	**722**
	Page faults (minor)	1219	1222
3) Feature Extraction	Time	1.08s	**0.31s**
	Page faults (major)	756	**715**
	Page faults (minor)	1072	1068
4) Template Matching	Time	0.2263s	**0.0062s**
	Page faults (major)	725	**709**
	Page faults (minor)	2259	2259

Table 47 Execution time of different EBGM stages

Name	Size	Description
_cos_tbl[]	1Kb (256 * 4 bytes)	Common table for Sine and Cosine (Half cycle)
_tan_tbl[]	0.5Kb (128 * 4 bytes)	Table for Tangent and Arc Tangent. (1/4 cycle)

Table 48 Storage requirement for lookup tables

Function Name	Description	Before	After
1. convolvePoint	Filter convolution	14.76%	25.50%
2. DEPredictiveIter	Distance estimation	27.63%	45.01%
3. sin, cos, atan	Trigonometric functions	25.87%	**~0.0%**

Table 49 Execution time breakdown (Landmark localization)

5.5 Summary

In this section, the verification accuracy of the optimized EBGM implementation will be evaluated and a summary of the optimization techniques and their effects will be given.

Similar to Section 3.4, the FERET face database and evaluation protocol was used to investigate the verification accuracy. Figure 43 to Figure 46 show the Receiver Operating Characteristic (ROC) curves of the optimized code and the baseline system for different probe sets. The overlapping curves indicate that verification accuracies of both systems are essentially identical.

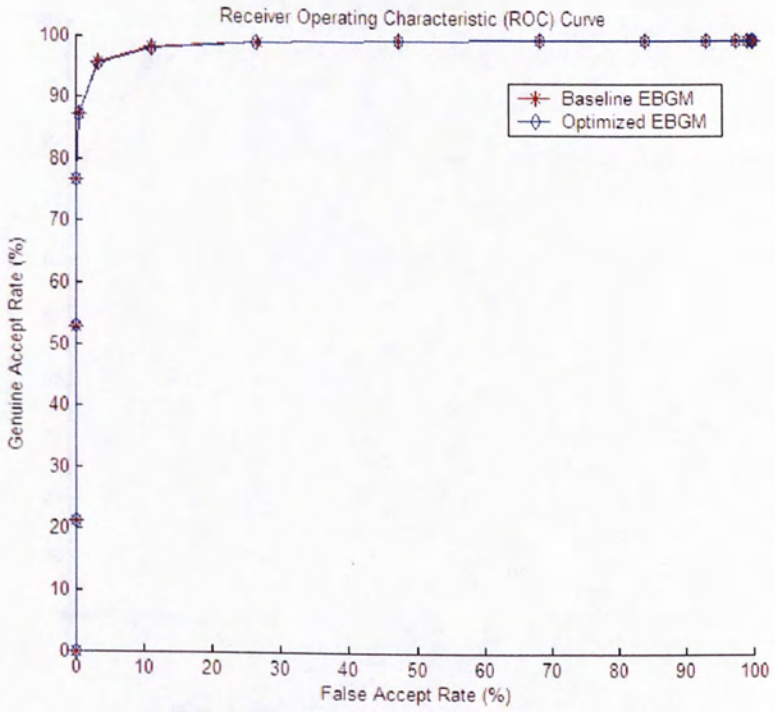

Figure 43 ROC Curves of EBGM (**FB** probe set)

