THE DEVELOPMENT OF AUTOMATED PALMPRINT IDENTIFICATION USING MAJOR FLEXION CREASES

THOMAS CHARLES COOK BSc

A thesis submitted in partial
fulfilment of the requirements of the
University of Wolverhampton for the
degree of Doctor of Philosophy

January 2012

Abstract

Palmar flexion crease matching is a method for verifying or establishing identity. New methods of palmprint identification, that complement existing identification strategies, or reduce analysis and comparison times, will benefit palmprint identification communities worldwide. To this end, this thesis describes new methods of manual and automated palmar flexion crease identification, that can be used to identify palmar flexion creases in online palmprint images. In the first instance, a manual palmar flexion crease identification and matching method is described, which was used to compare palmar flexion creases from 100 palms, each modified 10 times to mimic some of the types of alterations that can be found in crime scene palmar marks. From these comparisons, using manual palmar flexion crease identification, results showed that when labelled within 10 pixels, or 3.5 mm, of the palmar flexion crease, a palmprint image can be identified with a 99.2% genuine acceptance rate and a 0% false acceptance rate. Furthermore, in the second instance, a new method of automated palmar flexion crease recognition, that can be used to identify palmar flexion creases in online palmprint images, is described. A modified internal image seams algorithm was used to extract the flexion creases, and a matching algorithm, based on *k*d-tree nearest neighbour searching, was used to calculate the similarity between them. Results showed that in 1000 palmprint images from 100 palms, when compared to manually identified palmar flexion creases, a 100% genuine acceptance rate was achieved with a 0.0045% false acceptance rate. Finally, to determine if automated palmar flexion crease recognition can be used as an effective method of palmprint identification, palmar flexion creases from two online palmprint image data sets, containing images from 100 palms and 386 palms respectively, were automatically extracted and compared. In the first data set, that is, for images from 100 palms, an equal error rate of 0.3% was achieved. In the second data set, that is, for images from 386 palms, an equal error rate of 0.415% was achieved.

Acknowledgements

I would like to thank my director of studies, Dr. Raul Sutton, and my supervisor, Dr. Kevan Buckley, for their invaluable help, support, and advice throughout the duration of the project. I would also like to thank Esmin Powner and Matthew Eardley for their help in manually identifying palmar flexion creases, Darren Riley for describing and demonstrating automated palmprint identification using IDENT-1, and Abhishek Gupta for his help and support in the important, early stages of the project.

I would also like to thank the University of Wolverhampton Research Centre in Applied Sciences for funding and support, without which this work would not be possible. Finally, I wish to thank my parents, Tim and Celia Cook, and my girlfriend, Charlie Ellis, for their patience and support.

Contents

List of tables

List of figures

1. Introduction

This chapter describes the aims and objectives of the thesis, provides a motivation for the work, and defines the original contributions of the thesis. Furthermore, as an aid to the reader, the structure of the thesis is described.

1.1. Motivation

The aim of this thesis is to investigate the following research question: to what extent can automated palmar flexion crease recognition be used to identify online palmprint images?

In answering this question, the thesis describes the design and implementation of two palmar flexion crease identification and matching methods, which are capable of addressing some of the problems in palmprint identification. Furthermore, to achieve this aim, a number of primary objectives are determined, and a motivation for each objective is described.

In forensic science, traditional palmprint identification strategies use matching of minutia points, that is, the same methodology that is used to match fingerprints, to identify an individual. This method is time consuming for fingerprint examiners, and in database search times, as the surface area of a palmprint is approximately 40 times larger than that of a fingerprint, and may contain up to 8 times the number of minutia points (Jain and Feng, 2009). For this reason, alternative methods of palmprint identification, that complement existing identification strategies, or reduce analysis and comparison times, will be of considerable benefit to palmprint identification communities worldwide. However, as a prerequisite to automated identification, any new identification metric, such as palmar flexion creases, must first be tested to determine its feasibility for identification, before other external factors can be introduced. If these initial tests cannot be established, the usefulness of such a metric is limited. To this end, the first objective of this thesis is to determine the feasibility of palmar flexion crease identification in a forensic context using manual palmar flexion crease identification and matching in complete palmprint images.

Given the feasibility of an identification metric, automated identification represents the single biggest advance in identification technology (Fisher, 2004). In forensic science and biometrics, automated identification enables efficient identification at local, national, and international level, by automating slow, labour-intensive processes previously undertaken only by specially trained examiners (Cole, 2004). Therefore, given the feasibility of palmar flexion creases as a method of palmprint identification, a viable palmar flexion crease identification system, that is, a palmar flexion crease identification system with real-world applications, must be capable of automated identification comparable with existing identification systems. For this reason, the second objective of this thesis is to describe the development of an automated palmar flexion crease identification and matching method, which can be used to improve traditional palmprint identification strategies through integration with existing identification platforms, or as a standalone identification system.

Furthermore, as palmar marks recovered from scenes of crime are often partial, smudged, or otherwise distorted, an effective palmprint identification metric must also be capable of partial palmprint identification. This is highlighted by Libal (2006) and Parker (2006), who determined that 30% of all hand print impressions recovered from scenes of crime are palmprints. To this end, the third objective of this thesis is to investigate the effects of partial palmprint images on palmar flexion crease identification, to determine the feasibility of palmar flexion crease identification in partial palmprint images.

1.2. Thesis contributions

Given the objectives described in Chapter 1.1, the following original contributions have been identified, and are investigated in this thesis:

1. The design and implementation of a manual palmar flexion crease extraction, modification, and matching method, which enables the user to efficiently map the location of palmar flexion creases in a given palmprint image, assess the effects of palmar flexion crease distortions through rotation, displacement, and additive noise, and determine the similarity between two or more palmar flexion creases.
2. The design and implementation of an automated palmar flexion crease recognition and matching algorithm, which is capable of automatically extracting palmar flexion creases from online palmprint images, and calculating a matching score that can be used to determine the similarity between two or more palmar flexion creases.

1.3. Thesis structure

Chapter 2 surveys and critical assesses previous work relevant to this thesis, and provides an indication of the current state of knowledge. Chapter 3 describes a method of manual palmprint identification using palmar flexion creases, and presents an analysis and discussion of experimental results that were collected using the proposed method. In the same way, Chapter 4 presents a new method of automated flexion crease recognition, and presents an analysis and discussion of experimental results compared to manually identified palmar flexion creases. Chapter 5, using the same identification and matching method as in Chapter 4, uses two online palmprint image data sets to determine if automated palmar flexion extraction and matching can be used as an effective method of palmprint identification. Finally, as an aid to the reader, Chapter 6 restates the research objectives, gives a brief overview of the palmar flexion crease identification and matching methods presented in this thesis, presents a summary of the experimental results collected using those methods, and provides an indication of possible improvements and future work.

2. Literature review

The work presented in this thesis has foundations in numerous fields of research, including forensic science, biometrics, computer vision, and pattern recognition. This chapter surveys and critically assesses related work in these fields to provide an indication of the current state of knowledge. The literature review begins with an overview of palmprint identification, in which the definition of a palmprint, the formation and structure of palmprint features, and palmprint identification metrics are discussed. This is followed by a discussion of automated palmprint identification systems, and a summary of a number of computer vision and pattern recognition algorithms, which form the basis for much of the investigative work in this thesis.

2.1. Palmprint formation and structure

Palmprints provide a vast source of identification information that can be used in a number of important fields, such as biometrics (Kong *et al.*, 2009), dermatoglyphics (Qiao *et al.*, 2005), and forensic science (Ashbaugh, 1991a). The general flow of skin, its pattern configuration, and minutiae formation, all contribute towards making the palmprint a unique identification metric (Ashbaugh, 1999a). However, many palmprint identification systems are based on features, such as minutiae, ridge flow, singular points, and flexion creases (Zhang, 2004), which are formed during embryological development. Therefore, an understanding of palmprint morphogenesis, that is, of palmar friction ridge skin development, is necessary when discussing palmprint identification.

Friction ridges first appear in the foetus in the form of localised cell proliferations in the deepest layer of the epidermis, that is, the outer layer of the skin, called the basal layer. The cell proliferations project into the superficial layers of the dermis, that is, the inner layer of the skin, and increase in number as new ridges begin to form between, or at the lateral surface, of existing ridges. The pattern of the epidermal ridges are created by forces driving differential growth in the basal layer, resulting in continued cell proliferation. This causes interruptions in the continuous flow of ridges, and an isolation of short ridge segments. These are known as branchings and islands respectively, and are grouped under the general term of minutiae (Hale, 1949). As the epidermis continues to develop, the ridges at epidermal-dermal junction begin to define the basic ridge configurations of the surface of the skin. The number of primary ridges increase, both in number and width, while penetrating deeper into the underlying dermis (Hale, 1952). Subsequent to the completion of ridge growth, peg-like formations called dermal papillae begin to develop, and irregularities appear in the regions between the epidermal ridges (Babler, 1991). At this stage, epidermal ridges begin to show on the surface of the skin, and the foetus has an epidermal ridge configuration and morphology comparable to that of an adult (Babler, 1977). Figure 1 shows an illustration of a cross section of fully developed friction ridge skin.

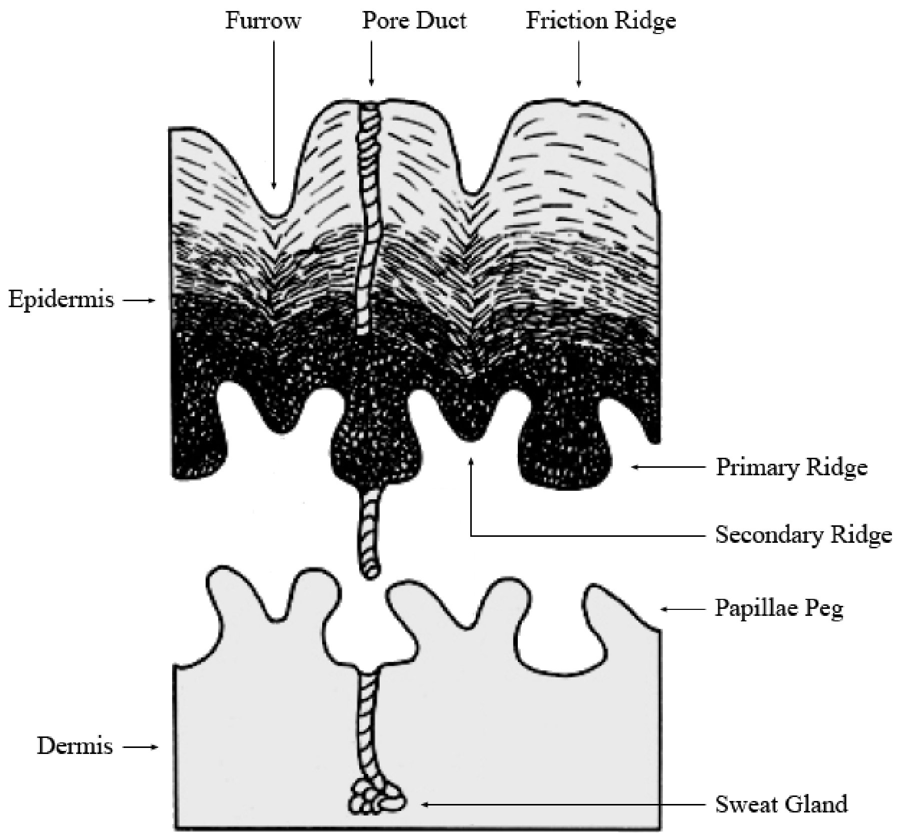


Figure 1. A cross section of friction ridge skin, modified from Ashbaugh (1999a). The dermis is continuous with the epidermis, but has been separated in this diagram for clarity.

To enforce the permanence of the ridge configuration, only cells in the basal layer have the capacity for DNA synthesis and mitosis (Fuchs, 1999). Under a trigger of terminal differentiation, a basal cell will begin the process of keratinisation, undergoing a series of morphological and biochemical changes that culminate in the production of dead, flattened, enucleated epithelial cells, which are continually sloughed from the surface and replaced by inner cells differentiating outwards. Cells generated in the basal layer are connected together by intracellular fibrils which remain connected until exfoliation occurs at the surface of the skin (Matolsy, 1976). The secure intercellular junction prevents epidermal cells generated in the basal layer moving in discord (Matolsy, 1976), and therefore, ensures the intricate detail of the epidermal ridges and furrows remain intact as the cells differentiate towards the surface of the skin (Wertheim and Maceo, 2002).

The second structure that enforces permanence in friction ridge skin is the delicate membrane between the epidermis and dermis (Wertheim and Maceo, 2002). Misumi and Akiyoshi (1984) identified the pattern on the dermal surface as a reflection of the upper surface in a negative image, confirming earlier theories by Hale (1952) and Penrose and O'Hara (1973). Misumi and Akiyoshi (1984) also confirmed that, despite showing smaller and more complicated papillae than a younger person, an older person retains their original fingerprint pattern: "no such variations reflected site-specific characteristics on the fingerprint pattern" (Misumi and Akiyoshi, 1984, p.50). Furthermore, fibres seen on the dermal surface prevent the basal layer sliding along the epidermal-dermal junction (Misumi and Akiyoshi, 1984), thereby preventing a different formation developing on the surface ridge, and enforcing friction ridge permanence.

Similar to epidermal ridge patterns, palmar flexion creases are formed during early intrauterine life, and can be influenced by various factors interfering with normal foetal development. Studies by Lacroix *et al*. (1984), Kimura and Kitagawa (1986), and Stevens *et al*. (1988), based on the observations of palmar development at different foetal ages, have shown that palmar flexion creases develop by thirteen weeks of gestation (Kimura, 1991). However, while the timing of flexion crease development is well established, their origin is less certain. Flexion creases are typically located, even in cases of hand malformation, in correspondence with the underlying joint (Popich and Smith, 1970). However, it is believed that physical movements and environmental factors, as well as genetic programming, contribute to palmar flexion crease development (Ashbaugh, 1999a). This theory is supported by the observation of missing flexion creases in individuals with normal joint function and abnormal underlying bone structure (Kimura, 1991). Furthermore, in a study by Tay (1979), 659 children and 613 first degree relatives were observed with regard to four palmar flexion crease variations. Tay (1979) found that the four flexion crease variations occurred more significantly in the parents of the subjects compared with

400 controls, indicating the presence of genetic factors in their embryogenesis. Table 1 shows the stages of foetal hand development as observed by Kimura and Kitagawa (1986).

Table 1. The stages of foetal hand development (Ashbaugh, 1999a).

Gestation (weeks)	6		8		10		12		14		16		18		20
Hand formation	>	>	>	>	>										
Volar pad growth	>	>	>	>	>	>	>								
Flexion creases					>	>	>	>	>	>					
Volar pad regression							>	>	>	>	>				
Friction ridges							>	>	>	>	>	>			
Secondary creases										>	>	>	>	>	>

After development, the surface of the palm can be divided into topographic areas, where each area is consistent with early volar pad development. Furthermore, three main groups of flexion creases can be defined: major flexion creases, minor flexion creases, and secondary creases (Ashbaugh, 1999a). The major flexion creases are the most prominent and permanent of the flexion crease groups, and, in general, three major flexion creases can be defined: the thenar, the proximal transverse, and the distal transverse.

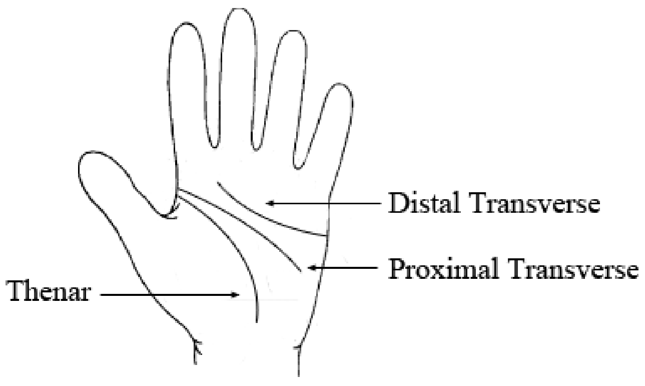


Figure 2. The position of the major flexion creases.

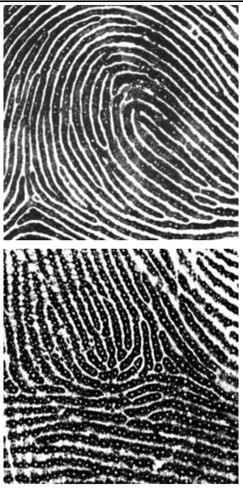
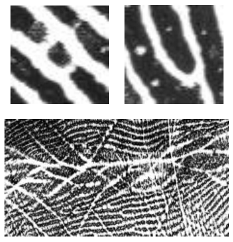
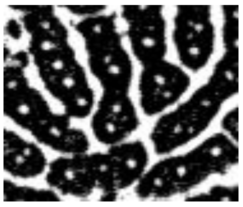
The thenar crease is a curved crease which encircles the thenar eminence, and typically begins in the mid-palm above the wrist. The thenar ends on the radial side of the hand, and often joins the proximal transverse near the radial border, but may also extend to a separate radial terminus. The proximal transverse crease typically starts at the radial border, and extends horizontally in a gentle, concave bow toward the ulnar border on the opposite side of the palm. Finally, the distal transverse crease is located between the proximal transverse and the heads of the underlying metacarpal bones. The distal transverse begins at the ulnar border, and curves toward the radial edge. In some cases, the distal transverse extends further toward the palmar border, and ends between the index and middle fingers (Kimura, 1991). Figure 2 shows the position of the major flexion creases.

At times, the major flexion creases appear to be constructed of smaller creases, called major accessory creases, that branch off or separate the main crease into two or more smaller creases. Major accessory creases can be considered part of the secondary crease group, which may be found anywhere on the palmar surface, and are typically of short length (Ashbaugh, 1999a). Finally, the minor flexion crease group contains creases that are more stable than secondary creases, but are typically more variable in presence, prominence, and length than the major flexion creases. While many minor flexion creases may be present on the palmar surface, they often differ in location and direction from person to person, and so generally, cannot be reliably defined (Kimura, 1991).

2.2. Palmprint identification

Palmprint identification is the process of matching an unknown palmprint against a database of known prints to establish a person’s identity. To this end, given an understanding of palmprint structure, that is, of friction ridge skin and flexion crease structure, a number of palmprint identification metrics can be defined. Ashbaugh (1999b) proposed that three levels of information, which describe the visible features of the skin, that can be distinguished in a section of friction ridge skin: level 1, level 2, and level 3. Depending on the size (or magnification) and clarity of the friction ridge skin impression, an identification metric may be classified as level 1, 2, or 3, as shown in Table 2.

Table 2. A summary of the three levels of friction ridge skin features.

Level	Definition	Examples
1	Level 1 features refer to the general flow of friction ridge skin and the overall pattern formed by the flow of papillary ridges (Champod *et al.*, 2004). A number of classification systems, most notably Galton (1892) and Henry (1900), which simplify the continuum of papillary ridge flow, have been proposed for level 1 features. However, as stated in Ashbaugh (1999a), first level detail is not sufficient to individualise as the flow of ridges may be influenced by heredity through the volar pads, or repeated by chance due to the limited number of pattern configurations. Therefore, first level detail is often used during the identification process only to reduce the number of possible donors, or to estimate the position of an isolated section of friction ridge skin (Tietze and Witthuhn, 1975 in Champod *et al.*, 2004).	
2	Level 2 features refer to major ridge path deviations, or minutiae, palmar flexion creases (Ashbaugh, 1991a), and occasional features such as scars, warts, creases, and wrinkles (Champod *et al.*, 2004). If sufficient information is present, second level detail can be used to individualise, and often accounts for a major part of the identification process (Ashbaugh, 1999a).	
3	Level 3 features refer to the relative location of the pores, the shape and alignment of each ridge unit (Champod *et al.*, 2004), and small features found in accidental damage to the friction ridges. Third level details are compared during the identification process, and often form part of the comparison at the second level (Ashbaugh, 1999a).	

The identification process using first, second, and third level detail typically follows an analysis, comparison, evaluation, and verification (ACE-V) structure, as developed by the Royal Canadian Mounted Police (Huber, 1972), and adopted by Ashbaugh (1991b). The first stage in this process, analysis, requires an expert, that is, a fingerprint examiner, to survey the quality of the friction ridge skin impression (Ashbaugh, 1999a). The analysis stage allows the fingerprint examiner to determine, among other things, what part of the palmar or plantar surface deposited the print, which features are visible on the print, how clearly the print was deposited, and which features were affected by pressure, distortion, or the nature of the recipient surface (Champod *et al.*, 2004). If the fingerprint examiner determines that the print is of sufficient quality for assessment, an iterative comparison, typically using an automated identification system, between the unknown print and a known print is performed (Ashbaugh, 1999a). The outcome of the comparison process is a comparison image, as shown in Figure 3, with which the similarities and differences between the unknown print and one or more known prints can be highlighted (Champod *et al.*, 2004).

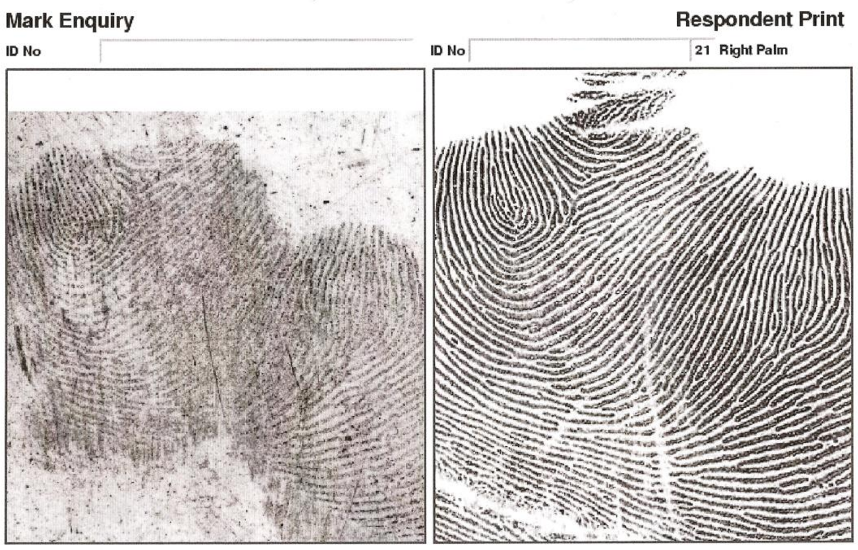


Figure 3. A comparison of two sections of friction ridge skin.

After the comparison stage, the fingerprint examiner then evaluates each charted comparison to reach one of three conclusions: elimination, where the specific details in each print are not the same, and therefore the donors are different; individualisation, where the specific details in both prints are the same, and there is sufficient uniqueness to eliminate all other possible donors; or insufficient uniqueness to individualise or eliminate, where, although a number of specific details may be in agreement, they lack quality or quantity, and so an opinion cannot be formed (Ashbaugh, 1999a). At this stage in the identification process, after a thorough analysis, comparison, and evaluation of the unknown print, the fingerprint examiner is able to reach a valid conclusion. However, as the opinion of individualisation is subjective, a verification process must be performed (Champod *et al.*, 2004). Verification is a peer reviewed consultation that is used to verify the integrity of the identification process, ensure the objectivity of the fingerprint examiner, and confirm that there is sufficient evidence to justify the stated conclusion (Ashbaugh, 1999a). After the verification process, the friction ridge skin impression can be reliably identified, eliminated, or determined to be inconclusive.

The ACE-V protocol aims to provide a structured and documented approach to friction ridge skin identification. However, following several Daubert hearings in legal proceedings in the United States, that is, a challenge to the admissibility of expert evidence based on the validation of scientific techniques, where the acceptability of friction ridge skin evidence was challenged, a number of critical reactions have been presented (Cole, 2006; Haber and Haber, 2008). A comprehensive discussion and critical analysis of the ACE-V protocol can be found in Tuthill (1994), Ashbaugh (1991b; 1999a), Clark (2002), Epstein (2002), Champod *et al*. (2004), and Champod (2008).

2.3. Automated identification systems

Automated palmprint identification has received considerable attention in literature in recent years, resulting in the growth of commercial identification systems in law enforcement and biometrics. In forensic science, the National Policing Improvement Agency (NPIA) in the United Kingdom, and the Federal Bureau of Investigation (FBI) in the United States, have included initiatives for integrated national palmprint identification in their existing automated fingerprint identification systems (AFIS) (Federal Bureau of Investigation, 2005b; Police IT Organisation, 2005). While in biometrics, palmprint identification methods based on texture analysis, feature line extraction, and subspace analysis have shown promising recognition rates, that are comparable with other methods of identification, such as face recognition and automated fingerprint identification (Zhang, 2004; Kong *et al.*, 2009).

A typical automated identification system consists of four distinct stages: acquisition, pre-processing, feature extraction, and matching, and should be designed in consideration of five main objectives: cost, user acceptance and environmental constraints, accuracy, computation speed, and security. A practical identification system should balance all of these aspects (Kong *et al.*, 2009).

2.3.1. Acquisition

There are two distinct methods of palmprint acquisition: offline and online. In offline acquisition, palmprint data is captured from a non-digital source. Instead, a reference palmprint is collected by printing with ink on to paper, or by visible and latent mark deposition detection at scenes of crime (Champod *et al.*, 2004), then digitised, often with a scanner, and stored for further processing (Zhang and Shu, 1999). Offline reference palmprints can be collected in a controlled environment, in which set procedures and processes are followed. However, at scenes of crime, visible and latent palmprint deposition is caused by a complex mixture of natural secretions and contaminants from the environment, in which surface characteristics play an important role. Therefore, the properties of each surface must be considered before any attempt is made to develop a mark (Champod *et al.*, 2004). To this end, a wide range of detection sequences, based on varying conditions of temperature, surface structure, and electrostatic forces, have been suggested in literature. This literature will not be reviewed here; however, a comprehensive review of offline scenes of crime acquisition can be found in Margot and Lennard (1994) and Champod *et al*. (2004).

Unlike in offline acquisition, where considerations must be made before a palmprint image can be captured, a real-time palmprint identification system requires a palmprint scanner that can quickly capture high quality palmprint images. To this end, online palmprint acquisition is the most direct way to digitise palmprint data (Zhang *et al.*, 2003). An online system captures a palmprint using a capture sensor, usually a scanner, digital camera, or video recorder, that is typically connected directly to the identification system, obviating the need for a third medium, such as paper (Zhang, 2004). A typical device must be able to acquire good quality palmprint images, which users find comfortable and intuitive to use. Wong *et al.* (2005) identified a number of user interface and optical system requirements that should be present in an effective palmprint acquisition device:

- The palmprint should be captured in real-time to provide a development platform for practical applications.
- The palmprint image should be at a quality as to enable subsequent image processing.
- The system should provide the best performance for reasonable costs.
- The system should provide a user interface so that the user feels comfortable during the acquisition process, including easy guides to position the palm on the device and at an appropriate size.

The structure of one proposed online palmprint acquisition system is shown in Figure 4. The user interface consists of a flat platen surface, which acts as an input channel for the user and system, and for the acquisition of palmprint data. The light from the palm passing through the aperture is converged by the lens to form an image. A ring light source provides a high intensity white light that is used to increase the contrast of the palmprint when the skin surface is uneven. A video frame grabber processes the analogue signal from a charge-coupled device (CCD) sensor, and an analogue to digital conversion takes place in the on-board processor, before the palmprint image is stored in system memory (Zhang, 2000).

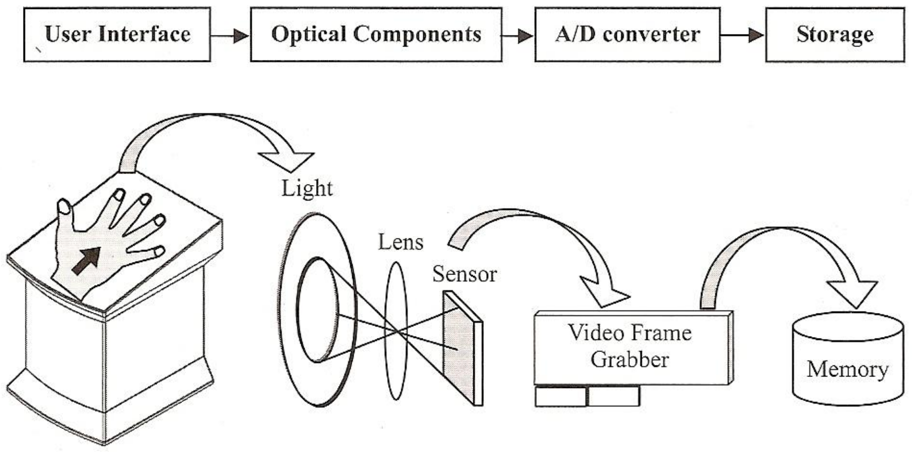


Figure 4. The structure of an online palmprint acquisition system (Zhang, 2004).

The first palmprint capture device of this type was created in December 1999 at The Hong Kong Polytechnic University (Zhang, 2000). The device was made using an L-shaped plastic box, a light source, a mirror, a glass plate, and a CCD camera. However, after testing it was decided that an image formed through a mirror was not as effective as direct reflection, because the second surface created a ghost image on the final output. Furthermore, the glass plate used to hold the palm distorted the surface of the skin so that features of the palm were not clear enough for feature extraction (Zhang, 2004). To alleviate these problems, a number of improved palmprint acquisition devices were designed.

The most important concern in palmprint acquisition is the quality of the resultant image. Therefore, to address the problems arising from their previous L-shaped design, The Hong Kong Polytechnic University group made a number of changes to their palmprint acquisition system. Using a traditional straight-through optical axis, a long horizontal tube device was designed, allowing the glass plate and mirror to be removed, and for the palmprint to be captured without distortion. Furthermore, a more powerful light source was used in the optical system to illuminate the palm, in an approach that greatly improved image quality (Zhang, 2004).

After testing, a number of users found the horizontal arrangement of The Hong Kong Polytechnic University's second device to be inconvenient. Consequently, a vertical device was created. To ensure correct lighting, the light source was changed from a standard light bulb to a fluorescent ring light, which, in terms of illumination, produces a more uniform palmprint image (Zhang, 2004). Furthermore, a flat platen surface was designed, allowing the palm to be digitised without contact, and thereby eliminating distortion, which is often a problem in traditional devices. In order to help users correctly place their palm, a number of pegs, as shown in Figure 5(a), were added for guidance. Additionally, a marked area, as shown in Figure 5 by a dashed line, was removed from the flat platen surface to allow the CCD camera underneath to acquire a palmprint image. With this design, an image can be obtained from large or small hands, while still retaining the important palmprint features (Wong *et al.*, 2005).

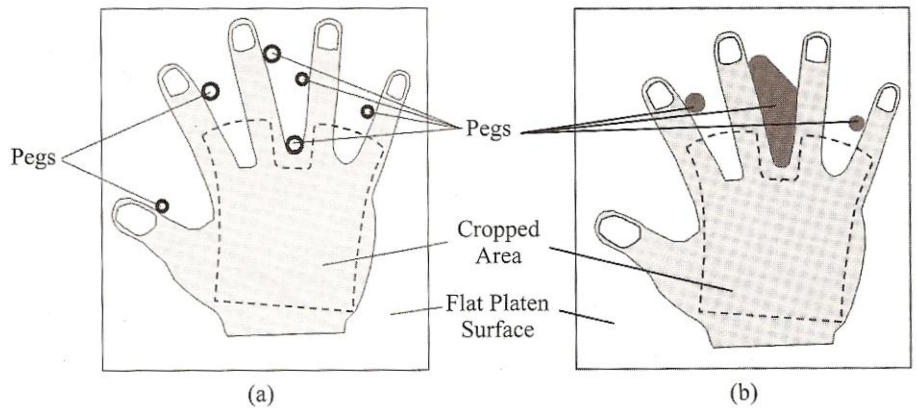


Figure 5. A flat platen surface design with a) guide pegs, and b) simplified guide pegs (Zhang, 2004).

However, testing showed that a proliferation of pegs on the flat platen surface confused some users as to how they should place their palms. Wong *et al.*'s (2005) final design takes this into consideration, with the use of pegs balancing user convenience and system stability. As shown in Figure 5(b), the flat platen surface requires only three pegs: a large triangular peg to guide the middle and ring fingers, a round peg on the left to guide the index finger, and a final peg on the right to guide the little finger. This provides a clear and easy-to-use user interface, and a stable palmprint alignment positioning mechanism (Wong *et al.*, 2005).

A schematic diagram of Wong *et al.*'s (2005) final design is shown in Figure 6. When an enhanced flat platen surface, fluorescent ring light, ½ inch lens, and ⅓ inch CCD sensor are used, palmprint features such as flexion creases, wrinkles, and ridge texture can be obtained at 150 dots per inch (dpi) in an average of 0.09 seconds.

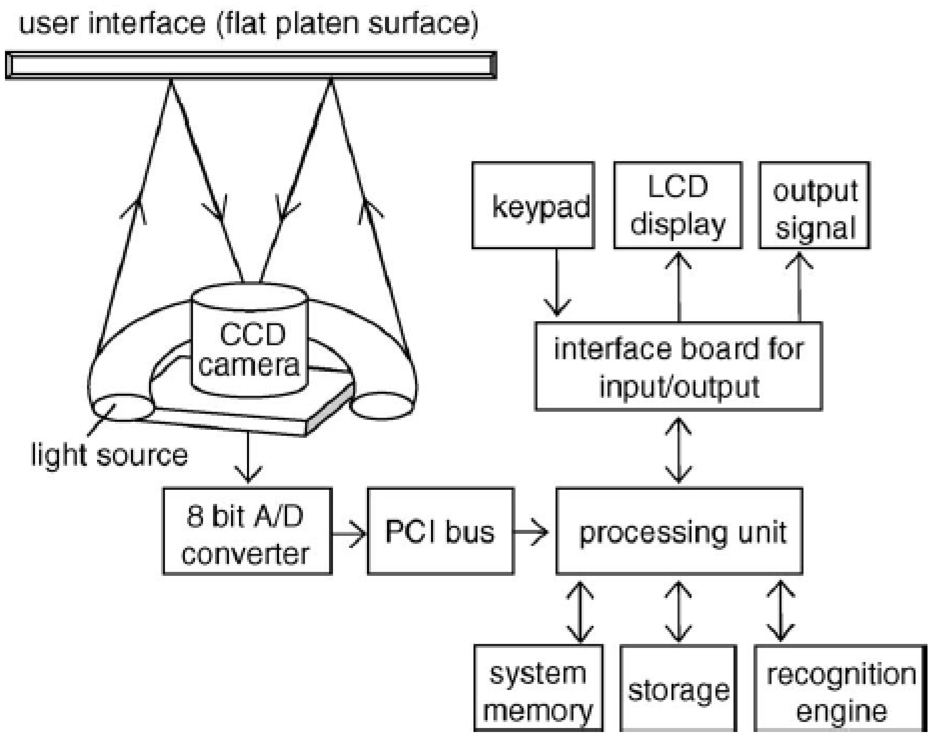


Figure 6. A schematic design of Wong *et al.*'s (2005) final palmprint acquisition device.

For biometric applications, such as access control, time and attendance systems, and personal verification, where ridge texture, flexion creases, and wrinkles are used as identification metrics, low resolution palmprint images are acceptable. However, in forensic science, where low level details, such as minutiae, pores, and ridge flow, contribute to offender identification, high resolution palmprint acquisition devices are required. Furthermore, to qualify for FBI Integrated Automated Fingerprint Identification System (IAFIS) certified status (Federal Bureau of Investigation, 2005a), or to meet the requirements of the Interpol standard for exchanging computerised fingerprint images (Interpol, 2004), which also includes palmprints (National Institute of Standards and Technology, 1999), the device must be capable of producing images that exhibit good geometric fidelity, sharpness, detail rendition, grey-level uniformity, and greyscale dynamic range, with low noise characteristics, and without creating any significant artefacts, anomalies, false detail, or cosmetic image restoration effects (Federal Bureau of Investigation, 2005a). In addition, the acquisition device must also adhere to output resolution guidelines:

> The scanner's final output resolution, in both sensor detector row and column directions, shall be in the range: (R–0.01R) to (R+0.01R) and shall be gray-level quantized to 8 bits per pixel (256 gray-levels). The magnitude of "R" is either 500 pixels per inch (ppi) or 1000 ppi; a scanner may be certified at either one, or

both, of these resolution levels. The scanner's true optical resolution shall be greater than or equal to R (Federal Bureau of Investigation, 2005a, pp.F-2).

Since the introduction of the IAFIS image quality standard, a number of certified palmprint acquisition devices, often called Livescan, have become commercially available, many of which use optical frustrated total internal reflection (FTIR) to detect the presence of finger or palm friction ridge skin. When light passes through one medium, such as a lens or prism, at a given angle of incidence with a given index of refraction, to another medium, such as a finger or palm, energy from the light will be reflected at an angle of reflection equal to the angle of incidence (Schneider, 2007). Therefore, a light ray incident upon a specular reflector, that is, upon an area not in contact with the lens or prism, such as a valley of friction ridge skin, will be reflected back without obstruction at an angle of reflection equal to that of the angle of incidence. Conversely, a light ray incident upon a surface in contact with the lens or prism, such as a ridge of friction ridge skin, will be partially absorbed, and the amount of energy reflected back at the specular angle will be significantly reduced (Schneider and Wobschall, 1991). Using this theory, and by placing a photo detector at an appropriate angle to detect the reflected light, as shown in Figure 7, an image of the friction ridge skin can be captured.

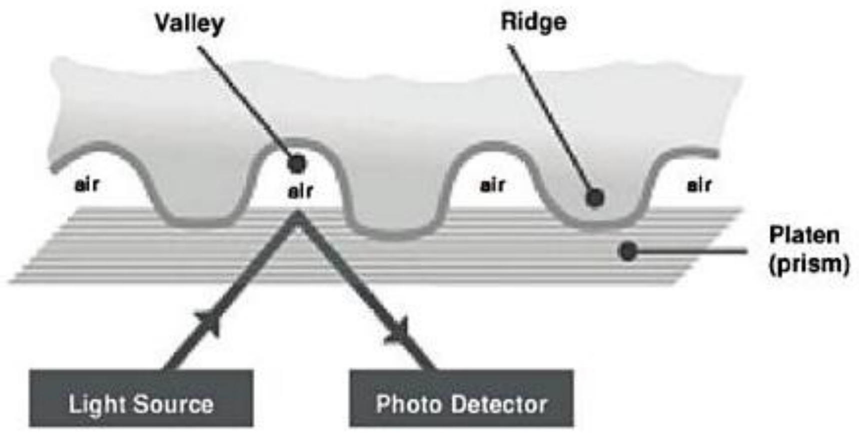


Figure 7. Imaging friction ridge skin using optical frustrated total internal reflection (Schneider, 2007).

However, the physics of FTIR present a number of limitations when capturing certain types of friction ridge skin. The skin must be in contact with the platen surface, or within the depth of penetration, to interfere with the light source. For this reason, if the skin contains irregular ridges, or if contamination on either the skin or platen is greater than the depth of penetration, then an accurate image of the friction ridge skin is not possible (Schneider, 2007).

In response to these limitations, some palmprint acquisition devices use ultrasonic imaging to capture an image of the palmar friction ridge skin (Schneider, 2007). Ultrasonic imaging of friction ridge skin is based on the difference in acoustical impedance between human tissue, that is, a ridge, and air, that is, a valley. An image of the friction ridge skin can be produced from the difference in acoustic energy between the two receivers (Schneider and Wobschall, 1991). However, to obtain a uniform, high quality image, the reflectivity coefficient, that is, the relative acoustic impedance between a ridge and a valley, along the surface of the skin must vary significantly. This can be achieved by maximising the ultrasonic return caused by a valley, while minimising the ultrasonic return caused by a ridge, allowing most of the acoustical energy to pass into the skin (Schneider, 2007). Figure 8 shows an example of a fingerprint image captured using FTIR and ultrasonic imaging, both of which provide an accurate representation of the friction ridge skin of the finger (Schneider and Gojevic, 2001).

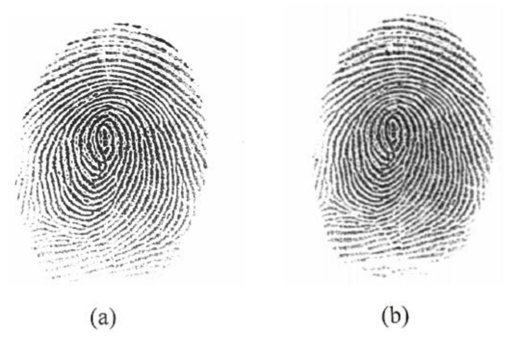


Figure 8. Example fingerprint images taken using a) optical frustrated total internal reflection, and b) ultrasonic imaging (Schneider and Gojevic, 2001).

2.3.2. Pre-processing

As a palmprint is captured, it may exhibit some degree of distortion through varying conditions of time, temperature, humidity, brightness, or other external factor, regardless of the acquisition method. Pre-processing aims to correct these distortions, by placing each image under the same coordinate system, so that the correct area of each palmprint can be extracted for feature extraction and matching (Zhang and Shu, 1999). However, before palmprint pre-processing can take place, a number of specific notations and definitions must be considered.

In general, three major palmar flexion creases can be defined on the surface of the palm: the distal transverse, the proximal transverse, and the thenar. Due to the stability of the major flexion creases, two end points, a and b, and their midpoint, o, can be obtained from their intersection with each side of the palm. Furthermore, using these definitions, Zhang and Shu (1999) identified a number of significant properties:

- The location of the end points and their midpoint are rotation invariant in a given palmprint image.
- A two-dimensional right angle coordinate system can be established, of which the origin is the midpoint, with the main axis passing through the end points.
- A palm can be divided into three regions, as shown in Figure 9, named the finger-root region (I), the inside region (II), and the outside region (III), by connections between the end points and their perpendicular bisector.
- The size of a palm can be estimated by both the Euclidean distance between the end points and the length of their perpendicular bisector.

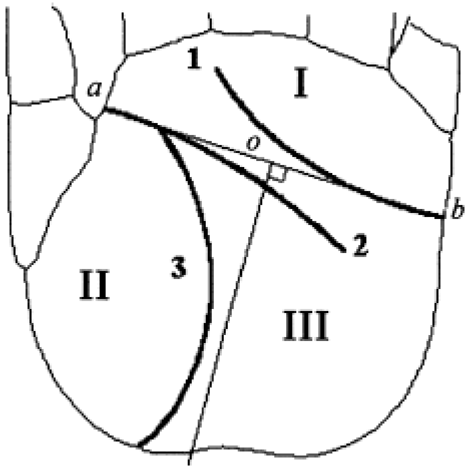


Figure 9. The features of a palmprint: 1, the distal transverse crease, 2, the proximal transverse crease, and 3, the thenar crease; I, the finger-root region, II, the inside region, and III, the outside region; and the datum points, a b, and o (Zhang and Shu, 1999).

Using these features, Li *et al.* (2003b) proposed an offline palmprint image alignment method, as shown in Figure 10. The outside boundary of an offline palmprint image is usually clear and stable, and can often be described by a straight line. As such, a vertical axis can be defined along the outside boundary of the image to describe its orientation. Furthermore, an intersect point, between the outer boundary and the major flexion creases, as described by Zhang and Shu (1999), can be determined as the origin for the rotation.

