Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Matematica

Metodi numerici per la segmentazione di immagini digitali

Tesi di Laurea in Analisi Numerica

Relatore:
Chiar.ma Prof.ssa
ELENA LOLI PICCOLOMINI

Presentata da:
ELENA MOROTTI

II Sessione
Anno Accademico 2009-2010

Introduzione

La segmentazione di un'immagine é un processo di separazione dell'immagine stessa in diverse regioni, che in seguito vengono raggruppate secondo specifiche condizioni. Il risultato finale rileva oggetti che l'occhio umano puó facilmente distinguere. Poiché i calcolatori non hanno mezzi propri per riconoscere gli elementi di un'immagine, sono stati sviluppati numerosi algoritmi per segmentare le immagini, che sfruttano caratteristiche insite nell'immagine stessa, quali le informazioni fornite dal colore di ciascun pixel.
Tali algoritmi utilizzano strumenti matematici per l'elaborazione di immagini in scala di grigio, che corrispondono a matrici 2-dimensionali per il calcolatore. Si fa largo uso, infatti, delle discretizzazioni delle derivate, del calcolo matriciale e di diverse trasformate per progettare i metodi operativi di segmentazione.

Questa tesi analizza, testa e confronta alcuni metodi di eleborazione e segmentazione di immagini digitali.
In particolare, nel primo capitolo si spiega che cosa sia un'immagine digitale e in che cosa consista la sua elaborazione digitale (*Digital Image Processing*) dal punto di vista numerico, quindi si prosegue analizzando diverse tecniche di filtraggio e di estrazione dei contorni. Questi sono passaggi preliminari alla segmentazione vera e propria, eseguiti per migliorare e ripulire l'input da fattori che danneggerebbero il risultato finale (in primo luogo il cosiddetto *rumore*).
Nel secondo capitolo si approfondiscono tre principali metodi di segmen-

tazione (la *sogliatura*, la *segmentazione region-based* e la *segmentazione con la trasformata watershed*), studiandone alcune varianti e perfezionamenti.
Il terzo capitolo riprende gli argomenti precedenti, riportandone il codice di implementazione scritto nel linguaggio di Matlab. Inoltre, ogni algoritmo é seguito da alcuni output esemplificativi che ne mostrano pregi e difetti.

Indice

Capitolo 1

Elaborazione delle immagini digitali

1.1 Immagini digitali

Un'immagine digitale, o numerica, è costituita da una collezione di elementi discreti che possono essere considerati elementi di una matrice. Questi elementi sono chiamati pixel, abbreviazione del termine inglese *"picture elements"*. Ad ognuno di questi è associato un valore, scalare o vettoriale a seconda del tipo di immagine.

Nell'acquisizione in digitale, un sensore converte l'immagine in un numero discreto di pixel, assegnando a ciascuno un indice numerico di locazione ed un valore di livello di grigio o di colore (a seconda delle caratteristiche dell'immagine da digitalizzare). Si considera infatti l'immagine digitale come un array 2-dimensionale di valori rappresentanti l'intensitá luminosa, ossia come una funzione f (x,y) dove f corrisponde alla luminositá del punto (x,y) mentre x e y sono alle coordinate di un singolo pixel. Per convenzione, il pixel (0,0) é quello situato nell'angolo in alto a sinistra. Detto ció, é chiaro

che

$$\boldsymbol{f(x,y)} = \begin{pmatrix} f(0,0) & f(0,1) & \dots & f(0,n-1) \\ f(1,0) & f(1,1) & \dots & f(1,n-1) \\ & \vdots & & \\ f(m-1,0) & f(m-1,1) & \dots & f(m-1,n-1) \end{pmatrix}$$

é l'equazione che rappresenta un'immagine digitale.

Esistono tre principali tipi di immagine digitalizzata:

Immagini a toni di grigio. In un'immagine a toni di grigio ad ogni pixel viene associato un numero intero positivo che indica l'intensitá luminosa del corrispettivo punto dell'immagine. Il numero di livelli di grigio corrisponde alla risoluzione in ampiezza dell'immagine.

Immagini binarie. Le immagine binarie sono un caso particolare delle immagini a toni di grigio in cui ogni pixel puó assumere solo un valore binario (0-1), quindi ci sono solo due valori di intensitá (bianco o nero). Immagini binarie sono spesso prodotte da operazioni di sogliatura applicate a immagini a toni di grigio o a colori e danno informazioni sulla forma degli oggetti, piú che sulla distribuzione dell'intensitá luminosa.

Immagini multispettrali. Un'immagine multispettrale é un'immagine in cui il valore di ogni pixel é un vettore e non piú uno scalare. Ogni componente del vettore rappresenta l'informazione corrispondente ad una certa banda spettrale. Classici esempi di immagine multispettrale sono dati dalle immagini a colori RGB, in cui il valore di ogni pixel é costituito da tre componenti che corrispondono ai colori fondamentali rosso (Red), verde (Green) e blu (Blue), e dalle immagini a colori HSL, le cui tre componenti sono tinta (Hue), saturazione (Saturation) e luminanza (Luminance).

