Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea Magistrale in Matematica ed Applicazioni

Image Analysis for Forensic Purposes

Tesi sperimentale in Analisi Numerica

Relatori:

Prof. Pierluigi Maponi

Prof. Riccardo Piergallini

Laureando:

Filippo Santarelli

ANNO ACCADEMICO 2014-2015

Indice

Capitolo 1

Introduzione

L'analisi delle immagini è un processo che attuiamo in continuazione, senza rendercene conto. La vista è, difatti, il nostro senso più sviluppato; quasi ogni nostra azione è la risposta ad un impulso visivo. Vedere una persona e riconoscerla ci induce a salutarla, oppure a cambiare strada; leggere quest'elaborato, ovvero acquisirne l'immagine e dare ordine e significato alle lettere impresse, permette al lettore di apprendere, o di fare qualche correzione; un ultimo esempio può essere legato all'acquisizione di dati quantitativi, ovvero possiamo conoscere il colore e la forma oppure stimare dimensioni e peso di qualche oggetto.

Questo non sarà, comunque, l'ambito del quale ci occuperemo in questo lavoro di tesi; infatti le immagini che verranno trattate non sono quelle acquisite dall'occhio umano, bensì da occhi artificiali: ci interesseremo dell'analisi delle immagini digitali. Il termine "digitale" sottintende che sono state effettuate delle misurazioni della realtà, quindi i dati sono stati resi idonei alla manipolazione tramite computer. Questo procedimento, che prende il nome di *digitalizzazione*, verrà approfondito nel capitolo 2.

Ad ogni modo, nonostante il mezzo con il quale si acquisiscono le immagini non sia l'occhio umano, l'obiettivo per cui si fa è lo stesso: interagire autonomamente con il mondo reale. Da molti anni, infatti, si sta cercando di rendere le macchine più "intelligenti"; ad esempio, numerose sono le tecniche di *machine learning* che sono state ideate, ossia algoritmi che consentono ad un computer di essere addestrato a rispondere correttamente a particolari input. Fare in modo che le macchine possano prendere delle decisioni non è un vezzo della scienza, ma una necessità proveniente da molti settori: la robotica e l'automazione nell'industria, la domotica, oppure, più semplicemente, affiancare l'uomo in situazioni in cui i suoi sensi e capacità non bastano, sono solo alcune delle possibili applicazioni. Affinché l'elaboratore si comporti come l'essere umano, quindi interagisca con la realtà e prenda conseguentemente decisioni, occorre fornirgli dei sensi, come ad esempio la vista, ovvero renderlo capace di analizzare le

immagini (digitali).

I primi computer sufficientemente potenti da riuscire ad elaborare immagini sono apparsi nei primi anni '60; questo costituisce sostanzialmente la nascita dell'elaborazione delle immagini digitali. Con l'intensificarsi delle trasmissioni d'immagini, vennero di pari passo studiate e migliorate le tecniche per manipolarle in modo automatico; infatti sin dall'inizio le trasmissioni non erano in grado di rendere buone immagini, per cui c'era bisogno di qualche algoritmo che le migliorasse. Uno dei primi casi avvenne nel 1964 ad opera del Jet Propulsion Laboratory di Pasadena, in California, che elaborò tramite computer le immagini della luna che giungevano dai satelliti. Di pari passo iniziarono ad essere utilizzate tecniche analoghe in medicina, soprattutto per migliorare o evidenziare parti degli esami radiologici dei pazienti. I campi di applicazione aumentarono notevolmente nel tempo, dalla sorveglianza, all'autenticazione per l'accesso a risorse riservate, dall'automazione industriale, al riconoscimento per scopi forensi, fino all'uso quotidiano che ne facciamo oggi con la fotocamera dei nostri smartphone.

Nel capitolo 2 mostreremo alcuni dei metodi più comuni per l'elaborazione e l'analisi delle immagini digitali; gli argomenti che tratteremo saranno la modifica dell'immagine, sia nel dominio dello spazio che in quello della frequenza, la segmentazione ed il riconoscimento di oggetti. Nonostante la differenza sembra impercettibile, elaborazione e analisi indicano intenti differenti: la prima è un processo che parte da un'immagine e ne restituisce un'altra, con caratteristiche differenti e magari più adatte per i processi che seguiranno; l'analisi invece è il procedimento per cui da un'immagine se ne riescono ad estrarre caratteristiche ed attributi, che verranno utilizzati da successivi algoritmi per consentire alla macchina di prendere decisioni. Sostanzialmente possiamo dire che l'elaborazione è la prima fase dopo l'acquisizione, quindi seguono l'analisi e la *computer vision*, ovvero l'insieme di tecniche per utilizzare e dare senso agli attributi estratti con l'analisi.

Il titolo di questo elaborato contiene un riferimento all'ambito giudiziario; molte sono le immagini che entrano in tribunale: impronte digitali, fotografie, esami radiologici, ne sono alcuni esempi. Ma a che cosa serve l'analisi di queste immagini? Principalmente viene utilizzata per decidere l'appartenenza delle impronte digitali, confrontando le cosiddette *impronte latenti*, rinvenute nelle scene del crimine, con le impronte immagazzinate in un database.

Come evidenziato abbondantemente nel capitolo 3 è già dall'inizio del '900 che si fa uso delle impronte digitali in ambito forense; queste venivano rilevate e confrontate a mano, da esperti che avevano investito molto tempo e molte risorse nella loro formazione. Ovviamente il processo di confronto era piuttosto lento, basti pensare che per ogni coppia di impronte ritenute simili, occorreva confrontarle un pezzetto alla volta. Il lavoro ripetitivo poteva finire spesso

per diminuire la concentrazione dell'esaminatore, aumentando ulteriormente il tempo impiegato nel confronto. Nel corso degli anni, prima che la tecnologia potesse intervenire in maniera efficace, sono state ideate tecniche per diminuire il lavoro svolto da questi esperti; principalmente si utilizzavano tecniche di catalogazione, per cui ad ogni impronta immagazzinata veniva subito assegnata una classe; ogni volta che si rilevava una nuova impronta bastava andarla a confrontare con quelle della stessa classe, diminuendo considerevolmente il numero di test da compiere e quindi il tempo totale impiegato. Nonostante questi miglioramenti, il procedimento risultava comunque scomodo; anche la formazione stessa degli esperti era molto lunga e dispendiosa; tutti motivi che condussero le agenzie investigative a investire risorse sui sistemi di identificazione automatici. Il capitolo 3 tratta proprio dei vari sistemi di identificazione, accennando ai principali indicatori biometrici in uso oggigiorno e spiegando come venivano catalogate le impronte digitali prima dell'avvento del computer. Nel successivo capitolo 4 vengono presentati alcuni tra i più noti metodi di analisi automatica delle impronte digitali, focalizzando l'attenzione sul miglioramento dell'immagine e l'estrazione di minutiae, ovvero caratteristiche salienti che permettono di eseguire il confronto.

L'ultima parte del capitolo 4 è dedicata alla descrizione dei metodi che sono stati utilizzati per creare il codice allegato (Appendice A). Obiettivo di questo lavoro è stato, infatti, sia conoscere ed approfondire alcune delle principali tecniche di analisi d'immagini e, più nello specifico, d'impronte digitali, sia implementare in MatLab un algoritmo che riuscisse a determinare la paternità delle impronte.

Capitolo 2

Teoria dell'elaborazione delle immagini

2.1 Introduzione

Prima di procedere ad esaminare i concetti fondamentali dell'elaborazione delle immagini, occorre introdurre i principi di base e convenire sulla notazione usata. Gran parte del materiale esposto è tratto dal libro [11], che ne contiene una trattazione approfondita.

L'acquisizione delle immagini avviene tramite le seguenti fasi: emissione di energia da una sorgente (l'energia in questione può essere la luce, ma anche una qualunque radiazione elettromagnetica o acustica, come avviene per molti esami medici), riflessione sugli elementi della scena, acquisizione dell'energia riflessa da parte dei sensori e digitalizzazione della quantità di energia da essi registrata.

In questo studio non ci interessa focalizzare l'attenzione su come lavorano i sensori, piuttosto ci fa comodo capire come il fenomeno fisico, dato di tipo analogico, venga convertito in un'immagine, ovvero un dato digitale. Tratteremo solo il caso di un sensore di tipo matriciale, poiché questo è quello che viene generalmente usato per la rilevazione delle impronte digitali, argomento centrale di questo lavoro. Rappresentiamo il dato analogico, ossia i valori di energia che giungono al sensore, mediante una funzione reale a valori bidimensionali $f(x, y)$ continua; in base a come si sceglie il sistema di riferimento, le coordinate x ed y variano entro particolari domini, diciamo $x \in X$ e $y \in Y$. Per giungere alla digitalizzazione di un dato di questo tipo, vengono compiute due operazioni, che prendono il nome di *sampling* (campionamento) e *quantizzazione*.

- Il sampling è il processo che digitalizza le coordinate (x, y); per eseguire il campionamento dei valori del dominio X basta prendere una sequenza di

valori equispaziati $x_i \in X$, con $i = 0, 1, \ldots, M$ e restringere il dominio della funzione all'insieme di questi x_i. Analogamente si procede con Y.

- la quantizzazione, invece, è l'operazione di digitalizzare il modulo della funzione f. Anche in questo caso è tutto abbastanza semplice; occorre suddividere il codominio della funzione f in L elementi, quindi approssimare ogni valore $f(x, y)$ con essi.

Nella pratica le operazioni di campionamento e quantizzazione vengono compiute direttamente dal dispositivo di acquisizione, in funzione della meccanica dell'apparecchio stesso. Il risultato finale è un'immagine rettangolare, che nella nostra trattazione assumeremo di dimensioni $M \times N$, a valori interi nell'intervallo $[0, L-1]$. Le immagini con cui avremo a che fare maggiormente sono immagini a 8 bit, per le quali $L = 256$, più comunemente conosciute con il nome di immagini in scala di grigi.

2.2 Miglioramento nel dominio dello spazio

In questa sezione vengono esposti alcuni metodi per il miglioramento delle immagini, volti a rendere queste ultime più adatte per alcune specifiche applicazioni, piuttosto che per altre. Per questo motivo le tecniche trattate sono strettamente legate al problema da risolvere. Per chiarire le idee: un algoritmo che migliora l'immagine di un'impronta digitale può non essere adatto a rendere più nitida la scansione della superficie del pianeta Marte.

Le tecniche per migliorare le immagini possono essere raggruppate in due grandi insiemi: i metodi che operano nel dominio dello spazio e quelli che lavorano nel dominio della frequenza. I primi manipolano direttamente i punti dell'immagine, qui il termine "spaziale" si riferisce al piano su cui è definita l'immagine stessa, mentre gli altri manipolano la trasformata di Fourier dell'immagine, definita appunto nel dominio della frequenza; tecniche appartenenti a gruppi differenti sono spesso usate in combinazione per beneficiare dei vantaggi di ogni singolo metodo.

In generale non esiste un criterio unico per giudicare la qualità di un'immagine; spesso la si ritiene "buona" se essa ha una buona valutazione visiva dell'osservatore, il che rende questo concetto fortemente legato al giudizio personale dell'operatore. Per questo motivo la scelta della tecnica giusta può essere preceduta da una serie di prove e tentativi fallimentari.

Utilizzando la stessa notazione della Sezione 2.1, supponiamo di avere un'immagine $M \times N$ in scala di grigi, i cui valori sono indicati con $f(x, y)$. Le manipolazioni dell'immagine nel dominio dello spazio possono essere scritte nella forma

generale

$$g(x, y) = T[f(x, y)]$$

dove T è un operatore che agisce in un intorno del punto (x, y). A seconda del tipo di tecnica scelta per modificare l'immagine, è necessario scegliere un appropriato intorno; nella maggior parte dei casi viene preferito un intorno rettangolare di ordine $m \times n$, che semplifica di molto l'implementazione dell'algoritmo, ma ne esistono numerosi altri tipi.

2.2.1 Trasformazioni dell'intensità

Per iniziare la trattazione, vediamo alcuni tipi di trasformazioni che prevedono intorni rettangolari degeneri, ovvero di ordine 1×1.

- Il **negativo** di un'immagine si ottiene applicando ad essa la trasformazione

  $$s = L - 1 - r$$

  dove r ed s sono rispettivamente i valori di grigio iniziale e finale. Questo tipo di trasformazione è molto usata nel caso in cui bisogna enfatizzare particolari bianchi o grigi, immersi in uno sfondo nero (vedi Figura 2.2.1).

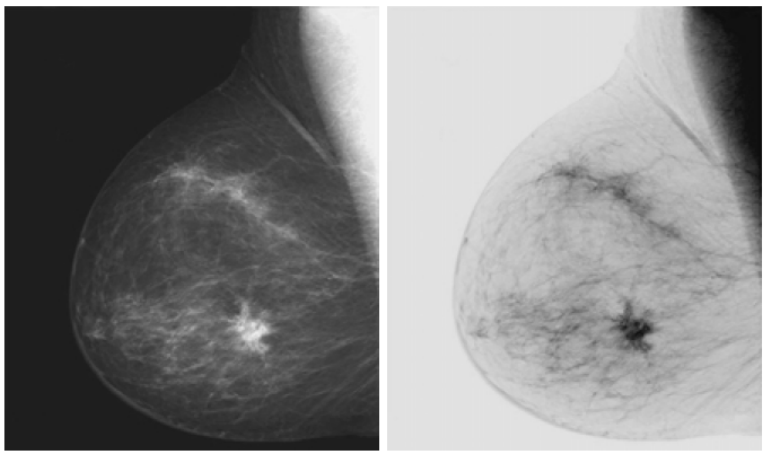


Figura 2.2.1: Mammografia originale, a sinistra, e suo negativo, a destra (tratta da [11]).

- La forma generale di una **trasformazione logaritmica** è

  $$s = c \log(1 + r)$$

dove si assume che $r \geq 0$ e c sia una costante arbitraria. Se facciamo riferimento al grafico della trasformazione logaritmica visualizzato in Figura 2.2.2, possiamo renderci conto che essa dilata i valori di grigio più bassi, mentre comprime quelli superiori ad una certa soglia.

- La **trasformazione logaritmica inversa**, invece, assolve al compito opposto, comprimendo i valori di grigio più bassi e dilatando i più alti.
- Le **trasformazioni a potenze** si comportano in maniera simile alle trasformazioni logaritmiche oppure logaritmiche inverse, a seconda dei parametri utilizzati; la forma generica delle trasformazioni a potenze è:

$$s = cr^{\gamma}$$

dove c e γ sono costanti, r ed s come al solito i valori iniziali e finali.

Oltre quelle appena elencate, che sono le più semplici, nonché le più usate, esistono infinite possibilità di definire una trasformazione dei livelli di grigio. Un modo molto usato, per combinare i vantaggi delle trasformazioni appena viste è quello di definire funzioni per parti, con cui, per esempio, si possono mettere in risalto solamente valori di grigio in un certo intervallo.

2.2.2 Equalizzazione dell'istogramma

Passiamo ora a definire una delle più utilizzate caratteristiche di un'immagine: l'*istogramma*. Il suo uso non è solamente ai fini del miglioramento delle immagini digitali, che vedremo di seguito, ma anche per la compressione e la segmentazione. L'istogramma di un'immagine, di dimensione $M \times N$ con valori di grigio in $[0, L-1]$ è la funzione $h(k) = n_k$, con $k = 0, 1, \dots, L-1$, che associa ad ogni valore di grigio, il numero n_k di occorrenze di quel valore all'interno dell'immagine. In genere i valori dell'istogramma vengono normalizzati, per cui chiamiamo

$$p(k) = \frac{n_k}{N \cdot M} \qquad k = 0, 1, \dots, L-1$$

il k-esimo valore dell'istogramma; possiamo pensare ad esso come una misura della probabilità che, scelto un punto a caso, esso abbia il valore di grigio k. La Figura 2.2.3 mostra l'istogramma dei valori di grigio delle quattro immagini a sinistra. Chiamiamo $\xi(i)$ la variabile aleatoria a supporto in $[0, L-1]$ che associa all'i-esimo punto, dell'immagine di partenza, il corrispondente valore di grigio; indichiamo poi con $p_{\xi}(k)$, per $k = 0, \dots, L-1$, la densità di probabilità della variabile ξ, ovvero la probabilità che essa assuma il valore di grigio k. Definiamo l'analoga variabile η per l'immagine di arrivo, indichiamo con $p_{\eta}(k)$ la sua

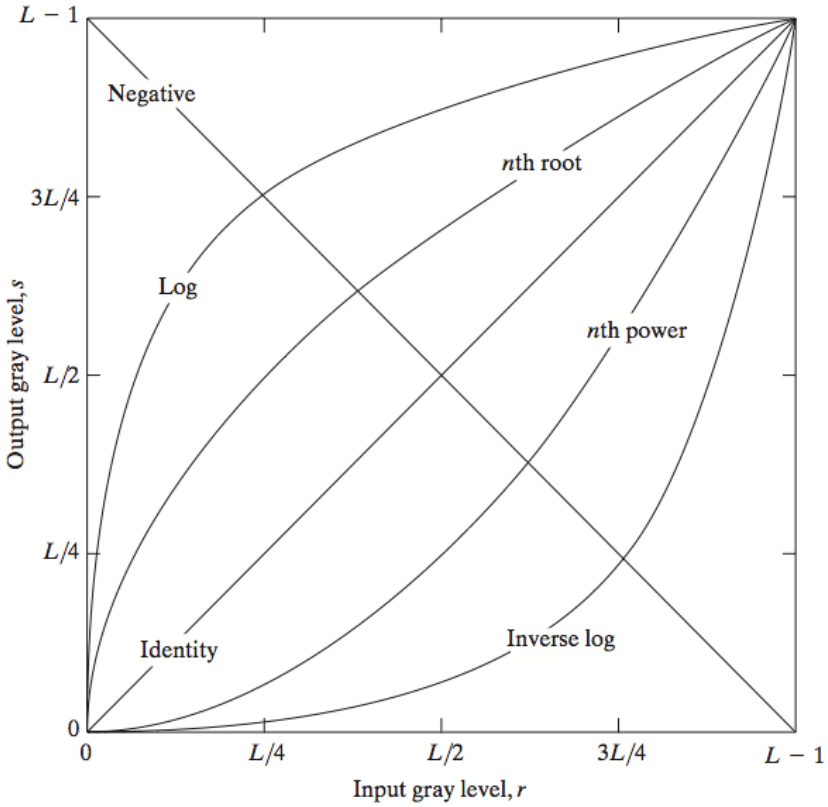


Figura 2.2.2: Grafico delle principali trasformazioni sulle immagini in scala di grigi (tratto da [11]).

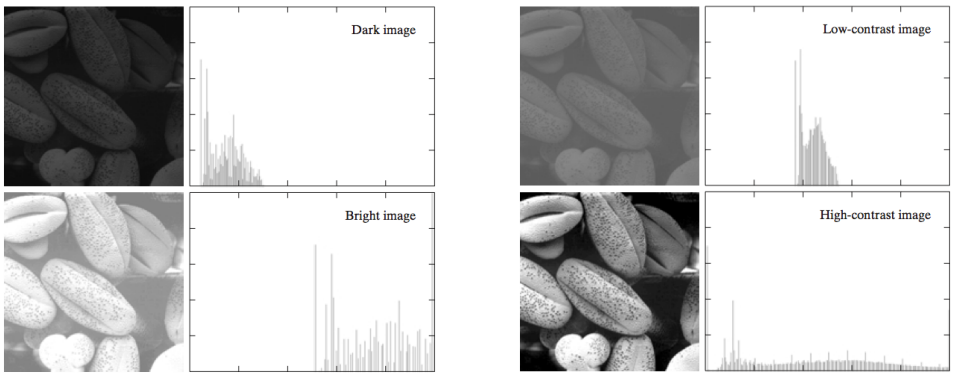


Figura 2.2.3: Quattro immagini ed il corrispondente istogramma dei valori di grigio.

densità di probabilità. Supponiamo per il momento di trattare variabili aleatorie continue, così da utilizzare alcuni utili risultati della teoria delle probabilità. Uno di questi è la formula seguente:

$$p_\eta(\eta(t)) = p_\xi(\xi(t))\left|\frac{d\xi}{d\eta}\right| \qquad \eta = T[\xi] \tag{2.2.1}$$

dove T è monotona non decrescente e a valori in $[0, L-1]$. Una delle trasformazioni che verifica queste ipotesi, molto usata nel contesto dell'elaborazione delle immagini, è:

$$\eta(t) = T[\xi](t) = (L-1)\int_0^{\tilde{t}} p_\xi(s)ds \qquad \tilde{t} = \xi(t) \tag{2.2.2}$$

che sostanzialmente rappresenta la probabilità cumulativa della variabile aleatoria η. A partire dalla (2.2.2) andiamo a calcolare:

$$\begin{aligned}\frac{d\eta}{d\xi} &= \frac{dT[\xi]}{d\xi} \\ &= (L-1)\cdot p_\xi(\xi(t))\end{aligned}$$

e sostituendo nella (2.2.1) otteniamo

$$p_\eta(\eta(t)) = p_\xi(\xi(t))\cdot\frac{1}{(L-1)\cdot p_\eta(\xi(t))} = \frac{1}{L-1} \tag{2.2.3}$$

Riconosciamo che la (2.2.3) esprime il fatto che la densità di probabilità della variabile aleatoria η è uniforme. Volendo tornare al caso discreto, occorre cambiare terminologia e parlare di probabilità (ossia valori dell'istogramma), anziché densità di probabilità, e di sommatorie invece che integrali; pertanto possiamo scrivere i valori della variabile ξ come:

$$\eta(i) = T[\xi](i) = (L-1)\sum_{j=0}^{\xi(i)} p_\xi(j) = \frac{L-1}{MN}\sum_{j=0}^{k} n_j \tag{2.2.4}$$

dove, con la notazione usata in precedenza, n_j è il numero di punti con valore di grigio j, $M \cdot N$ è il numero di punti totali dell'immagine. Questa trasformazione dei livelli di grigio prende il nome di *equalizzazione dell'istogramma*; infatti, nel caso continuo, abbiamo visto come l'utilizzo della (2.2.2) produca un'immagine con un istogramma costante (distribuzione uniforme), cioè equalizzato. Nel caso discreto, però, è molto raro che questo avvenga; ciò che accade realmente è il fatto che l'istogramma equalizzato tenda sempre ad occupare una gamma di valori di grigio ben più ampia dell'istogramma originale. Questa proprietà è molto importante in molte applicazioni pratiche, cosa che, unitamente al fatto che questo metodo è completamente automatico e semplice da implementare, lo rende molto utilizzato.

2.2.3 Histogram Matching

Nonostante quanto appena visto, esistono casi in cui un istogramma uniforme non è desiderabile; al contrario potrebbe far comodo specificare la forma dell'istogramma. Il metodo che stiamo per proporre, chiamato *histogram matching*, si prefigge proprio questo obiettivo.

Torniamo momentaneamente al caso continuo. Supponiamo di avere r, che descrive l'intensità dell'immagine di partenza, con densità di probabilità $p_r(r)$; immaginiamo quindi di voler arrivare ad un'immagine con intensità z non nota, ma di cui conosciamo la densità di probabilità $p_z(z)$, poiché scelta arbitrariamente. Quello che ci manca è la trasformazione che consenta di portare r in z. L'idea è quella di trasformare ambedue le variabili con la (2.2.2), quindi avere una variabile s tale che

$$s = T(r) = (L-1)\int_0^r p_r(w)dw$$
$$s = G(z) = (L-1)\int_0^z p_z(w)dw$$

Risulta semplice allora concludere che necessariamente $z = G^{-1}T(r)$. Trovare la funzione inversa G^{-1} nel caso continuo non è affatto semplice, ma se torniamo al caso discreto la situazione si semplifica. Infatti il metodo da seguire è il seguente.

1. Calcolare l'istogramma dei valori r_k, equalizzarlo per ottenere i valori s_k, per $k = 0, 1, \dots, L-1$, quindi arrotondarli all'intero più vicino.

2. Utilizzare la formula (2.2.4) per calcolare l'inversa di G; infatti, per un particolare valore di $q = 0, 1, \dots, L-1$, abbiamo

$$G(z_q) = (L-1)\sum_{j=0}^{q} p_z(z_j)$$

   che corrisponderà ad un preciso valore s_k; a questo punto sappiamo che, se troviamo s_k dobbiamo trasformarlo in z_q tramite la G^{-1}. A differenza del caso continuo possiamo pensare di calcolare $G(z_q) = s_k$ per ogni $q = 0, 1, \dots, L-1$, quindi arrotondare i valori all'intero più vicino; bisogna poi lasciare i valori ottenuti in una tabella in cui sia evidente la corrispondenza tra ogni z_q ed il rispettivo s_k.

3. Per ogni valore s_k per $k = 0, 1, \dots, L-1$, trovo il valore più vicino tra quelli calcolati al passo 2; supponendo che quest'ultimo sia $G(z_q)$, sappiamo che $z_q = G^{-1}(s_k)$.

4. Ora che abbiamo tutta la trasformazione G^{-1}, possiamo prendere i valori s_k calcolati al passo 1 e trasformali per ottenere, infine, il risultato voluto.

Nelle applicazioni pratiche di questo metodo, si può notare che i valori ottenuti non sono esattamente quelli specificati tramite la $p_z(z)$; questo è dovuto alla discretizzazione, ovvero all'errore di arrotondamento che si commette ad ogni passo.

I due algoritmi di equalizzazione e histogram matching possono essere applicati anche localmente. Senza scendere nei dettagli, si suddivide l'immagine tramite una griglia e si applicano i metodi esposti ad ogni sottoimmagine ottenuta. Il risultato è generalmente migliore rispetto all'approccio globale, a scapito della velocità di esecuzione.

2.2.4 Filtri spaziali lineari

Il nome *filtro* viene preso in prestito dal contesto del filtraggio nel dominio della frequenza, in cui *filtrare* si riferisce all'accettare o rifiutare determinate frequenze. Un filtro che, ad esempio, lascia passare le basse frequenze viene chiamato *lowpass filter*, che ha l'effetto di rendere l'immagine apparentemente sfocata, eventualmente diminuendo il rumore puntiforme; il termine tecnico per questo effetto è *blur*, mentre una generica operazione che permette di ottenerlo prende il nome di *smoothing*. La controparte di un lowpass filter, ovvero un filtro che sia in grado di preservare le alte frequenze, attenuando le basse, è l'*highpass filter*; l'effetto è evidenziare le brusche transizioni tra i livelli di grigio, ovvero mettere in risalto i contorni, e l'operazione associata si chiama *sharpening*. Smoothing e sharpening si possono eseguire anche tramite alcuni filtri spaziali, infatti dimostreremo nella sezione 2.3 che esiste una corrispondenza uno ad uno tra i filtri nel dominio dello spazio e quelli nel dominio della frequenza.

Un filtro spaziale consta di due elementi principali:

- un intorno, generalmente rettangolare, di dimensione $m \times n$, con $m = 2a+1$ e $n = 2b + 1$, per $a, b \in \mathbb{N}$; infatti nella maggior parte dei casi si ha a che fare con intorni di dimensione dispari, per i quali il centro ha coordinate intere, più semplici da gestire;

- un'operazione da eseguire sui pixel compresi nell'intorno.

Un filtro spaziale permette di sostituire il valore del centro dell'intorno con il risultato dell'operazione eseguita sui pixel di tale intorno; per generare un'intera immagine filtrata si centra l'intorno su ogni pixel dell'immagine di partenza e si eseguono le operazioni richieste. Se l'operazione che definisce il filtro è lineare allora si parla di filtro lineare, altrimenti il filtro si dice non lineare.

Nel caso di un filtro spaziale lineare w, la risposta $g(x,y)$ ha un'espressione piuttosto semplice:

$$g(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} w(i,j) \cdot f(x+i, y+j) \tag{2.2.5}$$

La (2.2.5) può anche essere espressa mediante l'operazione di correlazione, che ora andiamo a descrivere. Si dice **correlazione** di una matrice $A \in \mathbb{R}^{2l+1 \times 2s+1}$ con una matrice $B \in \mathbb{R}^{2a+1 \times 2b+1}$ la matrice $C = A \star B \in \mathbb{R}^{2l+1 \times 2s+1}$ dove l'elemento c_{ij} di C si ottiene centrando la matrice B sull'elemento a_{ij} della matrice A (aggiungendo linee di 0 nel caso in cui la matrice B esca fuori da essa) e calcolando la somma dei prodotti degli elementi sovrapposti. Possiamo quindi considerare f e w come matrici e riscrivere la (2.2.5) come

$$g(x,y) = f(x,y) \star w(x,y)$$

Un'altra importante operazione che dobbiamo definire in questo contesto è la convoluzione. Si dice **convoluzione** della matrice $A \in \mathbb{R}^{2l+1 \times 2s+1}$ con la matrice $B \in \mathbb{R}^{2a+1 \times 2b+1}$ la matrice $C = A * B \in \mathbb{R}^{2l+1 \times 2s+1}$ ottenuta mediante la correlazione di A e della matrice B ruotata di 180°. Se vogliamo scrivere esplicitamente la correlazione dell'immagine f con il filtro w scriveremo allora:

$$f(x,y) * w(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} w(i,j) \cdot f(x-i, y-j) \tag{2.2.6}$$

Ogni volta che si definisce un filtro spaziale, bisogna sempre indicare anche il tipo di operazione, correlazione oppure convoluzione, per cui è stato creato. Nel seguito della trattazione, a meno che non sia esplicitamente indicato, utilizzeremo sempre la formula (2.2.5).

Per generare un filtro spaziale lineare è, quindi, sufficiente specificare i coefficienti della maschera w, poi le operazioni di correlazione e convoluzione permettono il rapido calcolo della risposta; per filtri spaziali non lineari invece bisogna specificare il tipo di operazione da compiere nell'intorno di ogni punto: ad esempio, possiamo creare un filtro la cui risposta è il massimo valore rilevato nell'intorno 5×5 di ogni punto, ma questa operazione non è lineare e non si può eseguire tramite correlazione oppure convoluzione.

2.2.5 Smoothing filters

Un'importante categoria di filtri spaziali lineari è costituita da filtri che eseguono lo smoothing dell'immagine. Filtri di questo tipo sono, ad esempio, quelli che eseguono la media, aritmetica o ponderata, delle intensità nell'intorno

(averaging filters). La forma generale della risposta ad un filtro di questo tipo è

$$g(x,y) = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)f(x+a,y+b)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)}$$

A titolo di esempio, nelle Figure 2.2.4(a) e 2.2.4(b) riportiamo rispettivamente il filtro che esegue la media aritmetica in un intorno 3×3 ed un filtro per la media ponderata di dimensione 3×3. Questi filtri, come abbiamo detto, sostituiscono

$$\frac{1}{9} \cdot \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

(a) Implementazione di un averaging filter.

$$\frac{1}{16} \cdot \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

(b) Implementazione di un weighted averaging filter.

Figura 2.2.4: Nella figura sono mostrati due diversi filtri, entrambi volti a mediare le intensità dell'intorno del punto corrente; l'effetto di entrambi sarà, quindi, l'appannamento dell'immagine.

il valore di ogni pixel con l'intensità media nel suo intorno; questo procedimento porta l'immagine ad avere cambiamenti di intensità meno bruschi: da un lato questo permette di eliminare il rumore di dimensione inferiore al filtro, di contro, però, vengono smussati anche i contorni, caratterizzati anch'essi da brusche variazioni di intensità.

2.2.6 Order-Statistic Filters

Facciamo un rapido cenno anche ai filtri non lineari chiamati *order-statistic filters*. La cosa che accomuna questi filtri è il fatto che i valori d'intensità nell'intorno di ogni punto vengano prima di tutto ordinati, quindi successivamente viene calcolata una statistica su di essi ed il pixel centrale è rimpiazzato con tale statistica. Il più comune filtro di questo tipo è il filtro mediano, in cui la statistica utilizzata è proprio la mediana; esso è molto utile per eliminare rumore impulsivo (ovvero puntiforme), senza però offuscare l'immagine di partenza come farebbe un averaging filter. Oltre al filtro mediano esistono numerosi altri filtri a statistiche d'ordine; giusto per fare un esempio: la mediana è il 50-esimo percentile, ma possiamo creare filtri per qualunque percentile; filtri spesso utili sono quello a percentile 0 (che corrisponde a calcolare il minimo nell'intorno) ed il 100-esimo percentile (ovvero il massimo valore d'intensità nell'intorno).

2.2.7 Sharpening filters

Abbiamo visto che l'effetto di appannamento dell'immagine può essere ottenuto tramite filtri che eseguono la media delle intensità; se ora vogliamo ottenere l'effetto opposto, ovvero l'esaltazione dei margini e delle variazioni di intensità, sembra ragionevole utilizzare filtri che eseguano l'operazione inversa; poiché mediare corrisponde all'operatore di integrazione, ora dovremo utilizzare la differenziazione. Il contesto non è quello delle funzioni continue, per cui sarà necessario ridefinire il concetto di derivata. A prescindere dalla definizione che daremo, sicuramente la derivata prima dovrà avere le seguenti caratteristiche:

- si deve annullare nelle aree ad intensità costante;
- deve essere diversa da zero all'inizio di rampe o gradini;
- deve essere non nulla lungo le rampe.

Anche la derivata seconda dovrà rispettare alcune regole, ovvero:

- deve annullarsi in aree ad intensità costante;
- non deve annullarsi all'inizio ed alla fine di rampe e gradini;
- deve annullarsi lungo rampe a pendenza costante.

2.2.7.1 Sharpening filters: il laplaciano

Diamo innanzitutto la definizione di derivate parziali seconde, il cui utilizzo nel miglioramento dell'immagine è più semplice da trattare. Possiamo renderci conto che le seguenti definizioni rispettano i vincoli appena stabiliti:

$$\begin{aligned} \frac{\partial^2 f}{\partial x^2} &= f(x+1,y) - 2f(x,y) + f(x-1,y) \\ \frac{\partial^2 f}{\partial y^2} &= f(x,y+1) - 2f(x,y) + f(x,y-1) \end{aligned} \tag{2.2.7}$$

Vediamo come utilizzare le derivate seconde per creare dei filtri che mettano in risalto i bordi. Limitiamoci a considerare solamente filtri isotropi, ovvero filtri che siano invarianti per rotazioni: applicare il filtro e ruotare l'immagine deve dare lo stesso risultato che ruotare l'immagine ed applicare il filtro. È stato dimostrato in [31] che il più semplice filtro isotropo che dipenda dalle derivate dell'immagine è quello che usa il laplaciano, operatore definito come

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{2.2.8}$$

Unendo la (2.2.7) e la (2.2.8) otteniamo la formula per calcolare il laplaciano discreto dell'immagine f nel punto (x, y):

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \quad (2.2.9)$$

L'equazione (2.2.9) si può implementare usando il filtro mostrato in Figura 2.2.5(a); questo filtro è isotropo per intervalli di 90°. Se vogliamo generare un filtro laplaciano isotropo per intervalli di 45°, allora dobbiamo includere anche le diagonali; sostanzialmente bisogna aggiungere alla (2.2.9) i contributi

$$f(x+1,y+1) - 2f(x,y) + f(x-1,y-1)$$
$$f(x-1,y+1) - 2f(x,y) + f(x+1,y-1)$$

Utilizzando questi contributi la (2.2.9) diviene:

$$\begin{aligned}\nabla^2 f(x,y) = {} & f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 8f(x,y) \\ & + f(x+1,y+1) + f(x-1,y-1) + f(x+1,y-1) + f(x-1,y+1)\end{aligned} \quad (2.2.10)$$

L'implementazione della (2.2.10) è mostrata in Figura 2.2.5(b). Il laplaciano di

0	1	0
1	−4	1
0	1	0

(a) Implementazione della (2.2.9).

1	1	1
1	−8	1
1	1	1

(b) Implementazione della (2.2.10).

Figura 2.2.5: In figura sono mostrati i filtri ottenuti utilizzando due diverse definizioni di laplaciano, una isometrica per angoli di 90° e l'altra isometrica per angoli di 45°.

un'immagine è ancora un'immagine in cui sono state evidenziate le discontinuità e le brusche variazioni d'intensità, mentre le zone a livello di grigio pressoché costante si confondono con lo sfondo. Per recuperare le caratteristiche dell'immagine che sono andate perdute, basta semplicemente sottrarre la risposta ottenuta all'immagine originale. In questo modo, si vanno a diminuire i valori di grigio in corrispondenza dei minimi e ad aumentarne l'intensità nei pressi dei massimi: così facendo vengono evidenziati i cambi di intensità. L'immagine finale, usando la (2.2.9), sarà data da:

$$\begin{aligned} g(x,y) &= f(x,y) - \nabla^2 f(x,y) \\ &= 5f(x,y) - f(x+1,y) - f(x-1,y) - f(x,y+1) - f(x,y-1) \end{aligned} \quad (2.2.11)$$

Una formula analoga si può ottenere utilizzando l'equazione (2.2.10). Il filtro da utilizzare per ottenere l'immagine finale descritta dalla (2.2.11) è quello riportato in Figura 2.2.6(a).

0	−1	0
−1	5	−1
0	−1	0

(a) Implementazione della (2.2.11).

−1	−1	−1
−1	9	−1
−1	−1	−1

(b) Implementazione della formula finale ottenuta impiegando la (2.2.10).

Figura 2.2.6: Implementazione dei filtri per migliorare l'immagine tramite laplaciano, uno isometrico per angoli di 90° e l'altro per angoli di 45° rispettivamente.

2.2.7.2 Sharpening filters: il gradiente

Le derivate prime sono implementate nel miglioramento dell'immagine utilizzando il modulo del gradiente. Ricordiamo che il gradiente di una funzione bidimensionale f nel punto (x, y) è

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}(x,y) \, , \, \frac{\partial f}{\partial y}(x,y) \right)$$

Questo vettore ha l'importante proprietà che punta sempre in direzione della maggiore variazione di f attorno al punto (x, y). Il modulo del vettore gradiente rappresenta, di conseguenza, il valore di tale variazione attorno al punto (x, y) e si calcola come

$$M(x,y) = \sqrt{\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2}$$

La funzione M può essere calcolata in tutti i punti ed è essa stessa un'immagine, chiamata *immagine gradiente*. Per ragioni di efficienza il modulo del gradiente $M(x, y)$ viene spesso approssimato con la formula

$$M(x,y) \approx \left| \frac{\partial f}{\partial x} \right| + \left| \frac{\partial f}{\partial y} \right|$$

Questa approssimazione permette un calcolo più rapido, poiché evitiamo di calcolare i due quadrati e l'estrazione di radice, inoltre preserva le variazioni di intensità relative. Come abbiamo fatto nel caso del laplaciano, dobbiamo ridefinire gli operatori di derivazione; tra i molteplici operatori di derivazione che si trovano in letteratura, conveniamo di utilizzare quelli di Sobel, rappresentati dalle

maschere in Figura 2.2.7. Occorre notare come gli operatori di Sobel eseguano anche la media pesata di alcuni valori, dando maggiore importanza a quelli centrali. Paragonato all'uso del laplaciano discreto, il metodo del gradiente è più

−1	−2	−1
0	0	0
1	2	1

(a) Implementazione dell'operatore di Sobel per la differenziazione lungo l'asse x.

−1	0	1
−2	0	2
−1	0	1

(b) Implementazione dell'operatore di Sobel per la differenziazione lungo l'asse y.

Figura 2.2.7: La figura mostra le due maschere da utilizzare in convoluzione con l'immagine per applicarle gli operatori di Sobel.

utile ad enfatizzare variazioni costanti nei livelli di grigio, poiché l'operatore di laplace sarebbe non nullo solamente all'inizio ed alla fine della rampa; il laplaciano, invece, viene usato principalmente se si devono enfatizzare rapidi cambi di intensità.

2.2.7.3 Sharpening filters: unsharp masking

L'ultimo filtro nel dominio dello spazio di cui si tratterà sarà il cosiddetto *unsharp masking*. Il procedimento è il seguente:

- innanzitutto si utilizza un filtro per lo *smoothing* sull'immagine f, ottenendo l'immagine $\bar{f}$;
- successivamente si calcola l'immagine differenza, chiamata *maschera*,

$$g_{mask} = f - \bar{f} \tag{2.2.12}$$

- infine l'immagine finale g viene ottenuta sommando la maschera all'immagine iniziale

$$g = f + g_{mask} = 2f - \bar{f}$$

Una generalizzazione di questo algoritmo è l'*highboost filtering*; in questo caso l'immagine finale viene calcolata aggiungendo la maschera, scalata tramite un peso $k > 1$:

$$g = f + k \cdot g_{mask} = (k+1)f - k\bar{f} \tag{2.2.13}$$

Notiamo che per $k = 1$ si ritorna all'unsharp masking. Per tutti i valori di $k \geq 1$ si ottiene sempre un effetto di esaltazione dei contorni, così come per il gradiente e per il laplaciano.

2.3 Miglioramento nel dominio della frequenza

Vediamo ora, in generale, come si esegue la modifica delle immagini nel dominio della frequenza.

Iniziamo con il ricordare la formula della trasformata di Fourier discreta (DFT) di un'immagine $f(x,y)$ di dimensione $M \times N$:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)} \tag{2.3.1}$$

con $u = 0, 1, \dots, M-1$ e $v = 0, 1, \dots, N-1$. Inoltre, data la DFT $F(u,v)$, possiamo calcolare la sua inversa, ovvero $f(x,y)$ tramite la trasformata di Fourier inversa discreta (IDFT):

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)} \tag{2.3.2}$$

con $x = 0, 1, \dots, M-1$ e $y = 0, 1, \dots, N-1$. È facile notare che ogni elemento dell'immagine $F(u,v)$ ha bisogno di tutti i valori di f, moltiplicati per opportuni esponenziali, per poter essere calcolato; questo sicuramente rende difficile fare un collegamento diretto tra componenti specifiche dell'immagine e la sua trasformata, eccetto che in casi molto semplici. Nonostante questo, si possono dare delle nozioni generali sul legame tra le caratteristiche spaziali dell'immagine e la sua trasformata di Fourier. Facciamo un esempio: la frequenza nella trasformata di Fourier è legata alla velocità di variazione dell'intensità dell'immagine; per essere più specifici, in $(0,0)$ il valore della trasformata è proporzionale all'intensità media dell'immagine, le basse frequenze corrispondono a basse variazioni di intensità (quindi a zone con livelli di grigio quasi costanti), le alte frequenze, molto lontane dall'origine, sono associate a rapidi cambiamenti di intensità nell'immagine, quindi generalmente a bordi o rumore.

I metodi di filtraggio nel dominio della frequenza hanno tutti la stessa struttura.

- Data un'immagine $f(x,y)$ di dimensione $M \times N$ si esegue il *padding*, ovvero si aggiungono linee nulle, fino ad ottenere un'immagine $f_P(x,y)$ di dimensioni $P \times Q$, dove $P = 2M$ e $Q = 2N$. Senza scendere nei dettagli, che richiederebbero una trattazione abbastanza lunga, questa tecnica permette di velocizzare il calcolo e di evitare alcuni errori al bordo.

- Moltiplichiamo l'immagine $f_P(x, y)$ per $(-1)^{x+y}$ in modo da centrare la trasformata di Fourier di f_P. Questo procedimento è utile nel caso in cui si utilizzi una *funzione filtro* $H(u, v)$ simmetrica rispetto al centro dell'immagine; viene generalmente scelto un tale tipo di filtro per via della semplicità nella sua definizione.
- Calcoliamo la DFT $F(u, v)$ dell'immagine ottenuta, tramite la formula (2.3.1).
- Scegliamo una funzione filtro $H(u, v)$ della stessa dimensione di F, ovvero $P \times Q$.
- Eseguiamo il prodotto

$$G(u, v) = F(u, v) \cdot H(u, v) \tag{2.3.3}$$

  in modo da modificare la trasformata dell'immagine a nostro piacimento.
- Torniamo indietro al dominio dello spazio, tramite la IDFT definita dalla formula (2.3.2); calcoliamo quindi:

$$g_P(x, y) = \{\mathrm{Re}\,[\mathfrak{F}^{-1}\,(G(u, v))]\} \cdot (-1)^{x+y} \tag{2.3.4}$$

  dove $\mathfrak{F}^{-1}$ indica l'inversa della trasformata di Fourier e il pedice P serve a ricordare che stiamo ancora manipolando immagini a cui è stato applicato il padding.
- L'ultimo passo da compiere è togliere le linee aggiunte al primo passo tramite padding, quindi selezionare la regione di dimensione $M \times N$ all'angolo in alto a sinistra dell'immagine g_P.

Come abbiamo detto in precedenza, le frequenze corrispondono a variazioni d'intensità nel dominio dello spazio. Per questo motivo, le funzioni filtro H che attenuano le alte frequenze, lasciando praticamente invariate le basse, prendono il nome di lowpass filters ed daranno l'effetto smoothing nel dominio dello spazio; viceversa, le funzioni filtro che tendono ad annullare i bassi valori della frequenza si chiamano highpass filters e consentono di evidenziare i margini dell'immagine (sharpening filters).

Il legame tra la rappresentazione del filtro in frequenza e nello spazio è dato dal teorema di convoluzione. Esso afferma che: date due funzioni f e g, indicate con F e G rispettivamente le loro trasformate di Fourier, allora

$$\begin{aligned} \mathfrak{F}\,(f(x, y) * g(x, y)) &= F(u, v) \cdot G(u, v) \\ \mathfrak{F}\ (f(x, y) \cdot g(x, y)) &= F(u, v) * G(u, v) \end{aligned} \tag{2.3.5}$$

dove $\mathfrak{F}$ indica la trasformata di Fourier e $*$ è l'operatore di convoluzione. Supponiamo ora di avere un filtro $H(u,v)$ nel dominio della frequenza e di voler trovare la sua rappresentazione spaziale. Prendiamo allora l'immagine $f(x,y) = \delta(x,y)$, ossia un'immagine definita come l'impulso discreto

$$\delta(x,y) = \begin{cases} 1 & \text{se } x = y = 0 \\ 0 & \text{altrimenti} \end{cases} \tag{2.3.6}$$

Dall'analisi sappiamo che la trasformata dell'impulso è la funzione unitaria, quindi per l'immagine f si ha $F(u,v) = 1$; per questo motivo, utilizzando il teorema di convoluzione, l'output del filtro H è $\mathfrak{F}^{-1}[F(u,v)H(u,v)] = \mathfrak{F}^{-1}[H(u,v)]$, che è l'inversa della trasformata di Fourier del filtro nel dominio della frequenza, ovvero il corrispondente filtro $h(x,y)$ nel dominio dello spazio. Poiché h è l'output che si ha utilizzando il filtro H sull'impulso, essa viene chiamata risposta all'impulso. In sostanza il filtro H nel dominio di Fourier corrisponde al filtro spaziale h tramite la DFT.

2.3.1 Smoothing nel dominio di Fourier

Vediamo nel seguito di questa sezione, alcuni tipi di filtri, definiti nel dominio della frequenza, che eseguono lo smoothing dell'immagine; per ogni filtro daremo la definizione e mostreremo la rappresentazione della risposta all'impulso.

Lo smoothing, come già anticipato, si ottiene eliminando le alte frequenze dalla trasformata di Fourier dell'immagine di partenza; per questo motivo i filtri che verrano di seguito trattati saranno tutti del tipo passa-basso (lowpass). Questi si dividono sostanzialmente in tre categorie, in ordine decrescente di aggressività: ideali, Butterworth e gaussiano. Come vedremo, i filtri più aggressivi permettono uno smoothing maggiore, pur introducendo artefatti nell'immagine, spesso indesiderati e pericolosi.

2.3.1.1 Ideal Lowpass Filters

Un lowpass filter bidimensionale, che lascia invariate tutte le frequenze nel cerchio di raggio D_0 dall'origine e taglia qualunque frequenza al suo esterno, viene chiamato *ideal lowpass filter* (ILPF). Questo filtro ha un'espressione del tipo:

$$H(u,v) = \begin{cases} 1 & \text{se } D(u,v) \leq D_0 \\ 0 & \text{altrimenti} \end{cases} \tag{2.3.7}$$

con $D_0 > 0$ costante e $D(u,v)$, che è la distanza del punto (u,v) dal centro dell'immagine nel dominio della frequenza, di equazione:

$$D(u,v) = \sqrt{\left(u - \frac{P}{2}\right)^2 + \left(v - \frac{Q}{2}\right)^2} \tag{2.3.8}$$

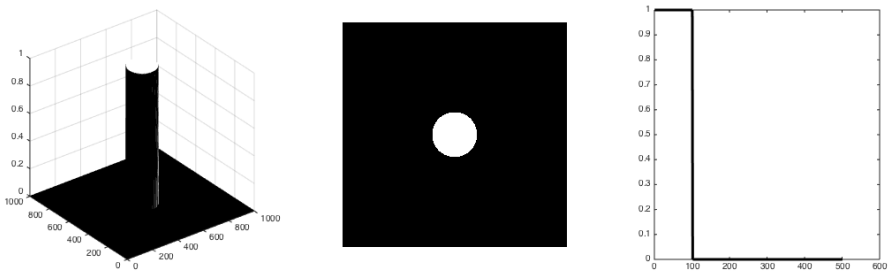
dove, ricordiamo, P e Q sono le dimensioni dell'immagine nel dominio della frequenza. La frequenza D_0, in cui avviene il repentino cambiamento della funzione, viene detta *frequenza critica*.

Utilizzando un filtro ILPF, tutte le frequenze in un certo intervallo vengono lasciate intatte, mentre le altre vengono tutte attenuate; un tale tipo di filtro, che passa bruscamente da 1 a 0, non può essere realizzato da componenti elettronici. L'aggettivo "ideale", presente nel nome del filtro, si riferisce proprio a questo aspetto.

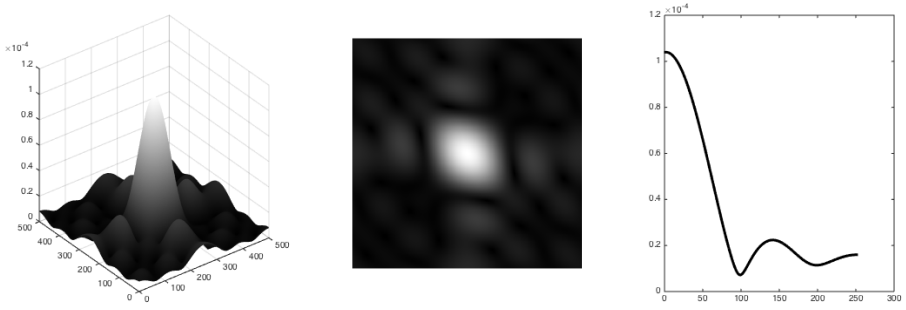
Spendiamo due parole per descrivere il cosiddetto effetto *ringing*: gli artefatti dovuti al ringing sono piccole anomalie presenti in prossimità di rapide transizioni dell'intensità dell'immagine; sono riconoscibili poiché formano delle bande di intensità anomala vicino ai contorni, oppure rendono l'effetto di sdoppiamento di parti dell'immagine. Cerchiamo ora di spiegare cosa causa gli effetti blur e ringing, che si hanno utilizzando questo tipo di filtro. In Figura 2.3.1(a) è rappresentato il filtro nel dominio della frequenza, mentre in Figura 2.3.1(b) è la rappresentazione spaziale. Possiamo notare come il corrispondente spaziale di un ILPF abbia sezione radiale simile ad una funzione sinc[1]. Possiamo rappresentare ogni punto (x_0, y_0) dell'immagine, d'intensità $f(x_0, y_0)$, come $\delta(x - x_0, y - y_0) \cdot f(x_0, y_0)$, dove $\delta(x,y)$ è l'impulso centrato nell'origine. Se eseguiamo la convoluzione di un impulso con la funzione sinc, otteniamo ancora una funzione sinc; questo significa che quando si va a calcolare la convoluzione di tutta l'immagine, che come abbiamo visto si compone di un certo numero di impulsi, con una funzione sinc, otteniamo un'immagine data dalla sovrapposizione di funzioni sinc, che creano interferenze e possono modificare di molto l'intensità di alcuni punti dell'immagine iniziale. L'effetto blur invece è dovuto all'attenuazione delle alte frequenze. Inoltre grazie al fatto che le ampiezze delle creste della funzione sinc sono inversamente proporzionali al raggio D_0 del filtro $H(u,v)$, nel caso limite in cui $D_0 \to \infty$ la funzione sinc diviene un impulso, non inserendo nessun effetto blurring né ringing nell'immagine una volta eseguita la convoluzione.

[1] La funzione sinc viene definita come:

$$\mathrm{sinc}(x) = \frac{\sin(x)}{x} \tag{2.3.9}$$



(a) Filtro ILPF di un'immagine 1000×1000 con $D_0 = 100$ nel dominio della frequenza.



(b) Filtro ILPF di un'immagine 1000×1000 con $D_0 = 5$ nel dominio spaziale.

Figura 2.3.1: Rappresentazione spaziale ed in frequenza di filtri ILPF. A partire da sinistra abbiamo una visuale in prospettiva, il filtro interpretato come immagine, la sezione radiale.

2.3.1.2 Butterworth Lowpass Filters

La funzione di trasferimento di un *Butterworth Lowpass Filter* (BLPF) di ordine n e frequenza critica D_0, ovvero la funzione che lo descrive nel dominio della frequenza, è definita come:

$$H(u,v) = \frac{1}{1 + \left|\frac{D(u,v)}{D_0}\right|^{2n}} \tag{2.3.10}$$

dove la distanza $D(u,v)$ è data ancora dalla (2.3.8). In Figura 2.3.2(a) possiamo vedere alcune rappresentazioni di un BLPF nel dominio della frequenza, invece in Figura 2.3.2(b) ne vediamo le corrispondenti rappresentazioni spaziali; i parametri utilizzati sono gli stessi di Figura 2.3.1, così da poterle confrontare.

Per quanto riguarda gli effetti di ringing e blurring, possiamo dire che un BLPF ha comportamenti differenti a seconda del grado n utilizzato. Infatti il filtro per $n = 1$ non genera alcun effetto ringing nel dominio dello spazio, un ringing quasi impercettibile è generalmente dato dai filtri di ordine $n = 2$, mentre infine diventa molto pronunciato in filtri di ordine superiore. Giusto per avere un'idea dell'effetto ottenuto al variare di n, si pensi che un BLPF di ordine 20 si comporta quasi come un ILPF (al limite per $n \to \infty$ un BLPF tende sempre ad un ILPF).