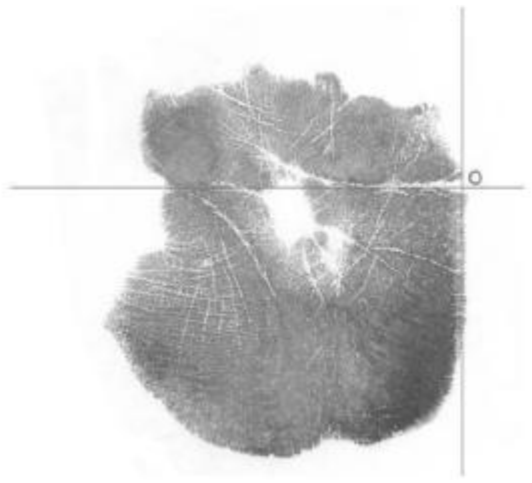


Figure 10. A two-dimensional right angle coordinate system using two invariant features: outer boundary detection, and the flexion crease end points (Li *et al.*, 2003b).

Using this coordinate system, any number of palmprint images may be rotated to the same direction, so that a rotation and translation invariant comparison may be performed. The vertical axis of a given offline palmprint image can be defined as $y = ax + b$, where

$$a = \overline{y} - b\overline{x}, \text{ and } b = \frac{l_{xy}}{l_{xx}},$$

and $\overline{x}$, $\overline{y}$, l_{xx}, and l_{xy} are defined as:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \,,\; \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \,,\; l_{xx} = \sum_{i=1}^{n}(x_i - \overline{x})^2, \text{ and } l_{xy} = \sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y}),$$

where $(x_i, y_i)(i = 1, \ldots, n)$ are points on the edge of the outer boundary of the palmprint (Li *et al.*, 2003b). Figure 11 illustrates vertical axis determination by this method, where Figure 11(a) is the original image, Figure 11(b) is a binary image on which boundary tracing is performed, and Figure 11(c) is the outer boundary and vertical axis, which describes the direction of the palmprint image.

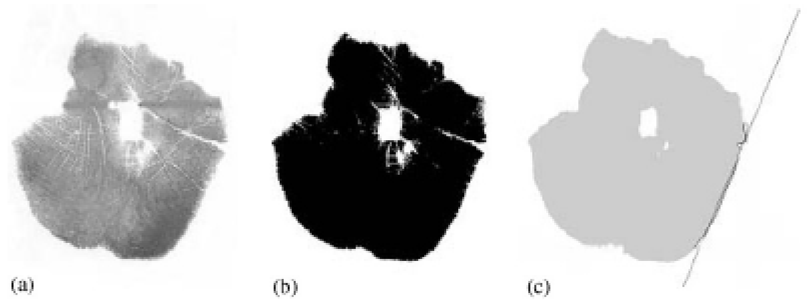


Figure 11. The process of vertical axis determination (Li *et al.*, 2003b).

To determine the origin of the coordinate system, the end point of the distal transverse must be detected. After the vertical axis is discovered, a projection, that is, the sum of pixel values in each row, is calculated to obtain the position of the end point of the distal transverse (Li *et al.*, 2003b). This process is shown in Figure 12. Figure 12(a) shows a rotated palmprint image from which a rectangular sub-image, shown in Figure 12(b), is extracted. From a horizontal projection of this sub-image, the intersect point between the distal transverse and the outer boundary can be determined, as shown in Figure 12(c).

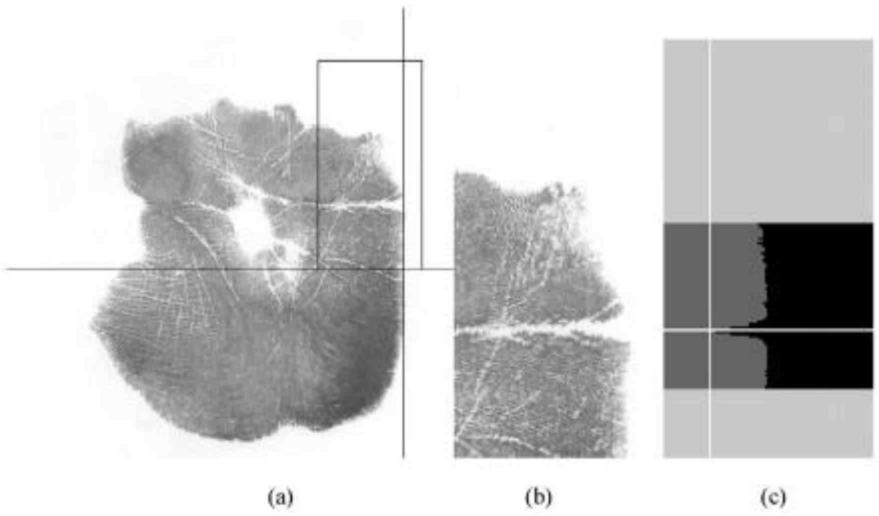


Figure 12. A palmprint coordinate system origin detection process (Li *et al.*, 2003b).

Having determined the vertical axis and origin of the coordinate system, each palmprint image can be rotated so that all palmprints are aligned in the correct position. However, since the image quality and format of offline palmprints differs from that of online images, the offline pre-processing method proposed by Li *et al.* (2003b) cannot be applied as successfully to online palmprint images. As a result, Li (2003) proposed a square-based segmentation approach for online palmprint images.

Li (2003) defined three key points, based on the position of the fingers, from which an orthogonal coordinate system can be derived. An area, or region of interest, with a fixed size can then be extracted from a predefined position in the coordinate system, the size and shape of which can be determined as required. The three key points, $k1$, $k2$, and $k3$, are shown in Figure 13, and represents the midpoint between the index finger and the middle finger, the midpoint between the middle finger and the ring finger, and the midpoint between the ring finger and the little finger. $k1$ and $k3$ are connected to form a vertical axis, then a line, through $k2$, perpendicular to the vertical axis, is used to determine the origin of the coordinate system (Li, 2003).

To obtain the three key points, and subsequently the region of interest, Li (2003) defined a number of key stages:

1. Use a threshold, α, to convert the original image into a binary image.

$$I_{binary}(i,j) = \begin{cases} 1 & I_{original}(i,j) \geq \alpha \\ 0 & I_{original}(i,j) < \alpha \end{cases}$$

2. Smooth the binary image using a Gaussian filter, A.

$$I_{smoothed} = I_{binary} * A$$

3. Trace the boundary of the holes between the fingers.
4. Calculate the centre of gravity between the holes and decide the key points, $k1$, $k2$, and $k3$.
5. Line up $k1$ with $k3$ to obtain a horizontal axis, and draw a line through $k2$, perpendicular to the horizontal axis, to determine the origin of the coordinate system.
6. Once the coordinate system has been determined, the region of interest can be extracted.

As shown in Figure 13, the area, $ABCD$, is the region of interest, the position of A in the coordinate system is fixed, and the size of $ABCD$ is the same in all palmprint images. An assumption is made that the fingers are not joined together, and that at least four fingers are present in the palmprint image (Li, 2003).

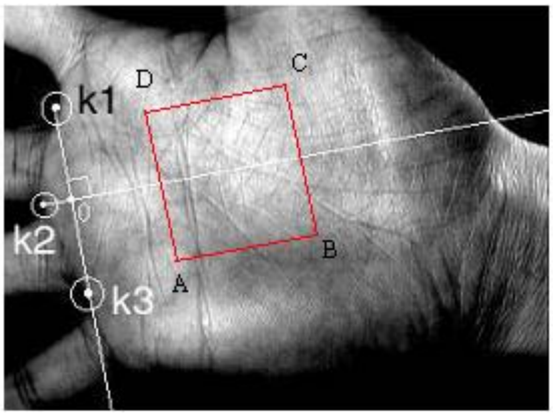


Figure 13. An orthogonal coordinate system on a palmprint image (Li, 2003).

In contrast to square-based pre-processing methods, a circle-based approach, as proposed by Li *et al.* (2003a), can be used to extract as large an area as possible from the central part of an online palmprint image. The palmprint image is first converted into a binary image, and the boundary is smoothed to obtain an accurate contour of the palmprint. The biggest inscribed circle for the contour is calculated, and its centre and radius are obtained. As each palmprint is different, the radii will be variable across individual palmprint samples. If the radii of two samples are significantly different, it can be concluded that the samples are from two different palms. Otherwise, further matching is needed within the circle. Once the circle is determined, extraction involves obtaining all the pixels inside the circle (Li *et al.*, 2003a). Figure 14 shows several segmented central part palmprint sub-images.

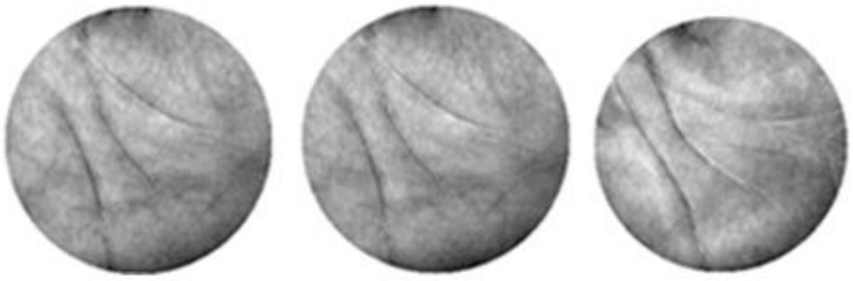


Figure 14. Palmprint samples taken from the same palm using an inscribed circle-based pre-processing approach (Li *et al.*, 2003a).

A comparison and analysis of square- and circle-based methods was performed by Li (2003), in which three criteria were used to evaluate each segmentation approach:

1. How precisely does it extract the required region of interest?
2. Can the algorithm handle distortions in the palmprint image?
3. Does the extracted region contain an effective number of useful features?

Commonly, there are three major flexion creases visible on a palmprint image. From the end points of these flexion creases, two key points, A and B, can be defined, as shown in Figure 9 as a and b, and again in Figure 15. The centre of the square region of interest, $O1$, and of the circular region of interest, $O2$, can also be defined. In addition, the distance between $O1$ and A can be denoted as $D_{square-a}$, the distance between $O1$ and B as $D_{square-b}$, the distance between $O2$ and A as $D_{circle-a}$, and the distance between $O2$ and B as $D_{circle=b}$ (Li, 2003).

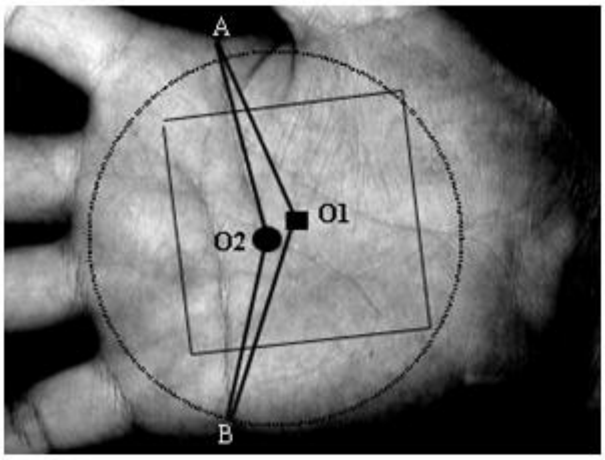


Figure 15. Definitions for Li's (2003) pre-processing accuracy test: flexion crease end points, A and B, the centre of the square region of interest, O1, and the centre of the circle region of interest, O2.

To test the evaluation criteria, Li (2003) took four hundred palmprint images from eighty palms, and manually determined the two key points, A and B, for each image. The distances, $D_{square-a}$, $D_{square-b}$, $D_{circle-a}$, and $D_{circle-b}$, were automatically calculated, and the average distance between the centre points of five square and five circle regions was determined. The pixel shift of $D_{square-a}$ and $D_{square-b}$ was six pixels, while the shift of $D_{circle-a}$ and $D_{circle-b}$ was five pixels, thereby showing the circle-based approach to be a little more accurate than the square-based approach between samples of the same palm (Li, 2003). Furthermore, in some cases a palmprint cannot be segmented correctly using a square-based method, while it can be using a circle-based method. In a database of over ten thousand samples, Li (2003) determined that only 80% could be correctly segmented using a squared-based approach, but all of them could be segmented using a circle-based approach.

Palmprint pre-processing involves extracting a region of interest from a palmprint image for feature extraction and matching. For square-based approaches, Li (2003) determined that this region of interest is normally 128×128 pixels (16348 pixels) in size, while the average size for circle-based approaches, calculated from a database of ten thousand images, is $100 \times 100 \times 3.14$ pixels (31400 pixels). Therefore, it was concluded that more palmprint features can be extracted by circle-based segmentation than in analogous squared-based methods. Furthermore, different palms may exhibit large differences between their circle sizes, providing a useful means of palmprint classification, in which the circle size is the first step in the classification criteria. If the circle size of two palmprint samples is too variable, it may be concluded immediately that they are from different palms. Only when the circles of two images are of a similar size, will they be compared with other features in order to decide whether they are from the same palm (Li, 2003).

However, despite Li's (2003) work on circle-based pre-processing, square-based methods continue to prevail, so much so that many feature extraction and matching algorithms are reliant on square or rectangular regions of interest (Kong *et al.*, 2009). Han (2003), Zhang *et al.* (2003), Kumar *et al.* (2003), Poon *et al.* (2004), and Han (2004) have all proposed pre-processing methods for square- or rectangular-based palmprint segmentation. However, in each of these methods, the same basic algorithm is employed:

1. The palmprint image is converted to a binary image using a given threshold.
2. The contour of the hand and/or fingers are extracted from the binary image.
3. A number of key points are detected, and a coordinate system is defined.
4. The central part sub-image is extracted.

Poon *et al.* (2004) and Zhang *et al.* (2003) used two key points, one between the index finger and the middle finger, and another between the ring finger and little finger, with a perpendicular axis, to define their coordinate systems. Kumar *et al.* (2003) used the centre of a binary hand image, without defining any key points, as the centre coordinate for extracting a fixed square region of interest. While Han *et al.* (2003) and Han (2004), used the position of the fingers. As shown in Figure 16, Han *et al.* (2003) used the tip of the middle finger, and a horizontal axis from the base of the same finger, to extract the region of interest.

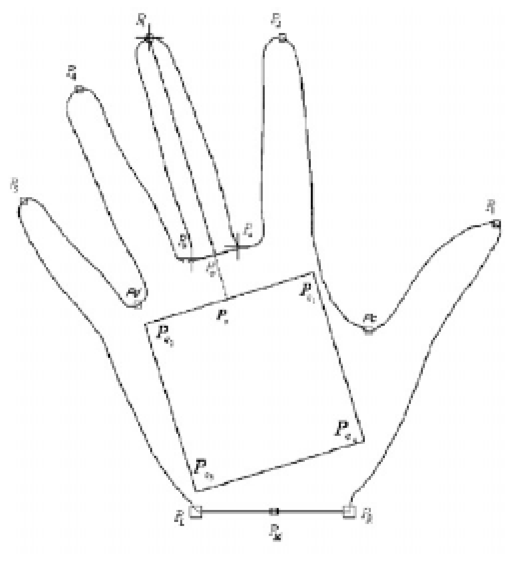


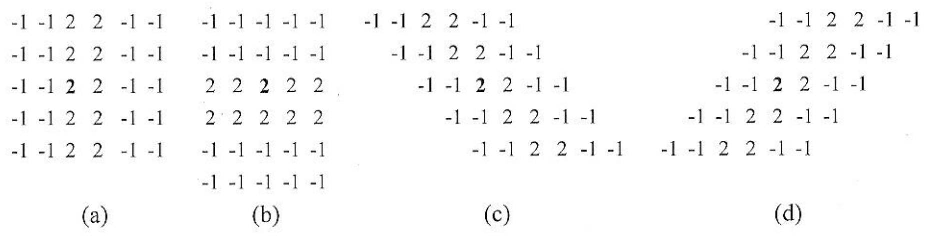
Figure 16. Han *et al.*'s (2003) coordinate system and region of interest.

Furthermore, using the same approach as Han *et al.* (2003), Han (2004) found the position of index finger, middle finger, and ring finger to further refine his finger-based pre-processing method.

2.3.3. Feature extraction and matching

After pre-processing, palmprint identification, as with any automated identification system, requires two further operations: feature extraction and matching. In biometrics, feature extraction and matching methods using texture analysis, subspace analysis, and feature line extraction, have been covered extensively in literature. This review will consider line-based feature extraction and matching methods for primary flexion creases, secondary flexion creases, and wrinkles. The literature on texture analysis and subspace analysis will not be reviewed here, but is instead available in Zhang (2004) and Kong *et al.* (2009).

Feature line extraction is an important stage in image processing and verification that has resulted in a number of proposed general line detection algorithms. However, many general line detection algorithms cannot generate a precise edge map of textured stripe images, such as those of offline fingerprints or palmprints (Wu and Li, 1997). Therefore, a specific feature line extraction algorithm is often required for these types of images. Kung *et al*. (1995) used a face and palm recognition algorithm based on a low-resolution palmprint edge map, and a decision-based neural network, to recognise palmar line patterns from similar palm configurations. Rodrigues and Silva (1996) and Boles and Chu (1997) used a combination of Sobel edge detection and morphological filtering to construct a binary edge image, which was then used to compare palmar lines. Wu and Li (1997) proposed a pyramid edge detection method for stripe images, which, when tested against a series palmprint images, performed well. However, Wu and Li’s (1997) method is only capable of detecting strong flexion creases. This can be a problem in some images as many important line features, including major and minor flexion creases, have the same width, but a shorter length, as coarse wrinkles, making pyramid edge detection cumbersome in acquiring palmprint line features. Furthermore, the mass of ridges and fine wrinkles in an offline palmprint image may dirty the line features (Zhang and Shu, 1999).



```
-1 -1 2 2 -1 -1   -1 -1 -1 -1 -1   -1 -1 2 2 -1 -1                          -1 -1 2 2 -1 -1
-1 -1 2 2 -1 -1   -1 -1 -1 -1 -1      -1 -1 2 2 -1 -1                    -1 -1 2 2 -1 -1
-1 -1 2 2 -1 -1    2  2  2  2  2         -1 -1 2 2 -1 -1              -1 -1 2 2 -1 -1
-1 -1 2 2 -1 -1    2  2  2  2  2            -1 -1 2 2 -1 -1        -1 -1 2 2 -1 -1
-1 -1 2 2 -1 -1   -1 -1 -1 -1 -1               -1 -1 2 2 -1 -1  -1 -1 2 2 -1 -1
                  -1 -1 -1 -1 -1
      (a)               (b)                  (c)                          (d)
```

Figure 17. Four improved directional templates for line segment determination: a) vertical, b) horizontal, c) left diagonal, and d) right diagonal (Zhang and Shu, 1999).

An improved pyramid edge detection algorithm was proposed by Zhang and Shu (1999). Using four directional templates, Zhang and Shu’s (1999) method is able to effectively extract the flexion creases and wrinkles from an offline palmprint image. Their algorithm consists of the following three steps:

1. Determine the vertical line segments using four near-vertical templates, as shown in Figure 17, apply morphological thinning, and then remove short line segments, leaving only the major lines.

2. Using the same method, detect lines in the horizontal, left diagonal, and right diagonal directions.

3. Combine the results of each direction, and remove overlapping segments.

The results of Zhang and Shu’s (1999) improved line detection algorithm for offline palmprint images is shown in Figure 18.

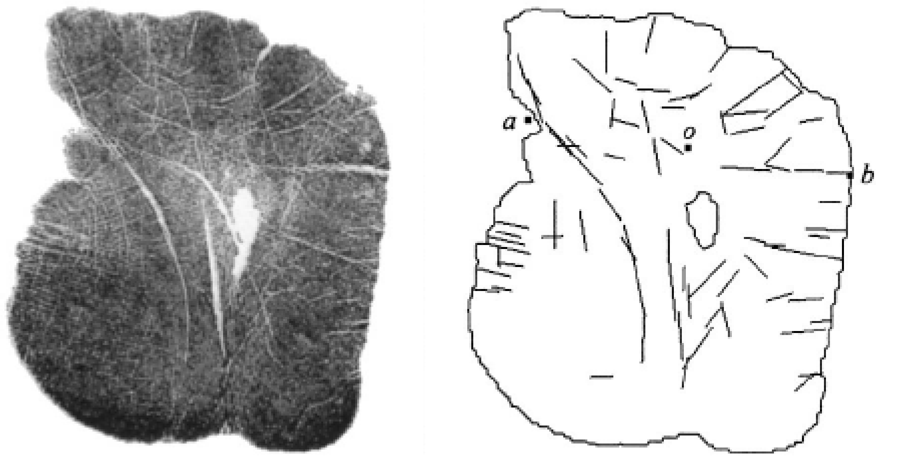


Figure 18. The results of Zhang and Shu's (1999) improved directional line detection algorithm.

Following feature line detection and extraction is feature line matching. In general, there are many ways to represent a single line. One such method, as used by Keegan (1977), is to store the end points of each straight line in a line segment. In a two-dimensional right angle coordinate system, line segments can be described by end points, $(x_1(i), y_1(i)), (x_2(i), y_2(i)), i = 1, \ldots, n$, where n is the number of line segments. The end points of each line segment can then be exchanged so that $x_1(i) < x_2(i), i = 1, \ldots, n$, and if $x_1(i) = x_2(i)$, so that $y_1(i) \leq y_2(i)$. Furthermore, a number of parameters, such as slope, intercept, and angle of inclination, α, can be calculated for each line segment:

$$\mathrm{slope}(i) = \frac{y_2(i) - y_1(i)}{x_2(i) - x_1(i)}$$

$$\mathrm{intercept}(i) = y_1(i) - x_1(i) \times \mathrm{slope}(i)$$

$$\alpha(i) = \tan^{-1}(\mathrm{slope}(i)).$$

The aim of this type of matching is to determine whether the line segments from a pair of palmprint images were taken from the same palm. For example, two line segments from two images can be represented as $(x_1(i), y_1(i)), (x_2(i), y_2(i))$ and $(x_1(j), y_1(j)), (x_2(j), y_2(j))$, with the Euclidean distances, that is, the straight line distance between two points, between the end points represented as:

$$\nabla_1 = \sqrt{((x_1(i) - x_1(j))^2 + ((y_1(i) - y_1(j))^2}$$

$$\nabla_2 = \sqrt{((x_2(i) - x_2(j))^2 + ((y_2(i) - y_2(j))^2}$$

Based on these definitions, Zhang and Shu (1999) proposed a number of conditions for line segment matching:

1. If both ∇_1 and ∇_2 are less than a given threshold, D, then it indicates that the two line segments are the same.
2. If the difference in the angle of inclination between the two line segments is less than a given threshold, β, and that of the intercepts is also less than a given threshold, B, then it shows that the two line segments have an equal angle of inclination and intercept. Within those equal segments, if ∇_1 and ∇_2 are less than D then the two line segments are considered to be from the same line.
3. When two line segments overlap, they are regarded as the same line segment if the midpoint of one line segment is between the two end points of the other line.

By applying the above rules to a pair of palmprint images, a corresponding pair of lines can be identified. Furthermore, a verification function, r, can be defined as:

$$r = \frac{2N}{N_1 + N_2}$$

where N is the number of corresponding pairs, and N_1 and N_2 are the number of line segments determined from each image. In principle, two images are from the same palm if r is more than a given threshold, T, between 0 and 1 (Zhang and Shu, 1999).

To test their method, Zhang and Shu (1999) used 200 palmprint images from 20 palms. The false rejection rate (FRR), that is, the number of palms that were incorrectly identified as non-matching, and false acceptance rate (FAR), that is, the number of palms that were incorrectly identified as matching, were used to determine the experimental results. Figure 19 shows the results for offline palmprint identification using Zhang and Shu's (1999) feature line extraction method at various threshold, T, levels, where $D = 5$, $\beta = 0.157$, and $B = 10$. An ideal result, that is, where the FAR and FRR are minimised, can be obtained while T is between 0.1 and 0.12.

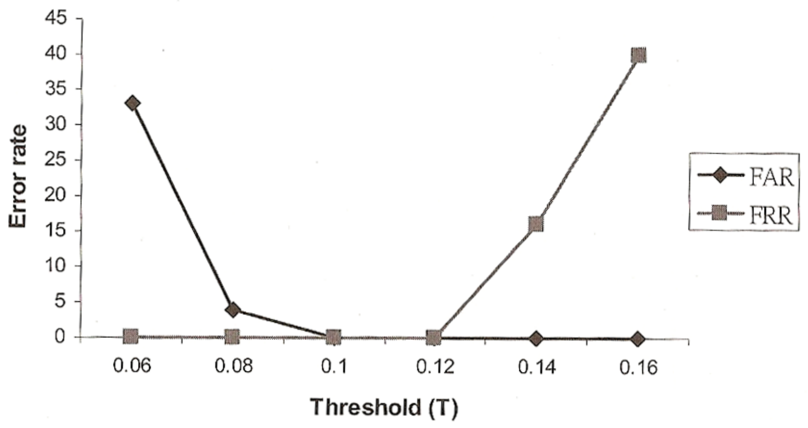


Figure 19. Experimental results for Zhang and Shu’s (1999) feature line matching method (Zhang, 2004).

Flexion creases and wrinkles in an online palmprint image can be described as a type of roof edge (Wu *et al.*, 2002b), that is, a region in an image where the intensity steadily increases and then, after a certain point, steadily decreases (Haralick, 1984). Furthermore, roof edges can be defined as a discontinuity in the first-order derivative of a grey-level profile, and the magnitude of the edge points’ second-derivative can reflect the strength of the edge points (Wu *et al.*, 2002b). Wu *et al.* (2002b) successfully used these properties to detected palmar flexion creases and wrinkles in online palmprint images.

To improve the connectivity and smoothness of the palmar lines, the image, $I(x,y)$, is first smoothed horizontally using a 1D Gaussian function, G_{σ_s}, with variance, σ_s:

$$I_s = I * G_{\sigma_s},$$

where $*$ is the convolve operation. The first- and second-order vertical derivatives can then be computed by convolving the smoothed image with the first-, G'_{σ_d}, and second-order, G''_{σ_d}, derivative of a 1D Gaussian function, G_{σ_d}, with variance, σ_d:

$$I' = I * H_1^0 \text{ and } I'' = I * H_2^0,$$

where $H_1^0 = G_{\sigma_s} * (G'_{\sigma_d})^T$, $H_2^0 = G_{\sigma_s} * (G''_{\sigma_d})^T$, T is the transpose operation, and $*$ is the convolve operation (Wu *et al.*, 2002b).

The horizontal lines and their strengths can be determined by looking for the zero-cross points of I' in the vertical direction, and then at their corresponding points in I'':

$$E_H(x,y) = \begin{cases} I''(x,y), \text{ if } I'(x,y) = 0 \text{ or } I'(x,y) \times I'(x+1,y) < 0 \\ 0 \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}.$$

Furthermore, the type of roof edge, that is, valley or peak, can be determined from the sign of the values in $E_H(x,y)$. A positive value represents a valley, while a negative value represents a peak. Since all palmar lines can be considered valleys, the negative values in $E_H(x,y)$ can be discarded:

$$E_H(x,y) = \begin{cases} L_0^1(x,y), \text{ if } L_0^1(x,y) > 0 \\ 0 \qquad\quad \text{otherwise} \end{cases}.$$

Flexion creases and strong wrinkles are much thicker than small wrinkles and ridges. Therefore, one or more thresholds can be used to obtain a binary image, called the binary edge image, of only the flexion creases and strong wrinkles. Furthermore, the directional line detectors, H_1^0 and H_2^0, can be obtained for any direction, θ, by rotating H_1^0 and H_2^0 with the appropriate θ angle. Finally, each binary edge image can be combined using a logical OR operation to obtain a new image containing line edges from all extracted directions. After morphological thinning the combined binary edge image, that is, reducing the lines to be minimally connected, a palmar line image is obtained (Wu *et al.*, 2002b). Figure 20 shows the results of Wu *et al.*'s (2002b) line extraction algorithm.

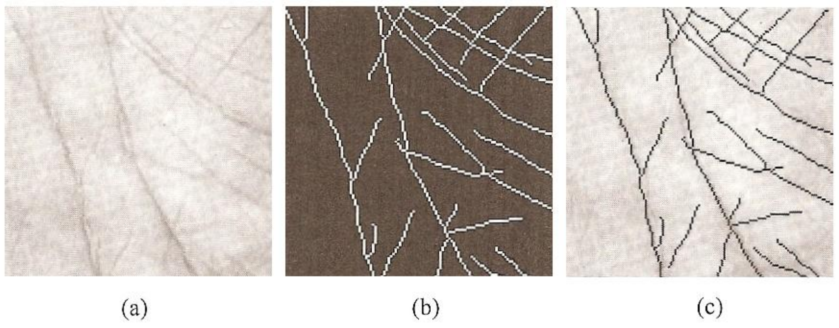


Figure 20. The results of palmar line extraction: a) the original image, b) the resultant palmar line image, and c) the original image with the extracted palmar lines overlaid (Wu *et al.*, 2002b).

The two parameters, σ_s and σ_d, in the directional line detectors control the performance of the detection algorithm. σ_s controls the connectivity and smoothness of the lines, while σ_d controls the width of the lines that can be detected. A small σ_s value results in poor connectivity and smoothness, while a large σ_s value results in the loss of some short or curved line segments. Furthermore, thin edges cannot be extracted when the σ_d value is large. For palmar lines, σ_s should be large and σ_d should be small (Wu *et al.*, 2002b). During experimentation, Wu *et al*. (2002b) used 1.0 for σ_s and σ_d. However, Zhang (2004) determined that σ_s and σ_d should be 1.8 and 0.5 for effective palmar line extraction.

Palmar lines extracted using Wu *et al.*'s (2002b) method are difficult to represent in a mathematical form, as they are represented by irregular patterns. However, an efficient method for storing irregular lines is chain code. A direction code is defined, according to its neighbour, for each point on a line, which is then stored in a list called a chain code. Each line can be recreated by the coordinates of its beginning point, and its chain code. This representation allows palmar lines to be easily, and accurately, stored (Wu *et al.*, 2002b).

Let S denote a palmar line image, and T denote a palmar line image restored from template represented by chain codes. To decrease computation complexity, the partial directed Hausdorff distance, $h_k(S,T)$, from S to T is compared. If the distance between a point in S and a point in T are below a given maximum, δ, they are regarded as matching. However, due to noise and other external factors, the line points may not be placed in the same position on a pair of palmprint images captured from the same palm but at different times. To overcome this problem, Wu *et al*. (2002b) dilated T with a δ-sized structuring element, and identified the overlapping points using a logical AND operation, to give a new palmar line image, K. However, due to imperfect pre-processing, rotation and translation may be present between samples of the same palmprint image. To minimise the effects of rotation and translation, a transformation, t, for S which maximises K is calculated, giving $K_{\max}$. A matching score between S and T can then be given by:

$$\text{Score} = \frac{2 \times K_{\max}}{m_1 + m_2},$$

where m_1 and m_2 are the number of edge points in S and T (Wu *et al.*, 2002b).

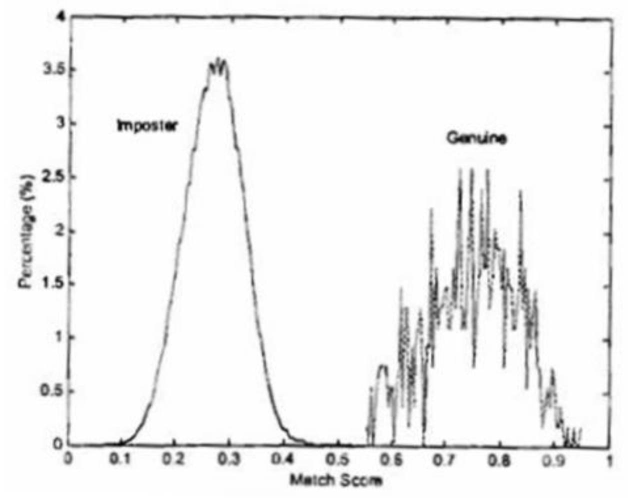


Figure 21. The distribution of correct and incorrect matching scores (Wu *et al.*, 2002b).

To test the performance of their palmar line extraction and matching algorithms, Wu *et al.* (2002b) took each image from a database of 675 palmprint images from 135 right hand palms, and compared them to one another. Figure 21 shows the distribution of correct and incorrect comparisons, in which there are two distinct peaks. The distribution curve of the correct matching scores intersects very little with that of the incorrect matching scores, demonstrating how effectively the system distinguishes palmprints from different people (Wu *et al.*, 2002b).

Directional element features (DEF) are a statistical measure for calculating the structure information of line-based characters, and are one of the most efficient methods of offline Chinese character recognition (Wu *et al.*, 2002a). To identify the major flexion creases in online palmprint images, Wu *et al.* (2002a) developed a modification of DEF, named fuzzy directional element energy features (FDEEF), of which three key stages can be defined: line edge detection, fuzzy dot orientation, and vector construction.

To obtain a line edge map of the major flexion creases, the FDEEF algorithm uses a Canny edge detection algorithm to generate an edge magnitude image, Mag, and a gradient angle image, $Angle$, where $-90° \leq Angle(i, j) < 90°$. For fuzzy dot orientation, in which each contour pixel is assigned a direction, let U be a collection of all edge points in a palmprint, and define four fuzzy line element sets, $F_{0°}$, $F_{45°}$, $F_{90°}$, and $F_{135°}$, in U:

$$F_{A°} = \{\mu_{A°}(i, j)/(i, j)\},$$

where (i, j) is the coordinates of an edge point, A is the fuzzy line element angle and $\mu_{A^\circ}(i, j)/(i, j)$ is the membership function:

$$\mu_{0^\circ}(i, j) = \begin{cases} \cos(2 * Ang(i, j)), 0^\circ \le Ang(i, j) < 45^\circ \text{ and } 135^\circ \le Ang(i, j) < 180^\circ \\ 0, \quad 45^\circ \le Ang(i, j) < 135^\circ \end{cases},$$

$$\mu_{45^\circ}(i, j) = \begin{cases} \sin(2 * Ang(i, j)), & 0^\circ \le Ang(i, j) < 90^\circ \\ 0, & 90^\circ \le Ang(i, j) < 180^\circ \end{cases},$$

$$\mu_{90^\circ}(i, j) = \begin{cases} 0, & 0^\circ \le Ang(i, j) < 45^\circ \text{ and } 135^\circ \le Ang(i, j) < 180^\circ \\ -\cos(2 * Ang(i, j)), & 45^\circ \le Ang(i, j) < 135^\circ \end{cases},$$

$$\mu_{135^\circ}(i, j) = \begin{cases} 0, & 0^\circ \le Ang(i, j) < 90^\circ \\ -\sin(2 * Ang(i, j)), 90^\circ \le Ang(i, j) < 180^\circ \end{cases},$$

where $Ang(i, j) = Angle(i, j) + 90^\circ$, which is the angle of the line element containing the point (i, j). Using these membership functions, two properties can be defined:

1. For each point, at least two of its membership grades are zero.
2. Given an angle, α, $45^\circ \le \alpha > 90^\circ$, at point (i, j), when α varies from 45° to 90°, μ_{45° decreases from 1 to 0, while μ_{90° increases from 0 to 1.

To construct the FDEEF vector, the palmprint edge image is divided into equal $M \times M$ blocks, and labelled $1, \ldots, M \times M$. For the block labelled p, its fuzzy directional element energies can be defined as:

$$E^p_{A^\circ} = \sum_{i=1}^{m} (Mag(x_i, y_i) \times \mu_{A^\circ}(x_i, y_i))^2,$$

where A is the fuzzy line element angle, m is the total number of points in the block, and $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ are the coordinates of these points. Four fuzzy directional element energies in each block can form a four-dimensional vector, $E^p_{0^\circ}$, $E^p_{45^\circ}$, $E^p_{90^\circ}$, and $E^p_{135^\circ}$. Therefore, a $M \times M \times 4$-dimensional feature vector can be obtained from the complete image:

$$V = (E^1_{0^\circ}, E^1_{45^\circ}, E^1_{90^\circ}, E^1_{135^\circ}, \ldots, E^{M\times M}_{0^\circ}, E^{M\times M}_{45^\circ}, E^{M\times M}_{90^\circ}, E^{M\times M}_{135^\circ}).$$

Finally, to remove the effects of illumination variance, the vector can be normalised by the total energy:

$$\tilde{V} = (e^1_{0^\circ}, e^1_{45^\circ}, e^1_{90^\circ}, e^1_{135^\circ}, \ldots, e^{M\times M}_{0^\circ}, e^{M\times M}_{45^\circ}, e^{M\times M}_{90^\circ}, e^{M\times M}_{135^\circ}),$$

$$e^k_A = E^k_A \times \frac{1000}{\sum_{i=1}^{M\times M}(E^i_{0^\circ} + E^i_{45^\circ} + E^i_{90^\circ} + E^i_{135^\circ})},$$

where $k = 1, \ldots, M \times M$, and $A = 0^\circ, 45^\circ, 90^\circ, 135^\circ$. The normalised vector, $\tilde{V}$, is the fuzzy directional element energy feature (Wu *et al.*, 2002a). Figure 22 shows an example FDEEF extraction process.

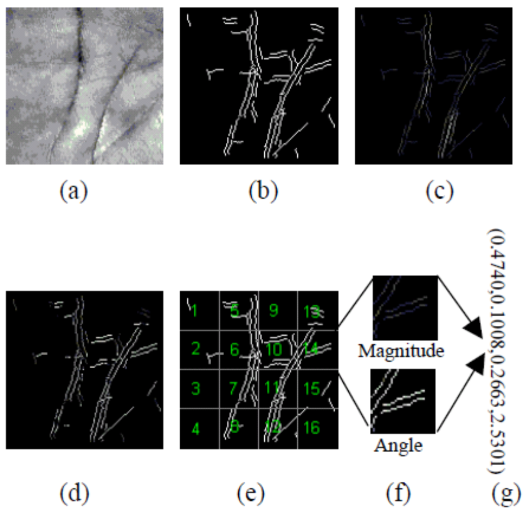


Figure 22. An example FDEEF extraction: a) an original palmprint image and its b) binary edge image, c) edge magnitude image, d) edge angle image, e) blocked edge image, f) partial magnitude and edge map, and g) partial feature vector (Wu *et al.*, 2002a).

To test the performance of their FDEEF-based flexion crease identification algorithm, Wu *et al.* (2002a) compared the feature vectors of 450 palmprint images, measuring 128×128 pixels, from 50 different right hand palms. From each palm, 9 images were captured, of which 4 were used as training templates, and 5 were used as comparison tests. The training templates were created by taking the mean vector value from each training sample set. Each remaining palmprint was then compared, using the Euclidean distance, against the training templates. The template vector nearest to the test vector was considered the identification result. The genuine acceptance rate (GAR), that is, the number of palms that were correctly identified as matching, was 97.2% (Wu *et al.*, 2002a).

Similarly, Wu *et al.* (2004a) used morphological operations to explicitly extract palmar flexion creases and wrinkles. Mathematical morphology is a set-based method of image analysis for providing a quantitative description of geometrical structures, that is often used to extract components in binary or greyscale images.

In greyscale morphology, two basic morphological operations, dilation and erosion, can be defined as:

$$(f \oplus b)(s,t) = \max\{f(s-x,t-y)+b(x,y) \mid (s-x,t-y)\in D_f \text{ and } (x,y)\in D_b\},$$

$$(f \Theta b)(s,t) = \min\{f(s-t,t-y)-b(x,y) \mid (s-x,t-y)\in D_f \text{ and } (x,y)\in D_b\},$$

where $\oplus$ is the erosion operation, Θ is the dilation operation, and D_f and D_b represent the domains of the image, f, and structuring element, b. Furthermore, two additional operations, opening and closing, can be defined by combining the dilation and erosion operations:

$$f \circ b = (f \Theta b) \oplus b,$$

$$f \bullet b = (f \oplus b) \Theta b.$$

Finally, using the closing operation, the bottom hat operation, that is, the image minus the closing of the image, which can be used to detect valleys in an image, can be defined:

$$h = (f \bullet b) - f.$$

Morphological operations apply a structuring element to an input image, and create an output image of the same size, in which the value of each pixel is based on a comparison of the corresponding pixel in the input image with its neighbours. To this end, the shape of the structuring element heavily influences the results of the operation. Since the direction of the palmar lines is irregular, a number of directional structuring elements are required. The directional structuring element, b_{0°, which is used to extract 0° palmar lines, is shown in Figure 23(a). Furthermore, the directional structuring element, b_{θ°, which is used to extract θ° palmar lines, can be obtained by rotating b_{0° by θ°, as shown in Figure 23.

$$\begin{array}{c} 1\\ 1\\ 1\\ 1\\ 1 \end{array} \qquad \begin{array}{ccccc} 1 & 1 & 1 & 1 & 1 \end{array} \qquad \begin{array}{ccccc} 1 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 1 \end{array} \qquad \begin{array}{ccccc} 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 1 & 0 & 0\\ 0 & 1 & 0 & 0 & 0\\ 1 & 0 & 0 & 0 & 0 \end{array}$$

(a) b_{0° (b) b_{90° (c) b_{45° (d) b_{135°

Figure 23. Wu *et al.*'s (2002a) directional structuring elements.

Using these directional structuring elements, Wu *et al*. (2004a) defined the following process to extract $\theta°$ palmar lines:

1. Smooth the original image, I, by convolution with $b_{\theta+90°}$.
2. Process the smoothed image using a bohat operation, with a structuring element, $b_{\theta°}$, to get the $\theta°$-directional magnitude image, $M_{\theta°}$.
3. Find the local maximum along $\theta + 90°$ in $M_{\theta°}$.
4. Apply a threshold to the maximum magnitude image.

However, while the majority of palmar lines can be extracted using this method, some weak line segments fail to be identified. To this end, Wu *et al*. (2004a) defined a fine-level feature extraction method to predict the position and direction of the missing line segments. Figure 24 shows the process of fine-level palmar line extraction.

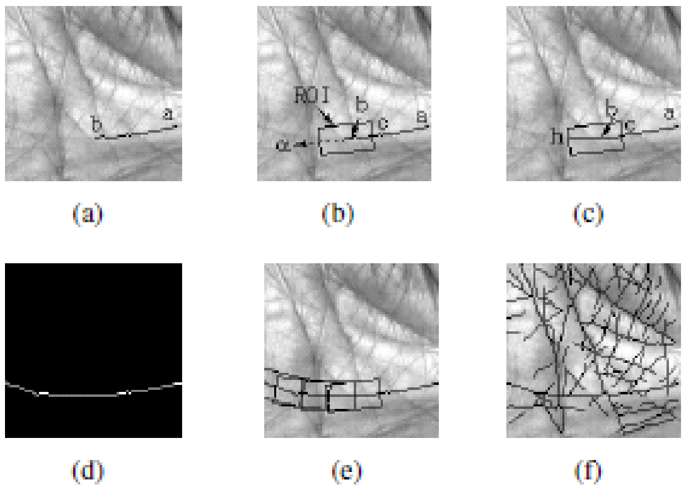


Figure 24. Wu *et al.*'s (2002a) fine-level palmar line extraction process: a) an extracted line, and b) its predicted region of interest and direction, c) the next line segment, d) the entire line, e) all regions of interest, and f) the extracted palmar lines.