Le proprietá fondamentali di un'immagine digitalizzata sono la risoluzione spaziale, la definizione e il numero di piani. La risoluzione spaziale é il prodotto m×n tra il numero m delle righe e quello n delle colonne dei pixel;

ció significa che l'immagine discretizzata ha m pixel lungo l'asse verticale e n pixel lungo l'asse orizzontale. Per definizione di un'immagine intendiamo il numero di sfumature distinguibili nell'immagine, mentre il *bit-depth* di un'immagine é il numero di bit impiegato nel codificare il valore di un pixel: per un dato *bit-depth* di n, l'immagine ha una definizione di 2n, cioé ogni pixel puó assumere un valore in un range di 2n livelli. Infine, il numero di piani corrisponde al numero di array di pixel che compongono l'immagine: un'immagine in scala di grigio, o in pseudo-color, é composta da un unico piano, mentre una true-color da tre piani, uno per componente (rosso, blu e verde).

Le immagini che veranno analizzate nel terzo capitolo sono in *greyscale*, dunque composte da un singolo piano di pixel. Ogni pixel é codificato usando un 8-bit unsigned integer che rappreseta il valore del livello di grigio tra 0 (nero) e 255 (bianco).

1.2 Elaborazione digitale

L'elaborazione digitale di un'immagine (*Digital Image Processing*) é una disciplina che prevede l'utilizzo di una serie di algoritmi per modificare l'immagine in input. Lavorando sui pixel dell'immagine con trasformazioni numeriche, tali algoritmi restituiscono in output ancora delle immagini, a differenza dell'analisi numerica di un'immagine che é un processo il cui output é costituito da misurazioni relative all' immagine stessa.
La tipologia piú nota di elaborazione digitale é l'editing (o fotoritocco) che mira a migliorare nitidezza e gamma cromatica.
Nello specifico, modificare un'immagine significa variare i valori dei pixel tramite algoritmi detti operatori. Tali operatori sono raggruppabili in due categorie:

- operatori puntuali
- operatori spaziali

Gli operatori puntuali eseguono semplici elaborazioni delle immagini, variando la scala di intensitá pixel per pixel. Un tale processo puó essere descritto con una *funzione dimapping* (funzione di trasferimento) del tipo

$$s = M(r)$$

dove r e s sono rispettivamente il valore iniziale e quello finale del pixel in esame.
Le operazioni puntuali vengono eseguite sulla base dell'analisi dei livelli di grigio nell'immagine; lo strumento principale per questa indagine é l'istogramma dell'immagine digitale, cioé la rappresentazione grafica della distribuzione tonale nell'immagine stessa. Questo grafico monodimensionale ha in ascissa l'intensitá dei pixel e in ordinata il numero di pixel che hanno quel determinato valore di intensitá. Per esempio, per un'immagine a toni di grigio l'istogramma ha 256 valori sulle ascisse, mentre un'immagine a colori ha tre distinti istogrammi, corrispondenti ai tre canali Red, Green e Blue, ciascuno con 256 punti in ascissa.
Un esempio di algoritmo puntuale é l'*operatore thresholding* (operatore di sogliatura), di cui parla il paragrafo 2.1.

Gli operatori spaziali determinano il valore finale di un pixel analizzandone il valore originario assieme a quello di alcuni pixel ad esso vicini.

Per ottenere risultati soddisfacenti, la segmentazione si appoggia a tecniche di filtraggio e riconoscimento dei contorni.
Nelle immagini reali, i contorni sono spesso mal definiti, perché affetti da distorsioni quali la mancata messa a fuoco, la presenza di ombre causate da sorgenti luminose non puntiformi o scale cromatiche con cui si rendono gli spigoli arrotondati. Molte volte un'immagine é anche affetta da rumore, ossia una sorta di granulositá e puntinatura che appiattisce e sfuma i toni dell'immagine. Per ridurre questi disturbi si utilizzano tecniche di filtraggio. Al contrario, se l'immagine é giá ben definita, si puó utilizzare questa tecnica per sfumare i contorni, rendendoli piú grossi e quindi meglio individuabili dagli algoritmi di *edge detection*.

Utilizzo

L'elaborazione digitale di immagini offre molti vantaggi rispetto all'elaborazione analogica: sulle immagini digitali é possibile eseguire, in tempi significativamente bassi, operazioni molto complesse e improponibili per via analogica e, inoltre, si evita di distorcere il segnale di partenza, durante l'elaborazione dell'immagine.

Questa disciplina, con tutte le sue branchie, trova applicazioni in svariati settori, per esempio per il montaggio di film si usa il *digital and optical composition* che combina assieme piú immagini, mentre per estrarre un testo da un'immagine o riconoscere i volti presenti in un'immagine, si sfrutta l'*image recognition*. Altre funzionalitá del *digital image processing* sono alla base dello studio di immagini microscopiche e morfologiche, del telerilevamento, dell'*optical sorting* e della visione artificiale.
La segmentazione conta molte applicazioni pratiche, tra cui il *Medical Imaging* che non solo si occupa di individuare e diagnosticare tumori o altre patologie, dalle analisi di tomografie assiali computerizzate (TAC) e radiografie, e di misurare il volume dei tessuti, ma pone anche le basi per la chirurgia *computer-guided*. Anche l'identificaazione delle impronte digitali e i sistemi di riconoscimento del traffico sono alcuni esempi di come si possa utilizzare la segmentazione di immagini digitali.

1.3 Tecniche di filtraggio

Un filtro digitale o filtro numerico é un filtro che permette di compiere alcune funzioni matematiche su dei campioni di segnali discreti nel tempo, al fine di aumentare o ridurre alcuni aspetti del segnale analizzato.
Si distinguono filtri *passa-basso*, che attenuano le alte frequenze dell'immagine lasciando inalterate le basse in modo tale da aumentare il rapporto tra segnale e rumore dell'immagine, e filtri *passa-alto* che, agendo nel modo opposto, ne esaltano i contorni.