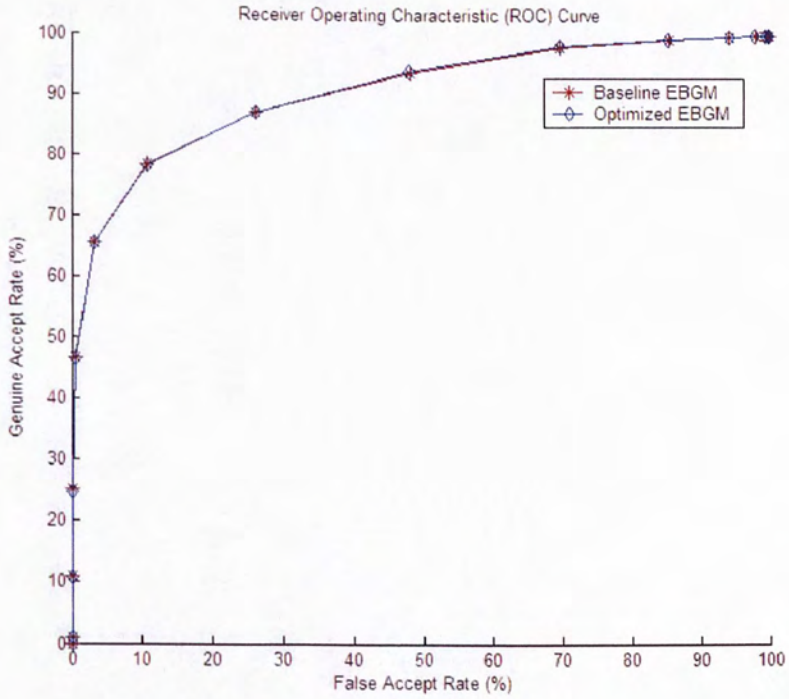

Figure 44 ROC Curves of EBGM (**dup1** probe set)

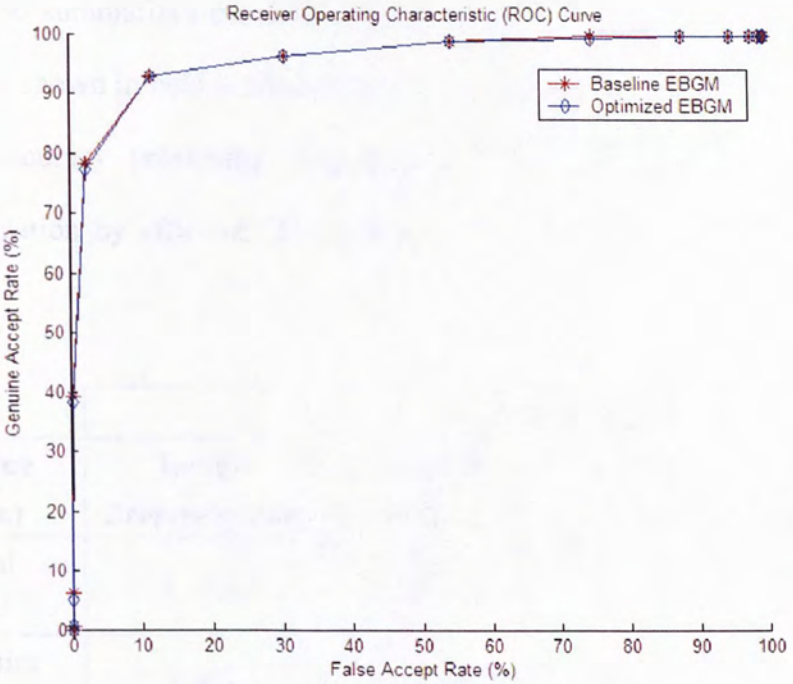

Figure 45 ROC Curves of EBGM (**fc** probe set)