Let ab be an extracted segment from line A. To extract the next line segment, that is, the segment connected to b, the line is traced from a to b to get the Kth point, c. Joining points b and c gives a straight line cb, of which the slope angle is α. Since palmar lines do not curve greatly, a region of interest, in which the next line segment is predicted to be located, can be defined as an $L \times W$ rectangular region, with midpoint b. In this region of interest, directional structuring elements, as described above, can be used to identify $\alpha°$ line branches. If only one of these line branches connects with cb, it is regarded as the next line segment, bh. Otherwise, the branch that is smoothest at point b is chosen (Wu *et al.*, 2004a).

After the next line segment has been obtained, the line can be completed. If the line, ch, satisfies one of the following conditions, the line is considered to have reached it end point:

1. If ch has reached the border of the image, then h is the end point.
2. If the minimum distance from the end point, h, to three sides of the region of interest exceeds a given threshold, T_d, then h is the end point.
3. If the angle cmh is less than a given threshold, T_α, having joined points c and h, given that point m is the farthest point to ch, and having joined cm and hm, then m is the end point.

If none of these conditions are satisfied by ch, the process is repeated using ah until the line is considered complete (Wu *et al.*, 2004a).

A hierarchical identification system, such as that proposed by Wu *et al*. (2004a), allows palmar lines of any strength to be extracted. To this end, Li and Leung (2006) used coarse- to fine-level hierarchical matching in their line-based Hough transform identification system. In the coarse-level stage, a Sobel line detector, morphological thinning, contour extraction, and polygon approximation are applied to a given online palmprint image to extract a line edge map (Goa and Leung, 2002). The line edge map, which is a series of approximated straight line segments, is then transformed using a line-based Hough transform into a two-dimensional feature vector, $L(p, \theta)$, where all the line segments are converted to points in parametric space. By converting the line segments into parametric space, the location and orientation of the flexion creases and wrinkles can be easily described by the feature vector:

$$L = \{L_{p_{\min}\theta_{\min}}, \ldots, L_{p\theta}, \ldots, L_{p_{\max}\theta_{\max}}\}.$$

Furthermore, to reduce the number of cells in L, the $p-\theta$ space is evenly divided into 16×9 bins. The feature vector, L, is therefore simplified to 16×9 values, which represent the complete palmar line pattern (Li and Leung, 2006).

To extract the major flexion creases from the simplified feature vector, a line parameter deviation filter was proposed. As there are only slight deviations in the line segment parameters for each approximated curve, line segments with similar parameters can be identified and grouped. Using this method, the three longest curves, and therefore, the major flexion creases, can be dynamically extracted:

```
Find 3 peak cells in p-θ as seeds
For 3 seeds $P_i$
   Sort other cells, $C_j$, according to Distance($P_i$,$C_j$)
For 3 links
   Merge into one with ascend Distance($P_i$,$C_j$)
Keep first half of the combined link for seed growing
For 3 seeds
   Merge nearest neighbours of current seed
```

Each major flexion crease can be identified by L, p, and θ in the feature vector:

$$Major = \{(L_1, p_1, \theta_1), (L_2, p_2, \theta_2), (L_3, p_3, \theta_3)\},$$

which represents the central location and accumulated length of the three peak clusters (Li and Leung, 2006).

Given the global and major flexion crease feature vectors of two palmprint images, the distance between them can be defined as:

$$D(m,t) = dist(global) + dist(major),$$

where $dist(global)$ and $dist(major)$ are defined as:

$$dist(global) = \sum_{p\in[0,9],\theta\in[0,16]} (L_{m_{p\theta}} - L_{t_{p\theta}})^2, \text{ and}$$

$$dist(principal) = \sum_{i\in[0,2]} L_{m_i}((p_{m_i} - p_{t_i})^2 + (\theta_{m_i} - \theta_{t_i})^2).$$

Using these definitions, Li and Leung (2006) processed 600 palmprint images from 100 different palms to achieve 71% accuracy for the top match, and 100% accuracy for the top 15 matches. However, to improve these results, a fine-level matching stage was introduced. Based on the work of Goa and Leung (2002), the distance algorithm was improved to compare line segments rather than line points:

$$d_\theta(m,t) = \left(\frac{width}{d_{displace}(m,t)}\right) \times \tan(\theta(m,t)),$$

where $width$ is the width of the image, m is a line segment from a template set, t is a line segment from a test set, $\theta(m,t)$ is the smallest intersecting angle between m and t, and $d_{displace}(m,t)$ is the position distance of the two lines:

$$d_{displace}(m,t) = \sqrt{d_\perp(m,t)^2 + d_\parallel(m,t)^2},$$

where $d_\perp(m,t)$ is the perpendicular distance between two line segments, and $d_\parallel(m,t)$ is the minimum displacement to align either the left or right end points of m and t (Li and Leung, 2006).

To test the performance of their improved distance measure, Li and Leung (2006) again compared 600 images from 100 different palms, that is, 6 images from each palm. For each palm, 3 images were randomly chosen as template images, while the remaining 3 were used as test images. A hierarchical matching scheme was used, that is, coarse-level matching and then fine-level matching, to achieve an identification rate of 99%.

Another approach to palmar flexion crease identification, as proposed by Huang *et al.* (2008) and Jia *et al.* (2008), is to use the Radon transform. The Radon transform of a two-dimensional function $f(x.y)$ is defined as:

$$R(r,\theta)[f(x,y)] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\delta(r - x\cos\theta - y\sin\theta)dxdy,$$

where r is the perpendicular distance of a line from the origin, and θ is the angle between the line and the vertical axis. The Radon transform accentuates linear features, and therefore, can be used to detect linear trends in an image. However, the Radon transform cannot effectively detect line segments which are significantly shorter than the image dimensions. To this end, the finite Radon transform (FRAT) was proposed (Matus and Flusser, 1993) as an effective method for performing the Radon transform on finite length signals. Denoting $\{Z_p = 0,1,\ldots,p-1\}$, where p is a prime number, the FRAT of a real function, $f[x,y]$, on the finite grid Z_p^2 is defined as:

$$r_k[l] = FRAT_f(k,l) = \frac{1}{\sqrt{p}} \sum_{(i,j)\in L_{k,l}} f[i,j],$$

where $L_{k,l}$ is the set of points in a line on the finite grid, Z_p^2. Therefore,

$$L_{k,l} = \{(i,j): j = ki + l(\text{mod}, p), i \in Z_p\}, 0 \le k < p, \text{ and}$$

$$L_{p,l} = \{(l,j): j = \in Z_p\},$$

where k is the corresponding slope of the line, and l represents the intercept (Huang *et al.*, 2008).

Palmar flexion creases and wrinkles in an online palmprint image can be regarded as straight line segments in a small, localised area. Therefore, palmar flexion creases and wrinkles can be detected by FRAT. However, as FRAT treats the input as a periodic image, the detected lines exhibit a wrap around effect. To this end, Huang *et al*. (2008) and Jia *et al.* (2008) proposed a modified finite Radon transform (MFRAT) algorithm to detect and extract palmar line features. Again, denoting $\{Z_p = 0,1,\ldots,p-1\}$, where p is a positive integer, the MFRAT of a real function, $f[x,y]$, on the finite grid Z_p^2 is defined as:

$$r[L_k] = MFRAT_f(k) = \frac{1}{C} \sum_{(i,j)\in L_k} f[i,j],$$

where C controls the scale of $r[L_k]$, and L_k is the set of points in a line on the finite grid, Z_p^2:

$$L_k = \{(i,j): j = k(i-i_0) + j_0, i \in Z_p\},$$

where (i_0, j_0) is the centre point of Z_p^2, and k is the corresponding slope of L_k. Furthermore, the direction, θ_k, and the energy, e, of the centre point, $f(i_0, j_0)$, of Z_p^2 can be calculated as:

$$\theta_{k(i_0,j_0)} = \arg(\min{}_k(r[L_k])),\ k = 1,2,\ldots,N \text{, and}$$

$$e_{(i_0,j_0)} = \left|\min(r[L_k])\right|,\ k = 1,2,\ldots,N,$$

where | | denotes the absolute operation. Using these definitions, the direction and energy of the entire image can be calculated (Huang *et al.*, 2008; Jia *et al.*, 2008). In addition, before palmprint matching, Huang *et al.* (2008) obtained a binary image, $Lines$, using a threshold, T, from the energy image, e. The binary image, $Lines$, was then be divided in to LA and LB according the direction, $\theta_{(x,y)}$, of each pixel, to separate the flexion creases from the wrinkles:

$$LA(x,y) = \begin{cases} 1, \text{ if } Lines(x,y) = 1, \text{ and } 0° < \theta_{(x,y)} \le \pi/2 \\ 0, \qquad\qquad \text{otherwise} \end{cases},$$

$$LB(x,y) = \begin{cases} 1, \text{ if } Lines(x,y) = 1, \text{ and } \pi/2 \le \theta_{(x,y)} \le \pi \\ 0, \qquad\qquad \text{otherwise} \end{cases}.$$

To determine the location of the flexion creases, LA or LB, two Radon transform energy maps can be created, $R[LA(x,y)]$ and $R[LB(x,y)]$, in which the energy of the flexion creases will be greater than that of the wrinkles. For a binary image, $F(x,y)$, the Radon transform energy of $R[F(x,y)]$ can be calculated as:

$$E_{total}(F(x,y)) = \sum_{x=1}^{m}\sum_{y=1}^{n}(R[F(x,y)]),$$

where $E_{total}(F(x,y))$ is the total sum of all of the values of $R[F(x,y)]$. By comparing the energies of LA and LB, the location of the major flexion creases can be determined (Huang *et al.*, 2008).

To calculate the degree of similarity between two palmprint images, Huang *et al.* (2008) and Jia *et al.* (2008) used a matching algorithm based on pixel to area comparison. In contrast with pixel to pixel matching, as is used in a number of palmprint identification methods (Zhang *et al.*, 2003; Kong and Zhang, 2004; Kong *et al.*, 2006a), pixel to area comparison is more robust to changes in rotation and translation. In many cases, pixel to pixel comparison does not perform well, as it is often difficult to obtain perfect superposition of two palmprint images captured at different times but from the same palm (Jia *et al.*, 2008). Huang *et al.* (2008) compared flexion crease binary images, LA or LB, as determined above, while Jia *et al.* (2008) compared Radon energy images, e. Suppose A is a test image and B is a template image, and the size of A and B is $m \times n$. The matching score from A to B can be defined as:

$$s(A,B) = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} A(i,j) \cap \overline{B}(i,j)\right) \Big/ N_A ,$$

where $\cap$ is the logical AND operation, N_A is the number of comparison points in A, and $\overline{B}(i,j)$ is a small area around $B(i,j)$. Similarly, the matching score from B to A can be defined as:

$$s(B,A) = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} B(i,j) \cap \overline{A}(i,j)\right) \Big/ N_A .$$

Finally, the final matching score between A and B can be defined as:

$$s(A,B) = s(B,A) = \max(s(A,B), s(B,A)),$$

where $s(A,B)$ is between 0 and 1. The larger the matching score, the greater the match between A and B (Huang *et al.*, 2008; Jia *et al.*, 2008).

To test their identification methods, Huang *et al.* (2008) and Jia *et al.* (2008) compared 7 752 online palmprint images, measuring 128×128 pixels at 75 dpi, from 386 different palms. Huang *et al.* (2008) created two databases, I and II, containing palmprint images from 100 palms in database I, and 386 palms in database II. In database I, an equal error rate (EER), that is, the rate at which the FAR and FRR are equal, of 0.49% was achieved, while in database II, an EER of 0.565% was achieved. Similarly, Jia *et al.* (2008), using palmprint images from all 386 palms, performed a number of experiments, and achieved a GAR of 98.37%, an FAR of 4×10^{-5} %, and an EER of 0.16%.

Table 3 shows a summary of palmprint feature line identification methods. Of the results presented, Huang *et al.* (2008) recorded the most effective palmar line identification method, with a GAR of 99.43%. However, Jia *et al.* (2008), using a similar method, that is, based on a modified Radon transform, recorded the most secure, with an FAR of 4×10^{-5} %.

Table 3. A summary of palmprint feature line identification methods.

Algorithm	Type	Sample Size		Image Size		Recognition Rate
		Images	Palms	Dimensions (pixels)	Resolution (dpi)	
Kung *et al.* (1995)	Online	96	32	[a]	[a]	99% GAR
Rodrigues and Silva (1996)	Online	15	5	[a]	[a]	[a]
Boles and Chu (1997)	Online	10	3	[a]	[a]	77% GAR
Zhang and Shu (1999)	Offline	200	20	400×400	100	[a]
Wu *et al.* (2002a)	Online	450	50	128x128	[a]	97.2% GAR
Wu *et al.* (2002b)	Online	675	135	128×128	[a]	99.5% GAR
Wu *et al.* (2004a)	Online	Several thousand		128×128	[a]	[a]
Li and Leung (2006)	Online	600	100	160×160	[a]	99% GAR
Huang *et al.* (2008)	Online	7 752	386	384×284	75	99.43% GAR 0.565% FAR
Jia *et al.* (2008)	Online	7 752	386	384×284	75	98.37% GAR 4×10^{-5}% FAR

[a] Data not available.

2.4. Computer vision and pattern recognition methods

In addition to the methods described above, a number of computer vision, pattern recognition, and image processing algorithms can be used to identify palmar flexion creases. A typical palmar flexion crease identification system contains an edge detection algorithm, which identifies the palmar flexion creases, morphological processing, which enhances and segments the palmar flexion crease line edge map, and a line matching algorithm, which is used to compare two or more palmprint images (Zhang, 2004).

As stated in Wu *et al.* (2002b), and detailed above, palmar flexion creases can be described as a kind of roof edge. In an intensity image, that is, a greyscale image, a roof edge occurs where a discontinuity in intensity causes a crease to form on the intersection of two planes, and a local maximum is observed (Chen and Don, 1992). Figure 25 shows the gradient profiles of a number of 2-dimensional roof edge intensity instances.

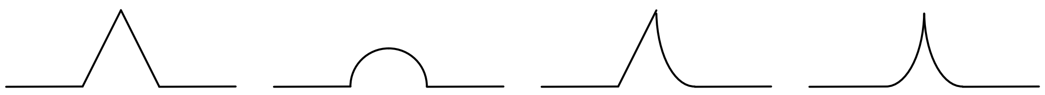


Figure 25. An example of 2-dimensional roof edge types (Chen and Don, 1992).

Furthermore, a roof edge can also be considered an intersection of two step (or jump) edges. A step edge can be defined as “the boundary between two regions whose brightness values are significantly different, and usually correspond to occluding boundaries of objects in a scene” (Chen and Don, 1992, p.2). To this end, with a sufficiently defined neighbourhood, that is, a sub-region of the image, any step edge detector can be used to detect roof edges, and therefore, palmar flexion creases (Chen and Don, 1992).

There are two generic approaches to detecting edge in images: differential detection, where spatial processing is performed to produce a differential image with accentuated spatial amplitude, and model fitting, where a local region of pixel values are fit to a model of the edge, line, or spot to be detected (Pratt, 2007). Since the introduction of image processing, many methods have been proposed for detecting edges in images. Most edge detection algorithms differ only in their smoothing filters, differentiation operators, labelling processes, goals, computational complexity, and the mathematical models used to derive them (Ziou and Tabbone, 1998). For this reason, an exhaustive review of edge detection will not be presented here. Instead, a comprehensive discussion of edge detection may be found in Davis (1975), Torre and Poggio (1986), Nawal (1993), Zamperoni (1995), and Ziou and Tabbone (1998).

In palmar flexion crease identification, the Sobel (Sobel and Feldman in Duda and Hart, 1973, p.271) and Canny (Canny, 1986) edge detectors are popular (Rodrigues and Silva, 1996; Boles and Chu, 1997; Wu *et al.*, 2002a; Li and Leung, 2006). The Sobel operator is a type of first-order derivative edge gradient measure (a name by which a number of other measures can be described), which emphasises regions of high spatial frequency. As described in Pratt (2007), given an image, the gradient magnitude, ∇, and the edge direction, α, can be calculated at each point, (x, y), in the image as:

$$\nabla(x,y) = \sqrt{Gx^2 + Gy^2} \text{ and } \alpha(x,y) = \arctan(Gx/Gy),$$

where Gx and Gy are difference operators, and a $0°$ edge direction refers to the direction of maximum contrast from black to white from left to right. Alternatively, for computational efficiency, an approach used frequently is to approximate the gradient using absolute values:

$$\nabla(x, y) = |G_x| + |G_y|.$$

Given the gradient magnitude and edge direction equations, the difference operator for Sobel's (Sobel and Feldman in Duda and Hart, 1973, p.271) algorithm consists of two 3×3 kernels, as shown in Figure 26, which, when convolved with an input image, respond maximally to horizontal and vertical edges (Duda and Hart, 1973).

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

Gx

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

Gy

Figure 26. Sobel difference operators (Duda and Hart, 1973).

As stated earlier, a number of edge detectors can be described as first-order derivative edge gradient measures. The Sobel (Sobel and Feldman in Duda and Hart, 1973, p.271), Prewitt (Prewitt and Mendelsohn, 1966), Roberts (Roberts, 1965), Frei-Chen (Frei and Chen, 1977), pixel difference, and separated pixel difference operators differ only in the type of impulse response kernels they employ. A comparison of first-order derivative difference operators can be found in Pratt (2007, p.478).

In addition, second-order derivative operators, which are similar to first-order derivative operators, are also used to detect edges in images. For a given step edge, first-order derivative operators generally give a stronger response, and therefore, produce thick edges, while second-order derivative operators give a weaker response, and therefore, produce thin edges. Furthermore, second-order derivative operators are much more aggressive in enhancing sharp changes in intensity. Therefore, a second-order derivative operator is much more likely to enhance fine detail, such as isolated points and thin lines. However, as expected, a stronger response to fine detail allows second-order derivative operators to be more sensitive to noise. For this reason, second-order derivative operators are more suited to image enhancement, while first-order derivative operators are more suited to edge detection (Gonzalez and Woods, 2008). However, a second-order derivative operator that is often used for edge detection is the Laplacian. The Laplacian of a 2-dimensional function, $f(x,y)$, in the continuous domain is defined as:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}.$$

The Laplacian is zero if $f(x,y)$ is constant or changing linearly in amplitude. If the rate of change of $f(x,y)$ is greater than linear, the Laplacian exhibits a sign change at the point of inflection. The zero crossing, that is, the point at which the sign changes, indicates the presence of an edge (Pratt, 2007). For the Laplacian operator to be useful for digital image processing, the equation needs to be expressed in discrete form. A discrete approximation of a four-directional Laplacian, using a 3×3 neighbourhood, can be calculated from the difference of slopes along each axis:

$$\nabla^2 f = 4z_5 - (z_2 + z_4 + z_5 + z_8),$$

and, for an eight-directional Laplacian, that is, including the diagonal neighbours, by:

$$\nabla^2 f = 8z_5 - (z_1 + z_2 + z_3 + z_4 + z_5 + z_6 + z_7 + z_8 + z_9),$$

where Figure 27 shows the four- and eight-directional masks, and the location of each z in a 3×3 region of an image (Gonzalez and Woods, 2008).

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

(a)

0	-1	0
-1	4	-1
0	-1	0

(b)

-1	-1	-1
-1	8	-1
-1	8	-1

(c)

Figure 27. An a) 3x3 region of an image, and masks used to implement b) a four-directional Laplacian and c) an eight-directional Laplacian (Gonzalez and Woods, 2008).

As a second-order derivative operator, the Laplacian is very sensitive to noise, unable to detect edge direction, and often produces double edges, that is, where multiple edges are detected at the location of a single true edge (Gonzalez and Woods, 2008). As such, the Laplacian is generally not used in its original form. Instead, Marr and Hildrith (1980) proposed the Laplacian of Gaussian (LoG), where the Laplacian is combined with a smoothing function as a precursor to edge detection. Consider the function

$$h(r) = -e^{-\frac{r^2}{2\sigma^2}},$$

where $r^2 = x^2 + y^2$ and σ is the standard deviation. Convolving this function with a given image blurs the image, where σ determined the strength of the blur. Given this equation, the Laplacian of h, and consequently, the LoG, can be calculated as:

$$\nabla^2 h(r) = -\left[\frac{r^2 - \sigma^2}{\sigma^4}\right] e^{-\frac{r^2}{2\sigma^2}}.$$

The effect of the LoG is to reduce the noise on the image, and therefore, counter the increased noise caused by the second-derivative operator (Gonzalez and Woods, 2008).

Despite the popularity of gradient-based operators, many first- and second-order derivative difference operators often identify false or multiple edges, and do not provide adequate localisation. To this end, the Canny operator (Canny, 1986) is a modified first-order derivative operator based on three basic objectives: a low error rate, which states that all edges should be found as close as possible to the true edge; a localised response, which states that each edge point, in terms of distance, should correspond closely to the true edge, and a single edge point response, which states that the detector must not identify multiple edges where a single true edge exists (Canny, 1986).

First, the image is smoothed using a Gaussian filter, with a specified standard deviation, σ, to reduce noise, by weighting each pixel, (x, y), according to:

$$g(x, y) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{d^2}{2\sigma^2}},$$

where $d = \sqrt{(x - x_c)^2 + (y - y_c)^2}$ is the distance of the neighbourhood pixel, (x, y), from the centre pixel, (x_c, y_c), of the output image (Shapiro and Stockman, 2001). Second, given a smoothed image, the local gradient magnitude, ∇, and edge direction, α, are then calculated for each point, (x, y), as described earlier, as:

$$\nabla(x, y) = \sqrt{Gx^2 + Gy^2} \text{ and } \alpha(x, y) = \arctan(Gx/Gy),$$

where Gx and Gy are difference operators (Pratt, 2007), using two 3×3 kernels, typically Sobel and Prewitt. Third, an edge point is defined, using a process called non-maximal suppression, as a point where gradient magnitude is locally maximum in the direction of the gradient, and a threshold is applied, using two values, $T1$ and $T2$, where $T1 < T2$. Pixels with a value greater than $T2$ are said to be strong pixels, and pixels with a value between $T1$ and $T2$ are said to be weak pixels. Finally, edge linking is performed by connecting weak pixels that are 8-connected, that is, in a 3×3 neighbourhood of pixels, X, if the value of $X_0, X_1, \ldots, X_7$ is logical 1, to the strong pixels (Gonzalez and Woods, 2008).

While Canny's (1986) algorithm improved on earlier edge detection, such as those proposed by Sobel (Sobel and Feldman in Duda and Hart, 1973, p.271) and Prewitt (Prewitt and Mendelsohn, 1966), all gradient-based edge detection methods, including Canny, are sensitive to variations in image illumination, orientation, blurring, and magnification. For this reason, it can be difficult, if not impossible, to set appropriate thresholds in extended image sequences, where variations in illumination, orientation, and translation may be present (Kovesi, 2003). To address these problems, a model of feature perception using local energy, which has been used successfully in palmprint texture verification (Štruc and Pavešić, 2009a; Štruc and Pavešić, 2009b), was developed by Morrone *et al.* (1986) and Morrone and Owens (1987), which proposed that local features, such as step edges, lines, and roof edges, can be perceived at points in an image where Fourier components are maximally in phase (Kovesi, 1999). The local Fourier components at a location, x, in a one-dimensional signal will have an amplitude, $A_n(x)$, a phase angle, $\phi_n(x)$, and, when represented as complex vectors, adding head to tail, a magnitude, or local energy, $|E(x)|$. Using these definitions, a measure of phase congruency, as developed by Morrone *et al.* (1986), can be given as:

$$PC_1(x) = \frac{|E(x)|}{\sum_n A_n(x)},$$

where the phase congruency is the ratio of $|E(x)|$ to the overall path length of the local Fourier components in reaching the end point. If all the Fourier components are in phase, the complex vectors will be aligned, and the ratio will be 1. Conversely, If there is no coherence of phase, the ratio will fall to a minimum of 0 (Kovesi, 2003). Using these definitions, it can be shown that phase congruency is a function of the cosine of the deviation of each phase component from the mean:

$$PC_1(x) = \frac{\sum_n A_n(\cos(\phi(x) - \bar{\phi}(x))}{\sum_n A_n(x) + \varepsilon}.$$

However, this measurement of phase congruency is sensitive to noise and does not provide adequate localisation (Kovesi, 2003). To this end, Kovesi (1999) developed an improved phase congruency measure which produces a localised response and incorporates noise compensation:

$$PC_2(x) = \frac{\sum_n W(x) \lfloor A_n(x)(\cos(\phi_n(x) - \bar{\phi}(x)) - \left|\sin(\phi_n(x) - \bar{\phi}(x))\right|) - T \rfloor}{\sum_n A_n(x) + \varepsilon},$$

where $W(x)$ is a factor that weights frequency spread, ε is a small constant to avoid division by zero, and $\lfloor \ \rfloor$ denotes that the enclosed quantity is equal to itself when its value is positive, and otherwise, zero. Furthermore, to compensate for noise, only values that exceed T, which can be determined from the filter responses to the image, are included in the result (Kovesi, 2003).

Given the definition of phase congruency for one-dimensional signals, and to obtain an overall measurement of phase congruency in two dimensions, that is, to detect edges and corners in an image, local energy can calculated in several orientations using data from two-dimensional Gabor wavelets (Lee, 1996). Furthermore, to define how phase congruency varies with orientation, local energy can be calculated independently, and the variation of the moments at each orientation can be calculated. The principal axis, corresponding to the axis about which the moment is minimised, provides an indication of the orientation of the feature, while the magnitude of the maximum moment, corresponding to the moment about an axis perpendicular to the principal axis, provides an indication of the significance of the feature. (Kovesi, 2003). Given this theory, a, b, and c can be calculated at each point in an image:

$$a = \sum (PC(\theta)\cos(\theta))^2,$$

$$b = 2\sum (PC(\theta)\cos(\theta)) \cdot (PC(\theta)\sin(\theta)), \text{ and}$$

$$c = \sum (PC(\theta)\sin(\theta))^2 ,$$

where PC refers to the phase congruency value determined at orientation θ, and the sum is performed over a discrete set of orientations. The angle of the principal axis, Φ, can then be given by:

$$\Phi = \frac{1}{2}\operatorname{atan2}\left(\frac{\mathrm{b}}{\sqrt{\mathrm{b}^2 + (a-c)^2}}, \frac{a-c}{\sqrt{b^2 + (a-c)^2}} \right),$$

and the maximum, M, and minimum, m, moments as:

$$M = \frac{1}{2}(c + a + \sqrt{b^2 + (a-c)^2}), \text{ and } m = \frac{1}{2}(c + a - \sqrt{b^2 + (a-c)^2}).$$

Phase congruency, unlike gradient-based methods, provides a measure of edge strength that is invariant to variations in illumination and contrast, meaning a fixed threshold of feature significance can be applied to complete image sequences (Kovesi, 2002).

After edge detection, the palmprint line edge map, that is, an image containing only edge-like features, is typically processed using mathematical morphology to, for example, remove unwanted edge points, improve localisation, and link missing edge segments (Zhang, 2004). As described earlier, mathematical morphology is a set-based method of image analysis for providing a quantitative description of geometrical structures, that is often used to extract components in binary or greyscale images. Morphological operations apply a structuring element to an input image, and create an output image of the same size, in which the value of each pixel is based on a comparison of the corresponding pixel in the input image with its neighbours (Wu *et al.*, 2004a).

Two fundamental operations in binary morphology, as described earlier for greyscale morphology, are dilation and erosion, both of which can be defined and implemented by a hit-and-miss transform. A small odd-sized binary mask, typically 3×3, is passed over an image. If the pattern of the mask matches the state of the image under the mask, a hit occurs, and the image pixel corresponding to the spatial location of the centre of the mask is set to a given binary state. If the pattern of the mask does not match the state of the image under the mask, a miss occurs, and the image pixel corresponding to the spatial location of the centre of the mask is set to the opposite binary value (Pratt, 2007). Given this process, the shape of the structuring element heavily influences the results of the operation. In algebraic terms, Ritter and Wilson (2001) define the hit-and-miss transform of a set, A, by the expression:

$$A \otimes B = \{p : D_p \subset A \text{and } \mathrm{E_p} \subset A'\},$$

where $\otimes$ is the hit-and-miss operation, $B = (D, E)$ are a pair of structuring elements, and $D \cap E = \varnothing$. The dilation, that is, the expansion of the boundary of a region of foreground pixels, and the erosion, that is, the reduction of the boundary of a region of foreground pixels, of A by B, where $B = (D, E)$ and $E = \varnothing$, as:

$$A \times B = \{a + b : a \in A, b \in B\}$$

for the dilation, and:

$$A / B = (A' \times B^*)'$$

for the erosion, where B^* is the reflection of B, and A' is the complement of A.

Furthermore, in addition to erosion and dilation, a number of additive, subtractive, and conditional operations can be defined. Additive operations cause the centre pixel in a 3×3 region to be converted from a logical 0, for example, a white pixel, to a logical 1, for example, a black pixel, if the neighbouring pixels meet certain requirements. Conversely, subtractive operators cause the centre pixel in a 3×3 region to be converted from a logical 1 to a logical 0. Conditional operations, such as thinning and thickening, are erosion or dilation operations, which are controlled to prevent total erasure, while ensuring total connectivity (Pratt, 2007). Table 4 describes a number of important additive, subtractive, and conditional morphological operations.

Table 4. A summary of additive, subtractive, and conditional morphological operations, adapted from Pratt (2007).

Name	Output	Condition
Additive operations		
Interior fill	Logical 1	If all four-connected neighbour pixels are logical 1.
Diagonal fill	Logical 1	If creations eliminates the eight-connectivity of the background.
Bridge	Logical 1	If creation results in connectivity of previously unconnected neighbouring logical 1 pixels.
Eight-neighbour dilate	Logical 1	If at least one eight-connected neighbour pixel is logical 1.
Fatten	Logical 1	If at least one eight-connected neighbour pixel is logical 1, provided that creation does not result in a bridge between previously unconnected logical 1 pixels.
Subtractive operations		
Isolated pixel remove	Logical 0	If all eight-connected neighbour pixels of a logical 1 pixel are logical 0.
Spur remove	Logical 0	If a logical 1 pixel is connected to a single eight-connected logical 1.
Interior remove	Logical 0	If all four-connected neighbour pixels of a logical 1 pixel are logical 1.
H-break	Logical 0	If a logical 1 pixel is H-connected, that is, for example, if pixels to the northeast, east, southeast, southwest, west, and northwest are logical 1.
Eight-neighbour erode	Logical 0	If at least one eight-connected pixel of a logical 1 pixel is logical 0.
Conditional operations		
Majority black	Logical 0	If less than five pixels are logical 1.
	Logical 1	If five or more pixels are logical 1.
Shrink	Logical 0	Until an object, consisting of logical 1 pixels, without holes erodes to a single pixel at, or near, its centre of mass. An object with holes erodes to a connected ring midway between each hole and its nearest boundary.
Thinning	Logical 0	Until an object, consisting of logical 1 pixels, without holes erodes to a minimally connected stroke located equidistant from its nearest outer boundaries. An object with holes erodes to a minimally connected ring between each hole and its nearest boundary.
Skeletonising	Logical 0	Until an object, consisting of logical 1 pixels, is reduced to a minimally connected set of points that are equally distant from the two closest points of the object boundary.
Thickening	The thickening process ensures that objects separated by a double width boundary are not fused together when fattened. This is achieved by iteratively thinning the background of the image, and then performing a diagonal fill operation.	

A combination of morphological operations can be used to process a palmprint image after edge detection. In palmar flexion crease identification, thinning, thickening, skeletonising, and isolated pixel remove are commonly used operations (Zhang and Shu, 1999; Wu *et al.*, 2002b; Li and Leung, 2006). However, depending on the requirements and objectives of the system, any number of morphological operations can be used. A discussion of mathematical morphology, and the implementation details of many morphological operations, can be found in Serra (1986), Haralick *et al.* (1987), and Soille (2003).

After edge detection and morphological processing, palmar flexion creases can be represented in a variety of storage formats, such as chain code (Wu *et al.*, 2002b), straight line segments (Zhang and Shu, 1999), or parametric space vectors (Li and Leung, 2006), which depend on the nature of the feature extraction algorithm. Consequently, the feature extraction algorithm, and therefore, the palmar flexion crease storage format, determines the basis of the feature matching algorithm (Li, 2003).

A matching algorithm is a component in an identification system which returns a score, s, indicating the similarity of two or more given samples. The reliability of such a matching score is influenced by many factors, typically caused by user, sensor, or algorithm variability. Given these variables, two samples, and a score threshold, T, the matching algorithm returns one of two outcomes:

$$\text{The } \textit{null} \text{ hypothesis: } h_o \Rightarrow \text{the two samples match;}$$
$$\text{The } \textit{alternate} \text{ hypothesis: } h_a \Rightarrow \text{the two samples do not match.}$$

From these hypotheses, the identification rate (and error rates) of an identification metric can be calculated (Bolle *et al.*, 2004). In palmprint identification, as in most identification systems, the identification rate is used to demonstrate the accuracy, fallibility, and efficiency of a given system. This identification rate is typically determined from experimental results as:

$$rate = \frac{successes}{attempts},$$

where $attempts$ is the number of times the system was queried, that is, the number of palmprint samples which were given to be identified, and $successes$ is the number of times the system returned the correct answer (Li, 2003).

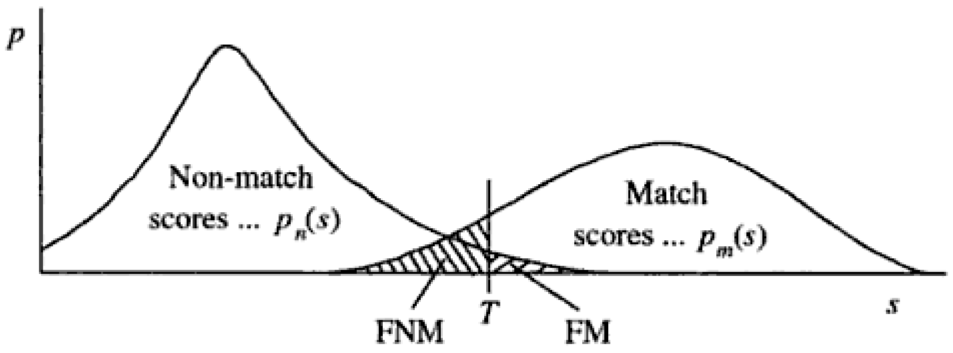


Figure 28. Probability densities of matching and non-matching scores (Bolle *et al.*, 2004).

In deciding between the two hypotheses, that is, if the sample matches or not, a matching algorithm can make two types of error: a false acceptance (or type I error), where a matching score, s, is greater than the threshold, T, and the algorithm decides H_o is true, when actually H_a is true; and a false rejection (or type II error), where a matching score, s, is less than or equal to the threshold, T, and the algorithm decides H_a is true, when actually H_o is true. As described earlier, the false acceptance rate (FAR) and the false rejection rate (FRR) can then be given as the frequencies at which false acceptance and false rejection occur. As illustrated in Figure 28, the non-matching score distribution, $p_n(s)$, and the matching score distribution, $p_m(s)$, often overlap. In such cases, it is not possible to determine a threshold value where both the FAR and FRR are zero (Bolle *et al.*, 2004), and so a compromise between false acceptance and false rejection is required (Li, 2003).

However, to determine the optimal performance characteristics of a given matching algorithm at various threshold levels, the FAR and FRR can be plotted against each other as:

$$ROC(T) = (FAR(T), FRR(T)),$$

where *ROC* is a two-dimensional curve, which is referred to as a receiver operating characteristic (ROC) curve. As shown in Figure 29, any point on the ROC curve defines an operating point of the matching algorithm, which can be chosen using a threshold value, T, or a FAR or FRR value (Bolle *et al.*, 2004).

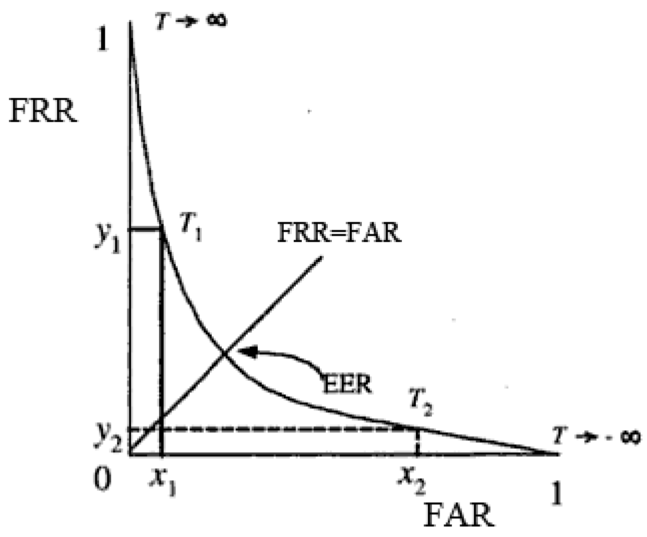


Figure 29. A ROC curve expresses the compromise between FAR and FRR (Bolle *et al.*, 2004).

The equal error rate (EER), as shown in Figure 29, is the value of the error rates for an operating point of a matching algorithm at the intersection of the line $FRR = FAR$. The EER is a single value representing the quality of the matching algorithm that can be used to benchmark system performance. However, as many systems operate with an unequal FRR and FAR to maintain a balance between efficiency and accuracy, the EER is typically an unreliable summary of operational performance for real world applications (Bolle *et al.*, 2004).

2.5. Summary

This chapter presented an overview of palmprint identification, in which the definition of a palmprint, the formation and structure of palmprint features, palmprint identification metrics, automated palmprint identification systems, and a number of computer vision and pattern recognition algorithms were discussed. In Chapter 3, a new method of manual palmprint identification using palmar flexion creases is described, and experimental results are presented that show palmar flexion creases to be an effective method of palmprint identification.

3. Manual palmprint identification

Traditional palmprint identification strategies, particularly in forensic science, use matching of minutia points, that is, the same methodology that is used to match fingerprints, to identify an individual. Therefore, new methods of palmprint identification, that complement existing identification strategies, or reduce analysis and comparison times, will be of considerable benefit to palmprint identification communities worldwide. To this end, this chapter describes a method of manual palmprint identification and matching using palmar flexion creases, and presents an analysis and discussion of experimental results that were collected using the proposed method.

3.1. Introduction

Despite the frequency of palmprint impressions at scenes of crime, where they constitute 30% of all hand print impressions (Libal, 2006; Parker, 2006), little work has been reported on improving palmprint identification for forensic science (Jain and Feng, 2009). Alternative methodologies, new identification metrics, or additions to existing identification strategies, that can be used to improve palmprint identification, will be of considerable assistance to fingerprint experts and law enforcement communities. To this end, one group of palmprint features, that provide a rich source of information in palmprint images, are palmar flexion creases. Palmar flexion creases have often been considered an accessory feature in palmprint identification (Zhang *et al.*, 2003). However, they provide valuable information about a person, and present an opportunity for identification based on low-resolution, distorted, or partial palmprint images (Kong and Zhang, 2002). As shown in Figure 30, palmar flexion creases are visible in palmprint images, such as those from a digital camera, scanner, or digital video, and in palmar marks, such as inked, Live Scan, or those recovered from scenes of crime. Furthermore, although palmar flexion creases are genetically dependent (Kong *et al.*, 2006b), they can be differentiated based solely on their spatial characteristics (Zhang and Shu, 1999), even in monozygotic twins using secondary creases and wrinkles (Kong *et al.*, 2006b), and therefore, are a prime candidate in an identification metric.

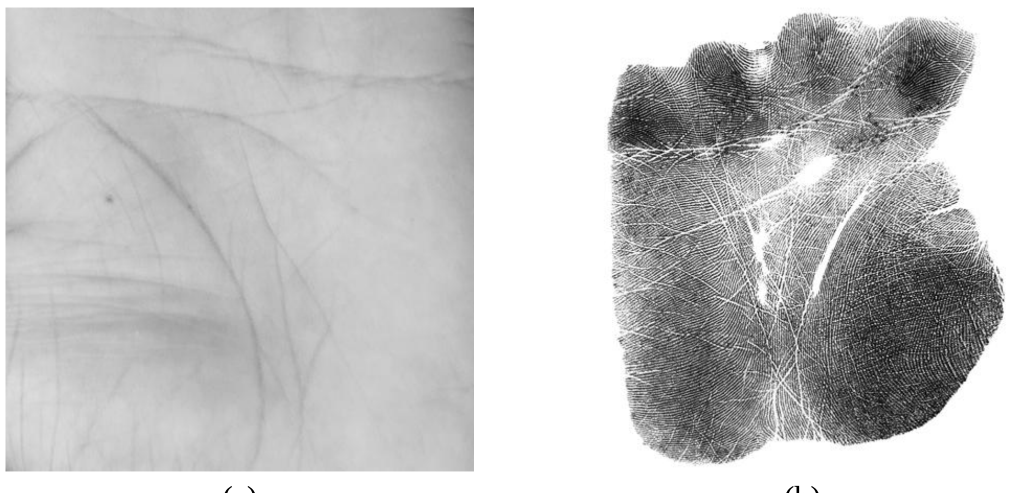

(a) (b)

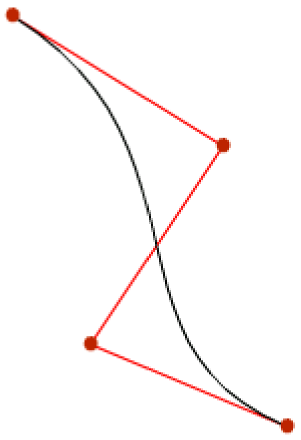
Figure 30. Palmar flexion creases in a) a digital camera image (The Hong Kong University of Science and Technology, 2003), and b) a Live Scan image.

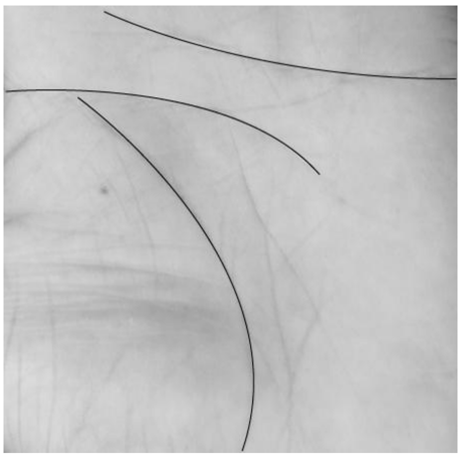
As described in Chapter 2.2, palmprints are currently identified by applying a similar method to that used for fingerprints, in which a fingerprint expert identifies minutiae from crime scene prints, enters the marked prints into an AFIS, which compares them to stored palmar images, finding the best matches. The expert then compares the best matches with the crime scene prints in order to establish identity. If no match is found, or if identity cannot be established, then a new database record is stored, allowing for future identification (Townley and Ede, 2004). However, as the surface area of a palmprint is approximately 40 times larger than that of a fingerprint, and may contain 8 times the number of minutia points (Jain and Feng, 2009), identification by minutiae can be a time consuming process. Furthermore, approximations by Jain and Feng (2009) suggest that a modern AFIS can be up to 64 times slower when matching palmprints than when matching fingerprints. To this end, alternative methodologies, that can complement existing identification strategies, reduce analysis and comparison times by fingerprint experts, or partially automate the process, will be of considerable benefit to palmprint identification communities worldwide. The importance of palmprint identification is recognised by the NPIA in the United Kingdom (Police IT Organisation, 2005) and the FBI in the United States, whose Next Generation Integrated Automated Fingerprint Identification (NGI) system includes initiatives for integrated national palmprint identification functionality (Federal Bureau of Investigation, 2005b).