Queste funzioni possono essere lineari o meno. In particolare, il filtraggio lineare avviene tramite la convoluzione dell'immagine *f(x,y)* con la funzione di filtro *h(x,y)*, detta *kernel*, che si rappresenta tramite una "matrice", chiamata maschera. Si agisce sempre localmente sul dominio spaziale dell'immagine e le dimensioni della maschera (s×t) sono molto inferiori rispetto a quelle dell'immagine. Fissato il kernel

$H_{1,1}$	$H_{1,2}$	$\cdots$	$H_{1,t}$
$H_{2,1}$	$H_{2,2}$	$\cdots$	$H_{2,t}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$H_{s,1}$	$H_{,2}$	$\cdots$	$H_{s,t}$

Tabella 1.1: Filtro digitale

lo si fa scorrere su tutta l'immagine, partendo dall'angolo in alto a destra, in modo che resti sempre tutto interno all'immagine: ad ogni posizione corrisponde un singolo pixel di output. Se fissiamo una maschera quadrata s×s (con s dispari) e la centriamo sul pixel (i,j) dell'immagine, allora la convoluzione restituisce il valore

$$\mathbf{g(i,j)} = \sum_{u=-\frac{s-1}{2}}^{\frac{s-1}{2}} \sum_{v=-\frac{s-1}{2}}^{\frac{s-1}{2}} h(u,v) f(i-u, j-v) \tag{1.1}$$

da sostiuire a f(i,j). Per i pixel sui bordi, si esegue l'operazione di convoluzione sui soli pixel interni all'immagine; il filtraggio rende dunque un output delle stesse dimensioni dell'input.
Ecco alcuni filtri lineari.

1.3.1 Filtro a media

Il filtro a media é un filtro di tipo passa-bassa che si basa sulla semplice idea di calcolare la convoluzione con un kernel a valori uniformi, dove la somma dei coefficienti H(i,j) é sempre pari a 1, in modo da preservare l'energia

dell'immagine (calcolabile come integrale dei valori). Fissata una maschera del tipo

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

Tabella 1.2: Filtro a media

si calcola la media dei pixel che appartengono al piccolo intorno 3×3 del pixel centrale, ottenendo come output un'immagine *"smoothed"*, dove sono state abbattute le alte frequenze. Questo risultato é utile se l'immagine in input é affetta da molto rumore (rumore sale e pepe o rumore gaussiano): il disturbo sará decisamente meno vistoso, a scapito peró della nitidezza dell'intera immagine. Per questo motivo, d'altra parte, il filtro a media si presta anche nei casi in cui l'input sia ben definito, per sfuocare l'immagine (il che corrisponde ad abbassare proprio le frequenze) e ingrossare i contorni tra le figure.

1.3.2 Filtro Gaussiano

Il filtro gaussiano é un operatore di *smoothing*, come il filtro a media, e utilizza un kernel costruito come approssimazione discreta della distribuzione gaussiana (continua) 2-dimensionale

$$\mathbf{G}(\mathbf{x}, \mathbf{y}) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1.2}$$

dove σ é la deviazione standard e si assume media nulla.
Si approssima G(x,y) = 0 a distanze maggiori di 3σ dal valore medio, cosí si tronca la maschera di convoluzione a dimensioni finite. Se fissiamo $\sigma = 1$ il kernel diventa

$\frac{1}{273}\cdot$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Tabella 1.3: Filtro gaussiano

In generale, questo tipo di filtro compie una media pesata sull'intorno quadrato del pixel centrale che risiede nella posizione di maggior rilievo.
Dal punto di vista computazionale, la convoluzione con questo kernel é estremamente veloce, poiché separabile nelle sue componenti x e y. L'espressione

$$\mathbf{G}(\mathbf{x},\mathbf{y}) = \frac{1}{2\pi\sigma^2}\cdot e^{-\frac{x^2+y^2}{2\sigma^2}} = \left[\frac{1}{\sqrt{2\pi}\sigma}\cdot e^{-\frac{x^2}{2\sigma^2}}\right]\cdot\left[\frac{1}{\sqrt{2\pi}\sigma}\cdot e^{-\frac{y^2}{2\sigma^2}}\right] = \mathbf{G}(\mathbf{x})\cdot\mathbf{G}(\mathbf{y}) \tag{1.3}$$

suggeruisce che la convoluzione 2-D equivale alla convoluzione con un kernel di Gauss 1-dimensionale nella direzione x, poi con lo stesso kernel nella direzione y, orientato cioé in verticale. La proprietá di essere una funzione a variabili separabili fa sí che siano sufficienti 2n operazioni per la convoluzione con un kernel di dimensione n×n, contro le n^2 di un generico processo analogo; questo velocizza sensibilmente l'esecuzione dell'operatore.