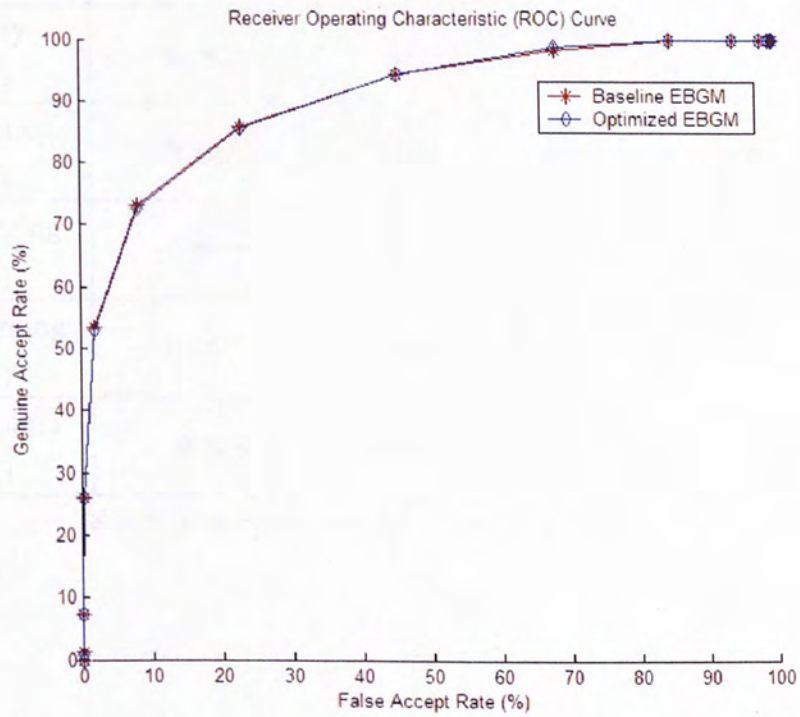

Figure 46 ROC Curves of EBGM (**dup2** probe set)

Table 50 summarizes the optimization techniques employed and their effects. Figures shown in bold indicated that the particular stage is improved. Note that only accuracy preserving techniques are included here, so the effect of optimization by efficient Gabor filter set selection (Section 5.4.4) is omitted here.

	Execution Time				
Technique (Section)	**Image Preprocessing**	**Landmark Localization**	**Feature Extraction**	**Template Matching**	**Total**
Original (N/A)	8.335s	218.2s	324.8s	1.849s	553.2s
Fixed-point (5.4.1)	**1.93s**	**15.07s**	**29.45s**	**0.2812s**	46.73s
Pre-computation (5.4.2)	1.93s	**12.91s**	**20.97s**	0.2812s	36.09s
1D Array (5.4.3)	**0.882s**	**7.026s**	**2.989s**	0.2970s	11.19s
Cache Policy (5.4.5)	0.882s	**6.06s**	**2.78s**	**0.2263s**	9.948s
Loop Merging (5.4.6)	0.882s	**5.87s**	**2.12s**	0.2263s	9.098s
Array Merging (5.4.7)	0.882s	**5.85s**	**1.08**	0.2263s	8.043s
Table Lookup (5.4.8)	**0.48s**	**0.52s**	**0.31s**	**0.0062s**	1.320s

Table 50 Summary of optimization techniques and their effects

6. Conclusions

Despite the growing importance of mobile devices and increasing demand for more secure authentication methods, the adoption of reliable and affordable schemes such as face authentication is slow to keep up with the demand. Mobile devices, due to their scarce computation and storage resources, have never been the real target of traditional face recognition systems.

In this thesis, we investigate the feasibility of real time face authentication on mobile devices. In particular, two representative and fundamentally different face recognition algorithms, Principal Component Analysis (PCA) and Elastic Bunch Graph Matching (EBGM), are implemented and optimized. Various computation and memory optimization techniques such as fixed-point arithmetic and array merging are also developed and employed. Experimental results show a significant improvement in execution time for both PCA and EBGM. For PCA, the time for one authentication session reduces from 30 seconds to 1.3 seconds. For EBGM, a 420 times reduction is achieved – from 553 seconds down to 1.3 seconds. Further verification experiments show that the real time performance is achieved without any significant loss in accuracy. Sub-second performance may be possible by making use of architecture specific techniques, such as cache mapping[53], programmable page attributes[45], preload instructions[54], and DSP extension[45].

At the beginning of the PCA and EBGM optimization, we encounter the problem of not having a formal optimization model to follow. There may be tips, hints and advice for some facets of the optimization problem, but not one simple and general enough to be adapted to our particular problem. The arsenal of tools and techniques developed by the software community over the years has only added to the confusion. Optimization, then, becomes more a personal artistry than a science that anyone can follow, adapt and extend. It soon became clear to us that a software engineering model dedicated to the general optimization problem is needed. In light of this, we proposed a high level view of optimization techniques and a feedback oriented workflow. The high level view gives insights on how general optimization problems should be approached and how techniques can be categorized; the workflow keeps us sensitive to changes in program behaviour and measure the effectiveness of a technique to a particular problem. The merit of the two lies in their simplicity and flexibility, and they may serve as a basis for a full-fledged model.