In biometrics, palmprint identification using palmar flexion creases, secondary creases, and wrinkles has shown recognition rates that are comparable with other methods of identification (Kong *et al.*, 2009), such as face recognition (Zhao *et al.*, 2003; Kong *et al.*, 2005) or automated fingerprint identification (Yager and Amin, 2004a; Yager and Amin, 2004b). However, despite the advances in biometrics, little consideration has been given to forensic applications. Many palmprint identification systems are specific to one type of image, such as digital camera images in biometrics, and use format dependent identification metrics. This makes forensic identification difficult, if not impossible, as cross-format identification, such as developed latent prints to inked or Livescan images, is not possible. Consequently, a format independent identification method, using features that persist across formats, is required. Furthermore, palmar marks recovered from scenes of crime are often partial, smudged, or otherwise distorted. For this reason, a forensic palmprint identification system must be capable of correcting distortions, such as rotation and translation, and identifying palmar marks in which limited identification features are available. However, any new identification metric, such as palmar flexion creases, must first be tested to determine its robustness to orientation, translation, and deformation, such as that resulting from skin elasticity, before other factors, such as partial identification, distortions from the recipient surface, or other external factors, can be introduced. If these initial tests cannot be established, the usefulness of such a metric is limited. To this end, this chapter presents a new method of identification, based on palmar flexion creases, using complete palmprint images, which aims to establish the feasibility of palmar flexion crease identification in a forensic context.

3.2. Materials and methods

Given a palmprint image, the most effective method for representing the palmar flexion creases is by a collection of lines. To this end, one method for representing the palmar flexion creases, that has been used previously in automated palmprint identification (Zhang and Shu, 1999; Wu *et al.*, 2002a), is by a series of connected straight line segments. However, since the palmar flexion creases do not typically follow a straight line, and manually modelling a curved line using straight line segments may be difficult or time consuming, an alternative method is required. To this end, a common method for describing curved lines, that is used extensively in computer graphics and computer aided design (Marsh, 2005), is by Bézier curves.





(a) (b)

Figure 31. Example of a) a Bézier curve, and b) a palmprint image (The Hong Kong University of Science and Technology, 2003) with the major palmar flexion creases overlaid by three Bézier curves.

A Bézier curve, as shown in Figure 31(a), is a polynomial curve which is represented by a sequence of $n+1$ points, called control points, which are connected together to design the basic shape of the curve (Marsh, 2005). In computer graphics, the most common type of Bézier curves are quadratic, that is, three control point, or cubic, that is, four control point, which are connected together, or patched, to create larger, more complex curves (Kodicek, 2005). As described in Marsh (2005), given three control points, $b_0(p_0, q_0)$, $b_1(p_1, q_1)$, and $b_2(p_2, q_2)$, a quadratic Bézier curve can be defined as:

$$B(t) = (1-t)^2(p_0, q_0) + 2(1-t)t(p_0, q_0) + t^2(p_2, q_2), \text{ for } t \in [0,1],$$

and, given four control points, b_0, b_1, b_2, and b_3, a cubic Bézier curve as:

$$B(t) = (1-t)^3 b_0 + 3(1-t)^2 t b_1 + 3(1-t)t^2 b_2 + t^3 b_3, \text{ for } t \in [0,1].$$

As shown in Figure 31(b), a collection of Bézier curves can be used to accurately represent the major palmar flexion creases. However, as shown in Figure 31(a) by the location of the control points in relation to the curve, the input system can be complicated for the user, as while every point influences the curve, the curve only passes through the end points. To this end, a more intuitive representation of the control points, which is commonly used in graphic design applications, is by nodes and anchor points. As shown in Figure 32, using nodes and anchor points, the curve passes through each node, V_1, V_2, and V_3, and is controlled by anchor points, C_{12},

C_{21}, C_{22}, and C_{31}, which control the tangent and curvature of the line (Kodicek, 2005).

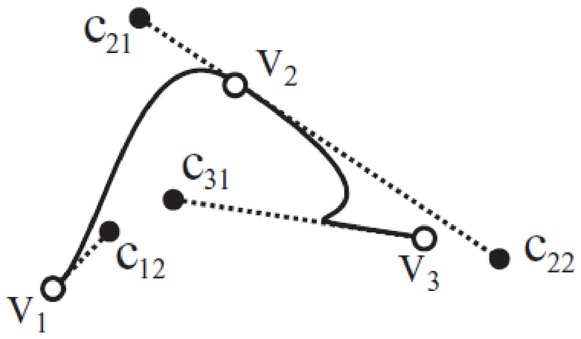


Figure 32. A three-node Bézier curve control by control and anchor points (Kodicek, 2005).

Given these improvements, the curve can be controlled more intuitively by the user. However, due to the global nature of the Bézier curve, that is, because every point on a Bézier curve is the result of a calculation using all of the defining vertices, a change in one control point influences the entire curve (Rogers and Adams, 1990). For this reason, despite improvements to the control system, the lack of local curve control can be detrimental. To this end, for applications that require local curve control, a spline curve, that is, a sequence of curve segments that are connected together for form a single continuous curve, such as a piecewise Bézier, is often used. However, an alternative type of curve, that is a generalisation of the Bézier curve, and which exhibits non-global behaviour, is a B-spline curve. A B-spline curve with $n+1$ control points, $B_0, B_1, \ldots, B_n$, can be defined as:

$$P(t) = \sum_{i=0}^{n} B_i N_{i,k}(t),$$

where $t_{k-1} \le t \le t_{n+1}$ and $N_{i,k}$ are the basis functions:

$$N_{i,k}(t) = \frac{(t - x_i) N_{i,k-1}(t)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - t) N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}},$$

and

$$N_{i,1}(t) = \begin{cases} 1 & \text{if } x_i \le t < x_{i+1} \\ 0 & \text{otherwise} \end{cases},$$

where the values of x_i are elements of a vector satisfying the relation $x_i \le x_{i+1}$.

As each vertex is associated with a unique basis function, the B-spline curve, unlike the Bézier curve, exhibits non-global behaviour, such that each vertex affects the shape of the curve only over the range of parameter values where the basis function is non-zero (Rogers and Adams, 1990). Given this property, the B-spline curve is an ideal candidate for efficiently describing curved lines. However, for manually identifying palmar flexion creases, where the curvature of the line is often slight, the relationship between the control points and the curve is still more complex than is required. To this end, another method for representing the palmar flexion creases is by a collection of cardinal splines.

A cardinal spline is a cubic polynomial curve that has tangent lines defined by a series of control points and a tension parameter. A cardinal spline guarantees smooth interpolated motion between control points, and ensures that each point will be passed, generating a seamless, continuous curve through multiple control points (Kochanek and Bartels, 1984). Given $n+1$ points, $p_0, \ldots, p_{n+1}$, to be interpolated with n segments, each curve has a starting point, p_i, and an ending point, p_{i+1}, with a starting tangent, m_i, and an ending tangent, m_{i+1}, defined by:

$$m_i = c \cdot (p_{i+1} - p_{i-1}),$$

where c is a constant between 0 and 1 that modifies the length, or tension, of the tangent (Kochanek and Bartels, 1984). Figure 33 shows an example of a cardinal spline with three different tension values.

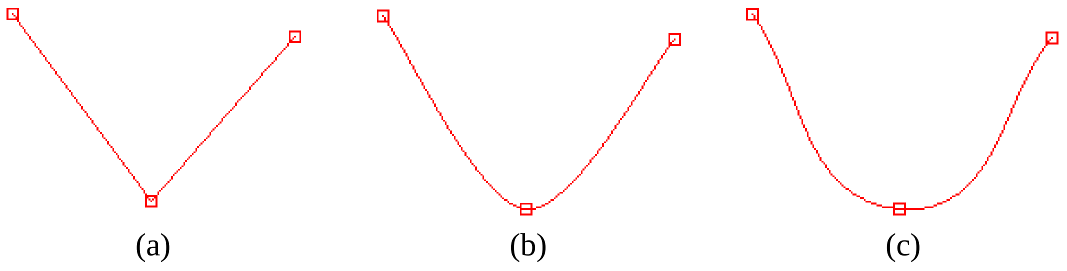


Figure 33. Example of a cardinal spline with three different tension values: a) 0, b) 0.5, and c) 1.

As with a Bézier curve, using a series of manually identified control points, any number of line features can be represented by a collection of cardinal splines. A palmar flexion crease can be identified by adding a control point at the start point, end point, and points of interest along the flexion crease. In contrast to a Bézier curve, which requires anchor points to define the tangents and curvature of the line, a cardinal spline can be interpolated simply using control points. As shown in Figure

34, when a series of control points are interpolated by a cardinal spline, an accurate representation of the palmar flexion creases is achieved.

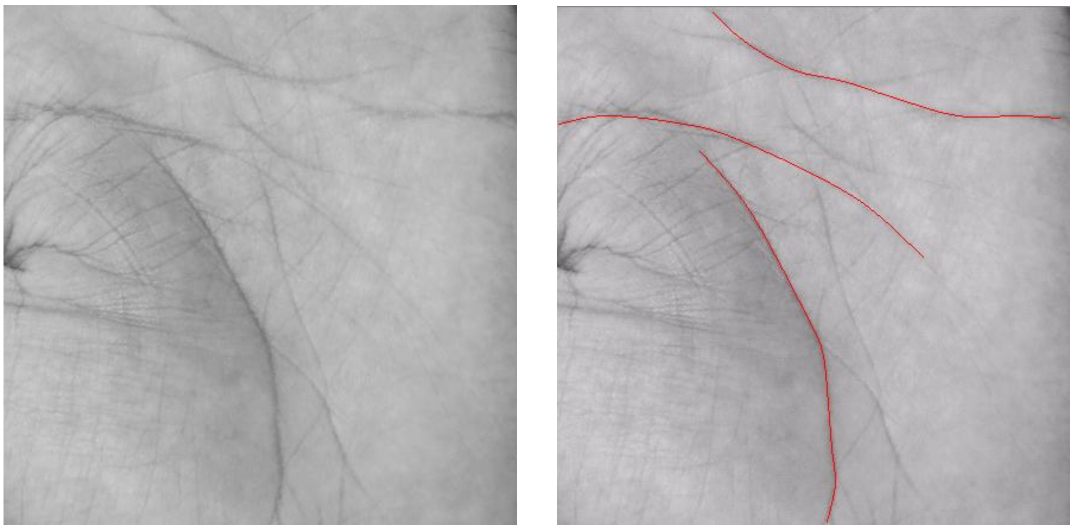

(a) (b)

Figure 34. Example a) online palmprint image (The Hong Kong University of Science and Technology, 2003) with b) three cardinal splines overlaid. The tension value for all three cardinal splines is 0.5.

Given the definition of cardinal splines, and to enable user interaction with the identification system, a suitable user interface, that allows the user to accurately and efficiently identify the palmar flexion creases using cardinal splines, is required. To this end, a number of objectives, which form the requirements user interface, can be defined:

1. To be able to accurately identify the palmar flexion creases using a suitable control system.
2. To be able to delete or modify existing cardinal splines to enable repeated identification.
3. To be able to adjust the tension parameter of each cardinal spline to allow maximum user control.
4. To save the palmar flexion crease configuration, that is, the control points and tension parameter, to enable palmar flexion crease matching.

Given these objectives, a simple user interface, as shown in Figure 35, can be designed. The curve tension slider or curve tension text box can be used to adjust the tension parameter of each individual cardinal spline, while the delete and delete all buttons can be used remove existing cardinal splines from the palmprint image. The go to image text box and button, previous button, and next button can be used to move forwards and backwards through the image database, and the save button can be used to save the palmar flexion crease configuration.

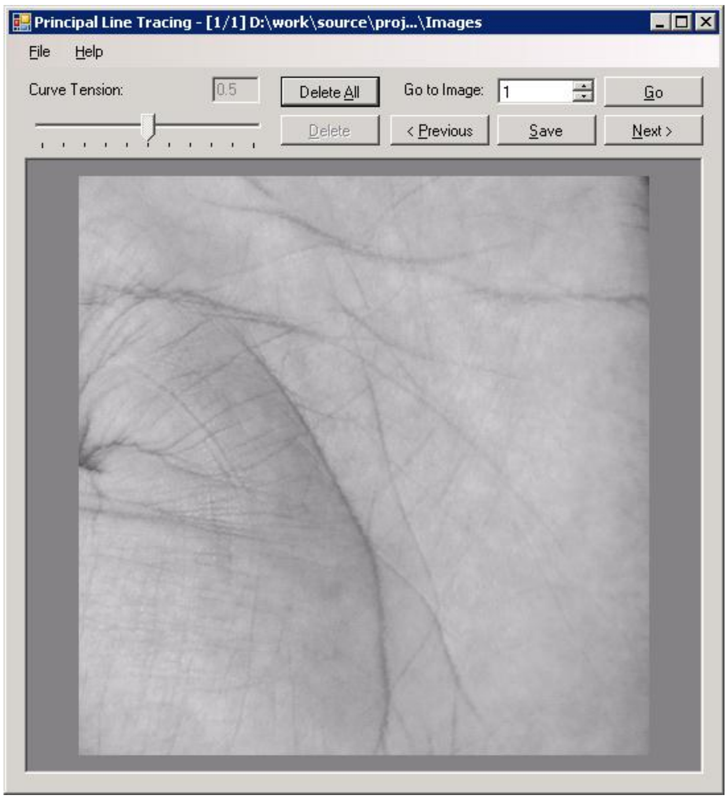
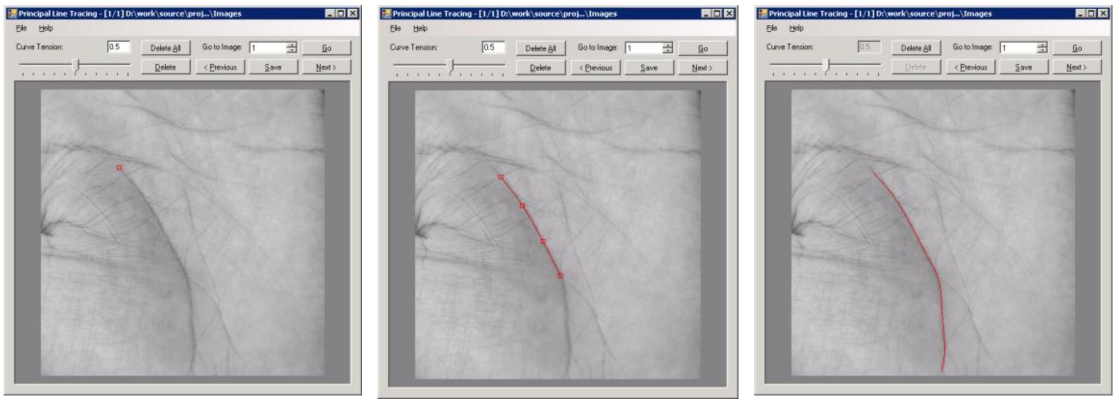


Figure 35. A manual palmar flexion crease identification system user interface.

Furthermore, as shown in Figure 36, an intuitive control system for identifying the palmar flexion creases using cardinal splines can be designed. To identify a palmar flexion crease, as shown in Figure 36(a), the user clicks the left mouse button at the start of the palmar flexion crease to create a control point. The user then continues to use the left mouse button to add control points along the palmar flexion crease, as shown in Figure 36(b). Any number of control points can be added by the user to identify the palmar flexion creases. However, once the user has finished identifying a palmar flexion crease, the right mouse button can be clicked to finish identification. As shown in Figure 36(c), once identification is complete, the control points are no longer displayed, and the palmar flexion crease configuration is automatically saved. The user can then repeat the identification process for any number of line features in the palmprint image.



(a) (b) (c)

Figure 36. A manual palmar flexion crease identification control system: a) the user creates a control point to start the identification process, then b) continues to identify the palmar flexion crease using additional control points, and c) completes the identification process.

After identification, the palmar flexion creases are represented as a collection of control points and a tension parameter, which can be used to reconstruct the palmar flexion creases using cardinal splines, or in palmar flexion crease matching to calculate the degree of similarity between two or more sets of palmar flexion creases to determine if they belong to the same palm.

To calculate the similarity between palmar flexion crease configurations from two or more palmprint images, the control points from each palmar flexion crease need to be compared. To this end, the simplest method for comparing two sets of control points is to compare their spatial location, that is, point by point, without any special processing or normalisation. Point by point comparison can be used to quickly identity matching palmar flexion creases, and a match percentage can be obtained. However, as each palmar flexion crease may be identified by a different number of control points, and without a reliable algorithm to determine which control points to compare, point by point comparison may perform poorly. Therefore, prior to palmar flexion crease matching, a control point normalisation algorithm can be designed. To this end, to ensure each palmar flexion crease is represented by an equal number of control points, a second group of cardinal splines, controlled by a fixed number of equally spaced control points, can be automatically created to match the original input. This automated normalisation allows any configuration of control points, while ensuring reliable annotation for palmar flexion crease matching. Figure 37 shows an example of a cardinal spline before and after automated point normalisation.

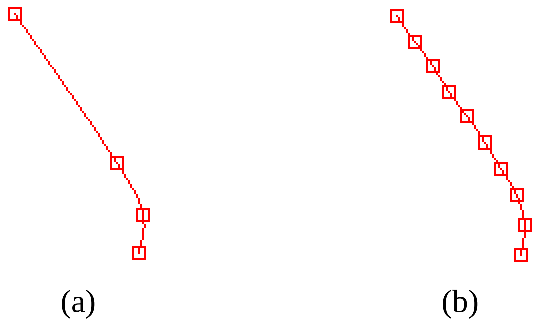


Figure 37. Example of a cardinal spline a) before, and b) after automated point normalisation.

Furthermore, given a normalised collection of palmar flexion creases, an alternative method to point by point comparison for comparing two sets of control points, which can be used to determine the nearest control points to compare, is a nearest neighbour algorithm. Given a set, P, of n points in a d-dimensional space, $\Re^d$, and a query point, $q \in \Re^d$, a nearest neighbour algorithm can be used to find the closest point, that is, closest by Euclidean distance, in P to q (Chen and Lu, 2008). To this end, a number of algorithms, such as linear search, space partitioning, and locality hashing, can be designed to solve nearest neighbour problems, each of which will have their own advantages and disadvantages (Chen and Lu, 2008). However, due to the small number of control points and low dimensionality in each palmar flexion crease configuration, a simple linear nearest neighbour algorithm is sufficient. A linear nearest neighbour algorithm can be completed in linear time, and is solved by a sequential search. The algorithm calculates the distance from q to all points in P, and records the distance between each query. The point in P with the minimum distance from q once all points have been processed is deemed to be the nearest neighbour (Sedgewick, 1983). Using this theory, the nearest control point in a palmar flexion crease configuration, A, to each query control point in another palmar flexion crease configuration, B, can be calculated, and the distance between A and B can be measured.

However, as slight variations are present in nearly all palmprint images captured over time, a matching algorithm is required that allows considerations to be made for variations in rotation and translation. To this end, generalised Procrustes analysis (GPA) is a multivariate statistical method, used in shape analysis, to find the optimal superimposition of two or more point configurations. The algorithm consists of three transformations: translation, in which the centroid of each configuration is shifted to a common origin; rotation, in which all points are displaced by a common angle, keeping the distance of each point from the origin unchanged; and scaling, in which all points are contracted or expanded in a straight line from the origin. The optimal transformation is defined as the smallest sum of squared distances between corresponding points in the configurations (Goodall, 1991). GPA can be used, in this context, to minimise the distance, in terms of rotation and translation, between two sets of palmar flexion creases. Figure 38 shows the point configurations for two sets of palmar flexion creases separately derived from the same palm before and after GPA.

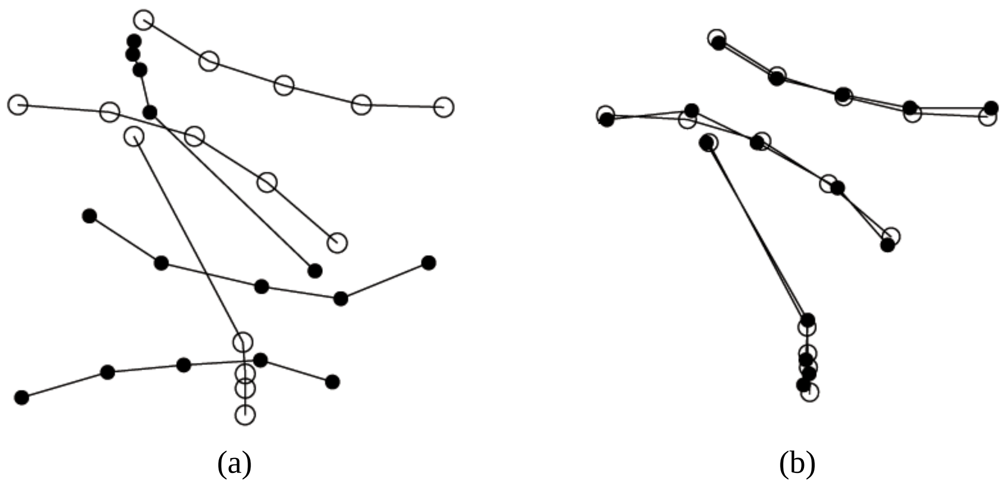


Figure 38. Two flexion crease configurations from the same palm a) before and b) after GPA.

Given a nearest neighbour algorithm to calculate the distance between control points, and GPA as a pre-processing method to correct rotation and translation, the sum of distances between two sets of point configurations can be calculated and used as a measure of similarity. Furthermore, given a threshold to separate matching and non-matching palmar flexion crease configurations, the GAR and FAR for palmar flexion crease matching can be calculated. The lower the distance between two sets of palmar flexion crease configurations, the higher the probability they come from the same palm.

The algorithms described in this chapter were implemented on Microsoft Windows XP using Mathworks MATLAB R2007b, and the C# programming language. The algorithms were tested using The Hong Kong University of Science and Technology’s (2003) hand image database. The source code for each algorithm is available in appendix A.

3.3. Results and discussion

To evaluate the effectiveness of palmar flexion creases as a method of identification, 100 online palmprint images, that is, 100 images from 100 different left hand palms, captured during a single session, from The Hong Kong University of Science and Technology's (2003) hand image database, measuring a mean of 380×380 pixels at 72 pixels per inch, were identified, then compared. The three major palmar flexion creases, that is, the thenar crease, the distal transverse crease, and the proximal transverse crease, were manually identified in each palmprint image using the method described in Chapter 3.2. Then, each set of manually identified palmar flexion creases was compared, again, using the method described in Chapter 3.2, to every other set of manually identified palmar flexion creases. Furthermore, as shown in Figure 39, during the comparison process, the extracted data was subjected to a variety of dynamic image alterations, 10 times each, to mimic some of the types of alterations that can be found in palmar marks from scenes of crime.

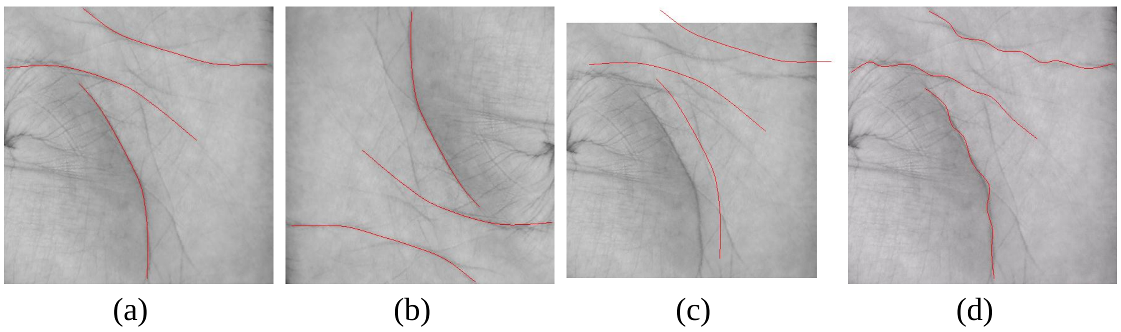


Figure 39. An example flexion crease configuration a) before, and b) after rotation by 180 degrees, c) after translation by 25 pixels, and d) after noise corruption with a boundary size of 16 pixels or 5.6 mm.

Mark orientation was tested by random rotation from 0 to 360 degrees; spatial translation was tested by random translation from -50 to 50 pixels, that is, -17.6 to 17.6 mm; and deformation resulting from skin elasticity, degradation caused by smudging and slippage, or between and within user input variations, that is, variability of manual annotation relative to the flexion crease location, was tested by noise corruption, using random translation of each point along the flexion crease within a defined boundary size. All image alterations were applied to every palmar flexion crease configuration in every comparison, and the comparison process was repeated with 8 different noise corruption boundary sizes: 2, 4, 6, 8, 10, 12, 14, and 16 pixels, or 0.7, 1.4, 2.1, 2.8, 3.5, 4.2, 4.9, and 5.6 mm.

Thus, to test robustness, the comparison process was repeated 8 times, and each time, every palmprint image was subjected to dynamic image alterations to produce 10 modified palmprint images. For each of the 8 repetitions, a different noise corruption boundary size was used, and the 10 variations of each manually identified palmar flexion crease configuration were compared. 1000000 comparisons were recorded in each repetition. Therefore, for a single palm in a single repetition, 1%, or 10000, of all comparisons were from the same palm, and 99%, or 990000, were not. Figure 40 shows the upper and lower distances, that is, the sum of distances of each compared control point using the method described in Chapter 3.2, of correct and incorrect matches at each boundary size. The upper and lower distances indicate at which point, in terms of boundary size, incorrect matches begin to occur. As the upper distance between matching palms increases, and the lower distance between non-matching palms decreases, the chance of an incorrect identification rises. As shown in Figure 40, when the boundary passes 8 pixels, that is, a distance of 2.8 mm, incorrect matches may occur. The probability of false acceptance, that is, when incorrect matches are identified as correct, and of false rejection, that is, when correct matches are identified as incorrect, increases as the boundary size expands.

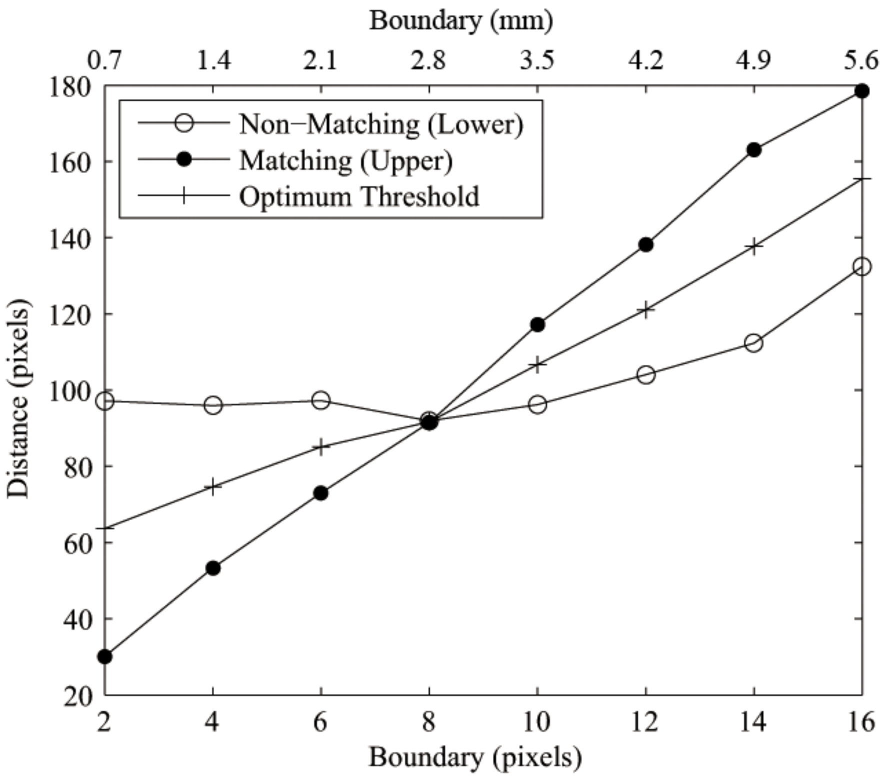


Figure 40. Upper and lower distances of correct and incorrect matches at each boundary size.

Using the values from Figure 40, the optimum threshold, where distances above the threshold are considered matches, and distances below the threshold are considered non-matches, can be calculated, and the GAR, that is, the number of matching palms that were correctly identified, and FAR, that is, the number of non-matching palms that were identified as matching, at each boundary size can be determined. Figure 41 shows the ROC curve of GAR against FAR at each boundary size. A single comparison is considered genuine if two conditions are met: first, if both features are from the same palm, and second, if the distance between those features is above a given threshold. All other comparisons are considered false. Again, 1000000 comparisons were recorded in each experiment, and the threshold was calculated as half the difference between the upper and lower distances of correct and incorrect matches.

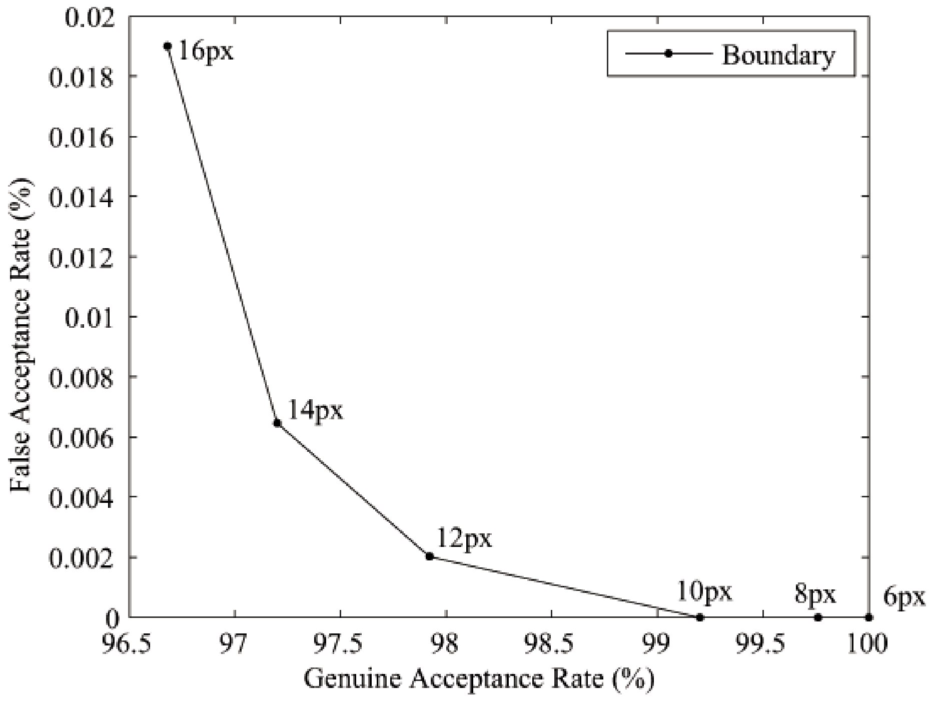


Figure 41. Receiver operating characteristic curve for flexion crease identification.

As shown in Figure 41, palmar flexion creases can be recognised with a 100% GAR, and 0% FAR, when identified within 6 pixels, or 2.1 mm, of their actual location. However, as the boundary size increases, the GAR falls and the FAR rises. Figure 42 shows an example of the boundary for 6 (2.1 mm), 10 (3.5 mm), and 16 (5.6 mm) pixels.

These results show palmar flexion creases to be a robust identification metric, which allows input variation, that is, allows the examiner to label the flexion creases within a distance of their actual location, while still producing reliable results. Using a 10 pixel, or 3.5 mm, boundary, a 99.2% GAR can be achieved with a 0% FAR, providing a good compromise between input variation and false rejection. To be clear, when a set of palmar flexion creases are identified within 3.5 mm of their actual location, correct identification occurs 99.2% of the time.

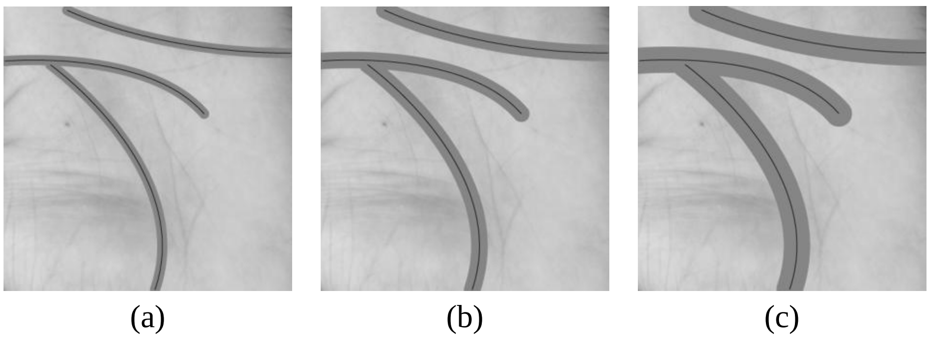


Figure 42. Example of the boundary at a) 6 pixels (2.1 mm), b) 10 pixels (3.5 mm), and c) 16 pixels (5.6 mm).

However, the results presented here are for a small sample, low variation, palmprint database (The Hong Kong University of Science and Technology, 2003), with annotation by a single user. In all cases, each palmar flexion crease could be easily identified, and while image defects are present, they are relatively low compared to a real world data set, where there are often inconsistencies in image capture quality and palmprint coverage. Furthermore, various factors can affect the quality of a palmprint image, such as bruising, scars, and cuts in inked, digital, or Livescan images, while uncontrolled environments result in unspecified distortions in latent prints, based on artefacts introduced during print deposition such as slippage, smudging, and the nature of the recipient surface. In all cases, distortions are easily introduced due to the characteristics of human skin, such as elasticity and worn friction ridge skin (Ashbaugh, 1999a). That said, through artificial rotation, translation, and additive noise, as in the experiments described in this thesis, many image defects and user input variation can be tested. Furthermore, despite its small sample size, the palmprint database contains examples from each palmprint category, as defined by Wu *et al.* (2004c), in which 13800 palmprints were classified based on their flexion crease configurations.

The results show cardinal spline interpolation and GPA normalisation to be effective methods in palmar flexion crease identification. A cardinal spline guarantees smooth interpolated motion between control points, and generates a seamless, continuous curve. In contrast with other spline interpolation methods (Boor, 2001), cardinal spline tangents can be computed from their corresponding control point locations, without the need for user defined coordinates. This ensures that each point will be passed, providing an efficient method of user input. Furthermore, GPA is a promising method of normalisation, that can be used to offset the effects of rotation, translation, scaling, and reflection in manually identified palmar flexion creases. GPA can be computed quickly and efficiently (Gower, 1975; Berge, 1977; Goodall, 1991), and without changing the relative position of the cardinal splines, while maximising the agreement of corresponding point configurations. Palmprint identification using palmar flexion creases shows promising results as a standalone identification metric, or as a complement to existing identification systems.

3.4. Summary

This chapter described a method of manual palmprint identification and matching using palmar flexion creases, and presented an analysis and discussion of experimental results that were collected using the proposed method. Experimental results showed that palmar flexion creases from 100 palms, each modified 10 times to mimic some of the types of alterations that can be found in crime scene palmar marks, when labelled within 10 pixels, or 3.5 mm of the flexion crease, can be identified with a 99.2% genuine acceptance rate and a 0% false acceptance rate. In Chapter 4, a method of automated flexion crease recognition is described, that can be used to automatically identify palmar flexion creases in online palmprint images.

4. Automated flexion crease recognition

Improved methods of automated palmprint identification have the potential to reduce palmprint analysis and comparison times by complementing existing identification strategies, and through the development of new identification systems. To this end, this chapter describes a new method of automated flexion crease recognition, and presents an analysis and discussion of experimental results, in which the palmar flexion creases from 100 online palmprint images (The Hong Kong University of Science and Technology, 2003) are compared.

4.1. Introduction

Automated identification systems represent the single biggest advance in friction ridge skin identification technology (Fisher, 2004). In forensic science, automated identification systems allow fast identification at local, national, and international level, by automating slow, labour-intensive processes previously undertaken only by specially trained examiners (Cole, 2004). Similarly, automated identification using biometrics offers advances such as negative recognition, that is, the process by which a system determines that an individual is enrolled despite the individual denying it, and non-repudiation, that is, an auditing process whereby an individual cannot deny accessing a system they have previously accessed. Furthermore, integrated verification procedures, such as quality assessment by the capture sensor, allow an additional level of quality control that is prohibitive in manual identification processes (Ross *et al.*, 2006). To this end, improved methods of automated palmprint identification have the potential to reduce palmprint analysis and comparison times, by complementing existing identification strategies, and through the development of new automated identification systems.

In this chapter, a new method of automated flexion crease identification is proposed that uses internal image seams (Avidan and Shamir, 2007) to identify palmar flexion creases in online palmprint images. Using the proposed approach, in the feature extraction stage, the importance of each pixel in a palmprint image is calculated using an energy function, that is, an edge detection algorithm or an image importance measure, and the palmar flexion creases are extracted as optimal internal image seams. Then, in the matching stage, a *k*d-tree nearest neighbour algorithm (Bentley,

1975) is used to measure the similarity between two or more sets of palmar flexion creases to determine if they belong to the same palm. Using this method, any number of major flexion creases, secondary creases, or wrinkles, can be extracted from an online palmprint image.

4.2. Materials and methods

Given a palmprint image, and an energy function, that is, a measure of pixel importance or intensity, the palmar flexion creases are the lines that contain the highest energy values. Therefore, an optimal strategy for locating the palmar flexion creases would be to identify, in ascending order, the lines with the highest overall energy. As described in Chapter 2.4, a typical palmar flexion crease identification system begins with an edge detection algorithm, which is used to segment the palmprint image, that is, separate the line features from the background texture. To this end, there are numerous edge detection algorithms found in literature that could be used to segment a palmprint image, many of which have already been used in palmprint identification, and are described in detail in Chapter 2.3 and 2.4. Figure 43 shows an example of a palmprint image that has been filtered by the author using the following edge detection algorithms: the Sobel operator, the LoG operator, the Canny edge detector, and the phase congruency operator.

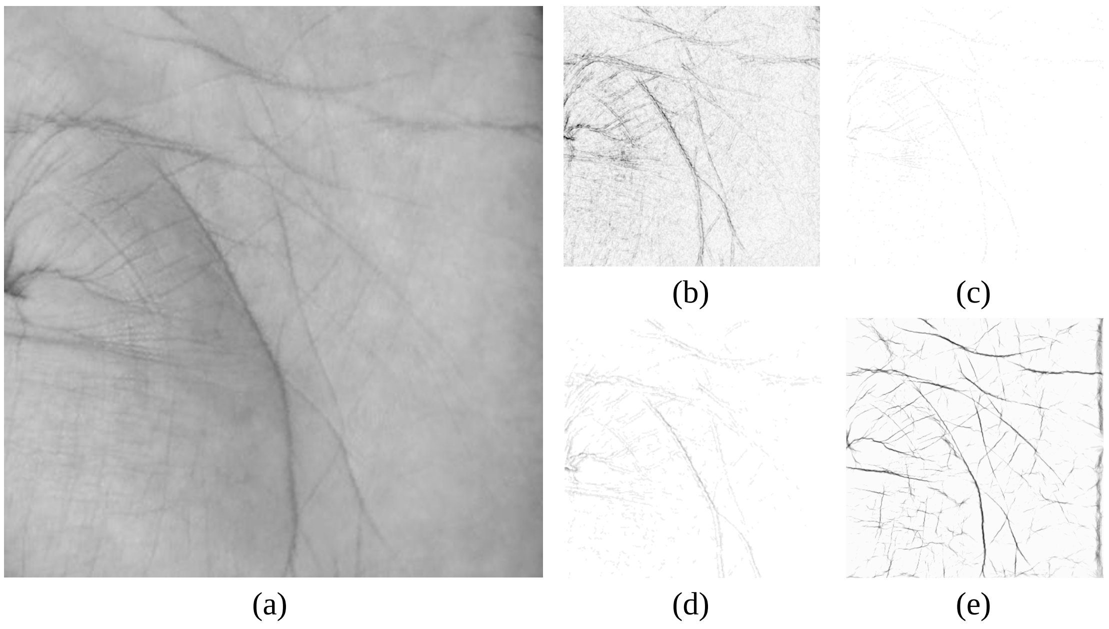


Figure 43. A palmprint image (The Hong Kong University of Science and Technology, 2003) a) before, and after applying b) the Sobel operator, c) the Laplacian of Gaussian operator, d) the Canny edge detector, and e) the phase congruency operator. For clarity, the colours of the edge images have been inverted.

When choosing an algorithm for edge detection, a number of factors need to be considered. For automated palmar flexion crease identification, of which the main objective, in this case, is to extract the lines with the highest overall energy, an accurate response to step and roof edges is required. The advantages and disadvantages of each algorithm are described in Chapter 2.4. However, for palmprint images, a number of specific observations can be made by examining Figure 43. The LoG and Canny operators only respond accurately to the strongest edges, while discounting many of the palmar flexion crease lines. Conversely, the Sobel operator responds to the major palmar flexion creases, secondary creases, and wrinkles, but highlights isolated points as edges, and creates false or second edges where only a single line exists. As shown in Figure 43(e), of the four algorithms that were tested, the phase congruency operator responds most accurately to palmar flexion creases.

Given a palmprint image that has been processed using an edge detection algorithm, one method of extracting the major flexion creases, that is, extracting the lines that contain the highest energy values, it to use a threshold. As described in Chapter 2.4, a threshold is a segmentation method, which is typically used prior to binary morphological processing, to convert an intensity image, that is, a greyscale image, to a binary image. However, by defining appropriate threshold parameters, a threshold can be used to segment an image by separating areas of high intensity from areas of low intensity. To this end, a threshold can be used to extract the pixels, and therefore, the lines, with the highest energy values from a palmprint image. Figure 44 shows an example of a palmprint image that has been processed by the author using phase congruency and then separated using two different threshold values.

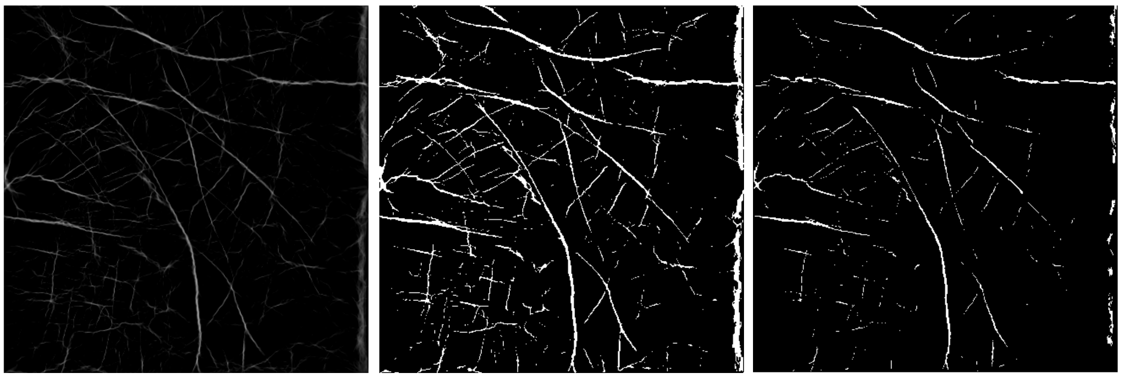

(a) (b) (c)

Figure 44. A palmprint image (The Hong Kong University of Science and Technology, 2003) after applying a) the phase congruency operator, followed by b) a threshold using a value of 0.126 calculated automatically with Otsu's (1979) method, and c) a threshold using a value of 0.25.