1.3.3 Laplaciano di Gaussiana

Il filtro Laplaciano di Gaussiana $L \circ G$ é un filtro passo-alto, capace di evidenziare le regioni con rapidi cambi di intensitá, quindi lo si uilizza anche per far risaltare i contorni stessi delle regioni.
Presa in input l'immagine f(x,y), il laplaciano dell'immagine é

$$\mathbf{L}(\mathbf{x},\mathbf{y}) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

In quanto operatore lineare, il laplaciano é calcolato convolvendo l'immagine con un kernel discreto; le maschere piú comuni sono quelle mostrate nelle tabelle 1.4,

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

-1	2	-1
2	-4	2
-1	2	-1

Tabella 1.4: Filtri a 5 punti, a 9 punti e a 5 punti pesati

rispettivamente a 5 punti, 9 punti e 9 punti pesati.
Questo operatore aumenta notevolmente il rumore, per cui si prefiltra l'immagine di input con un filtro passo-basso di tipo Gaussiano, che limita il range di frequenze presenti nell'immagine, prima che venga applicato il filtro differenziale. In questo modo, si applicano in sequenza due filtri lineari (uno passo-basso e poi uno passo-alto) e per la proprietá associativa, si ottiene

$$\mathbf{g}(\mathbf{x},\mathbf{y}) = L(x,y) \circ (G(x,y) \circ f(x,y)) = (L(x,y) \circ G(x,y)) \circ f(x,y)$$

dove il laplaciano é convoluto con l'operatore di Gauss. Questo passaggio si calcola analiticamente come derivata seconda della distribuzione di Gauss, ossia come

$$\mathbf{L} \circ \mathbf{G}(\mathbf{x},\mathbf{y}) = -\frac{1}{\pi\sigma^4} \cdot \left[1 - \frac{x^2+y^2}{2\sigma^2}\right] \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1.4}$$

Tale operatore é il filtro Laplaciano della gaussiana, che utilizza come maschera l'approssimazione discreta del grafico di $L \circ G(x,y)$, mostrata in tabella 1.5

1.4 Estrazione dei contorni

Lo scopo dell'estrazione dei contorni (*edge detection*) é marcare i punti di un'immagine digitale in cui l'intensitá luminosa cambia bruscamente, per ricavarne le strutture di interfaccia (di salto) tra aree con luminositá uniforme.

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

Tabella 1.5: Filtro LoG

Gli algoritmi piú primitivi di estrazione consistono in semplici maschere 3×3, i cui coefficienti sono pesati attorno al termine centrale. Se si vuole localizzare un punto solitario all'interno dell'immagine, si ricorre al filtro

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

Tabella 1.6: Filtro per punto isolato

che ricorda i filtri a media giá analizzati, ma evidenzia solo quei pixel in forte contrasto coi circostanti. Successivamente, infatti, si calcola $\mathbf{g}(\mathbf{i},\mathbf{j})$ come da equazione 1.1 e si verifica se

$$| \mathbf{g}(\mathbf{i},\mathbf{j}) | \geq T$$

dove T é un valore non negativo. In caso affermativo, il pixel (i, j) é un punto isolato.

Il riconoscimento dei contorni costituisce spesso un grosso problema: non esiste un criterio univoco per fissare quanto valga la differenza di intensitá

fra due pixel, per poter affermare che fra di essi sia presente un contorno.
Comunemente, l'operazione di rilevamento dei contorni genera immagini contenenti molte meno informazioni rispetto alle originali, poiché si eliminano i dettagli degli oggetti presenti nell'immagine, conservandone solo le informazioni essenziali per descriverne la forma e le caratteristiche strutturali/geomentriche.
Esistono molti metodi per discernere i bordi delle figure e si dividono in due categorie:

metodi search-based i metodi basati sulla ricerca individuano i contorni cercando i massimi e i minimi della derivata di primo ordine dell'immagine, determinando la direzione in cui si ha il massimo gradiente locale;

metodi zero-crossing i metodi basati sull'attraversamento dello zero sono quelli che cercano i punti in cui il laplaciano si annulla.

Discretizzazione delle derivate

Lavorando con immagini digitali, occorre lavorare con le derivate discrete. Si ricorda che, fissata la funzione *I(x,y)* , le derivate parziali discrete nel punto (x_i, y_j) sono definite come

$$\frac{\partial \mathbf{I}(\mathbf{x_i}, \mathbf{y_j})}{\partial \mathbf{x}} = \frac{I(x_{i+1}, y_j) - I(x_{i-1}, y_j)}{2h} + O(h^2)$$

$$\frac{\partial \mathbf{I}(\mathbf{x_i}, \mathbf{y_j})}{\partial \mathbf{y}} = \frac{I(x_i, y_{j+1}) - I(x_i, y_{j-1})}{2k} + O(k^2)$$

e

$$\frac{\partial^{\mathbf{2}} \mathbf{I}(\mathbf{x_i}, \mathbf{y_j})}{\partial \mathbf{x^2}} = \frac{I(x_{i+1}, y_j) + 2 \cdot I(x_i, y_j) + I(x_{i-1}, y_j)}{h^2} + O(h^2)$$

$$\frac{\partial^{\mathbf{2}} \mathbf{I}(\mathbf{x_i}, \mathbf{y_j})}{\partial \mathbf{y^2}} = \frac{I(x_i, y_{j+1}) + 2 \cdot I(x_i, y_j) + I(x_i, y_{j-1})}{k^2} + O(k^2)$$

$$\frac{\partial^{\mathbf{2}} \mathbf{I}(\mathbf{x_i}, \mathbf{y_j})}{\partial \mathbf{x} \partial \mathbf{y}} = \frac{I(x_{i+1}, y_{j+1}) - I(x_{i-1}, y_{j+1}) - I(x_{i+1}, y_{j-1}) + I(x_{i-1}, y_{j-1})}{4hk} + O\left((h+k)^2\right)$$

Qua *I(x,y)* rappresenta il valore dell'intensitá nel pixel *(x,y)*. Il gradiente della funzione I(x,y) é, per ciascun punto (x,y) dell'immagine, il vettore bi-dimensionale le cui componenti sono le derivate del valore della luminositá in direzione orizzontale $\left(\frac{\partial I(x,y)}{\partial x}\right)$ e verticale $\left(\frac{\partial I(x,y)}{\partial y}\right)$. Tale vettore punta nella direzione del massimo aumento possibile di luminositá e la lunghezza del vettore corrisponde alla rapiditá con cui la luminositá stessa cambia, spostandosi in quella direzione. Ad esempio, se l'intensitá dei pixel é costante in una regione, in questa regione il gradiente é nullo. Al contrario, se si ha un contorno netto, il gradiente ha alti valori.