Our real time, accuracy preserving implementations of PCA and EBGM show that by combining suitable optimization techniques and adequate knowledge of the target platform and application at hand, the computational power of constrained devices can be fully exploited. The same optimization approach, techniques and workflow can be applied generally to other problems which demands fast execution time on slow devices. On the other hand, our real time PCA and EBGM implementations may serve as parts of a larger evaluation

framework for mobile authentication methods. Possible addition to this framework includes a face database which consists of images collected from mobile devices, so that the mobile face authentication problem can be more accurately modeled.

Real time face authentication on mobile devices makes a myriad of applications possible, for instance, the combination of difference face recognition 'experts' (e.g. PCA, EBGM) to improve accuracy. Previous efforts in other biometric modalities such as voice [32] and fingerprint [33, 34, 35, 36] can also be combined to construct a multimodal authenticator similar to the scheme in [55]. However, such extensions would involve derivation of sophisticated combination rules which are non-trivial and are beyond the scope of this thesis.

In conclusion, we believe that our experience will be useful to software optimization on such devices so mobile phones, and PDAs. Such devices share one common set of characteristics: relatively slow CPU, fixed hardware platform, support multiple programs, and multimedia applications. For such machines, traditional hardware-software co-design is not applicable because of the diversified programs to be supported. Under such conditions, our techniques will be dominant for their performance optimization.

7. Bibliography

[1] U. S. Department of Defense, "Biometrics 101 Tutorial", http://www.biometrics.dod.mil/bio101/index.aspx.

[2] H. Thomas, "Biometrics: Face Recognition", http://www-users.cs.york.ac.uk/~tomh/Biometrics.html.

[3] J. Zhang, Y. Yan, and M. Lades, *"Face Recognition: Eigenface, Elastic Matching, and Neural Nets,"* the IEEE, vol. 85, pp. 1423-1435, 1997.

[4] S. A. Rizvi, P. J. Phillips, and H. Moon, *"The FERET Verification Testing Protocol for Face Recognition Algorithms,"* Proceedings of the International Conference on Automatic Face and Gesture Recognition, Nara, Japan, pp. 48-53, 1998.

[5] P. Phillips, H. Wechsler, J. Huang, and P. Rauss., *"The FERET Database and Eevaluation Procedure for Face Recognition Algorithms,"* Image and Vision Computing, vol. 16, pp. 295-306, 1998.

[6] "Intel XScale Microarchitecture - Technical Summary," Intel Corporation, 2000.

[7] "Camera phones are 'must-haves'," in *BBC News*, 2004.

[8] M. Turk and A. Pentland, *"Face Recognition Using Eigenfaces,"* Proceedings of Computer Vision and Pattern Recognition, pp. 586-591, 1991.

[9] K. H. Pun, Y. S. Moon, C.C. Tsang, C. T. Chow, and S. M. Chan, *"A Face Recognition Embedded System,"* Proceedings of Biometric Technology for Human Identification II, SPIE Defense and Security Symposium, Florida, USA, pp. 390-397, 2005.

[10] L. Wiskott, J. Fellous, N. Kruger, and C. v. d. Malsburg, *"Face Recognition by Elastic Bunch Graph Matching,"* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, pp. 775-779, 1997.

[11] C. Liu and H. Wechsler, *"Evolutionary Pursuit and Its Application to Face Recognition,"* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, pp. 570-582, 2000.

[12] Marian Stewart Bartlett, H. Martin Lades, and T. J. Sejnowski, *"Independent Component Representations for Face Recognition,"* Proceedings of the SPIE, Vol 3299: Conference on Human Vision and Electronic Imaging III, pp. 528-539, 1998.

[13] M. Turk and A. Pentland, *"Face Recognition Using Eigenfaces,"* Journal of Cognitive Neuroscience, vol. 3, pp. 71-86, 1991.

[14] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, *"Eigenfaces vs. fisherfaces: Recognition Using Class Specific Linear Projection,"* IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 19, pp. 711-720, 1997.