As shown in Figure 44(b), when using Otsu's (1979) method to determine the threshold value, that is, a threshold value automatically chosen to minimise the intra-class variance of the black and white pixels, edge like features are extracted. However, a palmprint image contains many edge like features, such as major and minor flexion creases, secondary creases, and wrinkles, a number of which may not be of interest. Therefore, it can be difficult to differentiate major flexion creases from secondary creases and wrinkles, making feature enhancement and segmentation using a simple threshold complicated. This is particularly true for an extended sequence of images, for which a global threshold value cannot be used. For palmprint images, when using a simple threshold for segmentation, parameter selection requires a balance between missing valid edges and noise-induced false edges, as shown in Figure 44(c). For this reason, as the required threshold may vary for each palmprint image, a simple threshold is not adequate for palmar flexion crease identification in this instance.

However, given the same objective, that is, to extract the lines in a palmprint image that contain the highest energy values, a threshold like algorithm can be used. To this end, in contrast to a simple threshold, which is applied to the whole image, each row or column can be processed individually, and the pixel with the highest energy value, that is, level of intensity, in each row or column can be extracted.

(a) (b) (c)

Figure 45. A palmprint image (The Hong Kong University of Science and Technology, 2003) after applying the phase congruency operator and then a) vertical line detection, b) horizontal line detection, and c) horizontal and vertical line detection.

As shown in Figure 45(c), as created from the author's work, when vertical line detection, shown in Figure 45(a), and horizontal line detection, shown in Figure 45(b), are combined, the major palmar flexion creases can be detected while minimising interference from noise and without creating false edges. However, compared to the original palmprint image, as shown in Figure 43(a), a number of major line segments are incomplete. To extract complete palmar flexion creases from a palmprint image, an alternative method of extraction is required.

Using the same theory as above, in which a single pixel from each row or column is extracted, a new palmar flexion crease identification algorithm can be defined. The new algorithm, instead of extracting individual pixels from each row or column, should extract a path of pixels, that is, a sequence of connected pixels, that contains the highest overall energy compared to all other paths. Using this theory, and the definition of internal image seams (Avidan and Shamir, 2007), an automated method for identifying palmar flexion creases in a palmprint image can be defined.

4.2.1. Palmar flexion creases as internal image seams

An internal image seam, as defined by Avidan and Shamir (2007), is an 8-connected path of pixels from the top of an image to the bottom, or from the left of an image to the right, containing one, and only one, pixel in each row or column. Figure 46 shows an example of a horizontal image seam, a vertical image seam, and their structuring element, an 8-connected neighbourhood. A pixel is said to be 8-connected when at least one, and in this case, only one, of its neighbouring pixels to the north, north east, east, south east, south, south west, west, or north west are connected. To be clear, an 8-connected path of pixels, that is, a horizontal or vertical image seam, can only be connected by a single horizontal, vertical, or diagonally connected pixel, as shown in Figure 46(b) and Figure 46(c).

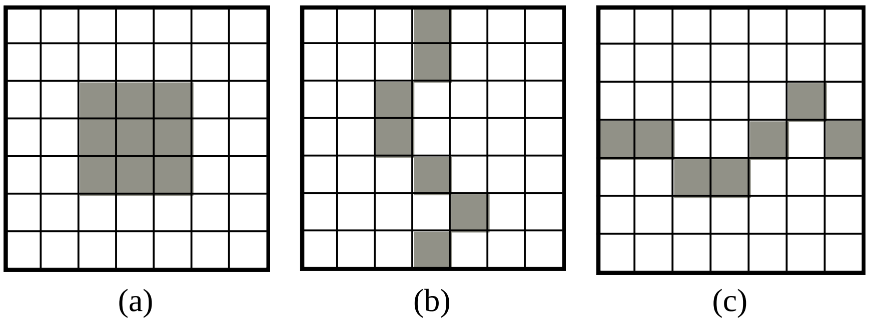


Figure 46. Internal image seams represented as an 8-connected path of pixels: (a) an 8-connected neighbourhood, (b) a vertical internal image seam, and (c) a horizontal internal image seam.

A palmar flexion crease can be recognised as an internal image seam that contains higher than expected energy compared to all possible connected seams. Given an energy function, the strength, and therefore validity, of a palmar flexion crease can be determined by the sum of its image seam energy values. Avidan and Shamir (2007) used internal image seams in content-aware image retargeting to find and remove image seams with the lowest possible intensity. However, using a modified internal image seams algorithm, image seams with the highest intensity, and therefore, the palmar flexion creases, can be effectively identified.

Given an energy function, e, the importance of each pixel in a palmprint image can be identified, and the overall energy, $E(s)$, of a given palmar flexion crease can be calculated by:

$$E(s) = E(I_s) = \sum_{i=0}^{n} e(I(s_i)),$$

and the palmar flexion crease with the highest overall energy, s^*, by:

$$s^* = \max_s E(s) = \max_s \sum_{i=0}^{n} e(I(s_i)),$$

where I is an $n \times m$ image, and s is an image seam.

Furthermore, the maximum energy, M, for all possible connected vertical image seams for each entry, (i, j), can be calculated by traversing the image from the second row to the last row:

$$M(i, j) = e(i, j) + \max(M(i-1, j-1), M(i-1, j), M(i-1, j+1)),$$

and, for all possible connected horizontal image seams, from the second column to the last column:

$$M(i, j) = e(i, j) + \max(M(i-1, j+1), M(i, j+1), M(i+1, j+1)).$$

Using these definitions, any number of major palmar flexion creases, secondary creases, or wrinkles can be individually extracted from a palmprint image. Figure 47(b) shows an example of two image seams, one horizontal and one vertical, which have been used to accurately identify the major palmar flexion creases. For comparison, a palmprint image before internal image seam detection is shown in Figure 47(a).

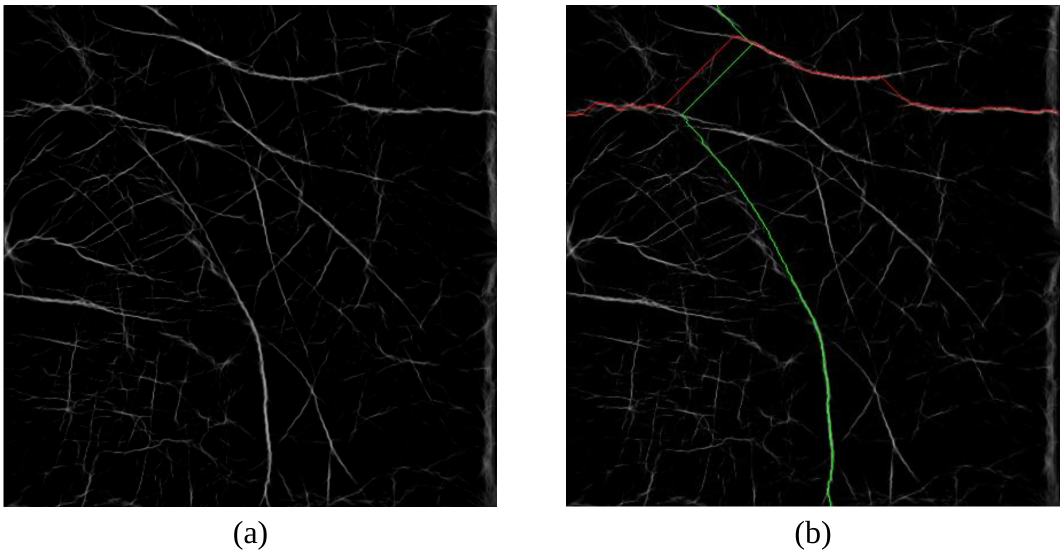


Figure 47. A palmprint image (The Hong Kong University of Science and Technology, 2003) after applying a) the phase congruency operator, and then b) internal image seam detection. For clarity, the horizontal internal image seam is marked in red, while the vertical internal image seam is marked in green.

However, the success of internal image seams, as a method for identifying palmar flexion creases, depends on the effectiveness of the energy function it receives. To this end, Figure 48 shows an example of two internal image seams which do not accurately identify the major palmar flexion creases. In Figure 48(a), a vertical image seam incorrectly identifies a false palmar flexion crease. This is caused by the writer’s palm, that is, the edge of the palm opposite the thumb, where many secondary creases and wrinkles are typically present. Similarly, in Figure 48(b), a horizontal image seam incorrectly identifies line segments from two separate palmar flexion creases, where ideally, two horizontal image seams would be used.

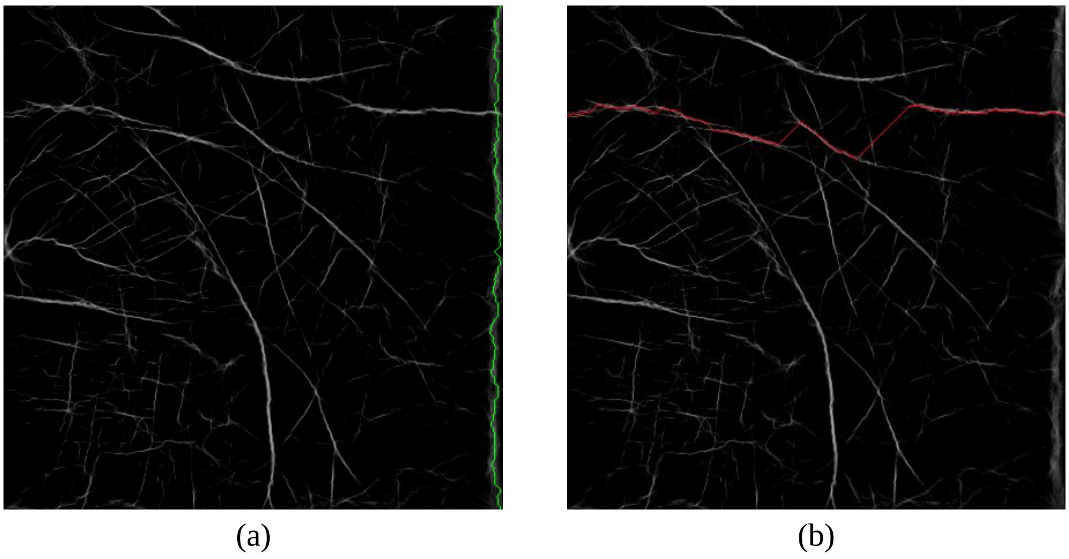


Figure 48. An internal image seam which a) marked in green, follows a false palmar flexion crease, and b) marked in red, identifies two palmar flexion creases.

To address the problems described above, two separate approaches can be used. In the first instance, to prevent the writer's palm from interfering with the extraction process, a specific region of interest, which is focused on the major palmar flexion creases, can be extracted. As described in Chapter 2.2, a common approach to region of interest selection is to extract a 128×128 pixel square from the centre of the palm (Zhang, 2004). However, to extract as much palmar flexion crease information as possible, the size of the region of interest must be maximised. To this end, only a small subset of pixels should be removed from each palmprint image. This can be achieved by removing, or cropping, pixels from the edge of the image. Furthermore, for the method to be invariant to handedness, that is, applicable to both left and right hand palms, the same number of pixels should be removed from the left edge as is removed from the right edge. For example, as shown in Figure 49, for a palmprint image measuring 432×437 pixels at 72 dpi, or 15.24×15.42 cm, 30 pixels, or 1.05 cm, can be removed from the left and right edges to create a large region of interest, while removing the secondary creases and wrinkles of the writer's palm.

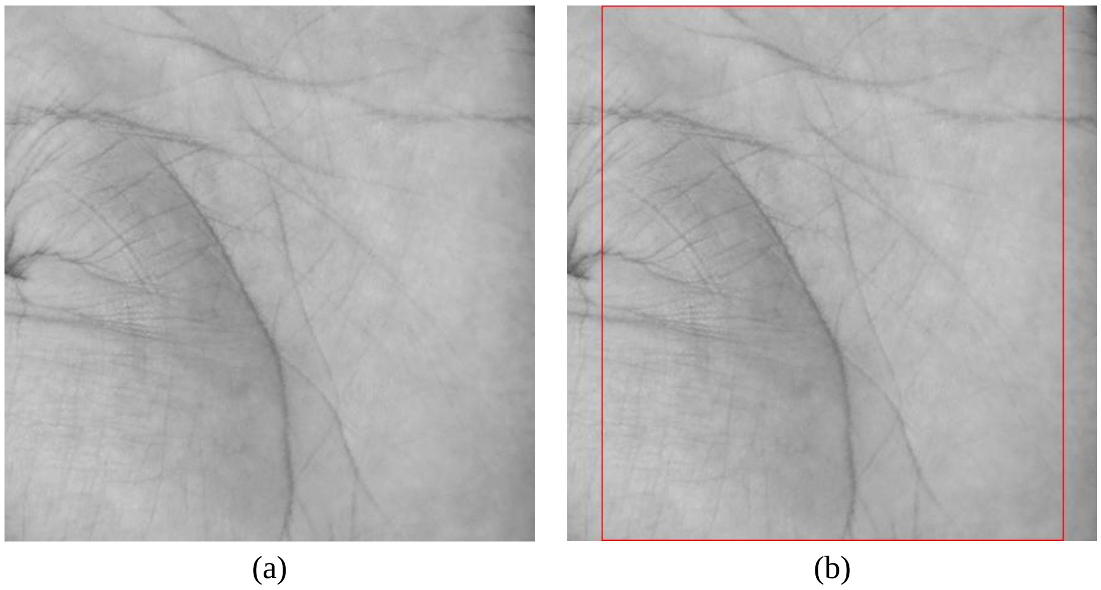


Figure 49. A palmprint image (The Hong Kong University of Science and Technology, 2003) a) before, and b) after region of interest selection.

To address the problem recognised in Figure 48(b), where a single image seam identifies line segments from two separate palmar flexion creases, an alternative image importance measure is required. Given a palmprint image that has been processed using the phase congruency operator, the difference in intensity between the foreground pixels, that is, the palmar flexion creases, and the background pixels is too great. For this reason, when an image seam encounters a weak palmar flexion crease line segment, it fails to identify the palmar flexion crease, and instead identifies the nearest area of high intensity. To this end, a new image importance measure, that can highlight the palmar flexion creases, and remove noise and superficial wrinkles, while retaining features from the background texture, is required. Given these objectives, an effective method for identifying palmar flexion creases, as determined by Wu and Li (1997), that is not typically used for edge detection, but allows feature extraction at multiple scales, is a Laplacian pyramid (Burt and Adelson, 1983). A Laplacian pyramid is a sequence of error images, $L_0, L_1, \ldots, L_N$, which are calculated as the difference between two levels of a Gaussian pyramid. First, an image is convolved with a Gaussian kernel, resulting in low-pass filtered version of the original image, where each pixel contains the local average of the corresponding pixel neighbourhood of the previous image. Then, to calculate the Laplacian, the difference between the original image and the low-pass filtered version is computed. Formally, a single level from a Laplacian pyramid can be defined as:

$$L_l = g_l - g_{l+1},$$

where g is a Gaussian pyramid, and l is the desired pyramid level. Figure 50 shows how a series of filtered images create a stack of images to form the levels of a Laplacian pyramid.

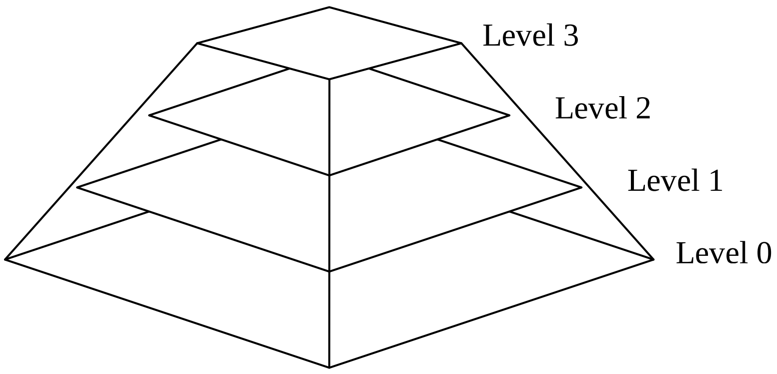


Figure 50. A stack of filtered images form a Laplacian pyramid.

Using the proposed approach, by calculating the local average of each pixel using a Gaussian kernel, line features in a palmprint image can be successively filtered, from superficial wrinkles at level 1 to strong secondary creases at level 3. To this end, Figure 51, generated by the author, shows how a Laplacian pyramid can be used at higher levels to highlight major palmar flexion creases, and at lower levels to highlight secondary palmar flexion creases.

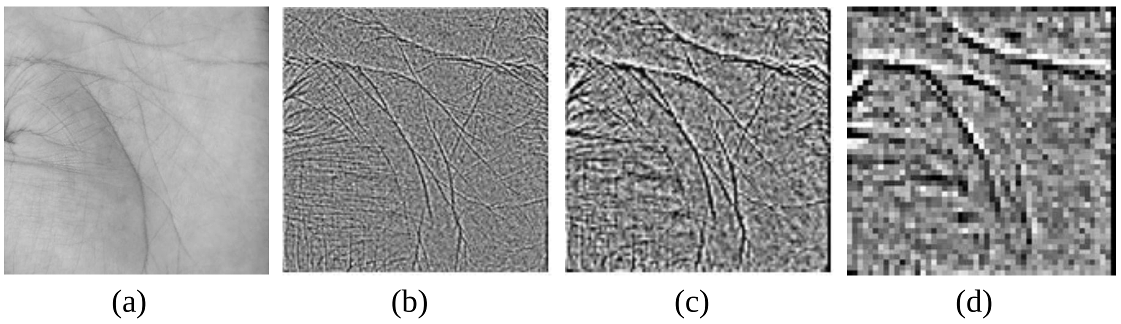


Figure 51. A palmprint image represented as a Laplacian pyramid at b) level 1, c) level 2, and d) level 3.

Furthermore, given the definition of a Laplacian pyramid, in combination with that of internal image seams, palmar flexion creases can be extracted from a palmprint image using the following approach:

1. Given a palmprint region of interest, create an error image L_l, where l is the desired pyramid level.

2. For n iterations, find the optimal vertical palmar flexion crease in L_l, where n is the number of vertical flexion creases to extract.
3. For n iterations, find the optimal horizontal palmar flexion crease in L_l, where n is the number of horizontal flexion creases to extract.

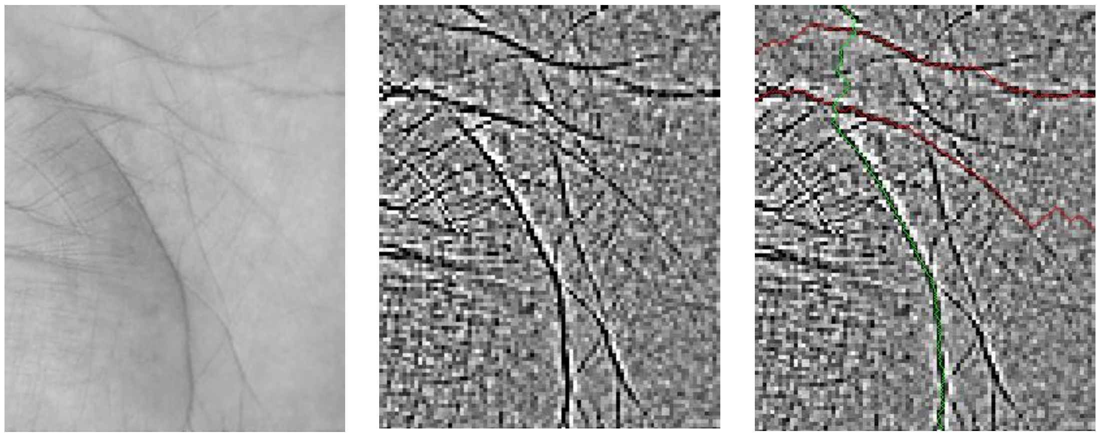


Figure 52. A palmprint image (The Hong Kong University of Science and Technology, 2003) region of interest with b) its Laplacian pyramid representation at level 2, and c) three internal image seams overlaid. For clarity, the horizontal internal image seams are marked in red, and the vertical internal image seam is marked in green.

After this process, the major palmar flexion creases, or secondary creases, are represented as internal image seams and can be used in feature matching. Figure 52 shows an example of a palmprint region of interest, its Laplacian pyramid representation at level 2, and three image seams, two horizontal and one vertical, which are used to identify the major palmar flexion creases.

4.2.2. Palmar flexion crease matching

The aim of palmar flexion crease matching is to calculate the degree of similarity between two or more sets of palmar flexion creases, in order to determine if they belong to the same palm. As described in detail in Chapter 2.3.3, a number of matching algorithms, based on pixel-to-pixel matching, pixel-to-area matching, or straight line matching, have been proposed. However, many such methods, for which the acquisition process often captures each palmprint image with a fixed orientation, rely on each palmprint being relatively aligned, and simple movement beyond a few degrees causes the matching algorithm to perform poorly. This is a concern for palmar flexion crease identification as each image may exhibit varying degrees of rotation or translation. To this end, an approach taken by Huang *et al.* (2008) uses pixel-to-area matching, in which a small area around each palmar flexion crease is compared. Huang *et al.*'s (2008) method produces good results even in cases where rotation and translation are present. Therefore, for palmar flexion crease matching, a similar method, allowing considerations to be made for variations in rotation and translation by calculating whether a point is contained within the boundary space of a palmar flexion crease, is required.

To this end, as described in Chapter 3, a nearest neighbour algorithm can be used. For manual palmprint matching, again, as described in Chapter 3, due to the small number of control points and low dimensionality in each palmar flexion crease configuration, a simple linear nearest neighbour algorithm is sufficient. However, for automated palmprint matching, where the palmar flexion creases contain a greater number of control points, a more efficient nearest neighbour algorithm is required. To this end, a *k*d-tree is a special type of binary space-partitioning tree for organising points in a *k*-dimensional space, in which every node stores an approximately equivalent number of objects. For low dimensions, a *k*d-tree structure can be used for nearest neighbour queries in logarithmic time and linear space, creating an ideal candidate for 2-dimensional nearest neighbour searches (Panigrahy, 2008). Therefore, to compare the flexion creases of two palmprints, A and B, the control points from each palmar flexion crease in A can be added to a 2-dimensional *k*d-tree (Bentley, 1975), to which the control points from each palmar flexion crease in B can be compared using a nearest neighbour algorithm.

To find the nearest neighbour to a point, q, the *k*d-tree can be traversed from the root node downwards, and at each step the direction of q can be determined. This process is repeated until the node cell, c, that contains q is found. As c is bordered by a point, p, the distance, $d(p,q)$, from p to q can be calculated, along with all cells that lies within a distance of $d(p,q)$, to determine the nearest point and its distance. The distance between q and its nearest neighbour can be used to determine if q is within the boundary space of a palmar flexion crease. Any points that are within a certain distance can be considered correct matches. Given this theory, the total number of points and total number of matches can be used to create a match percentage.

The algorithms described in this chapter were implemented on Microsoft Windows XP using Mathworks MATLAB R2007b, and the Python programming language with the NumPy, SciPy, and the Python Imaging Library (PIL) extensions. The algorithms were tested using The Hong Kong University of Science and Technology's (2003) hand image database. The source code for each algorithm is available in appendix A.

4.3. Results and discussion

To evaluate the effectiveness of the algorithm described in Chapter 4.2, 100 palms from The Hong Kong University of Science and Technology's (2003) hand image database were identified using manual and automated palmar flexion crease recognition, then compared. 10 images from each palm, that is, 1000 palmprint images in total, were identified using both the manual method described in Chapter 3, and the automated method described above. Furthermore, prior to identification, as shown in Figure 53(a), each palmprint image was pre-processed using the method described in Chapter 4.2.1, where 30 pixels, or 1.05 cm, is removed from the left and right edges of each image, to extract a specific region of interest. After pre-processing, for manual palmar flexion crease identification, the three major palmar flexion creases, that is, the thenar, the distal transverse, and the proximal transverse, were manually identified using the method described in Chapter 3. As shown in Figure 53(b), the result is a collection of smooth paths, that gently curve through the palmprint image, providing an accurate representation of the palmar flexion creases. Similarly, for automated palmar flexion crease identification, the three major palmar flexion creases were identified using the automated method described in Chapter 4.2.1. However, as a palmprint image may contain wide flexion creases, that is, wider than an internal image seam, the same palmar flexion crease may be identified multiple times by successive image seams. For this reason, three vertical image seams and four horizontal image seams were automatically identified in each palmprint image, allowing the thenar, the distal transverse, and the proximal transverse to be accurately identified, even in the presence of strong secondary creases and wrinkles. However, to avoid multiple comparisons of the same palmar flexion crease, when multiple image seams, or partial image seams, follow the same path, only one image seam is retained. Figure 53(c) shows an example of two palmprint images from different palms with their image seams overlaid.

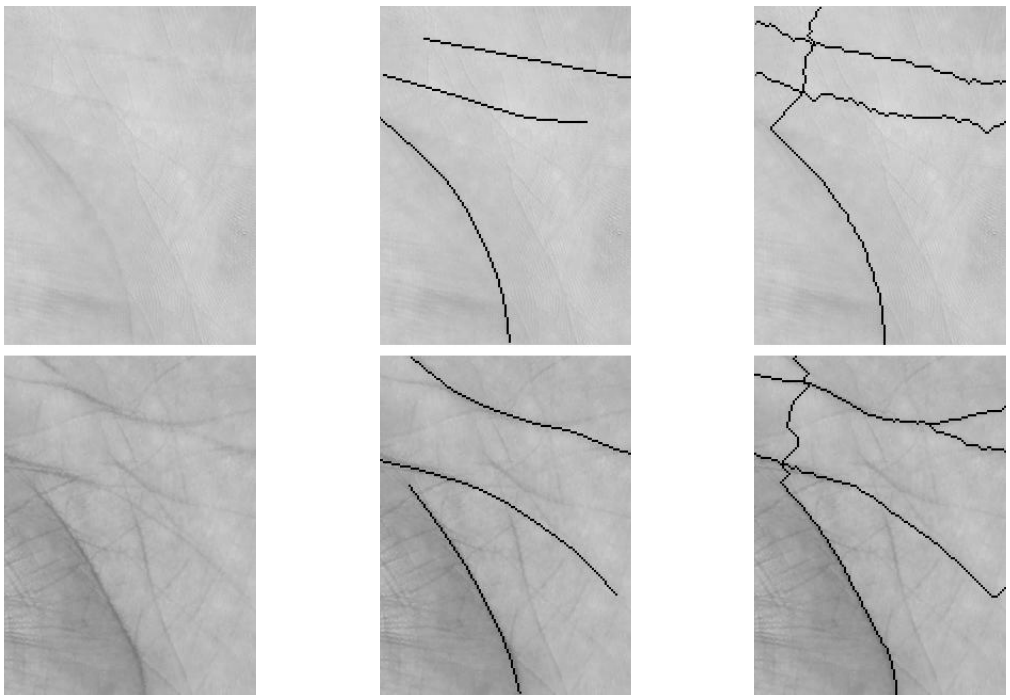


Figure 53. Palmprint images with their flexion creases identified using b) manual and c) automatic methods.

To verify the accuracy of the proposed method, every manually identified set of palmar flexion creases was compared to every automatically identified set of palmar flexion creases using the method described in Chapter 4.2.2. For 100 palms, with 10 palmprint images from each, a total of 1000000 comparisons were recorded in each experiment. For a single palm, 1% of these comparisons were considered genuine, while 99% were considered false. The failure to enrol rate, that is, the number of times the algorithm failed to identify any palmar flexion creases, for all experiments was zero.

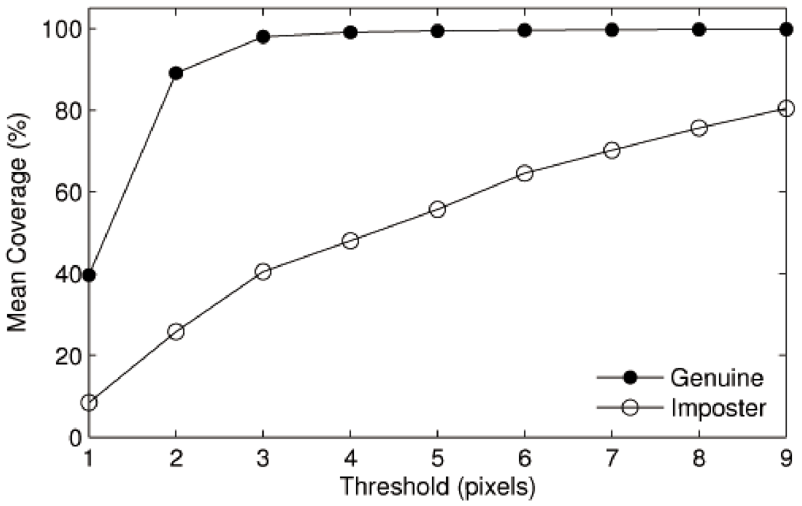


Figure 54. Mean coverage of manually identified flexion creases by automatically identified image seams when the two are overlaid.

Figure 54 shows the mean palmar flexion creases coverage, that is, the amount of overlap between the boundary surrounding manually identified palmar flexion creases and automatically identified palmar flexion creases when the two are overlaid, for correct and incorrect comparisons at various boundary thresholds. For pixel-to-pixel comparisons, that is, for a boundary threshold of 1 pixel, the mean palmar flexion crease coverage is 39.68% for correct matches, and 8.46% for incorrect matches, giving a difference of 31.22%. For pixel-to-boundary matching, that is, at thresholds greater than 1 pixel, the difference between correct and incorrect coverage rises to 63.32%. However, this difference declines as the threshold size increases and false matches become frequent. A compromise between false acceptance coverage and genuine acceptance coverage is required.

The ROC curve of genuine acceptance rates against false acceptance rates for various thresholds is shown in Figure 55. A single comparison is considered genuine if two conditions are met: first, if both features are from the same palm, and second, if the palmar flexion crease coverage is above a given threshold. All other comparisons are considered false. As shown in Figure 55, palmar flexion creases can be identified in online palmprint images with a 100% GAR, and with a 0.0045% FAR, at a 2 or 3 pixel matching boundary. This equates to 10000 correct matches, and 45 incorrect matches, per 1000000 comparisons. Furthermore, at a 2 pixel matching boundary, if the GAR is reduced to 99.9%, the FAR can be improved approximately fourfold to 0.0011%.

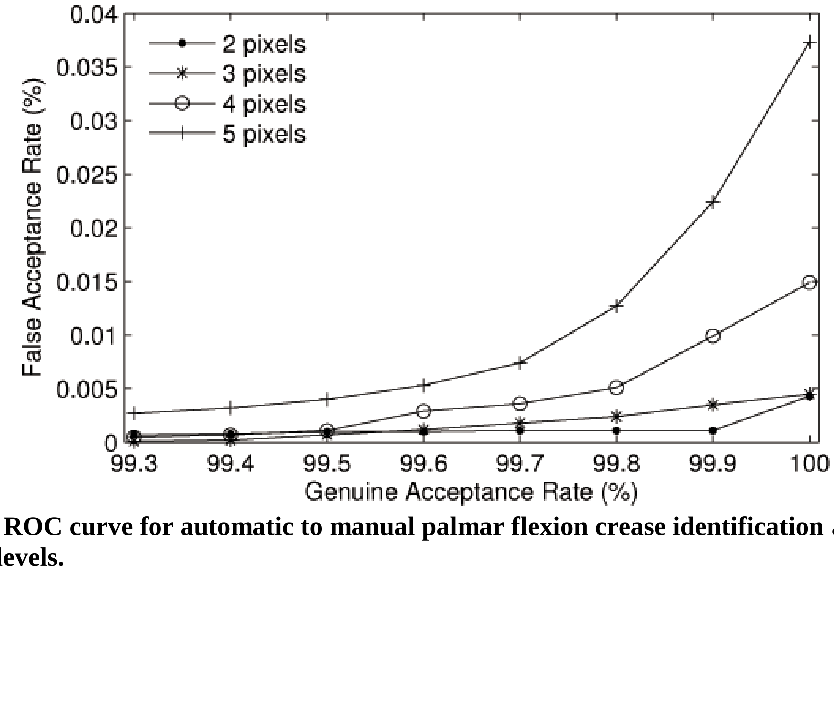


Figure 55. ROC curve for automatic to manual palmar flexion crease identification at different threshold levels.

These results show internal image seams to be an effective method of palmar flexion crease recognition, by confirming that internal image seams are capable of identifying the actual location of palmar flexion creases in online palmprint images. Furthermore, internal image seams provide a number of benefits over existing palmar flexion crease identification methods. Non-uniform contrast and noise are filtered by the energy function prior to flexion crease identification, efficiently removing false edges, superficial secondary creases, and wrinkles. Line direction and wide flexion creases are identified inherently during the image seam extraction process, ensuring the most prominent lines, and therefore the major palmar flexion creases, are identified correctly. Furthermore, an alternative application of automated flexion crease recognition can be found in forensic science, where flexion creases can be identified in partial palmprint images, and in multiple image formats, such as developed latent palmprints recovered from scenes of crime, and inked or Livescan images, as collected by law enforcement agencies (Police IT Organisation, 2005). Consequently, the method presented here has been designed with the potential to be format independent, allowing palmar flexion creases to be compared across multiple fields, and through changing image formats. This is not true for many palmprint identification methods, as they are often specific to one type of image, and use format dependent identification metrics. However, the success of internal image seams as a method for identifying palmar flexion creases, in any image format, depends on the effectiveness of the energy function it receives. The importance of a pixel in an online palmprint image differs from that in an inked palmprint image, and therefore considerations need to be made for each image format.

Internal image seams are an efficient method of palmar flexion crease recognition, that can be implemented with minimal storage requirements. As only the image seams are stored, the reduction in requirements, compared to a complete palmprint image, are substantial. This is particularly important in forensic science systems, where over 10 million records are routinely stored (Scottish Police Services Authority, 2007). Given these advantages, automated flexion crease identification using internal image seams shows promising results as an automated palmar flexion crease recognition system.

4.4. Summary

This chapter presented a method of automated palmar flexion crease identification for online palmprint images. In 1000 online palmprint images from 100 palms, when compared to manually identified palmar flexion creases, experimental results showed that palmar flexion creases can be identified automatically with a 100% genuine acceptance rate and a 0.0045% false acceptance rate. In Chapter 5, palmar flexion creases are automatically extracted and compared in two online palmprint image data sets to determine if palmar flexion creases can be used as an effective method of online palmprint identification.

5. Automated palmprint identification

The results presented in Chapter 4 showed that palmar flexion creases can be automatically extracted from online palmprint images with a 100% GAR, and a 0.0045% FAR, when compared to manually identified palmar flexion creases. However, any method of automated palmprint identification must be capable of comparison between palmprint images, that is, automated to automated comparison. For this reason, this chapter, using the same feature extraction and matching method as in Chapter 4, uses two online palmprint image data sets to determine if automated palmar flexion crease recognition can be used as an effective method of online palmprint identification.

5.1. Introduction

An improved method of automated palmprint identification has the potential to simplify operating procedure, increase identification speed, and reduce database search times, leading to more successful identifications, and improved offender identification speed. To this end, the results presented in Chapter 3 established the feasibility of palmar flexion crease identification, by showing that palmar flexion creases can be used to identify palmprint images with a 99.2% GAR and a 0% FAR, when labelled within 10 pixels, or 3.7mm, of the palmar flexion crease. Similarly, the results presented in Chapter 4 showed that, using a new method of automated palmar flexion crease recognition, palmar flexion creases can be automatically extracted from online palmprint images with a 100% GAR and a 0.0045% FAR, when compared to manually identified palmar flexion creases. Given these results, the basis for an improved method of automated palmprint identification can be defined. However, any method of automated palmprint identification must be capable of comparison between palmprint images, that is, automated to automated comparison. For this reason, this chapter, using the same feature extraction and matching method as in in Chapter 4, uses two online palmprint image data sets to determine if automated palmar flexion crease recognition can be used as an effective method of online palmprint identification.

5.2. Materials and methods

To determine if automated palmar flexion crease recognition can be used as an effective method of online palmprint identification, an online palmprint image data set needs to be created. To this end, The Hong Kong University of Science and Technology's (2003) hand image database provides 1000 online palmprint images, measuring a mean of 380×380 pixels at 72 pixels per inch, from the left hand palm of 100 people, captured during a single session. Figure 56(a) shows an example of a palmprint image from The Hong Kong University of Science and Technology's (2003) hand image database.

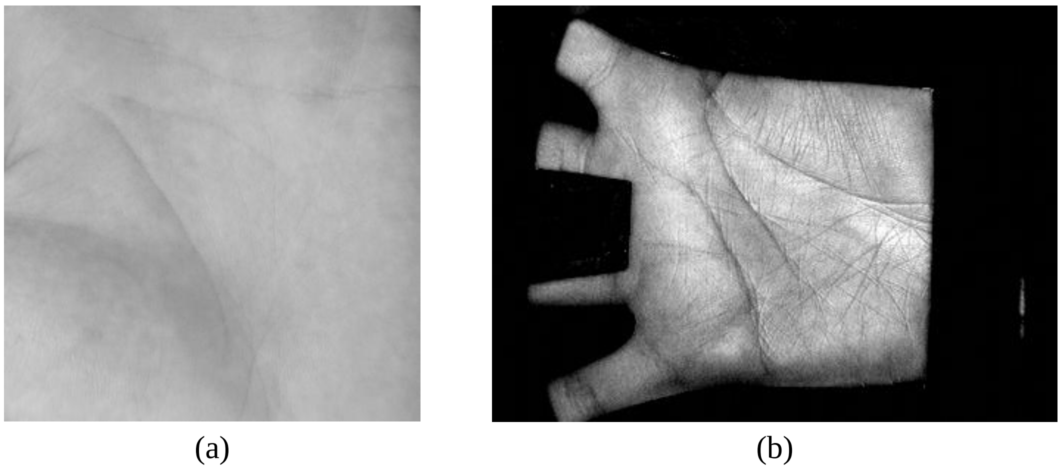


Figure 56. An online palmprint image from a) The Hong Kong University of Science and Technology's (2003) hand image database, and b) The Hong Kong Polytechnic University's (2003) palmprint image database.

However, as the database contains palmprint images from only 100 people, the variation in palmar flexion crease configurations is low, and inconsistencies in image capture quality, orientation, and palmprint coverage, as found in real world data sets, are few. For this reason, The Hong Kong University of Science and Technology's (2003) hand image database cannot be used to provide a suitable comparison against other palmar flexion crease identification studies, in which larger, more variable online palmprint image data sets are used (Wu *et al.*, 2004b; Wu *et al.*, 2006; Huang *et al.*, 2008; Jai *et al.*, 2008). An alternative online palmprint image data set is required.

To this end, The Hong Kong Polytechnic University's (2003) palmprint image database is an online palmprint image database containing 7752 grayscale palmprint images corresponding to 386 different palms. Using a real time palmprint acquisition device (Zhang *et al.*, 2003), approximately 20 online palmprint images, measuring

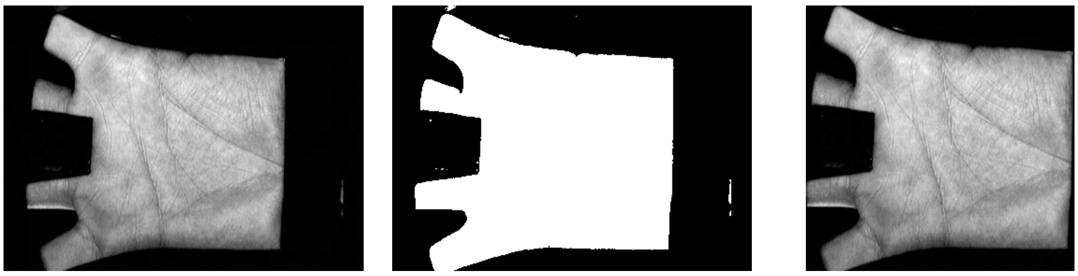
384×284 pixels at 72 pixels per inch, were collected over two sessions from each of the 386 palms. The mean interval between the first and second session was two months, and approximately 10 palmprint images were captured per palm in each session. Figure 56(b) shows an example of an online palmprint image from The Hong Kong Polytechnic University's (2003) palmprint image database.

The Hong Kong Polytechnic University's (2003) palmprint image database provides a suitable data set for determining the efficacy of palmprint identification using palmar flexion creases, and provides a platform for establishing a comparison with existing palmar flexion crease identification methods, many of which use the same data set (Wu *et al.*, 2004b; Wu *et al.*, 2006; Huang *et al.*, 2008; Jai *et al.*, 2008). However, unlike The Hong Kong University of Science and Technology's (2003) hand image database, some pre-processing of the data set is required. As shown in Figure 56(b), The Hong Kong Polytechnic University's (2003) palmprint image database contains extraneous data from the image acquisition process, such as the fingers, background, and outline of the palm, that is not required for feature extraction and matching. To remove this data, and extract the region of interest, a pre-processing method must be defined.

The aim of pre-processing is to place each image under the same coordinate system, so that the correct area of each palmprint can be extracted for feature extraction and matching (Zhang and Shu, 1999). To this end, a number of pre-processing methods, based on the type of acquisition device, format of the palmprint image, and desired region of interest, have been proposed (Han, 2003; Zhang *et al.*, 2003; Han, (2004); Poon *et al.*, (2004); Kong *et al.*, (2009). However, as described in Chapter 2, in each of these methods, the same basic algorithm is employed:

1. The palmprint image is converted to a binary image using a given threshold.
2. The contour of the hand and/or fingers are extracted from the binary image, a number of key points are detected, and a coordinate system is defined.
3. The central part sub-image is extracted.

To this end, given an image from The Hong Kong Polytechnic University's (2003) palmprint image database, and using Otsu's (1979) method to automatically determine a threshold value that minimises the intra-class variance of the black and white pixels, the palmprint image can be converted to a binary image. Then, by identifying each component, that is, each 8-connected area in the image where the pixel value is set to logical 1, and calculating the number of pixels in each component, the largest object, and therefore, the palmprint image, can be isolated. Figure 57 shows stage one of the palmprint pre-processing method.



(a) (b) (c)

Figure 57. a) A palmprint image (The Hong Kong Polytechnic University, 2003), b) converted to a binary image, and c) isolated from the background.

In stage two, to define a coordinate system, the contour of the hand must be determined. Given a binary image, the contour of a component can be calculated by traversing the Moore neighbourhood, that is, the set of pixels which share an edge or vertex with a given pixel (Gonzalez and Woods, 2008). Figure 58 shows the Moore neighbourhood of a pixel, P.

P1	P2	P3
P8	P	P4
P7	P6	P5

Figure 58. The Moore neighbourhood (P1, P2, …, P8) of a pixel, P.

Using the Moore neighbourhood, when a starting pixel, P, in a component of a binary image, C, is set to logical 1, the contour of C can be determined by examining each pixel in a clockwise direction, advancing until a new pixel that is set to logical 1 is encountered. The algorithm continues pixel by pixel until it is terminated by Jacob's stopping criteria, that is, when the starting pixel has been visited n times, where n is at least 2, or when the starting pixel is visited a second time in the same manner in which it was entered (Gonzalez and Woods, 2008). As shown in Figure 59(a), given the above theory, and a binary image of an online palmprint image, the contour of the hand can be determined.