In realtá, in molti operatori il calcolo del gradiente é approssimato, per cui é sufficiente considerare le derivate 1-dimensionali

$$I'(x) = -\frac{1}{2} \cdot I(x-1) + 0 \cdot I(x) + \frac{1}{2} \cdot I(x+1) \tag{1.5}$$

e

$$I''(x) = 1 \cdot I(x-1) + 2 \cdot I(x) + 1 \cdot I(x+1) \tag{1.6}$$

Si consideri un pixel dell'immagine e si fissi un suo intorno quadrato

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Tabella 1.7: Schematizzazione dell'intorno di un pixel

centrato nel pixel stesso (z_5). In ciascun operatore, il calcolo del gradiente si implementa utilizzando delle maschere 2×2 o 3×3(ottenute dai vettori dei coefficienti delle discretizzazioni appena viste) mostrate nella tabella 1.8

$\frac{1}{2}\cdot$

-1	0	1

1	2	1

Tabella 1.8: Vettori per le derivate prima e seconda

1.4.1 Riconoscimento di linee

Le seguenti maschere permettono di individuare segmenti verticali, orizzontali e inclinati di +45° oppure di -45°. Si vede che sono ottenute dagli array sopra citati.

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

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

Tabella 1.9: Filtri per righe verticali e orizzontali

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

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

Tabella 1.10: Filtro per righe a +45 ° e -45°

1.4.2 Operatore di Roberts

L'operatore di Roberts é stato uno dei primi filtri digitali, utilizzato dalla visione artificiale nell'ambito dell'*edge detection*. É un operatore di primo ordine a passa-alto: utilizza infatti la coppia di kernel 2×2 che individuano principalmente i contorni orientati come le diagonali principale e secondaria, rispetto alla griglia dei pixel.

+1	0
0	-1

0	+1
-1	0

Tabella 1.11: Filtri per G_x e G_y

Le maschere di convoluzione mostrate nelle tabelle 1.11, possono essere applicate all'immagine di input separatamente, per produrre le due componenti del gradiente G=(G_x, G_y), approssimate come

$$G_x = z_9 - z_5$$

$$G_y = z_8 - z_6$$

Invece, per ottenere una sola immagine di output, si calcola il modulo $|G| = \sqrt{{G_x}^2 + {G_y}^2}$ oppure la sua approssimazione $|G| = |G_x| + |G_y|$, che é piú veloce da stimare. Infine, si ricava l'angolo di orientazione del contorno, calcolando $\theta = \arctan\left(\frac{G_y}{G_x}\right)$.

Questo metodo é ancora uilizzato per la sua semplicitá e velocitá computazionale, sebbene non sia uno strumento molto valido: amplifica infatti il rumore ed evidenzia anche le minime variazioni, fornendo troppi contorni. É dunque necessario introdurre un filtro passa-basso.

1.4.3 Operatore di Sobel

L'operatore di Sobel é un altro algoritmo che calcola il gradiente della luminositá in ciascun punto dell'immagine, trovando la direzione lungo la quale si ha il massimo incremento possibile dallo scuro al chiaro, attraverso un eventuale contorno. É simile all'operatore di Roberts, ma é decisamente piú robusto (ossia meno sensibile al rumore) poiché utilizza una coppia di kernel 3×3 per stimare il gradiente nelle direzioni di x e y. Tali maschere sono mostrate nelle tabelle 1.12

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

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

Tabella 1.12: Filtri per G_x e G_y

Ciascuna di essa viene convoluta con l'immagine di input, ottenendo le due

immagini coi valori di G_x e G_y, qui approssimate come

$$G_x = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

$$G_y = (z_1 + 2z_2 + z_3) - (z_7 + 2z_8 + z_9)$$

Successivamente si calcola il modulo del gradiente come $|G| = \sqrt{{G_x}^2 + {G_y}^2}$ oppure con l'approssimazione $|G| = |G_x| + |G_y|$. Ancora si puó ricavare la direzione del gradiente calcolando $\theta = \arctan\left(\frac{G_y}{G_x}\right)$.

Si osserva che la prima maschera é data dalla convoluzione

$$G_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

per cui si capisce che il valore G_x si puó calcolare anche con la discretizzazione a tre punti della derivata prima, su x, e con la successiva media pesata su tre punti verticali, a meno del fattore moltiplicativo $\frac{1}{8} = \frac{1}{2} \cdot \frac{1}{4}$. Analogamente la seconda maschera puó essere pensata come media pesata sulle x e derivata discreta lungo le y. Questo modo di stimare le derivate é vantaggioso, perché richiede meno oprazioni su ciascun punto dell'immagine.
Infine, in questo caso si parla di filtro passa-banda poiché si unisce l'effetto del passo-alto (col gradiente) a quello del passo-basso (col filtro a media) e vediamo che la somma dei coefficienti di ciascuna maschera é nulla.