2.3.1.3 Gaussian Lowpass Filters

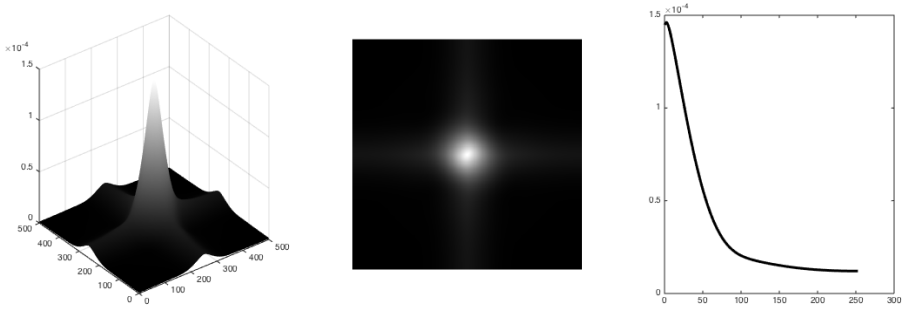
Si dice *Gaussian Lowpass Filter* (GLPF) un filtro che abbia una funzione di trasferimento della forma

$$H(u,v) = e^{\frac{-D^2(u,v)}{2D_0^2}} \tag{2.3.11}$$

dove per il calcolo della distanza abbiamo ancora usato la formula (2.3.8) e D_0 è la distanza della frequenza critica dal centro del dominio della frequenza.

Dai risultati dell'analisi sappiamo che la trasformata di Fourier di una gaussiana è anch'essa una gaussiana. Inoltre, guardando la sezione centrale del filtro gaussiano, vediamo che il profilo non subisce un cambiamento tanto ripido quanto quello del BLPF di ordine 2 per le stesse frequenze critiche. Questo ha ripercussioni dirette sulla capacità di smoothing dei due filtri: il GLPF ha un potere di smoothing minore rispetto al BLPF, pur non presentando imperfezioni dovute al ringing, al contrario, invece, il maggiore controllo offerto dal BLPF, può portare ad artefatti causati dal ringing. La scelta del filtro dipende, ovviamente, dal contesto in cui si usa: se, ad esempio, ci troviamo a modificare immagini mediche, non possiamo permetterci di introdurre artefatti, quindi saremo costretti a scegliere un filtro gaussiano.

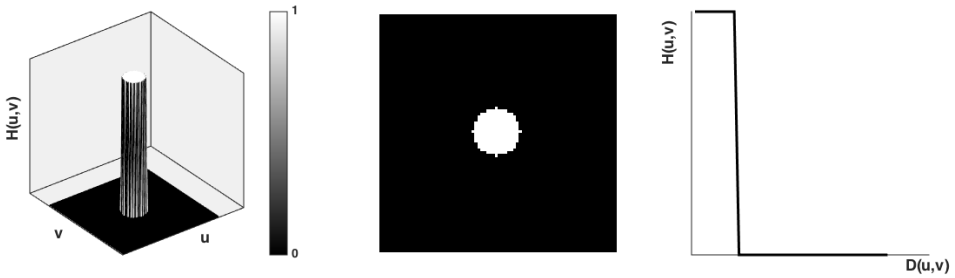
(a) Filtro BLPF di un'immagine 1000×1000 con $D_0 = 100$ nel dominio della frequenza.



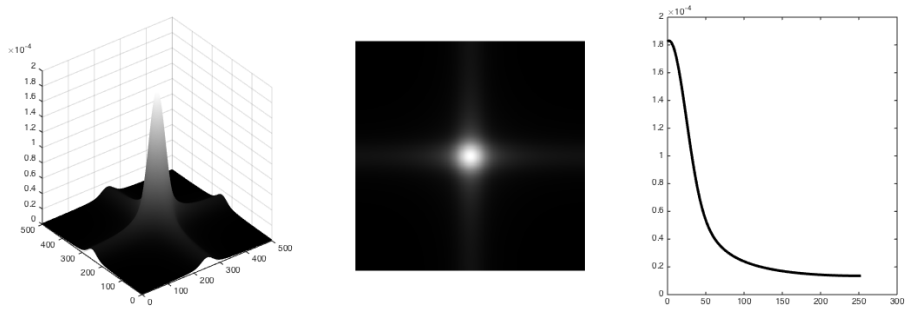
(b) Filtro BLPF di un'immagine 1000×1000 con $D_0 = 5$ nel dominio spaziale.

Figura 2.3.2: Rappresentazione spaziale ed in frequenza di filtri BLPF. A partire da sinistra abbiamo una visuale in prospettiva, il filtro interpretato come immagine, la sezione radiale.

In Figura 2.3.3(a) troviamo la rappresentazione di alcuni filtri GLPF; i parametri utilizzati sono gli stessi delle figure 2.3.2 e 2.3.1, in modo che sia chiaro il confronto tra le caratteristiche dei differenti filtri.



(a) Filtro GLPF di un'immagine 1000×1000 con $D_0 = 100$ nel dominio della frequenza.



(b) Filtro GLPF di un'immagine 1000×1000 con $D_0 = 5$ nel dominio spaziale.

Figura 2.3.3: Rappresentazione spaziale ed in frequenza di filtri GLPF. A partire da sinistra abbiamo una visuale in prospettiva, il filtro interpretato come immagine, la sezione radiale.

2.3.2 Sharpening nel dominio della frequenza

Abbiamo visto nella sezione precedente che lo smoothing si può ottenere attenuando le alte frequenze della trasformata di Fourier dell'immagine; poiché lo sharpening è il processo opposto allo smoothing, in questo caso dovremo tendere ad annullare le basse frequenze, senza disturbare le alte, ovvero dovremo utilizzare high-pass filters. Se indichiamo con $H_{LP}(u, v)$ la funzione di trasferimento di un lowpass filter, possiamo ottenere il corrispettivo highpass filter semplicemente con la formula

$$H_{HP}(u, v) = 1 - H_{LP}(u, v) \tag{2.3.12}$$

Come abbiamo fatto per il lowpass filters, anche in questa sezione esamineremo i filtri ideale, Butterworth e gaussiano, di nuovo elencati in ordine decrescente di aggressività. In aggiunta vedremo anche come operano i filtri laplaciano, unsharp masking e highboost filtering nel dominio della frequenza.

2.3.2.1 Ideal Highpass Filters

Un *ideal highpass filter* (IHPF) bidimensionale, ottenuto tramite la (2.3.12) da un ILPF, ha una funzione di trasferimento del tipo

$$H(u,v) = \begin{cases} 0 & \text{se } D(u,v) \leq D_0 \\ 1 & \text{altrimenti} \end{cases} \tag{2.3.13}$$

dove D_0 è la frequenza critica e $D(u,v)$ è ancora data dall'equazione (2.3.8). Questo filtro annulla tutte le frequenze comprese nel cerchio di raggio D_0, mentre lascia inalterate quelle frequenze che si trovano al suo esterno. Come nel caso di un ILPF, l'IHPF non è realizzabile con componenti elettronici ed è pertanto solamente ideale o simulabile al computer. Poiché derivato direttamente dall'ILPF, anche il tipo di filtro appena definito ha un consistente effetto ringing sull'immagine filtrata.

2.3.2.2 Butterworth Highpass Filters

La funzione di trasferimento di un *Butterworth highpass filter* (BHPF) di ordine n e frequenza critica D_0 si può ottenere dalla (2.3.12) ed ha equazione

$$H(u,v) = \frac{1}{1 + \left| \frac{D_0}{D(u,v)} \right|^{2n}} \tag{2.3.14}$$

dove anche in questo caso $D(u,v)$ è data dalla formula (2.3.8). Come nel caso dei lowpass filters, anche i BHPF hanno un comportamento meno aggressivo degli IHPF, con conseguente diminuzione della capacità di smoothing e dell'effetto ringing.

2.3.2.3 Gaussian Highpass Filters

Un *gaussian highpass filter* (GHPF), con frequenza critica posta a distanza D_0 dal centro del dominio della frequenza, ha funzione di trasferimento:

$$H(u,v) = 1 - e^{\frac{-D^2(u,v)}{2D_0^2}} \tag{2.3.15}$$

dove $D(u,v)$ è data (2.3.8). Come detto anche per i GLPF, questi filtri raggiungono il loro scopo in maniera più graduale rispetto agli altri, ma hanno più robustezza nei confronti dell'effetto ringing.

2.3.2.4 Il laplaciano nel dominio della frequenza

In aggiunta ai tre filtri appena proposti, dobbiamo fare un cenno anche al filtro laplaciano, che avevamo già trattato nel dominio spaziale. Si può mostrare (cfr. [11], Problema 4.26) che il laplaciano può essere implementato nel dominio della frequenza tramite la funzione di trasferimento

$$H(u,v) = -4\pi^2 D^2(u,v) \tag{2.3.16}$$

dove $D(u,v)$ è la distanza del punto (u,v) dal centro del dominio della frequenza, data dalla (2.3.8). Il laplaciano discreto dell'immagine si può ottenere allora come

$$\nabla^2 f(x,y) = \mathfrak{F}^{-1}\{H(u,v)F(u,v)\} \tag{2.3.17}$$

dove $F(u,v) = \mathfrak{F}(f(x,y))$ e $\mathfrak{F}$ è la DFT. Come fatto anche per il laplaciano, si ottiene la risposta del filtro all'immagine f con la formula

$$g(x,y) = f(x,y) - \nabla^2 f(x,y) \tag{2.3.18}$$

Nella pratica, prima di utilizzare la (2.3.18), poiché la DFT può aumentare di molto l'ordine di grandezza dei singoli elementi del laplaciano, occore riscalare sia l'immagine f, normalizzandola nell'intervallo $[0,1]$, sia il laplaciano ottenuto con la (2.3.17), dividendolo per il suo elemento massimo ed ottenendo valori nell'intervallo $[-1,1]$.

2.3.2.5 Unsharp Masking, Highboost Filtering

In questa parte discuteremo le tecniche di unsharp masking e highboost filtering, introdotte nella sezione 2.2, nel dominio della frequenza.

Come nel caso spaziale, dobbiamo creare la maschera per differenza tra l'immagine e la risposta dell'immagine ad un lowpass filter; questa è data dall'equazione (2.2.12), che ripetiamo di seguito per comodità:

$$g_{mask}(x,y) = f(x,y) - f_{LP}(x,y) \tag{2.3.19}$$

dove questa volta $f_{LP}(x,y)$ è la risposta dell'immagine $f(x,y)$ ad un lowpass filter con funzione di trasferimento $H_{LP}(u,v)$, per cui

$$f_{LP}(x,y) = \mathfrak{F}^{-1}[H_{LP}(u,v)F(u,v)] \tag{2.3.20}$$

con $F(u,v) = \mathfrak{F}[f(x,y)]$. Per ottenere l'immagine finale, come fatto in (2.2.13), dobbiamo aggiungere la maschera, eventualmente riscalata, all'immagine iniziale:

$$g(x,y) = f(x,y) + k \cdot g_{mask}(x,y) \tag{2.3.21}$$

La precedente equazione definisce l'unsharp masking per $k = 1$ e l'highboost filtering per $k > 1$. Ora possiamo riscrivere tutto nel dominio della frequenza, ottenendo

$$g(x,y) = \mathfrak{F}^{-1}\{[1 + k \cdot [1 - H_{LP}(u,v)]]\, F(u,v)\} \tag{2.3.22}$$

oppure, utilizzando un highpass filter

$$g(x,y) = \mathfrak{F}^{-1}\{[1 + k \cdot H_{HP}(u,v)]\, F(u,v)\} \tag{2.3.23}$$

L'espressione contenuta in parentesi quadre, ovvero $1 + k \cdot H_{HP}(u,v)$, può essere considerata la funzione di trasferimento di un filtro. Generalizziamo tale funzione per ottenere $k_1 + k_2 \cdot H_{HP}(u,v)$, con $k_1, k_2 \geq 0$; questo è un nuovo tipo di filtro chiamato high-frequency emphasis filter, la cui risposta dell'immagine è

$$g(x,y) = \mathfrak{F}^{-1}\{[k_1 + k_2 \cdot H_{HP}(u,v)]\, F(u,v)\} \tag{2.3.24}$$

Così facendo abbiamo creato un filtro che ha come casi particolari l'unsharp masking e l'highboost filtering, ma con un grado di libertà in più: infatti, k_1 controlla il discostamento dall'origine, mentre k_2 regola il contributo delle alte frequenze.

2.3.3 Filtri selettivi

Concludiamo la lista dei filtri più comuni parlando dei filtri selettivi, che permettono di manipolare solamente una determinata banda di frequenze, anziché tutto il dominio come avviene per i lowpass e highpass filters.

2.3.3.1 Bandreject e bandpass filters

Questi due tipi di filtri sono facili da costruire usando i concetti esposti nella sezione precedente. Supponiamo di voler costruire un bandreject filter; volendo per facilità e comodità utilizzare filtri simmetrici, la sua rappresentazione nel dominio della frequenza ricorderà una corona circolare centrata nel mezzo del dominio. Indichiamo con $D(u,v)$ la distanza del punto (u,v) dal centro del dominio delle frequenze, con D_0 il centro radiale della banda, ovvero la distanza media (raggio) dei punti della corona circolare dal suo centro, e con W la larghezza della banda. A partire dai filtri ideale, Butterworth e gaussiano si possono creare rispettivamente i seguenti bandreject filters:

- *ideal bandreject filter* (IBRF)

$$H(u,v) = \begin{cases} 0 & textseD_0 - \frac{W}{2} \leq D \leq D_0 + \frac{W}{2} \\ 1 & \text{altrimenti} \end{cases} \tag{2.3.25}$$

- *Butterworth bandreject filter* (BBRF)

$$H(u,v) = \frac{1}{1+\left[\frac{DW}{D^2-D_0^2}\right]^{2n}} \tag{2.3.26}$$

- *gaussian bandreject filter* (GBRF)

$$H(u,v) = 1 - e^{-\left[\frac{D^2-D_0^2}{DW}\right]^2} \tag{2.3.27}$$

I corrispondenti bandpass filters si possono ottenere facilmente come

$$H_{BP}(u,v) = 1 - H_{BR}(u,v) \tag{2.3.28}$$

Per chiarire le idee, la Figura 2.3.4 mostra un gaussian bandreject filter ed un gaussian bandpass filter, rappresentati in prospettiva nel dominio della frequenza.

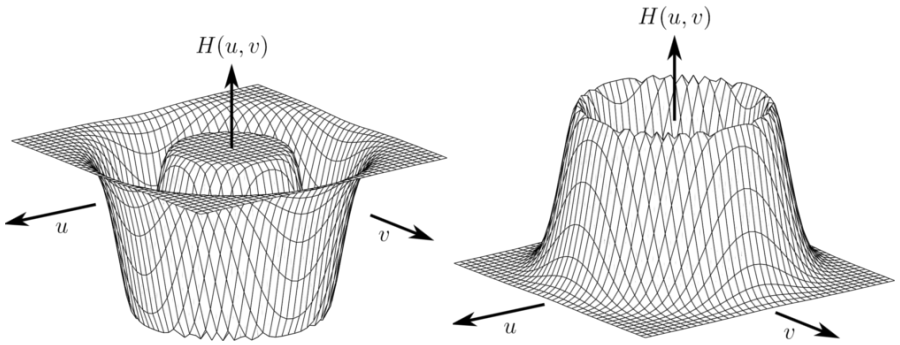


Figura 2.3.4: Da sinistra: rappresentazione prospettica di un gaussian bandreject filter e di un gaussian bandpass filter. Immagini prese da [3].

2.3.3.2 Notch filters

I notch filters sono i filtri selettivi più utili; essi permettono di attenuare tutte le frequenze nell'intorno di un particolare punto nel dominio della frequenza, oppure attenuare tutte quelle ad esso esterne. Dato che i filtri che costruiamo devono essere simmetrici, se scegliamo di modificare l'intorno del punto (u_0, v_0), dobbiamo per forza includere anche il punto diametralmente opposto $-(u_0, v_0)$.

Se vogliamo costruire un notch reject filter, basta definirlo tramite il prodotto di opportuni highpass filters centrati nei punti che ci interessano; la formulazione generale, quindi, è:

$$H_{NR}(u,v) = \prod_{k=1}^{Q} H_k(u,v) H_{-k}(u,v) \tag{2.3.29}$$

dove con H_k abbiamo indicato un highpass filter con centro in (u_k, v_k), con H_{-k} l'highpass filter con centro opposto $-(u_k, v_k)$ e Q è il numero di coppie di punti attorno cui vogliamo attenuare le frequenze. Per il calcolo degli highpass filters si dovranno usare le distanze

$$\begin{aligned} D_k(u,v) &= \sqrt{\left(u - \frac{M}{2} - u_k\right)^2 + \left(v - \frac{N}{2} - v_k\right)^2} \\ D_{-k}(u,v) &= \sqrt{\left(u - \frac{M}{2} + u_k\right)^2 + \left(v - \frac{N}{2} + v_k\right)^2} \end{aligned} \tag{2.3.30}$$

I corrispondenti notch pass filters si possono ottenere, come consueto, tramite la formula

$$H_{NP}(u,v) = 1 - H_{NR}(u,v) \tag{2.3.31}$$

In Figura 2.3.5 possiamo osservare la rappresentazione in prospettiva, nel dominio della frequenza, dei tre principali notch reject filters: ideal, Butterworth e gaussian. Essi sono stati generati utilizzando come highpass filters i corrispondenti ideal, Butterworth e gaussian highpass filters. I notch filters utilizzati per la Figura 2.3.5 sono stati creati con una sola coppia di buchi, quindi $Q = 1$, per semplificare la visualizzazione.

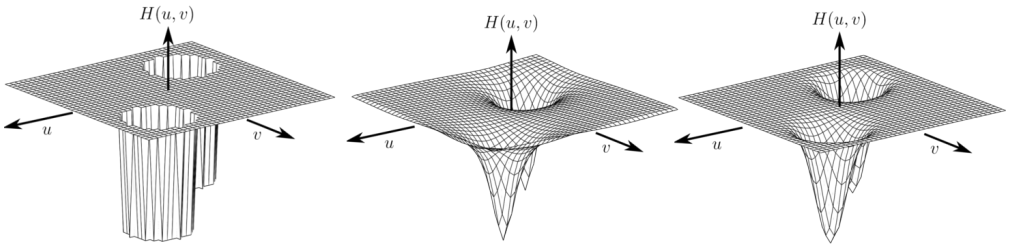


Figura 2.3.5: Rappresentazione prospettica di tre notch reject filters ad una sola coppia di buchi. Da sinistra vediamo un ideal notch reject filter, un Butterworth notch reject filter ed infine un gaussian notch reject filter. Immagini prese da [3].

2.3.4 Implementazione

Per ora abbiamo focalizzato la nostra attenzione sugli aspetti teorici delle tecniche di filtraggio, lasciando da parte il loro utilizzo. In effetti va considerato che i requisiti computazionali per il calcolo di una DFT non sono affatto banali; basti pensare che utilizzare un metodo brute-force per il calcolo della DFT di un immagine di dimensione $M \times N$ costerebbe $(MN)^2$ operazioni: per un immagine

ordinaria, di dimensioni 1024×1024, si richiederebbe un numero di operazioni dell'ordine di 10^{12}.

Riproponiamo per comodità la formula per il calcolo della DFT di un'immagine $f(x,y)$ di dimensioni $M \times N$, già data nella (2.3.1):

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{2.3.32}$$

Notiamo che è possibile separare il calcolo di una DFT bidimensionale ed ottenere due DFT unidimensionali:

$$F(u,v) = \sum_{x=0}^{M-1} e^{-j2\pi\frac{ux}{M}} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi\frac{vy}{N}} = \sum_{x=0}^{M-1} F(x,v) e^{-2j\pi\frac{ux}{M}} \tag{2.3.33}$$

dove $F(x,v)$ è la trasformata di Fourier discreta unidimensionale della riga x. Quindi la DFT bidimensionale può essere calcolata eseguendo le DFT unidimensionali di tutte le righe, quindi, a partire dal risultato ottenuto, si calcolano le DFT unidimensionali di tutte le colonne.

Riportiamo di seguito l'equazione (2.3.2) che descrive l'inversa della trasformata di Fourier discreta:

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{2.3.34}$$

Prendiamo il complesso coniugato della (2.3.34) e moltiplichiamolo per MN:

$$MN\, f^*(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F^*(u,v) e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{2.3.35}$$

Al secondo membro possiamo riconoscere la DFT di $F^*(u,v)$; questo significa che se utilizziamo $F^*(u,v)$ come input per il calcolo della DFT, otteniamo $MNf^*(x,y)$. Prendendone il complesso coniugato e quindi moltiplicandolo per MN, otteniamo $f(x,y)$, che è la trasformata inversa di $F(u,v)$. In sostanza ci basta trovare un algoritmo rapido per la DFT, poiché la IDFT può essere calcolata tramite lo stesso algoritmo.

2.3.4.1 Fast Fourier Transform (FFT)

Nonostante ci sia una vasta letteratura riguardante la FFT ed i vari algoritmi che la implementano, noi ci limitiamo, come in [11], a citare il metodo del

successivo sdoppiamento; questo infatti fu l'algoritmo che pose le basi per tutta la successiva ricerca sulle FFT. Questo particolare algoritmo assume che le dimensioni dell'immagine siano potenze di 2; esistono comunque altri metodi in letteratura che non prevedono restrizioni così pesanti, si veda ad esempio [2].

Sappiamo già che la DFT in due dimensioni si riduce al calcolo della DFT unidimensionale, pertanto basta che ci concentriamo sul calcolo di quest'ultima. La trasformata di Fourier della funzione $f(x)$ è la seguente:

$$F(u) = \sum_{x=0}^{M-1} f(x) e^{-j2\pi \frac{ux}{M}} \tag{2.3.36}$$

che, tramite la sostituzione $W_M = e^{-\frac{2j\pi}{M}}$, può essere riscritta

$$F(u) = \sum_{x=0}^{M-1} f(x) W_M^{ux} \tag{2.3.37}$$

con $u = 0, 1, \dots, M-1$. Per le assunzioni fatte possiamo scrivere $M = 2^n$, per qualche intero positivo n. Possiamo anche scrivere $M = 2K$, per qualche intero positivo K. A questo punto la (2.3.37) diviene

$$\begin{aligned} F(u) &= \sum_{x=0}^{2K-1} f(x) W_{2K}^{ux} \\ &= \sum_{x=0}^{K-1} f(2x) W_{2K}^{u(2x)} + \sum_{x=0}^{K-1} f(2x+1) W_{2K}^{u(2x+1)} \end{aligned} \tag{2.3.38}$$

Dalla definizione di W_M si deriva facilmente che $W_{2K}^{2ux} = W_K^{ux}$, per cui scriviamo

$$F(u) = \sum_{x=0}^{K-1} f(2x) W_K^{ux} + \sum x = 0^{K-1} f(2x+1) W_K^{ux} W_{2K}^{u} \tag{2.3.39}$$

Definiamo ancora

$$\begin{aligned} F_{pari}(u) &= \sum_{x=0}^{K-1} f(2x) W_K^{ux} \\ F_{dispari}(u) &= \sum_{x=0}^{K-1} f(2x+1) W_K^{ux} \end{aligned} \tag{2.3.40}$$

per $u = 0, 1, \dots, K-1$. A questo punto la (2.3.39) si può riscrivere come

$$F(u) = F_{pari}(u) + F_{dispari}(u) \cdot W_{2K}^{u} \tag{2.3.41}$$

Un'altra proprietà interessante viene fuori dall'osservazione che $W_M^{u+M} = W_M^u$ e che $W_{2M}^{u+M} = -W_{2M}^u$; sfruttando queste relazioni, infatti, si può dire che

$$F(u+K) = F_{pari}(u) - F_{dispari}(u) W_{2K}^{u} \tag{2.3.42}$$

Vediamo allora che il calcolo della DFT si può svolgere agevolmente; per calcolare la DFT di M punti, si può utilizzare la (2.3.41), quindi calcolare separatamente le due trasformate di $\frac{M}{2}$ punti F_{pari} ed $F_{dispari}$. L'utilizzo della (2.3.41) può essere iterato diminuendo piano piano la complessità del calcolo della DFT. I valori di F_{pari} ed $F_{dispari}$ andranno poi sostituiti nella (2.3.41) per calcolare i $K-1$ valori della $F(u)$; i restanti valori si posso ottenere, senza calcolare altre trasformate, a partire dalla (2.3.42).

La scoperta della Fast Fourier Transform ha permesso l'utilizzo effettivo della DFT, rendendo sfruttabile tutto ciò di cui si è parlato in questo capitolo. Per rendersi conto del beneficio computazionale apportato dalla FFT, si pensi che ha un costo di $MN \log_2 MN$, rispetto al metodo brute-force con $(MN)^2$; ora occorrono "solamente" $2 \cdot 10^7$ operazioni elementari, contro le 10^{12} del metodo brute-force, per il calcolo della DFT di un'immagine 1024×1024.

2.4 Metodi di segmentazione

Nelle sezioni precedenti abbiamo discusso metodi per il miglioramento delle immagini, tramite algoritmi che ne prevedevano una sia come input, sia come output. Ora ci muoviamo in direzione differente: tratteremo procedure che, a partire da immagini, estraggono alcuni loro attributi di interesse particolare. La segmentazione è il primo passo verso tale obiettivo e si prefigge, in particolare, di suddividere l'immagine nelle sue regioni costituenti; questo è uno dei compiti più ardui nell'ambito dell'elaborazione d'immagini: l'accuratezza della segmentazione è, per molti algoritmi di analisi, l'ago della bilancia tra il successo ed il fallimento.

Iniziamo la trattazione, come solito, introducendo la notazione ed i concetti preliminari a questa sezione. Sia R la regione occupata dall'immagine; possiamo vedere la segmentazione come il processo che conduce a partizionare R in n sotto-regioni $R_1, R_2, \dots, R_n$ tali che:

1. $\bigcup_{i=1}^{n} R_i = R$

2. R_i connesso $\forall i = 1, 2, \dots, n$
3. $R_i \cap R_j = \emptyset$ per ogni $i \neq j$
4. $Q(R_i)\ \forall i = 1, 2, \dots, n$
5. $\neg Q(R_i \cup R_j)$ per ogni R_i e R_j *adiacenti*, ovvero per ogni $R_i \cup R_j$ connesso

dove Q è un predicato logico definito sui punti degli insiemi R_i. Segmentare, quindi, significa partizionare seguendo il criterio indicato dal predicato Q: se $Q =$ "contiene punti con la stessa intensità", il risultato della segmentazione saranno regioni i cui punti hanno tutti la stessa intensità, ma regioni adiacenti devono per forza avere intensità differenti. Esistono fondamentalmente due tecniche di partizionamento: segmentazione *edge-based* e *region-based*, che hanno rispettivamente a che fare con discontinuità e similarità tra i valori di intensità dell'immagine.

2.4.1 Indiduare punti isolati, linee e contorni

Inizieremo parlando dei metodi edge-based, il cui passo preliminare è individuare le variazioni di intensità, quindi scovare punti isolati, linee e contorni. Metodi che si prefiggono questo obiettivo vengono chiamati *edge detector*.

Già sappiamo dalla sezione 2.2 che filtri spaziali basati sugli operatori differenziali, laplaciano e gradiente, esaltano i contorni dell'immagine. La Tabella 2.1 riassume le proprietà delle derivate prime e seconde dell'immagine, che abbiamo già evidenziato in precedenza.

2.4.1.1 Individuare i punti isolati

Dalle proprietà generali elencate nella Tabella 2.1, è ragionevole considerare punti isolati le coppie (x, y) in cui il modulo del laplaciano è superiore ad una soglia, stabilita a priori. A tali punti assegniamo il valore 1, lasciando a 0 tutti i rimanenti: così facendo creiamo un'immagine binaria in cui il valore "vero" corrisponde ai punti isolati. Riassumendo tutto in una formula, il risultato di questo procedimento è

$$g(x, y) = \begin{cases} 1 & \text{se } \left|\nabla^2 f(x, y)\right| \geq T \\ 0 & \text{altrimenti} \end{cases} \tag{2.4.1}$$

dove con $\nabla^2 f$ abbiamo indicato il laplaciano dell'immagine iniziale $f(x, y)$ e con T il valore di soglia fissato.

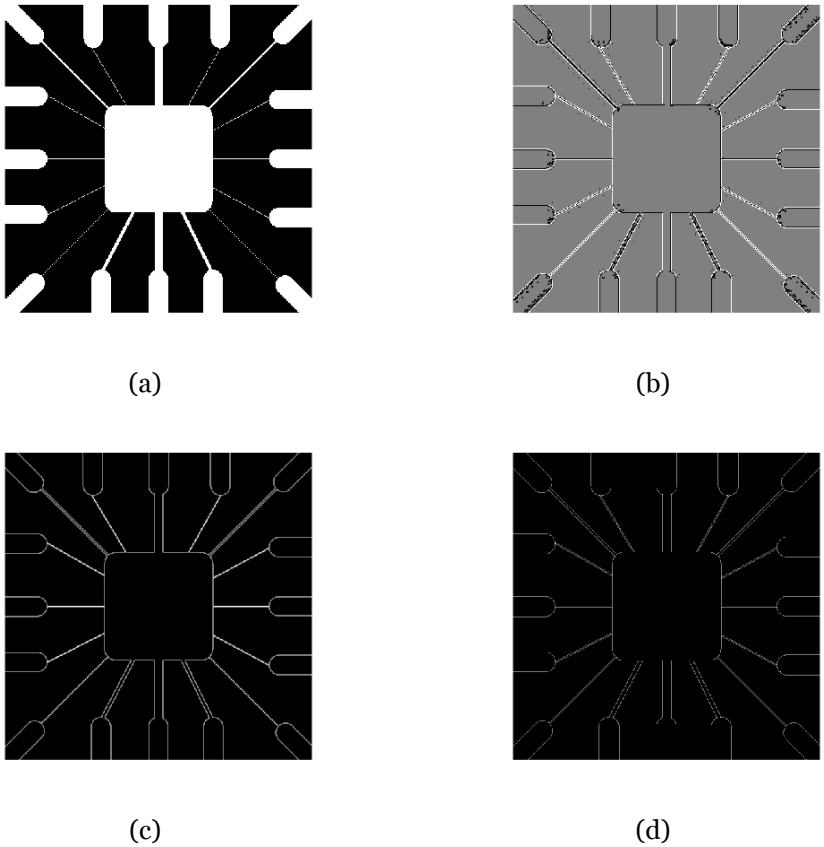
Derivate prime	Derivate seconde
• si annullano sulle aree ad intensità costante • sono non nulli sulle rampe • inspessiscono i contorni	• si annullano sulle aree ad intensità costante • sono non nulli ad inizio e fine rampa, raddoppiando così i contorni • il segno può essere usato per capire se un contorno determina una transizione verso valori di intensità maggiori o minori • hanno una risposta più forte sui piccoli dettagli (linee sottili, punti isolati e rumore puntiforme)

Tabella 2.1: Principali proprietà delle derivate prime e seconde rispetto ai valori di intensità dell'immagine.

2.4.1.2 Individuare le linee

Il livello di complessità successivo è l'individuazione delle linee. Dalla Tabella 2.1 si evince che la derivata seconda può essere utilizzata anche per trovare le linee nelle immagini, purché si gestisca lo sdoppiamento dei contorni.

L'esempio che segue, tratto da [11], può essere d'aiuto nel comprendere come individuare linee tramite laplaciano. Nella Figura 2.4.1(a) possiamo vedere l'immagine di una maschera usata in elettronica per il wire-bonding, ovvero la principale tecnica per realizzare connessioni tra circuiti integrati e schede per circuito stampato; nella Figura 2.4.1(b) abbiamo invece il laplaciano dell'immagine, riscalato ai fini della visualizzazione, in modo che sia bianco per valori positivi, grigio dove si annulla e nero nelle zone in cui è negativo. Si vede molto chiaramente l'effetto di sdoppiamento delle linee; nel tentativo di eliminare questo effetto, dovuto al segno opposto del laplaciano ad inizio e fine rampa, potremmo pensare di utilizzare il valore assoluto del laplaciano: l'immagine che si ottiene è mostrata in Figura 2.4.1(c). Vediamo subito che l'immagine presenta linee molto spesse; poiché questo non è desiderabile, passiamo ad un altro approccio: prendiamo solamente i valori positivi del laplaciano (eventualmente, in caso di molto rumore, valori superiori ad una soglia prestabilita). Notiamo nella Figura 2.4.1(d) come questo approccio produca linee più sottili dei precedenti,



(a) (b)

(c) (d)

Figura 2.4.1: Individuazione delle linee in una maschera usata in elettronica per il wire-bonding. (a) Immagine originale. (b) Laplaciano dell'immagine. (c) Valore assoluto del laplaciano. (d) Valori positivi del laplaciano.

sicuramente più utili all'atto pratico.

Nelle ultime tre immagini si nota che, quando le linee sono larghe se paragonate alla larghezza della maschera del laplaciano, tali linee sono separate da una zona a laplaciano nullo. Questo è un comportamento del tutto atteso, infatti, se la linea è più larga del filtro, quest'ultimo la considererà una regione, assegnandole di conseguenza valore nullo.

Il laplaciano è un filtro isotropico, quindi la risposta è indipendente dalla direzione; ciò non è sempre gradito, per esempio se vogliamo localizzare solamente linee verticali. A questo scopo ci vengono in aiuto le seguenti maschere:

-1	-1	-1
2	2	2
-1	-1	-1

2	-1	-1
-1	2	-1
-1	-1	2

-1	2	-1
-1	2	-1
-1	2	-1

-1	-1	2
-1	2	-1
2	-1	-1

A partire da quella a sinistra, esse metteranno più in risalto, rispettivamente, linee orizzontali, linee ruotate di 45° in senso orario, linee verticali, linee ruotate di 45° in senso antiorario.

2.4.1.3 Individuare i contorni

L'individuazione dei contorni è l'approccio più frequentemente utilizzato per la segmentazione delle immagini basata su brusche variazioni di intensità. I contorni si possono modellizzare principalmente nei tre seguenti modi.

- Gli *step edges*, ovvero "a gradino", per cui si ha tutta la variazione d'intensità nello spazio di un pixel; contorni di questo tipo occorrono in genere nelle immagini create al pc, ma sono rari nelle immagini reali, per questo si dice che sono ideali.
- Molto più comuni sono le rampe (*ramp edges*); le immagini reali, infatti, appaiono più sfocate, a causa delle limitazioni della meccanica del dispositivo di acquisizione, e ricche di rumore, principalmente per colpa dei componenti elettronici. La pendenza della rampa è inversamente proporzionale al livello di blur dell'immagine; invece di avere un contorno largo un solo pixel, come nel primo caso, qui ogni punto della rampa viene considerato parte del margine.
- Il terzo modello per i contorni prende il nome di *roof edge*, ovvero "a forma di tetto"; generalmente un roof edge modellizza una linea immersa in una regione, con la base del tetto proporzionale alla larghezza ed alla nitidezza della linea.

Nella realtà potremmo non trovare esattamente questi tre modelli, ma se la quantità di rumore presente non è eccessiva, si può utilizzare, con buoni risultati, il tipo di contorno che maggiormente approssima quello reale. Scegliere il modello giusto è fondamentale, poiché ci permette di avere un'espressione matematica che descriva al meglio i margini, cosa utile per lo sviluppo di un algoritmo; risulta evidente, di conseguenza, che l'utilizzo del modello sbagliato provoca un sostanziale peggioramento del risultato dell'algoritmo di edge detection.

In generale le tecniche per l'individuazione dei contorni seguono la scaletta seguente:

1. smoothing per ridurre il rumore
2. individuazione dei margini, ovvero un'operazione finalizzata all'estrazione dei punti di contorno candidati

3. localizzazione dei margini, cioè selezione dei punti che fanno effettivamente parte di un contorno.

Per raggiungere gli obiettivi appena posti, utilizzeremo principalmente il gradiente, per il suo comportamento lungo le rampe. Non ci soffermiamo a trattare come poter calcolare l'immagine gradiente, ovvero l'immagine con il modulo del gradiente in ogni punto, poiché già discussa nella sezione 2.2. Aggiungiamo solamente un punto, a completamento della trattazione già esposta: nel momento in cui si avesse bisogno di un operatore di Sobel con una risposta più forte lungo le diagonali, anziché lungo gli assi cartesiani, si possono utilizzare delle modifiche agli operatori standard, ovvero:

0	1	2
−1	0	1
−2	−1	0

−2	−1	0
−1	0	1
0	1	2

2.4.1.4 L'algoritmo di Marr-Hildreth

Uno dei primi tentativi (ben riusciti) di fare un'analisi più sofisticata del processo d'individuazione dei contorni è dovuta a Marr e Hildreth [24]; prima del loro contributo, venivano utilizzati operatori di piccola taglia come quelli di cui abbiamo discusso finora. Due ipotesi, ben argomentate, pongono le basi all'algoritmo che stiamo per descrivere:

1. le variazioni d'intensità non sono indipendenti dalla dimensione dell'immagine, quindi per l'individuazione dei contorni abbiamo bisogno di filtri di dimensione opportuna

2. bruschi cambiamenti nell'intensità danno luogo a picchi e solchi nel grafico della derivata prima e contemporaneamente al cambio di segno per la derivata seconda.

I due scienziati argomentarono che il filtro che sembrava soddisfare al meglio tutti i requisiti fosse il laplaciano di una gaussiana, indicato con $\nabla^2 G$ oppure LoG. L'argomentazione si basa su due punti fondamentali: da un lato, l'uso del filtro gaussiano serve a rendere l'effetto blurring, utile a rimuovere il rumore e strutture di dimensione inferiore a quella del filtro, senza introdurre artefatti dovuti, ad esempio, al ringing; d'altra parte la scelta del laplaciano, rispetto ad altri operatori differenziali, serve ad individuare bruschi cambi di intensità, inoltre abbiamo a che fare con un operatore isotropico per cui non ci dobbiamo preoccupare di utilizzare molteplici maschere per calcolare la risposta più elevata in ogni punto dell'immagine. Sia

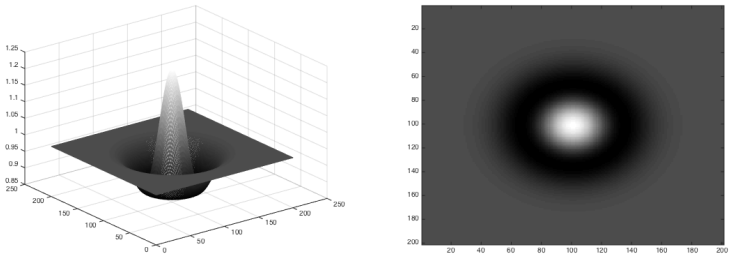
$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.4.2}$$

una gaussiana bidimensionale con deviazione standard σ. Per trovare l'espressione finale del filtro calcoliamo il laplaciano di questa espressione:

$$\begin{aligned}\nabla^2 G(x,y) &= \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2} \\ &= \frac{\partial}{\partial x}\left[\frac{-x}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}\right] + \frac{\partial}{\partial y}\left[\frac{-y}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}\right] \\ &= \left[\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} + \left[\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}\end{aligned} \tag{2.4.3}$$

Quindi infine

$$\nabla^2 G(x,y) = \left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.4.4}$$



(a) (b)

Figura 2.4.2: Rappresentazione del negativo del filtro LoG descritto dalla (2.4.4). (a) Rappresentazione prospettica. (b) Rappresentazione come immagine.

Nella Figura 2.4.2(a) è mostrato il grafico del negativo della funzione definita nella (2.4.4); nella pratica infatti viene sempre usato $-LoG$. Dall'equazione si può vedere come la funzione $\nabla^2 G$ si annulli lungo tutta la circonferenza $x^2+y^2 = 2\sigma^2$, di centro l'origine e raggio $\sqrt{2}\sigma$. Per utilizzare la funzione appena definita, occorre creare una maschera a partire dai suoi valori; quello che serve è, quindi, campionare la funzione, scegliendo la grandezza della maschera in modo tale da mantenere la forma e le caratteristiche della funzione e, non meno importante, far sì che la somma degli elementi del filtro sia nulla (questo serve per avere risposta nulla in aree costanti).

Operativamente il metodo di Marr-Hildreth consiste nell'eseguire la convoluzione della maschera ricavata dalla funzione LoG con l'immagine:

$$g(x,y) = [\nabla^2 G(x,y)] * f(x,y) \tag{2.4.5}$$

Dalle proprietà di linearità segue facilmente che

$$g(x,y) = \nabla^2 [G(x,y) * f(x,y)] \tag{2.4.6}$$

Di conseguenza l'algoritmo di Marr-Hildreth si articola come segue:

- filtrare l'immagine con un filtro gaussiano di dimensione $n \times n$;
- calcolare il laplaciano dell'immagine usando, per esempio, la maschera vista nella sezione 2.2;
- calcolare i punti corrispondenti a cambi di segno dell'immagine ottenuta.

Bisogna fare attenzione al valore da assegnare a n. Ricordiamo, infatti, che stiamo decidendo la dimensione di un filtro gaussiano e che circa il 99.7% del volume della funzione (2.4.2) è contenuto nel cerchio di raggio 3σ centrato all'origine. Per preservare la struttura della gaussiana, ovvero per non troncarla, dobbiamo scegliere n come il più piccolo intero dispari superiore a 6σ.

Per quanto riguarda il calcolo dei punti in cui l'immagine cambia segno, un possibile approccio è ispezionare l'intorno 3×3 di ogni punto p. Affinché vi sia un cambio di segno è necessario che almeno due punti vicini, diametralmente opposti rispetto a p, abbiamo segno opposto; in sostanza vi sono solo quattro direzioni in cui guardare, ovvero verticale, orizzontale e le due diagonali. Se vogliamo anche utilizzare un valore di soglia, per eliminare cambi di segno non significativi, allora i due punti vicini, diametralmente opposti, non dovranno solo avere segno differente, ma la loro differenza, in modulo, dovrà essere superiore al valore prestabilito.

2.4.1.5 L'algoritmo di Canny

Nonostante l'algoritmo sia più complesso, le prestazioni del metodo di Canny sono superiori a tutti i precedenti edge detectors. Gli obiettivi che stanno alla base di questo nuovo algoritmo sono:

- bassa possibilità di errore
- contorni ben localizzati
- spessore possibilmente unitario dei margini, ovvero il numero di punti restituiti attorno ai massimi locali deve essere minimo.

Canny riuscì a trasformare in formule matematiche tutti questi obiettivi, quindi, usando alcune approssimazioni numeriche e modellizzando i margini con step

edges corrotti da rumore gaussiano, riuscì a dimostrare che una buona approssimazione all'edge detector ottimale potesse essere la derivata di una gaussiana, ovvero:

$$\frac{d}{dx}e^{-\frac{x^2}{2\sigma^2}} = \frac{-x}{\sigma^2}e^{-\frac{x^2}{2\sigma^2}} \tag{2.4.7}$$

La generalizzazione di tale discussione al caso bidimensionale è semplice, purché si applichi lungo la direzione normale al contorno; poiché questa non è nota a priori, occorre prima di tutto eseguire uno smoothing con un filtro gaussiano bidimensionale, quindi calcolare il gradiente dell'immagine, il modulo $M(x, y)$ e la direzione in ogni punto (indichiamo questa immagine con $\sigma(x, y)$).

Usare il gradiente per trovare i contorni, significa ottenere linee molto spesse, con molti punti considerati parte del margine. Per soddisfare i requisiti stabiliti ad inizio paragrafo, è necessario assottigliare queste strutture; lo faremo tramite la soppressione dei punti che non sono massimi locali. Specifichiamo, innanzitutto, un certo numero di possibili orientazioni del gradiente $d_1, d_2, \dots, d_k$; approssimiamo le direzioni calcolate in precedenza con queste ultime, in modo da diminuire il carico computazionale. Scorriamo ogni punto (x, y) dell'immagine gradiente M, per creare l'immagine g_N, in cui i punti che non sono massimi locali vengono soppressi; eseguiamo pertanto quanto segue:

- prendiamo la direzione d_k più vicina a $\sigma(x, y)$;
- consideriamo l'intorno del punto corrente e siano $q_1, \dots, q_l$ i punti dell'intorno lungo la direzione indicata da d_k; se esiste qualche i per cui $M(x, y) < M(q_i)$, sopprimiamo il punto (x, y) con $g_N(x, y) = 0$, poiché non è un massimo locale, altrimenti assegniamo $g_N(x, y) = M(x, y)$.

L'operazione finale è applicare un threshold per ridurre i contorni non corrispondenti al vero. Utilizzare un solo valore di soglia è poco utile, poiché, se lo fissiamo troppo basso, rimarrebbero comunque dei falsi positivi, se al contrario fosse troppo alto, avremmo dei falsi negativi, andando ad eliminare contorni veri. Canny tentò di superare questo scoglio utilizzando una tecnica che prende il nome di *hysteresis thresholding*, in cui si utilizzano due valori di soglia T_H e T_L, con $T_L < T_H$; lo stesso autore suggerì di prendere T_H due o tre volte maggiore di T_L. Possiamo, allora, creare due immagini in risposta ai due threshold, diciamo:

$$\begin{aligned} g_{NH}(x, y) &= g_N(x, y) \geq T_H \\ g_{NL}(x, y) &= g_N(x, y) \geq T_L \end{aligned} \tag{2.4.8}$$

In generale si avranno in g_{NH} meno punti accettati che in g_{NL}, però quelli verranno considerati validi. Eliminiamoli allora da g_{NL} tramite

$$g_{NL}(x, y) = g_{NL}(x, y) - g_{NH}(x, y) \tag{2.4.9}$$

A seconda del valore di T_H scelto si avranno dei buchi, più o meno grandi, nei contorni individuati nell'immagine g_{NH}; riempiamoli tramite l'algoritmo seguente.

1. Sia p il successivo punto non ancora visitato di $g_{NH}(x, y)$.
2. Segniamo come validi tutti i punti di $g_{NL}(x, y)$ connessi con p (secondo qualche regola decisa a priori).
3. Se tutti i punti di $g_{NH}(x, y)$ sono stati visitati allora passiamo al punto 4, altrimenti torniamo a 1.
4. Diamo un valore nullo ad ogni punto di $g_{NL}(x, y)$ non segnato come valido.

La risposta dell'algoritmo di Canny si ottiene, infine, sovrapponendo i punti validi di $g_{NH}(x, y)$ con quelli di $g_{NL}(x, y)$.

2.4.1.6 Perfezionamento dei risultati degli edge detectors

Gli edge detectors restituiscono l'insieme dei punti che giacciono sui margini dell'immagine, però potrebbe non esserci piena corrispondenza a causa del rumore, della differenza d'illuminazione e altri possibili anomalie dell'immagine. La risposta ad un edge detector potrebbe, di conseguenza, essere costituita da punti isolati, linee discontinue e contorni esatti. Per le applicazioni pratiche occorre utilizzare algoritmi che riescano a restituire la forma corretta a tali insiemi di punti; tre sono gli approcci che si possono seguire, in base al tipo di applicazione: locale, regionale e globale.

Approccio locale Uno dei più semplici approcci per il collegamento di contorni interrotti è analizzare l'intorno di ogni pixel selezionato dagli edge detectors. Tutti i punti che sono simili, in accordo a qualche criterio deciso a priori, vengono collegati, formando un unico contorno, fatto da punti che condividono caratteristiche comuni.

Uno dei criteri più usati è quello descritto di seguito. Per ogni punto (x, y), consideriamo il suo intorno S_{xy}; supponiamo di trovare un punto $(s, t) \in S_{xy}$ che ha modulo del gradiente simile a (x, y):

$$\left|M(x, y) - M(s, t)\right| \leq E \tag{2.4.10}$$

dove E è un predefinito valore di soglia. Sicuramente il punto (s, t) è candidato ad essere collegato al punto (x, y), poiché ne condivide una caratteristica. Dobbiamo però verificare anche un altro requisito: la direzione del gradiente deve

essere anch'essa simile, facendo in modo che il primo punto sia associabile al secondo non solo per l'elevazione rispetto ai pixel circostanti, ma anche per come è orientata tale configurazione. In formule dobbiamo, quindi, verificare

$$\left|\alpha(x,y) - \alpha(s,t)\right| \leq A \tag{2.4.11}$$

dove con A abbiamo indicato un valore di soglia per l'orientazione del gradiente e con $\alpha(x,y)$ l'orientazione nel punto (x,y).

Questo semplice algoritmo collega infine coppie di punti che soddisfano sia la (2.4.10) che la (2.4.11), dando ai contorni una forma che si avvicini maggiormente alla realtà.

Approccio a regioni Spesso, nelle applicazioni pratiche, si conosce la posizione delle regioni di interesse in un'immagine; se questa conoscenza è disponibile, è bene utilizzare degli algoritmi per il perfezionamento dei contorni che ne facciano uso. In particolare, la maggior parte di tali metodi sfrutta le informazioni sull'appartenenza dei contorni individuati dagli edge detectors a particolari regioni. Sapendo, di conseguenza, per quali punti passa il contorno reale della regione, possiamo adattare facilmente una curva 2D su tali punti; generalmente, poiché l'interesse è spesso rivolto verso tecniche dalla rapida esecuzione, si approssimano i contorni con poligonali, che hanno la proprietà di mantenere la maggior parte delle caratteristiche dei margini, pur essendo estremamente facili da calcolare. Non ci addentriamo nella discussione degli algoritmi di *fitting*, poiché non è interessante ai fini di questo lavoro di tesi; numerosi risultati si possono trovare in letteratura, ma, volendoci limitare a fare un semplice esempio, rimandiamo all'algoritmo definito in [11].

Approccio globale Capita in maniera altrettanto frequente che le informazioni di cui abbiamo parlato nel paragrafo precedente non siano disponibili. Non potendo sapere quali punti facciano parte dello stesso contorno, abbiamo bisogno di ricorrere a tecniche con prospettiva globale, che analizzano le similarità tra i pixel utilizzando proprietà definite su tutta l'immagine.

Uno degli strumenti più frequentemente utilizzati è la *trasformata di Hough*, un potente strumento che permette di analizzare la distribuzione dei punti di un insieme nello spazio dei parametri di una particolare equazione. Per essere più specifici: supponiamo di sapere che un dato insieme di punti Ω sia distribuito lungo rette e di volerne trovare le equazioni; resta sicuramente difficile provare tutte le possibili combinazioni dei parametri per determinare le equazioni che meglio descrivono i punti. Si può invece eseguire la seguente procedura.

- Esprimiamo le equazioni delle rette da trovare tramite le coordinate polari, ovvero mediante la fase θ e la distanza dall'origine ρ:

$$x \cos\theta + y \sin\theta = \rho \tag{2.4.12}$$

- Se conosciamo qualche informazione sui possibili valori di ρ e θ, non risulta difficile trovarne i massimi e minimi; in base ad essi si possono estrarre dei valori equispaziati $\rho_1, \dots, \rho_m$ e $\theta_1, \dots, \theta_n$, rispettivamente con passo $\delta\rho$ e $\delta\theta$. Creiamo un *accumulatore* A, cioè una matrice, di dimensione $m \times n$, che corrisponderà allo spazio dei parametri discretizzato.

- Per ogni punto (x_k, y_k) dell'insieme Ω, sostituiamo le sue coordinate nella (2.4.12), imponendo alla curva il passaggio per quel punto.

- Per ogni valore θ_j, per $j = 1, \dots, n$, calcoliamo, sempre tramite la (2.4.12), il corrispondente $\bar{\rho}_{k,j}$; arrotondiamolo poi al più vicino ρ_i, per qualche $i = 1, \dots, m$. Incrementiamo, quindi, il valore di $A(i, j)$ di un'unità.

In questo modo si forma un accumulatore A che ha per elemento (i, j) il numero di punti che viene approssimato sufficientemente bene dalla (2.4.12) utilizzando i parametri ρ_i e θ_j; i passi con cui sono stati discretizzati i parametri, ovvero $\delta\rho$ e $\delta\theta$, determinano il grado di precisione nell'approssimazione dei punti con la curva descritta dalla (2.4.12). La procedura appena data si generalizza con facilità a vincoli di equazioni differenti e a spazi di parametri di dimensione maggiore.

Tornando al problema del perfezionamento dei contorni, si può utilizzare la trasformata di Hough, relativamente all'equazione delle rette, quindi cercare nell'accumulatore i più alti valori. Per ognuna di queste celle si avrà un insieme di punti che sono approssimati da una particolare retta; prima di collegarli tra loro occorre studiarne meglio la distribuzione, soprattutto per non associare punti molto lontani tra loro, ma non scendiamo nei dettagli.

2.4.2 Thresholding

A causa delle sue intuitive proprietà, semplicità nell'implementazione e velocità di esecuzione, il thresholding dell'immagine è una delle tecniche più usate nella segmentazione. Nonostante alcune semplici applicazioni sono state già trattate, in questa sezione provvediamo ad una più ampia e formale trattazione.

Finora abbiamo discusso la segmentazione individuando dapprima i contorni delle regioni d'interesse, quindi abbiamo cercato di uniformarli ed unirli. Ora utilizzeremo un approccio differente, andando a partizionare le immagini basandoci direttamente sui livelli d'intensità.

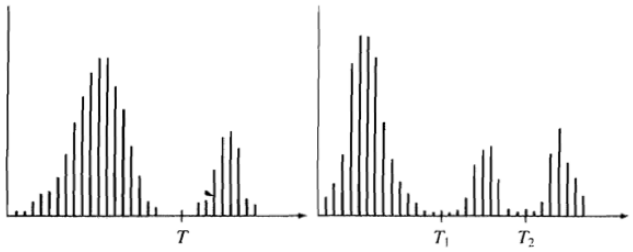


Figura 2.4.3: Istogrammi che è possibile partizionare con uno o due valori di soglia. Immagine tratta da [11].

Supponiamo di avere un'immagine $f(x,y)$ con istogramma mostrato a sinistra nella Figura 2.4.3; poiché l'istogramma è composto da due gruppi dominanti e ben separati (detti *modi*), possiamo intuire che l'immagine è composta da oggetti chiari su uno sfondo scuro. Risulta ovvio che, per estrarre gli oggetti dallo sfondo, basta selezionare il valore di soglia T, mostrato nella figura. Ogni punto dell'immagine per cui $f(x,y) > T$ viene detto *punto oggetto*, mentre tutti gli altri punti costituiscono lo sfondo. In altre parole, l'immagine segmentata $g(x,y)$ si può scrivere come:

$$g(x,y) = \begin{cases} 1 & \text{se } f(x,y) > T \\ 0 & \text{altrimenti} \end{cases} \tag{2.4.13}$$

Quando si può applicare un solo valore di soglia per tutta l'immagine, si parla di *global thresholding*; al contrario, se la segmentazione ha bisogno di valori che cambiano attraverso l'immagine, allora si usa il termine *variable thresholding*. Quando T cambia per ogni punto a seconda delle proprietà del suo intorno, a volte si parla di *local* oppure *regional thresholding*. Se infine T dipende direttamente dal valore delle coordinate spaziali (x,y), allora stiamo eseguendo un *dynamic* oppure *adaptive threshold*. Questa è la terminologia più utilizzata, anche se, non di rado, in letteratura alcuni di questi termini vengono scambiati.

Se ci concentriamo sulla seconda immagine di Figura 2.4.3, vediamo come l'istogramma sia composto da tre modi principali. La segmentazione in questo caso prevede l'utilizzo di due valori di threhsold, T_1 e T_2, suddividendo i punti in: sfondo, primo oggetto e secondo oggetto. La risposta al thresholding si può formulare come segue:

$$g(x,y) = \begin{cases} a & \text{se } f(x,y) > T_2 \\ b & \text{se } T_1 < f(x,y) \leq T_2 \\ c & \text{se } f(x,y) \leq T_1 \end{cases} \tag{2.4.14}$$

dove $a \neq b \neq c$ sono tre diversi livelli di intensità. Problemi con più di due valori di soglia sono difficili da risolvere, spesso impossibili, quindi non verranno trattati affatto in questo lavoro.