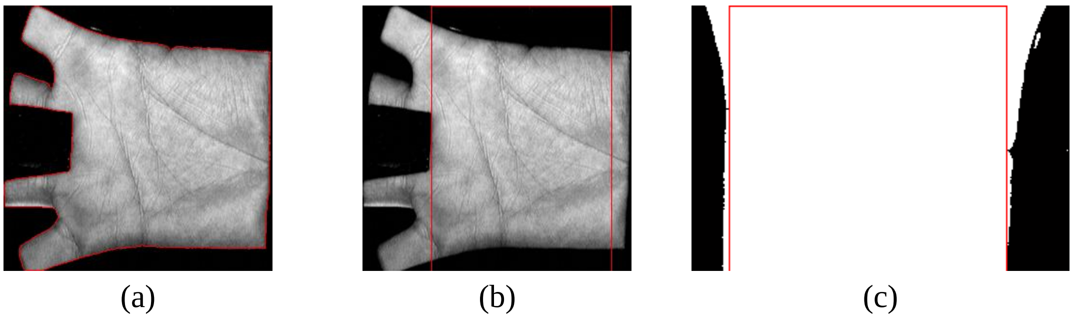


Figure 59. A palmprint image (The Hong Kong Polytechnic University, 2003) with a) the contour of the hand outlined, b) the central part sub-image outlined, and c) the region of interest outlined.

Following this, using the contour of the hand, the local minima and maxima can be calculated, and the spaces in between the fingers can be determined. Using these spaces, a coordinate system can be defined, and, as shown in Figure 59(b), the central part sub-image of the hand can be extracted. Finally, as shown in Figure 59(c), the central part sub-image can be rotated, and the largest possible rectangle within the palmprint component can be determined and extracted. Figure 60 shows the largest possible rectangle, and therefore, the region of interest, from an pre-processed online palmprint image from The Hong Kong Polytechnic University's (2003) palmprint image database.

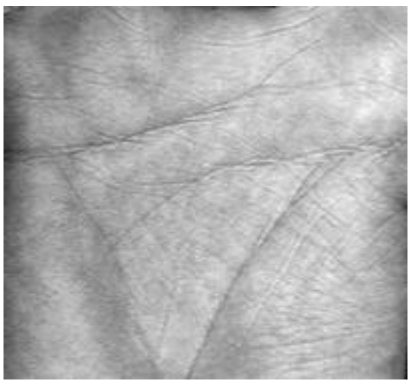


Figure 60. The region of interest from an online palmprint image (The Hong Kong Polytechnic University, 2003).

After pre-processing, the online palmprint image is ready for feature extraction and matching. In this chapter, the same feature extraction and matching methods as described in Chapter 4 are used, in which internal image seams (Avidan and Shamir, 2007) extract the palmar flexion creases, and a nearest neighbour algorithm defines a matching score for comparison.

The algorithms described in this chapter were implemented on Microsoft Windows 7 using Mathworks MATLAB R2007b. The algorithms were tested using The Hong

Kong Polytechnic University's (2003) palmprint image database. The source code for each algorithm is available in appendix A.

5.3. Results and discussion

Given the pre-processing method described in Chapter 5.2, and The Hong Kong Polytechnic University's (2003) palmprint image database, an online palmprint image data set, that can be used to determine if automated palmar flexion crease recognition is an effective method of online palmprint identification, can be created. However, in order to determine the effects of automated palmar flexion crease identification in databases of different sizes, and to compare the results for automated palmprint identification with those presented in Chapter 3, a comparison must first be performed using The Hong Kong University of Science and Technology's (2003) hand image database. Therefore, in this Chapter, palmar flexion creases from two online palmprint image databases are extracted and compared: first, from The Hong Kong University of Science and Technology's (2003) hand image database, and second, from The Hong Kong Polytechnic University's (2003) palmprint image database.

To compare the results for automated palmprint identification to those presented in Chapter 3 for manual palmprint identification, palmar flexion creases from 1000 online palmprint images, that is, 10 images each from 100 different left hand palms, from The Hong Kong University of Science and Technology's (2003) hand image database were pre-processed, extracted, then compared. Each palmprint image, measuring a mean of 380×380 pixels at 72 pixels per inch, was pre-processed using the method described in Chapter 4, where 30 pixels, or 1.05 cm, is removed from the left and right edges to extract a specific region of interest. Then, the three major palmar flexion creases, that is, the thenar crease, the distal transverse crease, and the proximal transverse crease, were automatically extracted from each palmprint image, again, using the method described in Chapter 4. Furthermore, after feature extraction, each set of palmar flexion creases were compared, using the nearest neighbour algorithm described in Chapter 4, and the GAR and FAR were calculated.

However, as a palmprint image may contain wide flexion creases, that is, wider than an internal image seam, the same palmar flexion crease may be identified multiple times by successive image seams. For this reason, three vertical image seams and four horizontal image seams were automatically identified in each palmprint image, allowing the thenar, the distal transverse, and the proximal transverse to be accurately identified, even in the presence of strong secondary creases and wrinkles. Furthermore, to avoid multiple comparisons of the same palmar flexion crease, when multiple image seams, or partial image seams, follow the same path, only one image seam is retained.

Figure 61 shows the receiver operating characteristic curve for automated palmar flexion crease identification for online palmprint images from The Hong Kong University of Science and Technology's (2003) hand image database.

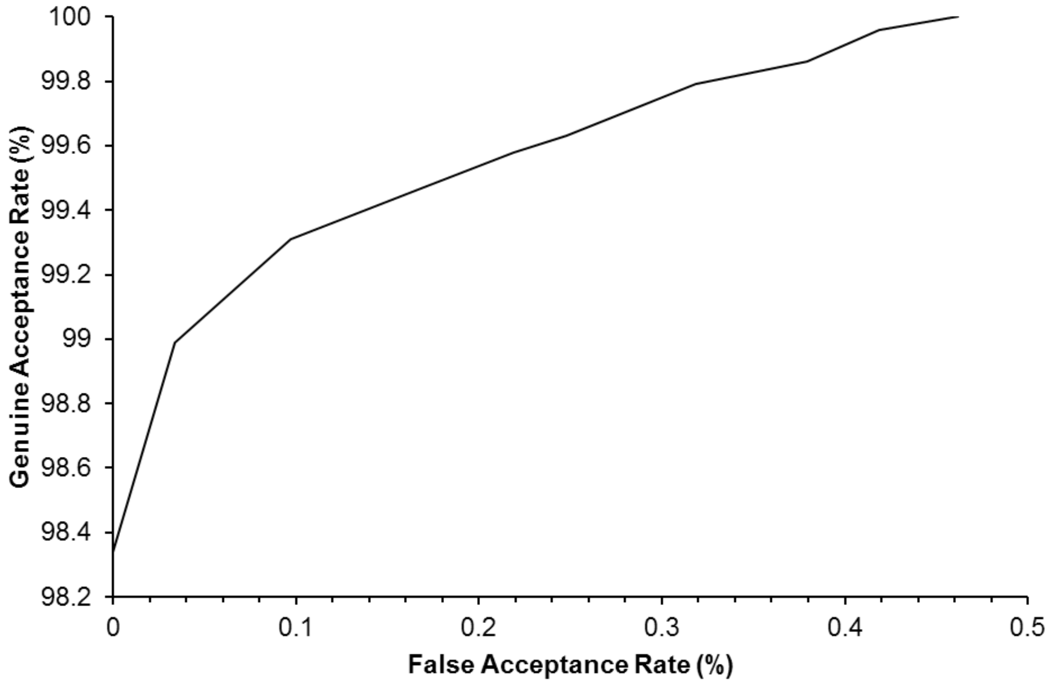


Figure 61. Receiver operating characteristics curve for automated palmar flexion crease identification for online palmprint images from The Hong Kong University of Science and Technology's (2003) hand image database.

The results presented in Figure 61, with a GAR of 98.3% and a FAR of 0%, show the automated palmprint identification method presented here to be comparable with the results shown in Chapter 3 for manual palmprint identification, when the palmar flexion creases are identified manually within 10–12 pixels, or 3.5–4.2 mm, of the actual flexion creases. Thus, the automated palmprint identification method achieves the same results as manual palmprint identification, when the palmar flexion creases are labelled within 10–12 pixels, or 3.5–4.2 mm. Furthermore, with an EER of 0.3%, these results are also comparable with other automated palmprint identification methods (Kong *et al.*, 2009), in particular, those presented by Wu *et al.* (2004b), Liu and Zhang (2005), Wu *et al*. (2006), Huang *et al.* (2008), and Jia *eta al.* (2008), who also used the palmar flexion creases as a metric for their identification methods. However, as the results presented here are for a small sample, low variation palmprint database (The Hong Kong University of Science and Technology, 2003), the FAR can be expected to increase, and the GAR decrease, when the sample size increases. Furthermore, various factors can affect the quality of a palmprint image, such as bruising, scars or cuts, and worn friction ridge skin. While image defects, rotation, translation, and non-uniform contrast are present, they are relatively low compared to a real world data set, where there are often inconsistencies in image capture quality, orientation, and palmprint coverage (Ashbaugh, 1999a). For this reason, The Hong Kong University of Science and Technology's (2003) hand image database does not provide a suitable platform for comparison, and, therefore, analysis using The Hong Kong Polytechnic University's (2003) palmprint image database is required.

Given The Hong Kong Polytechnic University's (2003) palmprint image database, and the automated palmar flexion crease extraction and matching methods described above, palmar flexion creases from the online palmprint image data set can be automatically extracted and compared. However, to provide an accurate comparison with other automated palmprint identification methods, the same method of analysis should be adopted for each algorithm in the comparison. To this end, the most appropriate method of analysis is to compare each algorithm against the same data set under the same test conditions. Therefore, for this approach, each algorithm is required. Table 5 lists the automated palmar flexion crease identification algorithms suitable for comparison.

Table 5. A summary of automated palmprint identification algorithms suitable for comparison.

Method	Title
Kung *et al.* (1995)	A neural network approach to face/palm recognition
Rodrigues *et al.* (1996)	Biometric identification by dermatoglyphics
Boles and Chu (1996)	Personal identification using images of the human palm
Wu *et al.* (2002a)	Fuzzy directional element energy feature (FDEEF) based palmprint identification
Wu *et al.* (2004a)	A novel approach of palm-line extraction
Wu *et al.* (2004b)	Palmprint recognition using directional line energy feature
Liu and Zhang (2005)	Palm-line detection
Li and Leung (2006)	Hierarchical identification of palmprint using line-based Hough transform
Wu *et al.* (2006)	Palm line extraction and matching for personal authentication
Huang *et al.* (2008)	Palmprint verification based on principal lines
Jia *et al.* (2008)	Palmprint verification based on robust line orientation code

To obtain each algorithm listed in Table 5, the corresponding author was determined, and a communication was sent, requesting access to the algorithm. Unfortunately, in all cases, the request was not returned, or the algorithm was no longer available. Appendix B lists the requests and responses from each author.

Given these results, an alternative method of analysis is required. Therefore, given the same data set, that is The Hong Kong Polytechnic University's (2003) palmprint image database, as in Jia *et al.* (2008), Huang *et al*. (2008), Wu *et al.* (2006), and Lui and Zhang (2005), it was decided that the same method of analysis as presented in these should be used. To this end, in the first instance, each online palmprint image was pre-processed using the method described in Chapter 5.2, then, 3 palmprint images from each of the 386 palms were randomly chosen as training set to be used as a template for matching. A further 10 images were then chosen, again, randomly, as a test set, except in one instance, where only 8 images were available. The training set contained 1158 online palmprint images, while the test set contained 3858 online palmprint images from all 386 palms in The Hong Kong Polytechnic University's (2003) palmprint image database. Furthermore, there was no overlap between images in the training set and the test set, in that no single image appeared in both sets. Finally, the palmar flexion creases were extracted from all images in both sets, and those from the test set were compared to all of those from the training set. The total number of comparisons over all images was 4467564, of which 11574 were considered correct, that is, from the same palm, and 4455990 were considered incorrect, that is, not from the same palm. Furthermore, as each image in the test set correctly matches three images in the training set, a comparison was considered correct if the following conditions were met:

- if the palmprint image from the test set was from any of the three palmprint images from the same palm in the training set,

if the matching score was above a given threshold.

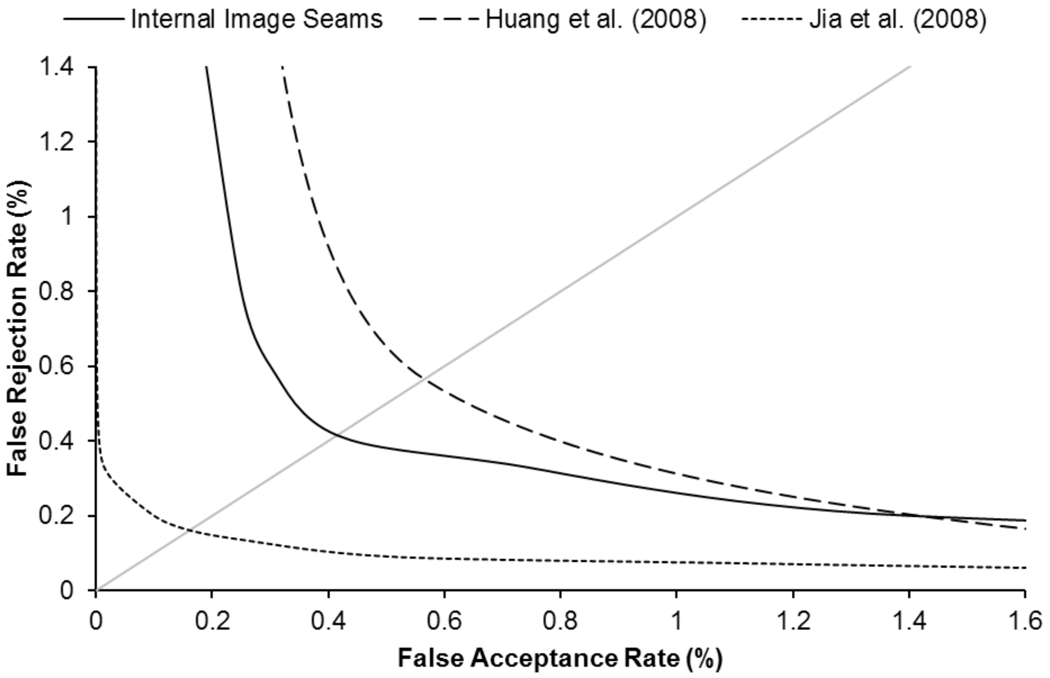


Figure 62 shows the receiver operating characteristic curve for automated palmar flexion crease identification in comparison with that of Huang *et al.* (2008) and Jai *et al.* (2008) for online palmprint images from The Hong Kong Polytechnic University's (2003) palmprint image database.

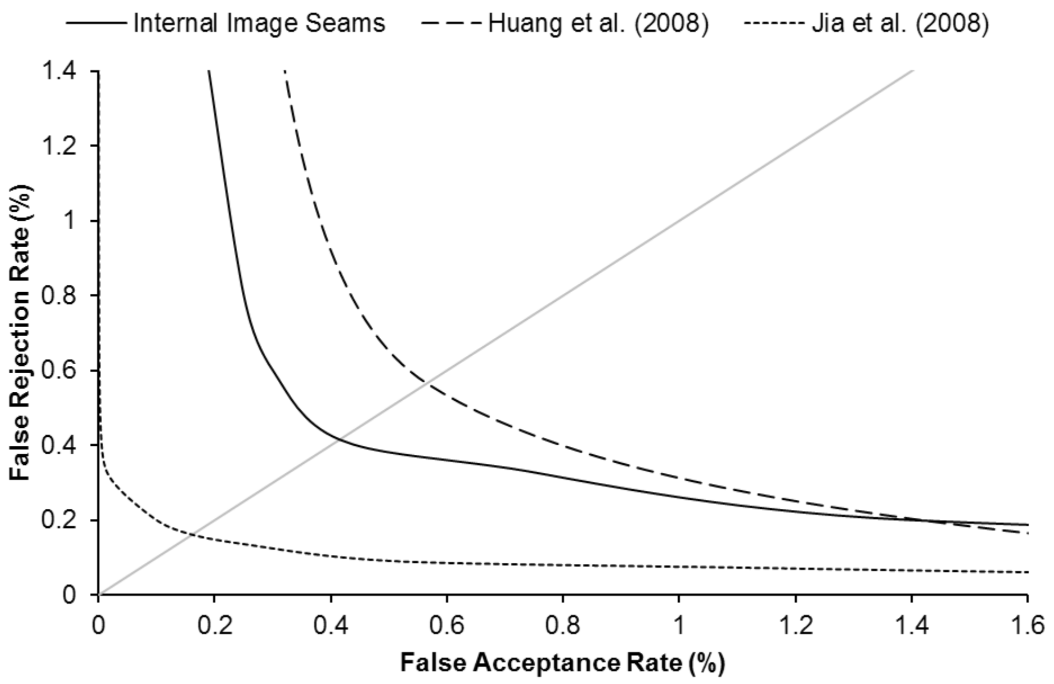


Figure 62. Receiver operating characteristic curve for automated palmar flexion crease identification for online palmprint images from The Hong Kong Polytechnic University's (2003) palmprint image database.

As shown in Figure 62, the EER, that is, the rate at which the FAR and FRR are equal, of 0.415% improves on that presented by Huang *et al.* (2003). However, it does not improve on the 0.16% EER of Jia *et al.* (2008), who used a similar approach to Huang *et al.* (2003), but with an improved matching method. To complete the comparison, Table 6 shows a summary of palmar flexion creases extraction and matching methods and their corresponding EER.

Table 6. A summary of palmar flexion crease extraction method results.

Method	Features	Images	Palms	EER (%)
Kung *et al.* (1995)	Major creases Secondary creases Wrinkles	96	32	[a]
Rodrigues and Silva (1996)	Major creases	15	[a]	[a]
Boles and Chu (1997)	Major creases Secondary creases	10	3	[a]
Zhang and Shu (1999)	Major creases Secondary creases	200	20	[a]
Wu *et al.* (2002a)	Major creases Secondary creases	450	50	[a]
Wu *et al.* (2004a)	Major creases Secondary creases Wrinkles	[a]	[a]	[a]
Wu *et al.* (2004b)	Major creases	3200	320	2.08

Method	Features	Images	Palms	EER (%)
Liu and Zhang (2005)	Major creases	600	100	1.00
	Secondary creases Wrinkles	7752	386	0.4
Li and Leung (2006)	Major creases	600	100	[a]
Wu *et al.* (2006)	Major creases	7605	[a]	0.4
	Secondary creases	7752	368	0.44
Huang *et al.* (2008)	Major creases	[a]	100	0.49
		7752	368	0.565
Jia *et al.* (2008)	Major creases	7752	368	0.16
Internal image seams	Major creases	7752	368	0.415

[a] Data not available.

Rodrigues and Silva (1996) used a Sobel filter with morphological thinning to identify palmar flexion creases in 15 palmprint images, and obtained a variance of 30.48 between palmprints from different persons, and of 0.4751 between those from the same person. Similarly, Wu *et al.* (2004a) used multiple morphological operations with directional structuring elements to effectively extract major palmar flexion creases, secondary creases, and wrinkles. However, with these approaches, false edges created by noise and non-uniform contrast, or strong secondary creases and wrinkles, can interfere with the extraction process, making explicit major palmar flexion crease identification difficult. Furthermore, without a two-stage process, such as that described in Wu *et al.* (2004a), missing or broken flexion creases are easily introduced.

Wu *et al.* (2002a) and Wu *et al.* (2004b) used fuzzy directional element energy features and directional line energy features to obtain identification rates of 97.2% and 97.5%, respectively, the latter with a 2.08% EER. Conversely, Liu and Zhang (2005) presented a line detector, using an isotropic non-linear filter, with which directional elements were not detected, and achieved a 1.0% EER. However, as reported in Huang *et al.* (2008) line direction is an important feature that can be used to distinguish major palmar flexion creases from secondary creases and wrinkles. Furthermore, Liu and Zhang's (2005) method often incorrectly extracts small dark patches as secondary creases, and may fail when the palmar flexion creases are not clearly visible. Wu *et al.* (2006) identified palmar flexion creases as roof edges, and extracted them using directed line detectors, to obtain a 0.4% EER. However, palmprint images containing wide flexion creases proved difficult to extract, resulting in missing or broken lines. Jia *et al.* (2008) remedied this by using a modified finite Radon transform to achieve a 0.16% EER. Also using a modified finite Radon transform, Huang *et al.* (2008) used two databases, I and II, to establish the efficacy of palmar flexion crease identification in data sets of different sizes. Database I contained palmprints from 100 palms, while database II contained palmprints from 386 palms. A 0.49% EER in database I and a 0.565% EER in database II was obtained. From these results, Huang *et al.* (2008) concluded that as the size of the data set increases, a larger percentage of palmprints will contain major flexion creases of a similar configuration, and false identification will increase. Consequently, this is true of the results presented here. As expected, the EER for comparisons using 100 palms from The Hong University of Science and Technology's (2003) hand image database increased 0.115% from 0.3% to 0.415% when 386 palms from The Hong Kong Polytechnic University's (2003) palmprint image database were compared.

However, despite this increase, as shown in Table 6, the EER of the method described here improves on that presented by Wu *et al.* (2004b), and is comparable with those shown by Huang *et al.* (2008), Wu *et al.* (2006), and Lui and Zhang (2005). The results show that palmar flexion crease recognition using internal image seams can be used as an effective method of online palmprint identification. That said, the 0.16% EER presented by Jai *et al.* (2008) suggests that an improved method can be designed. As shown by the receiver operating characteristic curve in Figure 62, the method presented here is less sensitive to changes in matching threshold, that is, the FAR and FRR do not improve as well as expected with changes in the matching threshold, which suggests that the matching method is less robust than other methods. However, as explained in Chapter 4, internal image seams provide a number of benefits over existing palmar flexion crease identification methods. Non-uniform contrast and noise are filtered by the energy function prior to flexion crease identification, efficiently removing false edges, superficial secondary creases, and wrinkles. Furthermore, line direction and wide flexion creases are identified inherently during the image seam extraction process, ensuring the most prominent lines, and therefore the major palmar flexion creases, are correctly identified.

5.4. Summary

This chapter, using the same feature extraction and matching method as in Chapter 4, presented results for automated palmar flexion crease identification in two online palmprint image data sets. The 0.3% EER in a database of 100 palms from The Hong Kong University of Science and Technology's (2003) hand image database, and 0.415% in a database of 386 palms from The Hong Kong Polytechnic University's (2003) palmprint image database, suggests that automated palmar flexion crease recognition using internal image seams can be used as an effective method of online palmprint identification. In Chapter 6, a summary and evaluation of the results and discussions from Chapters 3, 4, and 5 are presented, and an overview of the development of palmprint identification using palmar flexion creases, and the original contributions to knowledge in this thesis, are discussed.

6. Conclusions

As an aid to the reader, this chapter restates the research question, gives a brief overview of the palmprint identification methods presented in this thesis, and presents a summary of the experimental results collected using those methods. Furthermore, a discussion of the results, and the limitations of the proposed methods, as well as some possible solutions, are presented.

6.1. Summary

The aim of this thesis was to investigate the following research question: to what extent can automated palmar flexion crease recognition be used to identify online palmprint images?

To this end, in the first instance, that is, in Chapter 3, a new method of manual palmprint identification was presented, in which palmar flexion creases were identified using cardinal splines, subject to a variety of alterations to mimic the effects of rotation, displacement, and degradation, and compared using generalised Procrustes analysis and a sum of distances metric. The results for manual palmprint identification showed that palmar flexion creases from 100 palms, each modified 10 times, when labelled within 10 pixels, or 3.5 mm of the flexion crease, can be identified with a 99.2% genuine acceptance rate and a 0% false acceptance rate. While little work has been reported on palmprint identification for forensic applications, these results are comparable with hand-based biometric systems, such as hand geometry (Rahman *et al.*, 2008) and the natural layout of the hands (Adan *et al.*, 2008).

Furthermore, the results for manual palmprint identification show that current palmprint identification processes may be improved using new identification metrics, such as palmar flexion creases, that leverage the beneficial identification qualities of palmprint images. Palmar flexion crease matching can improve palmprint identification through integration with existing systems, and through dedicated palmprint identification applications. Furthermore, palmar marks, in which minutiae are not present, such as those that are smudged or distorted, can be identified, and minutiae-based matching can be improved through hierarchical identification, in which palmar flexion creases are used to filter potential matches. However, the results presented in this thesis, that is, for manual palmprint identification, are for a small sample, low variation, palmprint database (The Hong Kong University of Science and Technology, 2003). In all cases, each flexion crease could be easily identified, and while image defects are present, they are relatively low compared to a real world data set. Through imperfect user input, caused by image quality inconsistencies or partial marks from scenes of crime, incorrect matches may be introduced, for which considerations must be made. Furthermore, when a complete palmprint image is presented, sufficient minutiae will normally be available to achieve identification. To this end, the manual palmprint identification method presented in this thesis is not intended to be a replacement for palmprint identification using traditional methods. The value of palmar flexion crease identification comes from their use in partial palmprint images or as a fast pre-filter, where palmar flexion creases may be used to complement existing identification methods.

Given the importance of automated identification in forensic science and biometrics, any new palmprint identification metric must be capable of secure, efficient, and cost effective automation. To this end, in Chapter 4, a method of automated flexion crease recognition, that can be used to automatically identify palmar flexion creases in online palmprint images, was presented. Using a modified internal image seams algorithm, a palmar flexion crease can be identified as an optimal, 8-connected path of pixels in a palmprint image, from top to bottom, or left to right, that contains higher than expected energy compared to all possible connected seams. When combined with an appropriate energy function, the proposed algorithm can be used to identify major palmar flexion creases, or secondary creases, in online palmprint images. The experimental results for automated palmar flexion crease recognition showed that in 1000 palmprint images from 100 palms, when compared to manually identified palmar flexion creases, a 100% genuine acceptance rate can be achieved, with a false acceptance rate as low as 0.0045%. These results confirm that automated palmar flexion crease recognition using internal image seams is capable of identifying the actual location of palmar flexion creases in online palmprint images. Furthermore, internal image seams, as a method of palmar flexion crease identification, provide a number of benefits over existing automated flexion crease identification methods. Non-uniform contrast and noise are filtered prior to flexion crease identification, efficiently removing false edges, superficial secondary creases, and wrinkles. Line direction and wide palmar flexion creases are identified inherently during the internal image seam extraction process, ensuring the most prominent lines, and therefore, the palmar flexion creases, are identified correctly. Internal image seams are an efficient method of palmar flexion crease identification, that can be implemented with minimal storage requirements. As only the internal image seams are stored, the reduction in requirements compared to storing a complete palmprint image are substantial. This is particularly important in forensic science systems, where over 10 million records are routinely stored (Scottish Police Services Authority, 2007).

Finally, to determine if automated palmar flexion crease recognition can be used as an effective method of online palmprint identification, and to compare the results for automated palmar flexion crease identification to those presented in Chapter 3 for manual palmprint identification, Chapter 5 presented results for automated palmprint identification in two online palmprint image data sets. First, using the feature extraction and matching method as described in Chapter 4, the palmar flexion creases from 1000 online palmprint images from 100 palms (The Hong Kong University of Science and Technology, 2003) were extracted and compared, and an equal error rate of 0.3% was achieved. Then, using the same feature extraction and matching method, the palmar flexion creases from 7752 online palmprint images from 386 palms (The Hong Kong Polytechnic University, 2003) were extracted and compared, to achieve an equal error rate of 0.415%. These results are comparable with existing palmar flexion crease identification methods (Wu *et al.*, 2004b; Liu and Zhang, 2005; Wu *et al.*, 2006; Huang *et al.*, 2008; Jia *eta al.*, 2008), and show that palmar flexion crease recognition using internal image seams can be used as an effective method of online palmprint identification.

Given the research question defined above, the two methods of manual and automated palmar flexion crease recognition presented in this thesis show that palmar flexion creases can be used to identify online palmprint images with a genuine acceptance rate and false acceptance rate comparable with other palm- and hand-based identification methods (Kong *et al.*, 2009; Rahman *et al.*, 2008; Adan *et al.*, 2008). However, because of the experimental nature of the work presented in this thesis, and the limited sample size of the given data sets (The Hong Kong

Polytechnic University (2003); The Hong Kong University of Science and Technology, 2003), the data must be interpreted with caution. Given the observance of an increasing equal error rate with changes in the sample size, the results may not be transferrable to large scale data sets, such as those used in forensic science. That said, in forensic science, when a complete palmprint image is presented, sufficient detail will normally be available to achieve identification using established methods. To this end, the work presented in this thesis is not intended to be a replacement for palmprint identification using traditional methods. The value of palmar flexion crease identification comes from their use in palmprint images where minutiae are not present, such as those left on fabrics (Fraser *et al.*, 2011) or in blood (Ashbaugh, 1993), or as a fast pre-filter, where palmar flexion creases may be used to complement existing identification methods.

6.2. Thesis contributions

In this thesis, the design and implementation of two new methods of manual and automated palmar flexion crease identification have been used to determine the extent to which automated palmar flexion crease recognition can be used to identify online palmprint images. In answering this question, and given the aims and objectives specified in Chapter 1, the following original contributions have been identified and were investigated in this thesis:

1. The design and implementation of a manual palmar flexion crease extraction, modification, and matching method, which enables the user to efficiently map the location of palmar flexion creases in a given palmprint image, assess the effects of palmar flexion crease distortions through rotation, displacement, and additive noise, and determine the similarity between two or more palmar flexion creases.
2. The design and implementation of an automated palmar flexion crease extraction and matching algorithm, which is capable of automatically extracting palmar flexion creases from online palmprint images, and calculating a matching score that can be used to determine the similarity between two or more palmar flexion creases.

6.3. Future work

In recent years, significant efforts have been undertaken to integrate improved palmprint identification into existing automated identification systems (Federal Bureau of Investigation, 2005b; Police IT Organisation, 2005). Therefore, in the same way, it is important that efforts are made to ensure the successful integration of new palmprint identification methods with existing and future real-world applications. To this end, to aid integration with real-world applications, a number of recommendations can be given to expand the work in this thesis:

1. To improve palmar flexion crease identification in a forensic context, investigations into the type of palmprint images that are commonly found at scenes of crime, and the position, quality, and frequency of palmar flexion creases in those images, should be undertaken.
2. To improve palmprint identification when only small palmar flexion crease sections are available, additional related identification metrics, such as major accessory creases, should be investigated to help finger and palmprint examiners successfully, and confidently, identify partial palmprint impressions.
3. To determine the feasibility of automated palmar flexion crease identification in large scale systems, such as in forensic science, further research should be undertaken to determine the efficacy of automated palmar flexion crease recognition by comparison of automatically identified palmar flexion creases in a large data set.

Furthermore, given the experimental nature of the work presented in thesis, a number of enhancements can be recommended to improve the security, efficiency, and fallibility of the proposed identification algorithms:

1. Alternative image importance measures for automated palmar flexion crease identification should be investigated, particularly for use with latent and Livescan or inked images.
2. An algorithm should be developed to interpolate missing palmar flexion crease line segments in partial palmprint images to aid both manual and automated partial palmprint identification.

References

Adan, M., Adan, A., Vazquez, A.S. and Torres, R. (2008) Biometric verification/identification based on hands natural layout. *Image and Vision Computing*, 26(4), pp.451–465.

Ashbaugh, D.R. (1991a) Palmar flexion crease identification. *Journal of Forensic Identification*, 41(4), pp.255–273.

Ashbaugh, D.R. (1991b) Ridgeology: modern evaluative friction ridge identification. *Fingerprint Whorld*, 17(1), pp.14–16.

Ashbaugh, D.R. (1993) Palmar flexion crease identification. *Fingerprint Whorld*, 19(71), pp.7–15.

Ashbaugh, D.R. (1999a) *Quantitative-qualitative friction ridge skin analysis*. Florida: CRC Press.

Ashbaugh, D.R. (1999b) A science in transition. *In* The Forensic Science Service (1999b) *Proceedings of the 1st International Conference on Forensic Human Identification in the Next Millenium, 24–26 October*. London: The Forensic Science Service.

Avidan, S. and Shamir, A. (2007) Seam carving for content-aware image resizing. *ACM Transactions on Graphics*, 26(3), pp.10:11–10:19.

Babler, W.J. (1977) *The prenatal origins of population differences in human dermatoglyphics*. Ph.D. Thesis, University of Michigan.

Babler, W.J. (1991) Embryological development of epidermal ridges and their configurations. *Birth Defects: Original Article Series*, 27(2), pp.95–112.

Bentley, J.L. (1975) Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9), pp.509–517.

Berge, J. (1977) Orthogonal Procrustes rotation of two or more matrices. *Psychometrika*, 42(2), pp.267–276.

Boles, W.W. and Chu, S.Y.T. (1997) Personal identification using images of the human palm. *In* Queensland University of Technology, School of Electrical and Electronic System Engineering (1997) *Proceedings of the IEEE Region 10 Annual Conference on Speech and Image Technologies for Computing and Telecommunications, 2–4 December*. Brisbane, Australia: IEEE, pp.295–298.

Bolle, R.M., Ratha, N.K. and Pankanti, S. (2004) Performance evaluation in 1:1 biometric engines. *in* Li, S.Z. (ed.) *Advances in Biometric Person Authentication: 5th Chinese Conference on Biometric Recognition, 13–14 December*. Guangzhou, China: Springer,

Boor, C.D. (2001) *A practical guide to splines*. New York City, New York: Springer.

Burt, P.J. and Adelson, E.H. (1983) The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4), pp.532–540.

Canny, J. (1986) A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6), pp.679–698.

Champod, C. (2008) Fingerprint examination: towards more transparency. *Law, Probability, and Risk,* 7(2), pp.111–118.

Champod, C., Lennard, C., Margot, P. and Stoilovic, M. (2004) *Fingerprints and other ridge skin impressions*. Boca Raton, Florida: CRC Press.

Chen, F. and Lu, C.-T. (2008) Nearest neighbour query. *in* Shekhar, S. and Xiong, H. (eds.) *Encyclopedia of GIS*. New York: Springer, pp.776–787.

Chen, J.-C. and Don, H.-S. (1992) Roof edge detection: a morphological skeleton approach. *in* Archibald, C. and Petriu, E. (eds.) *Advances in machine vision: strategies and applications*. River Edge, New Jersey: World Scientific, pp.171–192.

Clark, J.D. (2002) ACE-V: is it scientifically reliable and accurate? *Journal of Forensic Identification,* 52(4), pp.401–408.

Cole, S.A. (2004) History of fingerprint pattern recognition. *in* Ratha, N.K. and Bolle, R.M. (eds.) *Automated fingerprint recognition systems*. New York: Springer,

Cole, S.A. (2006) Is fingerprint identification valid? Rhetorics of reliability in fingerprint proponents' discourse. *Law and Policy,* 28(1), pp.109–135.

Davis, L.S. (1975) A survey of edge detection techniques. *Computer Graphics and Image Processing,* 4(1), pp.248–270.

Duda, R. and Hart, P. (1973) *Pattern classification and scene analysis*. New York: John Wiley and Sons.

Epstein, R. (2002) Fingerprints meet Daubert: the myth of fingerprint science is revealed. *Southern California Law Review,* 75(1), pp.605–657.

Federal Bureau of Investigation (2005a) *Electonic fingerprint transmission specification*. IAFIS-DOC-01078-7.1, Clarksburg, West Virginia: Criminal Justice Information Services Division, Federal Bureau of Investigation.

Federal Bureau of Investigation (2005b) *Next generation integrated automated fingerprint identification (NGI) system*. Requirements overview & analysis project, Clarksburg, West Virginia: Criminal Justice Information Services Division, Federal Bureau of Investigation.

Fisher, B.A.J. (2004) *Techniques of crime scene investigation*. Florida: CRC Press.

Fraser, J., Sturrock, K., Deacon, P., Bleay, S., and Bremmer, D.H. (2011) Visualisation of fingermarks and grab impressions on fabrics. Part 1: Gold/zinc vacuum metal deposition. *Forensic Science Interntional,* 208(1–3), pp.74–78.

Frei, W. and Chen, C.C. (1977) Fast boundary detection: a generalization and a new algorithm. *IEEE Transactions on Computers,* 26(10), pp.988–998.

Fuchs, E. (1999) Epidermal differentiation: the bare essentials. *The Journal of Cell Biology,* 111(6), pp.2807–2814.

Galton, F. (1892) *Finger Prints*. London: MacMillan and Co.

Goa, Y. and Leung, M.K.H. (2002) Face recognition using line edge map. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(6), pp.764–779.

Gonzalez, R.C. and Woods, R.E. (2008) *Digital image processing*. 3rd ed., Upper Saddle River, New Jersey: Prentice Hall.

Goodall, C. (1991) Procrustes methods in the statistical analysis of shape. *Journal of the Royal Statistical Society: Series B*, 53(2), pp.285–339.

Gower, J.C. (1975) Generalized Procrustes analysis. *Psychometrika*, 40(1), p.33—51.

Haber, L. and Haber, R.N. (2008) Scientific validation of fingerprint evidence under Daubert. *Law, Probability, and Risk*, 7(2), pp.87–109.

Hale, A.R. (1949) Breath of epidermal ridges in the human fetus and its relation to the growth of the hand and foot. *The Anatomical Record*, 105(4), pp.763–776.

Hale, A.R. (1952) Morphogenesis of volar skin in the human fetus. *American Journal of Anatomy*, 91(1), pp.147–181.

Han, C.C. (2004) A hand-based personal authentication using a coarse-to-fine strategy. *Image and Vision Computing*, 22(11), pp.909–918.

Han, C.C., Cheng, H.L., Lin, C.L. and Fan, K.C. (2003) Personal authentication using palm-print features. *Pattern Recognition*, 36(2), pp.371–381.

Haralick, R.M. (1984) Digital step edges from zero crossing of second directional derivatives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1), pp.58–68.

Haralick, R.M., Sternberg, S.R. and Zhuang, X. (1987) Image analysis using mathematical morphology. *Pattern Analysis and Machine Intelligence*, 9(4), pp.532–550.

Henry, E.R. (1900) *Classification and uses of finger prints*. 4th ed., London: Georges Routledge.

Huang, D.-S., Jia, W. and Zhang, D. (2008) Palmprint verification based on principal lines. *Pattern Recognition*, 41(4), pp.1316–1328.

Huber, R.A. (1972) The philosophy of identification. *RCMP Gazette*, 34(7–8), pp.9–14.

Interpol (2004) *AFIS requirements*. Model clauses for AFIS acquisition, Lyon, France: Interpol AFIS Expert Group.

Jain, A. and Feng, J. (2009) Latent palmprint matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(6), pp.1032–1047.

Jia, W., Huang, D.-S. and Zhang, D. (2008) Palmprint verification based on robust line orientation code. *Pattern Recognition*, 41(4), pp.1316–1328.

Keegan, J.F. (1977) How can you tell if two line drawings are the same? *Computer Graphics and Image Processing*, 6(1), pp.90–92.

Kimura, S. (1991) Embryologic development of flexion creases. *Birth Defects: Original Article Series*, 27(2), pp.113–129.

Kimura, S. and Kitagawa, T. (1986) Embryological development of human palmar, plantar, and digital flexion creases. *The Anatomical Record*, 216(2), pp.191–197.

Kochanek, D.H.U. and Bartels, R.H. (1984) Interpolating splines with tension, continuity, and bias control. *Computer Graphics*, 18(3), pp.33–41.

Kodicek, D. (2005) *Mathematics and physics for programmers*. Hingham, Massachusetts: Charles River Media, Inc.

Kong, A. and Zhang, D. (2004) Competitive coding scheme for palmprint verification. *In* International Association of Pattern Recognition (2004) *Proceedings of the 17th International Conference on Pattern Recognition, 23–26 August*. Cambridge, UK: IEEE Computer Society, pp.520–523.

Kong, A., Zhang, D. and Kamel, M. (2006a) Palmprint identification using feature level fusion. *Pattern Recognition*, 39(3), pp.478–487.

Kong, A., Zhang, D. and Kamel, M. (2009) A survey of palmprint recognition. *Pattern Recognition*, 42(7), pp.1408–1418.

Kong, A., Zhang, D. and Lu, G. (2006b) A study of identical twins' palmprints for personal verification. *Pattern Recognition*, 39(11), pp.2149–2156.

Kong, J., Lu, Y., Wang, S., Qi, M. and Li, H. (2008) A two-stage neural network-based personal identification system using handprint. *Neurocomputing*, 71(4–6), pp.641–647.

Kong, S.G., Heo, J., Abidi, B.R., Paik, J. and Abidi, M.A. (2005) Recent advances in visual and infrared face recognition. *Computer Vision and Image Understanding*, 97(1), pp.103–135.

Kong, W. and Zhang, D. (2002) Palmprint texture analysis based on low-resolution images for personal authentication. *In* International Association of Pattern Recognition (2002) *Proceedings of the 16th International Conference on Pattern Recognition, 11–15 August*. Quebec, Canada: IEEE Computer Society, pp.807–810.

Kovesi, P. (1999) Image features from phase congruency. *Journal of Computer Vision Research*, 1(3), pp.2–26.

Kovesi, P. (2002) Edges are not just steps. *In* Australian Pattern Recognition Society (2002) *Proceedings of the 5th Asian Conference on Computer Vision, 23–25 January*. Melbourne, Australia: Australian Pattern Recognition Society, pp.822–827.

Kovesi, P. (2003) Phase congruency detects corners and edges. *In* Australian Pattern Recognition Society (2003) *Proceedings of the 7th Australian Pattern Recognition Society Conference in Digital Image Computing, 10–12 December*. Sydney, Australia: Australian Pattern Recognition Society, pp.309–318.

Kumar, A., Wong, D.C.M., Shen, H.C. and Jain, A.K. (2003) Personal verification using palmprint and hand geometry biometric. *Lecture Notes In Computer Science*, 2688(1), pp.668–678.

Kung, S.Y., Lin, S.H. and Fang, M. (1995) A neural network approach to face/palm recognition. *In* Institute of Electrical and Electronic Engineers, IEEE Signal Processing Society (1995) *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing, 1–2 September*. Cambridge, Massachusetts: IEEE Signal Processing Society, pp.323–332.

Lacroix, B., Wolff-Quenot, M. and Haffen, K. (1984) Early human hand morphology: an estimation of fetal age. *Early Human Development*, 9(2), pp.127–136.

Lee, T.S. (1996) Image representation using 2D Gabor wavelets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(10), pp.959–971.

Li, F. and Leung, M.K.H. (2006) Hierarchical identification of palmprint using line-based Hough transform. *In* International Association of Pattern Recognition (2006) *Proceedings of the 18th International Conference on Pattern Recognition, 20–24 August*. Hong Kong, China: IEEE Computer Society, pp.149–152.

Li, W. (2003) *Authenticating personal identities using palmprint recognition*. Ph.D. Thesis, The Hong Kong Polytechnic University.

Li, W., Xia, S., Zhang, D. and Xu, Z. (2003a) A new palmprint segmentation method based on an inscribed circle. *Image Processing and Communication*, 9(2), pp.63–70.

Li, W., Zhang, D. and Xu, Z. (2003b) Image alignment based on invariant features for palmprint identifciation. *Signal Processing: Image Communication*, 18(5), pp.373–379.

Libal, A. (2006) *Fingerprints, bite marks and ear prints*. Pennsylvania: Mason Crest.