1.4.4 Operatore di Canny

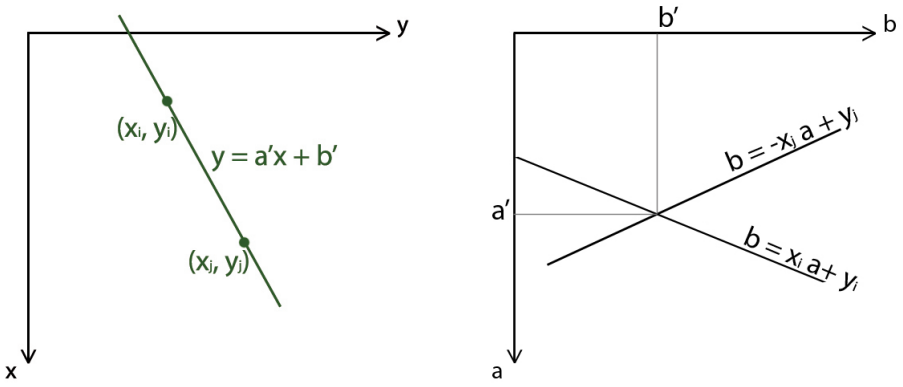
L'operatore di Canny é l'estrattore di contorni piú robusto e risale al 1986. L'algoritmo si divide in tre passaggi: inizialmente si usa un filtro gaussiano (filtro passa-basso) per ridurre il rumore a cui é sensibile il rilevatore di contorni, poi un operatore per il calcolo del gradiente (ad esempio quello di Sobel) individua le regioni candidate ad essere contorni e infine si applica un algoritmo per determinare se un pixel fa parte o meno della cresta di intensitá che caratterizza i veri contorni. In particolare, quest'ultimo processo pone a zero i pixel che non sono contorni, lasciando nell'output delle sottili linee, e

prende il nome di *soppressione dei non-massimali*.
Una caratteristica importante di questo algoritmo é la presenza di un ciclo di isteresi: l'inseguimento del pixel di cresta parte da un pixel di cresta con gradiente al di sopra di una soglia T1 abbastanza alta, poi continua nelle due direzioni orizzontale e verticale selezionando altri pixel di cresta col gradiente superiore alla soglia 2, dove $T2 \ll T1$. Tale ciclo permette di estrarre contorni anche da immagini con un basso rapporto segnale-rumore.

É chairo che entrano in gioco tre parametri: le soglie T1, T2 dell'isteresi e la dimensione del filtro gaussiano σ. La soglia T1 determina i punti di partenza: scegliendo un valore alto, si aumenta il numero di bordi evidenziati, col rischio di contornare troppi oggetti dell'immagine, se non zone di solo rumore. La soglia T2 determina invece i punti di arrivo, ossia la lunghezza dei contorni: piú alto si fissa T2, piú corto sará il bordo. É difficile settare dei valori che si adattino bene a tutta l'immagine, quindi si cerca di determinare il range dinamico per ciascun parametro e si tiene conto della quantitá T1-T2. Il valore di σ usato nel filtro gaussiano determina lo *smoothing* iniziale, quindi il raggruppamento di diversi dettagli e le dimensioni dei contorni da cercare: maggior σ porta all'identificazione di piú contorni in uno.

1.4.5 Line Detection con la trasformata di Hough

Gli algorimi di *edge detection* visti finora sono spesso seguiti da procedure che assemblano i pixel identificati in contorni piú significativi. Un approccio possibile per individuare e unire i segmenti rettilinei di un'immagine é quello elaborato nel 1972, sulla base della trasformata di Hough (1962).
Presa un'immagine, consideriamo il pixel (x_i, y_i) e tutte le linee che passano per il punto; tutte queste rette soddisfano l'equazione $y_i = a \cdot x_i + b$ per un'opportuna scelta di (a, b).



Nell' *ab-piano* dei parametri, l'equazione $b = -x_i a + y_i$ identifica una singola retta, $\forall(x_i, y_i)$. Se si fissa un secondo pixel (x_j, y_j), la retta $b = -x_j a + y_j$ interseca la prima nel punto (a', b'), dove a' rappresenta la pendenza della retta passante sia per (x_i, y_i) che per (x_j, y_j), vista nel piano XY, mentre b' ne sará il termine noto. In generale, le rette nel piano dei parameri corrispondono a tutti i pixel (x_i, y_i) che possono essere rilevati: in questo modo, i contorni dell'immagine possono essere identificati come zone dove si intersecano molte linee dello spazio dei parametri.
Un problema tecnico di questa trasformata é che una retta, con direzione quasi orizzontale, é rappresentata da una pendenza a che approssima l'infinito. Una soluzione sta nella rappresentazione polare della retta:

$$x \cdot \cos\theta + y \cdot \sin\theta = \rho$$

In queso modo, infatti, una linea orizzontale ha θ=0 ma ρ=x . Ora, ogni curva sinusoidale $x_i \cos\theta + y_i \sin\theta = \rho$ del piano $\rho\theta$ rappresenta una famiglia di linee che passano per il pixel fissato (x_i, y_i) e il punto (ρ', θ') di intersezione con $x_j \cos\theta + y_j \sin\theta = \rho$ corrisponde alla linea che passa per entrambi i punti.