Abbiamo appena visto l'applicazione del thresholding su istogrammi davvero semplici, praticamente ideali. Nella realtà intervengono alcuni fattori che rendono la situazione assai più complessa; i principali sono:

- ridotta separazione tra i modi dominanti,
- la rumorosità dell'immagine,
- la grandezza degli oggetti da estrarre rispetto allo sfondo,
- l'uniformità dell'illuminazione.

2.4.2.1 Global thresholding

Come abbiamo accennato, se la distribuzione delle intensità di oggetti e sfondo sull'istogramma è ben distinta, si possono applicare metodi di thresholding globali. Anche se questo è possibile, nella maggior parte delle applicazioni si ha a che fare con molte immagini, spesso non acquisite nelle stesse condizioni. Questo comporta che un valore di soglia scelto per la prima potrebbe non essere desiderabile per le altre; occorre di conseguenza un algoritmo che ci permetta di adattare il parametro T ad ogni immagine.

Una tecnica possibile, anche se molto semplice, potrebbe essere la seguente:

1. stimare un valore da utilizzare come global threshold iniziale T (la media delle intensità potrebbe essere una buona scelta);
2. usando la soglia T, segmentare l'immagine nei due gruppi di pixel $G_+ = \{f(x,y) > T\}$ e $G_- = \{f(x,y) \leq T\}$;
3. calcolare l'intensità media m_+ e m_- dei due gruppi G_+ e G_- rispettivamente;
4. calcolare un nuovo valore di threshold
$$T = \frac{1}{2}(m_+ + m_-)$$
5. ripetere i passi da 2 a 4 fino a che i valori di T tra due successive iterazioni siano sufficientemente vicini.

Come già detto questa tecnica permette di eseguire un threshold globale a varie immagini, purché abbiano istogrammi con una separazione sufficientemente marcata tra oggetti e sfondo.

Per favorire la scelta del valore di soglia in immagini corrotte da molto rumore, si possono, ad esempio, utilizzare i seguenti due approcci. Il primo è eseguire uno smoothing gaussiano, che, eliminando parte del rumore, tenderà a separare l'istogramma in due modi distinti. L'altro metodo prevede il calcolo preliminare dei contorni, anche una semplice stima con il laplaciano può bastare, quindi l'esecuzione dell'algoritmo appena visto sui punti prossimi ai contorni; in genere, infatti, a ridosso di questi si trovano i punti appartenenti alle due regioni in cui si vuole partizionare l'immagine, poiché i margini sono proprio gli elementi che le separano.

Il thresholding può essere visto come un problema di decisione basato su criteri statistici, il cui obiettivo è minimizzare l'errore medio che occorre assegnando i pixel di un'immagine ad una o più classi. Questo è un problema ben noto e se ne conosce un'elegante soluzione, che prende il nome di *Bayes decision rule*. La soluzione è basata solamente su due parametri: la densità di probabilità (PDF) dei livelli di intensità di ciascuna classe e la probabilità che ogni classe occorra in una data applicazione. Sfortunatamente, la stima della densità di probabilità non è banale, per questo motivo il problema è spesso semplificato ricorrendo ad assunzioni sulla forma della PDF. Nonostante tutte le semplificazioni possibili, il processo di implementare soluzioni può essere complesso e non sempre indicato nelle applicazioni pratiche.

Una valida alternativa a quanto appena detto, può essere il cosiddetto *metodo di Otsu* ([29]), dal nome dell'autore dell'algoritmo. Questa tecnica può essere considerata ottimale, nel senso che massimizza la varianza tra le classi, facendo sì che esse siano ben separate le une dalle altre e, come sappiamo, un buon algoritmo di threshold deve essere in grado di separare le classi al meglio. Un'altra buona proprietà del metodo di Otsu è che ha bisogno solamente dell'istogramma dell'immagine, per cui è molto veloce. Qui ci siamo limitati ad un cenno su questa tecnica, ma se ne può trovare un descrizione esauriente in [11]. Il metodo di Otsu si può anche estendere al multiple thresholding, se abbiamo l'obiettivo di separare l'immagine in più di due classi.

2.4.2.2 Variable thresholding

Come discusso in precedenza, fattori come rumore ed illuminazione non uniforme giocano un ruolo molto importante sull'efficacia degli algoritmi di thresholding. Lo smoothing dell'immagine, oppure l'uso preliminare di edge detectors, può essere di grande aiuto in molte situazioni, restano però alcuni casi in cui tali procedure non possono essere praticate. In queste situazioni è necessa-

rio ricorrere al variable thresholding, di cui andiamo ad elencare alcune delle tecniche più comuni.

Image partitioning Uno dei più semplici approcci al variable thresholding è suddividere l'immagine in rettangoli non sovrapposti; questo espediente serve a compensare la disomogeneità dell'illuminazione, infatti i rettangoli sono di dimensioni sufficientemente piccole da poter ritenere costante l'illuminazione sopra di essi. Successivamente ad ogni rettangolo viene applicato un algoritmo di threshold, che di conseguenza si adatterà alle caratteristiche locali. Questo approccio è molto utile quando le zone occupate degli oggetti e dallo sfondo sono paragonabili, altrimenti si potrebbero avere dei rettangoli privi di una delle due componenti; esistono modi per evitare che questo accada, però conviene generalmente cambiare approccio piuttosto che metterli in pratica.

Local thresholding Un approccio più generale del precedente è calcolare un valore di soglia per ogni punto dell'immagine, basandosi su proprietà locali specifiche. Nonostante a primo impatto possa sembrare un processo molto laborioso, bisogna evidenziare che i moderni calcolatori permettono di calcolare funzioni definite sugli intorni dei punti in modo molto rapido.

L'implementazione più semplice del local thresholding è quella che usa la media e la deviazione standard dei pixel nell'intorno di ogni punto; queste due quantità sono infatti ottimi descrittori locali, poiché rappresentano l'intensità media ed il contrasto dell'immagine. Siano, quindi, m_{xy} e σ_{xy} la media e la deviazione standard dei pixel dell'intorno S_{xy} centrato nel punto (x, y). Due tipici valori di soglia locali sono

- $T_{xy} = a\sigma_{xy} + bm_{xy}$, con a, b non negativi, scelti in modo sperimentale;
- $T_{xy} = a\sigma_{xy} + bm_G$, con m_G intensità media globale.

L'immagine segmentata si può calcolare come

$$g(x,y) = \begin{cases} 1 & \text{se } f(x,y) > T_{xy} \\ 0 & \text{altrimenti} \end{cases} \tag{2.4.15}$$

dove con $f(x, y)$ abbiamo indicato i valori d'intensità dell'immagine. L'esperienza mostra che l'utilizzo dell'intensità media globale, generalmente, dà risultati migliori quando si ha a che fare con uno sfondo pressoché costante ed oggetti con intensità molto differente.

Lo stesso approccio si può generalizzare utilizzando un qualunque predicato Q, quindi ricavare l'immagine segmentata tramite la formula:

$$g(x,y) = \begin{cases} 1 & \text{se } Q(\text{parametri locali}) \\ 0 & \text{se } \neg Q(\text{parametri locali}) \end{cases} \tag{2.4.16}$$

Medie mobili Un caso molto particolare di local thresholding è quello che prevede l'utilizzo di *medie mobili* (*moving average*) lungo le linee di scansione dell'immagine. Questa implementazione è particolarmente utile nell'elaborazione di documenti, per cui la velocità di esecuzione è molto importante.

Supponiamo che stiamo scansionando un'immagine linea dopo linea, con un percorso a zigzag, cosicché la scansione della prima linea inizi a sinistra, la seconda a destra, la terza di nuovo a sinistra e così via. Sia z_{k+1} l'intensità di un punto incontrato al passo $k+1$ durante la scansione. La media mobile nel punto corrente è data da

$$m(k+1) = \frac{1}{n}\sum_{i=k+2-n}^{k+1} z_i \;=\; m(k) + \frac{1}{n}(z_{k+1} - z_{n-k}) \tag{2.4.17}$$

dove n denota il numero di punti usati nel calcolo della media e $m(1) = \frac{z_1}{n}$. Facciamo notare che, passando da un punto al successivo, la media mobile si aggiorna aggiungendo il punto corrente e scartando l'$n+1$-esimo punto precedente. Inoltre l'algoritmo non necessita di ripartire ad ogni linea, quindi possiamo inizializzarlo alla prima e lasciargli eseguire la media mobile per tutta l'immagine; a livello computazionale questo aspetto è ottimo. Per ogni punto, appena aggiornatasi la media mobile, eseguiamo anche la segmentazione, utilizzando la formula (2.4.15) con $T_{xy} = bm_{xy}$, dove b è una costante scelta sperimentalmente e m_{xy} è proprio la media mobile appena calcolata.

All'atto pratico il thresholding utilizzando medie mobili dà buoni risultati se gli oggetti di interesse sono piccoli o sottili rispetto alle dimensioni dell'immagine (condizione sicuramente soddisfatta dai documenti).

2.4.3 Region-based thresholding

Inizialmente avevamo definito l'obiettivo della segmentazione come il partizionamento dell'immagine in regioni; abbiamo visto come raggiungere questo scopo sfruttando le discontinuità nei livelli d'intensità, quindi trovando i contorni di tali regioni, e utilizzando tecniche di thresholding basate sulla distribuzione delle intensità dei pixel. In questa parte ci occuperemo di metodi che segmentano l'immagine utilizzando un approccio diretto.

2.4.3.1 Region Growing

Come indica il nome, che può essere reso in italiano come "accrescimento di regioni", quella che stiamo per trattare è una procedura che raggruppa i pixel in regioni basandosi su criteri stabiliti a priori. L'approccio di base è partire con dei punti iniziali (*seed*) ed aggiungere a questi i punti vicini con cui condividono particolari proprietà. La selezione di uno o più punti iniziali in genere è basato sulla natura del problema, ma anche sul tipo d'immagini disponibili.

Molto importante è anche la scelta dei criteri di accrescimento; si utilizzano in genere dei descrittori locali, accompagnati da qualche informazione sulla connessione tra i punti: senza queste indicazioni, infatti, un algoritmo di region growing potrebbe unire punti distanti tra loro, quindi restituire regioni prive di significato in questo contesto. Altro problema è la formulazione dei criteri di terminazione dell'algoritmo; l'idea è che il processo si dovrebbe interrompere se non ci sono più punti che soddisfano i criteri per l'inclusione nella regione corrente. Utilizzare criteri come intensità, textures o colori è frequente, ma sono caratteristiche locali; nei casi in cui lo si ritiene necessario si possono, invece, utilizzare dei criteri ben più complessi, che tengono conto della grandezza della regione, della similarità del punto candidato con quelli già presenti, oppure, ancora, della forma della regione stessa. Tutti questi criteri si basano comunque sull'assunzione che siano disponibili informazioni sull'output atteso.

Sia $f(x, y)$ l'immagine di partenza, $S(x, y)$ una matrice di punti iniziali della stessa dimensione di f (in corrispondenza dei punti iniziali $S(x, y) = 1$, mentre $S(x, y) = 0$ altrove). Sia Q un predicato che indichi la similarità tra due punti, da utilizzare come criterio per accrescere le regioni; un possibile e semplice esempio potrebbe essere

$$Q = \begin{cases} \text{TRUE} & \text{se la differenza assoluta delle intensità tra il } \textit{seed} \\ & \text{e il pixel in } (x, y) \text{ è } \leq T \\ \text{FALSE} & \text{altrimenti} \end{cases} \tag{2.4.18}$$

Un algoritmo semplice per il region growing, considerando vicini i punti che si trovano nell'intorno di Moore di raggio unitario (8-*connectivity*), è quello esposto di seguito.

1. Trovare tutte le componenti connesse di $S(x, y)$, quindi erodere ogni componente sino a ottenere un singolo punto; assegnare a tutti questi punti isolati valore unitario.

2. Formare una matrice f_Q tale che $f_Q(x, y) = 1$ se l'immagine iniziale soddisfa Q in (x, y), $f_Q(x, y) = 0$ altrimenti.

3. Sia g l'immagine formata aggiungendo ad ogni *seed* in S tutti i punti (x, y) tali che $f_Q(x, y) = 1$ e che sono connessi al *seed*. I punti associati a ciascun *seed* devono avere in g valori differenti, così da ottenere la segmentazione dell'immagine iniziale.

2.4.3.2 Region splitting and merging

La procedura appena descritta accresce regioni inizialmente composte dai soli *seed*; un'alternativa è suddividere l'immagine in regioni disgiunte arbitrarie, quindi unirle o suddividerle ancora fino a soddisfare i criteri richiesti dalla segmentazione: questo processo prende il nome di *region splitting and merging*. Esponiamo di seguito la linea generale per un algoritmo di questo tipo.

Sia R la regione corrispondente all'intera immagine e sia Q un predicato. Se $Q(R) = FALSE$ allora dividiamo l'immagine in quadranti R_1, R_2, R_3, R_4, quindi testiamo Q su di essi; quando Q non è soddisfatto operiamo un'ulteriore suddivisione della regione. Ci si fermerà quando tutte le sottoregioni soddisfano il predicato, oppure quando abbiamo raggiunto la minima dimensione consentita per le regioni che non soddisfano il predicato. Questo modo di procedere ha una comoda rappresentazione tramite un *quadtree*, ovvero un albero in cui ogni nodo ha quattro discendenti. Alla fine del processo otteniamo una partizione finale composta da regioni adiacenti con identiche proprietà; questo problema può essere risolto associando tutte le regioni adiacenti R_j, R_k tali che $Q(R_j \cup R_k) = TRUE$.

Utilizzando, quindi, sia la suddivisione che la successiva associazione delle regioni simili, otteniamo la segmentazione dell'immagine; generalmente le regioni così ottenute si utilizzano come maschere per una successiva elaborazione.

2.4.4 Segmentazione tramite watersheds

Finora abbiamo discusso metodi per la segmentazione basati su tre concetti principali, ognuno con propri vantaggi e svantaggi: edge-detector, thresholding e region growing. In questa sezione vedremo un approccio basato sui cosiddetti *morphological watersheds* (in italiano *spartiacque morfologici*), che consente una segmentazione più stabile, contorni connessi ed è semplice inserire vincoli basati sulla conoscenza a priori della segmentazione.

Il concetto di watershed è effettivamente legato ad un'interpretazione dell'algoritmo che sfrutta bacini d'acqua e dighe. Innanzitutto occorre pensare al grafico della funzione $f(x, y)$ che descrive l'immagine, quindi ad una rappresentazione in tre dimensioni, due per il piano dell'immagine e la terza per l'intensità. Così facendo potremo distinguere tre differenti tipi di punti:

- regioni di minimo locale (la zona più profonda di un lago);

- bacini d'attrazione, costituiti da punti tali che, se vi facessimo cadere una goccia d'acqua, essa scivolerebbe verso un minimo locale (pareti e letto del lago);
- watersheds, i cui punti sono tali che, se vi facessimo cadere una goccia d'acqua, essa andrebbe verso più di un minimo locale (sommità delle pareti che dividono un lago dall'altro).

L'obiettivo di un algoritmo basato su questi concetti è trovare i punti dei watersheds. L'idea da seguire è semplice: supponiamo che dai minimi locali zampilli acqua e che, quindi, questa salga piano piano fino a riempire i bacini; ogni volta che l'acqua raggiunge un livello tale da esondare nel lago adiacente, si erge una diga per separarli, quindi si continua a far salire l'acqua. Alla fine di questo procedimento, guardando la scena dall'alto, si avranno laghi pieni d'acqua separati da sottili dighe: sostanzialmente quello che vorremmo ottenere con la segmentazione.

Vediamo quindi la formulazione matematica dell'algoritmo. Siano $M_1, M_2, \dots, M_R$ insiemi costituiti dalle coordinate delle regioni di minimo locale di un'immagine $g(x, y)$. Generalmente questo metodo si applica all'immagine gradiente, poiché in essa i contorni sono già sopraelevati rispetto alle regioni che racchiudono. Sia $C(M_i)$ l'insieme che contiene le coordinate dei punti nel bacino d'attrazione relativo alla regione di minimo M_i; infine sia $T[n]$ l'insieme delle coordinate dei punti che si trovano sotto il livello d'intensità n:

$$T[n] = \{(s,t) | g(s,t) < n\} \tag{2.4.19}$$

dove n è un intero a valori tra $\min g+1$ e $\max g+1$ (con $\min g$ e $\max g$ intendiamo i valori d'intensità minimo e massimo dell'immagine). Sia, poi, $C_n(M_i)$ l'insieme dei punti di $C(M_i)$ che si trovano sotto il livello d'intensità n, ovvero

$$C_n(M_i) = C(M_i) \cap T[n] \tag{2.4.20}$$

Inoltre definiamo $C[n]$ come l'insieme dei punti di tutti i bacini sottostanti il livello d'intensità n:

$$C[n] = \bigcup_{i=1}^{R} C_n(M_i) \tag{2.4.21}$$

Dall'ultima relazione segue che

$$C[\max g + 1] = \bigcup_{i=1}^{R} C(M_i) \tag{2.4.22}$$

Si può mostrare che, man mano che n aumenta, gli insiemi $C_n(M_i)$ e $T[n]$ non decrescono ed i loro punti non vengono mai scambiati. Segue da questo che

$$C[n-1] \subset C[n] \tag{2.4.23}$$

e utilizzando la (2.4.20)

$$C[n-1] \subset T[n] \tag{2.4.24}$$

Si può quindi provare che ogni componente connessa di $C[n-1]$ è contenuta in esattamente una componente connessa di $T[n]$.

L'algoritmo per trovare i watersheds è inizializzato con $C[\min g+1] = T[\min g+1]$, quindi prosegue ricorsivamente, calcolando $C[n]$ da $C[n-1]$. Spieghiamo di seguito come fare. Denotando con $Q[n]$ l'insieme delle componenti connesse di $T[n]$, per ogni componente $q \in Q[n]$, ci sono tre possibilità:

1. $q \cap C[n-1]$ è vuoto;
2. $q \cap C[n-1]$ contiene una sola componente connessa di $C[n-1]$;
3. $q \cap C[n-1]$ contiene più di una componente connessa di $C[n-1]$.

A seconda della situazione in cui ci troviamo il procedimento per calcolare $C[n]$ da $C[n-1]$ cambia. La condizione 1 ci dice che abbiamo incontrato un nuovo minimo locale, quindi basta calcolare $C[n] = q \cup C[n-1]$; se ci troviamo nel caso 2, significa che q è la nuova configurazione di una delle componenti connesse già presenti in $C[n-1]$, quindi per ottenere $C[n]$, come prima, facciamo $C[n] = q \cup C[n-1]$. La situazione descritta dal punto 3, invece, è quella in cui l'acqua ha raggiunto la sommità di due o più bacini, che rischiano di fondersi; occorre costruire una sorta di diga, larga un pixel, per evitare che ciò accada. In termini più tecnici, occorre dilatare le due o più regioni in $q \cap C[n-1]$ fino a che i bordi non si tocchino; a partire dai punti in cui le componenti si sovrappongono si può costruire una linea di intensità molto elevata ($\max g + 1$) che costituirà la diga di cui abbiamo parlato.

L'algoritmo, così come l'abbiamo descritto, potrebbe portare ad un'eccessiva segmentazione, soprattutto in presenta di rumore o irregolarità del gradiente. Il problema generalmente è dovuto al fatto che vengono considerate minimi locali troppe regioni; per rimediare si possono imporre delle condizioni iniziali, i *marker*. Questi sono regioni connesse, selezionate automaticamente utilizzando specifici criteri, che saranno la formulazione matematica della conoscenza che abbiamo a priori sulla segmentazione dell'immagine; l'algoritmo poi viene eseguito utilizzando solamente i marker come minimi locali, senza possibilità di aggiungerne altri.

2.5 Metodi di analisi e riconoscimento

Gli approcci al *pattern recognition*, ovvero il riconoscimento di forme o configurazioni, si suddividono in: teorico-decisionale e strutturale. Il primo ha a che fare con descrittori quantitativi, come lunghezza, area e texture e vedremo di seguito alcuni modelli che ne fanno uso. L'altro si occupa di forme meglio rappresentate da descrittori qualitativi, come ad esempio stringhe o *shape numbers* (numeri associati alla forma dei contorni); in questo lavoro di tesi non vedremo nessun metodo strutturale, per non divergere troppo dall'obiettivo principale.

Un *pattern* si può definire come un *sistema di descrittori*, chiamati anche *caratteristiche* (*features*) in questo contesto. Una *classe di patterns* è, invece, una famiglia di patterns che condividono qualche proprietà e sono spesso indicati con $\omega_1, \omega_2, \dots, \omega_W$, dove W è il numero totale di classi definite per la particolare situazione in esame. Lo scopo delle tecniche di pattern recognition è assegnare ogni pattern alla rispettiva classe, in modo completamente automatico, oppure con un lieve intervento umano.

2.5.1 Riconoscimento basato su metodi teorico-decisionali

Approcci teorico-decisionali per il riconoscimento sono basati sull'uso di *funzioni di decisione*, ovvero mappe il cui valore permetta di stabilire l'appartenenza dei pattern alle rispettive classi.

Sia $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ un pattern n-dimensionale, ovvero un insieme di n caratteristiche, rappresentate ognuna tramite una componente scalare di $\mathbf{x}$. Se abbiamo W classi di pattern cui assegnare nuove configurazioni, il problema principale dei metodi teorico-decisionali è trovare W funzioni di decisione $d_1(\mathbf{x}), d_2(\mathbf{x}), \dots, d_W(\mathbf{x})$ con la proprietà che, se il pattern $\mathbf{x}$ è nella classe ω_i, allora

$$d_i(\mathbf{x}) > d_j(\mathbf{x}) \qquad j = 1, 2, \dots, W,\ j \neq i \tag{2.5.1}$$

A partire dalle funzioni di decisione si possono anche calcolare le *frontiere di decisione*, ovvero l'insieme dei punti che separano due classi; in formule si può scrivere la frontiera di decisione tra ω_i e ω_j come

$$d_{ij}(\mathbf{x}) = d_i(\mathbf{x}) - d_j(\mathbf{x}) = 0 \tag{2.5.2}$$

Molte sono le possibili soluzioni per trovare funzioni di questo tipo: nel seguito ne vedremo alcune. Il processo con cui si stimano i parametri delle funzioni di decisione, a partire da patterns di cui si conosce la classificazione, prende il nome di *learning* oppure *training* del classificatore.

2.5.1.1 Matching

Le tecniche di riconoscimento basate sul matching assegnano ad ogni classe un pattern di riferimento, ovvero un vettore di caratteristiche da confrontare con quelle calcolate sull'immagine da analizzare. Un pattern viene associato alla classe il cui rappresentante è il più vicino possibile ad esso, secondo una qualche metrica specificata.

La metrica più semplice che ci può venire in mente è quella euclidea. Per utilizzarla in questo contesto, generalmente si utilizza il cosiddetto *minimun distance classifier*, nome che ribadisce la classificazione eseguita minimizzando la distanza. Calcoliamo la media dei patterns di ogni classe, per cui avremo

$$\mathbf{m}_j = \frac{1}{N_j} \sum_{\mathbf{x} \in \omega_j} \mathbf{x} \qquad j = 1, 2, \dots, W \tag{2.5.3}$$

dove $\mathbf{m}_j$ è il vettore medio, N_j il numero di patterns presenti nella classe ω_j, W il numero di classi. Supponiamo ora di aver calcolato un pattern $\mathbf{y}$ da un'immagine e di voler capire a quale classe appartenga; utilizziamo la metrica euclidea per trovare la classe più rappresentativa, per cui calcoliamo

$$D_j(\mathbf{y}) = \|\mathbf{y} - \mathbf{m}_j\| \qquad j = 1, 2, \dots, W \tag{2.5.4}$$

ed assegniamo $\mathbf{y}$ alla classe i tale che $D_i(\mathbf{y}) \leq D_j(\mathbf{y}) \quad \forall j = 1, 2, \dots, W$. Per poter dire di aver trovato delle funzioni di decisione, come quelle descritte nella (2.5.1), possiamo dimostrare che il procedimento appena illustrato equivale a calcolare le funzioni

$$d_j(\mathbf{y}) = \mathbf{y}^T \mathbf{m}_j - \frac{1}{2} \mathbf{m}_j^T \mathbf{m}_j \qquad j = 1, 2, \dots, W \tag{2.5.5}$$

ed assegnare $\mathbf{y}$ alla classe con il più alto valore. Dalla (2.5.5) si può anche scrivere l'equazione delle frontiere di decisione, ovvero

$$d_{ij}(\mathbf{y}) = \mathbf{x}^T(\mathbf{m}_i - \mathbf{m}_j) - \frac{1}{2}(\mathbf{m}_i - \mathbf{m}_j)^T(\mathbf{m}_i + \mathbf{m}_j) = 0 \tag{2.5.6}$$

Nelle applicazioni pratiche, l'indicazione generale è di utilizzare questo tipo di classificatore solamente in caso di distanze tra le medie interne alle classi sufficientemente elevate rispetto alla loro varianza.

Un'altra misura che si può utilizzare è il *coefficiente di correlazione normalizzato*, che permette di calcolare il grado di similarità tra due vettori, eventualmente due pattern. Supponiamo di avere due vettori $\mathbf{w}$ e $\mathbf{v}$; il coefficiente di correlazione normalizzato si calcola come:

$$\gamma(\mathbf{w}, \mathbf{v}) = \frac{\sum_i (w_i - \bar{w}) \sum_j (v_j - \bar{v})}{\left[\sum_i (w_i - \bar{w})^2 \sum_j (v_j - \bar{v})^2 \right]^{\frac{1}{2}}} \tag{2.5.7}$$

Nel contesto dell'object recognition, possiamo utilizzare come pattern di riferimento una porzione d'immagine contenente una forma da riconoscere, chiamiamola $w(s,t)$; per ogni immagine $f(x,y)$ che analizziamo possiamo calcolare il coefficiente di correlazione normalizzato tra w ed un intorno del punto (x,y) di dimensione pari a w, al variare del punto (x,y), adattando la (2.5.7) come segue:

$$\gamma(x,y) = \frac{\sum_s \sum_t [w(s,t) - \bar{w}] \sum_s \sum_t \left[f(x+s,y+t) - \bar{f}(x+s,y+t) \right]}{\left\{ \sum_s \sum_t [w(s,t) - \bar{w}]^2 \sum_s \sum_t \left[f(x+s,y+t) - \bar{f}(x+s,y+t) \right]^2 \right\}^{\frac{1}{2}}} \quad (2.5.8)$$

dove gli indici s e t variano in modo da scorrere tutti i punti di w. Il pattern w viene spesso chiamato *template*, mentre la correlazione prende il nome di *template matching*.

Il coefficiente di correlazione, per come lo abbiamo definito, è normalizzato rispetto ai cambiamenti di intensità; questo permette di riconoscere oggetti in un'immagine acquisita in condizioni d'illuminazione differenti dal template. In alcuni casi potrebbe essere necessario anche normalizzare l'immagine per cambiamenti di dimensioni e per rotazioni, ma il problema è molto più complesso, a meno che non si dispongano di informazioni riguardanti le condizioni di acquisizione, per questo motivo non ci addentriamo nella sua discussione.

2.5.1.2 Classificatori ottimi basati su statistiche

Nel seguito svilupperemo un approccio probabilistico al problema del riconoscimento, finalizzato a minimizzare la probabilità di commettere errori nella classificazione. Analogamente a ciò che avviene se misuriamo ed interpretiamo fenomeni fisici, nel campo del pattern recognition l'intervento della probabilità permette di gestire la casualità con cui si presentano i pattern provenienti dalle singole classi. Gli algoritmi che stabiliscono l'appartenenza di un pattern ad una classe vengono chiamati *classificatori*.

Classificatori di Bayes La probabilità che un particolare pattern $\mathbf{x}$ provenga dalla classe ω_j viene indicata con $p(\omega_j|\mathbf{x})$. Se il classificatore stabilisce che $\mathbf{x}$ appartiene alla classe ω_j, quando in realtà viene da ω_i, darà origine ad un errore che indicheremo con L_{ij}; il pattern $\mathbf{x}$ potrebbe provenire da una qualunque delle W classi presenti, sembra quindi ragionevole calcolare l'errore medio che il classificatore commette attribuendo ad $\mathbf{x}$ la classe ω_j:

$$r_j(\mathbf{x}) = \sum_{k=1}^{W} L_{kj} p(\omega_k|\mathbf{x}) \quad (2.5.9)$$

Sfruttando un risultato elementare di probabilità, ovvero

$$p(A|B) = \frac{p(A)}{p(B)} p(B|A) \tag{2.5.10}$$

dalla (2.5.9) possiamo scrivere

$$r_j(\mathbf{x}) = \frac{1}{p(\mathbf{x})} \sum_{k=1}^{W} L_{kj} p(\mathbf{x}|\omega_k) p(\omega_k) \tag{2.5.11}$$

dove $p(\mathbf{x}|\omega_k)$ è la densità di probabilità dei patterns nella classe ω_k calcolata in $\mathbf{x}$, mentre $p(\omega_k)$ è la probabilità che occorra la classe ω_k. Poiché la (2.5.10) è conosciuta come Teorema di Bayes, il classificatore creato si chiamerà *classficatore di Bayes*.

Se, per ogni nuovo pattern $\mathbf{x}$, calcoliamo tutte le funzioni $r_1(\mathbf{x}), r_2(\mathbf{x}), \dots, r_W(\mathbf{x})$ ed assegniamo ad $\mathbf{x}$ la classe con l'errore più piccolo, l'errore medio totale sarà il minimo possibile. Affinché si scelga per $\mathbf{x}$ la classe ω_j, occorre verificare quanto segue (il termine $p(\mathbf{x})$ è comune a tutte le funzioni, possiamo non considerarlo)

$$\sum_{k=1}^{W} L_{kj} p(\mathbf{x}|\omega_k) p(\omega_k) < \sum_{q=1}^{W} L_{qi} p(\mathbf{x}|\omega_q) p(\omega_q) \qquad \forall i \neq j \tag{2.5.12}$$

Possiamo ragionevolmente stabilire che:

- l'errore commesso, se la classe scelta è quella giusta, sia nullo, da cui $Ljj = 0$;
- l'errore generato, nel caso in cui la classe selezionata non fosse quella corretta, sia 1, per cui si ha $Lji = 1 \ \forall i \neq j$.

Le funzioni in (2.5.11) si possono riscrivere come

$$\begin{aligned} r_j(\mathbf{x}) &= \sum_{k=1}^{W} L_{kj} p(\mathbf{x}|\omega_k) p(\omega_k) \\ &= \sum_{k \neq j} p(\mathbf{x}|\omega_k) p(\omega_k) \\ &= \sum_{k=1}^{W} p(\mathbf{x}|\omega_k) p(\omega_k) - p(\mathbf{x}|\omega_j) p(\omega_j) \\ &= p(\mathbf{x}) - p(\mathbf{x}|\omega_j) p(\omega_j) \end{aligned} \tag{2.5.13}$$

Il classificatore di Bayes assegna un pattern $\mathbf{x}$ alla classe ω_j se

$$p(\mathbf{x}) - p(\mathbf{x}|\omega_j)p(\omega_j) < p(\mathbf{x}) - p(\mathbf{x}|\omega_i)p(\omega_i) \qquad \forall i \neq j \tag{2.5.14}$$

oppure, con qualche semplificazione

$$p(\mathbf{x}|\omega_i)p(\omega_i) > p(\mathbf{x}|\omega_j)p(\omega_j) \qquad \forall i \neq j \tag{2.5.15}$$

Dall'ultima relazione vediamo che è possibile rientrare nell'approccio generale dei metodi teorico-decisionali, definendo le funzioni di decisione

$$d_j(\mathbf{x}) = p(\mathbf{x}|\omega_j)p(\omega_j) \qquad j = 1, \dots, W \tag{2.5.16}$$

Tramite queste funzioni il classificatore deve scegliere, per ogni $\mathbf{x}$, la classe con valore più alto.

Le funzioni di decisione appena definite sono ottime nel senso che minimizzano l'errore medio commesso nella classificazione. Volendo mantenere questa ottimalità, la densità di probabilità dei patterns all'interno di ogni classe, così come le probabilità di occorrenza di ogni classe, deve essere nota. Generalmente, dalla conoscenza che si ha sul problema, si possono inferire le probabilità, mentre rimane un problema stimare la densità di probabilità. Se, infatti, i patterns sono n-dimensionali, stimare il valore della densità di probabilità richiederebbe un approccio tramite variabili aleatorie multivariate, difficili da utilizzare nella pratica. Per queste ragioni i classificatori di Bayes si basano spesso sull'assunzione che le densità di probabilità dei patterns siano particolari funzioni note, per le quali la stima dei parametri sia semplice. In questo modo però ci si allontana dall'ottimalità agognata, a meno che la realtà non sia descritta esattamente dalla funzione scelta: cosa assai improbabile in pratica. Una delle scelte più comuni è la densità di probabilità gaussiana, di cui parleremo nel prossimo paragrafo.

Classificatori di Bayes gaussiani Per iniziare consideriamo il problema unidimensionale, con due sole pattern classes, descritte da densità di probabilità gaussiane con medie m_1, m_2 e deviazioni standard σ_1, σ_2, rispettivamente. La funzione di decisione di questo classificatore bayesiano gaussiano è

$$d_j(x) = p(x|\omega_j)p(\omega_j) = \frac{1}{\sqrt{2\pi}\sigma_j} e^{\frac{(x-m_j)^2}{2\sigma_j^2}} p(\omega_j) \qquad j = 1, 2 \tag{2.5.17}$$

La frontiera di decisione in questo caso è data da $d_1(x) = d_2(x)$, la cui soluzione è x_0 mostrato in Figura 2.5.1. I pattern x che si trovano a destra di x_0 verranno classificati in ω_2, mentre quelli alla sua sinistra saranno assegnati ad ω_1.

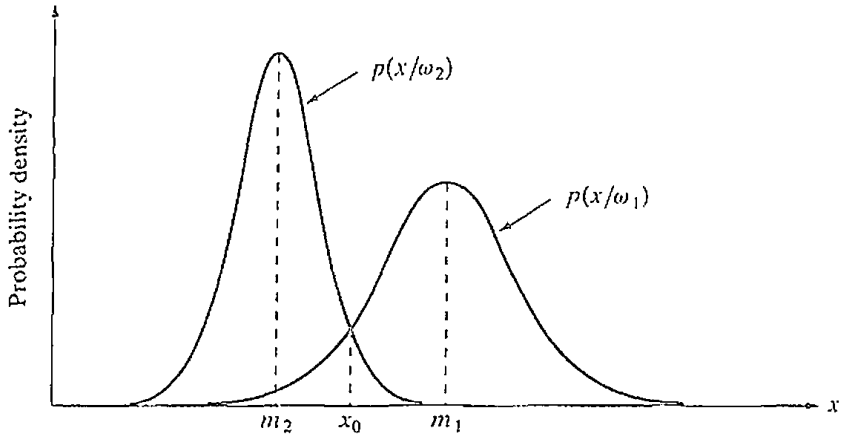


Figura 2.5.1: Densità di probabilità di due pattern classes unidimensionali (immagine tratta da [11]).

Nel caso n-dimensionale la densità gaussiana dei vettori della j-esima classe assume la forma

$$p(\mathbf{x}|\omega_j) = \frac{1}{(2\pi)^{\frac{n}{2}}|\mathbf{C}_j|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m}_j)^T \mathbf{C}_j^{-1}(\mathbf{x}-\mathbf{m}_j)} \tag{2.5.18}$$

dove la media $\mathbf{m}_j$ dei patterns della classe ω_j è definita come

$$\mathbf{m}_j = E_j[\mathbf{x}] \tag{2.5.19}$$

e $|\mathbf{C}_j|$ è il determinante della matrice di covarianza $\mathbf{C}_j$, definita da

$$\mathbf{C}_j = E_j\left[(\mathbf{x} - \mathbf{m}_j)(\mathbf{x} - \mathbf{m}_j)^T\right] \tag{2.5.20}$$

Approssimando il valore atteso E_j con la media aritmetica si ottiene

$$\mathbf{m}_j = \frac{1}{N_j} \sum_{\mathbf{x} \in \omega_j} \mathbf{x} \tag{2.5.21}$$

e

$$\mathbf{C}_j = \frac{1}{N_j} \sum_{\mathbf{x} \in \omega_j} \mathbf{x}\mathbf{x}^T - \mathbf{m}_j \mathbf{m}_j^T \tag{2.5.22}$$

dove N_j è il numero di elementi della classe ω_j e le somme sono fatte su di essi.

La matrice di covarianza è simmetrica e semidefinita positiva; il j-esimo elemento diagonale c_{jj} è la varianza della j-esima componente dei patterns, mentre l'elemento c_{jk} è la covarianza delle componenti x_j ed x_k dei vettori di una classe.

Poiché la densità di probabilità gaussiana comprende un esponenziale, ridefiniamo la funzione di decisione componendole un logaritmo (il suo andamento monotono non modifica le decisioni prese dal classificatore). Sia

$$d_j(\mathbf{x}) = \ln\left[p(\mathbf{x}|\omega_j)p(\omega_j)\right] = \ln p(\mathbf{x}|\omega_j) + \ln p(\omega_j) \tag{2.5.23}$$

da cui

$$d_j(\mathbf{x}) = \ln p(\omega_j) - \frac{n}{2}\ln 2\pi - \frac{1}{2}\ln|\mathbf{C}_j| - \frac{1}{2}\left[(\mathbf{x}-\mathbf{m}_j)^T\mathbf{C}_j^{-1}(\mathbf{x}-\mathbf{m}_j)\right] \tag{2.5.24}$$

ed eliminando il termine $\frac{n}{2}\ln 2\pi$, che è costante per tutte le classi e quindi ininfluente a livello decisionale, otteniamo

$$d_j(\mathbf{x}) = \ln p(\omega_j) - \frac{1}{2}\ln|\mathbf{C}_j| - \frac{1}{2}\left[(\mathbf{x}-\mathbf{m}_j)^T\mathbf{C}_j^{-1}(\mathbf{x}-\mathbf{m}_j)\right] \tag{2.5.25}$$

Da un punto di vista geometrico, il classificatore separa le classi mediante superfici $(n-1)$-dimensionali; nel caso in cui la popolazione di patterns è realmente distribuita in accordo a gaussiane, allora questa è la migliore superficie che potrebbe separare le classi tra loro.

In caso di particolari simmetrie nella matrice di covarianza, la (2.5.25) si può semplificare; se le matrici di covarianza sono tutte uguali tra le varie classi, allora otteniamo

$$d_j(\mathbf{x}) = \ln p(\omega_j) + \mathbf{x}^T\mathbf{C}^{-1}\mathbf{m}_j - \frac{1}{2}\mathbf{m}_j^T\mathbf{C}^{-1}\mathbf{m}_j \qquad j = 1,\dots,W \tag{2.5.26}$$

Se inoltre $\mathbf{C} = \mathbf{I}$, allora la (2.5.25) diviene

$$d_j(\mathbf{x}) = \mathbf{x}^T\mathbf{m}_j - \frac{1}{2}\mathbf{m}_j^T\mathbf{m}_j \qquad j = 1,\dots,W \tag{2.5.27}$$

La (2.5.27) e la (2.5.5) hanno la stessa forma; questo ci permette di trarre le seguenti conclusioni: il minimum distance classifier è ottimo come classificatore bayesiano se:

- le classi di pattern sono distribuiti come gaussiane,
- tutte le matrici di covarianza sono tutte matrici identità,
- ogni classe ha la stessa probabilità di occorrenza delle altre.

2.5.2 Reti neurali

Nella sezione precedente, abbiamo discusso approcci in cui la fase di learning del classificatore era molto semplice. I parametri delle funzioni di decisione venivano stimati a partire da alcuni patterns di riferimento, il tutto dopo aver fatto assunzioni sulla distribuzione dei patterns per semplificare i calcoli; la struttura poi rimaneva fissa per tutta l'esecuzione dell'algoritmo. Nelle applicazioni pratiche farebbero più comodo dei modelli in cui sia l'algoritmo stesso a gestire le funzioni di decisione; a quel punto non sarebbe più necessario fare assunzioni, spesso limitanti, sulla distribuzione reale dei patterns.

L'esposizione degli algoritmi in questa sezione sarà graduale, partendo dal modello più semplice di rete neurale, sino ad estendere i risultati all'approccio più generale ed efficiente. L'obiettivo sarà definire i *neuroni*, strutture elementari per il calcolo non lineare, e la loro organizzazione reticolare, chiamata appunto *neural network*, per la somiglianza al modo con cui i neuroni del cervello umano sono connessi e cooperano tra loro.

2.5.2.1 Il perceptron per due classi di patterns

Iniziamo con l'introdurre il *perceptron*, uno dei più semplici modelli che riescono ad adattarsi ai patterns di training senza dover inferire nulla, a priori, sulla loro distribuzione spaziale.

La struttura portante Supponiamo di avere solamente due classi di patterns linearmente separabili, ovvero separabili da una retta. La risposta del perceptron ad un pattern

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \tag{2.5.28}$$

è basata sulla media ponderata delle sue componenti, ovvero

$$d(\mathbf{x}) = \sum_{i=0}^{n} w_i x_i + w_{n+1} \tag{2.5.29}$$

dove w_i, per $i = 0, 1, \ldots, n+1$, sono i pesi associati alle componenti. Questa somma pesata viene poi sottoposta alla cosiddetta *funzione di attivazione*, dal comportamento simile ad un threshold, che produce in output:

- $+1$, se $d(\mathbf{x}) > 0$,

- -1, se $d(\mathbf{x}) < 0$.

Nel caso in cui $d(\mathbf{x}) = 0$, il pattern giace sull'iperpiano di separazione tra le due classi, per cui siamo in una situazione di indecisione. È stato dimostrato che il perceptron appena descritto, se addestrato su patterns classes linearmente separabili, converge sempre ad una soluzione in un numero finito di passi. Il funzionamento del perceptron è rappresentato dallo schema riportato in Figura 2.5.2.

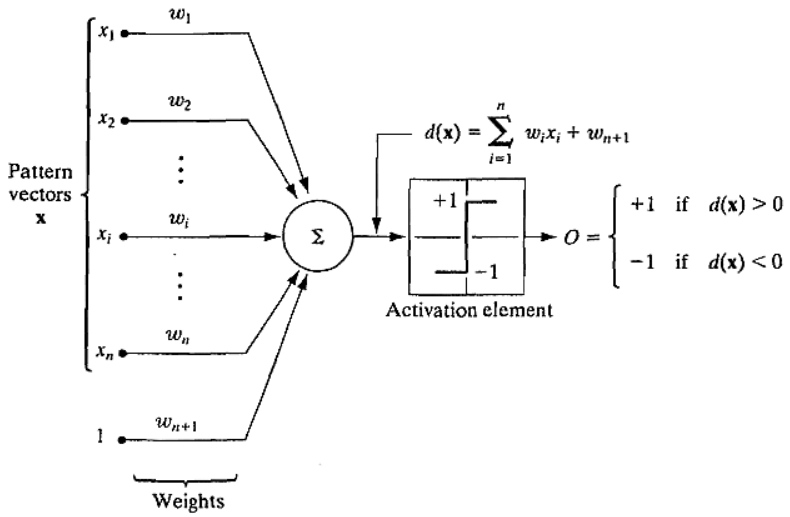


Figura 2.5.2: Rappresentazione schematica di un perceptron. Immagine tratta da [11].

Algoritmo di training per classi linearmente separabili Vediamo di seguito un algoritmo iterativo abbastanza semplice per ottenere i pesi $\mathbf{w}$ per due classi di patterns linearmente separabili.

Notiamo che la funzione di decisione data si può anche riscrivere in modo più compatto come

$$d(\mathbf{y}) = \sum_{i=0}^{n+1} w_i y_i = \mathbf{w}^T \mathbf{y} \tag{2.5.30}$$

dove $\mathbf{w}$ è un vettore $n+1$-dimensionale detto *vettore peso*, mentre $\mathbf{y} = [\mathbf{x}, 1]^T$ è detto *pattern aumentato*. Supponiamo di avere due classi di patterns denotate con ω_1 e ω_2. Sia $\mathbf{w}(1)$ un vettore peso iniziale arbitrario; vogliamo modificarlo iterativamente fino ad ottenere il vettore ideale, ovvero che permetta la corretta classificazione di tutti i pattern di riferimento. Sia analogamente $\mathbf{y}(k)$ il vettore

aumentato ottenuto a partire dal k-esimo pattern e $\mathbf{w}(k)$ il vettore peso adattato a tutti i $k-1$ pattern precedenti. Se $\mathbf{y}(k) \in \omega_1$, al passo k si possono presentare due situazioni:

- $\mathbf{w}^T(k)\mathbf{y}(k) > 0$, per cui il classificatore assegna a $\mathbf{y}(k)$ la classe ω_1, di conseguenza procediamo lasciando il vettore peso invariato $\mathbf{w}(k+1) = \mathbf{w}(k)$;
- $\mathbf{w}^T(k)\mathbf{y}(k) \leq 0$, ovvero il classificatore afferma $\mathbf{y}(k) \in \omega_2$ o indecisione; lo dobbiamo correggere aggiornando il vettore peso in

$$\mathbf{w}(k+1) = \mathbf{w}(k) + c\mathbf{y}(k) \tag{2.5.31}$$

  dove c è una costante correttiva.

Situazione pressoché analoga si presenta all'iterazione k nel caso $\mathbf{y}(k) \in \omega_2$:

- $\mathbf{w}^T(k)\mathbf{y}(k) < 0$, allora il classificatore è corretto;
- $\mathbf{w}^T(k)\mathbf{y}(k) \geq 0$, per cui bisogna correggere tramite

$$\mathbf{w}(k+1) = \mathbf{w}(k) - c\mathbf{y}(k) \tag{2.5.32}$$

Teniamo a ribadire che questo algoritmo di training dà ottimi risultati solamente per classi linearmente separabili, caso in cui è garantita la convergenza in un numero finito di passi; nella pratica, avere classi linearmente indipendenti è utopistico, quindi dobbiamo provvedere a correggere l'algoritmo per soddisfare alle richieste applicative.

Algoritmo di training per classi non linearmente separabili Uno dei primi modelli sviluppati per addestrare il perceptron, su classi non separabili, fu quello conosciuto con il nome di *Widrow-Hoff delta rule*, oppure *least-mean-square delta rule*, di solito abbreviato in *delta rule*. Questo algoritmo tende a minimizzare, ad ogni passo, l'errore tra la risposta ricevuta e quella desiderata.

Consideriamo la funzione obiettivo

$$J(\mathbf{w}) = \frac{1}{2}(r - \mathbf{w}^T\mathbf{y})^2 \tag{2.5.33}$$

dove r è la risposta desiderata, ovvero $r = +1$ se $\mathbf{y} \in \omega_1$ e $r = -1$ se $\mathbf{y} \in \omega_2$. L'obiettivo è adattare $\mathbf{w}$ iterativamente verso la direzione negativa del gradiente, cercando, cioè, di minimizzare la funzione obiettivo verso l'estremo $r = \mathbf{w}^T\mathbf{y}$.

Sia $\mathbf{w}(k)$ il vettore peso all'iterazione k-esima; aggiorniamolo in $\mathbf{w}(k+1)$ con la formula seguente

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha \left[\frac{\partial J(\mathbf{w}(k))}{\partial \mathbf{w}(k)} \right] \tag{2.5.34}$$

dove il parametro $\alpha > 0$ determina l'ampiezza della correzione. Dalla (2.5.33) abbiamo

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = -(r - \mathbf{w}^T \mathbf{y})\mathbf{y} \tag{2.5.35}$$

Andiamo a sostituire la (2.5.35) nella (2.5.34) per avere la formula completa per l'aggiornamento del vettore peso:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha \left[r(k) - \mathbf{w}^T(k)\mathbf{y}(k) \right] \mathbf{y}(k) \tag{2.5.36}$$

Per semplificare la notazione, definiamo $\Delta\mathbf{w} = \mathbf{w}(k+1) - \mathbf{w}(k)$ e

$$e(k) = r(k) - \mathbf{w}^T(k)\mathbf{y}(k) \tag{2.5.37}$$

da cui

$$\Delta\mathbf{w} = \alpha e(k)\mathbf{y}(k) \tag{2.5.38}$$

La quantità $e(k)$ è l'errore commesso alla k-esima iterazione, ovvero sul k-esimo pattern di addestramento, utilizzando il vettore peso $\mathbf{w}(k)$. Se utilizzassimo il vettore successivo $\mathbf{w}(k+1)$ avremmo

$$e'(k) = r(k) - \mathbf{w}^T(k+1)\mathbf{y}(k) \tag{2.5.39}$$

e di conseguenza una variazione nell'errore di

$$\begin{aligned} \Delta e(k) &= e'(k) - e(k) \\ &= [r(k) - \mathbf{w}^T(k+1)\mathbf{y}(k)] - [r(k) - \mathbf{w}^T(k)\mathbf{y}(k)] \\ &= (\mathbf{w}(k) - \mathbf{w}(k+1))^T \mathbf{y}(k) \\ &= -\Delta\mathbf{w}^T\mathbf{y}(k) \end{aligned} \tag{2.5.40}$$

Utilizzando quindi la (2.5.38) otteniamo

$$\Delta e(k) = -\alpha e(k)\mathbf{y}^T(k)\mathbf{y}(k) = -\alpha e(k) \left\| \mathbf{y}(k) \right\|^2 \tag{2.5.41}$$

La formula (2.5.41) ci dà un'importante informazione sul comportamento dell'algoritmo; infatti notiamo che l'aggiornamento del vettore peso alla k-esima iterazione, comporta un diminuzione dell'errore di un fattore $\alpha \left\| \mathbf{y}(k) \right\|^2$. Il coefficiente α controlla la stabilità e la velocità di convergenza dell'algoritmo. Si può dimostrare che anche in questo caso si ha sempre convergenza, ma le assunzioni fatte prevedono anche l'utilizzo di training patterns in classi non separabili.

2.5.2.2 Multilayer feedforward neural networks

In questa sezione ci focalizziamo su funzioni di decisione per problemi di classificazione a più classi, indipendentemente dalla loro separabilità.

L'architettura basilare Vogliamo costruire una rete neurale composta da perceptron, disposti in strati, detti *layer*; il termine *feedforward* sta ad indicare che ogni perceptron riceve un input dallo strato precedente e "dà in pasto" l'output ai perceptron del layer successivo.

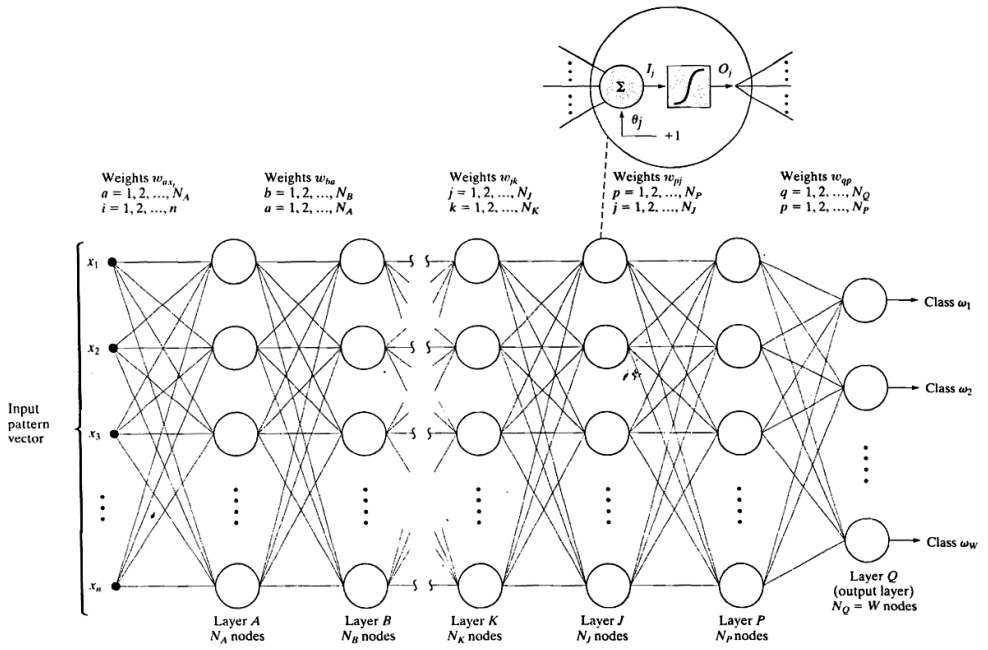


Figura 2.5.3: Schema del funzionamento di una rete neurale multilayer. Immagine tratta da [11].