[15] P. S. Penev and J. J. Atick, *"Local Feature Analysis: A General Statistical Theory for Object Representation,"* Network: Computation in Neural Systems, vol. 7, pp. 477-500, 1996.

[16] "Local Feature Analysis. Automated Identification and Data Capture Web Site", http://et.wcu.edu/aidc/BioWebPages/lfa.htm.

[17] P. J. Phillips, P. Grother, R. J. Micheals, D. M. Blackburn, E. Tabassi, and M. Bone, "Face Recognition Vendor Test (FVRT) 2002: Overview and summary", www.frvt.org.

[18] "Cognitec Systems GmbH. " http://www.cognitec-systems.de/index.html.

[19] "Neven Vision, Inc. " http://www.nevenvision.com.

[20] "Identix, FaceIt", http://www.identix.com/trends/face.html.

[21] E. Weinstein, P. Ho, B. Heisele, T. Poggio, K. Steele, and A. Agarwal, *"Handheld Face Identification Technology in a Pervasive Computing Environment,"* Proceedings of Pervasive 2002, Zurich, Switzerland, pp. 48-54, 2002.

[22] T. J. Hazen, E. Weinstein, and A. Park, *"Towards Robust Person Recognition on Handheld Devices using Face and Speaker Identification Technologies,"* Proceedings of the 5th International Conference on Multimodal interfaces, pp. 289-292, 2003.

[23] T. J. Hazen, E. Weinstein, R. Kabir, A. Park, and B. Heisele, *"Multi-modal Face and Speaker Identification on a Handheld Device,"* Proceedings of the Workshop on Multimodal User Authentication, pp. 113-120, 2003.

[24] S. L. Wijaya, M. Savvides, and B. V. K. V. Kumar, *"Illumination-tolerant Face Verification of Low-bit-rate JPEG2000 Wavelet Images with Advanced Correlation Filters for Handheld Devices,"* Applied Optics, Special Issue on Biometric Recognition, vol. 44, pp. 655-665, 2005.

[25] H. Fatemi, R. Kleihorst, H. Corporaal, and P. Jonker, *"Real-time Face Recognition on a Smart Camera,"* Proceedings of Advanced Concepts for Intelligent Vision Systems 2003, Ghent, Belgium, pp. 222-227, 2003.

[26] J. Yang, X. Chen, and W. Kunz, *"A PDA-based Face Recognition System,"* Proceedings of the 6th IEEE Workshop on Applications of Computer Vision, pp. 19-23, 2002.

[27] J. Yang, X. Chen, W. Kunz, and H. Kundra, *"Face as in Index: Knowing Who is Who Using a PDA,"* International Journal of Imaging Systems and Technology, pp. 33-41, 2003.

[28] J.-L. Nagel, P. Stadelmann, M. Ansorge, and F. Pellandini, *"Comparison of Feature Extraction Techniques for Face Verification using Elastic Graph Mathcing on Low-power Mobile Devices,"* Proceedings of IEEE Region 8 EUROCON 2003, International Conference on Computer as a Tool, Ljubljana, Slovenia, pp. 365-369, 2003.

[29] "Omron, OKAO Vision Face Recognition Sensor", http://www.omron.com/news/n_280205.html.

[30] "OMRON Demonstrates 'OKAO Vision Face Recognition Sensor' for Mobile Phones at Security Show Japan 2005", http://www.japancorp.net/Article.asp?Art_ID=9820.

[31] "Earth Beat, Genelock Light for Cell-phone", http://www.earthbeat.co.jp/Web_EB3/index.html.

[32] Y. S. Moon, C. C. Leung, and K. H. Pun, *"Fixed-point GMM-based Speaker Verification over Mobile Embedded System,"* Proceedings of the 2003 ACM SIGMM workshop on Biometrics methods and applications, Berkley, California, pp. 53 - 57, 2003.

[33] J. S. Chen, Y. S. Moon, and K. F. Fong, *"Efficient Fingerprint Image Enhancement for Mobile Embedded Systems,"* Proceedings of ECCV 2004 International Workshop(BioAW), Prague, Czech Republic, pp. 146-157, 2004.