La trasformata di Hough é comoda perché si puó dividere lo spazio dei parametri nelle cosiddette *accumulator cells*. In particolare, gli intervalli attesi per i parametri $(\theta_{min}, \theta_{max})$ e (ρ_{min}, ρ_{max}), solitamente equivalgono al massimo a $-90° \leq \ \theta \leq \ +90°$ e $-D \leq \ \rho \leq \ +D$, dove D é la distanza fra

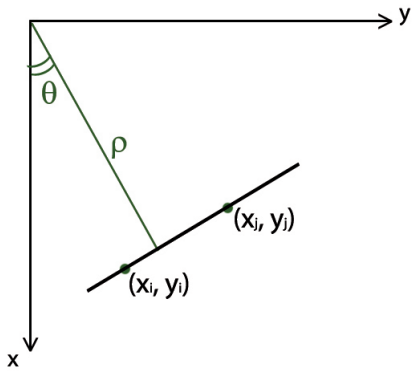


Figura 1.1: Piano $\rho\theta$ per la trasformata di Hough

due angoli dell'immagine. Si definisce poi un array 2-dimensionale di accumulazione, dove si setta A(i,j)=0 $\forall$ cella di coordiante (i,j), che corrisponde al quadrato in cui si trova (ρ_i, θ_j). Successivamente, per ogni punto (x_k, y_k) che non sia di sfondo nell'immagine piana, si lascia θ uguale a ciascun valore con cui é stato suddiviso il suo intervallo di dominio, mentre si calcolano i corrispondenti valori di ρ come $\rho = x_k \cos\theta + y_k \sin\theta$. I valori trovati vengono poi arrotondati al valore piú vicino corrispondente a una cella di accumulazione: per tale cella si incrementa il valore di una unitá. In questo modo, alla fine della procedura, avere A(i,j)=Q significa che Q punti del piano XY giacciono nella linea $\rho_i = x\cos\theta_j + y\sin\theta_j$. Chiaramente, maggiore é il numero di suddivisioni del piano $\rho\theta$, maggiore sará la precisione dell'algorimo.

Capitolo 2

Metodi numerici di segmentazione

Comunemente, gli operatori di estrazione dei contorni non sono in grado di dare in output dei contorni completi e anche l'operatore di Canny restituisce dei bordi frammentati, quindi esiste un gap tra l'immagine e la sua segmentazione in contorni chiusi. In particolare, il miglior metodo di estrazione dei contorni non potrá mai uguagliare i risultati del cervello umano; un esempio famoso é dato dal *triangolo di Kanizsa*

Figura 2.1: Triangolo di Kanizsa

dove si distingue un triangolo, con tanto di contorni chiusi, fissando il centro della figura. La corteccia visiva é dunque in grado di connettere dei contorni anche molo frammentati,. fino al caso estremo del triangolo qui mostrato.
Dal punto di vista applicativo, occorre sviluppare una tecnica di segmen-

tazione che superi i limiti dell'estrazione dei contorni, soprattutto se si desidera che il risultato finale abbia i bordi chiusi.

2.1 Sogliatura

L'operazione di sogliatura offre un modo molto semplice per segmentare le regioni di interesse e le regioni di sfondo, sulla base dell'intensitá luminosa. L'immagine di input é solitamente una immagine a toni di grigio, piú raramente a colori; l'output é sempre un'immagine binaria in cui i pixel neri identificano lo sfondo e quelli bianchi le figure (o viceversa).
Sia data in input un'immagine f(x,y) in bianco e nero, si fissi una gradazione di grigio, detta soglia di intensitá **T**. Si calcola l'immagine "sogliata" g(x,y) determinando per ogni pixel (x,y), il valore

$$g(x,y) = \begin{cases} 1 & f(x,y) \geq T \\ 0 & f(x,y) < T \end{cases}$$

La sogliatura é normalizzata, cosí i pixel etichettati con 1 sono detti *object points* poiché corrispondono agli oggetti dell'immagine, mentre quelli fissati a 0 sono i *background points* dello sfondo.
L'esito della segmentazione é fortemente influenzato dalla scelta del parametro T, che puó essere costante per tutta la procedura (*sogliatura globale*) oppure puó variare dinamicamente da pixel a pixel (*sogliatura locale*). Tuttavia, questo metodo cosí primitivo si presta bene solo per immagini semplici, con pochi oggetti posti su sfondi uniformi.

2.1.1 Sogliatura globale

Si considera preliminarmente un caso molto fortunato: su uno sfondo scuro sono illuminati degli oggetti. L'istogramma di questa immagine mostrerá i livelli di intensitá raggruppati attorno a due principali valori mediani. Viene abbastanza ovvio scegliere il valore di T come l'intensitá che separa le due mode.

Un altro modo per scegliere un valore opporuno per T é il *"trial and error"*, che consiste semplicemente nel provare piú valori, fino a determinarne uno che renda un output soddisfacente. Questo sta alla base di molte funzionalitá offerte da diversi programmi di elaborazione grafica, di uso anche commercale, come Photoshop: tramite ambienti interattivi, un utente puó testare diversi valori e vederne immediatamente i risultati.
Nel 2002, Gonzales e Woods hanno elaborato un ulteriore algorimo, iterativo, per scegliere automaticamente il valore della soglia; eccone i passaggi:

1. si fissa un valore iniziale per T, solitamente come valor medio tra il massimo e il minimo dell'intensitá dell'immagine in esame;

2. si divide l'immagine nelle regioni G_1, costituita dai pixel con f(x,y)$\geq$ T, e G_2, costituita dai pixel con f(x,y) $<$ T ;

3. si calcolano le medie delle intensitá, μ_1 e μ_2, rispettivamente per i pixel di G_1 e G_2;

4. si determina il nuovo valore di soglia come
$$T = \frac{1}{2}(\mu_1 + \mu_2);$$

5. si ripetono i passi 2-3-4 fino a che la differenza tra i valori calcolati per T da successive iterazioni é minore di un predefinito parametro t_0.