La Figura 2.5.3 mostra l'architettura che prenderemo in considerazione in tutta questa sezione. Notiamo, come già detto, che essa è suddivisa in layer, a loro volta composti da perceptron; ognuno prende input da tutti i precedenti e restituisce l'output a tutti i perceptron del layer successivo. Ogni layer può avere un numero arbitrario di perceptron, utilizziamo la convenzione che il layer A possieda N_A unità di calcolo, il layer B ne abbia N_B e così via; supponiamo di essere sul layer J, allora indichiamo gli *elementi di attivazione* dei suoi nodi con la notazione I_j, per $j = 1, 2, \ldots, N_J$. Come si può notare, abbiamo stabilito di indicare i nodi e le quantità associate ad un layer con la corrispondente lettera minuscola: I_j è l'elemento di attivazione del j-esimo nodo del layer J, mentre I_q è l'elemento d'attivazione q del layer Q. Se K è lo strato che precede J allora avrà N_K

output indicati con O_k con $k = 1, \dots, N_K$; questi output vengono elaborati da ogni perceptron in J, tramite una media pesata, per generare un proprio elemento di attivazione

$$I_j = \sum_{k=1}^{N_K} w_{jk} O_k \qquad j = 1, \dots, N_J \tag{2.5.42}$$

Possiamo notare che ogni nodo del layer J può avere pesi differenti, infatti con w_{jk} indichiamo la k-esima componente del vettore peso del j-esimo nodo. Abbiamo detto che ogni perceptron calcola un proprio elemento di attivazione, che sarà l'input della propria funzione di attivazione definita come segue

$$h_j(I_j) = \frac{1}{1 + e^{-\frac{I_j + \theta_j}{\theta_0}}} \tag{2.5.43}$$

Questa funzione è una *sigmoide* ed il parametro θ_0 ne controlla la forma; vediamo che per poter essere calcolata ha bisogno di conoscere i coefficienti θ_j, per $j = 1, \dots, N_J$, associati anch'essi ad ogni nodo del layer. Il j-esimo perceptron calcolerà, quindi, l'output tramite la formula

$$O_j = h_j(I_j) \tag{2.5.44}$$

La funzione di attivazione, sfruttando la (2.5.42), può essere riscritta come segue

$$h_j(I_j) = \frac{1}{1 + e^{-\frac{\sum_{i=0}^{N_K} w_{ji} O_i + \theta_j}{\theta_0}}} \tag{2.5.45}$$

Training per propagazione all'indietro Iniziamo concentrandoci sul layer di output, ovvero quello che in Figura 2.5.3 viene denominato Q. L'errore quadratico totale tra la risposta desiderata r_q e O_q, ovvero quella data dai nodi del layer Q, si calcola come segue:

$$E_Q = \frac{1}{2} \sum_{q=1}^{N_Q} (r_q - O_q)^2 \tag{2.5.46}$$

dove $\frac{1}{2}$ è utilizzato per comodità di notazione, infatti verrà utile per i calcoli successivi. Decidiamo di aggiustare i vettori peso in relazione alla derivata parziale dell'errore rispetto ad essi, ovvero utilizzando la formula

$$\Delta w_{qp} = -\alpha \frac{\partial E_Q}{\partial w_{qp}} \tag{2.5.47}$$

dove si intende che il layer P precede Q. L'errore E_Q è una funzione degli output O_q, che sono a loro volta funzioni degli input I_q; possiamo utilizzare la regola della derivazione composta per ottenere:

$$\frac{\partial E_Q}{\partial w_{qp}} = \frac{\partial E_Q}{\partial I_q}\frac{\partial I_q}{\partial w_{qp}} \tag{2.5.48}$$

Dall'equazione (2.5.42)

$$\frac{\partial I_q}{\partial w_{qp}} = \frac{\partial}{\partial w_{qp}}\sum_{p=1}^{N_P} w_{qp}O_p = O_p \tag{2.5.49}$$

Sostituendo la (2.5.48) e la (2.5.49) nella (2.5.47) si ha

$$\Delta w_{qp} = -\alpha\frac{\partial E_Q}{\partial I_q}O_p \tag{2.5.50}$$

Definiamo ora $\delta_q = -\frac{\partial E_Q}{\partial I_q}$. Possiamo utilizzare ancora la regola della catena per calcolarlo rispetto a O_q e quindi a I_q:

$$\delta_q = -\frac{\partial E_Q}{\partial O_q}\frac{\partial O_q}{\partial I_q} \tag{2.5.51}$$

Ora è possibile avere una forma esplicita per δ_q, infatti calcoliamo dapprima

$$\begin{aligned}\frac{\partial E_Q}{\partial O_q} &= -(r_q - O_q)\\ \frac{\partial O_q}{\partial I_q} &= \frac{\partial}{\partial I_q}h_q(I_q) = h_q'(I_q)\end{aligned} \tag{2.5.52}$$

quindi scriviamo

$$\delta_q = (r_q - O_q)h_q'(I_q) \tag{2.5.53}$$

Andando a sostituire il risultato in (2.5.50) abbiamo

$$\Delta w_{qp} = \alpha(r_q - O_q)h_q'(I_q)O_p \tag{2.5.54}$$

per cui l'incremento che serve ad ottimizzare il vettore peso è proporzionale all'errore commesso. Si noti che abbiamo tutti gli ingredienti per valutare la (2.5.54), infatti O_p, I_q ed O_q sono già stati calcolati dai perceptron, la funzione h_q' è nota, così come le risposte esatte r_q.

Siamo partiti dall'ultimo layer; ora dobbiamo correggere man mano tutti gli altri. Andando all'indietro troviamo il layer P, per il quale abbiamo

$$\Delta w_{pj} = \alpha \delta_p O_j \tag{2.5.55}$$

dove

$$\delta_p = (r_p - O_p) h'_p(I_p) \tag{2.5.56}$$

In questo caso sono noti tutti i termini fuorché r_p, poiché non ha senso all'interno della rete neurale: se i nodi del layer P fossero in grado di produrre una risposta in termini di classificazione, non ci sarebbe bisogno del layer successivo. Dobbiamo tornare indietro e ricavare nuovamente δ_p, facendo attenzione a non includere r_p. Scriviamo l'errore per il layer P come

$$\delta_p = -\frac{\partial E_P}{\partial O_p} \frac{\partial O_p}{\partial I_p} \tag{2.5.57}$$

Il termine $\frac{\partial O_p}{\partial I_p}$ non presenta difficoltà, poiché risulta

$$\frac{\partial O_p}{\partial I_p} = h'_p(I_p) \tag{2.5.58}$$

Invece di calcolare $\frac{\partial E_P}{\partial O_p}$ direttamente, sfruttiamo la regola della catena, per esprimere E_P rispetto a I_q, che dipende a sua volta da O_p:

$$\begin{aligned} -\frac{\partial E_P}{\partial O_p} &= -\sum_{q=1}^{N_Q} \frac{\partial E_P}{\partial I_q} \frac{\partial I_q}{\partial O_p} \\ &= \sum_{q=1}^{N_Q} \left(-\frac{\partial E_P}{\partial I_q} \right) \frac{\partial}{\partial O_p} \sum_{p=1}^{N_P} w_{qp} O_p \\ &= \sum_{q=1}^{N_Q} \left(-\frac{\partial E_Q}{\partial I_q} \right) w_{qp} \\ &= \sum_{q=1}^{N_Q} \delta_q w_{qp} \end{aligned} \tag{2.5.59}$$

La formulazione finale dell'errore è

$$\delta_p = h'_p(I_p) \sum_{q=1}^{N_Q} \delta_q w_{qp} \tag{2.5.60}$$

L'importanza di questa equazione risiede nel fatto che dipende solo da termini che vengono calcolati al layer successivo; questo significa che abbiamo trovato un modo per propagare l'errore all'indietro e quindi, per calcolare tutti i nuovi vettori peso, dobbiamo sempre spostarci verso il layer precedente, utilizzando la (2.5.60).

La procedura di training si svolge in queste fasi: vengono scelti arbitrariamente dei vettori peso per ogni layer, quindi il primo pattern di riferimento viene sottoposto alla rete; una volta calcolati anche gli output O_q, si utilizzano le risposte note r_q per calcolare l'errore e quindi aggiornare i pesi. Successivamente si propaga l'errore all'indietro, aggiornando i pesi ad ogni passo. Terminata questa fase, ovvero adattata la rete neurale al primo pattern di riferimento, si passa al successivo. La stessa procedura si applica anche ai coefficienti θ_j semplicemente trattandoli come pesi aggiuntivi, ma non scendiamo nel dettaglio. Generalmente il processo di training si ferma quando converge ad un insieme di pesi stabili che rispondono ai nuovi training patterns solo con lievi fluttuazioni.

Capitolo 3

Il riconoscimento dell'individuo

3.1 Introduzione

(a) (b) (c)

Figura 3.1.1: L'evoluzione della fotografia; a partire da sinistra abbiamo: (a) Dagherrotipo (1839) (b) Polaroid Model 1000 S (1977) (c) Reflex Nikon D3200 (2012)

Il problema dell'identificazione delle persone ha origini molto lontane nel tempo. Lo scopo non deve per forza essere forense, quindi legato al riconoscimento dei criminali, infatti basta pensare che, tra il 1955 ed il 1913 a.C., nell'antica Babilonia le impronte digitali venivano utilizzate per firmare i contratti; un altro esempio proviene dalla Cina del VII secolo dopo Cristo, i documenti legali venivano firmati a mano, ma per chi fosse analfabeta era previsto l'uso dell'impronta digitale. [1]

[1] I testi di riferimento per tutto il capitolo sono [16] e [13].

L'obiettivo dell'identificazione è separare un individuo dal resto della collettività, utilizzando i criteri più sicuri tra quelli conosciuti; l'errore nel riconoscimento delle persone può portare, infatti, a conseguenze molto gravi: in campo penale anche la condanna a morte, ove prevista.

All'alba dell'umanità questo obiettivo veniva perseguito direttamente. Si pensi all'era tribale, in cui gli uomini erano suddivisi in comunità sufficientemente stabili, nelle quali i membri spesso si conoscevano gli uni con gli altri. In un simile contesto, il riconoscimento tra persone che si riunivano per cacciare, mangiare, dormire insieme risultava sicuramente molto agevole.

Col passare del tempo ogni tribù inizio a volersi diversificare dalle altre, assumendo, consciamente o inconsciamente, un aspetto fisico differente; iniziò anche la differenziazione per famiglie, oppure per ceti sociali. Cicatrici, tatuaggi, abbigliamento e gioielli erano tutte manifestazioni della volontà di distinguersi, cosicché riconoscere un individuo proveniente dalla classe più abbiente non fosse difficile, ma anche uno schiavo era facile da notare. Persino la giustizia iniziò a prevedere punizioni che, per i reati più gravi, ponessero un marchio duraturo sui criminali: dalla semplice gogna, all'offesa di alcune parti del corpo sino alla mutilazione. Riconoscere quelli che si macchiavano di particolari crimini significava tenerli sotto controllo ed anche aggiungere alla punizione corporale lo scherno della comunità e il disonore per sé stesso e la famiglia.

Il progresso, gli scambi internazionali di beni e persone che s'intensificavano e, più di recente, la nascita di internet e del commercio elettronico, la globalizzazione, tutti passi che ci hanno condotto al bisogno di sistemi d'autenticazione e d'individuazione. L'autenticazione persegue la necessità di verificare l'identità di chi esegue particolari azioni, tenta di entrare in aree riservate e accedere a risorse protette, oppure cerca di fare acquisti sui mercati elettronici. L'individuazione si pone, invece, l'obiettivo di scoprire chi ha commesso determinate azioni, oppure chi è stato in un particolare luogo, o più in generale a chi appartengono specifiche caratteristiche. Al giorno d'oggi abbiamo elevate competenze in questi due campi, ma anche i metodi che utilizziamo correntemente vanno migliorati sempre più, mirando a crearne a prova di errore.

Uno dei primi metodi di riconoscimento non invasivi, ovvero non associato a pene corporali o all'emarginazione dalla società, fu la rappresentazione pittorica dell'individuo. Non è affatto semplice riprodurre in forma pittorica ciò che un'altra persona ha visto; l'artista si affida alle parole dell'interlocutore e alla propria creatività. Le rappresentazioni di questo tipo migliorarono nel tempo, anche grazie all'utilizzo di terminologia e descrittori specifici, che rendevano più oggettivo il lavoro dell'artista, nonostante si possa raggiungere al massimo una rappresentazione che assomigli al soggetto, ma non sia mai identica ad esso.

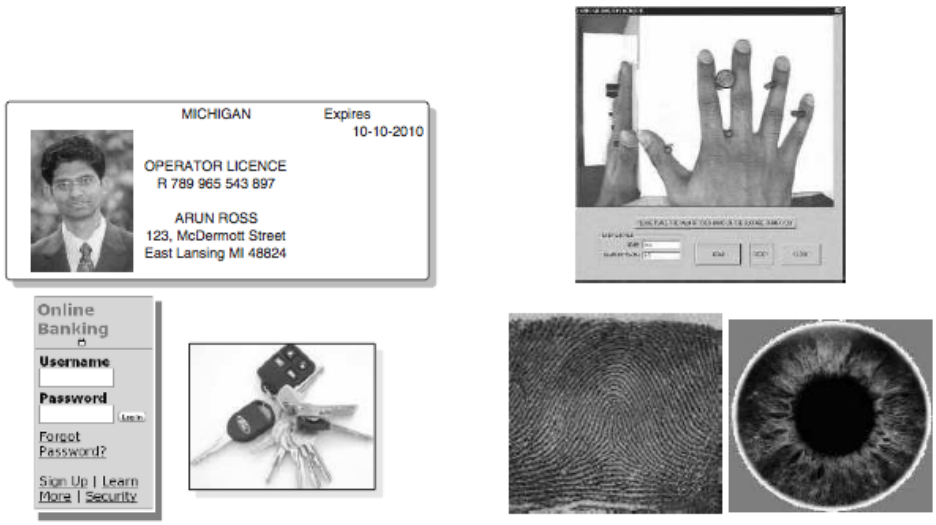
Successivamente la protagonista fu la fotografia, che poteva riprodurre il sog-

getto esattamente come era. Il primo modello di macchina fotografica, risalente al 1843, utilizzava una lastra metallica ricoperta di argento sensibile alla luce, su cui rimaneva impresso il riflesso della luce proveniente dal soggetto. A causa dell'ingombrante ed inizialmente costosa apparecchiatura, non era facile scattare foto in ambienti non controllati. Con l'invenzione delle pellicole in cellulosa, le macchine fotografiche rimpiccolirono e divennero meno costose; grazie a ciò crebbe moltissimo l'uso della documentazione fotografica. Un problema ancora rimaneva, infatti non era possibile vedere l'immagine catturata se non dopo un'elaborata procedura di sviluppo della fotografia. Posero un parziale rimedio le Polaroid, che riuscivano a stampare istantaneamente l'immagine fotografata, anche se la qualità e la durata di una tale stampa non erano sicuramente elevate. Fu solo quando, con lo sviluppo di nuove tecnologie, nacque la fotocamera digitale, che avvennero grossi miglioramenti; l'immagine può subito essere vista e, se insufficiente, rifatta, la spesa è minima e l'attrezzatura è tascabile. Tutto ciò che servirebbe è che qualcuno la usi mentre viene commesso un delitto: cosa assai difficile da ottenere. Per avere un metodo di identificazione efficace sono necessarie le caratteristiche della fotografia, ma vanno affiancate da altre informazioni che aiutino ad individuare una persona con maggiore accuratezza. Un primo rudimentale ma efficace tentativo fu fatto da Alphonse Bertillon, attorno al 1883; il sistema che prese il nome di *bertillonage* oppure *antropometria* era composto di quattro parti: antropometria vera e propria, ovvero le misurazioni salienti del corpo umano, descrizione verbale dell'individuo, sua fotografia ed impronta digitale.

3.1.1 Struttura di un sistema biometrico

È in questo contesto che trova applicazione la biometria: la scienza che studia come stabilire l'identità di un individuo a partire da attributi fisici, chimici o comportamentali. La biometria offre una soluzione affidabile per l'autenticazione e l'individuazione, utilizzando sistemi completamente o parzialmente automatizzati così da riconoscere una persona sulla base delle sue caratteristiche biologiche. Lo sfruttamento di tali procedure permette il riconoscimento tramite lo studio di come effettivamente è l'individuo, di sue proprie caratteristiche, piuttosto che affidandosi alla sua memoria, come nel caso delle password, oppure a ciò che possiede, come quando viene chiesto di esibire documenti d'identità. Per aumentare ulteriormente il livello di sicurezza, possono anche essere utilizzati due approcci contemporaneamente: ad esempio, biometrico e password. In Figura 3.1.2(a) e (b) vengono mostrati alcuni semplici esempi di sistemi di sicurezza.

Nel 2003, O'Gorman [12] fa un elenco dei principali possibili attacchi ai si-



(a) (b)

Figura 3.1.2: (a) Alcuni sistemi di sicurezza basati oggetti e password. (b) Esempi di caratteristiche biometriche utili per l'individuazione e l'autenticazione. Immagini tratte da [16].

stemi di sicurezza conosciuti, tra quelli basati su password e sull'esibizione di documenti:

1. attacchi verso gli utenti, cercando di rubare direttamente le credenziali di accesso;
2. attacchi diretti all'ente di verifica delle credenziali, nel tentativo di ottenere in massa la documentazione su tutti gli utenti;
3. intercettazione dei messaggi scambiati tra utente e sistema di autenticazione;
4. disconoscimento dell'avvenuta autenticazione, facendo finta che ad utilizzare il sistema di sicurezza sia stata un'altra persona;
5. attacco tramite i cosiddetti "cavalli di Troia", particolari programmi in grado di captare le informazioni manipolate da un elaboratore;
6. interruzione del servizio, provocata volontariamente inserendo numerose volte credenziali non valide.

Nonostante alcuni di questi attacchi possano essere dirottati incorporando appropriati meccanismi di difesa, non è possibile gestire tutti questi tipi di attacchi.

Poiché questi derivano dall'utilizzo di password e documentazione, potrebbe essere d'aiuto utilizzare sistemi biometrici. In effetti il loro utilizzo offre tre principali vantaggi, in questo ambito: difficoltà di sottrarre le credenziali, riconoscimento negativo e resistenza al disconoscimento. Quanto alla difficoltà di sottrarre il mezzo per accedere al sistema di sicurezza, risulta evidente che non si può privare un uomo delle proprie caratteristiche fisiche, inoltre resta comunque difficile assumere le stesse caratteristiche di un altro individuo. Il riconoscimento negativo è il processo tramite cui un sistema di sicurezza comprende di essere già stato sfruttato da un particolare individuo; per fare un esempio: si potrebbe pensare di richiedere due volte lo stesso finanziamento pubblico, accedendo con credenziali differenti; con la biometria si potrebbe capire che le differenti credenziali corrispondono, invero, ad un solo impostore. La resistenza al disconoscimento, di cui al punto 4, si riferisce al fatto che, utilizzando sistemi biometrici, non sia possibile utilizzare il sistema di sicurezza e successivamente far finta di non averlo fatto: la probabilità che due diversi utenti possiedano caratteristiche fisiche così simili da garantire loro lo stesso accesso è molto bassa.

I sistemi di sicurezza biometrici possono utilizzare un vasta gamma di caratteristiche fisiche e comportamentali, che include impronte digitali, geometria del volto, della mano o delle dita, iride, retina, firma, impronta del palmo della mano, voce, orecchio e DNA. Ci si riferisce ad ognuna di queste caratteristiche con il termine *indicatori biometrici* (o anche *tratti*, *identificatori*).

Un sistema biometrico è sostanzialmente un meccanismo in grado di raccogliere dati da un individuo, estrarre le caratteristiche d'interesse e confrontarle con i modelli già presenti nella banca dati. In base al risultato del confronto il sistema reagirà in maniera positiva o negativa. Possiamo, quindi, suddividere tale sistema in moduli.

1. **Sensore**: un lettore in grado di acquisire il dato biometrico grezzo da un individuo; ad esempio, nel campo delle impronte digitali si utilizzano dei sensori ottici che riescono bene a trasformare le increspature dell'epidermide in immagini. Sostanzialmente il sensore permette l'interazione dell'utente con il sistema di sicurezza, quindi è fondamentale ai fini della corretta identificazione od autenticazione; un sensore mal studiato può risultare in acquisizioni di bassa qualità, da cui non sia possibile estrarre indicatori biometrici con accuratezza.

2. **Accertamento della qualità ed estrazione degli indicatori**: una volta acquisito il dato grezzo, ne va verificata la qualità, per decidere se procedere oppure chiedere nuovamente i dati biometrici all'utente. Generalmente i dati grezzi vengono sottoposti a metodi di miglioramento che danno i risultati sperati; può comunque capitare che le acquisizioni siano di qualità davvero bassa. Se la qualità viene assicurata, il dato appena

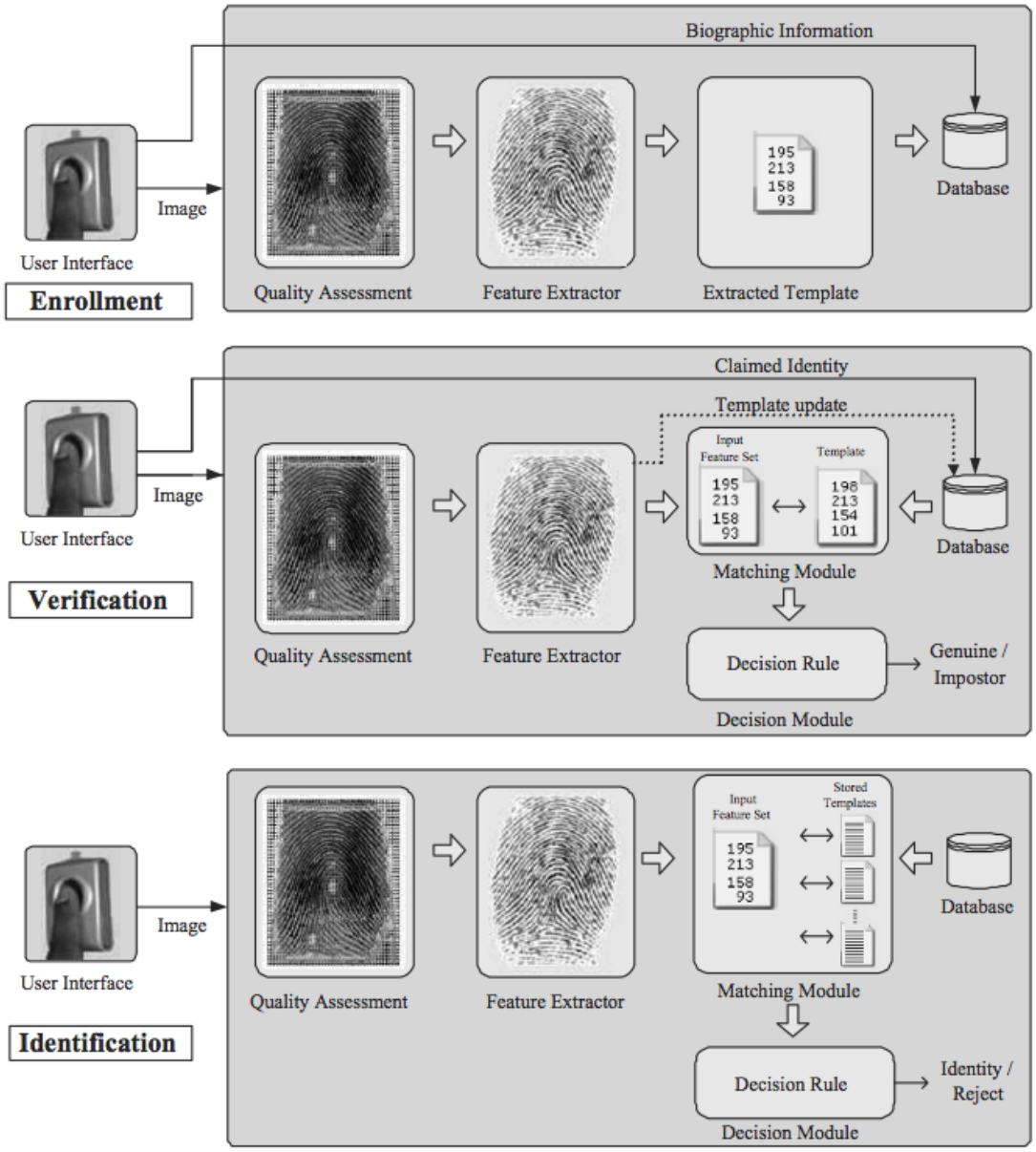


Figura 3.1.3: Processo di inserimento degli utenti nella banca dati e le due possibili applicazioni di un sistema biometrico: autenticazione ed individuazione. Per ogni procedura sono schematizzate le operazioni compiute dai singoli moduli. Immagine tratta da [16].

migliorato viene sottoposto ad ulteriore elaborazione per estrarne le caratteristiche biometriche che ci interessano. Per esempio, nel caso delle impronte digitali le immagini vengono prima migliorate, quindi si estraggono dei particolari punti chiamati *minutiae*, le cui posizione ed orientazione costituiscono gli indicatori biometrici associati all'impronta digitale.

3. **Confronto e decisione**: le caratteristiche acquisite vengono confrontate con quelle presenti nella banca dati, quindi viene associato un punteggio ad ogni confronto effettuato; se esiste almeno un punteggio superiore ad una soglia stabilita a priori, allora viene segnalata una corrispondenza tra i rispettivi indicatori biometrici, altrimenti si evidenzia che l'utente non è stato ancora inserito nella banca dati.

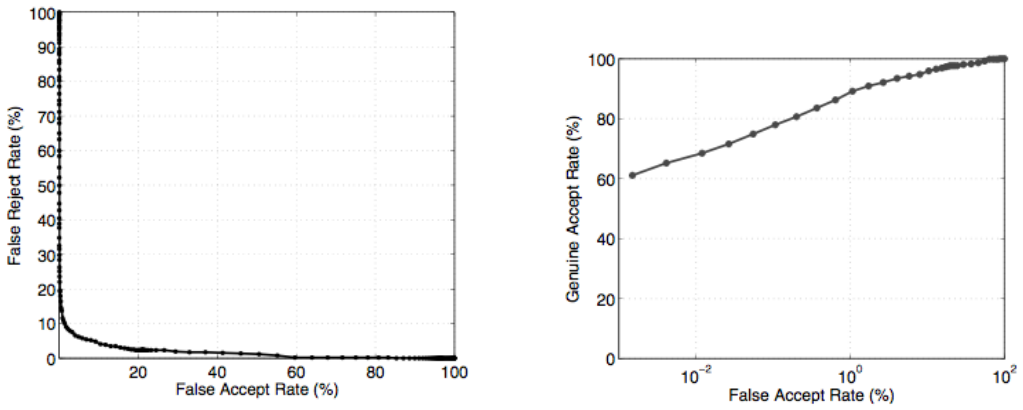
4. **Database**: ovvero la banca dati, che serve a tutto il sistema per attuare confronti tra gli indicatori biometrici elaborati. Durante il processo di inserimento dei dati nel database, vengono utilizzati i primi due moduli per ottenere gli indicatori biometrici da ogni individuo che si sottopone al sistema. Ad ognuno vengono associate sia le caratteristiche biometriche, sia informazioni biografiche, come nome, codice fiscale, residenza e così via.

3.1.2 Prestazioni di un sistema biometrico

Al contrario di ciò che avviene per un sistema di sicurezza basato su password, in cui la convalida dell'identità di un individuo prevede una corrispondenza perfetta tra stringhe alfanumeriche, per un sistema biometrico è raro trovare due letture successive dell'indicatore biometrico perfettamente identiche. I motivi principali sono le imperfezioni dell'acquisizione del dato grezzo, la variazione nel tempo, anche lieve, delle caratteristiche dell'individuo, i cambiamenti delle condizioni ambientali in cui il sensore opera e le differenze con cui l'utente interagisce col sistema (nel caso delle impronte digitali, diversa pressione del dito sul sensore). Alla luce di questo, una perfetta corrispondenza potrebbe essere il campanello di allarme per un attacco al sistema basato sulla replica dei campioni del database.

La variabilità osservata in un insieme di campioni di caratteristiche biometriche di un individuo viene chiamata *variazione intra-classe*, mentre la variabilità che si presenta tra insiemi di caratteristiche proveniente da due differenti persone prende il nome di *variazione inter-classe*. Un buon indicatore biometrico mostra piccole variazioni intra-classe e grandi variazioni inter-classe.

Per misurare il grado di similitudine tra due differenti campioni di caratteristiche biometriche si fa ricorso ad un punteggio, quindi si stabilisce un valore di soglia η che determina l'accettazione del campione, per valori superiori, op-



(a) (b)

Figura 3.1.4: (a) Esempio di curva ROC, con cui si può individuare il valore di soglia ottimale. (b) Esempio di curva ROC in cui il GAR prende il posto del FRR. Immagini tratte da [16].

pure il suo rifiuto, per valori inferiori. Il punteggio che si ottiene confrontando due campioni dello stesso utente, viene detto *autentico* o *genuino*; il punteggio ottenuto dal confronto tra campioni provenienti da utenti differenti si dice *impostore*. Un punteggio genuino inferiore ad η costituisce un *falso negativo* ed un punteggio impostore superiore ad η si dice *falso positivo*. La proporzione di falsi positivi in un sistema biometrico, indicata con *FAR* dall'inglese *False Accept Rate*, si esprime come rapporto tra la quantità di falsi positivi ed il numero totale di punteggi impostori; similmente la proporzione di falsi negativi, *FRR* da *False Reject Rate*, viene definita come frazione tra il numero di falsi negativi ed la totalità dei punteggi autentici. Il *GAR*, ovvero *Genuine Accept Rate*, è il numero di veri positivi, cioè punteggi genuini superiori alla soglia η, e si può calcolare come:

$$GAR = 1 - FRR \tag{3.1.1}$$

Cambiare il valore η può diminuire *FAR* oppure *FRR*, mai entrambi. Definiamo in ultimo una delle curve più utilizzate per verificare le prestazioni di un sistema biometrico, ovvero la *ROC*, dall'inglese *Receiver Operating Characteristic*; questa si ottiene tracciando in un grafico i valori del *FAR* contro i valori del *FRR*, al variare del parametro η. Osservando questo grafico si può capire la reciproca dipendenza tra le due variabili *FAR* e *FRR*, quindi si può scegliere il parametro η che meglio soddisfa le proprie esigenze; in Figura 3.1.4(a) è visibile un esempio di curva ROC. A volte il grafico ROC viene tracciato utilizzando il GAR, anziché il FRR, come mostra la Figura 3.1.4(b).

3.1.3 Caratteristiche dei sistemi biometrici

Molte sono le caratteristiche biometriche che vengono utilizzate nelle varie applicazioni. Ogni indicatore ha i suoi vantaggi e svantaggi, per cui la scelta dipende da una serie di fattori, che comprendono ma non si esauriscono con le prestazioni appena discusse. Gli autori di [15] hanno identificato sette fattori che determinano l'idoneità di un indicatore per una particolare applicazione.

- **Universalità**: ogni possibile utente deve possedere il tratto biometrico in questione.
- **Unicità**: l'indicatore deve possedere un'alta variabilità inter-classe.
- **Permanenza**: una caratteristica che muta troppo rapidamente nel tempo non è affatto utile.
- **Misurabilità**: deve essere possibile acquisire e digitalizzare il dato biometrico e deve essere in un forma adatta all'elaborazione successiva.
- **Prestazioni**: l'accuratezza del riconoscimento e le risorse richieste per ottenerla devono essere compatibili con l'applicazione.
- **Accettabilità**: gli individui della popolazione che utilizzerà il sistema di sicurezza dovrà essere propensa a sottoporre il proprio tratto biometrico.
- **Aggiramento**: consiste nella facilità di imitare o riprodurre il tratto per aggirare il controllo offerto dal sistema.

Ovviamente nessun sistema biometrico deve soddisfare tutti questi requisiti per ogni possibile applicazione. Quelle elencate sono caratteristiche che si riferiscono ad un sistema ideale e gli indicatori non devono per forza essere così, basta che siano ammissibili: vanno scelti in modo che, per la singola applicazione in oggetto, si avvicinino sufficientemente alle caratteristiche ideali.

3.2 I principali identificatori biometrici

In questa sezione verranno discussi i principali identificatori biometrici, mostrando i loro vantaggi e svantaggi in accordo allo schema delineato nell'introduzione. Poiché le impronte digitali sono l'argomento centrale di questa tesi, verrà loro dato un maggior peso nelle sezioni successive.

3.2.1 Impronte digitali

Le impronte digitali sono sempre state oggetto di curiosità e di studio, sin dalla nascita del genere umano. Ciò che una volta era visto solamente come ricoprimento più esterno del nostro corpo, ora è stato attentamente studiato ed ha un ruolo importantissimo nella società. Sin dal 1823, grazie allo studioso John Evangelista Purkinje, sappiamo che le impronte digitali sono una caratteristica che ci contraddistingue da tutti gli altri esseri umani; ogni primate possiede impronte digitali, ma nonostante questo, non solo non esistono due umani con le stesse impronte digitali, ma l'unicità rimane anche se estendessimo la nostra ricerca tra tutti i primati. Prima di allora ci sono reperti che documentano la conoscenza, a volte l'uso, delle impronte digitali, ma senza averne fatto un'analisi formale. Documenti pittorici preistorici mostrano l'utilizzo delle impronte a scopo di identificazione, probabilmente in un contesto legato al soprannaturale ed alla superstizione. Altri reperti sono stati rinvenuti attorno alle mummie dell'Antico Egitto, in Mesopotamia e in alcune pitture in Nuova Scozia e Francia. Ulteriori esempi del loro uso provengono dalla Cina della dinastia Tang, in cui venivano utilizzate per la firma dei contratti. In Palestina è stato persino trovato molto vasellame con raffigurazioni di impronte digitali. Nel 1686, Marcello Malpighi, professore di anatomia all'Università di Bologna, appuntò nei suoi scritti la presenza di creste, spirali e linee chiuse nelle impronte digitali. Da allora molti scienziati investigarono i motivi impressi sulla nostra pelle.

L'uso delle impronte digitali come mezzo d'identificazione iniziò nel XIX secolo, con l'avvento dei sistemi di classificazione delle impronte. Nel 1880, Henry Fauld intuì l'unicità della struttura delle impronte a partire da osservazioni empiriche. Nel 1888 Sir Francis Galton introdusse il concetto di minuzia, ancora oggi utilizzato, per il confronto tra impronte digitali. Nei primi decenni del XX secolo queste si affermarono come metodo di identificazione valido in campo forense; negli anni '60 l'FBI iniziò a dedicare copiose risorse per lo sviluppo di sistemi automatici per il riconoscimento basato su impronte digitali. Da allora sono stati fatti enormi progressi nell'analisi automatica, la tecnologia ha permesso di avere a poco prezzo ottimi calcolatori e l'uso delle impronte per il riconoscimento si è così diffuso che ne hanno a che fare ricercatori e forze di polizia di tutto il mondo.

Per capire fino in fondo il processo che porta all'identificazione tramite impronte digitali, occorre conoscere la relativa terminologia. Iniziamo a dare le definizioni basilari, ovvero classifichiamo le impronte lasciate da una persona a seconda della parte del corpo che l'ha impressa. Un'*impronta digitale* (*fingerprint*) è il residuo lasciato su un qualunque materiale dall'epidermide, ovvero lo strato più esterno del tessuto epiteliale. Molto spesso associamo le impronte digitali ai polpastrelli, ma in realtà tutto il palmo della mano è in grado di la-

sciarne: nel caso in cui si abbia l'impronta di tutta la parte interna della mano, ci si riferisce con i termini *impronta palmare* (*palm print*). Un discorso analogo vale anche per l'epidermide della parte del piede a contatto con il suolo, per cui si parla di *impronta del piede* (*footprint*).

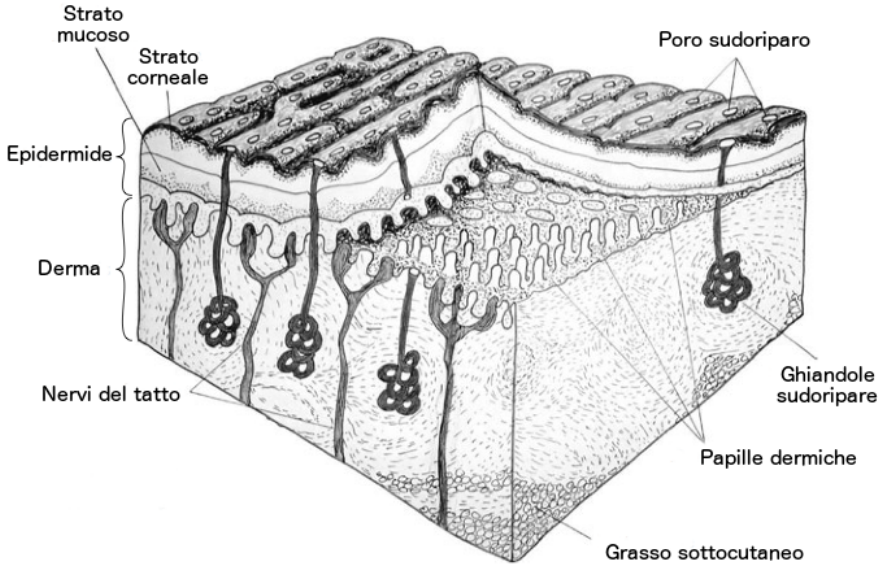


Figura 3.2.1: Sezione dello strato più esterno della pelle.

Tutte le impronte in questione sono caratterizzate da linee sopraelevate dette *creste* e zone di depressione chiamate *valli*. La loro funzione sull'epidermide è quella di aumentare l'attrito quando si afferrano oggetti. Questi motivi si formano sulla pelle durante il terzo o quarto mese di gravidanza (approssimativamente a 120 giorni dal concepimento). Il processo con cui si sviluppano le increspature della pelle avviene attraverso la formazione di piccole isole attorno ai pori, che piano piano si sviluppano sino a formare le creste. In Figura 3.2.1 si può vedere una sezione degli strati più esterni della pelle, ovvero derma ed epidermide.

I tratti più importanti delle impronte digitali, nel nostro ambito, sono quelli elencati di seguito.

- L'impronta, in condizioni normali, non cambia nel tempo, dalla nascita sino alla decomposizione dopo la morte. L'unica eccezione a questa regola è che durante l'accrescimento, come tutte le parti del corpo, anche l'epidermide si sviluppa; nonostante ciò comunque la struttura delle increspature rimane invariata. Anche con l'avanzamento dell'età la pelle subisce un cambiamento, ovvero si deteriora, però anche in questo caso la struttura che ci interessa per la classificazione non muta.

- Se una ferita intacca l'epidermide, le increspature vengono danneggiate; questa mutazione è solo temporanea, infatti l'epidermide si rigenera completamente. Se viene offeso anche il derma, ovvero la ferita oltrepassa le papille dermiche, allora il danno diviene permanente, poiché una cicatrice prende il posto della ferita; la lesione è però localizzata alla sola cicatrice, mentre la rimanente struttura è intatta. Ci sono stati casi in cui dei criminali hanno bruciato oppure chirurgicamente alterato l'epidermide delle proprie mani; va detto che le impronte digitali vengono lasciate da qualunque parte del corpo, per cui pensare di eliminare tutte le increspature del proprio epitelio è impensabile, a meno che non si preferisca mettere le proprie mani fuori uso a causa di lesioni troppo gravi. Oltre a questo va detto che, seppure si riuscisse a compromettere la classificazione della propria impronta, l'identificazione, ovvero l'associazione di un'impronta ad un individuo, è comunque possibile: l'alterazione diventa il tratto caratteristico della nuova impronta.

- Come abbiamo detto, lo sviluppo delle impronte digitali avviene durante la gravidanza. La loro formazione è simile a quanto avviene per lo sviluppo dei capillari in angiogenesi; il codice nel DNA dà alcune indicazioni generali sul modo con cui le increspature si devono formare sulla pelle, ma i dettagli sono frutto di eventi casuali, tra cui l'esatta posizione del feto nell'utero e la composizione del liquido amniotico. Proprio per questo motivo anche due gemelli hanno impronte digitali differenti, rendendo questa caratteristica fisica un appetibile indicatore biometrico.

3.2.2 Identificazione basata su DNA

Come abbiamo già detto, nessun indicatore biometrico consente un metodo d'identificazione ottimo per ogni applicazione possibile; ad ogni modo, a seconda delle situazioni, una particolare biometrica risulta essere migliore delle altre. Nel caso in cui si debbano confrontare caratteristiche genetiche, il test del DNA (acido deossiribonucleico) risulta essere il più potente ed affidabile strumento. Tutti gli organismi viventi sono il risultato di complesse interazioni tra l'originale patrimonio genetico e l'ambiente esterno. Nonostante molte cellule sono in grado di esprimere solo parte del loro patrimonio genetico, il DNA in ognuna (esclusi i gameti) contiene tutte le informazioni genetiche che riguardano quel particolare organismo. Queste caratteristiche garantiscono l'omogeneità del DNA in tutto l'organismo, cosa alquanto utile per la biometria.

La storia del DNA, in confronto delle impronte digitali, è molto breve. Inizia quando Oswald Avery nel 1944 definisce il ruolo del DNA all'interno di una cellula, come veicolo di trasmissione dei tratti ereditari. Nel 1953, James Watson e

Francis Crick individuano la struttura a doppia elica; nel 1980 David Botstein ed i suoi collaboratori iniziarono a studiare le piccole variazioni del genoma umano. Infine nel 1984, Jeffreys, Wilson e Thein scoprirono come applicare la variabilità del DNA al problema del riconoscimento, chiamando tale tecnica "DNA fingerprint", ovvero "l'impronta digitale del DNA". Il primo uso forense del DNA ebbe luogo in Inghilterra e venne utilizzato il metodo esposto proprio da Jeffreys; unitamente ad un'indagine classica, la polizia inglese fu in grado d'identificare l'assassino di due giovani ragazze. La cosa più significativa è che è stata la prima verifica dell'innocenza di un sospettato tramite il DNA. Quelli esposti sono stati solamente dei cenni ai passi salienti del percorso che ha portato a vedere il genoma come indicatore biometrico, in questo modo ne abbiamo una collocazione temporale.

Per comprendere come l'analisi del DNA riesca a differenziare gli individui, dobbiamo andare ad esplorare la sua variabilità. La maggior parte del genoma umano, tra il 99.5% ed il 99.9%, è condiviso tra tutti gli individui e dedicata alle specifiche fisiche e biologiche del generico corpo umano, lasciando solamente un piccolo margine, che va da 0.1% a 0.5%, per la personalizzazione. Può sembrare che, di conseguenza, dal punto di vista biologico siamo tutti uguali, invece anche in quella piccola percentuale di DNA risiedono milioni di coppie di basi azotate, ovvero i mattoncini di cui si compone la catena del genoma.

Il test del DNA ha i suoi lati negativi, che consistono principalmente nella laboriosità del procedimento di estrazione e confronto, nella difficoltà ad automatizzarlo e di conseguenza nella lentezza con cui si ottengono i risultati. Ad ogni modo, quando non si possiede nessun altro reperto biologico di un individuo se non poche cellule, le uniche tecniche che si possono utilizzare e che assicurano comunque una buona accuratezza sono quelle che riguardano il DNA. In effetti ogni volta che un individuo lascia delle cellule con nucleo, ovvero quasi tutte a parte eritrociti e cellule dell'epitelio, ha anche lasciato la sua firma, il suo DNA. L'utilità di questo metodo si manifesta particolarmente dove sono stati commessi crimini violenti, per cui spesso vengono lasciate involontariamente residui di sangue, seme, capelli o tracce di tessuto epiteliale.

Poiché la molecola del DNA contiene le informazioni ereditarie che vengono trasmesse di generazione in generazione, risulta la scelta più ovvia per determinare le parentele. Problemi legati all'immigrazione, alle dispute sulla paternità, al riconoscimento di cadaveri o parti di essi sono stati tra i primi utilizzi del test del DNA; in questo caso vengono confrontati i campioni di DNA prelevati, con quelli dei presunti familiari per determinare eventuali rapporti di parentela ed agevolare quindi il riconoscimento dell'immigrato o della vittima.

3.2.2.1 Le basi della genetica

La molecola di DNA è costituita dall'accostamento di moltissime unità di base, chiamate *nucleotidi* oppure *basi azotate*. Ne esistono solo quattro tipi: *adenina* (A), *citosina* (C), *guanina* (G) e *timina* (T). La sequenza di questi nucleotidi determina tutto il patrimonio genetico di un individuo.

Le proprietà chimiche e genetiche del DNA sono strettamente legate alla sua struttura fisica; in natura esso si presenta come una doppia elica: due filamenti di nucleotidi si attaccano uno con l'altro, tenuti insieme da legami idrogeno; l'aspetto è quello di una scala a pioli che si avvolge due volte su sé stessa. Ogni piolo, ovvero ogni legame idrogeno, unisce due nucleotidi andando a formare una coppia: i legami non sono affatto casuali, anzi possono formarsi solo specifiche coppie di basi, ossia A-T e C-G; pertanto A si dice *complementare* a T, così come C è il complementare di G (vedi Figura 3.2.2). In natura l'abbinamento delle basi

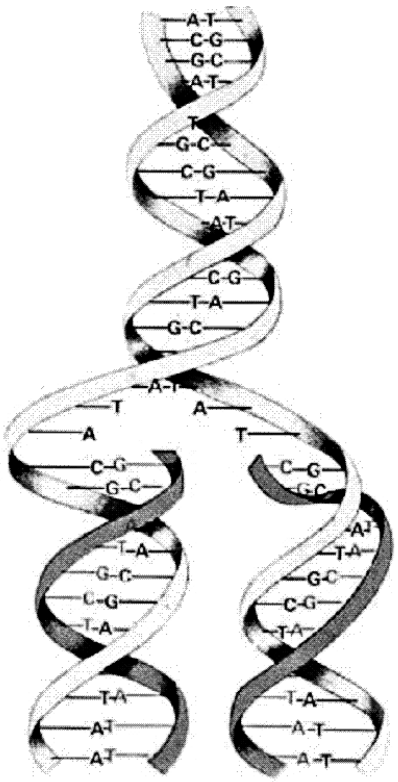


Figura 3.2.2: Nella figura si può vedere la struttura a doppia elica tipica della molecola di DNA

limitato ai complementari è il responsabile della corretta replica della molecola di DNA, per poi passarla alla generazione successiva. La copia del DNA, in parole molto povere, avviene per mezzo di alcuni enzimi che spezzano la doppia elica

separando i due filamenti di nucleotidi, successivamente ad ognuno si attaccano nuove basi azotate (seguendo la legge dei complementari) e si ottengono in questo modo due identiche doppie eliche. Questo processo può essere ricreato in vitro ed è la base per la tecnica che prende il nome di PCR, ovvero *polymerase chain reaction*, significa "reazione a catena della polimerasi".

3.2.2.2 Analisi e conservazione del DNA

Attualmente si utilizzano due tecniche, chiamate RFLP (Restriction Fragment Length Polymorphism) e PCR. Nel caso si utilizzi la prima, si preleva un ben definito segmento di DNA, di lunghezza fissata, e si sottopone il campione ad un autoradiografo, per scoprire la distribuzione delle basi azotate. Il risultato del test ricorda vagamente un tipico scontrino fiscale (vedi Figura 3.2.3), in cui si può leggere la risposta al test analizzando le bande di colore scuro corrispondenti al campione che ci interessa. Il vantaggio di questa tecnica è che risulta essere quella con il potere discriminante più elevato, grazie al fatto che essa può andare a scrutare molti punti della catena del DNA, quindi più sono i frammenti che si riescono a controllare, più sono le possibilità di trovare differenze tra le catene. Il problema che concerne l'utilizzo di questo metodo è che richiede un numero maggiore di campioni di DNA e, soprattutto, essi devono essere di buona qualità. Un'analisi del campione tramite PCR, ha bisogno della stessa procedura preparatoria, ma ha un tempo di risposta molto minore rispetto all'RFLP e può essere parzialmente automatizzato; il problema, in questo caso, è che le risposte non sono sempre attendibili, quindi risulta poco efficace in campo forense.

Un aspetto molto importante per l'analisi del DNA è sicuramente l'acquisizione e la conservazione del materiale genetico. Dal momento in cui il materiale biologico è al di fuori del corpo che lo ha prodotto, inizia la fase di degradazione, la quale può essere causa di test inefficaci. I fattori principali che conducono alla degradazione sono il tempo, la temperatura, l'umidità, la luce solare e ultravioletta oppure la presenza di altre sostanze chimiche o biologiche. Ultimamente sono stati condotti numerosi studi per capire come i risultati dei test varino in funzione del livello di degradazione dei campioni. È stato dimostrato che i fattori esterni possono spezzare le catene del DNA, ma non lo modificano da un tipo in un altro; in sostanza è stato provato che un test su un campione molto degradato può essere inconcludente, ma non attribuirà mai il campione ad un individuo che non sia il legittimo donatore. Possiamo effettivamente dire che la degradazione limita l'utilizzabilità del DNA, ma non l'attendibilità. Una delle principali mansioni di chi reperisce oppure conserva materiale genetico è bloccare il processo di degradazione, in maniera tale che il campione non si rovini ulteriormente. In generale i processi biologici sono rallentati rimuovendo l'umidità ed abbassando le temperature.

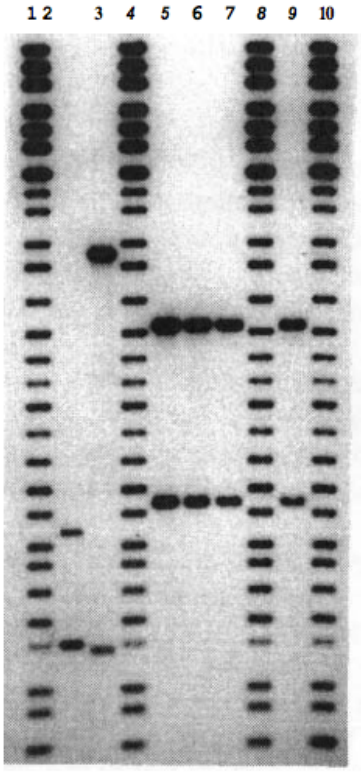


Figura 3.2.3: Risultato di un test RFLP. Le colonne 1, 4, 8, 10 si riferiscono a molecole sintetiche utilizzate come termine di paragone, mentre il test non distingue i donatori dei campioni 5, 6, 7, 9, che potrebbero essere, di conseguenza, la stessa persona. I donatori dei DNA in 2 e 3, invece, non hanno corrispondenza con gli altri, né tra loro, per cui secondo questo test non si possono considerare provenienti dallo stesso individuo. Immagine tratta da [15].

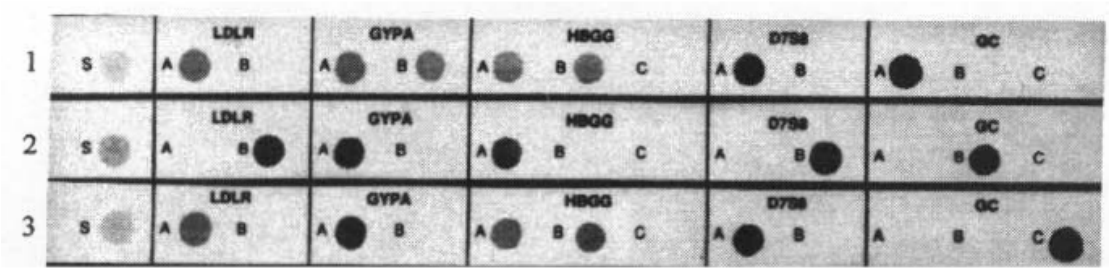


Figura 3.2.4: Risultato finale di un test del DNA tramite PCR. Ogni striscia orizzontale rappresenta un campione, mentre le suddivisioni verticali corrispondono a particolari zone della catena di nucleotidi, chiamate *loci*. I pallini segnati con una **S** sono solamente di controllo, mentre, affinché il test sia positivo, bisogna verificare che le striscie orizzontali abbiamo pallini corrispondenti; i nomi a fianco di tali pallini sono il nome del *locus* in esame. Notiamo in questo caso che i tre campioni probabilmente provengono da individui differenti, poiché i motivi non corrispondono. Immagine tratta da [15].

Occorre sicuramente fare menzione anche della possibile contaminazione del materiale reperito, che va ad interferire con le analisi successive. Esistono principalmente questi tipi di contaminazione: non biologica, non umana ed umana. Per quanto riguarda la contaminazione non biologica, per esempio la presenza di saponi o altri agenti chimici, possiamo dire che interferisce producendo risultati inconcludenti, ma mai errati; infatti la presenza di tali reagenti provoca semplicemente la degradazione ulteriore del materiale biologico presente. Situazione analoga si può dire per la contaminazione non umana, poiché il patrimonio genetico di altri organismi è ben diverso dal nostro e siamo in grado di riconoscerlo durante le analisi. Caso ben diverso è la contaminazione umana, poiché si aggiunge materiale compatibile ma proveniente da individui differenti, quindi è impossibile captare la contaminazione; inoltre, a seconda di quali cellule si scelgono per il processo di analisi, il test potrebbe scegliere l'individuo sbagliato. Un'importante differenziazione va fatta in questo caso: un conto è avere un campione misto, raccolto appositamente, ed un altro è avere un campione di un singolo individuo, che viene contaminato in fase di acquisizione, conservazione oppure elaborazione.

3.2.2.3 La retina come indicatore biometrico

La scansione della retina (RI, dall'inglese *retinal identification*) è un metodo automatico che assicura la vera identità di una persona, acquisendone ed analizzandone un'immagine della parte interna dell'occhio. Questa tecnica ha trovato numerose applicazioni negli ambienti che necessitano di altissima sicurezza,

come ad esempio centri per la ricerca nucleare, depositi di armi, servizi per la gestione delle comunicazioni e così via.

L'unicità della distribuzione dei vasi sanguigni della retina è nota sin dal 1935, quando due oftalmologi, Carleton Simon e Isodore Goldstein, mentre studiavano la malattie dell'occhio, fecero una sconcertante scoperta: i vasi sanguigni creano un motivo diverso in ogni occhio; i due scienziati successivamente pubblicarono un articolo in cui facevano uso di fotografie della retina per l'identificazione dei soggetti. Negli anni '50, le loro conclusioni furono supportate da Paul Twins, che, mentre era intento a studiare le caratteristiche che accomunano due gemelli, notò che la distribuzione dei vasi sanguigni della retina era l'indicatore biometrico che mostrava minore similarità. Negli anni '70, molte aziende produttrici di strumentazione per l'oftalmologia iniziarono a modificare le macchine per l'esame del fondo oculare, così da avere immagini adatte per analizzare in tempo reale la retina. Utilizzando quel tipo di apparecchiatura, c'erano tre principali svantaggi:

- l'allineamento della macchina con il fondo oculare doveva essere perfetto, per cui era necessario l'intervento di un operatore specializzato;
- era indispensabile un'ottima illuminazione, che risultava fastidiosa all'utente che si sottoponeva al test, di conseguenza le pupille si restringevano e la zona visibile della retina diminuiva;
- i macchinari erano troppo complessi, quindi troppo costosi per essere utilizzati su larga scala.

Invece di utilizzare la luce bianca, si iniziò a fare esperimenti con luce infrarossa, invisibile all'occhio umano e pertanto non fastidiosa. Emettitori di tali frequenze di luce si trovavano a buon mercato, per cui l'apparecchiatura risultante era economica. Il primo prototipo venne realizzato nel 1981. Da allora la tecnologia si è sviluppata e la strumentazione perfezionata, ma il concetto basilare è rimasto lo stesso.

Un punto, notevolmente a favore dell'uso della retina per il riconoscimento, è che quella parte dell'occhio si trova in uno degli ambienti più stabili del nostro corpo; la sua posizione molto interna, inoltre, fa sì che non sia soggetta ad agenti esterni che ne modifichino l'aspetto, come nel caso delle impronte digitali. A differenza di ciò che si può pensare, comunque, l'analisi della retina non ha un costo elevato, né abbiamo bisogno di apparecchiature di grandi dimensioni, poiché il dato grezzo può essere acquisito a bassa risoluzione e senza troppa attenzione per la messa a fuoco.

Facendo riferimento alla Figura 3.2.5, vediamo che la retina è localizzata sul fondo dell'occhio, esattamente nella zona di scansione evidenziata. In analogia

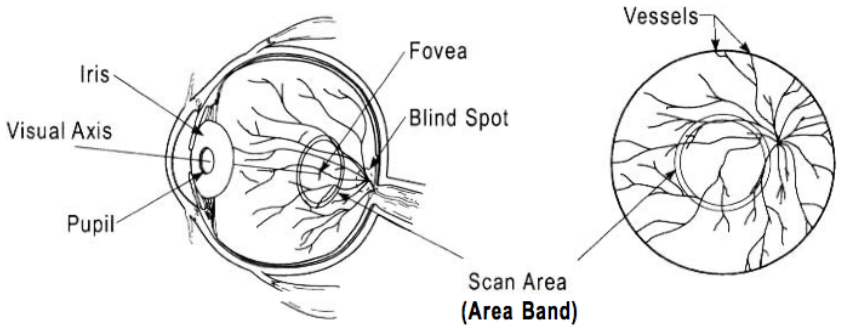


Figura 3.2.5: Struttura interna dell'occio e area scansionata per l'identificazione. Immagine tratta da [15].

con una fotocamera, possiamo pensare la retina come una pellicola: la luce filtra attraverso la pupilla e viene deflessa dal cristallino, che agisce come una lente, mettendo a fuoco l'immagine; infine la luce giunge sulla retina, in cui imprime l'immagine. L'informazione visiva sarà poi condotta al cervello attraverso il nervo ottico. Al di là del funzionamento come organo di senso, quello che ci interessa è la struttura della vascolarizzazione coroidale, ovvero la moltitudine di minuscoli vasi sanguigni che provengono dal nervo ottico ed irrorano tutta la retina.