[34] P. S. Cheng, Y. S. Moon, Z. G. Cao, K. C. Chan, and T. Y. Tang, *"Enabling Fingerprint Authentication in Embedded Systems for Wireless Applications,"* Proceedings of the 1st International Workshop on Signal Processing for Wireless Communications, pp. 228 - 231, 2003.

[35] T. Y. Tang, Y. S. Moon, and K. C. Chan, *"Efficient Implementation of Fingerprint Verification for Mobile Embedded Systems using Fixed-point Arithmetic,"* Proceedings of the 2004 ACM symposium on Applied computing, Nicosia, Cyprus, pp. 821-825, 2004.

[36] Y. S. Moon, F. T. LUK, H. C. HO, T. Y. TANG, K. C. CHAN, and C. W. LEUNG, *"Fixed-Point Arithmetic for Mobile Devices - A Fingerprint Verification Case Study,"* Proceedings of the SPIE 2002 Advanced Signal Processing Algorithms, Architectures, and Implementations XII, Seattle, pp. 144 -149, 2002.

[37] "The CSU Face Identification Evaluation System ", http://www.cs.colostate.edu/evalfacerec.

[38] R. Beveridge, D. S. Bolme, M. Teixeira, and B. Draper, "The CSU Face Identification Evaluation System User's Guide"

[39] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 2nd ed. Peking: Thomson Brooks/Cole, 2002.

[40] B. Jahne, *Digital Image Processing*, 5th revised ed. Berlin: Springer, 2002.

[41] "GNU profiler (Gprof)", http://www.gnu.org/software/binutils/manual/gprof-2.9.1/gprof.html.

[42] "Application Note 33: Fixed Point Arithmetic on the ARM", http://www.arm.com/pdfs/DAI0033A_fixedpoint.pdf.

[43] "arm-linux-gcc Cross Compiler", ftp://ftp.handhelds.org/pub/linux/arm/toolchain.

[44] D. S. Bolme, Thesis: "Elastic Bunch Graph Matching," Colorado State University, 2003.

[45] "Intel XScale Microarchitecture for the PXA255 Processor - User's Manual," Intel Corporation, 2003.

[46] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*: Morgan Kaufmann Publishers Inc., 1996.

[47] V. D. L. Luz and M. Kandemir, *"Array Regrouping and Its Use in Compiling Data-Intensive Embedded Applications,"* IEEE Transactions on Computers, vol. 53, pp. 1-19, 2004.

[48] P. R. Panda, N. D. Dutt, A. Nicolau, F. Catthoor, A. Vandecappelle, E. Brockmeyer, C/ Kulkarni, and E. D. Greef, *"Data Memory Organization and Optimizations in Application-Specific Systems,"* IEEE Design and Test of Computers, vol. 18, pp. 56-68, 2001.

[49] J. H. Lee, M. Y. Lee, S. U. Choi, and M. S. Park, *"Reducing cache conflicts in data cache prefetching,"* ACM SIGARCH Computer Architecture News, vol. 22, pp. 71-77, 1994.

[50] R. Xu and Z. Li, *"Using Cache Mapping to Improve Memory Performance Handheld Devices,"* Proceedings of 2004 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 106-114, 2004.

[51] W. Wolf and M. Kandemir, *"Memory System Organization of Embedded Software,"* the IEEE, vol. 91, pp. 165-182, 2003.

[52] "GNU Compiler Collection (GCC)", http://gcc.gnu.org/.

[53] Q. Yang, X. Ding, and Z. Chen, *"Discriminant Local Feature Analysis of Facial Images,"* Proceedings of IEEE International Conference on Image Processing (ICIP 2003), pp. 863-866, 2003.

[54] "Intel PXA250 and PXA210 Application Processors Application Developer's Optimization Guide," Intel Corporation, 2003.

[55] J. Bigun, J.Fierrez-Aguilar, J. Ortega-Garcia, and J. Gonzalez-Rodriguez, *"Multimodal Biometric Authentication using Quality Signals in Mobile Communications,"* Proceedings of the 12th International Conference on Image Analysis and Processing, pp. 2-11, 2003.