Liu, L. and Zhang, D. (2005) Palm-line detection. *In* Institute of Electrical and Electronic Engineers, IEEE Signal Processing Society (2005) *Proceedings of the International Conference on Image Processing, 11–14 September*. Genoa, Italy: IEEE Signal Processing Society, pp.269–272.

Lu, G., Zhang, D. and Wang, K. (2003) Palmprint recognition using eigenpalms features. *Pattern Recognition*, 24(9–10), pp.1463–1467.

Margot, P. and Lennard, C. (1994) *Fingerprint detection techniques*. 6th rev. ed., Lausanne, Switzerland: Institut de Police Scientifique et de Criminologie, Université de Lausanne.

Marr, D. and Hildrith, E. (1980) The theory of edge detection. *Proceedings of the Royal Society of London*, 207(1167), pp.187–217.

Marsh, D. (2005) *Applied geometry for computer graphics and CAD*. New York: Springer.

Matolsy, A.G. (1976) Keratinization. *The Journal of Investigative Dermatology*, 67(1), pp.20–25.

Matus, F. and Flusser, J. (1993) Image representations via a finite Radon transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(10), pp.996–1006.

Misumi, Y. and Akiyoshi, T. (1984) Scanning electron microscopic structure of the finger print as related to the dermal surface. *The Anatomical Record*, 208(1), pp.49–55.

Morrone, M.C. and Owens, R.A. (1987) Feature detection from local energy. *Pattern Recognition Letters*, 6(5), pp.303–313.

Morrone, M.C., Ross, J.R., Burr, D.C. and Owens, R.A. (1986) Mach bands are phase dependent. *Nature*, 324(6094), pp.250–253.

National Institute of Standards and Technology (1999) *Data format for the interchange of fingerprint, facial, and scar mark and tattoo (SMT) information*. ANSI/NIST-ITL 1-1999 (Draft), Gaithersburg, Maryland: National Institute of Standards and Technology, U.S. Department of Commerce.

Nawal, V.S. (1993) *A guided tour of computer vision*. Massachusetts: Addison-Wesley.

Otsu, N. (1979) A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1), pp.62–66.

Panigrahy, R. (2008) An improved algorithm finding nearest neighbor using *k*d-trees. *In* Pontifícia Universidade Católica do Rio de Janeiro (2008) *Proceedings of the 8th Theoretical Informatics Latin American Symposium, 7–11 April*. Búzios, Brazil: Springer, pp.388–398.

Parker, C.J. (2006) Austin's automated finger and palm print system. *Lone Star Forensic Journal*, 60(3), pp.15–16.

Penrose, I.S. and O'Hara, P.T. (1973) The development of epidermal ridges. *Journal of Medical Genetics*, 10(3), pp.201–208.

Police IT Organisation (2005) *Biometrics technology roadmap for person identification within the police Service*. Part 1: Identification Roadmap 2005–2020, London: Police IT Organisation.

Poon, C., Wong, D.C.M. and Shen, H.C. (2004) A new method in locating and segmenting palmprint into region-of-interest. *In* International Association of Pattern Recognition (2004) *Proceedings of the 17th International Conference on Pattern Recognition, 23–26 August*. Cambridge, UK: IEEE Computer Society, pp.533–536.

Popich, G.A. and Smith, D.W. (1970) The genesis and significance of digital and palmar hand creases: preliminary report. *Journal of Pediatrics*, 77(6), pp.1017–1023.

Pratt, W.K. (2007) *Digital image processing*. 4th ed., Hoboken, New Jersey: John Wiley & Sons, Inc.

Prewitt, J. and Mendelsohn, M. (1966) The analysis of cell images. *Annals of the New York Academy of Sciences*, 128(3), pp.1035–1053.

Qiao, Y., Li, Z., Wang, Q., Zeng, Y. and Liang, K. (2005) Identification of palm print using dermatoglyphics analysis and detection system. *Medical Engineering & Physics*, 27(3), pp.229–235.

Rahman, A., Anwar, F. and Azad, S. (2008) A simple and effective technique for human verification with hand geometry. *In* IEEE Communication Society (2008) *Proceedings of the International Conference on Computer and Communication Engineering, 13–15 May*. Kuala Lumpur, Malaysia: IEEE Communication Society, pp.1177–1180.

Ritter, G.X. and Wilson, J.N. (2001) *Handbook of computer vision algorithms in image algebra*. 2nd ed., Boca Raton, Florida: CRC Press.

Roberts, L.G. (1965) Machine perception of three-dimensional solids. *in* Tippet, J.T. (ed.) *Optical and Electro-Optical Information Processing*. Cambridge, Massachusetts: MIT Press, pp.159–197.

Rodrigues, P.A.R. and Silva, J.D.L. (1996) Biometric identification by dermatoglyphics. *In* Institute of Electrical and Electronic Engineers, IEEE Signal Processing Society (1996) *Proceedings of the International Conference on Image Processing, 16–19 September*. Lausanne, Switzerland: IEEE Signal Processing Society, pp.319–322.

Rogers, D.F. and Adams, J.A. (1990) *Mathematical elements for computer graphics*. 2nd ed., New York: McGraw-Hill Publishing Company.

Ross, A., Nandakumar, K. and Jain, A. (2006) *Handbook of multibiometrics*. New York: Springer.

Schneider, J.K. (2007) Ultrasonic fingerprint sensors. *in* Ratha, N.K. and Govindaraju, V. (eds.) *Advances in Biometrics*. Berlin: Springer, pp.63–74.

Schneider, J.K. and Gojevic, S.M. (2001) Ultrasonic imaging systems for personal identification. *In* Institute of Electrical and Electronic Engineers (2001) *Proceedings of the IEEE Ultrasonics Symposium, 7–10 October*. Atlanta, Georgia: IEEE, pp.595–601.

Schneider, J.K. and Wobschall, D.C. (1991) Live scan fingerprint imagery using high resolution C-scan ultrasonography. *In* Institute of Electrical and Electronic Engineers (1991) *Proceedings of the 25th Annual IEEE International Carnahan Conference on Security Technology, 1–3 October*. Taipei, Taiwan: IEEE, pp.88–95.

Scottish Police Services Authority (2007) *IDENT-1 technical bulletin*. 3, Scottish Police Services Authority Forensic Services.

Sedgewick, R. (1983) *Algorithms*. Massachusetts: Addison-Wesley.

Serra, J. (1986) Introduction to mathematical morphology. *Computer Vision, Graphics, and Image Processing*, 35(3), pp.283–305.

Shang, L., Huang, D.-S., Du, J.-X. and Zheng, C.-H. (2006) Palmprint recognition using FastICA algorithm and radial basis probabilistic neural network. *Neurocomputing*, 69(13–15), pp.1782–1786.

Shapiro, L.G. and Stockman, G.C. (2001) *Computer vision*. Upper Saddle River, New Jersey: Prentice Hall.

Soille, P. (2003) *Morphological image analysis: principles and applications*. 2nd ed., Secaucus, New Jersey: Springer-Verlag.

Stevens, C.A., Carey, J.C., Shah, M. and Bagley, G.P. (1988) Development of human palmar and digital flexion creases. *Journal of Pediatrics*, 113(1), pp.128–132.

Štruc, V. and Pavešić, N. (2009a) A palmprint verification system based on phase congruency features. *In* European Cooperation in Science and Technology (2009a) *Proceedings of the 1st European Workshop of Biometrics and Identity Management, 7–9 May*. Roskilde, Denmark: Springer-Verlag, pp.110–119.

Štruc, V. and Pavešić, N. (2009b) Phase congruency features for palm-print verification. *IET Signal Processing*, 3(4), pp.258–268.

Tay, J.S.H. (1979) The genetics of palmar creases: a study in the inheritance of liability estimated from the incidence among relatives. *Annals of Human Genetics*, 42(3), pp.327–332.

Taylor, R. and Knapp, M. (2004) Indianapolis PD converts to full-hand scanning. *Law Enforcement Technology*, June 2004.

The Hong Kong University of Science and Technology (2003) *Hand image database* [online]. Hong Kong: The Hong Kong University of Science and Technology. [Accessed August 2007]. Available at: <http://visgraph.cs.ust.hk/biometrics>.

The Hong Kong Polytechnic University (2003) *PolyU Palmprint Database* [online]. Hong Kong: The Hong Kong Polytechnic University. [Accessed February 2011]. Available at: <http://www.comp.polyu.edu.hk/~biometrics>.

Torre, V. and Poggio, T. (1986) On edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(2), pp.147–163.

Townley, L. and Ede, R. (2004) *Forensic practice in criminal cases*. London: The Law Society.

Tuthill, H. (1994) *Individualization: principles and procedures in criminalistics*. Jacksonville, Floride: Lightning Powder Company, Inc.

Wertheim, K. and Maceo, A. (2002) Friction ridge and pattern formation during the critical stage. *Journal of Forensic Identification,* 52(1), pp.35–85.

Wong, M., Zhang, D., Kong, W.K. and Lu, G. (2005) Real-time palmprint acquisition system design. *IEE Proceedings: Vision, Image, and Signal Processing*, 152(5), pp.527–534.

Wu, P. and Li, M. (1997) Pyramid edge detection based on stack filter. *Pattern Recognition Letters*, 18(4), pp.239–248.

Wu, X., Wang, K. and Zhang, D. (2002a) Fuzzy directional element energy feature (FDEEF) based palmprint identification. *In* International Association of Pattern Recognition (2002a) *Proceedings of the 16th International Conference on Pattern Recognition, 11–15 August*. Quebec, Canada: IEEE Computer Society, pp.95–98.

Wu, X., Wang, K. and Zhang, D. (2002b) Line feature extraction and matching in palmprint. *In* China Society of Image and Graphics (2002b) *Proceedings of the 2nd International Conference on Image Processing and Graphics, 16–18 August*. Heifei, China: SPIE, pp.583–590.

Wu, X., Wang, K. and Zhang, D. (2004a) A novel approach of palm-line extraction. *In* China Society of Image and Graphics (2004a) *Proceedings of the 3rd International Conference on Image and Graphics, 18–20 December*. Hong Kong, China: IEEE Computer Society, pp.230–233.

Wu, X., Wang, K. and Zhang, D. (2004b) Palmprint recognition using directional line energy feature. *In* International Association of Pattern Recognition (2004b) *Proceedings of the 17th International Conference on Pattern Recognition, 23–26 August*. Cambridge, UK: IEEE Computer Society, pp.475–478.

Wu, X., Zhang, D. and Wang, B. (2004c) Palmprint classification using principal lines. *Pattern Recognition*, 37(10), pp.1987–1988.

Wu, X., Zhang, D. and Wang, K. (2006) Palm line extraction and matching for personal authentication. *IEEE Transactions on Systems, Man and Cybernetics*, 36(5), pp.978–987.

Yager, N. and Amin, A. (2004a) Fingerprint classification: a review. *Pattern Analysis and Applications*, 7(1), pp.77–93.

Yager, N. and Amin, A. (2004b) Fingerprint verification based on minutiae features: a review. *Pattern Analysis and Applications*, 7(1), pp.94–113.

Yang, J., Yang, J.-Y. and Niu, B. (2007) Globally maximizing, locally minimizing: unsupervised discriminant projection with applications to face and palm biometrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(4), pp.650–664.

Yoruk, E., Konukoglu, E., Sankur, B. and Darbon, J. (2006) Shape-based hand recognition. *IEEE Transactions on Image Processing*, 15(7), pp.1803–1815.

Zamperoni, P. (1995) Image enhancements. *in* Greenfield, D., Hawkes, P. and Monastyrskii, M. (eds.) *Advances in Imaging and Electron Physics, Volume 92*. New York: Academic Press, pp.1–77.

Zhang, D. (2000) *Automated biometrics: technologies and systems*. Berlin: Springer.

Zhang, D. (2004) *Palmprint authentication*. Norwell, Massachusetts: Kluwer Academic.

Zhang, D., Kong, A., You, J. and Wong, M. (2003) Online palmprint identification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9), pp.1041–1050.

Zhang, D. and Shu, W. (1999) Two novel characteristics in palmprint verification: datum point invariance and line feature matching. *Pattern Recognition*, 32(4), pp.691–702.

Zhao, W., Challappa, R., Phillips, P.J. and Rosenfeld, A. (2003) Face recognition: a literature survey. *ACM Computer Surveys*, 35(4), pp.399–458.

Zhao, Z.-Q., Huang, D.-S. and Jia, W. (2007) Palmprint recognition with 2DPCA+PCA based on modular neural networks. *Neurocomputing*, 71(1–3), pp.448–454.

Ziou, D. and Tabbone, S. (1998) Edge detection techniques—an overview. *Pattern Recognition and Image Analysis*, 8(4), pp.537–559.

Appendix A

This appendix lists the source code for each algorithm, and describes the steps required to recreate the identification and matching process for manual, automated, and partial palmprint identification.

Manual palmprint identification

Palmar flexion crease identification

FlexionCreaseTracing.exe is a C# application that can be compiled against the .NET framework using the C# compiler (csc.exe) with the command:

```
csc.exe /target:winexe /unsafe
/out:FlexionCreaseTracing.exe DisplayPanel.cs
CardinalSpline.cs UnsafeBitmap.cs FormMain.cs Program.cs
```

The source files for FlexionCreaseTracing.exe are CardinalSpline.cs, UnsafeBitmap.cs, FormMain.cs, and Program.cs.

DisplayPanel.cs

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;
using System.Collections;
using System.IO;

namespace FlexionCreaseTracing
{
    class DisplayPanel : UserControl
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
        /// otherwise, false.</param>
        protected override void Dispose(bool disposing) {
            if (disposing && (components != null)) {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent() {
            this.SuspendLayout();
            //
            // DisplayPanel
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackColor = System.Drawing.SystemColors.AppWorkspace;
            this.Cursor = System.Windows.Forms.Cursors.Cross;
            this.Name = "DisplayPanel";
            this.Size = new System.Drawing.Size(84, 68);
            this.ResumeLayout(false);
        }

        private string fileDirectory;
        private int fileIndex;
        private FileInfo[] imageFiles;
        private Bitmap currentImage;
        private ArrayList cardinalSplines;
        private CardinalSpline currentSpline;

        public int FileIndex {
            get { return fileIndex; }
        }

        public int FileCount {
            get {
                if (imageFiles != null) {
                    return imageFiles.Length;
                }
                return 0;
            }
        }

        public string FileDirectory {
            get { return fileDirectory; }
            set {
                fileDirectory = value;
                OnDirectoryChanged(new EventArgs());
            }
        }

        public int ImageWidth {
            get {
                if (currentImage != null) {
                    return currentImage.Width;
                }
                return 0;
            }
        }

        public int ImageHeight {
            get {
                if (currentImage != null) {
                    return currentImage.Height;
                }
                return 0;
            }
        }

        public CardinalSpline CurrentSpline {
            get { return currentSpline; }
            set {
                currentSpline = value;
                OnCurrentSplineChanged(new EventArgs());
            }
        }

        public DisplayPanel() {
            this.cardinalSplines = new ArrayList();
            InitializeComponent();
            SetStyle(ControlStyles.AllPaintingInWmPaint |
                ControlStyles.ResizeRedraw | ControlStyles.OptimizedDoubleBuffer,
                true);
        }

        public void SaveImages(string directory) {
            if (imageFiles == null) {
                return;
            }
            if (!Directory.Exists(directory)) {
                try {
                    Directory.CreateDirectory(directory);
                } catch (Exception error) {
                    MessageBox.Show(error.Message, "Error",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                }
            }
            if (currentImage == null) {
                return;
            }
            UnsafeBitmap labelled = new UnsafeBitmap(currentImage);
            foreach (CardinalSpline spline in cardinalSplines) {
                labelled.DrawPath(spline.Path);
            }
            labelled.Image.Save(string.Format("{0}\\{1}", directory,
                imageFiles[fileIndex].Name));
        }

        public void SaveSplines(string directory) {
            if (imageFiles == null) {
                return;
            }
            if (!Directory.Exists(directory)) {
                try {
                    Directory.CreateDirectory(directory);
                } catch (Exception error) {
                    MessageBox.Show(error.Message, "Error", MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
                    return;
                }
            }
            string filePath = string.Format("{0}\\{1}.txt", directory,
                imageFiles[fileIndex].Name);
            StreamWriter fileWriter = null;
            try {
                fileWriter = new StreamWriter(filePath, false);
                fileWriter.WriteLine(imageFiles[fileIndex].Name);
                foreach (CardinalSpline cardinalSpline in cardinalSplines) {
                    fileWriter.WriteLine(cardinalSpline.Tension);
                    for (int i = 0; i < cardinalSpline.ControlPoints.Length;
                    i++) {
                        fileWriter.WriteLine(string.Format("{0},{1}",
                            cardinalSpline.ControlPoints[i].X,
                            cardinalSpline.ControlPoints[i].Y)
                        );
                    }
                }
            } catch (Exception error) {
                MessageBox.Show(error.Message, "Error", MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            } finally {
                if (fileWriter != null) {
                    fileWriter.Close();
                }
            }
        }

        public void OpenSplines(string directory) {
            if (imageFiles == null) {
                return;
            }
            string filePath = string.Format("{0}\\{1}.txt", directory,
                imageFiles[fileIndex].Name);
            if(!File.Exists(filePath)) {
                return;
            }
            StreamReader fileReader = null;
            CardinalSpline newSpline = null;
            try {
                fileReader = new StreamReader(filePath);
                if (fileReader.ReadLine().Equals(imageFiles[fileIndex].Name)) {
                    string nextLine;
                    while ((nextLine = fileReader.ReadLine()) != null) {
                        if (!nextLine.Contains(",")) {
                            newSpline = new CardinalSpline();
                            newSpline.Tension = float.Parse(nextLine);
                            cardinalSplines.Add(newSpline);
                        } else {
                            if (newSpline != null) {
```

```
                        newSpline.AddControlPoint(
                            new Point(
                                int.Parse(nextLine.Substring(0,
                                nextLine.IndexOf(','))),
                                int.Parse(nextLine.Substring(
                                nextLine.IndexOf(',') + 1))));
                    }
                }
            }
        }
    } catch (Exception error) {
        MessageBox.Show(error.Message, "Error", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    } finally {
        if (fileReader != null) {
            fileReader.Close();
        }
        Refresh();
    }
}

private void SetAutoScrollSize() {
    AutoScrollMinSize = new Size(ImageWidth, ImageHeight);
    Refresh();
}

private void ResetPanel() {
    imageFiles = null;
    fileIndex = -1;
    ClearImage();
}

public event ImageClickEvent ImageClick;
public event ImageChangedEvent ImageChanged;
public event DirectoryChangedEvent DirectoryChanged;
public event CurrentSplineChangedEvent CurrentSplineChanged;

protected virtual void OnImageClick(MouseEventArgs e) {
    if (ImageClick != null) {
        ImageClick(this, e);
    }
}

protected virtual void OnImageChanged(EventArgs e) {
    RemoveAllSplines();
    SetAutoScrollSize();
    if (ImageChanged != null) {
        ImageChanged(this, e);
    }
}

protected virtual void OnDirectoryChanged(EventArgs e) {
    if (Directory.Exists(fileDirectory)) {
        DirectoryInfo directoryInfo = new DirectoryInfo(fileDirectory);
        imageFiles = directoryInfo.GetFiles("*.bmp");
        if (imageFiles.Length > 0) {
            fileIndex = 0;
            LoadImageFromFile(imageFiles[fileIndex].FullName);
        } else {
            ResetPanel();
        }
    } else {
        ResetPanel();
    }
    if (DirectoryChanged != null) {
        DirectoryChanged(this, e);
    }
}

protected virtual void OnCurrentSplineChanged(EventArgs e) {
    if (CurrentSplineChanged != null) {
        CurrentSplineChanged(this, e);
    }
}

protected override void OnPaint(PaintEventArgs e) {
    if (currentImage == null) {
        return;
    }
    int X = (ClientRectangle.Width < ImageWidth) ?
        AutoScrollPosition.X : (ClientRectangle.Width - ImageWidth) / 2;
    int Y = (ClientRectangle.Height < ImageHeight) ?
        AutoScrollPosition.Y : (ClientRectangle.Height - ImageHeight) / 2;
    e.Graphics.InterpolationMode = InterpolationMode.NearestNeighbor;
    e.Graphics.DrawImage(currentImage, X, Y, ImageWidth, ImageHeight);
    Graphics imageGraphics = e.Graphics;
    imageGraphics.Clip = new Region(new Rectangle(X, Y, ImageWidth,
        ImageHeight));
    imageGraphics.TranslateTransform((float)X, (float)Y);
    foreach (CardinalSpline cardinalSpline in cardinalSplines) {
        cardinalSpline.DrawSpline(imageGraphics,
            (cardinalSpline == CurrentSpline) ? true : false);
    }
}

protected override void OnMouseClick(MouseEventArgs Event) {
    if (currentImage == null) {
        return;
    }
    int X = (ClientRectangle.Width < ImageWidth) ?
        AutoScrollPosition.X : (ClientRectangle.Width - ImageWidth) / 2;
    int Y = (ClientRectangle.Height < ImageHeight) ?
        AutoScrollPosition.Y : (ClientRectangle.Height - ImageHeight) / 2;
    if ((Event.X >= X && Event.X <= X + ImageWidth)
    && (Event.Y >= Y && Event.Y <= Y + ImageHeight)) {
        if (Event.Button == MouseButtons.Right) {
            OnImageRightClick();
        } else {
            OnImageLeftClick(new MouseEventArgs(Event.Button, Event.Clicks,
                Event.X - X, Event.Y - Y, Event.Delta));
        }
        OnImageClick(Event);
        Refresh();
    }
}

private void OnImageLeftClick(MouseEventArgs e) {
    bool controlPointClicked = false;
    if (CurrentSpline == null) {
        foreach (CardinalSpline cardinalSpline in cardinalSplines) {
            if (cardinalSpline.ControlPoints.Length > 1) {
                if (cardinalSpline.Path.IsOutlineVisible(new Point(e.X, e.Y),
                new Pen(Color.Black, 10.0F))) {
                    CurrentSpline = cardinalSpline;
                    return;
                }
            }
        }
        CurrentSpline = new CardinalSpline();
        cardinalSplines.Add(CurrentSpline);
    }
    if (!controlPointClicked) {
        CurrentSpline.AddControlPoint(new Point(e.X, e.Y));
    }
}

private void OnImageRightClick() {
    if (CurrentSpline != null) {
        if (CurrentSpline.ControlPoints.Length <= 1) {
            cardinalSplines.Remove(CurrentSpline);
        }
    }
    CurrentSpline = null;
}

public void RemoveSpline(CardinalSpline cardinalSpline) {
    cardinalSplines.Remove(cardinalSpline);
    CurrentSpline = null;
    Refresh();
}

public void RemoveAllSplines() {
    cardinalSplines.Clear();
    CurrentSpline = null;
```

```
            Refresh();
        }

        public void ShowNextImage() {
            if (imageFiles == null) {
                return;
            }
            fileIndex++;
            if (fileIndex == FileCount) {
                fileIndex = 0;
            }
            LoadImageFromFile(imageFiles[fileIndex].FullName);
        }

        public void ShowPreviousImage() {
            if (imageFiles == null) {
                return;
            }
            fileIndex--;
            if (fileIndex < 0) {
                fileIndex = FileCount - 1;
            }
            LoadImageFromFile(imageFiles[fileIndex].FullName);
        }

        private void LoadImageFromFile(string filePath) {
            try {
                currentImage =
                    (Bitmap)Bitmap.FromFile(imageFiles[fileIndex].FullName);
            } catch (Exception Error) {
                MessageBox.Show(Error.Message, "Error", MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            }
            OnImageChanged(new EventArgs());
            Refresh();
        }

        private void ClearImage() {
            currentImage = null;
            OnImageChanged(new EventArgs());
            Refresh();
        }
    }

    public delegate void ImageClickEvent(object sender, MouseEventArgs e);
    public delegate void ImageChangedEvent(object sender, EventArgs e);
    public delegate void DirectoryChangedEvent(object sender, EventArgs e);
    public delegate void CurrentSplineChangedEvent(object sender, EventArgs e);
}
```

CardinalSpline.cs

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.IO;

namespace FlexionCreaseTracing
{
    public class CardinalSpline
    {
        private float tension;
        private Point[] controlPoints;

        public Point[] ControlPoints {
            get { return controlPoints; }
        }

        public float Tension {
            get { return tension; }
            set { tension = value; }
        }

        public GraphicsPath Path {
            get {
                GraphicsPath newPath = new GraphicsPath();
                newPath.AddCurve(controlPoints, Tension);
                return newPath;
            }
        }

        public CardinalSpline() {
            this.Tension = 0.5F;
            this.controlPoints = null;
        }

        public void AddControlPoint(Point newPoint) {
            if (controlPoints == null) {
                controlPoints = new Point[1];
            } else {
                controlPoints = ResizeSpline(controlPoints,
                    controlPoints.Length + 1);
            }
            controlPoints[controlPoints.Length - 1] = newPoint;
        }

        public void DrawSpline(Graphics graphics, bool drawControlPoints) {
            if (drawControlPoints) {
                foreach (Point controlPoint in controlPoints) {
                    graphics.DrawRectangle(new Pen(Color.Black),
                        new Rectangle(controlPoint.X - 3,
                        controlPoint.Y - 3, 6, 6));
                }
            }

            if (controlPoints.Length > 1) {
                graphics.DrawPath(new Pen(Color.Black), Path);
            }
        }

        public static Point[] ResizeSpline(Array source, int newSize) {
            int oldSize = source.Length;
            Type elementType = source.GetType().GetElementType();
            Array newArray = Array.CreateInstance(elementType, newSize);
            int preserveLength = Math.Min(oldSize, newSize);
            if (preserveLength > 0) {
                Array.Copy(source, newArray, preserveLength);
            }
            return (Point[])newArray;
        }
    }
}
```

UnsafeBitmap.cs

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;

namespace FlexionCreaseTracing
{
    public unsafe class UnsafeBitmap
    {
        public struct PixelData {
            public byte blue;
            public byte green;
            public byte red;
        }

        private GraphicsUnit unit = GraphicsUnit.Pixel;
        private BitmapData data;
        private Bitmap image;

        public Bitmap Image {
            get { return this.image; }
            set { this.image = value; }
        }

        public Point PixelSize {
            get {
                RectangleF bounds = this.image.GetBounds(ref this.unit);
                return new Point((int)bounds.Width, (int)bounds.Height);
            }
        }

        private int width;
        private byte* pBase;

        public UnsafeBitmap() {

        }

        public UnsafeBitmap(Bitmap image) {
            this.image = new Bitmap(image);
        }

        public UnsafeBitmap(Size dimensions) {
            this.image = new Bitmap(dimensions.Width, dimensions.Height);
        }

        public void Dispose() {
            if (this.image != null)
            {
                this.image.Dispose();
            }
        }

        public void DrawPath(GraphicsPath path) {
            Graphics.FromImage(this.image).DrawPath(new Pen(Color.Red), path);
        }

        private PixelData* GetPixelAt(Point location) {
            return (PixelData*)
                (pBase + location.Y * width + location.X * sizeof(PixelData));
        }

        public PixelData GetPixel(Point location) {
            return *GetPixelAt(location);
        }

        public void LockBitmap() {
            RectangleF boundsF = this.Image.GetBounds(ref this.unit);
            Rectangle bounds = new Rectangle((int)boundsF.X, (int)boundsF.Y,
                (int)boundsF.Width, (int)boundsF.Height);
            width = (int)boundsF.Width * sizeof(PixelData);
            if (width % 4 != 0) {
                width = 4 * (width / 4 + 1);
            }
            this.data = this.Image.LockBits(bounds, ImageLockMode.ReadWrite,
                PixelFormat.Format24bppRgb);
            this.pBase = (Byte*)this.data.Scan0.ToPointer();
        }

        public void UnlockBitmap() {
            if (this.data != null) {
                this.Image.UnlockBits(this.data);
            }
            this.data = null;
            this.pBase = null;
        }
    }
}
```

FormMain.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Runtime.InteropServices;

namespace FlexionCreaseTracing
```

```
{
    class FormMain : Form
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
        ///     otherwise, false.</param>
        protected override void Dispose(bool disposing) {
            if (disposing && (components != null)) {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent() {
            this.Button_Next = new System.Windows.Forms.Button();
            this.Button_Previous = new System.Windows.Forms.Button();
            this.MenuStrip_Main = new System.Windows.Forms.MenuStrip();
            this.MenuItem_File = new System.Windows.Forms.ToolStripMenuItem();
            this.MenuItem_File_Change_Directory =
                new System.Windows.Forms.ToolStripMenuItem();
            this.MenuSeparator_Change_Directory_Exit =
                new System.Windows.Forms.ToolStripSeparator();
            this.MenuItem_File_Exit = new System.Windows.Forms.ToolStripMenuItem();
            this.MenuItem_Help = new System.Windows.Forms.ToolStripMenuItem();
            this.MenuItem_Help_About = new System.Windows.Forms.ToolStripMenuItem();
            this.trackBarTension = new System.Windows.Forms.TrackBar();
            this.labelTension = new System.Windows.Forms.Label();
            this.textBoxTension = new System.Windows.Forms.TextBox();
            this.buttonClearAll = new System.Windows.Forms.Button();
            this.buttonClearSelected = new System.Windows.Forms.Button();
            this.buttonSave = new System.Windows.Forms.Button();
            this.displayPanel = new PrincipalLineTracing.DisplayPanel();
            this.MenuStrip_Main.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)
                (this.trackBarTension)).BeginInit();
            this.SuspendLayout();
            //
            // Button_Next
            //
            this.Button_Next.Anchor =  ((System.Windows.Forms.AnchorStyles)
                ((System.Windows.Forms.AnchorStyles.Top |
                  System.Windows.Forms.AnchorStyles.Right)));
            this.Button_Next.FlatAppearance.BorderSize = 0;
            this.Button_Next.Location = new System.Drawing.Point(466, 60);
            this.Button_Next.Name = "Button_Next";
            this.Button_Next.Size = new System.Drawing.Size(75, 23);
            this.Button_Next.TabIndex = 9;
            this.Button_Next.Text = "&Next >";
            this.Button_Next.TextImageRelation =
                System.Windows.Forms.TextImageRelation.ImageAboveText;
            this.Button_Next.UseVisualStyleBackColor = true;
            this.Button_Next.Click +=
                new System.EventHandler(this.OnButtonNextClick);
            //
            // Button_Previous
            //
            this.Button_Previous.Anchor = ((System.Windows.Forms.AnchorStyles)
                ((System.Windows.Forms.AnchorStyles.Top |
                  System.Windows.Forms.AnchorStyles.Right)));
            this.Button_Previous.FlatAppearance.BorderSize = 0;
            this.Button_Previous.Location = new System.Drawing.Point(304, 60);
            this.Button_Previous.Name = "Button_Previous";
            this.Button_Previous.Size = new System.Drawing.Size(75, 23);
            this.Button_Previous.TabIndex = 7;
            this.Button_Previous.Text = "< &Previous";
            this.Button_Previous.TextImageRelation =
                System.Windows.Forms.TextImageRelation.ImageAboveText;
            this.Button_Previous.UseVisualStyleBackColor = true;
            this.Button_Previous.Click +=
                new System.EventHandler(this.OnButtonPreviousClick);
            //
            // MenuStrip_Main
            //
            this.MenuStrip_Main.Items.AddRange(
                new System.Windows.Forms.ToolStripItem[] {
            this.MenuItem_File,
            this.MenuItem_Help});
            this.MenuStrip_Main.Location = new System.Drawing.Point(0, 0);
            this.MenuStrip_Main.Name = "MenuStrip_Main";
            this.MenuStrip_Main.Size = new System.Drawing.Size(553, 24);
            this.MenuStrip_Main.TabIndex = 1;
            this.MenuStrip_Main.Text = "menuStrip";
            //
            // MenuItem_File
            //
            this.MenuItem_File.DropDownItems.AddRange(
               new System.Windows.Forms.ToolStripItem[] {
            this.MenuItem_File_Change_Directory,
            this.MenuSeparator_Change_Directory_Exit,
            this.MenuItem_File_Exit});
            this.MenuItem_File.Name = "MenuItem_File";
            this.MenuItem_File.Size = new System.Drawing.Size(35, 20);
            this.MenuItem_File.Text = "&File";
            //
            // MenuItem_File_Change_Directory
            //
            this.MenuItem_File_Change_Directory.Name =
                "MenuItem_File_Change_Directory";
            this.MenuItem_File_Change_Directory.Size =
                new System.Drawing.Size(176, 22);
            this.MenuItem_File_Change_Directory.Text = "&Choose Directory..";
            this.MenuItem_File_Change_Directory.Click +=
                new System.EventHandler(this.OnMenuItemChooseDirectoryClick);
            //
            // MenuSeparator_Change_Directory_Exit
            //
            this.MenuSeparator_Change_Directory_Exit.Name =
                "MenuSeparator_Change_Directory_Exit";
            this.MenuSeparator_Change_Directory_Exit.Size =
                new System.Drawing.Size(173, 6);
            //
            // MenuItem_File_Exit
            //
            this.MenuItem_File_Exit.Name = "MenuItem_File_Exit";
            this.MenuItem_File_Exit.Size = new System.Drawing.Size(176, 22);
            this.MenuItem_File_Exit.Text = "E&xit";
            this.MenuItem_File_Exit.Click +=
                new System.EventHandler(this.On_MenuItem_File_Exit_Click);
            //
            // trackBarTension
            //
            this.trackBarTension.LargeChange = 2;
            this.trackBarTension.Location = new System.Drawing.Point(12, 57);
            this.trackBarTension.Name = "trackBarTension";
            this.trackBarTension.Size = new System.Drawing.Size(187, 45);
            this.trackBarTension.TabIndex = 3;
            this.trackBarTension.Value = 5;
            this.trackBarTension.ValueChanged +=
                new System.EventHandler(this.OnTrackBarTensionValueChanged);
            //
            // labelTension
            //
            this.labelTension.AutoSize = true;
            this.labelTension.Location = new System.Drawing.Point(12, 34);
            this.labelTension.Name = "labelTension";
            this.labelTension.Size = new System.Drawing.Size(79, 13);
            this.labelTension.TabIndex = 2;
            this.labelTension.Text = "Curve Tension:";
            //
            // textBoxTension
            //
            this.textBoxTension.Location = new System.Drawing.Point(154, 31);
            this.textBoxTension.Name = "textBoxTension";
            this.textBoxTension.Size = new System.Drawing.Size(36, 20);
            this.textBoxTension.TabIndex = 4;
            this.textBoxTension.Text = "0.5";
            this.textBoxTension.KeyPress +=
                new System.Windows.Forms.KeyPressEventHandler(
                this.OnTextBoxTensionKeyPress);
            //
            // buttonClearAll
            //
            this.buttonClearAll.Location = new System.Drawing.Point(206, 60);
            this.buttonClearAll.Name = "buttonClearAll";
            this.buttonClearAll.Size = new System.Drawing.Size(75, 23);
            this.buttonClearAll.TabIndex = 6;
            this.buttonClearAll.Text = "Clear &All";
            this.buttonClearAll.UseVisualStyleBackColor = true;
            this.buttonClearAll.Click +=
                new System.EventHandler(this.OnButtonClearAllClick);
            //
            // buttonClearSelected
            //
            this.buttonClearSelected.Location = new System.Drawing.Point(206, 31);
            this.buttonClearSelected.Name = "buttonClearSelected";
            this.buttonClearSelected.Size = new System.Drawing.Size(75, 23);
            this.buttonClearSelected.TabIndex = 5;
            this.buttonClearSelected.Text = "&Clear";
            this.buttonClearSelected.UseVisualStyleBackColor = true;
            this.buttonClearSelected.Click +=
                new System.EventHandler(this.OnButtonClearSelectedClick);
            //
            // buttonSave
            //
            this.buttonSave.Anchor = ((System.Windows.Forms.AnchorStyles)
                ((System.Windows.Forms.AnchorStyles.Top |
                  System.Windows.Forms.AnchorStyles.Right)));
            this.buttonSave.Location = new System.Drawing.Point(385, 60);
            this.buttonSave.Name = "buttonSave";
            this.buttonSave.Size = new System.Drawing.Size(75, 23);
            this.buttonSave.TabIndex = 8;
            this.buttonSave.Text = "&Save";
            this.buttonSave.UseVisualStyleBackColor = true;
            this.buttonSave.Click += new System.EventHandler(this.OnButtonSaveClick);
            //
            // displayPanel
            //
            this.displayPanel.Anchor = ((System.Windows.Forms.AnchorStyles)
                ((((System.Windows.Forms.AnchorStyles.Top |
                    System.Windows.Forms.AnchorStyles.Bottom) |
                    System.Windows.Forms.AnchorStyles.Left) |
                    System.Windows.Forms.AnchorStyles.Right)));
            this.displayPanel.BackColor = System.Drawing.SystemColors.AppWorkspace;
            this.displayPanel.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D;
            this.displayPanel.CurrentSpline = null;
            this.displayPanel.Cursor = System.Windows.Forms.Cursors.Arrow;
            this.displayPanel.FileDirectory = null;
            this.displayPanel.Location = new System.Drawing.Point(12, 92);
            this.displayPanel.Name = "displayPanel";
            this.displayPanel.Size = new System.Drawing.Size(529, 540);
            this.displayPanel.TabIndex = 0;
            this.displayPanel.CurrentSplineChanged +=
                new PrincipalLineTracing.CurrentSplineChangedEvent(
                this.OnDisplayPanelCurrentSplineChanged);
            this.displayPanel.DirectoryChanged +=
                new PrincipalLineTracing.DirectoryChangedEvent(
                this.OnDisplayPanelDirectoryChanged);
            this.displayPanel.ImageChanged +=
                new PrincipalLineTracing.ImageChangedEvent(
                this.OnDisplayPanelImageChanged);
            //
            // FormMain
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(553, 644);
            this.Controls.Add(this.buttonSave);
            this.Controls.Add(this.buttonClearSelected);
            this.Controls.Add(this.buttonClearAll);
            this.Controls.Add(this.textBoxTension);
```

```
        this.Controls.Add(this.displayPanel);
        this.Controls.Add(this.labelTension);
        this.Controls.Add(this.trackBarTension);
        this.Controls.Add(this.Button_Previous);
        this.Controls.Add(this.Button_Next);
        this.Controls.Add(this.MenuStrip_Main);
        this.MinimumSize = new System.Drawing.Size(463, 150);
        this.Name = "FormMain";
        this.Text = "Principal Line Tracing";
        this.MenuStrip_Main.ResumeLayout(false);
        this.MenuStrip_Main.PerformLayout();
        ((System.ComponentModel.ISupportInitialize)
            (this.trackBarTension)).EndInit();
        this.ResumeLayout(false);
        this.PerformLayout();
    }

    private DisplayPanel displayPanel;
    private System.Windows.Forms.Button Button_Next;
    private System.Windows.Forms.Button Button_Previous;
    private System.Windows.Forms.MenuStrip MenuStrip_Main;
    private System.Windows.Forms.ToolStripMenuItem MenuItem_File;
    private System.Windows.Forms.ToolStripMenuItem
        MenuItem_File_Change_Directory;
    private System.Windows.Forms.ToolStripSeparator
        MenuSeparator_Change_Directory_Exit;
    private System.Windows.Forms.ToolStripMenuItem MenuItem_File_Exit;
    private System.Windows.Forms.TrackBar trackBarTension;
    private System.Windows.Forms.Label labelTension;
    private System.Windows.Forms.TextBox textBoxTension;
    private System.Windows.Forms.Button buttonClearAll;
    private System.Windows.Forms.Button buttonClearSelected;
    private System.Windows.Forms.Button buttonSave;

    private const int Max_Path = 260;

    [DllImport("shlwapi", EntryPoint = "PathCompactPathEx")]
    private static extern bool PathCompactPathEx(StringBuilder pszOut, string
        pszSrc, int cchMax, int dwFlags);

    public FormMain() {
        InitializeComponent();
        displayPanel.FileDirectory = string.Format("{0}\\Images",
            Application.StartupPath);
    }

    private void UpdateTitleBar() {
        Text = string.Format("Principal Line Tracing - [{0}/{1}] {2}",
            displayPanel.FileIndex + 1, displayPanel.FileCount,
            Compact_String(displayPanel.FileDirectory, 30));
    }

    private void ResetTension() {
        trackBarTension.Value = 5;
        textBoxTension.Text = string.Format("{0}", 0.5);
    }

    private void OnMenuItemChooseDirectoryClick(object sender, EventArgs e) {
        ChooseImageDirectory();
    }

    private void ChooseImageDirectory() {
        FolderBrowserDialog folderDialog = new FolderBrowserDialog();
        folderDialog.ShowNewFolderButton = false;
        folderDialog.SelectedPath = Application.StartupPath;
        folderDialog.Description = "Select an image directory:";
        if (folderDialog.ShowDialog() == DialogResult.OK) {
            displayPanel.FileDirectory = folderDialog.SelectedPath;
        }
        folderDialog.Dispose();
    }

    private void OnTextBoxTensionKeyPress(object sender, KeyPressEventArgs e) {
        e.Handled = true;
    }

    private void OnTrackBarTensionValueChanged(object sender, EventArgs e) {
```

```
    if (displayPanel.CurrentSpline == null) {
        ResetTension();
        return;
    }
    float tension = (float)trackBarTension.Value / 10.0F;
    textBoxTension.Text = string.Format("{0}", tension);
    displayPanel.CurrentSpline.Tension = tension;
    displayPanel.Refresh();
}

private void OnButtonClearSelectedClick(object sender, EventArgs e) {
    if (displayPanel.CurrentSpline == null) {
        return;
    }
    displayPanel.RemoveSpline(displayPanel.CurrentSpline);
    ResetTension();
}

private void OnButtonClearAllClick(object sender, EventArgs e) {
    displayPanel.RemoveAllSplines();
    ResetTension();
}

private void OnButtonNextClick(object sender, EventArgs e) {
    displayPanel.SaveSplines(string.Format("{0}\\Data",
        Application.StartupPath));
    displayPanel.SaveImages(string.Format("{0}\\Labelled",
        Application.StartupPath));
    displayPanel.ShowNextImage();
}

private void OnButtonSaveClick(object sender, EventArgs e) {
    displayPanel.SaveSplines(string.Format("{0}\\Data",
        Application.StartupPath));
}

private void OnButtonPreviousClick(object sender, EventArgs e) {
    displayPanel.SaveSplines(string.Format("{0}\\Data",
        Application.StartupPath));
    displayPanel.ShowPreviousImage();
}

private void OnDisplayPanelDirectoryChanged(object sender, EventArgs e) {
    UpdateTitleBar();
}

private void OnDisplayPanelImageChanged(object sender, EventArgs e) {
    displayPanel.OpenSplines(string.Format("{0}\\Data",
        Application.StartupPath));
    UpdateTitleBar();
}

private void OnDisplayPanelCurrentSplineChanged(object sender, EventArgs e) {
    if (displayPanel.CurrentSpline != null) {
        trackBarTension.Value =
            (int)(displayPanel.CurrentSpline.Tension * 10);
        trackBarTension.Enabled = true;
        textBoxTension.Enabled = true;
        buttonClearSelected.Enabled = true;
    } else {
        trackBarTension.Enabled = false;
        textBoxTension.Enabled = false;
        buttonClearSelected.Enabled = false;
    }
}

public static string Compact_String(string Path, int Length) {
    StringBuilder Buffer;
    bool Success;
    if (Path.Length > Max_Path) {
        return Path;
    }
    if (Length > Max_Path) {
        Length = Max_Path;
    }
    Buffer = new StringBuilder(Max_Path, Max_Path);
    Success = PathCompactPathEx(Buffer, Path, Length, 0);
```

```
            if (Success) {
                return Buffer.ToString();
            }
            return null;
        }

        private void On_MenuItem_File_Exit_Click(object sender, EventArgs e) {
            Close();
        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace FlexionCreaseTracing
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main() {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FormMain());
        }
    }
}
```

Converting cardinal splines to points

CardinalSplinesToPoints.exe is a C# application that can be compiled against the .NET framework using the C# compiler (csc.exe) with the command:

```
csc.exe /target:exe /unsafe
/out:CardinalSplinesToPoints.exe Line.cs LineList.cs
UnsafeBitmap.cs Program.cs
```

The source files for CardinalSplinesToPoints.exe are Line.cs, LineList.cs, UnsafeBitmap.cs, and Program.cs.