La scansione delle retina consiste in una fotografia di una sua regione anulare; i parametri che regolano le dimensioni di tale corona circolare sono scelti in modo da ottenere un'elevata qualità delle immagini anche in occhi con una pupilla molto piccola. Successivamente la rilevazione deve essere tradotta in immagine e quindi digitalizzata; generalmente si utilizzano due standard per la rappresentazione della retina: la prima, anche in ordine cronologico, è una firma di 40 bytes di informazioni di contrasto attinte dal dominio della frequenza dell'immagine, calcolato mediante la Fast Fourier Transform; il secondo modo di rappresentare l'immagine occupa più memoria, ovvero 48 bytes, ma richiede un hardware di minore potenza, poiché l'informazione memorizzata si riferisce al dominio dello spazio, quindi non ha bisogno di calcolare la trasformata di Fourier.

Successiva alla scansione è la modifica dell'immagine al fine di migliorarla e renderla utile per il confronto con altre scansioni. Viene calcolato, innanzitutto, il rapporto tra l'intensità di ogni punto e l'intensità locale media, al fine di eliminare artefatti dovuti alla non uniformità dell'illuminazione. Questo procedimento è altresì utile a normalizzare le immagini rispetto a differenti grandezze della pupilla. Successivamente le operazioni compiute a livello di software sono due: correzione della fase e confronto. Per quanto riguarda la prima, è necessaria poiché ogni volta che il soggetto punta il suo occhio verso la fotocamera,

Figura 3.2.6: Risultato della scansione della retina.

la sua testa potrebbe essere leggermente ruotata, così sarà anche l'acquisizione della sua retina; il software di acquisizione deve correggere questa rotazione e lo fa massimizzando il coefficiente di correlazione tra l'immagine corrente ed una di riferimento. La fase di confronto si articola in tre passi successivi:

- viene eseguito il campionamento, in modo che l'immagine risulti della stessa dimensione dell'immagine da confrontare;
- l'immagine acquisita e quella con cui confrontarla vengono normalizzate in modo da avere il valore dell'RMS pari ad 1; l'RMS è l'acronimo di *root mean square* ed è definito come la deviazione standard dei valori d'intensità dell'immagine, ovvero

$$RMS = \sqrt{\frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M}(I_{ij} - \bar{I})^2} \tag{3.2.1}$$

  dove $\bar{I}$ è l'intensità media dell'immagine;
- si calcola l'indice di correlazione normalizzato tra le due immagini; questo varia da -1, ovvero "immagini completamente differenti", a $+1$, cioè "immagini identiche", quindi possiamo considerare attendibile una correlazione di almeno 0.7, in corrispondenza della quale il software indicherà che il confronto ha avuto esito positivo.

Se l'obiettivo della scansione è il riconoscimento di un individuo, piuttosto che la sua identificazione, allora non abbiamo un'immagine di riferimento, ma dobbiamo confrontare l'acquisizione con tutti i modelli immagazzinati nella banca dati che abbiamo a disposizione. Questo processo è considerevolmente più laborioso rispetto all'identificazione, poiché il coefficiente di correlazione va calcolato tra la nuova scansione e tutti i modelli. Per ovviare a questo problema sono stati ideati degli algoritmi in grado di sfruttare in modo ottimale anche hardware poco potenti; oggigiorno questi sistemi sono diventati obsoleti, poiché gli elaboratori che abbiamo a disposizione sono molto potenti ed una semplice parallelizzazione può bastare a rendere il riconoscimento sufficientemente rapido.

Ovviamente anche l'indicatore biometrico basato sulla scansione della retina ha i suoi problemi di sicurezza. La scansione, infatti, potrebbe essere contraffatta, anche se non è certo una cosa molto semplice da ottenere. Per sottoporre alla fotocamera la scansione di un altro occhio, anziché il proprio, occorre possedere tale scansione, quindi replicare la forma ed il funzionamento dell'occhio umano, in modo tale che il sistema di sicurezza non rilevi anomalie. In aggiunta ad un compito già non troppo semplice, alcuni apparecchi di identificazione, per difendersi da attacchi del genere, visualizzano sulla lente dei codici, che l'utente dovrebbe digitare successivamente; per oltrepassare la sicurezza offerta da tali sistemi occorre che l'occhio contraffatto sia in grado di leggere o comunque di riprodurre quanto visualizzato. Si può concludere questa trattazione affermando che i sistemi biometrici basati sulla scansione della retina, nonostante teoricamente attaccabili, come tutti i sistemi di sicurezza, presentano difficoltà pratiche sufficientemente elevate da ritenerli tra i più sicuri.

3.3 L'analisi manuale delle impronte digitali

Concentriamoci ora solamente sulle impronte digitali, argomento centrale di questo lavoro di tesi. Nel capitolo **??** analizzeremo alcuni tra i principali metodi automatici di riconoscimento che usano questo importante indicatore biometrico, mentre ora accenneremo all'analisi manuale. È infatti molto importante avere un'idea di come venisse affrontato il problema fino a qualche decennio fa, poiché da un lato permette di cogliere la semplificazione del lavoro apportata dall'intervento dell'automazione, dall'altra aiuta a comprendere il funzionamento degli algoritmi esposti nel capitolo **??**.

I motivi che ricorrono nelle impronte digitali, detti *pattern*, sono classificati in tre categorie, ognuna, a sua volta, suddivisa in varie sottoclassi:

- *archi*, in inglese *arch*, costituiscono il $5 - 15\%$ dell'impronta, suddivisi in
  - *archi piani* (*plain arch*),

  - *archi a tenda* (*tented arch*);
- *cappi* (*loop*), presenti per il 60 – 65% dell'immagine, suddivisi in
  - *ulnare*, se le linee si chiudono verso il dito mignolo,
  - *radiale*, se si chiudono verso il pollice;
- *spirali* (*whorl*), che occupano il 30 – 35% dell'immagine, divisi in
  - *piani*
  - *a doppio cappio* (*double loop*)
  - *central pocket loop* (che letteralmente significa cappio centrale tascabile)
  - *accidentali*

A seconda di come si combinano le increspature si possono formare differenti *dettagli galtoniani* (da Sir Francis Galton, un antropologo che studiò le impronte digitali nel XIX secolo), che vengono classificati a seconda del loro aspetto:

- forcina
- convergenza
- appendice
- biforcazione
- divergenza
- barretta
- isola
- puntino
- cresta corta
- cresta lunga
- principio (di una cresta)
- terminazione (di una cresta)

Si definisce *pattern area* quella parte dei pattern a cappio o a spirale che è oggetto di studio ed in cui sono presenti *core* e *delta*. Vengono chiamate *type lines* le due linee più interne ma tali che iniziano parallele, divergono e racchiudono, o tendono a racchiudere, tutta la pattern area. Il core è indicativamente al centro della pattern area, anche se per i pattern a cappio esistono regole ben precise per individuarlo (vedi la sezione 3.3.1); il delta è posizionato sempre su un'increspatura, il più vicino possibile al punto in cui le type lines divergono, coincidente con esso oppure di fronte. La Figura 3.3.1 può aiutare il lettore a visualizzare meglio questi concetti.

Nelle pagine seguenti definiremo e faremo chiarezza nella classificazione dei pattern data all'inizio di questa sezione, spiegando anche a che cosa corrispondono i dettagli galtoniani elencati.

3.3.1 Pattern a cappio

Un pattern a cappio può essere definito come quel motivo, creato dalle increspature, in cui una o più creste entrano da un lato dell'immagine, racchiudono, toccano o attraversano la linea immaginaria che passa per il delta e per il core, quindi tornano dalla stessa parte da cui sono partite. Un pattern a cappio deve sempre possedere le seguenti tre caratteristiche:

- un *sufficient recurve*,
- un delta,
- uno o più *ridge count* lungo il cappio.

Nel seguito stabiliremo delle regole per individuare ognuno di questi elementi.

Un *sufficient recurve*, letteralmente "curvatura sufficiente", ovvero quella parte di un'increspatura curva che si trova tra le sue spalle (vedi Figura 3.3.5); si dicono *spalle* della cresta i due punti in cui essa curva definitivamente verso il suo interno (vedi Figura 3.3.4). Poiché, come regola generale, una cresta, che tra le sue spalle abbia un escrescenza perpendicolare alla curvatura, non viene considerata nell'analisi, allora non può nemmeno essere considerata un sufficient recurve, nonostante soddisfi tutte le altre caratteristiche. Per chiarire le idee su questa regola si veda la Figura 3.3.3.

Un delta, ferma restando la definizione data alla pagina 96, deve essere necessariamente uno dei seguenti dettagli galtoniani:

- una biforcazione con l'apertura rivolta verso il core
- la terminazione di una cresta

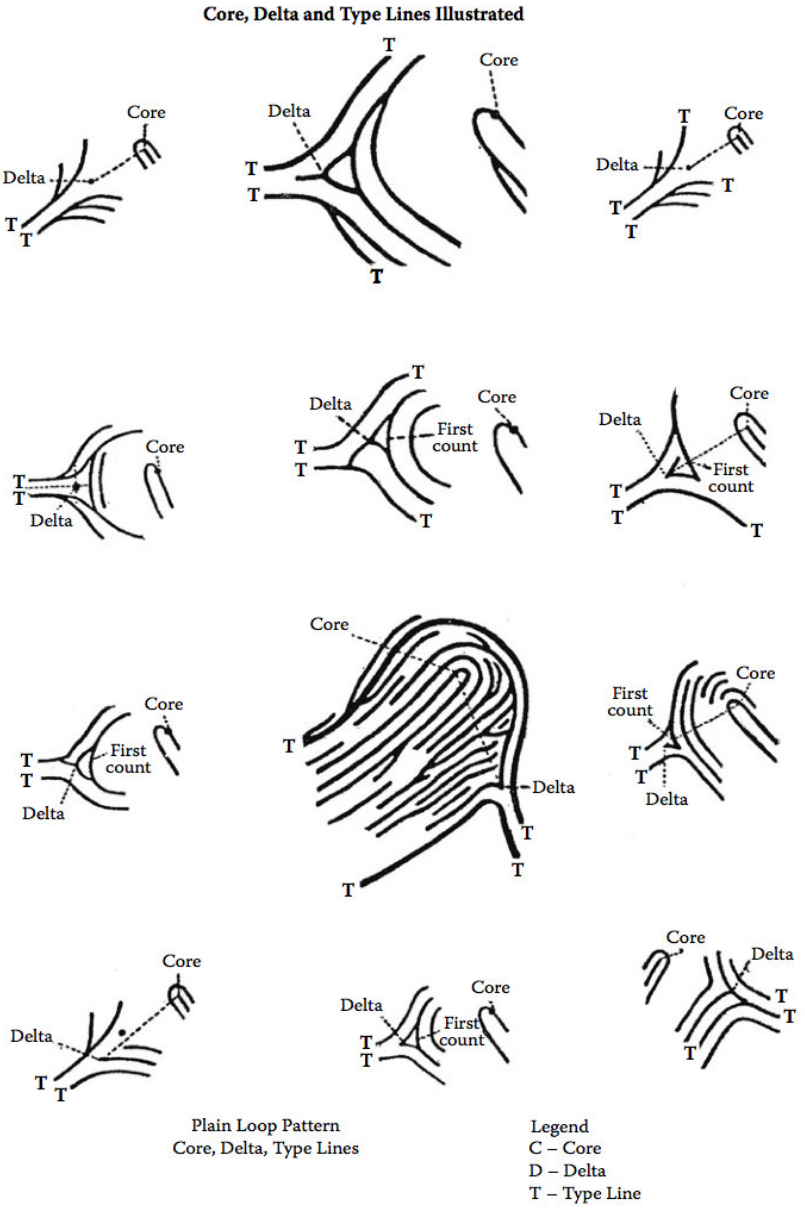


Figura 3.3.1: Pattern area di alcuni patter a cappio, in cui sono indicati core, delta e type lines. Immagine tratta da [13].

(a) (b) (c)

Figura 3.3.2: Tipici esempi di pattern a cappio. Immagini tratte da [13].

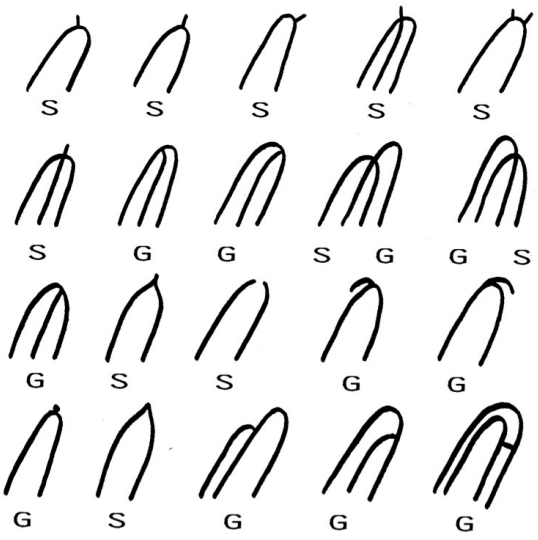


Figura 3.3.3: Nella figura sono rappresentati alcuni possibili creste con un'appendice tra le spalle. Sono segnate con S, dall'inglese *spoiled*, quelle che non vanno considerate per l'analisi e con G, *good*, quelle da non scartare, in accordo con la regola esposta a pagina 96. Immagine tratta da [9].

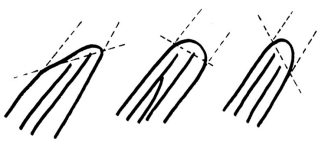


Figura 3.3.4: Tre esempi di pattern a cappio, in cui sono mostrati i prolungamenti delle creste laterali e la linea che congiunge le due spalle. Immagine tratta da [9].

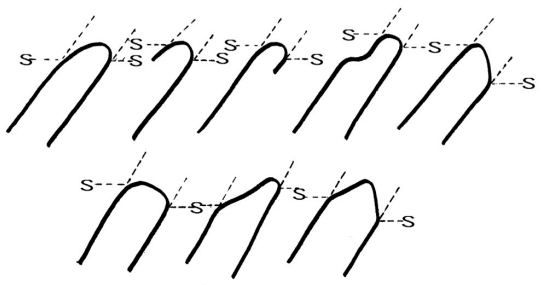


Figura 3.3.5: Nella figura sono mostrati alcuni sufficient recurve, in cui le spalle sono esplicitamente indicate. Immagine tratta da [9].

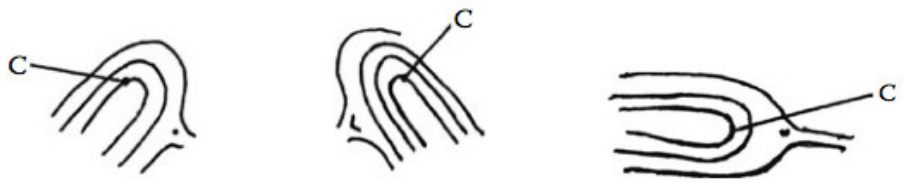
- un punto
- una cresta corta
- la convergenza di due creste
- un qualunque punto su un cappio che rispetti al definizione di pagina 96

Per guidare la scelta del delta intervengono alcune regole, che schematizziamo di seguito:

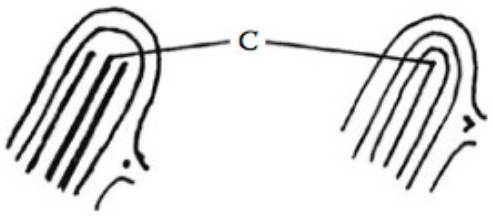
- se la scelta è tra un qualunque elemento ed una biforcazione, dobbiamo preferire quest'ultima;
- in caso di più di una possibilità nella scelta del delta, si preferisce l'elemento più vicino al core;
- un delta non può mai essere posizionato lungo una cresta che scorre tra le type lines verso il core, ma può trovarsi sulla sua terminazione in direzione di quest'ultimo;
- se la cresta è interamente contenuta nella pattern area, allora il delta può essere sulla sua terminazione vicina alla divergenza delle type lines;
- se la cresta entra nella pattern area da un punto sottostante alla divergenza, il delta si posiziona sulla terminazione dalla parte del core.

Anche per la localizzazione del core abbiamo delle linee guida:

- il core si trova sul sufficient recurve più interno, dentro di esso oppure sulle sue spalle;
- quando il sufficient recurve più interno non contiene terminazioni né barrette, il core si trova sulla spalla più distante dal delta; nel caso in cui le spalle ne fossero equidistanti, lo si posiziona tra di esse;



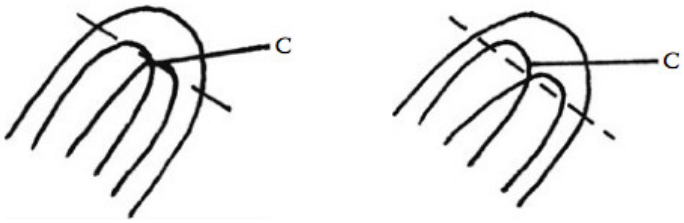
- se il sufficient recurve più interno contiene un numero dispari di barrette che terminano all'altezza delle spalle, allora il core si trova sulla terminazione della barretta centrale, a prescindere che essa tocchi o meno il recurve stesso;



- quando il sufficient recurve più interno contiene un numero pari di barrette, si posiziona il core sulla terminazione di una delle barrette centrali, in modo che sia il più lontano possibile dal delta;

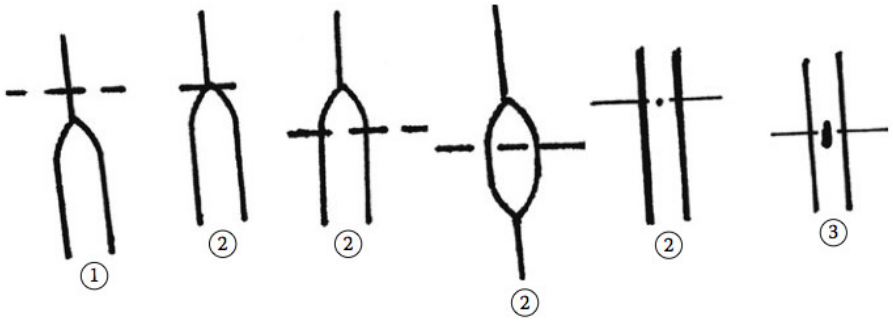
- se il sufficient recurve più interno ha un'appendice perpendicolare alla sua curvatura, come stabilito dalla regola di pagina 96, esso non va considerato e si passa a studiare quello appena più esterno;

- se all'interno della pattern area ci sono due cappi che si intersecano sopra l'altezza delle spalle, si posiziona il core all'intersezione, mentre se la sovrapposizione avviene al di sotto delle spalle, si predilige il cappio più lontano dal delta e si seguono le regole appena esposte;



Rimane ora da spiegare cosa si intende per *ridge count*; questo viene definito, in generale, come il numero di increspature che intervengono tra il core ed il delta. La procedura per eseguire il calcolo è leggermente più macchinosa, poiché prevede delle rigide regole da seguire. Innanzitutto consideriamo una linea immaginaria r che congiunge il core al delta, quindi contiamo le creste come segue:

- ogni linea che tocca r deve essere contata;
- il core ed il delta non si contano;
- se r attraversa una biforcazione allora si contano due linee;
- se r interseca le due sponde di un'isola, vengono contate entrambe;



- segmenti e punti sono contanti solamente se appaiono spessi quanto le creste vicine;
- uno spazio bianco deve intercorrere tra il delta e la prima cresta, altrimenti questa deve essere scartata;

- se il core si trova sulla terminazione di una cresta che tocca il sufficient recurve interno, allora il recurve viene considerato solamente se il delta si trova sotto alla retta tangente al recurve passante per il core; la seguente figura aiuta a comprendere meglio la regola, mostrando a sinistra il caso in cui viene contato e destra il caso opposto.

Abbiamo elencato tutti gli elementi che compongono un pattern a cappio, descrivendo come posizionarli; se uno di questi elementi manca allora il presunto cappio viene classificato arco a tenda (tented arch).

3.3.2 Pattern ad arco

(a) (b) (c)

Figura 3.3.6: Esempi di pattern ad arco piano. Immagini tratte da [13].

La prima tipologia di arco che definiamo è l'arco piano. Una conformazione di questo tipo si ha quanto le increspature dell'impronta digitale iniziano da un lato dell'immagine, ma poi fluiscono dell'altro lato, al contrario di come avveniva in un cappio, innalzandosi leggermente al centro. In un arco piano anche gli altri elementi, come punti, barrette, biforcazioni e terminazioni, seguono il profilo generale. Configurazioni di questo tipo sono anche dette assenza di pattern, nel senso che non hanno delta, né un reale core, né di conseguenza possiamo determinare il ridge count. Nella Figura 3.3.6 possiamo vedere dei tipici esempi di arco piano.

L'altra variante di arco è quello a tenda; come per l'arco piano, tutte le creste tendono a iniziare da un lato dell'immagine e terminare dall'altro, ma le increspature centrali non mantengono questa tendenza. Principalmente possiamo identificare tre classi di pattern che possono essere considerati archi a tenda:

- le creste centrali fluiscono formando un'ondina centrale, che risulta in un chiaro cambio di direzione di 90° o meno

- le creste centrali si ergono molto rapidamente al centro dell'immagine, ovvero formando un angolo maggiore di 45° col piano orizzontale

- se un pattern non ha le caratteristiche appena esposte, che ne farebbero direttamente un arco a tenda, può comunque essere classificato così se assomiglia ad un cappio ma non rispetta alcune delle regole esposte nella sottosezione 3.3.1.

3.3.3 Pattern a spirale

Un pattern a spirale è caratterizzato dalla presenza di due o più delta, ognuno con il proprio sufficient recurve di fronte. A seconda di specifiche caratteristiche si può classificare più dettagliatamente come spirale piana, a doppio cappio, *central pocket loop whorl*, accidentale.

(a) (b) (c)

Figura 3.3.7: Esempi di pattern a spirale piana. Immagini tratte da [13].

Spirale piana Una spirale piana possiede due delta ed almeno una cresta che fa un percorso completo, che può essere a spirale, ovale, una circonferenza oppure una sua qualunque variante. Poche sono le altre regole di cui tenere conto:

- le due type lines dei delta non devono per forza essere sulla stessa cresta;
- una linea immaginaria tra i due delta deve per forza toccare o attraversare le linee più interne;

- una cresta, con un'appendice perpendicolare alla tangente in un suo punto, non può essere considerata parte di un circuito, poiché per noi essa è priva di valore, come da regola.

La Figura 3.3.7 ci mostra degli esempi di spirali piane.

(a) (b) (c)

Figura 3.3.8: Esempi di central pocket loop whorl. Immagini tratte da [13].

Central pocket loop whorl Questa categoria di pattern a spirale mantiene alcune caratteristiche delle spirali piane, pur condividendone altre con i cappi. In particolare:

- nonostante assomigli ad un cappio, nella parte centrale, attorno al core, ha una o più piccole spirali; le increspature più interne, dopo aver curvato una prima volta, come a formare un cappio, si ripiegano una seconda volta per dare vita ad una spirale;
- sono presenti due delta, uno ai margini della pattern area, l'altro interno, appena al di sotto della cresta centrale;
- la linea immaginaria tra i delta non deve assolutamente toccare o attraversare le creste più interne.

In Figura 3.3.8 si possono osservare alcuni esempi di central pocket loop whorl.

Spirali a doppio cappio Le spirali a doppio cappio, come suggerisce il nome, si possono identificare poiché contengono due motivi a forma di cappio; ognuno di essi deve possedere un delta e due spalle. I due cappi possono anche

(a) (b) (c)

Figura 3.3.9: Esempi di spirali a doppio cappio. Immagini tratte da [13].

essere connessi da una cresta, purché essa non fuoriesca perpendicolarmente ad un cappio, tra le sue spalle, poiché in quel caso lo priverebbe di valore. Nonostante sono previste due formazioni a cappio, i requisiti per il ridge count dei cappi possono anche non essere soddisfatti, così come i due cappi possono avere grandezze differenti.

Spirali accidentali Una spirali può essere di tipo accidentale se l'immagine è composta da due motivi differenti, ad eccezione dell'arco piano, e due o più delta, oppure ci sono molte delle caratteristiche di due classi di pattern, ma l'impronta non è conforme a nessuna delle due definizioni.

Possiamo capire meglio cosa sono le spirali accidentali con degli esempi:

- cappio ed arco a tenda (solamente se il cappio sovrasta l'arco, altrimenti rientra nella classe degli archi a tenda)

- cappio e spirale piana

- cappio e central pocket loop whorl

- cappio e spirale a doppio cappio

Ridge Tracing e Ridge Count Anche per i pattern a spirale esistono delle tecniche per il ridge tracing e per il ridge count. Vediamo innanzitutto come eseguire il ridge tracing:

- individuare i delta, considerando che se sono più di due, verranno presi in esame solo quelli all'estrema sinistra e destra, scartando gli altri;
- iniziando dal delta di sinistra, seguire una cresta dirigendosi verso l'esterno del pattern in cui finisce tale cresta e continuare fino al raggiungimento del punto più vicino oppure opposto al delta di destra (opposto rispetto al pattern cui il delta destro si riferisce);

- se c'è una biforcazione va seguito il ramo inferiore;
- si conta il numero di creste che occorrono tra la linea tracciata e il delta destro;
- in base ai valori ottenuti si classificano le spirali come:

  I : dall'inglese *inner*, se ci sono tre o più increspature all'interno del delta destro;

  M : dall'inglese *meet*, se il valore è inferiore a tre, all'interno o all'esterno del delta;

  O : da *outer*, se il numero è maggiore o uguale a tre all'esterno del delta destro.

(a) **I** (b) **O** (c) **I**

Figura 3.3.10: Esempi di ridge tracing e successiva classificazione sulla base del numero di creste tra il delta destro e la linea tracciata. Immagini tratte da [13].

Per eseguire il ridge count nelle spirali si utilizzano le stesse tecniche viste per i cappi, solamente che c'è da convenire sul delta da scegliere. Più nello specifico, si contano le creste dal delta sinistro al core nella mano destra, mentre si utilizza il delta destro nella mano sinistra; se sono presenti più di due delta si utilizza quello dal lato giusto e più vicino al core. Non ci dilunghiamo ulteriormente sull'argomento, poiché il ridge count delle spirali è poco usato nell'analisi automatica delle impronte, solo alcuni codici per la classificazione delle impronte digitali lo richiedono.

3.3.4 Cenni alla classificazione delle impronte

Come ogni metodo di classificazione, l'obiettivo è stabilire un insieme di regole, formare un protocollo da utilizzare per poter ricercare e confrontare agevolmente le impronte digitali.

Il metodo cui accenneremo è quello denominato *Henry con estensione FBI*, uno dei primi criteri utilizzati ed anche uno dei più conosciuti. Il nome deriva dall'ideatore, Sir Edward Henry, che in origine suddivise la procedura in quattro parti: primaria, secondaria, sotto-secondaria e finale. In seguito il sistema fu rivisitato dall'FBI, Federal Bureau of Investigation, che incluse un'estensione di due parti aggiuntive; in alcuni casi viene utilizzata anche una settima parte, corrispondente alla seconda sotto-secondaria. Questo metodo viene oggigiorno chiamato brevemente "metodo manuale", in opposizione ai metodi automatici che si sono sviluppati ed affermati negli ultimi anni.

I componenti del sistema manuale sono:

1. *Key*: il ridge count del primo cappio che appare sul foglio delle impronte, escludendo i mignoli.
2. *Major*: il ridge count oppure il ridge tracing dei pollici.
3. *Primary*: il valore numerico di ogni dito che contiene una spirale.
4. *Secondary*: lettera maiuscola che indica il pattern degli indici.
5. *Subsecondary*: ridge count e tracing di cappi e spirali delle dita indice, medio e anulare.
6. *Final*: ridge count dei cappi che appaiono sui mignoli.

Un esempio della forma che assume la classificazione di un'impronta digitale usando il metodo di Henry è mostrato nella Tabella 3.1.

Key	Major	Primary	Secondary	Subsecondary	Final
4	O	5	U	IOI	10
	I	17	U	IOI	

Tabella 3.1: Esempio di classificazione tramite il metodo di Henry ed estensione FBI. Tabella tratta da [13].

In Figura 3.3.11 si può vedere il tipico foglio per acquisire le impronte digitali, quindi schedare un criminale; è da esso che si estraggono successivamente le informazioni necessarie per stilare la tabella 3.1.

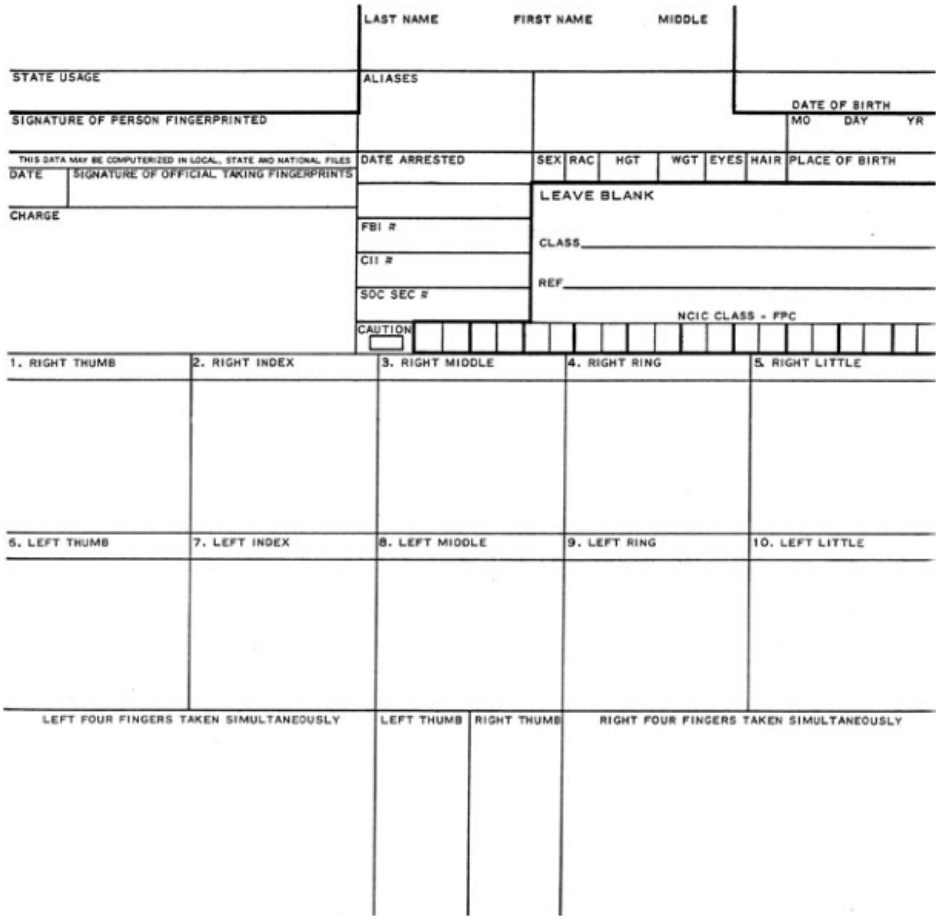


Figura 3.3.11: Esempio del foglio che serve a schedare un criminale. Immagine tratta da [13].

Per approfondire il metodo di classificazione è possibile consultare il libro [13], che ne offre una trattazione abbastanza dettagliata. In questo lavoro non possiamo dilungarci ulteriormente, poiché il campo della classificazione manuale delle impronte è molto vasto e necessita anche di molta pratica per essere compreso fino in fondo.

Capitolo 4

Metodi automatici per le impronte digitali

4.1 Introduzione

Nel capitolo 3 abbiamo dato dei cenni all'approccio manuale per il riconoscimento dell'individuo tramite impronte digitali; oltre tutto quello che abbiamo visto, negli anni ci sono stati molti miglioramenti, volti ad aumentare la velocità di classificazione e confronto. Mentre tutto questo poteva essere sufficiente qualche decade fa, ormai si ha a che fare con database molto voluminosi e con la necessità di effettuare confronti sempre più rapidi; inoltre, messa da parte l'euforia iniziale, le agenzie investigative cominciarono a farsi i conti in tasca: la maggior parte delle impronte digitali veniva classificata nelle stesse categorie, secondo il sistema proposto da Henry, quindi l'operazione non era di molto aiuto; le procedure per l'addestramento alla classificazione era molto lunghe e di conseguenza anche costose per le agenzie investigative; il lavoro che sta dietro ad ogni singolo confronto tra impronte è stancante per la vista e per la mente, è noioso e richiede costante attenzione, ovvero ha le migliori caratteristiche per condurre in errore l'operatore. Quando la tecnologia fu matura, ovvero una quarantina di anni fa, si cercò di sviluppare sistemi automatici di identificazione, che ponessero rimedio alla situazione appena descritta. Il passare degli anni, l'aumento della richiesta di metodi di autenticazione ed identificazione basati su indicatori biometrici e l'abbassamento dei prezzi dei macchinari utili all'acquisizione dei dati portarono ad un copioso utilizzo delle tecniche automatiche sviluppate in ambito investigativo e in altri contesti, come il mercato elettronico.[1]

Come abbiamo evidenziato in precedenza, quando si sottopone un campione ad un sistema ideale basato su indicatori biometrici, questo è in grado di indi-

[1] Il materiale esposto in questo capitolo è in gran parte tratto da [22].

viduarne correttamente la fonte, senza margine di errore. Nella realtà questo livello di precisione non viene mai raggiunto a causa di errori nell'acquisizione, nell'elaborazione dei dati oppure persino nello specifico tratto biometrico, che potrebbe avere un grado di similarità troppo elevato. Non potendo raggiungere la perfezione, l'auspicio per un sistema di sicurezza fondato sulla biometria è avere una rappresentazione invariante e sufficientemente realistica di un indicatore a partire da pochi campioni, per riuscire infine a discernere il tratto per un numero di utenti molto grande.

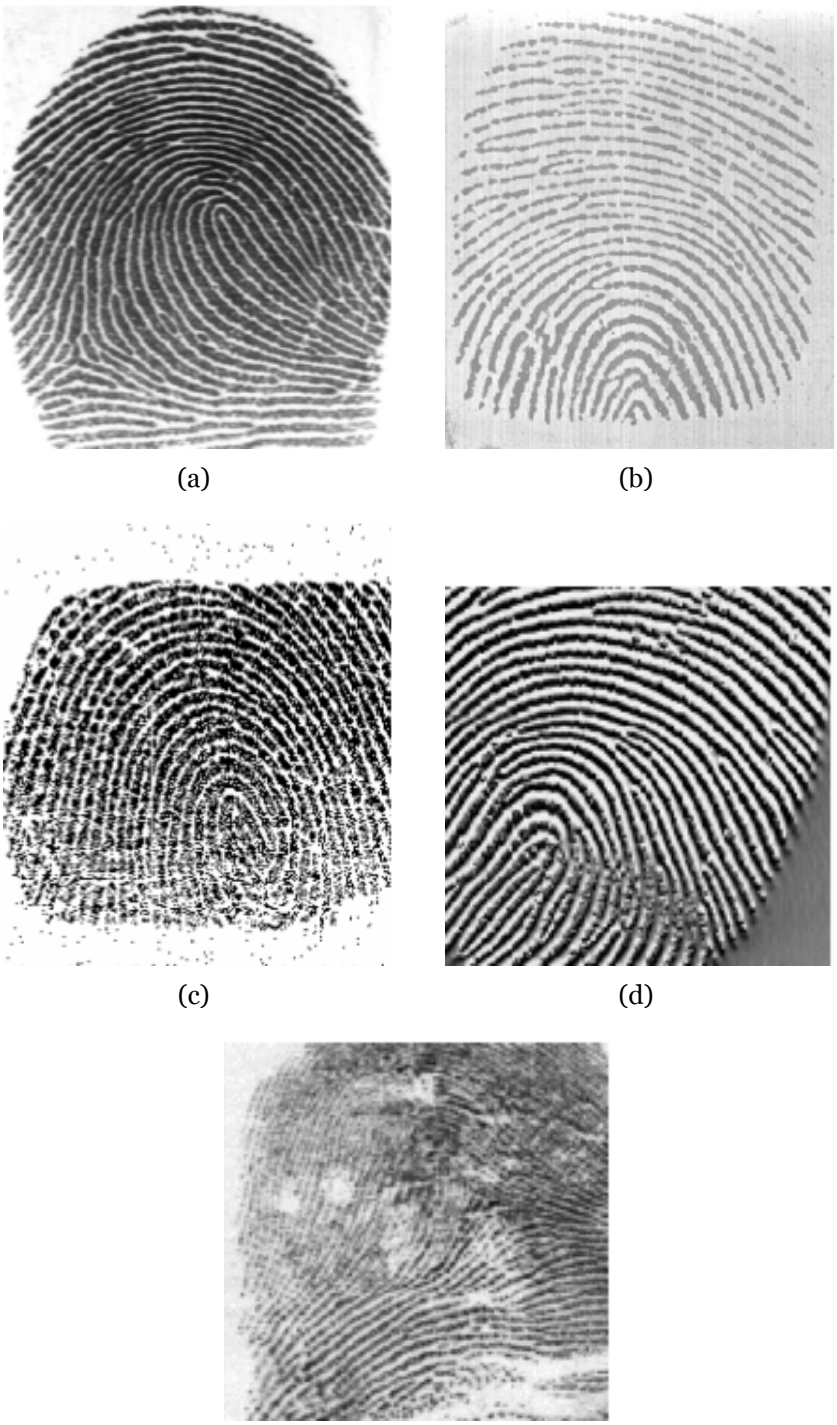
4.2 Studio dei metodi esistenti

4.2.1 Acquisizione

A seconda del modo con cui vengono acquisite, le immagini delle impronte digitali possono essere classificate come *off-line* oppure *live-scan*. Un'acquisizione off-line si ottiene generalmente ponendo dell'inchiostro direttamente sulla pelle dei polpastrelli, quindi si preme il dito sopra un foglio di carta; questo viene poi digitalizzato tramite scansione utilizzando uno scanner oppure una macchina fotografica di alta qualità. In campo forense le acquisizioni offline sono le più frequenti, sia perché le impronte registrate da molto tempo sono tutte impresse con inchiostro su carta, sia perché le impronte prese sulle scene del crimine vengono prelevate da superfici che sono state a contatto con la pelle del criminale. Quest'ultimo tipo di acquisizioni prende il nome di impronte latenti e, come si può capire, sono le più frequenti negli archivi giudiziari, ma sono altresì di qualità molto scarsa; esse infatti non sono lasciate intenzionalmente, ma sono frutto del deposito delle sostanze oleose secrete dalla nostra pelle sulle varie superfici. La loro acquisizione deve essere preceduta da una serie di processi chimici che ne riescano ad esaltare la struttura, altrimenti risultano di poca utilità.

Contrariamente, un'acquisizione live-scan, ovvero "scansione in tempo reale", è generata da sensori in grado di digitalizzare le increspature dell'epidermide direttamente, al contatto con la pelle. La parte più importante di un live-scanner di impronte digitali è il sensore, che è il componente in cui l'immagine si viene a formare; i sensori esistenti ricadono in una delle seguenti categorie: scanner ottici, a stato solido oppure ad ultrasuoni.

- Gli scanner ottici generano l'immagine sfruttando la riflessione della luce da parte della pelle (come i sensori di tipo *FTIR - Frustrated Total Internal Reflection*), come nel caso delle classiche fotocamere, oppure la luce emanata da particolari componenti che interagiscono direttamente con la pelle (sensori elettro-ottici).



(a) (b)

(c) (d)

(e)

Figura 4.2.1: Acquisizioni di impronte digitali tramite differenti sensori. (a) Live-scan di un sensore ottico basato su FTIR. (b) Live-scan di uno scanner capacitivo. (c) Live-scan di uno scanner piezoelettrico. (d) Live-scan di un sensore termico. (e) Acquisizione offline tramite inchiostro.

- I sensori a stato solido, nonostante introdotti già dagli anni '80, furono utilizzati solamente a partire dalla decade successiva. Furono creati con lo scopo di porre rimedio alla grandezza a al costo eccessivo degli altri tipi di sensori presenti sul mercato, problemi che sembravano impedire la diffusione di massa del riconoscimento tramite impronte digitali. Questo tipo di sensori si caratterizza per essere sostanzialmente una matrice di pixel, dove ogni pixel è un sensore miniaturizzato; in base alla tecnologia utilizzata per l'interazione tra la pelle ed i nano-sensori, si possono definire quattro categorie: capacitivo, termico, a campo elettrico, piezoelettrico.

- I sensori ad ultrasuoni sfruttano la stessa tecnologia utilizzata, ad esempio, nell'ecografia. All'interno del sensore c'è un trasmettitore di onde acustiche ad alta frequenza (ultrasuoni), che emana verso il dito dell'utente; queste onde vengono riflesse e tornano a colpire il sensore, in cui un ricevitore le registra. In base alle caratteristiche delle onde riflesse si può conoscere esattamente la distanza del punto in cui la riflessione ha avuto origine; replicando questo procedimento in tutta l'area di scansione si ottiene l'immagine di un'impronta digitale.

I principali parametri che caratterizzano le acquisizioni di un'impronta digitale sono: risoluzione, area, numero di pixel, accuratezza geometrica, contrasto e distorsione geometrica. Vediamo più in dettaglio la terminologia usata.

- *Risoluzione*: indica il numero di pixel per pollice (*pixel per inch, ppi*) oppure di punti (*dots per inch, dpi*). Una risoluzione di $500\,dpi$ è il minimo consentito dall'FBI, nonostante possono risultare utili anche immagini a $250\,dpi$.

- *Area*: è la misura della superficie acquisita dal sensore utilizzato.

- *Numero di pixel*: è il numero di punti effettivamente contenuto nell'immagine; può essere banalmente ricavato dall'area e dalla risoluzione, infatti se abbiamo un sensore che lavora a $R\,dpi$ in un area di $w \times h\,inch^2$, allora l'immagine ottenuta avrà area di $R \cdot w \times R \cdot h$ punti.

- *Distorsione geometrica*: è il valore assoluto della differenza tra la distanza reale tra due punti e quella misurata nell'immagine acquisita.

- *Accuratezza geometrica*: è determinata dal massimo valore assunto dalla distorsione geometrica.

- *Contrasto*: è il rapporto tra l'intensità massima e la minima.

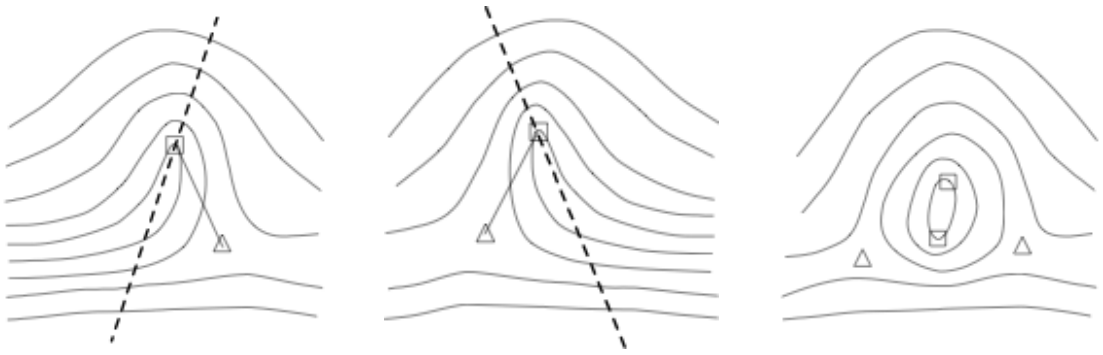
Al fine di massimizzare la compatibilità tra le acquisizioni di impronte digitali e per assicurare una buona qualità a prescindere dal sistema di riconoscimento automatico utilizzato, il Criminal Justice Information Services(CJIS) degli Stati Uniti ha stabilito una serie di specifiche che occorre rispettare in campo forense. La maggior parte delle applicazioni non destinate al campo giudiziario non rispetta i requisiti richiesti, infatti l'obiettivo degli strumenti di acquisizione commerciali non è la completa affidabilità, ma la rapidità di esecuzione. Nella Figura 4.2.1 possiamo vedere le acquisizioni ottenute utilizzando alcuni dei sensori disponibili sul mercato.

Almeno per quanto riguarda il campo forense può sembrare ovvio utilizzare i sensori che diano immagini con la massima risoluzione; questa idea cade subito se si pensa che l'intento è immagazzinare il maggiore numero di impronte possibile, idealmente quelle di tutti gli esseri umani del pianeta. Nonostante siamo ben lungi dal completare un tale obiettivo, esistono database sufficientemente ampi da avere problemi di memoria, se si utilizzassero acquisizioni con la massima risoluzione disponibile. Facciamo un rapido calcolo: la banca dati dell'FBI conta circa 200 milioni di impronte, che, con una risoluzione di 500 dpi, occupano 10 MB l'una; con una semplice moltiplicazione vediamo che l'intero archivio occuperebbe 2 000 TB. Per ridurre il problema dello spazio si sta cercando di utilizzare algoritmi di compressione delle immagini, come il JPEG-2000, con un grado di compressione di $10 : 1$ e abbassamento di qualità visivamente impercettibile, e WSQ, un metodo altamente prestante, basato sulla teoria delle wavelets, in cui i segnali vengono approssimati da forme d'onde di lunghezza finita.

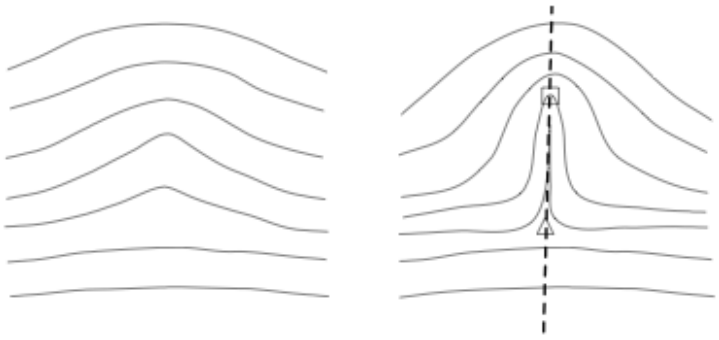
4.2.2 Rappresentazione

Prima di creare un metodo automatico che utilizzi impronte digitali, occorre definirne la rappresentazione; l'intensità dei pixel può variare tra un'acquisizione e la successiva, quindi utilizzare l'intera immagine per il confronto, generalmente non è consigliabile. Al contrario è necessario determinare le caratteristiche salienti dell'immagine, che rimangano invarianti tra campioni provenienti dallo stesso individuo, ma che riescano a discriminare acquisizioni di due diverse identità. In termini più tecnici dobbiamo trovare uno spazio di misura in cui inserire le immagini delle impronte digitali acquisite; la proprietà fondamentale che deve avere è che campioni provenienti dallo stesso individuo formino cluster compatti (bassa variabilità intra-classe), mentre campioni da individui differenti diano origine a raggruppamenti ben separati (alta variabilità inter-classe). Un tratto molto importante che deve avere la rappresentazione è che sia facilmente estraibile dall'acquisizione dell'impronta digitale, non ingombrante in termini

di memoria, utile per il successivo confronto con altre impronte.



(a) Cappio sinistro (b) Cappio destro (c) Spirale



(d) Arco piano (e) Arco a tenda

Figura 4.2.2: Struttura delle impronte digitali al livello più grossolano. I quadrati indicano i punti di massima curvatura dei cappi, i triangoli denotano la presenza di un delta. Immagine tratta da [22].

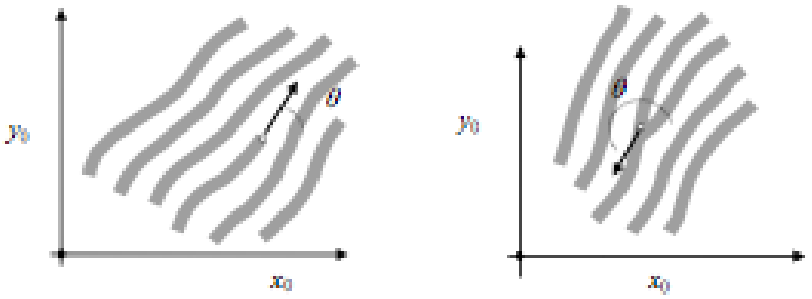
La struttura dell'impronta digitale, se analizzata a vari livelli esibisce caratteristiche di vario tipo.

- *Livello globale*: si osservano i pattern creati dalle creste, simili a quelli di Figura 4.2.2. Si possono estrarre dei punti critici, chiamati *loop* e *delta* e rispettivamente indicati con un quadrato e un triangolo nella figura. Altre informazioni che si possono ricavare sono, ad esempio, la forma generale dell'impronta, l'immagine orientazione (ogni punto delinea l'orientazione della cresta in quel punto), oppure l'impronta in frequenza (contenente l'immagine nel dominio di Fourier), oppure il *core*, molto utile come punto di riferimento per allineare due diverse scansioni. Punti critici e forma generale delle creste sono utili per la classificazione o l'indicizzazione dell'impronta, ma non sono adatti al confronto.

- *Livello locale*: sono state individuate un totale di 150 diverse caratteristiche locali, chiamate *minutiae*, ovvero dettagli minuscoli [28]. Tutte queste caratteristiche non sono distribuite in maniera uniforme, la maggior parte di esse sono dovute alla cattiva qualità dell'impronta o della sua acquisizione, quindi è raramente visibile. Le caratteristiche più ricorrenti, legate alle discontinuità delle increspature, sono sette e sono schematizzate in Figura 4.2.4. Ogni minutia può essere descritta da due coordinate spaziali e una angolare, ovvero l'orientazione della cresta di provenienza rispetto all'asse orizzontale dell'immagine (vedi Figura 4.2.3); è stato studiato che sono sufficienti poche corrispondenze di minutiae per poter asserire con sicurezza la paternità dell'impronta. Bisogna fare caso al fatto che la posizione delle minutiae in un'impronta, rimangono anche nel suo negativo, pur cambiando tipologia; va in questo caso evidenziata la dualità tra terminazione e biforcazione, che si scambiano i ruoli passando al negativo. Generalmente queste minutiae sono caratteristiche abbastanza stabili e robuste rispetto alla qualità dell'immagine; nonostante questo, nelle acquisizioni di qualità molto bassa, non riescono a rendere affidabile l'algoritmo di riconoscimento automatico.

- *Livello ultra-fine*: ad un livello ancora più fine di quello locale, possiamo osservare i dettagli presenti proprio sulle increspature, tra cui i pori sudoripari (vedi Figura 4.2.5). Questi sono molto importanti per il riconoscimento, infatti la loro posizione e la loro forma ci permettono di distinguere molto bene due individui differenti. Si pensi che i pori hanno una dimensione che varia da 60 a $250\mu m$ e che in una cresta di un centimetro ce ne sono tra i 9 e i 18; è stato provato che sono sufficienti da 20 a 40 pori corrispondenti per determinare l'identità di una persona. Il problema è che tali dettagli possono essere osservati solamente in immagini sufficientemente buone, acquisite ad una risoluzione di $1000\,dpi$, ovvero una qualità difficilmente raggiungibile da apparecchiature destinate all'ambito forense.

Prima di introdurre alcuni metodi di rappresentazione delle impronte digitali, cerchiamo di fare chiarezza sulla notazione utilizzata. Un'impronta per noi sarà sempre rappresentata da una superficie bidimensionale; denotiamo con I l'immagine dell'impronta in scala di g grigi, quindi con $I(x, y)$ il livello di grigio del punto (x, y) in I. Sia $z = S(x, y) = g - 1 - I(x, y)$ la superficie discreta che corrisponde all'immagine I. In questo modo abbiamo associato i pixel scuri con le creste della superficie, i punti chiari di I con le valli (Figura 4.2.6).



(a) (b)

Figura 4.2.3: (a) Terminazione: (x_0, y_0) sono le coordinate della minutia, θ è l'angolo che la tangente ad essa forma con l'asse orizzontale. (b) Biforcazione: (x_0, y_0) sono ancora le coordinate della minutia, ma θ ora è definito come l'angolo della corrispondente terminazione nel negativo dell'immagine.

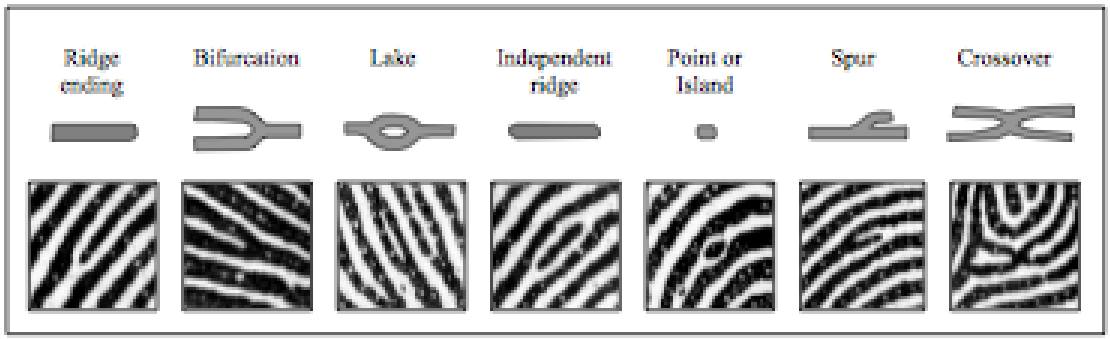


Figura 4.2.4: I sette tipi di minutiae più comuni. Immagine tratta da [22].

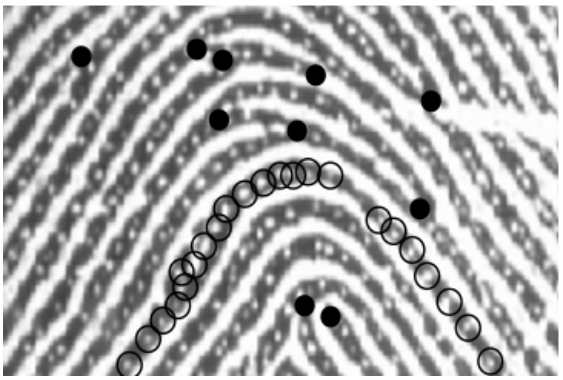


Figura 4.2.5: Sono evidenziate le minutiae di una porzione di un'impronta digitale con dei cerchi neri pieni; i pori sudoripari di una sola cresta sono invece evidenziati con dei cerchi neri vuoti. Immagine tratta da [22].

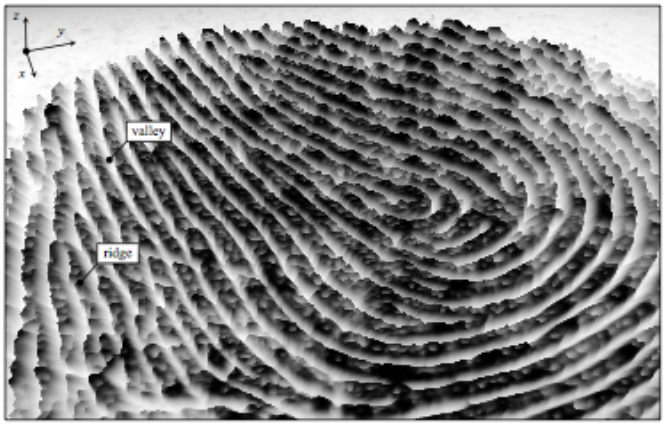


Figura 4.2.6: Una superficie S che rappresenta una parte di un'impronta digitale. Immagine tratta da [22].

4.2.2.1 Orientazione locale

L'orientazione locale di una cresta nel punto (x, y) è l'angolo θ_{xy} che la cresta forma con l'asse orizzontale. Poiché le increspature di un'impronta digitale non sono orientate, l'angolo θ_{xy} si può considerare nell'intervallo $[0°, 180°[$. Invece che calcolare l'orientazione per ogni singolo punto, molto spesso si preferisce velocizzare la procedura, calcolandola in corrispondenza di posizioni discrete. Si viene quindi a creare la cosiddetta *immagine orientazione*, ovvero una matrice D i cui elementi costituiscono l'orientazione locale delle creste di un'impronta. Ogni elemento θ_{ij} di questa matrice corrisponde al nodo (i, j) di una griglia rettangolare sovrapposta all'immagine; θ_{ij} è quindi centrato in (x_i, y_j) e ne rappresenta l'orientazione media dei punti dell'intorno. Spesso, accanto all'immagine orientazione, si può trovare un valore addizionale r_{ij} associato ad ogni θ_{ij}, che ne denota l'affidabilità: assumerà valori bassi per zone molto rumorose ed alti per regioni in buono stato.

Approcci basati su gradiente Il più semplice e naturale approccio per estrarre l'orientazione locale è basato sul calcolo del gradiente dell'immagine. Abbiamo già visto nel capitolo 2 come si definisce l'operatore gradiente nel caso discreto; anche in questo caso lo indichiamo con

$$\nabla I(x, y) = \begin{pmatrix} \dfrac{\partial}{\partial x} I(x, y) \\ \\ \dfrac{\partial}{\partial y} I(x, y) \end{pmatrix} \tag{4.2.1}$$

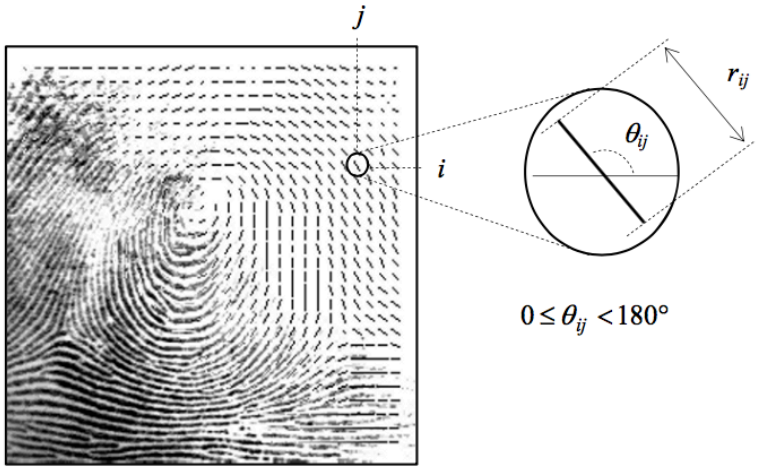


Figura 4.2.7: Un'impronta digitale che svanisce gradualmente, facendo luogo alla corrispondente immagine orientazione, calcolata su una griglia rettangolare 16×16. Ogni elemento denota l'orientazione locale delle creste, per cui è stato rappresentato con un freccia; la lunghezza della freccia è proporzionale alla sua affidabilità.

Già sappiamo che la direzione del gradiente indica dove si ha il massimo cambio di intensità; l'angolo θ_{xy}, che indica l'orientazione della cresta nel punto (x, y) si può pensare ortogonale al gradiente calcolato in (x, y). Si possono quindi utilizzare le tecniche discusse nel capitolo 2 per calcolare il gradiente in ogni punto dell'immagine con estrema facilità. Questo procedimento, nonostante semplice ed efficiente, ha i suoi problemi. Innanzitutto gli operatori di Sobel o suoi simili danno dei problemi dovuti alla non linearità e discontinuità attorno a 90°; un singola stima dell'orientazione tramite il gradiente, è molto soggetta al rumore dell'immagine, poiché riflette la struttura cresta-valle ad una scala troppo fine. Pensare di mediare l'orientazione, per conferirle robustezza al rumore, non è pensabile a causa della periodicità degli angoli: la media tra 5° e 175° è 0°, non 90°; la media tra 0° e 90° non è ben definita, perché potrebbe essere sia 45° che 135°.

Kass e Witkin [18] hanno proposto una soluzione tanto semplice quanto elegante al problema, che consente di fare la media delle stime ottenute col gradiente. L'idea di base è duplicare gli angoli, quindi mappare ogni orientazione con il vettore

$$\mathbf{d} = [r\cos(2\theta), r\sin(2\theta)] \tag{4.2.2}$$

dove l'angolo 2θ serve a tener conto della periodicità degli angoli, mentre r è proporzionale al quadrato della norma del gradiente, ovvero $\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2$. Se facciamo la media delle orientazioni in un intorno W di dimensione $n \times n$, otteniamo

una stima più robusta di **d**, ovvero:

$$\overline{\mathbf{d}} = \left[\frac{1}{n^2} \sum_{W} r\cos(2\theta), \frac{1}{n^2} \sum_{W} r\sin(2\theta) \right] \tag{4.2.3}$$

In questo caso se facciamo la media tra due orientazioni perpendicolari utilizzando la (4.2.3), allora ci troveremo a sommare vettori opposti: il risultato sarà il vettore nullo. In questo modo si risolve il problema per la somma di angoli perpendicolari, ma possiamo notare che viene risolto anche il problema delle discontinuità: ora mediare 5° con 175° equivale a mediare 10° con $350° = -10°$, ovvero da come risultato 0°.

Basandosi sulla stessa idea, Ratha, Chen e Jain [30] stimano la corretta orientazione θ_{ij} in un punto (x_i, y_j) combinando molteplici stime del gradiente in una finestra W di dimensione 17×17 centrata proprio in (x_i, y_j):

$$\begin{aligned}
\theta_{ij} &= 90° + \frac{1}{2}\text{atan2}\,(2G_{xy}, G_{xx} - G_{yy}) \\
G_{xy} &= \sum_{h=-8}^{8}\sum_{k=-8}^{8} \frac{\partial I}{\partial x}(x_i + h, y_j + k) \cdot \frac{\partial I}{\partial y}(x_i + h, y_j + k) \\
G_{xx} &= \sum_{h=-8}^{8}\sum_{k=-8}^{8} \left[\frac{\partial I}{\partial x}(x_i + h, y_j + k)\right]^2 \\
G_{yy} &= \sum_{h=-8}^{8}\sum_{k=-8}^{8} \left[\frac{\partial I}{\partial x}(x_i + h, y_j + k)\right]^2
\end{aligned} \tag{4.2.4}$$

dove $\frac{\partial I}{\partial x}$ e $\frac{\partial I}{\partial y}$ sono rispettivamente le componenti x e y del gradiente, calcolati con una maschera di Sobel 3×3, mentre atan2 è la funzione che calcola arcotangente tenendo conto dei segni dei suoi argomenti per determinare il quadrante di appartenenza. Bazen e Gerez [1] hanno dimostrato che questo metodo è matematicamente equivalente all'analisi delle componenti principali della matrice di autocorrelazione dei vettori gradiente. Altro approccio, che conduce comunque allo stesso risultato, è stato indipendentemente intrapreso da Donahue e Rokhlin [8], utilizzando la minimizzazione con il metodo dei minimi quadrati per calcolare la media delle orientazioni.

Come espresso dagli autori citati, l'affidabilità r della stima θ, si può derivare dalla concordanza dei vettori orientazione nella finestra W; infatti, a causa della continuità delle increspature dell'impronta, cambi troppo rapidi di orientazione spesso denotano un stima errata. Per esempio, Kass e Witkin [18] definiscono

l'affidabilità come

$$r = \frac{\left|\sum_{W} \mathbf{d}\right|}{\sum_{W} |\mathbf{d}|} \tag{4.2.5}$$

La (4.2.5) è uno scalare con valori in $[0, 1]$; assume il valore massimo se le orientazioni sono tutte concordi, mentre assume il minimo se esse sono tutte opposte. Se misuriamo l'affidabilità usando la (4.2.5) per il metodo esposto nella (4.2.4), allora otteniamo

$$r_{ij} = \frac{\sqrt{\left(G_{xx} - G_{yy}\right)^2 + 4G_{xy}^2}}{G_{xx} + G_{yy}} \tag{4.2.6}$$

Il problema principale degli approcci basati sul gradiente si ha nelle zone in cui questo ha valori molto vicini allo zero, ovvero in cima alle creste ed in fondo alle valli; infatti in queste regioni i valori molto bassi del gradiente lo rendono molto sensibile al rumore. Molti autori raccomandano di andare a scrutare anche le derivate di ordine successivo, ma se si guardano solo le derivate seconde, ad esempio, si sposta il problema sui punti di flesso. Una soluzione potrebbe essere il metodo esposto da Da Costa e collaboratori [6], che hanno creato un algoritmo che in ogni regione stabiliscono quale operatore utilizzare in base alle proprietà locali dell'immagine.

Stima dell'orientazione nel dominio della frequenza Il metodo proposto da Kamei e Mizoguchi [17] è basato sull'applicazione di 16 filtri direzionali nel dominio della frequenza. L'orientazione ottimale in ogni pixel è, quindi, scelta non solo a seconda della più alta risposta al filtro, ma utilizzando anche uno smoothing locale. Risultati analoghi si possono ottenere anche tramite l'uso di filtri di Gabor, ovvero filtri spaziali dal profilo cosinusoidale, introdotti per la prima volta dall'ingegnere ungherese Dennis Gabor [10], ma l'implementazione più semplice si ha nel dominio della frequenza.

Chikkerur, Cartwright e Govindaraju [5] propongono un approccio basato sull'analisi della STFT (*Short Time Fourier Transform*), ovvero la trasformata di Fourier di una parte del segnale (nel nostro caso una parte dell'immagine). L'immagine, infatti, viene suddivisa in blocchi non sovrapposti e per ognuno si calcola la trasformata di Fourier $F(u, v)$, scrivendo il suo spettro $|F(u, v)|$ in coordinate polari, ovvero $|F(r, \theta)|$. La probabilità di un dato valore θ all'interno di un

blocco si calcola tramite le densità di probabilità marginali:

$$p(\theta) = \int_r p(r,\theta)dr \qquad p(r,\theta) = \frac{|F(r,\theta)|}{\int_r \int_\theta |F(r,\theta)|drd\theta} \tag{4.2.7}$$

Il valore atteso di θ per il blocco è infine stimato tramite la (4.2.3) come:

$$E[\theta] = 90° + \frac{1}{2}\text{atan2}\left(\int_\theta p(\theta)\sin(2\theta)d\theta\ ,\ \int_\theta p(\theta)\cos(2\theta)d\theta\right) \tag{4.2.8}$$

4.2.2.2 Frequenza locale delle creste

La frequenza locale (anche detta densità) f_{xy} nel punto (x, y) è il numero di creste per unità di lunghezza lungo un ipotetico segmento centrato in (x, y) e ortogonale all'orientazione θ_{xy} della cresta. Calcolando la frequenza in ogni punto dell'immagine, oppure in ogni blocco, si può ottenere la cosiddetta immagine frequenza F.

Esistono numerosi metodi per generare l'immagine frequenza, ma in questa sezione ne vedremo solamente un paio.

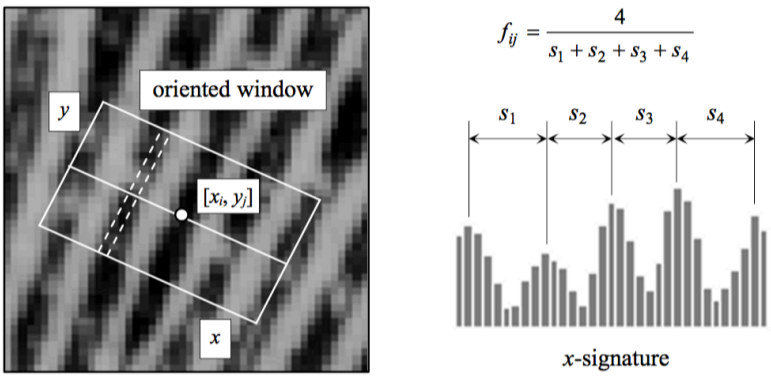


Figura 4.2.8: A sinistra possiamo vedere una finestra orientata centrata in (x_i, y_j); le linee tratteggiate mostrano i pixel i cui valori di grigio vengono mediati per ottenere un punto della segnatura. A destra vediamo, invece, l'x-signature corrispondente; notiamo che in essa sono presenti cinque picchi, la distanza reciproca verrà mediata per ottenere la stima della frequenza locale delle creste. Immagine tratta da [22].

Hong, Wan e Jain [14] stimano la frequenza locale delle creste contando il numero medio di pixel che intervengono tra due picchi successivi dei valori di grigio, lungo la direzione normale all'orientazione locale. A questo proposito la superficie S che corrisponde all'impronta digitale è sezionata con un piano parallelo all'asse z e ortogonale all'orientazione locale. Successivamente si può calcolare la frequenza f_{ij} del punto (x_i, y_j) nel modo seguente.

- Si centra una finestra di dimensione 32×16 nel punto (x_i, y_j) e la si orienta in modo che l'asse y coincida con l'orientazione locale.
- Si calcola l'*x-signature* dei valori di grigio della finestra, ovvero si mediano lungo l'asse y tutti i valori di grigio, ottenendo un profilo smussato delle creste nella finestra.
- f_{ij} è determinato come l'inverso della distanza media tra due picchi successivi dell'x-signature.

La Figura 4.2.8 schematizza il procedimento enunciato. Questo metodo è semplice e rapido, ma è sconsigliabile affidarsi troppo al profilo locale in immagini molto rumorose. In questi casi gli autori suggeriscono di utilizzare l'interpolazione e un lowpass filter. Yang e collaboratori [33] consigliano un metodo alternativo per trovare la frequenza dall'x-signature; essi infatti estraggono la distanza media tra le creste utilizzando le derivate prime e seconde.

Figura 4.2.9: Due immagini di impronte digitali differenti con corrispondente immagine frequenza calcolata utilizzando il metodo proposto in [21]. Dopo aver stimato la frequenza è stato applicato un lowpass filter per ridurre il rumore. I blocchi chiari indicano le frequenze più alte. Immagine tratta da [22].

Il secondo metodo che accenniamo è quello proposto da Maio e Maltoni [21]. In questo caso la forma assunta dalle increspature viene modellizzata localmente da una superficie sinusoidale e si utilizza il teorema della variazione totale per stimare la frequenza ignota. Si dice *variazione totale* V di una funzione h differenziabile nell'intervallo $[x_1, x_2]$:

$$V(h) = \int_{x_1}^{x_2} \left| \frac{dh(x)}{dx} \right| dx \tag{4.2.9}$$

Se la funzione h è periodica in $[x_1, x_2]$, oppure le variazioni d'ampiezza sono limitate, la variazione totale può essere espressa in funzione dell'ampiezza media α_m e della frequenza media f:

$$V(h) = (x_2 - x_1)2\alpha_m f \tag{4.2.10}$$

Quindi la frequenza ignota può essere calcolata con la formula inversa:

$$f = \frac{V(h)}{2(x_2 - x_1)\alpha_m} \tag{4.2.11}$$

Basandosi su questi risultati dell'analisi, Maio e Maltoni propongono un metodo pratico per stimare la frequenza delle creste; l'ampiezza media e la variazione totale vengono stimate tramite derivate parziali di primo e secondo ordine. In Figura 4.2.9 si possono vedere due esempi di immagini frequenza calcolati a partire da due diverse impronte con il metodo appena accennato.

4.2.2.3 Segmentazione

Il termine segmentazione è generalmente utilizzato per indicare la separazione dell'area dell'impronta digitale vera e propria dallo sfondo. Tale separazione è molto utile nel momento in cui si vogliano estrarre delle caratteristiche locali da un'immagine molto rumorosa, poiché generalmente il rumore è presente in maggiore quantità sullo sfondo.

Poiché le impronte digitali hanno una forma striata, usare le tecniche di thresholding globale o locale viste nel capitolo 2 non ci permette di isolare efficacemente l'area d'interesse. Infatti ciò che realmente discrimina l'impronta dallo sfondo circostante non è l'intensità media, ma da un lato la presenza di striature ed un pattern orientato, dall'altro una completa isotropia, ovvero un'assenza di orientazione dominante. Se per esempio l'impronta fosse uniformemente più chiara dello sfondo, allora potremmo utilizzare un threshold basato sulle differenze d'intensità, ma nella pratica abbiamo bisogno di tecniche di segmentazione più robuste. Vediamo nel seguito alcune delle tecniche che si possono sfruttare.

Le regioni dello sfondo sono caratterizzate, come abbiamo detto, da isotropia, al contrario l'impronta presenta direzioni ben precise. Si può innanzitutto calcolare l'immagine orientazione, quindi calcolare l'istogramma per ogni suo blocco di dimensione fissata; se in esso sono presenti dei picchi, evidentemente c'è una direzione prediletta, altrimenti da un profilo quasi piatto si può dedurre l'isotropia (vedi [26]). Il problema di questo metodo è quando si incontrano istogrammi perfettamente uniformi per il fatto che non si sono potute calcolare informazioni sull'orientazione, come magari in zone ad intensità costante. In

questo caso si può affiancare alla valutazione del profilo dell'istogramma anche un controllo sulla varianza dei livelli d'intensità (vedi [25]).

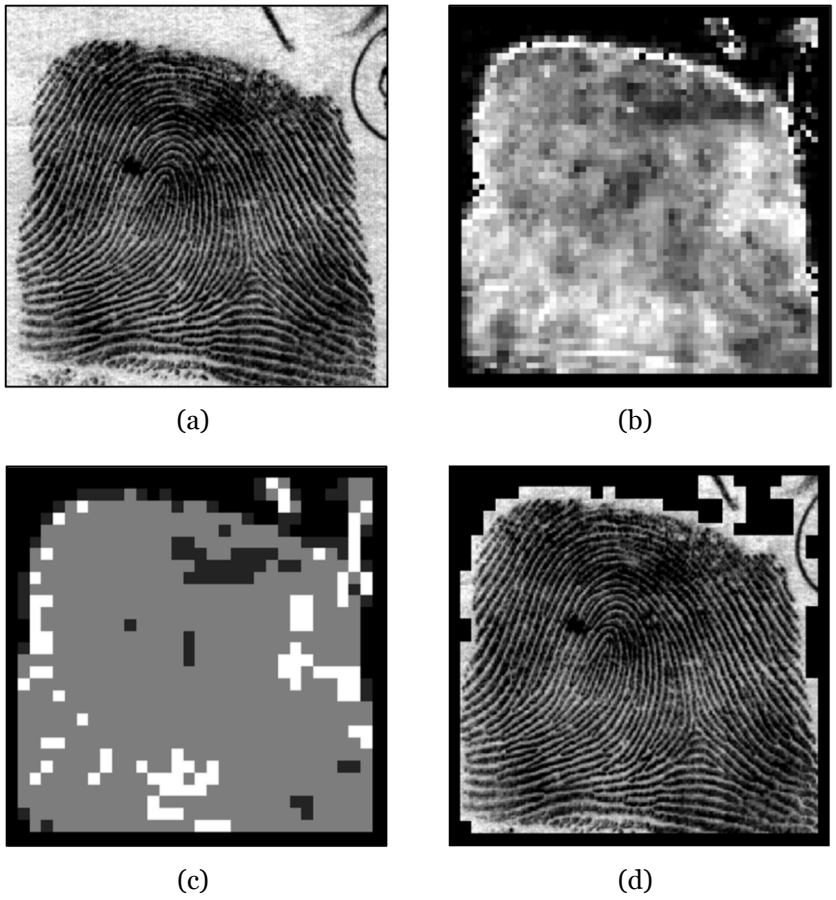


Figura 4.2.10: Segmentazioe di un'immagine col metodo proposto in [30]. (a) Immagine originale. (b) Immagine della varianza. (c) Immagine con la stima della qualità in base alla varianza. (d) Immagine segmentata. Immagine tratta da [22].

Ratha, Chen e Jain [30] hanno ideato un algoritmo che segmenta l'immagine assegnando ogni blocco allo sfondo oppure all'impronta a seconda della varianza dei livelli di grigio nella direzione ortogonale all'orientazione delle creste. L'idea di base è che le regioni dello sfondo non hanno elevata dipendenza dall'orientazione, quindi manifestano una bassa varianza, al contrario delle regioni d'interesse. Gli autori sono anche riusciti a ricavare un'indicazione sulla qualità dell'immagine in funzione della varianza, classificando ogni blocco come "buono", "medio", "scarso" e "sfondo". In Figura 4.2.10 si possono vedere i risultati di questo algoritmo per un'immagine di un'impronta digitale.

Un altro criterio con il quale selezionare lo sfondo dall'impronta è stato delineato da Maio e Maltoni [20], utilizzando il modulo medio del gradiente per ogni blocco dell'immagine. L'idea su cui si basa questo metodo è che la risposta del gradiente è molto elevata nelle zone in cui si alternano creste e valli, mentre si riduce drasticamente nelle zone dello sfondo.

Un'altra idea interessante è quella di andare a controllare lo spettro della trasformata di Fourier; infatti le zone che contengono creste e valli esibiscono un profilo sinusoidale, con definite frequenza ed orientazione, in corrispondenza delle quali lo spettro della trasformata di Fourier mostrerà dei picchi. Contributi alla segmentazione analizzando lo spettro di Fourier sono venuti da Pais Barreto Marques e Gay Thome [23] e da Chikkerur, Cartwright e Govindaraju [5].

Figura 4.2.11: Alcuni esempi di segmentazione con il metodo proposto da Chen et al. [4]. Immagine tratta da [22].

Sono state sviluppate anche molte tecniche di segmentazione attraverso algoritmi di machine learning, generalmente più accurate, ma anche più complesse, degli approcci basati sul thresholding rispetto a qualche caratteristica locale. Per avere un'idea si pensi al metodo proposto da Chen et al. [4]; un classificatore lineare viene "addestrato" a selezionare blocchi dell'impronta sulla base di:

- un indicatore dell'ammassamento del blocco,
- la differenza tra la media dell'intensità del blocco e la media delle intensità di tutta l'immagine,

- la varianza del blocco.

In Figura 4.2.11 possiamo vedere l'applicazione di questo algoritmo a tre impronte digitali.

4.2.2.4 Miglioramento

Figura 4.2.12: A partire da sinistra: immagine di buona qualità, immagine di qualità mediocre, con graffi e interruzioni anomale delle creste, immagine di scarsa qualità contenente molto rumore. Immagine tratta da [22].

Le prestazioni degli algoritmi per l'estrazione di minutiae, ma in generale anche di tutte le tecniche di riconoscimento basate sulle impronte digitali, dipendono moltissimo dalla qualità delle immagini acquisite. In un'impronta ideale, creste e valli si alternano e fluiscono con una direzione che varia in modo continuo. In tali situazioni le minutiae possono essere facilmente individuate. Nelle applicazioni pratiche, a causa delle condizioni della pelle, del rumore generato dal sensore, dalla differente pressione esercitata dalle dita, una significativa percentuale di immagini è di scarsa qualità; in molti casi addirittura la stessa immagine contiene regioni di buona, media e scarsa qualità (vedi Figura 4.2.12). Si possono distinguere tre principali tipi di degradazione delle acquisizioni:

1. le creste non sono continue, ma presentano molte interruzioni;
2. le creste parallele non sono ben separate; questo è dovuto principalmente alla presenza di rumore che tende a collegarle;
3. tagli, pieghe e ferite proprie del dito.

Immagini che presentano alcune di queste condizioni rappresentano un ostacolo alla corretta individuazione delle minutiae; infatti possono avvenire i seguenti problemi: vengono estratte minutiae fasulle, non si trovano alcune di quelle genuine, si compiono ampi errori di localizzazione delle minutiae. Per assicurare buone prestazioni agli algoritmi che lavorano sulle impronte digitali, un processo di miglioramento preliminare deve essere messo in atto.

Figura 4.2.13: Un esempio di normalizzazione dell'immagine tramite il metodo esposto da Hong et at. [14]. Immagine tratta da [22].

Un approccio che si può praticare è quello di modificare l'immagine un pixel per volta; in questo caso il nuovo valore che ogni pixel assume dipende solamente dal suo valore precedente e da alcune caratteristiche globali, ma non dall'intensità dei punti del suo intorno. Una delle tecniche di questo tipo è stata utilizzata da Hong et al. [14]; il valore che deve assumere il pixel (x, y) è

$$I'(x,y) = \begin{cases} m_0 + \sqrt{(I(x,y) - m)^2 \cdot \frac{v_0}{v}} & \text{se } I(x,y) > m \\ m_0 - \sqrt{(I(x,y) - m)^2 \cdot \frac{v_0}{v}} & \text{altrimenti} \end{cases} \tag{4.2.12}$$

dove m e v sono rispettivamente la media e la varianza dell'immagine, mentre m_0 e v_0 sono rispettivamente la media e la varianza che si desidera ottenere per la nuova immagine; si può pensare a questo procedimento come ad una sorta di normalizzazione dell'immagine. Un esempio dell'utilizzo di questa tecnica si può trovare in Figura 4.2.13. Kim e Park [19] utilizzarono la normalizzazione applicandola localmente, in modo da colmare le differenze di contrasto e luminosità tra le varie parti dell'immagine. Approcci del genere, comunque, riescono ad apportare solamente leggeri miglioramenti all'immagine, senza risolvere i problemi esposti in precedenza.

Piuttosto che andare a modificare i punti uno alla volta, senza tener conto della struttura del loro intorno, conviene eseguire il cosiddetto *contextual filering*, ovvero utilizzare un filtro differente per ogni blocco dell'immagine. I metodi di

filtraggio d'immagini convenzionali utilizzano un solo filtro per tutta l'immagine, mentre in questo caso un insieme di filtri differenti viene calcolato a priori, quindi uno di questi viene assegnato ad ogni blocco sulla base delle caratteristiche locali, quindi si esegue la convoluzione dei punti del blocco con il filtro. Il miglioramento che si ottiene, rispetto al tradizionale uso di un filtro per tutta l'immagine, è dovuto al fatto che le impronte digitali manifestano una forma localmente sinusoidale, con frequenza ed orientazione che variano da una zona all'altra. Nonostante ogni filtro ha una sua particolare definizione e quindi un diverso intento, il loro comportamento si può sintetizzare come:

- lowpass filtering lungo la direzione delle creste, con l'intento di colmare le interruzioni anomale;
- eseguire un bandpass filtering, cercando di evidenziare la separazione di creste e valli ed eliminando gli altri elementi.

Non ci soffermiamo molto su questa categoria di filtri, poiché nella sezione 4.3 verrà descritto in modo dettagliato un algoritmo di questo tipo.

Una interessante tipologia di miglioramento è quello che analizza l'immagine a diverse risoluzioni, per questo chiamato *multi-resolution enhancement*. L'idea è decomporre l'immagine in molte sottoimmagini, in base alla frequenza: a bassa frequenza sono colmate le interruzioni delle creste e rimossi gli errori più grossolani dell'acquisizione, mentre alle frequenze più elevate si cerca di eliminare il rumore preservando comunque i dettagli più piccoli.

Concludiamo con un accenno alle tecniche per riparare ai tagli, effettivamente presenti sulla pelle; la loro presenza influenza negativamente il calcolo dell'orientazione e dà luogo a false minutiae, anche se in alcuni casi anche le ferite stesse possono essere utili al riconoscimento di qualche individuo. Ad ogni modo alcune tecniche sono state create ad hoc per localizzare questi tagli e sopprimerli. Giusto per fare un esempio e chiarire le idee, accenniamo al metodo proposto da Vernon [32]. L'idea alla sua base è che i tagli sono caratterizzati da terminazioni delle creste collineari, per cui un metodo per individuarli può essere: calcolare le minutiae di tipo terminazione, quindi analizzare mediante la trasformata di Hough la loro distribuzione per poi eliminare i collineari.

4.2.2.5 Localizzazione delle minutiae

La maggior parte dei sistemi automatici di confronto tra impronte digitali sono basate sul rilevamento di minutiae, per cui la loro estrazione è uno degli aspetti più importanti e studiati del campo dell'analisi di impronte digitali.

I metodi proposti finora, per la maggior parte, ricadono in due categorie: quelli che richiedono la binarizzazione dell'immagine e quelli che operano direttamente in scala di grigi.

Metodi basati su binarizzazione La binarizzazione dell'immagine può avvenire in molti modi:

- usando un threshold globale, spesso inaffidabile a causa delle differenze delle condizioni di acquisizione, ma al contempo semplice e computazionalmente rapido;
- preferendo un threshold locale, in modo da non perdere informazioni a causa delle differenze di intensità e contrasto nelle varie regioni dell'immagine;
- utilizzando operatori differenziali, come nella tecnica proposta da Moayer e Fu [27], in cui viene calcolato il laplaciano dell'immagine ad ogni iterazione, assegnati valori 0 ed 1 ai pixel che escono dall'intervallo delimitato da due threshold dinamici, quindi avvicinati i valori di threshold in modo da assicurare la convergenza;
- metodi ben più complessi che utilizzano strumenti dell'analisi, della topologia o della geometria differenziale.

Non scendiamo nei dettagli della binarizzazione, poiché in letteratura esistono numerosissimi metodi e divergeremmo troppo dal nostro obiettivo; si rimanda eventualmente al testo di riferimento per questo argomento, ovvero [22].

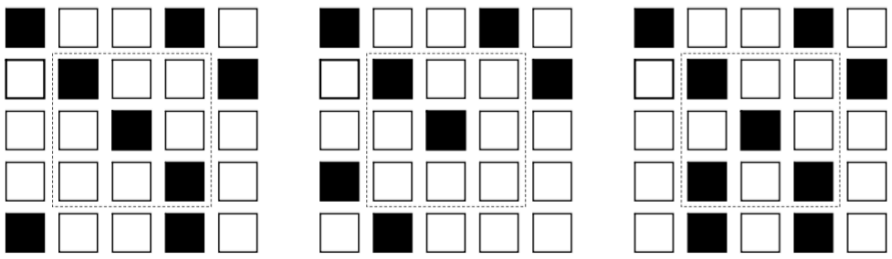


Figura 4.2.14: Possibili risultati di un algoritmo di thinning; a partire da sinistra abbiamo un pixel lungo una cresta, un pixel di terminazione, un punto di biforcazione, rispettivamente con 2, 1 e 3 punti nell'intorno (evidenziato dalle linee tratteggiate). Immagine tratta da [22].

Dopo aver provveduto alla binarizzazione con qualche algoritmo, in genere viene eseguito il *thinning*, assottigliamento, che riduce ogni linea ad un pixel di

larghezza; il risultato sarà lo scheletro dell'impronta. Prima di questo passo potrebbero essere necessari anche degli algoritmi per colmare le interruzioni delle creste, riempire buchi, eliminare collegamenti anomali tra creste.

Una volta eseguito il thinning basta semplicemente contare quanti punti ci sono nell'intorno quadrato 3×3 di ogni pixel, questo escluso:

- se ce sono 2, allora abbiamo un punto interno ad una cresta;
- se ne contiamo 1, siamo in presenza di una terminazione;
- se ne abbiamo 3, abbiamo una biforcazione;
- se il numero è superiore a 3 allora abbiamo minutiae più complesse.

In Figura 4.2.14 si può vedere un esempio di quanto appena discusso.

Esistono anche altri approcci basati su reti neurali, tracciamento delle creste, curve principali (ovvero curve che riescono ad approssimare le creste) e così via. Il metodo più semplice ed utilizzato è comunque quello appena descritto.

Metodi per scale di grigi In letteratura sono presenti molti metodi che, invece di binarizzare l'immagine, lavorano direttamente in scala di grigi. Questo è dovuto a tre principali motivi: durante il processo di binarizzazione si perdono molte informazioni che potrebbero essere utili; binarizzazione e thinning sono processi che richiedono tempo, quindi poco adatti a hardware di bassa fascia; in assenza di un miglioramento dell'immagine preliminare alla binarizzazione, questa non produce risultati soddisfacenti se applicata ad immagini di bassa qualità.

Uno dei principali e più noti algoritmi per la diretta estrazione di minutiae dall'immagine è quello proposto da Maio e Maltoni [20]. Ogni cresta viene seguita dall'algoritmo, dall'inizio fino al rilevamento di qualche anomalia che, in base a delle precise regole, viene indicata come minutia. Seguire il profilo delle creste significa, ad ogni passo, sezionare la cresta trasversalmente, trovare il massimo locale, spostarsi lungo l'orientazione della cresta per calcolare la nuova sezione. Il procedimento si itera fino a che non si riesca più a trovare un massimo locale, per cui siamo in presenza di una minutia.

Altri approcci sono stati utilizzati, alcuni sfruttando reti neurali, altri sfruttando le proprietà di simmetria delle minutiae; per conoscerne i dettagli si rimanda al testo [22].

4.2.3 Matching tra impronte

Ottenere un *matching* affidabile, ovvero un abbinamento tra due acquisizioni che appartengono allo stesso individuo, è un problema estremamente diffici-

le, dovuto principalmente ad un'ampia variabilità intra-classe. I principali fattori che incidono sono: diversa posizione od orientazione dell'impronta rispetto all'immagine, parziale sovrapposizione, distorsione non lineare, variazione di pressione durante l'acquisizione, cambiamento delle condizioni della cute, rumore ed infine errori nel processo di estrazione delle caratteristiche salienti dell'immagine. A causa di tutto ciò potrebbe anche verificarsi che campioni provenienti dallo stesso dito diano acquisizioni molto differenti, così come potrebbe avvenire che impronte da dita diverse vengano accomunate dalle stesse caratteristiche.

Chi analizza manualmente le impronte digitali considera vari loro aspetti:

- simile configurazione globale, ovvero controllano che due impronte siano dello stesso tipo;
- fattore qualitativo, chiedendo la piena corrispondenza tra i minute details;
- aspetto quantitativo, per cui si richiede che deve essere trovato almeno un certo numero di minute details corrispondenti (le direttive dell'FBI prevedono un minimo di 12 dettagli).

Sostanzialmente, nel corso degli anni, sono state definite solide regole da seguire ed è disponibile una sorta di diagramma di flusso che l'esaminatore può semplicemente seguire.

Per quanto riguarda l'esame automatico, l'algoritmo non deve per forza seguire la stessa procedura, nonostante molte procedure automatiche sia ad essa ispirate. Nel corso degli ultimi 40 anni sono stati delineati numerosi metodi, alcuni direttamente progettati per l'automazione; questi si possono classificare in tre grandi gruppi.

1. *Matching tramite correlazione*: due impronte digitali vengono sovrapposte e viene calcolato il coefficiente di correlazione, tra i pixel corrispondenti, per varie rotazioni e traslazioni (così da acquisire robustezza rispetto al disallineamento).
2. *Matching per confronto di minutiae*: le minutiae vengono estratte da entrambe le impronte digitali, così da ottenere un insieme di punti del piano; l'algoritmo prosegue cercando di allineare il maggior numero di minutiae di un'impronta alle corrispondenti dell'altra immagine.
3. *Matching basato su caratteristiche che non siano minutiae*: l'estrazione di minutiae nel caso d'immagini di qualità molto bassa è un procedimento molto difficile, mentre ci sono caratteristiche che possono essere estratte con molta più facilità ed affidabilità; alcuni esempi potrebbero essere l'orientazione e la frequenza locale, oppure informazioni sulla texture o sulla

forma delle creste. Generalmente tali caratteristiche sono molto meno attendibili delle minutiae, se l'immagine ha una qualità sufficiente, mentre sono consigliabili quando risulta pressoché impossibile individuare correttamente le minutiae.

4.2.4 Classificazione automatica

Un'elevato numero di impronte digitali sono acquisite ed immagazzinate ogni giorno sia per le applicazioni forensi, sia per quelle non giudiziarie. Nell'identificazione automatica basata sulle impronte digitali, si richiede che l'acquisizione sia confrontata con tutti i campioni immagazzinati nel database. L'operazione da eseguire è quindi molto complessa e, per essere ultimata nei tempi utili all'applicazione, richiederebbe risorse computazionali sicuramente non disponibili. Per ovviare a questo problema è preferibile classificare le impronte accuratamente, in modo che il confronto possa avvenire, senza rischio di falsi negativi, solamente tra "pochi" campioni. Per essere più chiari: si stabiliscono le possibili categorie in cui possono ricadere le varie impronte, quindi, in una fase preliminare, i campioni del database vengono assegnati ad ogni categoria; quando una nuova impronta da riconoscere viene acquisita, il software le assegna una categoria, quindi la va a confrontare con tutti i campioni presenti nel database in quella particolare categoria; ammesso che la classificazione sia avvenuta correttamente, nessuna delle impronte appartenenti alle altre categorie poteva essere abbinata a quella in esame, poiché esse possiedono sicuramente caratteristiche evidentemente differenti.

Uno dei primi e più noti approcci alla classificazione fu quello introdotto da Henry, che suddivideva le impronte in cinque categorie: spirale, cappio destro, cappio sinistro, arco piano e arco a tenda. Sfortunatamente la distribuzione delle impronte nelle cinque categorie non è uniforme, quindi esistono alcune categorie che contengono molti più elementi delle altre, quasi vanificando l'utilità di tale categorizzazione. Per risolvere il problema sono stati introdotti numerosi algoritmi di *indicizzazione*, ovvero il processo che assegna ad ogni campione una stringa alfanumerica detta *indice*; tra i più comuni metodi d'indicizzazione c'è la cosiddetta "classificazione continua", in cui le impronte non sono partizionate in classi non sovrapposte, ma ognuna viene segnata con un vettore numerico che ne riassume le caratteristiche principali. In questo modo ogni impronta è in corrispondenza biunivoca con un punto nello spazio n-dimensionale, se n è il numero di caratteristiche prese in considerazione; anche ad ogni nuova acquisizione si assegna un vettore di caratteristiche, ovvero le si associa un punto, quindi si considerano i punti vicini e vengono controllate solamente le impronte ad essi associate.

4.3 Il metodo proposto

Finora abbiamo accennato alle varie tecniche esistenti in letteratura per l'analisi delle impronte digitali; in questa sezione riassumeremo l'algoritmo utilizzato per sviluppare il software per l'individuazione delle minutiae. Abbiamo deciso di seguire l'approccio di Hong et al. [14], ovvero un metodo che affronta il miglioramento dell'impronta attraverso la normalizzazione, quindi il contextual filtering utilizzando un filtro di Gabor. Per aumentare l'efficacia dell'algoritmo abbiamo seguito i consigli di Yang et al. [33], che modificano leggermente il filtro utilizzato in modo da adattarlo meglio alle varie configurazioni di creste e valli. Successivamente verrà applicato all'immagine migliorata l'algoritmo di Maio e Maltoni [20] per l'individuazione delle minutiae; questo, come accennato nella sezione 4.2, agisce direttamente sull'immagine in scala di grigi. Infatti, poiché l'algoritmo di miglioramento è abbastanza complesso e richiede del tempo per l'esecuzione, abbiamo ritenuto giusto non consumare altre risorse computazionali per la binarizzazione e per il thinning.

4.3.1 Miglioramento dell'impronta

4.3.1.1 Normalizzazione

Figura 4.3.1: Esempio di normalizzazione: a sinistra immagine originale, a destra quella normalizzata con $M_0 = 100$ e $Var_0 = 100$. Immagine tratta da [14].

Sia I un'immagine di un'impronta digitale in scala di grigi, ovvero una matrice $N \times N$, e sia $I(i,j)$ l'intensità del pixel alla riga i e colonna j. Il primo passo del nostro algoritmo è la normalizzazione dell'immagine, cosicché sia uniformata a dei valori di media e varianza scelti opportunamente. Denotiamo con $M[I]$ e $Var[I]$ rispettivamente la media e la varianza dell'immagine I, che calcoliamo

come:

$$M[I] = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} I(i,j)$$
$$Var[I] = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} (I(i,j) - M[I])^2 \tag{4.3.1}$$

Chiamiamo con $G(i,j)$ l'intensità dell'immagine normalizzata nel punto alla riga i e colonna j; quindi calcoliamo:

$$G(i,j) = \begin{cases} M_0 + \sqrt{\frac{Var_0 \cdot (I(i,j)-M^2)^2}{Var[I]}} & \text{se } I(i,j) > M[I] \\ M_0 - \sqrt{\frac{Var_0 \cdot (I(i,j)-M^2)^2}{Var[I]}} & \text{altrimenti} \end{cases} \tag{4.3.2}$$

dove M_0 e Var_0 sono rispettivamente media e varianza desiderate. Come abbiamo già accennato nella sezione 4.2, questa tecnica agisce su ogni pixel, uno alla volta, senza modificare la chiarezza con cui vediamo la struttura di creste e valli dell'impronta; il suo principale scopo è ridurre le variazioni d'intensità lungo le creste, così da facilitare passi seguenti (vedi Figura 4.3.1).

4.3.1.2 Orientazione

Cerchiamo ora di stimare l'immagine orientazione, questo ci permetterà di definire un sistema di coordinate locale, adattato ad ogni pixel. Utilizziamo, come descritto in [14], un algoritmo basato sul metodo dei minimi quadrati, per cui dall'immagine normalizzata G, procediamo come segue.

- Calcoliamo le derivate parziali $\frac{\partial}{\partial x}G(i,j)$ e $\frac{\partial}{\partial y}G(i,j)$ in ogni punto (i,j); a seconda della precisione richiesta si possono utilizzare operatori di Sobel, oppure operatori più complessi, come ad esempio quello di Marr-Hildreth. Nell'algoritmo originale di Hong et al. ([14]) l'immagine era suddivisa in blocchi, ma abbiamo preferito scegliere l'approccio di Yang et al. ([33]), che dà maggiore precisione.

- Attraverso le formule indicate in (4.2.4) e suggerite da Ratha et al. ([30]) calcoliamo una stima dell'orientazione locale delle creste, che indichiamo con $\theta(i,j)$.

- In modo simile a quanto esposto nella sezione 4.2, calcoliamo il campo vettoriale continuo

$$\mathbf{d} = [\cos(2\theta(i,j)), \sin(2\theta(i,j))] \tag{4.3.3}$$

Figura 4.3.2: Esempio di immagine orientazione tratto da [14].

quindi eseguiamone la convoluzione con un lowpass filter, così da attenuare i bruschi cambiamenti di orientazione, generalmente legati ad errori di stima.

- Infine calcoliamo l'immagine orientazione finale

$$O(i,j) = \frac{1}{2}\tan\left(\frac{\overline{\mathbf{d}}_y(i,j)}{\overline{\mathbf{d}}_x(i,j)}\right) \tag{4.3.4}$$

dove $\overline{\mathbf{d}}$ è il campo $\mathbf{d}$ filtrato, mentre $\overline{\mathbf{d}}_x$ e $\overline{\mathbf{d}}_y$ sono rispettivamente le sue componenti x e y.

La Figura 4.3.2 mostra un esempio del calcolo dell'orientazione.

4.3.1.3 Ampiezza di creste e valli

A questo punto vediamo la procedura descritta in [33] per calcolare una sorta di immagine della frequenza delle creste. Infatti ciò che ci accingiamo a calcolare sono le ampiezze medie sia delle creste sia delle valli, invece di considerarle uguali come fatto in [14].

Innanzitutto eseguiamo uno smoothing gaussiano unidimensionale lungo l'orientazione individuata al passo precedente. Questa operazione consente di riempire parzialmente i buchi dovuti ai pori sudoripari e uniformare l'intensità lungo le creste e lungo le valli.

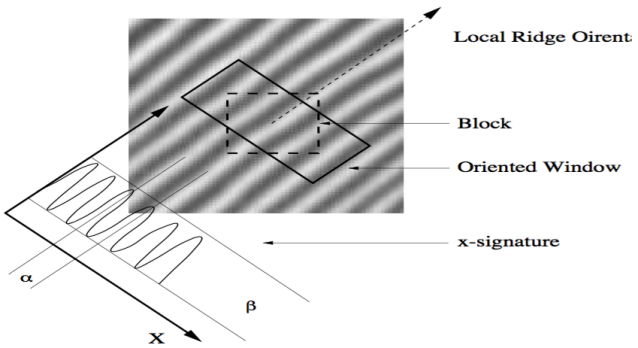


Figura 4.3.3: Finestra orientata utilizzata per calcolare l'x-signature. Immagine tratta da [14].

Procediamo suddividendo l'immagine G in blocchi di dimensione $w \times w$; per ogni blocco b supponiamo che (i_b, j_b) sia il suo centro, quindi consideriamo una finestra di dimensione $l \times w$ orientata con l'asse y lungo l'orientazione suggerita da $O(i_b, j_b)$ (vedi Figura 4.3.3). Successivamente si calcola l'x-signature X per ogni blocco, ottenendo ogni componente del vettore X come media lungo l'asse y, ovvero media delle colonne, dei valori d'intensità della finestra orientata. In formule:

$$X(k) = \frac{1}{w}\sum_{s=0}^{w-1} G(u,v) \qquad k = 0, 1, \dots, l-1 \tag{4.3.5}$$

dove le coordinate u e v sono date da

$$\begin{aligned} u &= i_b + \left(s - \frac{w}{2}\right)\cos O(i_b, j_b) + \left(k - \frac{l}{2}\right)\sin O(i_b, j_b) \\ v &= j_b + \left(s - \frac{w}{2}\right)\sin O(i_b, j_b) + \left(\frac{l}{2} - k\right)\cos O(i_b, j_b) \end{aligned} \tag{4.3.6}$$

Se non ci sono minutiae nella finestra orientata, allora l'x-signature mostrerà un profilo sinusoidale, con le stesse ampiezze delle valli e delle creste della finestra. Queste possono quindi essere stimate a partire dall'x-signature e per farlo utilizziamo le derivate prime e seconde. Poiché stime inaccurate delle ampiezze di creste e valli possono portare a un miglioramento dell'immagine non uniforme, cerchiamo di stimarle con la maggiore accuratezza possibile. In [33] si consiglia di eseguire il fitting dell'x-signature, in modo da riuscire a calcolarne con più precisione le derivate; noi abbiamo scelto di utilizzare le spline per interpolare i valori dell'x-signature, così da lavorare con polinomi semplici, che possiamo differenziare facilmente, e da avere un ottimo fitting dei valori.

Per stimare le ampiezze di creste e valli, utilizziamo i cambi di segno della derivata seconda e il modulo della derivata prima dell'x-signature. In sostanza, la distanza tra due punti in cui la derivata seconda cambia segno viene considerata un'ampiezza utile, se in quei punti la derivata prima ha modulo sufficientemente elevato; in altre parole, vengono considerati margini di un picco dell'x-signature i punti con sufficiente pendenza ed in cui si ha un cambio di concavità. Successivamente l'ampiezza trovata viene chiamata W_r, se il segno della derivata seconda è negativo, quindi viene associata ad una cresta, mentre la denotiamo W_v, se la derivata seconda è positiva, quindi corrisponde ad una valle. Poiché nelle applicazioni pratiche si conosce a priori l'intervallo in cui possono cadere le ampiezze, se troviamo dei valori che eccedono tale intervallo, allora dobbiamo sostituirli con la media dei valori trovati nei blocchi circostanti.

4.3.1.4 Segmentazione

Nel seguito ci farà comodo avere una segmentazione, anche approssimativa, dell'immagine, per cui vediamo come ottenerne una senza incidere troppo sul costo computazionale globale. L'approccio utilizzato è quello di un threshold locale, per cui si calcolano media m e varianza s per ogni blocco, con riferimento alla suddivisione dell'immagine eseguita al passo precedente. Per ogni blocco, ogni punto è classificato rispettivamente come cresta o come valle se la sua intensità supera o meno il valore di soglia $m+\delta s$, dove δ è un parametro arbitrario, che in [33] viene consigliato di impostare a 0.2. Se in qualche applicazione fossero necessari dei risultati più attendibili di quelli ottenuti con questo tipo di segmentazione, si possono scegliere anche altri approcci; il problema, in questo caso, potrebbe divenire il consumo eccessivo di risorse computazionali per questa fase.

4.3.1.5 La scelta del filtro da utilizzare

Le funzioni di Gabor sono conosciute per avere ottime proprietà sia nel dominio dello spazio, che nel dominio della frequenza; per questo motivo sono anche molto utilizzate e studiate nell'ambito dell'elaborazione di immagini. La famiglia di filtri di Gabor bidimensionali fu presentata inizialmente da Daugman nel 1980 [7]; un classico filtro di Gabor in 2D è un'onda piana sinusoidale orientata e con una particolare frequenza, tutta contenuta in una campana gaussiana. Un tale filtro ha forma generale:

$$H(x,y) = e^{-\frac{(x-x_0)^2}{2\sigma_x^2}-\frac{(y-y_0)^2}{2\sigma_y^2}} \cdot e^{-2\pi i(u_0(x-x_0)+v_0(y-y_0))} \tag{4.3.7}$$

dove (x_0, y_0) è il centro del filtro, quindi la sua posizione nell'immagine, σ_x e σ_y sono la deviazione standard della campana gaussiana rispettivamente lungo

l'asse x e l'asse y; (u_0, v_0) sono parametri che specificano la frequenza e l'orientazione che deve avere la sinusoide, determinando rispettivamente $\omega_0 = \sqrt{u_0^2 + v_0^2}$ e $\theta_0 = \tan^{-1} \frac{u_0}{v_0}$.

Dalla (4.3.7), selezionando adeguatamente i parametri, si può derivare l'equazione di un filtro di Gabor 2D simmetrico e reale:

$$h(x, y, T, \phi) = e^{-\frac{1}{2}\left[\frac{x_\phi^2}{\sigma_x^2} + \frac{y_\phi^2}{\sigma_y^2}\right]} \cos\left(\frac{2\pi x_\phi}{T}\right) \tag{4.3.8}$$

dove

$$\begin{aligned} x_\phi &= x \cos\phi + y \sin\phi \\ y_\phi &= -x \sin\phi + y \cos\phi \end{aligned} \tag{4.3.9}$$

dove ϕ è l'orientazione del filtro e T il periodo della sinusoide.

Notiamo che si può decomporre la (4.3.8) in due parti ortogonali, una parallela all'orientazione e l'altra perpendicolare:

$$\begin{aligned} h(x, y, T, \phi) &= h_x(x, y, T, \phi) \cdot h_y(x, y, \phi) \\ &= \left\{ e^{-\frac{x_\phi^2}{2\sigma_x^2}} \cos\left(\frac{2\pi x_\phi}{T}\right) \right\} \cdot e^{-\frac{y_\phi^2}{2\sigma_y^2}} \end{aligned} \tag{4.3.10}$$

La componente h_x si comporta come un filtro di Gabor unidimensionale, ovvero un bandpass filter in direzione perpendicolare all'orientazione, mentre la seconda componente h_y è una funzione gaussiana, che quindi rappresenta un lowpass filter lungo l'orientazione.

Poiché il pattern delle impronte digitali è costituito dall'alternanza di creste e valli di differenti ampiezze, non è corretto utilizzare un solo periodo T, poiché il filtro non riesce bene ad adattarsi alle caratteristiche locali. Sostituiamo allora la funzione coseno con un'altra funzione periodica

$$F(x, T_1, T_2) = f\left(x - \left\lfloor \frac{x}{\frac{T_1}{2} + \frac{T_2}{2}} \right\rfloor \cdot \left(\frac{T_1}{2} + \frac{T_2}{2}\right)\right) \tag{4.3.11}$$

dove

$$f(x) = \begin{cases} \cos\left(\frac{2\pi x}{T_1}\right) & 0 \le x \le \frac{T_1}{4} \\ -\cos\left(\frac{2\pi\left(x - \frac{T_1}{4} - \frac{T_2}{4}\right)}{T_2}\right) & \frac{T_1}{4} < x < \frac{T_1}{4} + \frac{T_2}{2} \\ \cos\left(\frac{2\pi\left(x - \frac{T_1}{2} - \frac{T_2}{2}\right)}{T_1}\right) & \frac{T_1}{4} + \frac{T_2}{2} \le x \le \frac{T_1}{2} + \frac{T_2}{2} \end{cases} \tag{4.3.12}$$

dove con $\lfloor x \rfloor$ abbiamo indicato il più grande intero non maggiore di x.

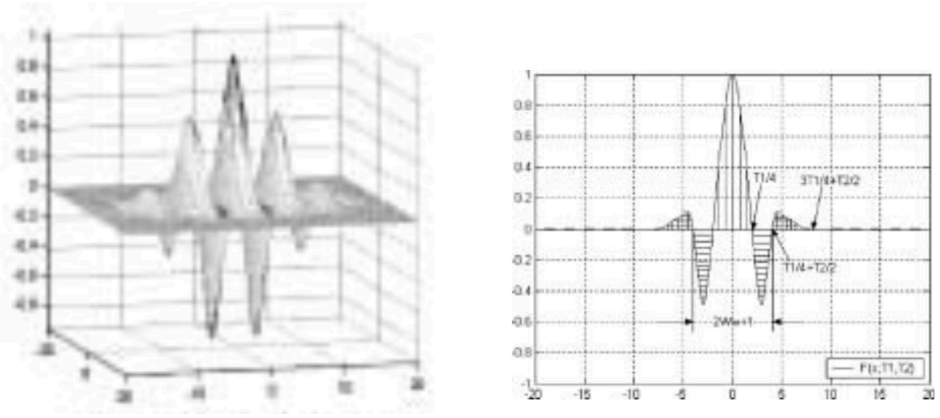
4.3.1.6 Selezione dei parametri

La selezione dei parametri gioca un ruolo cruciale nell'uso del filtro appena definito, ma non è un compito molto semplice. In [33] viene proposto un modo per calcolare i cinque parametri che intervengono nella definizione del filtro: ϕ, T_1, T_2, σ_x e σ_y.

Per quanto riguarda la stima del parametro ϕ, non abbiamo bisogno di fare altro che prendere il corrispondente valore nell'immagine orientazione che abbiamo calcolato.

Per stimare i parametri T_1 e T_2 dobbiamo fare affidamento, invece, alle ampiezze W_r e W_v ed alla segmentazione. T_1 e T_2 sono i periodi di due sinusoidi, quindi rappresentano lo spazio intercorso tra due picchi successivi; per come abbiamo calcolato le ampiezze di creste e valli, risulta evidente che un periodo corrisponde al doppio dell'ampiezza corrispondente. Supponiamo di dover filtrare attorno al punto (i_b, j_b) e che, in accordo con la segmentazione effettuata, siamo in presenza di una cresta, allora assegniamo $T_1 = 2 \cdot W_r$ e $T_2 = 2 \cdot W_v$; nel caso in cui il punto (i_b, j_b) sia di valle, allora calcoliamo $T_1 = 2 \cdot W_v$ e $T_2 = 2 \cdot W_r$.