Line.cs

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;

namespace CardinalSplinesToPoints
{
    public class Line
    {
        private Point[] controlPoints;

        public Point[] ControlPoints {
            get { return this.controlPoints; }
        }

        public GraphicsPath Path {
            get {
                GraphicsPath graphicsPath = new GraphicsPath();
                if (this.ControlPoints.Length > 1) {
                    graphicsPath.AddCurve(this.ControlPoints, 0.5f);
                }
                return graphicsPath;
            }
        }

        public Line() {
        }

        public void AddControlPoint(Point point) {
            if (this.controlPoints == null) {
                this.controlPoints = new Point[1];
            } else {
                this.controlPoints = ResizeArray(this.controlPoints,
                    this.controlPoints.Length + 1);
            }
            this.controlPoints[this.controlPoints.Length - 1] = point;
        }

        public static Point[] ResizeArray(Array source, int size) {
            Type elementType = source.GetType().GetElementType();
            Array newArray = Array.CreateInstance(elementType, size);
            int preserveLength = Math.Min(source.Length, size);
            if (preserveLength > 0) {
                Array.Copy(source, newArray, preserveLength);
            }
            return newArray as Point[];
        }
    }
}
```

LineList.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;

namespace CardinalSplinesToPoints
{
    class LineList
    {
        private string listName;

        public string ListName {
            get { return this.listName; }
            set { this.listName = value; }
        }

        public List<Line> Lines {
            get { return this.lines; }
        }
        private List<Line> lines;

        public LineList() {
            this.lines = new List<Line>();
        }

        public LineList(string listName) : this() {
            this.ListName = listName;
        }

        public void ToFile(string filePath) {
            try {
                using (StreamWriter writer = File.CreateText(filePath)) {
                    writer.WriteLine(this.ListName);
                    foreach (Line line in this.Lines) {
                        writer.WriteLine("Line");
                        if (line.ControlPoints != null) {
                            foreach (Point point in line.ControlPoints) {
                                writer.WriteLine("{0},{1}", point.X, point.Y);
                            }
                        }
                    }
                }
            } catch (PathTooLongException error) {
                throw (error);
            } catch (DirectoryNotFoundException error) {
                throw (error);
            } catch (IOException error) {
                throw (error);
            }
        }
    }
}
```

Program.cs

```
using System;
using System.IO;
using System.Drawing;

namespace CardinalSplinesToPoints
{
    class Program
    {
        static void Main(string[] args) {
            if (args.Length < 3) {
                Console.WriteLine("Converts cardinal splines to " +
                    "a series of points.");
                Console.WriteLine("Usage: CardinalSplinesToPoints <images>" +
                    "<splines> <output>");
                return;
            }
            FileInfo[] images;
            if (!GetImageFiles(args[0], out images)) {
                return;
            }
            foreach (FileInfo file in images) {
                Size size = GetImageSize(file.FullName);
                if (size.Equals(Size.Empty)) {
                    return;
                }
                LineList splines = null;
                if (!ReadLines(string.Format("{0}\\{1}.txt", args[1],
                    file.Name), out splines)) {
                    return;
                }
                LineList lines = new LineList(file.Name);
                foreach (Line spline in splines.Lines) {
                    UnsafeBitmap image = new UnsafeBitmap(size);
                    image.DrawPath(spline.Path);
                    image.LockBitmap();
                    Line line = new Line();
                    for (int x = 0; x < image.PixelSize.X; x++) {
                        for (int y = 0; y < image.PixelSize.Y; y++) {
                            UnsafeBitmap.PixelData data =
                                image.GetPixel(new Point(x, y));
                            if (data.red > 0) {
                                line.AddControlPoint(new Point(x, y));
                            }
                        }
                    }
                    lines.Lines.Add(line);
                    image.UnlockBitmap();
                }
                if (Directory.Exists(args[2])) {
                    lines.ToFile(string.Format("{0}\\{1}.txt", args[2],
                        file.Name));
                }
            }
        }

        private static bool ReadLines(string path, out LineList lines) {
            lines = new LineList();
            if (!File.Exists(path)) {
                return false;
            }
            Console.Write("Reading {0}.. ", Path.GetFileName(path));
            try {
                Line line = null;
                using (StreamReader stream = new StreamReader(path)) {
                    string fileLine = lines.ListName = stream.ReadLine();
                    while ((fileLine = stream.ReadLine()) != null) {
                        if (!fileLine.Contains(",")) {
                            lines.Lines.Add((line = new Line()));
                        } else {
                            if (line != null) {
                                try {
                                    int x = int.Parse(fileLine.Substring(0,
                                        fileLine.IndexOf(',')));
                                    int y = int.Parse(fileLine.Substring(
                                        fileLine.IndexOf(',') + 1));
                                    line.AddControlPoint(new Point(x, y));
                                } catch (FormatException) {
                                    Console.WriteLine("skipping '{0}'", line);
                                    Console.Write("Reading {0}.. ",
                                        Path.GetFileName(path));
                                } catch (OverflowException) {
                                    Console.WriteLine("skipping '{0}'", line);
                                    Console.Write("Reading {0}.. ",
                                        Path.GetFileName(path));
                                }
                            }
                        }
                    }
                }
            }
            catch (IOException) {
                Console.WriteLine("failed (unexpected io error)");
                return false;
            }
            Console.WriteLine("done");
            return true;
        }

        private static Size GetImageSize(string path) {
            Console.Write("Reading image.. ");
            Try {
                using (Bitmap image = (Bitmap)Bitmap.FromFile(path)) {
                    return new Size(image.Width, image.Height);
                }
            } catch (OutOfMemoryException) {
                Console.WriteLine("failed (out of memory)");
            } catch (FileNotFoundException) {
                Console.WriteLine("failed (file not found)");
            } catch (IOException) {
                Console.WriteLine("failed (unexpected io error)");
            }
            return Size.Empty;
        }

        private static bool GetImageFiles(string path, out FileInfo[] files) {
            Console.Write("Reading images.. ");
            files = null;
            if (!Directory.Exists(path)) {
                Console.WriteLine("failed (the directory does not exist)");
                return false;
            }
            try {
                DirectoryInfo info = new DirectoryInfo(path);
                try {
                    files = info.GetFiles("*.bmp");
                } catch (DirectoryNotFoundException) {
                    Console.WriteLine("failed (the directory does not exist)");
                    return false;
                }
            }
            catch (PathTooLongException) {
                Console.WriteLine("failed (the directory is too long)");
                return false;
            }
            Console.WriteLine("done");
            return true;
        }
    }
}
```

Normalising palmar flexion creases

ConvLines.m

```
function convlines( in, out, samples )
%CONVLINES Convert lines
%   Convert lines using linspacearc to even spaced samples
    files = dir(in);
    for i = 1:numel(files)
        if files(i).isdir ~= true
            lx = []; ly = [];
            [file, lines] = importlines([in, files(i).name]);
            for j = 1:numel(lines)
                source = lines{j};
                [x, y] = linspacearc(source(:, 1), source(:, 2), samples);
                lx = [lx, x]; %#ok<*AGROW>
                ly = [ly, y];
            end
            fprintf('writing %s\n', [out, file, '.txt']);
            output = fopen([out, file, '.txt'], 'w');
            fprintf(output, '%G,%G\r\n', [lx; ly]);
            fclose(output);
        end
    end
end
```

LinSpaceArc.m

```
function [x2,y2] = linspacearc(x,y,n)
    m = length(x);
    t = linspace(0,1,m);
    ppx = spline(t,x);
    ppy = spline(t,y);
    dppx = pp_deriv(ppx);
    dppy = pp_deriv(ppy);
    integrand = @(tt) sqrt(ppval(dppx,tt).^2 + ppval(dppy,tt).^2);
    arc_length = quadgk(integrand,0,1);
    s = linspace(0,arc_length,n);
    inv_arc_len = @(arc,est) fzero(@(u)(quadgk(integrand,0,u)) - arc,est);
    t2 = zeros(1,n);
    t2(1) = inv_arc_len(s(1),0);
    for i = 2:n
        t2(i) = inv_arc_len(s(i),t2(i-1));
    end
    x2 = ppval(ppx,t2);
    y2 = ppval(ppy,t2);

function dpp = pp_deriv(pp)
    % pp_deriv: derivative of piecewise polynomial (pp)
    dpp = pp;
    n = pp.order;
    dpp.coefs = bsxfun(@times,n-1:-1:1,pp.coefs(:,1:n-1));
    dpp.order = n - 1;
```

Modifying and comparing palmar flexion creases

ModLine.m

```
function [ line ] = modline( source, noise )
%MODLINE Modifies the line specified by SOURCE with random
%   rotation, translation, and noise. The amount of noise (independent
%   variation of each point in SOURCE) is specified by NOISE.
    angle = randint(1, 1, [0 360]); % random angle 0 to 360
    linex = source(:, 1) .* cos(angle) - source(:, 2) .* sin(angle);
    liney = source(:, 1) .* sin(angle) + source(:, 2) .* cos(angle);
    % random translation (of all points) -50 to 50
    xtran = randint(1, 1, [-50 50]);
    ytran = randint(1, 1, [-50 50]);
    linex = linex + xtran;
    liney = liney + ytran;
    % add random translation to each point
    for t = 1:numel(linex)
```

```
        xtran = randint(1, 1, [-noise noise]);
        ytran = randint(1, 1, [-noise noise]);
        linex(t) = linex(t) + xtran;
        liney(t) = liney(t) + ytran;
    end
    line = [linex liney];
end
```

CmpLines.m

```
function [ distance ] = cmplines( A, B )
%CMPLINES Compare two lines A and B
%   Find the Euclidean distance between two sets of points
%   after they have been normalised using Procrustes analysis.
%   DISTANCE is the sum of Euclidean distances between each nearest
%   point in A and B.
    [ssm BT] = procrustes(A, B, 'reflection', false, ...
        'scaling', false);
    clear ssm;
    A = A'; B = B'; BT = BT';
    % get the distance between A and BT
    index = nearestneighbour(A, BT);
    abt = A - BT(:, index);
    abt = abt .* abt;
    abt = sqrt(abt(1, :) + abt(2, :));
    abt = sum(abt);
    % get the distance between A and B
    index = nearestneighbour(A, B);
    ab = A - B(:, index);
    ab = ab .* ab;
    ab = sqrt(ab(1, :) + ab(2, :));
    ab = sum(ab);
    % find the least distance
    distance = min(abt, ab);
end
```

GetDist.m

```
tic;
directory = ['..\Normalised'];
verbose = 1;
transformations = 5; % the number of times to transform B
mindistance = Inf; % the minimum distance between incorrect palms
maxdistance = 0; % the maximum distance between correct palms
avgcdistance = 0; % the average distance between correct palms
avgidistance = 0; % the average distance between incorrect palms
avgcsum = 0; avgisum = 0; ccp = 0; icp = 0;
% get file list
files = dir(fullfile(directory, '*.txt'));
% for each file
for i = 1:numel(files)
    % read A
    A = importdata(fullfile(directory, files(i).name));
    % for each file
    for j = 1:numel(files)
        % read B
        B = importdata(fullfile(directory, files(j).name));
        % for 1 to nr of transformations
        for k = 1:transformations
            % modify A
            AR = modline(A, 8);
            for v = 1:transformations
                % modify B
                BR = modline(B, 8);
                % compare A to transformed B
                distance = cmplines(AR, BR);
                if strncmpi(files(i).name, files(j).name, 7) == 1
                    % from the same palm
                    ccp = ccp + 1;
                    avgcsum = avgcsum + distance;
                    avgcdistance = avgcsum / ccp;
                    if distance > maxdistance
                        % record the maximum distance
                        maxdistance = distance;
                        if verbose > 0
                            fprintf('Distances: %.2f / %.2f c: %.2f\n', ...
                                maxdistance, mindistance, avgcdistance);
                        end
                    end
                else
                    icp = icp + 1;
                    avgisum = avgisum + distance;
                    avgidistance = avgisum / icp;
                    % from different palms
                    if distance < mindistance
                        % record the minimum distance
                        mindistance = distance;
                        if verbose > 0
                            fprintf('Distances: %.2f / %.2f i: %.2f\n', ...
                                maxdistance, mindistance, avgidistance);
                        end
                    end
                end
            end
        end
    end
end
clear i j A B index distance directory files;
clear xtran ytran k bry brx angle AR BR t verbose;
clear avgisum avgcsum;
toc;
```

CmpDb.m

```
tic;
directory = ['..\Normalised'];
verbose = 1;
threshold = 0; % distance threshold
transformations = 5; % the number of transforms
cp = 0; % number of comparisons
ga = 0; % genuine acceptance rate
fa = 0; % false acceptance rate
fr = 0; % false negative rate
% get file list
files = dir(fullfile(directory, '*.txt'));
% for each file
for i = 1:numel(files)
    % read A
    A = importdata(fullfile(directory, files(i).name));
    % for each file
    for j = 1:numel(files)
        % read B
        B = importdata(fullfile(directory, files(j).name));
        % for 1 to nr of transformations
        for k = 1:transformations
            % modify A
            AR = modline(A, 8);
            for v = 1:transformations
                cp = cp + 1;
                % modify B
                BR = modline(B, 8);
                % compare A to transformed B
                distance = cmplines(AR, BR);
                % get ga, fa, or fn rate
                if distance <= threshold
                    % distance is below threshold
                    if strncmpi(files(i).name, files(j).name, 7) == 1
                        % and from the same palm: genunine acceptance
                        ga = ga + 1;
                        if verbose > 1
                            fprintf(['Genuine acceptance: ', ...
                                '%.2f (%s to %s)\n'], distance, ...
                                files(i).name, files(j).name);
                        end
                    else
                        % and from two different palms: false acceptance
                        fa = fa + 1;
                        if verbose > 0
                            fprintf(['False acceptance: ', ...
                                '%.2f (%s to %s)\n'], distance, ...
                                files(i).name, files(j).name);
                        end
                    end
                else
                    % distance is above threshold
                    if strncmpi(files(i).name, files(j).name, 7) == 1
                        % and from the same palm: false rejection
                        fr = fr + 1;
                        if verbose > 0
                            fprintf(['False rejection: ', ...
                                '%.2f (%s to %s)\n'], distance, ...
                                files(i).name, files(j).name);
                        end
                    end
                end
            end
        end
    end
end
clear i j A B index distance directory files;
clear xtran ytran k bry brx angle BR t verbose;
toc;
```

Automated palmprint identification

Palmprint pre-processing

PeakDet.m

```
maxtab = [];
mintab = [];
v = v(:);
mn = Inf; mx = -Inf;
mnpos = NaN; mxpos = NaN;
lookformax = 1;
for i=1:length(v)
  this = v(i);
  if this > mx, mx = this; mxpos = i; end
  if this < mn, mn = this; mnpos = i; end
  if lookformax
    if this < mx-delta
      maxtab = [maxtab ; mxpos mx];
      mn = this; mnpos = i;
      lookformax = 0;
    end
  else
    if this > mn+delta
      mintab = [mintab ; mnpos mn];
      mx = this; mxpos = i;
      lookformax = 1;
    end
  end
end
```

FindLargestRectanges.m

```
if (nargin<2)
  crit = [1 1 0];
end
if (nargin<3)
  minSize = [1 1];
end
p = crit;
[nR nC] = size(I);
if (minSize(1)<1), minSize(1)= floor(minSize(1)*nR); end
if (minSize(2)<1), minSize(2)= floor(minSize(2)*nC); end
if (max(I(:)) - min(I(:))==1),
  S = FindLargestSquares(I);
else
  S = I;
end
n = max(S(:));
```

```
W = S;
H = S;
C = ((p(1)+p(2)) + p(3)*S) .* S;
d = round((3*n)/4);
minH = max(min(minSize(1), d),1);
minW = max(min(minSize(2), d),1);
hight2width = zeros(n+1,1);
for r = 1 : nR
  hight2width(:) = 0;
  for c = nC: -1 : 1
    s = S(r,c);
    if (s>0
      MaxCrit = C(r,c
      for hight = s:-1:
        width = hight2width(hight
        width = max(width+1,s);
        hight2width(hight) = width;
        Crit = p(1)*hight + p(2)*width + p(3)*width*hight;
        if (Crit>MaxCrit),
          MaxCrit = Crit;
          W(r,c)  = width;
          H(r,c)  = hight;
        end
      end
      C(r,c)  = MaxCrit;
    end
    hight2width((s+1):end) = 0;
  end
end
clear hight2width
width2hight = zeros(n+1,1);
for c = 1 : nC
  width2hight(:) = 0;
  for r = nR: -1 : 1
    s = S(r,c);
    if (s>0)
      MaxCrit = C(r,c);
      for width = s:-1:1
        hight = width2hight(width);
        hight = max(hight+1,s);
        width2hight(width) = hight;
        Crit = p(1)*hight + p(2)*width + p(3)*width*hight;
        if (Crit>MaxCrit),
          MaxCrit = Crit;
          W(r,c)  = width;
          H(r,c)  = hight;
        end
      end
      C(r,c)  = MaxCrit;
    end
    width2hight((s+1):end) = 0;
  end
end
CC = C;
CC( H<minH | W<minW ) = 0;
[tmp pos] = max(CC(:));
if (isempty(pos)), [tmp pos] = max(C(:)); end
[r c] = ind2sub(size(C), pos);
M = false(size(C));
M( r:(r+H(r,c)-1), c:(c+W(r,c)-1) ) = 1;
```

FindLargestSquares.m

```
[nr nc] = size(I);
S = double(I);
for r=(nr-1):-1:1
  for c=(nc-1):-1:1
    if (S(r,c))
      a = S(r  ,c+1);
      b = S(r+1,c  );
      d = S(r+1,c+1);
      S(r,c) = min([a b d]) + 1;
    end
  end
end
```

PreProcess.m

```
%Load image
image = imread(filePath);
level = graythresh(image);
binary = im2bw(image,level);
%Preprocess the image.
label = bwlabel(binary);
regions = regionprops(label, 'Area', 'BoundingBox', 'FilledImage');
[area, index] = max([regions.Area]);
image = imcrop(image, regions(index).BoundingBox);
binary = regions(index).FilledImage;
boundary = bwboundaries(binary, 'noholes');
boundary = boundary{1};
[maxPeaks, minPeaks] = peakdet(boundary(:, 2), preProcessingPeakSize);
[imageHeight, imageWidth] = size(image);
peaks = boundary(maxPeaks(:, 1), 2);
maxPeak = max(peaks(peaks < 200));
regionOfInterest = [maxPeak 1 (imageWidth - maxPeak - 20) imageHeight];
image = imcrop(image, regionOfInterest);
image = imrotate(image, -90);
binary = image > preProcessingThreshold;
[S H W] = FindLargestRectangles(binary, [0 0 1]);
[tmp pos] = max(S(:));
[r c] = ind2sub(size(S), pos);
image = imcrop(image, [c, r, W(r, c), H(r, c)]);
```

Palmar flexion crease identification

FindCreases.py

```
#!/usr/bin/python

try:
    import pygtk
    pygtk.require('2.0')
except:
    pass

try:
    import gobject
    import gtk
    import gtk.glade
except:
    print 'Error: GTK is not installed'
    sys.exit(1)

import os
import sys
import glob
import Image
import numpy
import scipy.signal.signaltools

class Viewer(object):
    def __init__(self, image_path):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.connect("destroy", self.destroy_event)
        self.window.set_border_width(10)
        self.button = gtk.Button()
        self.button.connect_object("clicked", gtk.Widget.destroy, self.window)
        self.image = gtk.Image()
        self.image.set_from_file(image_path)
        self.image.show()
        self.button.add(self.image)
        self.window.add(self.button)
        self.button.show()
        self.window.show()

    def destroy_event(self, widget, data=None):
        gtk.main_quit()

    def main(self):
        gtk.main()
```

```
def array_to_image(data):
    """Convert an image array to an Image.Image object."""
    cmin = data.min()
    cmax = data.max()
    scale = 255 * 1.0 / (cmax - cmin or 1)
    bytedata = ((data * 1.0 - cmin)
                * scale + 0.4999).astype(numpy.uint8)
    bytedata = bytedata + numpy.cast[numpy.uint8](0)
    return Image.fromarray(bytedata)

def image_to_array(im):
    """Convert a Image.Image object to a float image array."""
    if im.mode == 'RGB':
        imr, img, imb = im.split()
        im = numpy.asarray(imb, numpy.float)
    elif im.mode == 'L':
        im = numpy.asarray(im, numpy.float)
    im = im / 256
    return im

def get_binomial_coefficients(size):
    """Returns a vector of binomial coefficients of order
    (size - 1)."""
    if size < 2:
        print "Size must be larger than 1."
    kernel = numpy.array([0.5, 0.5])
    for n in range(0, size - 2):
        kernel = numpy.convolve(numpy.array([0.5, 0.5]), kernel)
    return kernel.reshape(-1, 1)

def convolve(large, small):
    """Convolution of two matrices, with boundaries handled via
    reflection about the edge pixels. Result will be of size of the
    large matrix. The origin of the small matrix is assumed to be its
    center. """
    ly = large.shape[0]
    lx = large.shape[1]
    sy = small.shape[0]
    sx = small.shape[1]
    sy2 = numpy.floor((sy + 0 - 1) / 2)
    sx2 = numpy.floor((sx + 0 - 1) / 2)
    clarge1 = numpy.concatenate((numpy.concatenate(
              (large[(sy-sy2)-1:0:-1,(sx-sx2)-1:0:-1],
              large[(sy-sy2)-1:0:-1,:]), axis=1),
              large[(sy-sy2)-1:0:-1,lx-2:(lx-sx2)-2:-1]), axis=1)
    clarge2 = numpy.concatenate((numpy.concatenate(
              (large[:,(sx-sx2)-1:0:-1], large), axis=1),
              large[:,lx-2:(lx-sx2)-2:-1]), axis=1)
    clarge3 = numpy.concatenate((numpy.concatenate(
              (large[ly-2:(ly-sy2)-2:-1,(sx-sx2)-1:0:-1],
              large[ly-2:(ly-sy2)-2:-1,:]), axis=1),
              large[ly-2:(ly-sy2)-2:-1,lx-2:(lx-sx2)-2:-1]), axis=1)
    clarge = numpy.concatenate((numpy.concatenate(
             (clarge1, clarge2), axis=0), clarge3), axis=0)
    return scipy.signal.signaltools.convolve2d(clarge, small,
                                               mode='valid')

def correlate(im, filt, step):
    """Compute correlation of matrices im with filt followed by
    downsampling. These arguments should be 1d or 2d matrices, and im
    must be larger (in both dimensions) than filt. Downsampling
    factors are determined by step."""
    start = numpy.array([1, 1])
    stop = numpy.array(im.shape)
    tmp = convolve(im, filt)
    return tmp[start[0]-1:stop[0]:step[0],start[1]-1:stop[1]:step[1]]

def blur_and_downsample(im, levels=1):
    """Blur and down sample an image IM. The blurring is done with a
    symmetric quadrature mirror filter (QMF9) kernel. The downsampling
    is always by 2 in each direction."""
    filt = numpy.array([0.02807382, -0.060944743, -0.073386624,
                        0.41472545, 0.7973934,0.41472545, -0.073386624,
                        -0.060944743, 0.02807382]).reshape(-1,1)
    filt = filt / filt.sum(axis=0)
    res = correlate(im, filt, numpy.array([2, 1]))
    return correlate(res, filt.reshape(1, -1), numpy.array([1, 2]))

def find_vertical_path(im):
    height, width = im.shape
    cost = numpy.zeros((width, height), numpy.float)
    for y in range(1, height):
        for x in range(0, width):
            costUp = cost[x, y - 1]
            if x > 0:
                costUpLeft = cost[x - 1, y - 1]
            else:
                costUpLeft = 1000
            if x < width - 1:
                costUpRight = cost[x + 1, y - 1]
            else:
                costUpRight = 1000
            cost[x, y] = min(costUp, min(costUpLeft, costUpRight)) \
            + im[y, x]
    vpath = [(0, 0) for i in range(height)]
    best = 1000
    y = height - 1
    for x in range(0, width):
        if cost[x, y] < best:
            vpath[y] = (x, y)
            best = cost[x, y]
    x = vpath[y][0]
    for y in range(height - 1, -1, -1):
        vpath[y] = (x, y)
        best = cost[x, y]
        if x > 1 and cost[x - 1, y] <= best:
            vpath[y] = (x - 1, y)
            best = cost[x - 1, y]
        if x < width - 1 and cost[x + 1, y] <= best:
            vpath[y] = (x + 1, y)
            best = cost[x + 1, y]
        x = vpath[y][0]
    return vpath

def find_horizontal_path(im):
    height, width = im.shape
    cost = numpy.zeros((width, height), numpy.float)
    for x in range(1, width):
        for y in range(0, height):
            costRight = cost[x - 1, y]
            if y > 0:
                costRightUp = cost[x - 1, y - 1]
            else:
                costRightUp = 1000
            if y < height - 1:
                costRightDown = cost[x - 1, y + 1]
            else:
                costRightDown = 1000
            cost[x, y] = min(costRight, min(costRightUp, costRightDown)) \
            + im[y, x]
    hpath = [(0, 0) for i in range(width)]
    best = 1000
    x = width - 1
    for y in range(0, height):
        if cost[x, y] < best:
            hpath[x] = (x, y)
            best = cost[x, y]
    y = hpath[x][1]
    for x in range(width - 1, -1, -1):
        hpath[x] = (x, y)
        best = cost[x, y]
        if y > 1 and cost[x, y - 1] <= best:
            hpath[x] = (x, y - 1)
            best = cost[x, y - 1]
        if y < height - 1 and cost[x, y + 1] <= best:
            hpath[x] = (x, y + 1)
            best = cost[x, y + 1]
        y = hpath[x][1]
    return hpath

def remove_path(array, points, axis=0):
    """Removes points from array across axis (0 for horizontal, 1
    for vertical). If the number of points is incorrect, the array
    is returned unchanged."""
    h, w = array.shape
    if len(points) <> (axis and w or h):
        return array
    array = array.reshape(1, h * w, order=(axis and 'FORTRAN' or 'C'))
    for p in reversed(points):
        if p[1] >= h: p = (p[0], h - 1)
        if p[0] >= w: p = (w - 1, p[1])
        array = numpy.delete(array,
                (axis and p[1] + (p[0] * h) or (p[1] * w) + p[0]))
    h, w = (axis and (h - 1, w) or (h, w - 1))
    return array.reshape(h, w, order=(axis and 'FORTRAN' or 'C'))

def crop(array, margin):
    """Cut a margin (top, bottom, left, right) from array."""
    h, w = array.shape
    return array[margin[0]:h - margin[1], margin[2]:w - margin[3]]

def get_files(pattern, search_path, path_sep=os.pathsep):
    """Given a search path, yield all files matching pattern."""
    for path in search_path.split(path_sep):
        for match in glob.glob(os.path.join(path, pattern)):
            yield match

def display_image(image):
    """Show the image in a new GTK window."""
    if not isinstance(image, Image.Image):
        image = array_to_image(image)
    image.save('/tmp/file.bmp')
    Viewer('/tmp/file.bmp').main()

def create_path_image(im, hpath=None, vpath=None, hcolour=(255, 0, 0), \
vcolour=(0, 255, 0)):
    import ImageDraw
    ''' Draw HPATH and VPATH on IM. '''
    if hpath is None and vpath is None:
        return im
    if im.mode != 'RGB':
        im = im.convert('RGB')
    draw = ImageDraw.Draw(im)
    if hpath is not None:
        draw.point(hpath, fill=hcolour)
    if vpath is not None:
        draw.point(vpath, fill=vcolour)
    del draw
    return im

def find_and_remove_vertical_path(image, cuts, reslice_limit=5):
    paths = []
    cut = 0
    while cut < cuts:
        path = find_vertical_path(image)
        if(check_path_overlap(paths, path, overlap_limit)):
            image = remove_path(image, path, axis=0)
            paths.append(path)
            cut = cut + 1
    return (image, paths)

def find_and_remove_horizontal_path(image, slices, overlap_limit=5):
    paths = []
    cut = 0
    while cut < cuts:
        path = find_horizontal_path(image)
        if(check_path_overlap(paths, path, overlap_limit)):
            image = remove_path(image, path, axis=1)
            paths.append(path)
            cut = cut + 1
    return (image, paths)

def main(image):
    filter = get_binomial_coefficients(5)
    filter = filter * filter.reshape(1, -1)
    image = image_to_array(image)
    image = blur_and_downsample(image)
    image = image - convolve(image, filter)
    image = convolve(image, filter)
    image = crop(image, get_crop_size(image))
    image, vertical_lines = find_and_remove_vertical_path(image, 3)
    image, horizontal_lines = find_and_remove_horizontal_path(image, 4)
    return [horizontal_lines, vertical_lines]

def get_image(filename):
    image = None
    try:
        image = Image.open(filename)
    except IOError:
        sys.exit(1)
    return image

def print_data(filename, data):
    sys.stdout = open('%s' % filename, 'w')
    print os.path.basename(filename)
    for orientations in data:
        for lines in orientations:
            for points in lines:
                print '%s,%s' % (points[0], points[1])
    sys.stdout = sys.__stdout__

if __name__ == '__main__':
    files = get_files('*.bmp', '../images')
    for f in sorted(files):
        print_data('%s.txt' % f, main(get_image(f))
```

Palmar flexion crease matching

KdTree.py

```
import sys
import os
import glob
import math
import csv

class kdtree:
    def __init__(self,dim=2,index=0):
        self.dim = dim
        self.index = index
        self.split = None

    def add_point(self, p):
        if self.split is None:
            self.split = p
            self.left = kdtree(self.dim, (self.index + 1) % self.dim)
            self.right = kdtree(self.dim, (self.index + 1) % self.dim)
        elif self.split[self.index] < p[self.index]:
            self.left.addPoint(p)
        else:
            self.right.addPoint(p)

    def nearest_neighbor(self, q, maxdist):
        """Find pair (d, p) where p is nearest neighbor and d is squared
        distance to p."""
        solution = (maxdist, None)
        if self.split is not None:
            solution = min(solution, (dist2(self.split, q), self.split))
            d2split = (self.split[self.index] - q[self.index]) ** 2
            if self.split[self.index] < q[self.index]:
                solution = min(solution,
                    self.left.nearest_neighbor(q, solution[0]))
                if d2split < solution[0]:
                    solution = min(solution,
                        self.right.nearest_neighbor(q, solution[0]))
            else:
                solution = min(solution,
                    self.right.nearest_neighbor(q, solution[0]))
                if d2split < solution[0]:
                    solution = min(solution,
                        self.left.nearest_neighbor(q, solution[0]))
        return solution

def dist2(p,q):
    """Return the squared distance between p and q."""
    d = 0
    for i in range(len(p)):
        d += (p[i] - q[i]) ** 2
    return d

def get_files(pattern, search_path, path_sep=os.pathsep):
    """Given a search path, yield all files matching pattern."""
    for path in search_path.split(path_sep):
        for match in glob.glob(os.path.join(path, pattern)):
            yield match

def get_points(file_path):
    """Given a csv file, yield all rows where field is above limit."""
    file = open(file_path, 'rb')
    reader = csv.reader(file, delimiter=',', quoting=csv.QUOTE_NONE)
    try:
        for row in reader:
            if len(row) >= 2:
                yield (int(row[0]), int(row[1]))
    finally:
        file.close()

def write_comparison_data(file_path, data):
    """Write comparison data to file."""
    file = open(file_path, 'a')
    try:
        file.write(data)
    finally:
        file.close()

def compare_data(points1, points2, distance):
    """Return a match percentage for two data sets."""
    tree = kdtree()
    for point in points1:
        tree.add_point(point)
    total_points = 0
    matched_points = 0
    for point in points2:
        d, q = tree.nearest_neighbor(point, 1000)
        if math.sqrt(d) < distance:
            matched_points += 1
        total_points += 1
    return (matched_points / total_points) * 100

if __name__ == '__main__':
    input1 = '../automatic'
    input2 = '../manual'
    output = '../output'
    for file1 in sorted(get_files('*.txt', input1)):
        head, file1_tail = os.path.split(file1)
        points1 = list(get_points(file1))
        sys.stdout = open('%s/%s' % (output, os.path.basename(file1)), 'w')
        for file2 in sorted(get_files('*.txt', input2)):
            head, file2_tail = os.path.split(file2)
            rating = compare_data(points1, list(get_points(file2)))
            print '%s,%s,%s' % (file1_tail, file2_tail, rating)
        sys.stdout = sys.__stdout__"""
```

Appendix B

This appendix shows the correspondence with authors of existing palmar flexion crease recognition methods, who were contacted regarding access to their algorithms. The table below lists the methods that are suitable for comparison, and the request and response details.

Method	Request	Request Response	Follow Up Request	Follow Up Response
Kung *et al.* (1995)	[a]			
Rodrigues *et al.* (1996)	[a]			
Boles and Chu (1996)	23/07/2011	Yes, but the algorithm is no longer available.		
Wu *et al.* (2002a)	23/07/2011	Delivery failed. Email quota exceeded.	21/08/2011	None
Wu *et al.* (2004a)	23/07/2011	Delivery failed. Email quota exceeded.	21/08/2011	None
Wu *et al.* (2004b)	23/07/2011	Delivery failed. Email quote exceeded.	21/08/2011	None
Liu and Zhang (2005)	23/07/2011	Delivery failed. Unknown user.	21/08/2011	None
Li and Leung (2006)	23/07/2011	None	21/08/2011	None
Wu *et al.* (2006)	23/07/2011	Delivery failed. Email quota exceeded.	21/08/2011	None
Huang *et al.* (2008)	23/07/2011	None	21/08/2011	None
Jia *et al.* (2008)	23/07/2011	None	21/08/2011	None

[a] Contact details not available.

Correspondence with Professor Boles

RE: Flexion crease algorithm comparison

Wageeh Boles [w.boles@qut.edu.au]
Sent:25 July 2011 00:17
To: Cook, Tom

Dear Tom,

Thanks for your message. Unfortunately, I no longer have the code.
All the best for your research.

Regards,
Wageeh

From: Cook, Tom [mailto:t.cook@wlv.ac.uk]
Sent: 23 July 2011 11:18 PM
To: Wageeh Boles
Subject: Flexion crease algorithm comparison

Dear Professor Boles

I am a research student from the University of Wolverhampton, United Kingdom, and would like to perform a comparison of palmar flexion crease identification algorithms for my PhD thesis.

Following your publication, 'personal identification using images of the human palm' in Proceedings of the IEEE Region 10 Annual Conference on Speech and Image Technologies for Computing and Telecommunications, I would like to include your work in my comparison, and would be grateful if you could allow me access to the source code of your algorithms for this purpose.

The source code will only be used to compare the performance characteristics of your algorithm with our own, for which you will be acknowledged in any resulting publications. The source code, or any part of it, will not be modified, distributed, reproduced, or adapted in any form.

I look forward to your response

Yours sincerely

Tom Cook

Research Student
Research Centre in Applied Sciences
University of Wolverhampton
Wulfruna Street
Wolverhampton
WV1 1SB

Correspondence with Professor Wu

Flexion crease algorithm comparison

Cook, Tom
Sent:23 July 2011 14:18
To: xqwu@hit.edu.cn

Dear Professor Wu

I am a research student from the University of Wolverhampton, United Kingdom, and would like to perform a comparison of palmar flexion crease identification algorithms for my PhD thesis.

Following your publications:

- Fuzzy directional element energy feature (FDEEF) based palmprint identification
- A novel approach of palm-line extraction
- Palmprint recognition using directional line energy feature
- Palm line extraction and matching for personal authentication

I would like to include your work in my comparison, and would be grateful if you could allow me access to the source code of your algorithms for this purpose.

The source code will only be used to compare the performance characteristics of your algorithm with our own, for which you will be acknowledged in any resulting publications. The source code, or any part of it, will not be modified, distributed, reproduced, or adapted in any form.

I look forward to your response

Yours sincerely

Tom Cook

Research Student
Research Centre in Applied Sciences
University of Wolverhampton
Wulfruna Street
Wolverhampton
WV1 1SB

RE: Flexion crease algorithm comparison
Cook, Tom
Sent:21 August 2011 21:17
To: xqwu@hit.edu.cn

Dear Professor Wu

I am writing to enquire whether you have received my previous email requesting your help in performing a comparison of palmar flexion crease identification algorithms for my PhD thesis.

Following your publications:

- Fuzzy directional element energy feature (FDEEF) based palmprint identification
- A novel approach of palm-line extraction
- Palmprint recognition using directional line energy feature
- Palm line extraction and matching for personal authentication

I would like to include your work in my comparison, and would be grateful if you could allow me access to the source code of your algorithms for this purpose.

The source code will only be used to compare the performance characteristics of your algorithm with our own, for which you will be acknowledged in any resulting publications. The source code, or any part of it, will not be modified, distributed, reproduced, or adapted in any form.

I look forward to your response

Yours sincerely

Regards,

Tom Cook

Research Student
Research Centre in Applied Sciences
University of Wolverhampton
Wulfruna Street
Wolverhampton
WV1 1SB

Correspondence with Professor Lui

Flexion crease algorithm comparison
Cook, Tom

Sent:23 July 2011 14:18
To: csliliu@comp.polyu.edu.hk

Dear Professor Lui

I am a research student from the University of Wolverhampton, United Kingdom, and would like to perform a comparison of palmar flexion crease identification algorithms for my PhD thesis.

Following your publication, 'Palm-line detection' in Proceedings of the International Conference on Image Processing, I would like to include your work in my comparison, and would be grateful if you could allow me access to the source code of your algorithms for this purpose.

The source code will only be used to compare the performance characteristics of your algorithm with our own, for which you will be acknowledged in any resulting publications. The source code, or any part of it, will not be modified, distributed, reproduced, or adapted in any form.

I look forward to your response

Yours sincerely

Tom Cook

Research Student
Research Centre in Applied Sciences
University of Wolverhampton
Wulfruna Street
Wolverhampton
WV1 1SB

RE: Flexion crease algorithm comparison

Cook, Tom
Sent:21 August 2011 21:17
To: csliliu@comp.polyu.edu.hk

Dear Professor Lui

I am writing to enquire whether you have received my previous email requesting your help in performing a comparison of palmar flexion crease identification algorithms for my PhD thesis.

Following your publication, 'Palm-line detection' in Proceedings of the International Conference on Image Processing, I would like to include your work in my comparison, and would be grateful if you could allow me access to the source code of your algorithms for this purpose.

The source code will only be used to compare the performance characteristics of your algorithm with our own, for which you will be acknowledged in any resulting publications. The source code, or any part of it, will not be modified, distributed, reproduced, or adapted in any form.

I look forward to your response

Yours sincerely

Tom Cook

Research Student
Research Centre in Applied Sciences
University of Wolverhampton
Wulfruna Street
Wolverhampton
WV1 1SB

Correspondence with Professor Li

Flexion crease algorithm comparison

Cook, Tom
Sent: 23 July 2011 14:18
To: asfli@ntu.edu.sg

Dear Professor Li

I am a research student from the University of Wolverhampton, United Kingdom, and would like to perform a comparison of palmar flexion crease identification algorithms for my PhD thesis.

Following your publication, 'Hierarchical identification of palmprint using line-based Hough transform' in Proceedings of the 18th International Conference on Pattern Recognition, I would like to include your work in my comparison, and would be grateful if you could allow me access to the source code of your algorithms for this purpose.

The source code will only be used to compare the performance characteristics of your algorithm with our own, for which you will be acknowledged in any resulting publications. The source code, or any part of it, will not be modified, distributed, reproduced, or adapted in any form.

I look forward to your response

Yours sincerely

Tom Cook

Research Student
Research Centre in Applied Sciences
University of Wolverhampton
Wulfruna Street
Wolverhampton
WV1 1SB

RE: Flexion crease algorithm comparison

Cook, Tom
Sent: 21 August 2011 21:17
To: asfli@ntu.edu.sg

Dear Professor Li

I am writing to enquire whether you have received my previous email requesting your help in performing a comparison of palmar flexion crease identification algorithms for my PhD thesis.

Following your publication, 'Hierarchical identification of palmprint using line-based Hough transform' in Proceedings of the 18th International Conference on Pattern Recognition, I would like to include your work in my comparison, and would be grateful if you could allow me access to the source code of your algorithms for this purpose.

The source code will only be used to compare the performance characteristics of your algorithm with our own, for which you will be acknowledged in any resulting publications. The source code, or any part of it, will not be modified, distributed, reproduced, or adapted in any form.

I look forward to your response

Yours sincerely

Tom Cook

Research Student
Research Centre in Applied Sciences
University of Wolverhampton
Wulfruna Street
Wolverhampton
WV1 1SB

Correspondence with Professor Huang

Flexion crease algorithm comparison

Cook, Tom
Sent:23 July 2011 14:18
To: dshaung@iim.ac.cn

Dear Professor Huang

I am a research student from the University of Wolverhampton, United Kingdom, and would like to perform a comparison of palmar flexion crease identification algorithms for my PhD thesis.

Following your publications, 'Palmprint verification based on principal lines' and 'Palmprint verification based on robust line orientation code' in Pattern Recognition, I would like to include your work in my comparison, and would be grateful if you could allow me access to the source code of your algorithms for this purpose.

The source code will only be used to compare the performance characteristics of your algorithm with our own, for which you will be acknowledged in any resulting publications. The source code, or any part of it, will not be modified, distributed, reproduced, or adapted in any form.

I look forward to your response

Yours sincerely

Tom Cook

Research Student
Research Centre in Applied Sciences
University of Wolverhampton
Wulfruna Street
Wolverhampton
WV1 1SB

RE: Flexion crease algorithm comparison

Cook, Tom
Sent:21 August 2011 21:17
To: dshaung@iim.ac.cn

Dear Professor Huang

I am writing to enquire whether you have received my previous email requesting your help in performing a comparison of palmar flexion crease identification algorithms for my PhD thesis.

Following your publications, 'Palmprint verification based on principal lines' and 'Palmprint verification based on robust line orientation code' in Pattern Recognition, I would like to include your work in my comparison, and would be grateful if you could allow me access to the source code of your algorithms for this purpose.

The source code will only be used to compare the performance characteristics of your algorithm with our own, for which you will be acknowledged in any resulting publications. The source code, or any part of it, will not be modified, distributed, reproduced, or adapted in any form.

I look forward to your response

Yours sincerely

Tom Cook

Research Student
Research Centre in Applied Sciences
University of Wolverhampton
Wulfruna Street
Wolverhampton
WV1 1SB