Per quanto riguarda i parametri σ_x e σ_y, essi regolano l'ampiezza del filtro, rispettivamente perpendicolarmente e lungo l'orientazione locale. Il parametro σ_y non va impostato ad un valore troppo grande, perché potrebbe eliminare le minutiae, per cui un valore di 4 potrebbe essere la scelta ideale. Paragonato a σ_y, sicuramente σ_x gioca un ruolo più importante, regolando il miglioramento del contrasto dell'immagine, per cui va impostato con delle regole più precise. Se lo impostiamo ad un valore troppo grande, il fattore h_x nella (4.3.10) avrò componenti di frequenza troppo elevata, che fanno oscillare troppo il filtro attorno al



(a) (b)

Figura 4.3.4: In (a) si può vedere un esempio di filtro di Gabor 2D nell'dominio dello spazio. In (a) vediamo una sezione del filtro di Gabor parallela all'asse x, da cui si possono meglio interpretare le equazioni (4.3.13) e (4.3.14). Immagini tratte da [33].

centro, dando origine ad artefatti nell'immagine finale. D'altra parte selezionarlo ad un valore eccessivamente basso, porta ad avere quasi una semplice gaussiana invece che h_x, per cui si potrebbe andare a smussare le differenze tra creste e valli. Sicuramente il parametro σ_x dipende dal calcolo di T_1 e T_2, ovvero dalla struttura locale dell'impronta, per cui conviene stabilirlo a partire da essi; in particolare occorre risolvere

$$\frac{\int_0^{T_1/4} e^{-\frac{x^2}{2\sigma_x^2}} \cos\left(\frac{2\pi x}{T_1}\right) dx}{\int_{T_1/4}^{T_1/4+T_2/2} e^{-\frac{x^2}{2\sigma_x^2}} \cos\left(\frac{2\pi\left(x - T_1/4 - T_2/4\right)}{T_2}\right) dx} = Q \tag{4.3.13}$$

e

$$\int_{T_1/4+T_2/2}^{3T_1/4+T_2/2} e^{-\frac{x^2}{2\sigma_x^2}} \cos\left(\frac{2\pi\left(x - T_1/2 - T_2/2\right)}{T_1}\right) dx \approx 0 \tag{4.3.14}$$

Fissato Q, si può calcolare σ_x corrispondente a particolari T_1 e T_2 semplicemente utilizzando metodi numerici. Nei vincoli (4.3.13) e (4.3.14), Q rappresenta la proporzione tra l'area del picco principale e l'area dei due picchi adiacenti; scegliendo $Q > 1$ si può conferire stabilità all'oscillatore, facendo in modo che svaniscano tutti i picchi successivi. La risoluzione numerica della (4.3.13) e della

(4.3.14) dovrebbe avvenire per ogni blocco, ma possiamo calcolare a priori un insieme di σ_x collegate a varie combinazioni di T_1 e T_2, poiché l'intervallo in cui questi oscillano è noto. La Figura 4.3.4 può aiutare a comprendere la scelta del parametro σ_x.

L'obiettivo finale è eseguire la convoluzione di ogni singolo blocco con il filtro corrispondente, ma bisogna ancora decidere la grandezza del filtro. In [33] si propone di utilizzare, anche in questo caso, una grandezza variabile a seconda dei parametri T_1 e T_2 assegnati ad ogni blocco. In particolare gli autori consigliano di scegliere maschere di dimensione $(T_2 + \frac{T_1}{2}) \times 11$, per evitare sia il troncamento del filtro, sia l'instabilità; si noti che la prima dimensione del filtro risulta collegata, tramite la (4.3.13) e la (4.3.14), al parametro σ_x, mentre la seconda dimensione è costante così come lo è σ_y.

4.3.2 Individuazione delle minutiae

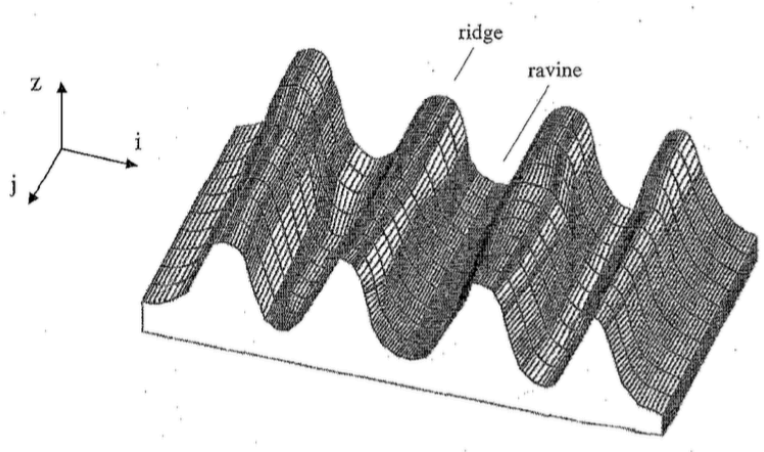


Figura 4.3.5: Superficie corrispondente ad una piccola area di un'impronta digitale. Immagine tratta da [20].

Con le convenzioni stabilite nella sezione 4.2, abbiamo che l'impronta rispecchia il profilo visualizzato in Figura 4.3.5. Da un punto di vista matematico, possiamo definire una cresta come l'insieme dei massimi locali lungo una direzione. Maio e Maltoni [20] propongono allora di utilizzare un algoritmo che segua le creste, calcolando ad ogni iterazione il massimo locale di una sezione dell'impronta ortogonale all'orientazione; supponendo che (i_s, j_s) sia un punto di massimo locale e ϕ_0 l'orientazione locale, l'algoritmo che segue il profilo della cresta può essere scritto in pseudo-codice nel modo seguente:

```
seguiCresta (i_s, j_s, φ_0)
{
```

```
    (i_c, j_c) := (i_s, j_s);
    φ_c := φ_0;
    while (fineNonRaggiunta)
    {
        (i_t, j_t) := (i_c, j_c) + μ pixel lungo la direzione φ_c;
        Ω := sezione in (i_t, j_t) con direzione φ_c + π/2
                  e lunghezza 2σ + 1;
        (i_n, j_n) := massimo locale di Ω;
        fineNonRaggiunta := controlla i criteri di stop;
        (i_c, j_c) := (i_n, j_n);
        φ_c := O(i_c, j_c);
    }
}
```

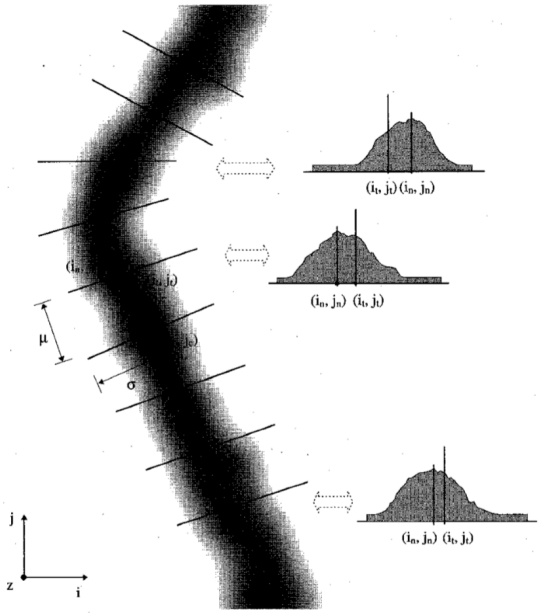


Figura 4.3.6: Alcuni passi dell'algoritmo "seguiCresta", con raffigurazione delle intensità lungo ogni sezione. Immagine tratta da [20].

L'algoritmo continua fino a che un criterio di stop non viene verificato; ad ogni passo viene calcolato il nuovo punto (i_t, j_t), spostandosi di μ pixel da (i_c, j_c) in direzione ϕ_c. Quindi viene calcolata la sezione Ω, costituita dai punti appartenenti al segmento che ha per punto medio (i_t, j_t), direzione $\phi_c + \frac{\pi}{2}$ e lunghezza $2\sigma + 1$. Successivamente viene scelto un nuovo punto (i_n, j_n) tra i punti di Ω che corrispondono a massimi locali, quindi questo appartiene sicuramente alla cresta. Prima di passare all'iterazione successiva, il punto (i_c, j_c) viene aggiornato

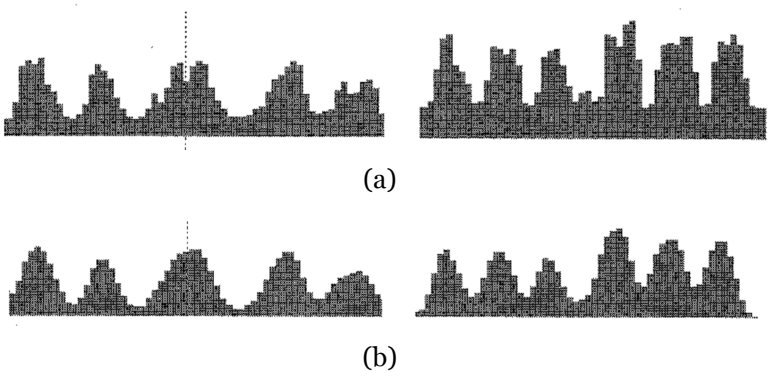
con le coordinate (i_n, j_n) e la direzione ϕ_c viene letta dall'immagine orientazione precedentemente calcolata. Per essere più precisi, indichiamo con $\Omega((i_t, j_t), \phi, \sigma)$ la sezione centrata in (i_t, j_t), con direzione $\phi + \frac{\pi}{2}$ e lunghezza $2\sigma + 1$. Questa si può definire come

$$\Omega = \{(i,j) | (i,j) \in \text{ segmento di estremi } (i_{start}, j_{start}), (i_{end}, j_{end})\} \tag{4.3.15}$$

dove

$$\begin{aligned} (i_{start}, j_{start}) &= (round(i_t - \sigma \cdot \cos\phi), round(j_t - \sigma \cdot \sin\phi)) \\ (i_{end}, j_{end}) &= (round(i_t + \sigma \cdot \cos\phi), round(j_t + \sigma \cdot \sin\phi)) \end{aligned} \tag{4.3.16}$$

Si può fare riferimento alla Figura 4.3.6 per avere un'idea di come agisce l'algoritmo.



(a)

(b)

Figura 4.3.7: (a) Due sezioni di una cresta ottenute durante l'esecuzione dell'algoritmo. (b) Regolarizzazione tramite convoluzione. La linea tratteggiata mostra il punto in cui dovrebbe essere il punto di massimo locale più vicino al centro della sezione. Immagine tratta da [20].

Il calcolo del massimo locale lungo la sezione Ω può essere eseguito semplicemente comparando i livelli di grigio, ma il rumore e la presenza di pori sudoripari possono rendere indesiderabile questa scelta (vedi Figura 4.3.7(a)). Maio e Maltoni consigliano innanzitutto di mediare i valori di sezioni parallele, ma questo abbiamo deciso di non farlo, poiché nella fase di miglioramento abbiamo già applicato un filtro gaussiano unidimensionale lungo le orientazioni locali. Successivamente gli autori consigliano di eseguire la convoluzione delle intensità lungo la sezione Ω con un lowpass filter, in modo da eliminare piccole irregolarità (vedi Figura 4.3.7(b)).

I criteri di stop che gli autori propongono sono i seguenti:

- uscita dall'area di interesse, quando il nuovo punto (i_t, j_t) si trova fuori dall'immagine;

- terminazione, ovvero quando non si riescono a trovare dei massimi locali su Ω, oppure quando i massimi locali trovati sono troppo distanti dal centro della sezione; per evitare che l'algoritmo segnali la presenza di una minutia a causa di piccole interruzioni della cresta, facciamo calcolare delle ulteriori sezioni prima di decidere se la minutia è realmente presente;
- intersezione con un'altra cresta, quando il nuovo punto è già stato etichettato come appartenente ad un'altra cresta;
- eccessiva curvatura, se il nuovo punto farebbe cambiare di molto la curvatura della cresta, allora probabilmente sono stati commessi degli errori e l'algoritmo si interrompe.

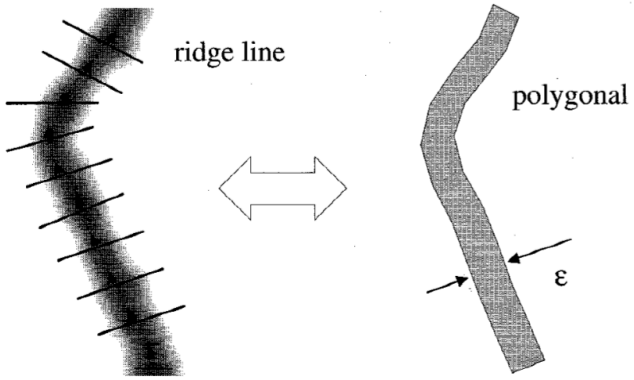


Figura 4.3.8: Una cresta e la corrispondente poligonale. Immagine tratta da [20].

Prima di vedere l'algoritmo completo per l'identificazione delle minutiae definiamo un'immagine ausiliaria T, della stessa dimensione dell'immagine di partenza, con tutti valori nulli. Questa immagine ci servirà per etichettare le creste che sono state già controllate dall'algoritmo; infatti ogni volta che viene seguita una cresta, l'insieme dei punti considerati forma una poligonale; allargando la poligonale di ε pixel, otteniamo le coordinate che bisogna etichettare con 1 nell'immagine T (si veda la Figura 4.3.8 per visualizzare questo procedimento).

Di seguito lo pseudocodice per l'individuazione delle minutiae.

```
trovaMinutia (i_s, j_s)
{
    (i_c, j_c) := massimoVicino (i_s, j_s);
    se T(i_c, j_c) = 1
        esci dalla funzione;
    altrimenti
    {
```

```
        φc = O(ic, jc);
        seguiCresta (ic, jc, φc);
        se terminazione o eccessiva curvatura
            trovata minutia ``terminazione'';
        se intersezione
            se è stata trovata precedentemente una
                        ``terminazione'' in questo punto
                elimina la ``terminazione'';
            altrimenti
                trovata minutia ``biforcazione'';
    aggiorna l'immagine T con la nuova poligonale;
        seguiCresta (ic, jc, φc + π);
        …stesse operazioni in direzione opposta
    }
}
```

La funzione `massimoVicino` serve per trovare un punto (i_c, j_c) appartenente ad una cresta, vicino al punto di partenza (i_s, j_s); questa operazione può essere eseguita calcolando la sezione passante in (i_s, j_s) e perpendicolare all'orientazione in quel punto, quindi trovare il massimo locale come nella funzione `seguiCresta`. Una precisazione va fatta per quanto riguarda i criteri di stop:

- nel caso in cui si incontra un'intersezione, possono verificarsi due casi; o c'è una vera e propria intersezione (Figura 4.3.9(b), ovvero una minutia di tipo biforcazione, oppure, come in Figura 4.3.9(c), potrebbe essere stata precedentemente trovata un'erronea minutia di tipo terminazione, che va eliminata;

- nel caso in cui l'algoritmo verifica un'eccessiva curvatura, si possono presentare due situazioni; l'eccessiva curvatura può essere dovuta alla terminazione della cresta, che l'algoritmo non ha visto ed ha proseguito con una cresta differente, per cui si deve interrompere l'algoritmo segnalando la presenza di una minutia di tipo terminazione; oppure si possono verificare situazioni in cui l'algoritmo abbandona la cresta, ovvero situazioni analoghe alla Figura 4.3.9(c), per cui conviene segnalare una terminazione ed aspettare che l'algoritmo si accorga successivamente dell'errore.

L'algoritmo `trovaMinutia` appena presentato serve ad individuare tutte le minutiae che si trovano in una cresta; per ottenere le minutiae di tutta l'immagine, dobbiamo suddividerla in piccoli blocchi, quindi far partire l'algoritmo `trovaMinutia` da ogni loro centro. Un'applicazione di questa tecnica si può vedere nella Figura 4.3.10.

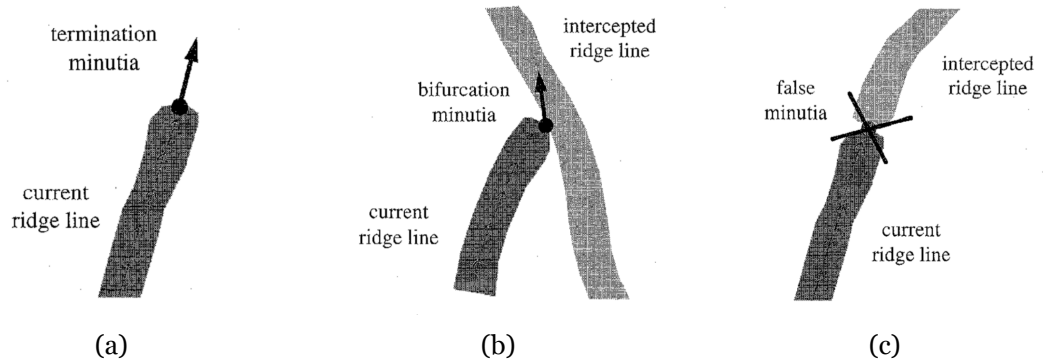


Figura 4.3.9: Esempi di interruzioni dell'algoritmo per rilevamento di minutiae. Immagine tratta da [20].

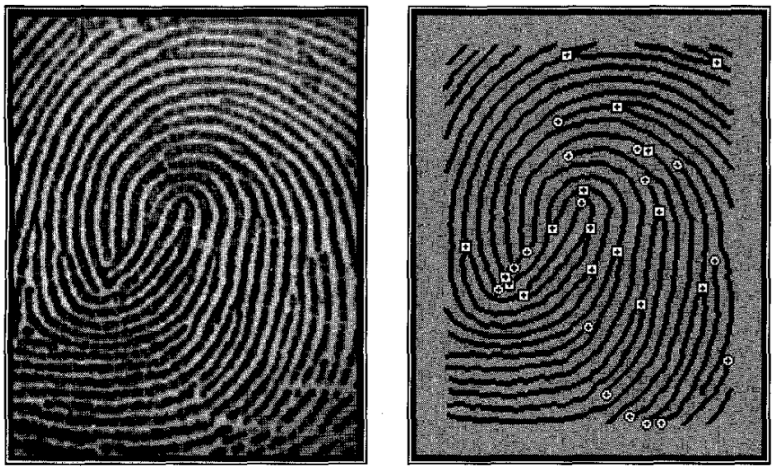


Figura 4.3.10: Esempio di individuazione di minutiae utilizzando l'algoritmo di Maio e Maltoni. A sinistra vediamo l'immagine originale, mentre a destra abbiamo l'immagine T corrispondente, ovverto la rappresentazione dell'impronta tramite le poligonali inspessite; sopra l'immagine T sono evidenziate le minutiae trovate. Immagine tratta da [20].

4.3.3 Risultati

(a) (b)

Figura 4.3.11: Immagine modificata utilizzando il metodo da noi proposto. (a) Immagine originale presa da FVC2004. (b) Immagine migliorata tramite il nostro algoritmo.

In Figura 4.3.11 possiamo apprezzare i miglioramenti apportati dal nostro software all'immagine di partenza. In questa infatti risultano molto visibili i pori sudoripari e ci sono delle zone in cui le creste non sono ben separate, nonostante complessivamente l'immagine sia buona; nell'immagine modificata questi elementi negativi vengono eliminati quasi completamente. In Figura 4.3.12 si può vedere un secondo esempio di miglioramento utilizzando il nostro algoritmo.

Per quanto riguarda l'individuazione delle minutiae, il software presenta ancora alcuni bug, che dobbiamo risolvere; per questo motivo non forniamo alcuna immagine dei risultati. Da questo e poiché manca l'algoritmo di confronto tra le minutiae, si evince che il lavoro non è ancora concluso. Per poter scrivere anche questa parte occorre del tempo: bisogna capire quale sia il migliore algoritmo da utilizzare, quindi implementarlo in MatLab.

(a) (b)

Figura 4.3.12: (a) Immagine originale non buona tratta da FVC2004. (b) Immagine modificata tramite il nostro algoritmo.

Capitolo 5

Conclusioni

Nella tesi abbiamo visto varie tecniche sia per l'elaborazione delle immagini, che per la loro analisi, con particolare riferimento alle impronte digitali. In sezione 4.3 abbiamo mostrato quale algoritmo abbiamo scelto per la creazione del nostro software d'identificazione. L'implementazione è stata eseguita in MatLab, per la sua facilità di uso e per il gran numero di funzioni messe a disposizione nel campo della visione artificiale. Il codice che abbiamo prodotto è consultabile in Appendice A.

Tale lavoro è da considerare come un primo passo verso la realizzazione di un software per l'analisi delle impronte digitali. Quindi in futuro ci prefiggiamo l'obiettivo di terminare la stesura di questo codice, migliorare ulteriormente la parte già funzionante, quindi eseguire una statistica su campioni reali di impronte digitali, per verificare quale sia il numero minimo di caratteristiche che devono corrispondere affinché si possa giudicare sicuro un matching.

Un altro interessante ambito di studio, affine agli argomenti qui presentati, è la morfogenesi delle impronte digitali. Tale indagine permetterebbe di analizzare lo sviluppo prenatale umano e le dinamiche che determinano la struttura delle impronte digitali, con la conseguente valutazione della reale potenzialità di differenziare diversi individui.

Appendice A

Codice Sorgente

```
function angle = angleFromVectors (v,w)
% Genero la funzione che calcola l'angolo compreso tra due vettori
% v  pu essere una matrice, con i vettori scritti per righe
% w deve essere della stessa dimesione di v
if ~all(size(v)==size(w))
    error('Vettori di dimensioni errate');
end
if isempty(v)||isempty(w)
    angle = [];
else
    angle = acos( sum(v.*w,2) ./ (sqrt(sum(v.^2,2)).*sqrt(sum(w.^2,2)) ) );
end
end
```

```
% Questa funzione trasforma l'input (matrice,vettore o scalare) in un
% vettore e poi ne calcola any(...)
function value = any2(mat)
value = any(reshape(mat,[1,numel(mat)]));
end
```

```
function imgOut = applyGaborFilter (img,phi,T1,T2,varargin)
% ------------------------------------------------------- %
% Imposto i parametri di default
% ------------------------------------------------------- %
freqStep = 1;
angleStep = pi/32;
sigmax = 4;
sigmay = 4;
filterSize = 5;
showSteps = false;
thresholdT = 1.5;
w = 16;
lFun = @(w) 3*w;
% ------------------------------------------------------- %
% Lettura dei parametri inseriti
% ------------------------------------------------------- %
for i = 1:2:nargin-5
    if strcmp(varargin{i},'sigmax')
        sigmax = varargin{i+1};
        if ~isnumeric(sigmax)
            error('sigmax value must be a number');
```

```
22          end
23      elseif strcmp(varargin{i},'sigmay')
24          sigmay = varargin{i+1};
25          if ~isnumeric(sigmay)
26              error('sigmay value must be a number');
27          end
28      elseif strcmp(varargin{i},'filterSize')
29          filterSize = varargin{i+1};
30          if ~isnumeric(filterSize)
31              error('filterSize value must be a number');
32          end
33      elseif strcmp(varargin{i},'center')
34          center = varargin{i+1};
35          if numel(center) ~= 2 || ~isnumeric(center)
36              error('center value must be a couple of number');
37          end
38      elseif strcmp(varargin{i},'angleStep')
39          angleStep = varargin{i+1};
40          if ~isnumeric(angleStep)
41              error('angleStep value must be a number');
42          end
43      elseif strcmp(varargin{i},'thresholdT')
44          thresholdT = varargin{i+1};
45          if ~isnumeric(thresholdT)
46              error('thresholdT value must be a number');
47          end
48      elseif strcmp(varargin{i},'freqStep')
49          freqStep = varargin{i+1};
50          if ~isnumeric(freqStep)
51              error('freqStep value must be a number');
52          end
53      elseif strcmp(varargin{i},'showSteps')
54          showSteps = varargin{i+1};
55          if ~islogical(showSteps)
56              error('showSteps value must be a logical value');
57          end
58      elseif strcmp(varargin{i},'windowSize')
59          w = varargin{i+1};
60          if ~isnumeric(w)
61              error('windowSize value must be a number');
62          end
63      elseif strcmp(varargin{i},'lFun')
64          lFun = varargin{i+1};
65      end
66  end
67
68  % ------------------------------------------------------------- %
69  % Creo le funzioni per calcolare il Gabor Filter
70  % Utilizzo i parametri trovati sopra
71  % ------------------------------------------------------------- %
72  x_phi = @(x,y,phi) x .* cos (phi) + y .* sin (phi);
73  f = @(t,T1,T2) ...
74      or( t>=0, t<=T1/4 ) .* cos(2 * pi * t ./ T1) + ... % primo ramo
75      or( t>T1/4, t<T1/4+T2/2 ) * (-1) .* cos(2 * pi * (t-T1/4-T2/4) ./ T2 ) + ... % secondo ramo
76      or( t>=T1/4+T2/2, t<=T1/2+T2/2 ) .* cos(2 * pi * (t-T1/2-T2/2) ./ T1 ); % terzo ramo
77  F = @(x,y,phi,T1,T2) ...
78      f(x_phi(x,y,phi) - floor(x_phi(x,y,phi)./(T1/2+T2/2)) .* (T1/2 + T2/2), T1, T2);
79
80  h_x = @(x,y,T1,T2,phi,sigmax) ...
81      exp ( - (x_phi(x,y,phi).^2)./(2*sigmax^2) ) .* F(x,y,phi,T1,T2);
82
83  y_phi = @(x,y,phi) - x .* sin (phi) + y .* cos (phi);
```

```
84  h_y = @(x,y,phi,sigmay) ...
85      exp ( - (y_phi(x,y,phi).^2)./(2*sigmay^2) );
86
87  G = @(x,y,phi,T1,T2,sigmax,sigmay) ...
88      h_x(x,y,T1,T2,phi,sigmax) .* h_y(x,y,phi,sigmay);
89
90  % Discretizzo T1, T2 e phi
91  % rangeT1 = max([min2(T1(T1>0)),thresholdT]):freqStep:max2(T1);
92  % rangeT2 = max([min2(T2(T2>0)),thresholdT]):freqStep:max2(T2);
93  rangeT1 = 1:freqStep:25;
94  rangeT2 = rangeT1;
95  rangePhi = 0:angleStep:2*pi;
96  [circX,circY] = meshgrid(-filterSize:filterSize);
97  disp(['Occorrono:␣',num2str(numel(rangeT1)*numel(rangeT2)*numel(rangePhi)),'␣filtri']);
98
99  % Calcolo i filtri in corrispondenza di ogni valore
100 try
101     load('prealloc/applyGaborFilterPrealloc.mat','filters');
102     disp('Caricati␣filtri');
103 catch exception
104     filters = cell(numel(rangePhi),numel(rangeT1),numel(rangeT2));
105     for i = 1:numel(rangePhi)
106         for j = 1:numel(rangeT1)
107             for k = 1:numel(rangeT2)
108                 A = G(circX,circY,rangePhi(i),rangeT1(j),rangeT2(k),sigmax,sigmay);
109                 A = (A-min2(A)) ./ (max2(A)-min2(A)); % normalizzazione
110                 filters{i,j,k} = A./sum2(A); % riscalamento
111             end
112         end
113     end
114     save('prealloc/applyGaborFilterPrealloc.mat','filters');
115     disp('Creati␣nuovi␣filtri');
116 end
117
118 if showSteps
119     figure('WindowStyle','docked');
120 end
121
122 halfFilterSize = 3*floor(filterSize/2);
123 halfWindowSize = floor(w/2);
124 l = lFun(w);
125 r = ceil(0.5 * sqrt(w^2+l^2)); % mezza diagonale del rettangolo
126 if halfWindowSize + halfFilterSize > r
127     % In questo caso si hanno problemi sul ciclo
128     error('filterSize␣is␣too␣big');
129 end
130
131 subImgIdx = repmat( (-halfWindowSize-halfFilterSize) : (halfWindowSize+halfFilterSize-1) ,2,1);
132 [subImgIdxI,subImgIdxJ] = ndgrid(subImgIdx(1,:),subImgIdx(2,:));
133 subImgIdx2 = repmat( (-halfWindowSize) : (halfWindowSize-1) ,2,1);
134 [subImgIdxI2,subImgIdxJ2] = ndgrid(subImgIdx2(1,:),subImgIdx2(2,:));
135 subImgValidIdx = repmat( (halfFilterSize+1) : (halfFilterSize+2*halfWindowSize) ,2,1);
136 [subImgValidIdxI,subImgValidIdxJ] = ndgrid(subImgValidIdx(1,:),subImgValidIdx(2,:));
137 imgOut = img;
138
139 for I = 1:size(T1,1)
140     i = (I-1)*w+r+1;
141     for J = 1:size(T1,2)
142         j = (J-1)*w+r+1;
143         currT1 = T1(I,J);
144         currT2 = T2(I,J);
145         if (currT1<thresholdT) || (currT2<thresholdT)
```

```
146             % Se T1 o T2 troppo piccoli (quindi non possono corrispondere
147             % a effettive larghezze di ridge), vai al pixel successivo
148             continue
149         end
150         currPhi = phi(i,j)+pi/2; % l'asse x del filtro deve essere lungo l'orientazione
151         [~,phiIdx] = min(abs(currPhi-rangePhi));
152         [~,T1Idx] = min(abs(currT1-rangeT1));
153         [~,T2Idx] = min(abs(currT2-rangeT2));
154         currFilter = filters{phiIdx,T1Idx,T2Idx};
155         linIdx = sub2ind(size(img),subImgIdxI+i,subImgIdxJ+j);
156         subImg = conv2(img(linIdx),currFilter,'same');
157         subLinIdx = sub2ind(size(subImg),subImgValidIdxI,subImgValidIdxJ);
158         linIdx = sub2ind(size(img),subImgIdxI2+i,subImgIdxJ2+j);
159         imgOut(linIdx) = subImg(subLinIdx);
160 
161         if showSteps
162             subplot(2,2,1);cla;
163             localImg = img(linIdx);
164             [x,y] = meshgrid(1:size(localImg,1),1:size(localImg,2));
165             [xx,yy] = meshgrid(1:0.1:size(localImg,1),1:0.1:size(localImg,2));
166             temp = interp2(x,y,localImg,xx,yy,'cubic');
167             mesh(temp);
168 
169             subplot(2,2,2);cla;
170             localImgOut = imgOut(linIdx);
171             [x,y] = meshgrid(1:size(localImgOut,1),1:size(localImgOut,2));
172             [xx,yy] = meshgrid(1:0.1:size(localImgOut,1),1:0.1:size(localImgOut,2));
173             temp = interp2(x,y,localImgOut,xx,yy,'cubic');
174             mesh(temp);
175 
176             subplot(2,2,3);cla;
177             [x,y] = meshgrid(1:size(currFilter,1),1:size(currFilter,2));
178             [xx,yy] = meshgrid(1:0.1:size(currFilter,1),1:0.1:size(currFilter,2));
179             temp = interp2(x,y,currFilter,xx,yy,'cubic');
180             mesh(temp);
181 
182             disp(num2str([currPhi,currT1,currT2]));
183             pause(0.1);
184         end
185     end
186 end
187 
188 end
```

```
1  function imgOut = autoCropImage (img,parameters)
2      % ------------------------------------------------------- %
3      % Imposto i parametri di default
4      % ------------------------------------------------------- %
5      w = 16;
6      display = 'off';
7      % ------------------------------------------------------- %
8      % Lettura dei parametri inseriti
9      % ------------------------------------------------------- %
10     if ~iscellstr(parameters)
11         error('Parameters␣must␣be␣cell␣string');
12     end
13     for i = 1:2:numel(parameters)
14         if strcmp(parameters(i),'blockWidth')
15             w = str2double(parameters(i+1));
16             if isempty(w)
17                 error('blockWidth␣value␣must␣be␣a␣number');
```

```
18              end
19          elseif strcmp(parameters(i),'display')
20              display = parameters(i+1);
21              if ~( strcmp(display,'on') || strcmp(display,'off') )
22                  error('display␣value␣must␣be␣''on''␣or␣''off''');
23              end
24          end
25      end
26
27      % Prendo come valore di riferimento il pixel in alto a sx
28      img = double(img);
29      refVal = mode(img(:));%img(1,1);
30      margins = [1, size(img,1), 1, size(img,2)]; % [t,b,l,r]
31
32      % Scorro le colonne da sinistra a destra
33      for j = 1:size(img,2)
34          col = img(:,j);
35          m = mean(col);
36          s = std(col);
37          if abs(refVal-m) <= s && s < 10
38              margins(3) = j;
39          else
40              break;
41          end
42      end
43
44      % Scorro le colonne da destra a sinistra
45      for j = size(img,2):-1:1
46          col = img(:,j);
47          m = mean(col);
48          s = std(col);
49          if abs(refVal-m) <= s && s < 10
50              margins(4) = j;
51          else
52              break;
53          end
54      end
55
56      % Scorro le righe dell'alto verso il basso
57      for i = 1:size(img,1)
58          row = img(i,:);
59          m = mean(row);
60          s = std(row);
61          if abs(refVal-m) <= s && s < 10
62              margins(1) = i;
63          else
64              break;
65          end
66      end
67
68      % Scorro le righe dal basso verso l'alto
69      for i = size(img,1):-1:1
70          row = img(i,:);
71          m = mean(row);
72          s = std(row);
73          if abs(refVal-m) <= s && s < 10
74              margins(2) = i;
75          else
76              break;
77          end
78      end
79
```

```
80      % Ritaglio immagine
81      if strcmp(display,'on')
82          disp(margins');
83      end
84      margins([1,3]) = margins([1,3]) - w * (margins([1,3])>1+w);
85      margins([2,4]) = margins([2,4]) + w * (margins([2,4])<size(img)-w);
86      imgOut = img(margins(1):margins(2),margins(3):margins(4));
87      if strcmp(display,'on')
88          figure;imshow(uint8(img));figure;imshow(uint8(imgOut));
89      end
90  end
```

```
1   function [sections,linSegment] = computeSection(img,angle,i,j,sigma)
2
3   if (length(angle) ~= length(i)) || (length(angle) ~= length(j))
4       error('angle,i,j must have the same size');
5   end
6
7   angle = angle+pi/2;
8   iStart = round(i - sigma * sin(angle));
9   jStart = round(j - sigma * cos(angle));
10  iEnd = round(i + sigma * sin(angle));
11  jEnd = round(j + sigma * cos(angle));
12  %
13  % angle = angle-pi/2;
14  % squareI = -sigma:1:sigma;
15  % squareJ = -H:1:H;
16  % [idxI,idxJ] = ndgrid(squareI,squareJ);
17  % idxI = round(cos(angle) .* idxI + sin(angle) .* idxJ + i + mu*cos(angle));
18  % idxJ = round(-sin(angle) .* idxI + cos(angle) .* idxJ + j - mu*sin(angle));
19  % linSegment = sub2ind(size(img),idxI,idxJ);
20  % section = mean(img(linSegment),2);
21  % linSegment = linSegment(:,1+H);
22
23  t = 0:0.01:1;
24  sections = cell(length(i),1);
25  linSegment = cell(length(i),1);
26
27  for k = 1:length(i)
28      iSegment = round((1-t) .* iStart(k) + t .* iEnd(k));
29      jSegment = round((1-t) .* jStart(k) + t .* jEnd(k));
30      linSegment{k} = (jSegment-1) * size(img,1) + iSegment;
31      linSegment{k} = unique(sort(linSegment{k}));
32      sections{k} = img(linSegment{k});
33  end
34
35  end
```

```
1   function xSignature = computeXSignature (img,phi,varargin)
2       % ------------------------------------------------------- %
3       % Lettura parametri di input
4       % ------------------------------------------------------- %
5       w = 16;
6       lFun = @(w) 3 * w;
7       for k = 1:2:nargin-2
8           if strcmp(varargin{k},'windowSize')
9               w = varargin{k+1};
10              if (~isnumeric(w)) && (mod(w,2)~=0)
11                  error('windowSize must be an even number');
12              end
13          elseif strcmp(varargin{k},'lFun')
```

```
14              lFun = varargin{k+1};
15              % Nessun controllo se l'input   effettivamente una funzione
16          end
17      end
18
19      filterSize = 7; % dispari
20      gaussFilter = exp( - ( (1:filterSize)-floor(filterSize/2+1) ).^2 ./ (2*(filterSize/2)^2) );
21      gaussFilter = gaussFilter./sum(gaussFilter);
22      l = lFun(w) + (filterSize-1); % aggiungo la larghezza del filtro gaussiano
23      r = ceil(0.5 * sqrt(w^2+l^2)); % mezza diagonale del rettangolo
24      rowNum = floor( (size(img,1)-2*r) / w );
25      colNum = floor( (size(img,2)-2*r) / w );
26      xSignature = zeros(rowNum,colNum,l-(filterSize-1));
27      if sum(size(img) < 2 * r) > 0
28          return
29      end
30      hh = figure('Name','Analisi␣xSignature','WindowStyle','docked');
31      xSignatureRowIdx = 1;
32      for i = (1+r):w:(size(img,1)-r) % ciclo sugli elementi interni dell'immagine
33          xSignatureColIdx = 1;
34          for j = (1+r):w:(size(img,2)-r)
35              [d,k] = meshgrid(-r:r); % da in output (x,y) => (j,i)
36              linIdx = sub2ind(size(img),k+i,d+j);
37              localImg = img(linIdx);
38              % Nella seguente formula ho cambiato il verso della rotazione
39              % in modo che il verso positivo sia in senso antiorario
40              [d0,k0] = meshgrid(-floor(l/2):floor(l/2)-1,-floor(w/2):floor(w/2)-1);
41              u = d0 .* cos (phi(i,j)+pi/2) - k0 .* sin (phi(i,j)+pi/2) + r;
42              v = d0 .* sin (phi(i,j)+pi/2) + k0 .* cos (phi(i,j)+pi/2) + r;
43              localImgInterp = interp2(d+r+1,k+r+1,localImg,u,v,'spline'); % qua ci vuole (x,y)
44              localSum = sum(localImgInterp,1);
45              localSum = localSum ./ w;
46              localSum = conv(localSum,gaussFilter,'valid');
47              xSignature(xSignatureRowIdx,xSignatureColIdx,:) = localSum;
48
49              % Visualizzazione temporanea
50              if false
51                  figure(hh);
52                  disp([i,j,phi(i,j)]);
53                  subplot(2,2,1);
54                  imshow(uint8(localImg));
55                  localImg2 = localImg;
56                  linIdx2 = sub2ind(size(localImg2),round(u)+r,round(v)+r);
57                  localImg2(linIdx2) = 0;
58                  subplot(2,2,3);
59                  imshow(uint8(localImg2));
60                  xsign = reshape(xSignature(xSignatureRowIdx,xSignatureColIdx,:),[l,1]);
61                  subplot(2,2,2);
62                  plot(xsign);
63                  subplot(2,2,4);
64                  showOrientation(hh,localImg,phi(linIdx));
65                  pause;
66              end
67
68              % Incremento l'indice di colonna
69              xSignatureColIdx = xSignatureColIdx+1;
70          end
71          % Incremento l'indice di riga
72          xSignatureRowIdx = xSignatureRowIdx+1;
73      end
74
75      close;
```

```
76  end
```

```
1   function  cylCoordsMaker (varargin)
2
3   % Imposto i parametri di default
4   maxAngle = 2*pi;
5   maxWidth = 30;
6   maxLength = 40;
7   angleStep = pi/16;
8   widthStep = 1;
9   lengthStep = 1;
10  showCompletionParam = true;
11  completionStep = .1;
12  savePath = '../prealloc/';
13  saveName = 'cylCoordsPrealloc.mat';
14  % Controllo l'input ed eventualmente modifico i parametri
15  for k = 1:2:nargin
16      if strcmp(varargin{k},'maxAngle')
17          maxAngle = varargin{k+1};
18          if ~isnumeric(maxAngle)
19              error('maxAngle␣must␣be␣a␣number');
20          end
21      elseif strcmp(varargin{k},'maxWidth')
22          maxWidth = varargin{k+1};
23          if ~isnumeric(maxWidth)
24              error('maxWidth␣must␣be␣a␣number');
25          end
26      elseif strcmp(varargin{k},'maxLength')
27          maxLength = varargin{k+1};
28          if ~isnumeric(maxLength)
29              error('maxLength␣must␣be␣a␣number');
30          end
31      elseif strcmp(varargin{k},'angleStep')
32          angleStep = varargin{k+1};
33          if ~isnumeric(angleStep)
34              error('angleStep␣must␣be␣a␣number');
35          end
36      elseif strcmp(varargin{k},'widthStep')
37          widthStep = varargin{k+1};
38          if ~isnumeric(widthStep)
39              error('widthStep␣must␣be␣a␣number');
40          end
41      elseif strcmp(varargin{k},'lengthStep')
42          lengthStep = varargin{k+1};
43          if ~isnumeric(lengthStep)
44              error('lengthStep␣must␣be␣a␣number');
45          end
46      elseif strcmp(varargin{k},'completionStep')
47          completionStep = varargin{k+1};
48          if ~isnumeric(completionStep)
49              error('completionStep␣must␣be␣a␣number');
50          end
51      elseif strcmp(varargin{k},'showCompletion')
52          showCompletionParam = varargin{k+1};
53          if ~islogical(showCompletionParam)
54              error('showCompletion␣must␣be␣a␣logical␣value');
55          end
56      elseif strcmp(varargin{k},'savePath')
57          savePath = varargin{k+1};
58          if ~ischar(savePath)
59              error('savePath␣must␣be␣a␣string');
```

```
60          end
61      elseif strcmp(varargin{k},'saveName')
62          saveName = varargin{k+1};
63          if ~ischar(saveName)
64              error('saveName␣must␣be␣a␣string');
65          end
66      end
67  end
68
69  angles = 0:angleStep:maxAngle;
70  widths = 1:widthStep:maxWidth; % da 1 alla larghezza massima delle creste
71  lengths = 1:lengthStep:maxLength; % da 1 al massimo valore che si vuole dare a mu
72  cylCoords = cell(numel(angles),numel(widths),numel(lengths),2);
73  cylParamStruct = struct(...
74      'maxAngle',maxAngle,'maxWidth',maxWidth,'maxLength',maxLength,...
75      'angleStep',angleStep,'widthStep',widthStep,'lengthStep',lengthStep);
76
77  p00 = [0,0]; % pongo lo zero nell'origine
78
79  if showCompletionParam
80      % Mostro all'utente il numero complessivo di iterazioni
81      oldIter = 0;
82      totIter = numel(cylCoords)/2;
83      disp(['Numero␣totale␣iterazioni:␣',num2str(totIter)]);
84  end
85
86  for i = 1:numel(angles)
87      for j = 1:numel(widths)
88          for k = 1:numel(lengths)
89              % Recupero i valori dei parametri del cilindretto
90              angle = angles(i);
91              width = widths(j);
92              length = lengths(k);
93              p01 = p00 + length * [cos(angle),sin(angle)];
94
95              % Calcolo tre vertici del rettangolo che  sezione del
96              % cilindretto
97              p1 = p00 + width * [cos(angle+pi/2),sin(angle+pi/2)];
98              p2 = p00 + width * [cos(angle-pi/2),sin(angle-pi/2)];
99              p3 = p01 + width * [cos(angle+pi/2),sin(angle+pi/2)];
100
101             % Assumo che il numero di punti sufficiente per un'ottima
102             % definizione del cilindretto sia 5 volte il massimo tra la
103             % larghezza e l'altezza del cilindretto
104             np = ceil(5 * max([width,length]));
105
106             % Genero la sezione del cilindretto e poi la perturbo al centro
107             % in modo da arrotondarne le  estremit
108             t = linspace(0,1,np);
109             s = linspace(0,1,np);
110 %             theta = linspace(0,pi,np);
111 %             spany = linspace(-1,1,np);
112             [t,s] = ndgrid(t,s);
113 %             [spany,~] = ndgrid(spany);
114 %             [~,circy] = ndgrid(sin(theta));
115 %             t = t + width.*spany.*circy./norm(p3-p1);
116
117             % Genero le coordinate cartesiane del cilindretto
118             px = round( p1(1) + s .* (p2(1)-p1(1)) + t .* (p3(1)-p1(1)) );
119             py = round( p1(2) + s .* (p2(2)-p1(2)) + t .* (p3(2)-p1(2)) );
120
121             % Traslo il cilindretto al centro di un'immagine che
```

```
            % sicuramente lo contenga (quadrata di lato length+2*width+2)
            % Questa operazione serve solamente per eliminare gli indici
            % corrispondenti allo stesso punto; per questo motivo gli
            % indici vanno riportati all'origine degli assi cartesiani
            imgSize = 2*length+4*width;
            px = px + round(imgSize/2);
            py = py + round(imgSize/2);

            % Genero le coordinate lineari del cilindretto, da cui riesco
            % ad eliminare i doppioni
            linIdx = sub2ind(imgSize*[1,1],px,py);
            linIdx = unique(sort(linIdx));

            % Torno alle coordinate cartesiane attorno all'origine
            [npx,npy] = ind2sub(imgSize*[1,1],linIdx);
            npx = npx - round(imgSize/2);
            npy = npy - round(imgSize/2);

            % Metto le coordinate trovate nel cellArray cylCoords
            cylCoords{i,j,k,1} = npx;
            cylCoords{i,j,k,2} = npy;

            % Mostro all'utente la percentuale di completamento
            % dell'operazione
            currIter = k + numel(lengths) * ( (j-1) + numel(widths) * (i-1) );
            oldIter = showCompletion(oldIter,currIter,totIter,.1,'Percentuale␣completamento:␣&&&␣%');
        end
    end
end

completePath = [savePath,saveName];
try
    save(completePath,'cylCoords','cylParamStruct');
catch exception
    [saveName,savePath] = uiputfile('*.*','Scegliere␣il␣file␣di␣destinazione',saveName);
    completePath = [savePath,saveName];
    save(completePath,'cylCoords','cylParamStruct');
end

if showCompletionParam
    disp(['Salvataggio␣completato␣con␣successo␣alla␣posizione␣',completePath]);
end

end
```

```
function [px,py] = cylCoordsRecover (p00,p01,width,cylCoords,cylParamStruct)

% Carico i parametri da file
maxAngle = cylParamStruct.maxAngle;
maxWidth = cylParamStruct.maxWidth;
maxLength = cylParamStruct.maxLength;
angleStep = cylParamStruct.angleStep;
widthStep = cylParamStruct.widthStep;
lengthStep = cylParamStruct.lengthStep;

% Creo gli array con tutti i valori di angle,width e length
angles = 0:angleStep:maxAngle;
widths = 1:widthStep:maxWidth; % da 1 alla larghezza massima delle creste
lengths = 1:lengthStep:maxLength; % da 1 al massimo valore che si vuole dare a mu
% Recupero i valori che mi servono
angle = angleFromVectors([1,0],(p01-p00));
```

```
length = norm(p01-p00);
% Recupero l'indice dei valori che mi servono
[~,angleIdx] = min(abs(angle-angles));
[~,widthIdx] = min(abs(width-widths));
[~,lengthIdx] = min(abs(length-lengths));
% Recupero le coordinate del cilindretto riferite all'origine
px = cylCoords{angleIdx,widthIdx,lengthIdx,1};
py = cylCoords{angleIdx,widthIdx,lengthIdx,2};
% Traslo tali coordinate sul punto p00
px = px + p00(1);
py = py + p00(2);

end
```

```
classdef exitCondition_
    enumeration
        exitArea,termination,intersection,bending,none
    end
end
```

```
function minutiae = findMinutiae (img,phi,varargin)
% Lettura parametri di input
mu = 3;
sigma = 7;
thresholdStd = std2(img)./7;
showStep = false;
loadPath = './prealloc/';
loadName = 'cylCoordsPrealloc.mat';
for k = 1:2:nargin-2
    if strcmp(varargin{k},'mu')
        mu = varargin{k+1};
        if ~isnumeric(mu)
            error('mu must be a number');
        end
    elseif strcmp(varargin{k},'sigma')
        sigma = varargin{k+1};
        if ~isnumeric(sigma)
            error('sigma must be a number');
        end
    elseif strcmp(varargin{k},'showStep')
        showStep = varargin{k+1};
        if ~islogical(showStep)
            error('showStep myst be a logical value');
        end
    elseif strcmp(varargin{k},'loadPath')
        loadPath = varargin{k+1};
        if ~ischar(loadPath)
            error('loadPath must be a string');
        end
    elseif strcmp(varargin{k},'loadName')
        loadName = varargin{k+1};
        if ~ischar(loadName)
            error('loadName must be a string');
        end
    end
end

% Faccio in modo che le creste siano sui massimi
img = 255-img;
% Traccio delle linee immaginarie verticali e prendo i massimi locali come
% punti di partenza per l'algoritmo di Maio&Maltoni
```

```
42  step = 16;
43  is = [];
44  js = [];
45  for j = step:step:size(img,2)-step
46      line = img(:,j);
47      pp0 = spline(1:length(line),line);
48      pp1 = ppder(pp0);
49      pp2 = ppder(pp1);
50      roots1 = pproots(pp1);
51      roots1 = roots1(imag(roots1)==0);
52      der2val = ppval(pp2,roots1);
53      der2val = round(der2val .* 1e4) .* 1e-4; % arrotondamento 4° cifra
54      roots1 = roots1(der2val<0);
55      maxIdx = unique(sort(round(roots1)));
56      maxIdx = maxIdx( and(maxIdx > step,maxIdx < size(img,1)-step) );
57      is = [is;maxIdx];
58      js = [js;outerRep(j,length(maxIdx))];
59  end
60  % Elimino i punti di partenza che non hanno una deviazione standard
61  % sufficientemente elevata tra i punti nel loro intorno
62  kernel = ones(16);
63  meanMatrix = conv2(img,kernel,'same')./numel(kernel);
64  stdMatrix = sqrt(conv2( (img-meanMatrix).^2 ,kernel,'same')./numel(kernel));
65  linStd = sub2ind(size(stdMatrix),is,js);
66  stdCondition = stdMatrix(linStd) > thresholdStd;
67
68  % Temp. visualizzazione
69  if showStep%||true
70      img2 = zeros(size(img,1),size(img,2),3);
71      img2(:,:,1) = 255-img;
72      img2(:,:,2) = 255-img;
73      img2(:,:,3) = 255-img;
74      linStd2 = sub2ind(size(img2),is(stdCondition),js(stdCondition),...
75          1*ones(size(is(stdCondition))));
76      img2(linStd2) = 0; % R
77      linStd2 = sub2ind(size(img2),is(stdCondition),js(stdCondition),...
78          2*ones(size(is(stdCondition))));
79      img2(linStd2) = 255; % G
80      linStd2 = sub2ind(size(img2),is(stdCondition),js(stdCondition),...
81          3*ones(size(is(stdCondition))));
82      img2(linStd2) = 0; % B
83
84      linStd2 = sub2ind(size(img2),is(~stdCondition),js(~stdCondition),...
85          1*ones(size(is(~stdCondition))));
86      img2(linStd2) = 255; % R
87      linStd2 = sub2ind(size(img2),is(~stdCondition),js(~stdCondition),...
88          2*ones(size(is(~stdCondition))));
89      img2(linStd2) = 0; % G
90      linStd2 = sub2ind(size(img2),is(~stdCondition),js(~stdCondition),...
91          3*ones(size(is(~stdCondition))));
92      img2(linStd2) = 0; % B
93      figure('WindowStyle','docked');
94      imshow(uint8(img2),'InitialMagnification','fit');
95  end
96
97  is = is(stdCondition);
98  js = js(stdCondition);
99
100 % Inizializzo la matrice che  terr traccia delle linee  gi esaminate
101 alreadyChecked = false(size(img));
102
103 % Inizilizzo l'elenco delle minutiae con il relativo indice
```

```
104 minutiaeIdx = 1;
105 nm = floor(size(img,1) * size(img,2) / 16);
106 minutiae = struct('pos',zeros(nm,2),'angle',zeros(nm,1),...
107     'exitCondition',zeros(nm,1));
108
109 % Carico da file la struttura dati contenente le coordinate dei cilindretti
110 completePath = [loadPath,loadName];
111 try
112     load(completePath,'cylCoords','cylParamStruct');
113 catch exception
114     [loadName,loadPath] = uigetfile('*.*','Scegliere␣il␣file␣di␣destinazione');
115     completePath = [loadPath,loadName];
116     load(completePath,'cylCoords','cylParamStruct');
117 end
118
119 % Mostro all'utente il numero complessivo di iterazioni
120 totIter = numel(is);
121 oldIter = 0;
122 disp(['Numero␣totale␣punti:␣',num2str(totIter)]);
123
124 % Ciclo su tutti gli elementi trovati
125 for k = 1:numel(is)
126     % Ricerca del massimo  pi vicino al punto iniziale
127     phic = phi(is(k),js(k));
128     [ic,jc] = nearestRidgeLineMaximum(img,is(k),js(k),phic,...
129         'mu',mu,'sigma',sigma);
130
131     % Controllo se la linea   gi stata controllata
132     if ~alreadyChecked(ic,jc)
133         alreadyChecked2 = alreadyChecked;
134         [minutia,alreadyChecked] = ridgeLineFollowing(img,phi,...
135             alreadyChecked,ic,jc,cylCoords,cylParamStruct,...
136             'mu',mu,'sigma',sigma,'showStep',showStep);
137         if (minutia.exitCondition ~= exitCondition_.none)&&false
138             showMinutiae(255-img,minutia);
139             ridgeLineFollowing(img,phi,alreadyChecked2,ic,jc,cylCoords,...
140                 cylParamStruct,'mu',mu,'sigma',sigma,'showStep',true);
141         end
142         if (minutia.exitCondition == exitCondition_.bending) || ...
143                 (minutia.exitCondition == exitCondition_.termination)
144             minutiae(minutiaeIdx) = minutia;
145             minutiaeIdx = minutiaeIdx+1;
146         elseif minutia.exitCondition == exitCondition_.intersection
147             % Potrebbe esistere una biforcazione se non ci sono altre
148             % minutiae in corrispondenza
149             nearMinutiae = false;
150             detectedNumber = -1;
151             for s = 1:numel(minutiae)
152                 nearMinutiae = nearMinutiae || ...
153                     (sum((minutia.pos-minutiae(s).pos).^2)<=4^2); % 2*width=4
154                 if nearMinutiae
155                     detectedNumber = s;
156                     break;
157                 end
158             end
159
160             if nearMinutiae % Se c'era una minutia nello stesso posto la cancello
161                 minutiae(detectedNumber:minutiaeIdx-2) = minutiae(detectedNumber+1:minutiaeIdx-1);
162                 minutiaeIdx = minutiaeIdx-1;
163                 minutiae{minutiaeIdx}.oos = [-1,-1];
164                 minutiae{minutiaeIdx}.angle = 0;
165                 minutiae{minutiaeIdx}.exitCondition = exitCondition_.none;
```

```
166             else % Se la biforcazione  valida aggiungo la minutia all'elenco
167                 minutiae(minutiaeIdx) = minutia;
168                 minutiaeIdx = minutiaeIdx+1;
169             end
170         end
171 
172         % Eseguo le stesse operazioni in direzione opposta
173         alreadyChecked2 = alreadyChecked;
174         [minutia,alreadyChecked] = ridgeLineFollowing(img,phi+pi,...
175             alreadyChecked,ic,jc,cylCoords,cylParamStruct,...
176             'mu',mu,'sigma',sigma,'showStep',showStep);
177         if (minutia.exitCondition ~= exitCondition_.none)&&false
178             showMinutiae(255-img,minutia);
179             ridgeLineFollowing(img,phi,alreadyChecked2,ic,jc,cylCoords,...
180                 cylParamStruct,'mu',mu,'sigma',sigma,'showStep',true);
181         end
182         minutia.angle = minutia.angle - pi;
183         if (minutia.exitCondition == exitCondition_.bending) || ...
184                 (minutia.exitCondition == exitCondition_.termination)
185             minutiae(minutiaeIdx) = minutia;
186             minutiaeIdx = minutiaeIdx+1;
187         elseif minutia.exitCondition == exitCondition_.intersection
188             % Potrebbe esistere una biforcazione se non ci sono altre
189             % minutiae in corrispondenza
190             nearMinutiae = false;
191             detectedNumber = -1;
192             for s = 1:numel(minutiae)
193                 nearMinutiae = nearMinutiae || ...
194                     (sum((minutia.pos-minutiae(s).pos).^2)<=4^2); % 2*width=4
195                 if nearMinutiae
196                     detectedNumber = s;
197                     break;
198                 end
199             end
200 
201             if nearMinutiae % Se c'era una minutia nello stesso posto la cancello
202                 minutiae(detectedNumber:minutiaeIdx-2) = minutiae(detectedNumber+1:minutiaeIdx-1);
203                 minutiaeIdx = minutiaeIdx-1;
204                 minutiae{minutiaeIdx}.oos = [-1,-1];
205                 minutiae{minutiaeIdx}.angle = 0;
206                 minutiae{minutiaeIdx}.exitCondition = exitCondition_.none;
207             else % Se la biforcazione  valida aggiungo la minutia all'elenco
208                 minutiae(minutiaeIdx) = minutia;
209                 minutiaeIdx = minutiaeIdx+1;
210             end
211         end
212     end
213 
214     % Mostro all'utente la percentuale di completamento e lo stato di
215     % avanzamento
216     currIter = k;
217     oldIter = showCompletion(oldIter,currIter,totIter,1,'Percentuale␣completamento:␣&&&␣%');
218 end
219 
220 end
```

```
1  function phi = findOrientation (img,parameters)
2      % ------------------------------------------------------ %
3      % Lettura dei parametri inseriti
4      % ------------------------------------------------------ %
5      if ~iscellstr(parameters)
```

```
6           error('Parameters␣must␣be␣cell␣string');
7       end
8       subSampling = true;
9       W = 16;
10      filterBlockWidth = 5;
11      filterSigma = 0.7;
12      for i = 1:2:numel(parameters)
13          if strcmp(parameters(i),'subSampling')
14              if strcmp(parameters(i+1),'true')
15                  subSampling = true;
16              elseif strcmp(parameters(i+1),'false')
17                  subSampling = false;
18              else
19                  error('subSampling␣value␣must␣be␣either␣''true''␣or␣''false''');
20              end
21          elseif strcmp(parameters(i),'filterBlockWidth')
22              filterBlockWidth = str2double(parameters(i+1));
23              if isempty(filterBlockWidth)
24                  error('The␣value␣of␣the␣''filterBlockWidth''␣paremeter␣must␣be␣number');
25              end
26          elseif strcmp(parameters(i),'filterSigma')
27              filterSigma = str2double(parameters(i+1));
28              if isempty(filterSigma)
29                  error('The␣value␣of␣the␣''filterSigma''␣paremeter␣must␣be␣number');
30              end
31          elseif strcmp(parameters(i),'blockWidth')
32              W = str2double(parameters(i+1));
33              if isempty(W);
34                  error('blockWidth␣value␣must␣be␣either␣a␣number');
35              end
36          end
37      end
38
39      % ------------------------------------------------------- %
40      % Verifica  compatibilit dimensioni
41      % ------------------------------------------------------- %
42      if sum(size(img) <= W) >= 1
43          return
44      end
45
46      % ------------------------------------------------------- %
47      % Algoritmo di calcolo dell'orientazione
48      % ------------------------------------------------------- %
49      % Calcola i gradienti dell'immagine
50      [gx,gy] = gradient(img);
51
52      % Calcola 2*gx*gy e gx^2-gy^2
53      matNum = 2 * gx .* gy;
54      matDen = gx.^2 - gy.^2;
55      zeroGradientMask = and(gx==0,gy==0);
56
57      % Calcola l'orientazione
58      kernel = ones(W);
59      num = conv2(matNum,kernel,'same');
60      den = conv2(matDen,kernel,'same');
61      clear matNum matDen;
62
63      % Sampling dei valori ottenuti (prendo l'orientazione del
64      % punto centrale del blocco, come scritto in Yang et al.)
65      if subSampling
66          iIdx = 1:W:size(img,1);
67          jIdx = 1:W:size(img,2);
```

```
68          num = num(iIdx,jIdx);
69          den = den(iIdx,jIdx);
70      end
71      phi = 0.5 * atan2(num,den);
72      phi(zeroGradientMask) = 0;
73      k = 0.5 * or( and(phi<0,num<0) , and(phi>=0,num>0) ) + ...
74          1 * and(phi<0,num>=0) + ...
75          0 * and(phi>=0,num<=0);
76      phi = phi + k .* pi;
77      clear num den;
78
79
80      % ------------------------------------------------------- %
81      % Algoritmo per il filtro passa basso (smoothing)
82      % ------------------------------------------------------- %
83      Phi_x = cos (2 * phi);
84      Phi_y = sin (2 * phi);
85      gaussianFilter = fspecial('gaussian',filterBlockWidth,filterSigma);
86      Phi_x = conv2(Phi_x,gaussianFilter,'same');
87      Phi_y = conv2(Phi_y,gaussianFilter,'same');
88      phi = 0.5 * atan2 (Phi_y , Phi_x);
89  end
```

```
1   % ---------------------------------------------------------------- %
2   % Pulizia finestra di controllo e variabili
3   % ---------------------------------------------------------------- %
4   clear;
5   clc;
6   close all;
7   addpath('./matFun','./fun','./findMinutiae','./prealloc');
8
9   % ---------------------------------------------------------------- %
10  % Lettura immagine da file
11  % ---------------------------------------------------------------- %
12  disp('Lettura␣immagine␣da␣file...');
13  imgPath = './img/';
14  imgName = '101_2.tif';
15  img = imread([imgPath,imgName]);
16
17  % ---------------------------------------------------------------- %
18  % Conversione immagine in scala di grigi (inutile)
19  % ---------------------------------------------------------------- %
20  if numel(size(img))==3
21      img = rgb2gray(img);
22  end
23  img = double(img);
24  disp('OK');
25
26  % ---------------------------------------------------------------- %
27  % Ritaglio automatico immagine
28  % ---------------------------------------------------------------- %
29  disp('Ritaglio␣automatico␣...');
30  autoCropImageParams = {'display','off'};
31  img = autoCropImage(img,autoCropImageParams);
32  allNames{1} = 'Immagine␣ritagliata';
33  allImg{1} = img;
34  disp('OK');
35
36  % ---------------------------------------------------------------- %
37  % Definizione funzione di riscalamento
38  % ---------------------------------------------------------------- %
```

```
39 scale0_255 =@(img) (img-min(img(:))) ./ (max(img(:))-min(img(:))) .* 255;
40
41 % ---------------------------------------------------------------- %
42 % Filtro mediano
43 % ---------------------------------------------------------------- %
44 % disp('Applicazione filtro mediano...');
45 % r = 3;
46 % h = floor(r/2);
47 % linMask = -h:h;
48 % for i = 1+h:size(img,1)-h
49 %     for j = 1+h:size(img,2)-h
50 %         localImg = img(i+linMask,j+linMask);
51 %         img(i,j) = median(localImg(:));
52 %     end
53 % end
54 % allImg{numel(allImg)+1} = img;
55 % allNames{numel(allNames)+1} = 'Filtro mediano';
56 % disp('OK');
57
58 % ---------------------------------------------------------------- %
59 % Immagine migliorata col gradiente
60 % ---------------------------------------------------------------- %
61 % disp('Miglioramento immagine con gradiente...');
62 % [gx,gy] = gradient(img);
63 % filterGrad = fspecial('average',5);
64 % img = img + conv2(abs(gx) + abs(gy),filterGrad,'same');
65 % allImg{numel(allImg)+1} = img;
66 % allNames{numel(allNames)+1} = 'Gradiente';
67 % disp('OK');
68
69 % ---------------------------------------------------------------- %
70 % Riscalamento dell'immagine
71 % ---------------------------------------------------------------- %
72 % disp('Riscalamento immagine...');
73 % img = scale0_255(img);
74 % allImg{numel(allImg)+1} = img;
75 % allNames{numel(allNames)+1} = 'Riscalamento';
76 % disp('OK');
77
78 % ---------------------------------------------------------------- %
79 % Equalizzazione dell'istogramma
80 % ---------------------------------------------------------------- %
81 % disp('Equalizzazione dell''istogramma...');
82 % img = histEqualization(img);
83 % allImg{numel(allImg)+1} = img;
84 % allNames{numel(allNames)+1} = 'Eq. Istogramma';
85 % disp('OK');
86
87 % ---------------------------------------------------------------- %
88 % Normalizzazione dell'immagine
89 % ---------------------------------------------------------------- %
90 disp('Normalizzazione␣immagine␣...');
91 desiredMean = 100;
92 desiredStd = 10;
93 img = imgNormalization(img,'mean',desiredMean,'variance',desiredStd^2);
94 allImg{numel(allImg)+1} = img;
95 allNames{numel(allNames)+1} = 'Immagine␣normalizzata';
96 disp('OK');
97
98 % ---------------------------------------------------------------- %
99 % Calcolo orientazioni per tutta l'immagine
100 % ---------------------------------------------------------------- %
```

```
disp('Calcolo orientazioni ...');
findOrientationParams = {...
    'subSampling';'false';...
    'blockWidth';'8';...
    'filterBlockWidth';'7';
    'filterSigma';'3'};
phi = findOrientation(img,findOrientationParams);
disp('OK');

% ---------------------------------------------------------------- %
% Filtro gaussiano 1-dimensionale
% ---------------------------------------------------------------- %
disp('Applico filtro gaussiano 1D ...');
gaussianFilter1DParams = {'sigma','1','filterSize','7'};
img = gaussianFilter1D(img,phi,gaussianFilter1DParams);
allImg{numel(allImg)+1} = img;
allNames{numel(allNames)+1} = 'Filtro gaussiano 1D';
disp('OK');

% ---------------------------------------------------------------- %
% Rapida segmentazione dell'immagine
% ---------------------------------------------------------------- %
% disp('Segmentazione ...');
% % SegmentationRidgeValleysParams = {'delta';'0.2';'blockWidth';'8'};
% % ridgeLocations = SegmentationRidgeValleys (img,SegmentationRidgeValleysParams);
% ridgeLocations = segmentationHessian(img,phi,'d',0.2,'w',8);
% allImg{numel(allImg)+1} = (~ridgeLocations).*255;
% allNames{numel(allNames)+1} = 'Segmentazione';
% disp('OK');

% ---------------------------------------------------------------- %
% Ridge Frequency Image
% ---------------------------------------------------------------- %
disp('Ridge Frequency Image ...');
windowSize = 16;
lFun = @(w) 2*w;
xSignature = computeXSignature (img,phi,'windowSize',windowSize,'lFun',lFun);
disp('OK');

disp('Calcolo ampiezza valli e creste ...');
[T1, T2] = ridgeValleyWidth (xSignature,img,phi,...
    'stdThreshold',0.1*desiredStd,'derThreshold',0.05*desiredStd,...
    'windowSize',windowSize,'lFun',lFun);
disp('OK');

% ---------------------------------------------------------------- %
% Applico il gabor filter
% ---------------------------------------------------------------- %
disp('Applying Gabor Filter ...');
img = applyGaborFilter(img,phi,T1,T2,'sigmax',4,'sigmay',4,'showSteps',...
    false,'windowSize',windowSize,'lFun',lFun);
img = scale0_255(img);
allImg{numel(allImg)+1} = img;
allNames{numel(allNames)+1} = 'Gabor Filter';
disp('OK');

% ---------------------------------------------------------------- %
% Miglioramento  visibilit
% ---------------------------------------------------------------- %
% disp('Miglioramento  visibilit...');
% img = (img-min2(img)) ./ (max2(img)-min2(img)) .* 65535;
% img = histEqualization(img,'bit',16,'mode','local','w',16);
```

```
% img = histCut(img);
% img = scale0_255(img);
% figure;
% histogram(img);
% allImg{numel(allImg)+1} = img;
% allNames{numel(allNames)+1} = 'Aum. Contrasto';
% disp('OK');

% ------------------------------------------------------------- %
% Visualizzazione risultati
% ------------------------------------------------------------- %
hh = figure('WindowStyle','docked');
n = numel(allImg);
nc = ceil(sqrt(n));
if nc*(nc-1) >= n
    nr = nc-1;
else
    nr = nc;
end
for k = 1:n
    subplot(nr,nc,k);
    hold on;
    imshow(uint8(allImg{k}),'InitialMagnification','fit');
    title(allNames{k});
end


% Eseguo l'algoritmo di maio-maltoni per il tracciamento delle creste e
% l'individuazione delle minutiae
disp('Tracciamento delle creste e individuazione minutiae...');
minutiae = findMinutiae (img,phi,'showStep',true);
% Visualizzazione
showMinutiae(img,minutiae);
disp('OK');
```

```
function imgOut = gaussianFilter1D(img,phi,parameters)
% ------------------------------------------------------- %
% Imposto i parametri di default
% ------------------------------------------------------- %
sigma = 1.2;
filterSize = 3;
center = [0,0];
angleStep = pi/32;
% ------------------------------------------------------- %
% Lettura dei parametri inseriti
% ------------------------------------------------------- %
if ~iscellstr(parameters)
    error('Parameters must be cell string');
end
for i = 1:2:numel(parameters)
    if strcmp(parameters(i),'sigma')
        sigma = str2double(parameters(i+1));
        if isempty(sigma)
            error('sigma value must be a number');
        end
    elseif strcmp(parameters(i),'filterSize')
        filterSize = str2double(parameters(i+1));
        if isempty(filterSize)
            error('filterSize value must be a number');
        end
    elseif strcmp(parameters(i),'center')
```

```
27          center = str2double(parameters(i+1));
28          if numel(center) ~= 2 || ~isnumeric(center)
29              error('mu␣value␣must␣be␣a␣couple␣of␣number');
30          end
31      elseif strcmp(parameters(i),'angleStep')
32          angleStep = str2double(parameters(i+1));
33          if isempty(angleStep)
34              error('angleStep␣value␣must␣be␣a␣number');
35          end
36      end
37  end
38
39  % Definisco una gaussiana bidimensionale per creare il set di filtri
40  xphi = @(x,y,phi) cos(phi) .* x - sin(phi) .* y;
41  yphi = @(x,y,phi) sin(phi) .* x + cos(phi) .* y;
42  g = @(x,y,phi) exp(- ( (xphi(x,y,phi)-center(1)).^2./(2*sigma^2) + ...
43      (yphi(x,y,phi)-center(2)).^2./(2*0.01^2) ) );
44
45  % Calcolo gli indici di una regione circolare centrata nell'origine,
46  %  servir per calcolare gli indici dell'intorno di ogni punto da
47  % elaborare
48  [circX,circY] = meshgrid(-filterSize:1:filterSize);
49  condition = (circX.^2 + circY.^2) <= filterSize^2;
50  circX = circX(condition);
51  circY = circY(condition);
52
53  % Creo un vettore di angoli che  servir per indicizzare i filtri
54  % e per calcolarne i valori
55  angles = 0:angleStep:(2*pi-angleStep);
56
57  % Costruisco un array di filtri, se non   gi in memoria
58  try
59      load('prealloc/gaussianFilter1DPrealloc.mat','filters');
60      disp('Caricati␣filtri');
61  catch exception
62      filters = cell(1,numel(angles));
63      for i = 1:numel(filters)
64          filters{i} = g(circX,circY,angles(i));
65          filters{i} = filters{i}./sum2(filters{i});
66      end
67      save('prealloc/gaussianFilter1DPrealloc.mat','filters');
68      disp('Creati␣nuovi␣filtri');
69  end
70
71  % Definisco una funzione da applicare all'immagine per ottimizzare
72  % il filtro
73  imgFun = @(t) (t.^2) .* sign(t);
74  imgFunInv = @(t) sqrt(abs(t)) .* sign(t);
75
76  % Rendo i valori di creste e valli simmetrici rispetto allo zero,
77  %  cosicch t.^2 funziona meglio per separarli
78  initialMean = mean2(img);
79  imgOut = img - initialMean;
80
81  % Scorro tutti i punti dell'immagine e le applico il filtro
82  for i = (1+filterSize):(size(img,1)-filterSize)
83      for j = (1+filterSize):(size(img,2)-filterSize)
84          linIdx = size(img,1).*(j+circX-1)+(i+circY);
85          [~,angleIdx] = min(abs(phi(i,j)-angles));
86          imgOut(i,j) = imgFunInv( sum( imgFun(imgOut(linIdx)) .* filters{angleIdx} ) );
87      end
88  end
```

```

% A questo punto l'immagine presenta una cornice di ampiezza halfFS,
% per toglierla assegno a tutti i punti della cornice il valore di
% grigio del punto (1+halfFS,1+halfFS)
value = imgOut(1+filterSize,1+filterSize);
imgOut(1:filterSize,:) = value;
imgOut(size(imgOut,1)-filterSize:size(imgOut,1),:) = value;
imgOut(:,1:filterSize) = value;
imgOut(:,size(imgOut,2)-filterSize:size(imgOut,2)) = value;

% Riporto l'immagine ad avere la media che aveva all'inizio
imgOut = imgOut + initialMean;
end
```

```
function outImg = imgNormalization(img,varargin)
    % ------------------------------------------------------- %
    % Imposto i parametri di default
    % ------------------------------------------------------- %
    desiredMean = 100;
    desiredVariance = 100;
    % ------------------------------------------------------- %
    % Lettura dei parametri inseriti
    % ------------------------------------------------------- %
    for i = 1:2:nargin-1
        if strcmp(varargin{i},'mean')
            desiredMean = varargin{i+1};
            if ~isnumeric(desiredMean)
                error('mean value must be a number');
            end
        elseif strcmp(varargin{i},'variance')
            desiredVariance = varargin{i+1};
            if ~isnumeric(desiredVariance)
                error('variance value must be a number');
            end
        end
    end

    % ------------------------------------------------------- %
    % Algoritmo di normalizzazione
    % ------------------------------------------------------- %
    % Calcolo media e varianza dell'immagine originale
    originalMean = mean2(img);
    originalVariance = var2(img);
    % Calcolo l'immagine finale (Hong et al. - 1998)
    logicalValue = ((img > originalMean).*2-1); % true = 1, false = -1
    varianceRatio = sqrt( desiredVariance / originalVariance );
    sqrtValue = varianceRatio * abs(img-originalMean);
    outImg = desiredMean + logicalValue .* sqrtValue;
end
```

```
function vecOut = innerRep (vec,n)
    vecOut = shape(vec( idivide(uint8(1:(n*numel(vec)))-1,n,'floor')+1 ),'col');
end
```

```
function M = max2 (mat)
    M = max (reshape(mat,[numel(mat),1]));
end
```

```
function m = mean2 (A)
```

```
    m = mean(reshape(A,[numel(A),1]));
end
```

```
function m = min2 (mat)
    m = min(reshape(mat,[numel(mat),1]));
end
```

```
function [io,jo] = nearestRidgeLineMaximum (img,is,js,phis,varargin)
% Controllo sull'input
if (numel(is) > 1) || (numel(js) > 1)
    error('The starting point must be unique');
end
% Lettura parametri di input
mu = 3;
sigma = 7;
for k = 1:2:nargin-4
    if strcmp(varargin{k},'mu')
        mu = varargin{k+1};
        if ~isnumeric(mu)
            error('mu must be a number');
        end
    elseif strcmp(varargin{k},'sigma')
        sigma = varargin{k+1};
        if ~isnumeric(sigma)
            error('sigma must be a number');
        end
    end
end

% Nell'articolo viene richiesto di mediare su h sezioni parallele
% Nel nostro caso, prevedendo l'algoritmo di miglioramento
% dell'immagine come preliminare a questo algoritmo, non  necessario
% farlo.
[sections,linSegments] = computeSection(img,phis,is,js,sigma);
section = sections{1};
linSegment = linSegments{1};
section = conv(section,[1/23,2/23,5/23,7/23,5/23,2/23,1/23],'same');
clear sections linSegments it jt;

% Calcolo tutti i massimi locali
idx = 1:length(section);
pp0 = spline(idx,section);
pp1 = ppder(pp0);
pp2 = ppder(pp1);
roots1 = real(pproots(pp1));
der2ValOnRoots1 = ppval(pp2,roots1);
localMaxIdx = round( roots1( der2ValOnRoots1<0 ) );
localMaxLinIdx = linSegment(localMaxIdx);
[im,jm] = ind2sub(size(img),localMaxLinIdx);

% Scelgo il massimo  pi vicino al punto iniziale
distance = sum(([im',jm']-repmat([is,js],length(im),1)).^2,2);
[~,minIdx] = min(distance);
io = im(minIdx);
jo = jm(minIdx);

end
```

```
function vecOut = outerRep (vec,n)
    vecOut = shape(vec ( mod(0:(n*numel(vec)-1),numel(vec))+1 ),'col');
```

```
end
```

```
function ppOut = ppder(pp)
    [breaks,oldCoeff,l,k,d] = unmkpp(pp);
    derCoeff = repmat(k-1:-1:1,l*d,1);
    newCoeff = derCoeff .* oldCoeff(:,1:k-1);
    ppOut = mkpp(breaks,newCoeff);
end
```

```
function rootsOut = pproots (pp)
    precision = 5;
    [breaks,coefs,l,k,~] = unmkpp(pp);
    % prealloco lo spazio per tutte le possibili radici
    rootsOut = NaN(l,k-1);
    for i = 1:l
        currRoots = roots(coefs(i,:))+breaks(i);
        correctMask = and(currRoots<=breaks(i+1),currRoots>=breaks(i));
        correctRoots = currRoots(correctMask);
        rootsOut(i,1:length(correctRoots)) = correctRoots;
    end
    rootsOut = rootsOut(~isnan(rootsOut));
    rootsOut = reshape(rootsOut,[length(rootsOut),1]);
    % Arrotondo i risultati alla precision-th cifra decimale
    rootsOut = round(rootsOut*(10^precision))/(10^precision);
    % Ordino in modo crescente i risultati ed elimino i ripetuti
    rootsOut = unique(sort(rootsOut));
end
```

```
function [minutia,alreadyChecked] = ...
    ridgeLineFollowing (img,phi,alreadyChecked,is,js,cylCoords,cylParamStruct,varargin)
% Controllo sull'input
if (numel(is) > 1) || (numel(js) > 1)
    error('The␣starting␣point␣must␣be␣unique');
end
% Lettura parametri di input
showStep = false;
mu = 3;     % parametro consigliato nell'articolo
sigma = 7;  % parametro consigliato nell'articolo
width = 2;  % parametro consigliato nell'articolo (=2)
beta = pi/4;
psi = pi/2;

for k = 1:2:nargin-7
    if strcmp(varargin{k},'mu')
        mu = varargin{k+1};
        if ~isnumeric(mu)
            error('mu␣must␣be␣a␣number');
        end
    elseif strcmp(varargin{k},'sigma')
        sigma = varargin{k+1};
        if ~isnumeric(sigma)
            error('sigma␣must␣be␣a␣number');
        end
    elseif strcmp(varargin{k},'showStep')
        showStep = varargin{k+1};
        if ~islogical(showStep)
            error('showStep␣must␣be␣a␣logical');
        end
    elseif strcmp(varargin{k},'width');
        width = varargin{k+1};
```

```
33          if ~isnumeric(width)
34              error('width␣must␣be␣a␣number');
35          end
36      elseif strcmp(varargin{k},'nearCenterAngle');
37          beta = varargin{k+1};
38          if ~isnumeric(beta)
39              error('nearCenterAngle␣must␣be␣a␣number');
40          end
41      elseif strcmp(varargin{k},'excBendAngle');
42          psi = varargin{k+1};
43          if ~isnumeric(psi)
44              error('excBendAngle␣must␣be␣a␣number');
45          end
46      end
47  end
48
49  % Genero la figura per la visualizzazione dei risultati
50  if showStep
51      figure('WindowStyle','docked');
52      hold on;
53  end
54
55  % Inizializzo le variabili principali
56  ic = is;
57  jc = js;
58  phic = phi(is,js);
59  polygonalRidgeLineTemp = cell(size(img,2),1);
60  polygonalRidgeLineTemp{1} = [ic,jc,phic];
61  polygonalIdx = 2;
62
63  % Inizializzo una classe per capire il tipo di terminazione
64  exitCondition = exitCondition_.none;
65
66  while true
67      % Avanzo di mu pixel lungo phic
68      it = ic + mu * sin(phic);
69      jt = jc + mu * cos(phic);
70      % Controllo che l'avanzamento non fuoriesca dall'immagine (uno dei
71      % criteri di stop)
72      if any(or([it,jt]<1,[it,jt]>size(img)))
73          exitCondition = exitCondition_.exitArea;
74          break
75      end
76      % Nell'articolo viene richiesto di mediare su h sezioni parallele
77      % Nel nostro caso, prevedendo l'algoritmo di miglioramento
78      % dell'immagine come preliminare a questo algoritmo, non  necessario
79      % farlo.
80      [sections,linSegments] = computeSection(img,phic,it,jt,sigma);
81      section = sections{1};
82      linSegment = linSegments{1};
83      section = conv(section,[1/23,2/23,5/23,7/23,5/23,2/23,1/23],'same');
84      clear sections linSegments;
85
86      if showStep
87          subplot(2,2,1);
88          cla;
89          img2 = zeros(size(img,1),size(img,2),3);
90          img2(:,:,1) = 255-img;
91          img2(:,:,2) = 255-img;
92          img2(:,:,3) = 255-img;
93          [visI,visJ] = ind2sub(size(img),linSegment);
94          [visLinSegment] = sub2ind(size(img2),visI,visJ,ones(size(visI)));
```

```
95          img2(visLinSegment) = 255;
96          [visLinSegment] = sub2ind(size(img2),visI,visJ,2*ones(size(visI)));
97          img2(visLinSegment) = 0;
98          [visLinSegment] = sub2ind(size(img2),visI,visJ,3*ones(size(visI)));
99          img2(visLinSegment) = 0;
100         img2(ic,jc,:) = [0,0,255];
101         imshow(uint8(img2),'InitialMagnification','fit');
102         subplot(2,2,2);
103         cla;
104         localImg = img2(...
105             max([ic-3*sigma,1]):min([ic+3*sigma,size(img2,1)]),...
106             max([jc-3*sigma,1]):min([jc+3*sigma,size(img2,2)]),:);
107         imshow(uint8(localImg),'InitialMagnification','fit');
108         subplot(2,2,3);
109         cla;
110         bar(1:length(section),section);
111         hold on;
112         clear img2 localImg;
113     end
114
115     % Se il profilo contiene tutti valori molto bassi, significa che siamo
116     % arrivati in una zona che all'occhio umano risulta bianca. Quindi
117     % siamo usciti dal ridge. Impostiamo la soglia a 25, ma va
118     % parametrizzato.
119 %     if max(section) < 25
120 %         exitCondition = exitCondition_.termination;
121 %         break;
122 %     end
123
124     % Trovo il massimo locale  pi vicino al centro della sezione
125 %     [~,maxIdx] = max(section);
126     pp0 = spline(1:length(section),section);
127     pp1 = ppder(pp0);
128     pp2 = ppder(pp1);
129     roots1 = pproots(pp1);
130     roots1 = roots1(imag(roots1)==0);
131     der2val = ppval(pp2,roots1);
132     der2threshold = 0.1;
133     maxIdx = round(roots1(der2val < -der2threshold));
134     [maxI,maxJ] = ind2sub(size(img),linSegment(maxIdx));
135     v1 = [shape(maxI,'col')-innerRep(ic,numel(maxI)) , ...
136         shape(maxJ,'col')-innerRep(jc,numel(maxJ))];
137     v2 = repmat([it-ic,jt-jc],numel(maxI),1);
138     angles = angleFromVectors(v1,v2);
139     [~,angleIdx] = min(angles);
140     if isempty(angleIdx)||(angles(angleIdx) >= beta)
141         maxI = [];
142         maxJ = [];
143     else
144         maxI = maxI(angleIdx);
145         maxJ = maxJ(angleIdx);
146     end
147
148 %     [in,jn] = ind2sub(size(img),linSegment);
149 %     nearCenterCondition = abs(...
150 %         angleFromVectors([in',jn']-repmat([ic,jc],length(linSegment),1),...
151 %         [cos(phic-pi/2),-sin(phic-pi/2)])) < beta;
152 %     threshold = mean(section(nearCenterCondition))+0.5*std(section(nearCenterCondition));
153 %     section(~nearCenterCondition) = 0;
154     if showStep
155         plot(1:.01:length(section),ppval(pp0,1:.01:length(section)),...
156             'LineWidth',3,'Color','red');
```

```
157         stem(roots1,ppval(pp0,roots1),'fill','Color','blue','MarkerSize',...
158             5,'LineStyle','none');
159         text(roots1,ppval(pp0,roots1)-sign(der2val)*(max(section)-...
160             min(section))*0.4,num2str(der2val),'Color','red');
161         subplot(2,2,4);
162         cla;
163         imshow(~alreadyChecked,'InitialMagnification','fit');
164     end
165 %     clear nearCenterCondition in jn;
166     % Controllo il secondo criterio di stop (verifico che esiste un
167     % massimo locale - per farlo chiedo che il massimo non sia pari
168     % alla moda)
169     if numel(maxI) == 0
170         % Se non esiste un massimo locale
171         if polygonalIdx > 3
172             % Se ha fatto almeno 3 passi
173             exitCondition = exitCondition_.termination;
174         end
175         break
176     else
177         % Se  stato trovato un massimo locale
178 %         maxLinIdx = linSegment(maxIdx);
179 %         [in,jn] = ind2sub(size(img),maxLinIdx);
180 %         clear maxLinIdx maxIdx section;
181         in = maxI;
182         jn = maxJ;
183         if any(or([in,jn]<1,[in,jn]>size(img)))
184             % Se il nuovo punto esce dai limiti dell'immagine
185             exitCondition = exitCondition_.exitArea;
186             break
187         else
188             % Se il nuovo punto rientra nei limiti dell'immagine
189             if alreadyChecked(in,jn) > 0
190                 % Se il punto appartiene ad un ridge  gi controllato (criterio
191                 % di stop)
192                 if polygonalIdx > 3
193                     % Se ha fatto almeno 3 passi
194                     exitCondition = exitCondition_.intersection;
195                 end
196                 break
197             else
198                 % Se il punto non  mai stato controllato
199                 excessiveBending = false;
200                 if polygonalIdx > 4
201                     % Calcolo la direzione media tra gli ultimi 3 segmenti
202                     % della poligonale, per sottoporla all'ultimo controllo
203                     p1 = polygonalRidgeLineTemp{polygonalIdx-4};
204                     p2 = polygonalRidgeLineTemp{polygonalIdx-3};
205                     p3 = polygonalRidgeLineTemp{polygonalIdx-2};
206                     p4 = polygonalRidgeLineTemp{polygonalIdx-1};
207                     averageDirection = [p4-p3;p3-p2;p2-p1];
208                     averageDirection = averageDirection(:,1:2);
209                     averageAngle = mean(abs(angleFromVectors(...
210                         averageDirection,repmat([1,0],size(averageDirection,1),1))));
211                     currAngle = abs(angleFromVectors([in,jn]-[ic,jc],[1,0]));
212                     excessiveBending = abs(averageAngle-currAngle) > psi;
213                     clear p1 p2 p3 p4 averageDirection;
214                 end
215 
216                 if excessiveBending
217                     % Se la poligonale si piega troppo
218                     exitCondition = exitCondition_.bending;
```

```
                    break
                else
                    % Se la poligonale non si piega troppo
                    ic = in;
                    jc = jn;
                    phic = phi(ic,jc);
                    polygonalRidgeLineTemp{polygonalIdx} = [ic,jc,phic];
                    polygonalIdx = polygonalIdx + 1;
                end
            end
        end
    end
end
polygonalRidgeLine = polygonalRidgeLineTemp(1:polygonalIdx-1);
% A partire dalla poligonale aggiorno la matrice alreadyChecked creando un
% tubo largo eps attorno alla poligonale
for k = 1:length(polygonalRidgeLine)-1
    p0 = polygonalRidgeLine{k};
    p1 = polygonalRidgeLine{k+1};
    [px,py] = cylCoordsRecover(p0(1:2),p1(1:2),width,cylCoords,cylParamStruct);
    linIdx = sub2ind(size(img),px,py);
    alreadyChecked(linIdx) = true;
end

if showStep
    disp(exitCondition);
    close;
end

minutia = struct('pos',[-1,-1],'angle',0,'exitCondition',exitCondition);
if (exitCondition ~= exitCondition_.none) && ...
        (exitCondition ~= exitCondition_.exitArea) && (polygonalIdx > 2)
    minutiaPos = polygonalRidgeLine{end};
    minutia.pos = minutiaPos(1:2);
    minutia.angle = minutiaPos(3);
end

end
```

```
function [T1, T2] = ridgeValleyWidth (xSignature, img, phi, varargin)
% ------------------------------------------------------- %
% Imposto i parametri di default
% ------------------------------------------------------- %
stdThreshold = 2;
derThreshold = 5;
filterSize = 3;
filterSigma = 5;
finalFilterSize = 5;
finalFilterSigma = 4;
w = 16;
lFun = @(w) 3*w;
% ------------------------------------------------------- %
% Lettura dei parametri inseriti
% ------------------------------------------------------- %
for i = 1:2:nargin-3
    if strcmp(varargin{i},'stdThreshold')
        stdThreshold = varargin{i+1};
        if ~isnumeric(stdThreshold)
            error('stdThreshold␣value␣must␣be␣a␣number');
        end
    elseif strcmp(varargin{i},'derThreshold')
```

```
23          derThreshold = varargin{i+1};
24          if ~isnumeric(derThreshold)
25              error('derThreshold␣value␣must␣be␣a␣number');
26          end
27      elseif strcmp(varargin{i},'filterSize')
28          filterSize = varargin{i+1};
29          if ~isnumeric(filterSize)
30              error('filterSize␣value␣must␣be␣a␣number');
31          end
32      elseif strcmp(varargin{i},'filterSigma')
33          filterSigma = varargin{i+1};
34          if ~isnumeric(filterSigma)
35              error('filterSigma␣value␣must␣be␣a␣number');
36          end
37      elseif strcmp(varargin{i},'finalFilterSize')
38          finalFilterSize = varargin{i+1};
39          if ~isnumeric(finalFilterSize)
40              error('finalFilterSize␣value␣must␣be␣a␣number');
41          end
42      elseif strcmp(varargin{i},'finalFilterSigma')
43          finalFilterSigma = varargin{i+1};
44          if ~isnumeric(finalFilterSigma)
45              error('finalFilterSigma␣value␣must␣be␣a␣number');
46          end
47      elseif strcmp(varargin{i},'windowSize')
48          w = varargin{i+1};
49          if ~isnumeric(w)
50              error('windowSize␣must␣be␣a␣number');
51          end
52      elseif strcmp(varargin{i},'lFun')
53          lFun = varargin{i+1};
54          % Manca il controllo sul tipo di input
55      end
56  end
57
58  l = lFun(w);
59  r = ceil(0.5 * sqrt(w^2+l^2));
60  show2 = false;
61
62  dim = size(xSignature);
63  roughSegmentation = NaN(dim(1),dim(2));
64  ridgeWidth = NaN(dim(1:2));
65  valleyWidth = ridgeWidth;
66  for i = 1:dim(1)
67      for j = 1:dim(2)
68  %         disp(['( ',num2str(i),' , ',num2str(j),' )']);
69          x = (1:dim(3))';
70          y = reshape(xSignature(i,j,:),[dim(3),1]);
71          % Controllo se la x-signature sono utili
72          if std(y) < stdThreshold
73              continue;
74          end
75
76          % Utilizzo una spline cubica per interpolare le x-signature
77          pp0 = spline(x,y);
78          pp1 = ppder(pp0);
79          pp2 = ppder(pp1);
80          roots2 = pproots(pp2);
81
82          % Accetto solamente le radici in cui corrispondenza
83          % la derivata prima ha valore maggiore di derThreshold
84          roots2Accepted = roots2(abs(ppval(pp1,roots2))>derThreshold);
```

```
85
86          if length(roots2Accepted)<2
87              continue
88          end
89
90          % Calcolo le radici della derivata prima, che
91          % serviranno nel passaggio successivo
92          roots1 = pproots(pp1);
93          roots1 = unique(sort(real(roots1)));
94
95          % Trovo tutti gli zeri della der. I contenuti tra due zeri della
96          % der. II (in questo modo trovo tutte le possibili creste e valli)
97          roots2Accepted = sort(roots2Accepted);
98          roots2AcceptedRep = [roots2Accepted(1);...
99              innerRep(roots2Accepted(2:end-1),2);...
100             roots2Accepted(end)];
101         roots2ARight = roots2AcceptedRep(2:2:end);
102         numEven = numel(roots2ARight);
103         roots2ALeft = roots2AcceptedRep(1:2:numEven*2);
104         roots1Rep = shape(innerRep(roots1,numEven),'col');
105         roots2ARightRep = outerRep(roots2ARight,numel(roots1));
106         roots2ALeftRep = outerRep(roots2ALeft,numel(roots1));
107         betweenRoots2Condition = ...
108             and(roots1Rep>roots2ALeftRep,roots1Rep<roots2ARightRep);
109
110         if ~any(betweenRoots2Condition)
111             continue
112         end
113         idx = find(shape(betweenRoots2Condition,'col'));
114         roots1 = roots1Rep(idx);
115         roots2ALeft = roots2ALeftRep(idx);
116         roots2ARight = roots2ARightRep(idx);
117         clear roots2AcceptedRep roots2Accepted  roots2ALeftRep ...
118             roots2ARightRep idx numEven betweenRoots2Condition;
119
120         % Impongo che sul punto centrale ci sia una cresta o una valle
121         withinRootsWidth = roots2ARight-roots2ALeft;
122         eps = mean(withinRootsWidth);
123         nearCenterCondition = and( roots1<dim(3)/2+eps , roots1>dim(3)/2-eps );
124         if ~any(nearCenterCondition)
125             continue
126         end
127         clear eps nearCenterCondition;
128
129         % Considero due radici adiacenti alla volta:
130         % se tra le due c ' un massimo o minimo
131         % (si annulla la derivata prima, der seconda >0 o <0)
132         % allora catalogo la loro differenza rispettivamente
133         % come valleyWidth e ridgeWidth
134         % (minimo corrisponde a ridge)
135         localRidgeWidth = NaN(1,length(withinRootsWidth));
136         localValleyWidth = localRidgeWidth;
137
138         I = (i-1)*w+r+1;
139         J = (j-1)*w+r+1;
140 %         if (I>150-w)&&(I<150+w)&&(J>101-w)&&(J<101+w)
141 %             disp('eccomi');
142 %         end
143
144         for k = 1:length(withinRootsWidth)
145             currWidth = withinRootsWidth(k);
146             % Controllo se c ' una radice della derivata prima
```

```
147             % tra queste due radici della derivata seconda
148             condition = and(roots1 > roots2ALeft(k),roots1 < roots2ARight(k));
149             currRoot1 = roots1(condition);
150 
151             currDer2Val = ppval(pp2,currRoot1);
152             condition2 = abs(currDer2Val)>0.001;
153             if ~any(condition2)
154                 % La width corrente  un falso dovuto al rumore
155                 continue
156             end
157             currDer2Val = currDer2Val(condition2);
158             currRoot1 = currRoot1(condition2);
159             if mode(currDer2Val > 0)
160                 localRidgeWidth(k) = currWidth;
161                 if (dim(3)/2>roots2ALeft(k))&&(dim(3)/2<roots2ARight(k))
162                     roughSegmentation(i,j) = true; % Siamo in presenza di una cresta
163                 end
164             elseif mode(currDer2Val < 0)
165                 localValleyWidth(k) = currWidth;
166                 if (dim(3)/2>roots2ALeft(k))&&(dim(3)/2<roots2ARight(k))
167                     roughSegmentation(i,j) = false; % Siamo in presenza di una valle
168                 end
169             else
170                 distFromCenter = abs(currRoot1 - dim(3)/2);
171                 [~,minDistIdx] = min(distFromCenter);
172                 if currDer2Val(minDistIdx) > 0 % Ridge
173                     localRidgeWidth(k) = currWidth;
174                     if (dim(3)/2>roots2ALeft(k))&&(dim(3)/2<roots2ARight(k))
175                         roughSegmentation(i,j) = true; % Siamo in presenza di una cresta
176                     end
177                 elseif currDer2Val(minDistIdx) < 0 % Valley
178                     localValleyWidth(k) = currWidth;
179                     if (dim(3)/2>roots2ALeft(k))&&(dim(3)/2<roots2ARight(k))
180                         roughSegmentation(i,j) = false; % Siamo in presenza di una valle
181                     end
182                 else
183                     continue
184                 end
185             end
186         end
187         % Ora viene eseguita la media di tutte le
188         % localValleyWidth e localRidgeWidth
189         localRidgeWidth = mean(localRidgeWidth(~isnan(localRidgeWidth)));
190         localValleyWidth = mean(localValleyWidth(~isnan(localValleyWidth)));
191         if all(~isnan([localRidgeWidth,localValleyWidth]))
192             ridgeWidth(i,j) = localRidgeWidth;
193             valleyWidth(i,j) = localValleyWidth;
194         elseif ~isnan(localRidgeWidth)
195             ridgeWidth(i,j) = localRidgeWidth;
196             valleyWidth(i,j) = localRidgeWidth;
197         elseif ~isnan(localValleyWidth)
198             ridgeWidth(i,j) = localValleyWidth;
199             valleyWidth(i,j) = localValleyWidth;
200         else
201             continue
202         end
203         % Visualizzazione
204         if show2
205             h = figure('WindowStyle','docked');
206             I = (i-1)*w+r+1;
207             J = (j-1)*w+r+1;
208             localImgRange = -w/2:1:w/2-1;
```

```
209             localImg = img(I+localImgRange,J+localImgRange);
210             imshow(uint8(localImg));
211             disp(['I=',num2str(I),'␣J=',num2str(J),'␣R=',num2str(ridgeWidth(i,j)),...
212                 '␣V=',num2str(valleyWidth(i,j)),'␣Seg=',num2str(roughSegmentation(i,j))]);
213             close(h);
214         end
215     end
216 end
217 % Interpolazione dei valori, per uniformare blocchi
218 % dove la width non  definita
219 ridgeWidth(isnan(ridgeWidth)) = 0;
220 valleyWidth(isnan(valleyWidth)) = 0;
221 roughSegmentation(isnan(roughSegmentation)) = -1;
222 roughSegmentation = double(roughSegmentation);
223 gaussianFilter = fspecial('gaussian',filterSize,filterSigma);
224 while any(roughSegmentation(:)<0)
225     temp = conv2(roughSegmentation,gaussianFilter,'same');
226     % in Hong et al. era differente la formula, qua dovrebbe andare bene  cos
227     roughSegmentation(roughSegmentation<0) = temp(roughSegmentation<0);
228 end
229 roughSegmentation = round(roughSegmentation);
230 while any(ridgeWidth(:) == 0)
231     temp = conv2(ridgeWidth,gaussianFilter,'same');
232     % in Hong et al. era differente la formula, qua dovrebbe andare bene  cos
233     ridgeWidth(ridgeWidth == 0) = temp(ridgeWidth == 0);
234 end
235 while any(valleyWidth(:) == 0)
236     temp = conv2(valleyWidth,gaussianFilter,'same');
237     % in Hong et al. era differente la formula, qua dovrebbe andare bene  cos
238     valleyWidth(valleyWidth == 0) = temp(valleyWidth == 0);
239 end
240 % Eseguo un ulteriore low-pass filter per uniformare le width
241 gaussianFilter = fspecial('gaussian',finalFilterSize,finalFilterSigma);
242 ridgeWidth = conv2(ridgeWidth,gaussianFilter,'same');
243 valleyWidth = conv2(valleyWidth,gaussianFilter,'same');
244 % Calcolo i periodi del filtro T1 e T2
245 T1 = zeros(size(ridgeWidth));
246 T2 = T1;
247 for i = 1:dim(1)
248     for j = 1:dim(2)
249         currRidgeLocation = roughSegmentation(i,j);
250         currRidgeWidth = ridgeWidth(i,j);
251         currValleyWidth = valleyWidth(i,j);
252         if currRidgeLocation==true % ridge
253             T1(i,j) = 2 * currRidgeWidth;
254             T2(i,j) = 2 * currValleyWidth;
255         elseif currRidgeLocation==false % valley
256             T1(i,j) = 2 * currValleyWidth;
257             T2(i,j) = 2 * currRidgeWidth;
258         else
259             disp(['NaN␣in␣posizione␣(',num2str(i),',',num2str(j),')']);
260         end
261     end
262 end
263 end
```

```
1 function vecOut = shape(vec,str)
2     if strcmp(str,'row')
3         vecOut = reshape(vec,[1,numel(vec)]);
4     elseif strcmp(str,'col')
5         vecOut = reshape(vec,[numel(vec),1]);
```

```
    else
        error('str must be ''col'' or ''row''');
    end
end
```

```
function oldIter = showCompletion(oldIter,currIter,totIter,step,str)

oldPerc = oldIter / totIter * 100;
currPerc = currIter / totIter * 100;
exponent = round(log10(step));

if currPerc-oldPerc >= step
    oldIter = currIter;
    percStr = round(currPerc*10^(-exponent))/(10^(-exponent));
    [matchStart,matchEnd] = regexp(str,'&&&');
    for k = 1:length(matchStart)
        str = [str(1:matchStart-1),num2str(percStr),str(matchEnd+1:end)];
    end
    disp(str);
    pause(1e-5);
end

end
```

```
function showMinutiae(img,minutiae,varargin);
% Controllo i parametri di input
if ~isstruct(minutiae)
    error('Minutiae must be a structure');
end
r = 3;
for k = 1:2:nargin-2
    if strcmp(varargin{k},'triangleSize')
        r = varargin{k+1};
        if ~isnumeric(r)
            error('triangleSize must be a number');
        end
    end
end

% Creo una funzione che restituisce le coordinate dei vertici di un
% triangolo ruotato attorno all'origine di un angolo phi e successivamente
% traslato di un vettore (t1,t2)
triangle = @(t1,t2,phi) ...
    [cos(phi-pi/2),  sin(phi-pi/2),  t1;... % matrice di rot. e trasl. in coord. omog.
    -sin(phi-pi/2),   cos(phi-pi/2),   t2;...
    0,          0,          1] * ...
    [2*r, r*cosd(120),    r*cosd(240),    2*r;... % coordinate dei tre vertici del triangolo
    0,  r*sind(120),    r*sind(240),    0;...
    1,  1,              1,              1];

figure('Name','ShowMinutiae','WindowStyle','docked');
hold on;
imshow(uint8(img),'InitialMagnification','fit');

% Scorro al struttura delle minutiae e le disegno
for k = 1:length(minutiae)
    if any(or(minutiae(k).pos <= 1+r,minutiae(k).pos >= size(img)-r))
        continue
    end
    t = triangle(minutiae(k).pos(1),minutiae(k).pos(2),minutiae(k).angle);
    switch minutiae(k).exitCondition
```

```
        case exitCondition_.termination
            color = [28/255,97/255,51/255]; % verde scuro
        case exitCondition_.intersection
            color = [28/255,51/255,97/255]; % blu scuro
        case exitCondition_.bending
            color = [97/255,51/255,28/255]; % rosso scuro
    end
    plot(t(2,:),t(1,:),'Color',color,'LineWidth',2);
end


end
```

```
function showOrientation (varargin)
    % ------------------------------------------------------ %
    % Lettura parametri di input
    % ------------------------------------------------------ %
    % Se viene passata in input una finestra, allora uso quella
    % altrimenti ne creo una nuova
    if nargin == 2
        img = varargin{1};
        orientation = varargin{2};
        figure;
    elseif nargin == 3
        figure(varargin{1});
        img = varargin{2};
        orientation = varargin{3};
    else
        error('Wrong␣number␣of␣input␣parameters');
    end
    hold on;

    % ------------------------------------------------------ %
    % Conversione dell'immagine in valori interi in [0,255]
    % ------------------------------------------------------ %
    if max2(img) > 1
        img = uint8(img);
    end

    % ------------------------------------------------------ %
    % Mostro l'immagine e disegno sopra i vettori
    % che mostrano l'orientazione
    % ------------------------------------------------------ %
    w = max([min([round(mean(size(img))/10),16]),1]);
    length = 3/4*w;
    imshow(img,'InitialMagnification','fit');
    utemp = cos(orientation);
    vtemp = sin(orientation);
    u = zeros(size(img,1),size(img,2));
    v = zeros(size(img,1),size(img,2));
    u(1:w:size(img,1),1:w:size(img,2)) = utemp (1:w:size(img,1),1:w:size(img,2));
    v(1:w:size(img,1),1:w:size(img,2)) = vtemp (1:w:size(img,1),1:w:size(img,2));
    quiver(u,v,length,'Color','yellow');
end
```

```
function s = std2 (A)
    s = std(reshape(A,[numel(A),1]));
end
```

```
function m = sum2 (A)
```

```
2       m = sum(reshape(A,[numel(A),1]));
3   end
```

```
1   function v = var2 (A)
2       v = var(reshape(A,[numel(A),1]));
3   end
```

Bibliografia

[1] Asker M Bazen and Sabih H Gerez. Systematic methods for the computation of the directional fields and singular points of fingerprints. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):905–919, 2002.

[2] E. O. Brigham. *The Fast Fourier Transform and its Appications*.

[3] Philippe Cattin. Introduction to signal- and image-processing.

[4] Xinjian Chen, Jie Tian, Jiangang Cheng, and Xin Yang. Segmentation of fingerprint images using linear classifier. *EURASIP Journal on Applied Signal Processing*, 2004:480–494, 2004.

[5] Sharat Chikkerur, Alexander N Cartwright, and Venu Govindaraju. Fingerprint enhancement using stft analysis. *Pattern Recognition*, 40(1):198–211, 2007.

[6] JR Da Costa, Franck Le Pouliquen, Christian Germain, and Pierre Baylou. New operators for optimized orientation estimation. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, volume 3, pages 744–747. IEEE, 2001.

[7] John G Daugman. Two-dimensional spectral analysis of cortical receptive field profiles. *Vision research*, 20(10):847–856, 1980.

[8] MJ Donahue and SI Rokhlin. On the use of level curves in image analysis. *CVGIP: Image Understanding*, 57(2):185–203, 1993.

[9] Federal Bureau of Investigation. *Fingerprint Training Manual*.

[10] Dennis Gabor. Theory of communication. part 1: The analysis of information. *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, 93(26):429–441, 1946.

[11] Rafael C Gonzalez and Richard E Woods. Digital image processing, 2002.

[12] Lawrence O Gorman. Comparing passwords, tokens, and biometrics for user authentication. *Proceedings of the IEEE*, 91(12):2021–2040, 2003.

[13] Mark Hawthorne. *Fingerprints: analysis and understanding*. CRC Press, 2008.

[14] Lin Hong, Yifei Wan, and Anil Jain. Fingerprint image enhancement: algorithm and performance evaluation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):777–789, 1998.

[15] Anil Jain, Ruud Bolle, and Sharath Pankanti. *Biometrics: personal identification in networked society*, volume 479. Springer Science & Business Media, 2006.

[16] Anil K Jain, Patrick Flynn, and Arun A Ross. *Handbook of biometrics*. Springer Science & Business Media, 2007.

[17] Toshio Kamei and Masanori Mizoguchi. Image filter design for fingerprint enhancement. In *Computer Vision, 1995. Proceedings., International Symposium on*, pages 109–114. IEEE, 1995.

[18] Michael Kass and Andrew Witkin. Analyzing oriented patterns. *Computer vision, graphics, and image processing*, 37(3):362–385, 1987.

[19] Byung-Gyu Kim and Dong-Jo Park. Adaptive image normalisation based on block processing for enhancement of fingerprint image. *Electronics Letters*, 38(14):696–698, 2002.

[20] Dario Maio and Davide Maltoni. Direct gray-scale minutiae detection in fingerprints. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(1):27–40, 1997.

[21] Dario Maio and Davide Maltoni. Ridge-line density estimation in digital images. In *icpr*, page 534. IEEE, 1998.

[22] Davide Maltoni, Dario Maio, Anil K Jain, and Salil Prabhakar. *Handbook of fingerprint recognition*. Springer Science & Business Media, 2009.

[23] Airam Carlos Pais Barreto Marques and Antonio Carlos Gay Thome. A neural network fingerprint segmentation method. In *Hybrid Intelligent Systems, 2005. HIS'05. Fifth International Conference on*, pages 6–pp. IEEE, 2005.

[24] David Marr and Ellen Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London B: Biological Sciences*, 207(1167):187–217, 1980.

[25] Babu M Mehtre and B Chatterjee. Segmentation of fingerprint images—a composite method. *Pattern recognition*, 22(4):381–385, 1989.

[26] Babu M Mehtre, NN Murthy, Surendra Kapoor, and B Chatterjee. Segmentation of fingerprint images using the directional image. *Pattern Recognition*, 20(4):429–435, 1987.

[27] Bijan Moayer and King-Sun Fu. A tree system approach for fingerprint pattern recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (3):376–387, 1986.

[28] Andre A Moenssens. *Fingerprint techniques*. Chilton Book Company London, 1971.

[29] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[30] Nalini K Ratha, Shaoyun Chen, and Anil K Jain. Adaptive flow orientation-based feature extraction in fingerprint images. *Pattern Recognition*, 28(11):1657–1672, 1995.

[31] Azriel Rosenfeld and Avinash C Kak. *Digital Picture Processing: Vol.: 1*. Academic Press, Incorporated, 1982.

[32] David SG Vernon. Automatic detection of secondary creases in fingerprints. *Optical Engineering*, 32(10):2616–2623, 1993.

[33] Jianwei Yang, Lifeng Liu, Tianzi Jiang, and Yong Fan. A modified gabor filter design method for fingerprint image enhancement. *Pattern Recognition Letters*, 24(12):1805–1817, 2003.