2.1.2 Sogliatura locale

La sogliatura con paramentro T costante fallisce nei numerosi casi in cui il contenuto dell'immagine non é illuminato in maniera uniforme: un rimedio efficace consiste nella sogliatura locale.
Questo algoritmo varia il valore di soglia T secondo la legge
$$T(x, y) = f_0(x, y) + T_0$$
dove $f_0(x, y)$ é il valore di aperura dell'immagine e T_0 é il valore stimato dell'algoritmo di Gonzalez-Woods, applicato sull'intera immagine.

Cosí facendo, la sogliatura locale

$$g(x,y) = \begin{cases} 1 & f(x,y) \geq T(x,y) \\ 0 & f(x,y) < T(x,y) \end{cases}$$

corrisponde a un processo di sogliatura globale, applicato a un'immagine pre-elaborata per ridurre i problemi dovuti alla scarsa illuminazione.

2.2 Segmentatione region-based

Ora si analizzano dei metodi di segmentazione che distinguono direttamente le regioni di un'immagine, senza appoggiarsi alla ricerca delle discontinuitá dei livelli di illuminazione per ricavare i contorni.

Sia R la regione corrispondente all'intera immagine e si pensi alla segmentazione come a un processo di partizione, dell'immagine stessa, in n sottoregioni $R_1, R_2, \ldots, R_n$ tali che

1. $\bigcap_{i=1}^{n} R_i = R$
2. $R_i \cap R_j = \varnothing \ , \forall i \not\equiv \ j$
3. R_i é una regione connessa, $\forall i = 1, 2, \ldots, n$
4. $P(R_i) =$ TRUE, $\forall i = 1, 2, \ldots, n$
5. $P(R_i \cup R_j) =$ FALSE, $\forall R_i, R_j$ regioni adiacenti

dove P(R_i) é un predicato logico definito sui punti delle regioni, ad esempio si usa spesso porre $P(R_i) =$ TRUE se tutti i pixel della regione i^{esima} hanno lo stesso livello di grigio (o comunque una simile intnsitá). In tal caso, mentre le prime due condizioni definiscono una partizione dell'immagine, l'ultimo requisito impone che le regioni adiacenti corrispondano a diverse tonalitá di grigio. Il terzo vincolo, invece, stabilisce che ciascuna regione non puó essere disgiunta in aree separate.

2.2.1 Metodo region growing

Il metodo basilare per questo tipo si segmentazione é il cosiddetto *region growing*. Come giá suggerisce il nome, il *region growing* raggruppa i pixel in regioni sempre piú estese, secondo un criterio di crescita ed ampliamento. Si parte da un insieme di *seed points* e da tali "semi" si sviluppano le regioni: passo a passo si estende l'area corrispondente al seme con l'aggiunta dei pixel circostanti che soddisfano le proprietá sopra elencate.

Il risultato della segmentazione é subordinato alla scelta dei semi. Se si possiede l'istogramma dell'immagine, si possono scegliere i semi in modo che corrispondano a valori piú alti del grafico. In generale, se non si hanno informazioni riguardanti l'immagine, per ciascun pixel si eseguono tutti i passaggi che assegnerebbero il pixel a una regione e, infine, si considerano le regioni ottenute analizzandone i valori centrali. Tali valori saranno i semi.

Un altro parametro importante é la *misura di somiglianza* δ, cioé la massima differenza tollerabile tra l'intensitá del pixel in esame e quella che caratterizza la regione in cui accorpare il pixel stesso. Una giusta scelta di δ raffina molto il risultato finale di segmentazione: ancora, la conoscenza dell'istogramma puó indirizzare verso un'opportuna stima del parametro.

Con questi accorgimenti, il metodo della *region growing* é un algoritmo semplice che si presta bene alla segmentazione di molte immagini. In realtá, la presenza di rumore o di brusche variazioni di intensitá porta spesso all'*oversegmentation*: per evitare questa eccessiva suddivisione, in regioni prive di interesse, si impone una soglia minima che regoli le aree delle regioni individuate.

2.2.2 Metodo splitting-and-merging

Un metodo di segmentazione *region-based*, alternativo a quello dei semi, consiste nel frazionare l'immagine iniziale in un insieme di arbitrarie e disgiunte sotto-regioni, per poi unirle o dividerle ulteriormente in modo che siano

verificate le cinque proprietá sopra elencate. Si fissi ancora un predicato P e sia R la regione corrispondente all'intera immagine di input. Successivamene si divida R in quadranti e per ciascuno si calcoli $P(R_i)$. Se $P(R_i) = FALSE$, si divide R_i in quadranti ancora piú piccoli e si itera il procedimento finché $P(R_i) = TRUE$ per ogni sotto-regione R_i.
Questo algoritmo ha una conveniente rappresenazione nel cosiddetto *quadree*, una albero in cui da ciascun nodo si diramano esattamente quattro rami. La radice dell'albero corrisponde invece all'immagine intera mentre il nodo é una regione per cui $P(R_i) = FALSE$. Suddividendo in quadranti molto piccoli, si arriverá a un punto in cui regioni adiacenti godono di proprietá identiche: se $P(R_i \cup R_j) = TRUE$, si uniscono le regioni, sempre in accordo con le cinque condizioni del metodo. L'algoritmo ha fine quando non é piú possibile unire regioni confinanti.

2.3 Segmentazione con la trasformata Watershed

Alla base della segmentazione con la "trasformata dello spartiacque", c'é l'idea che un'immagine a toni di grigio si possa interpretare come un rilievo topografico, in cui il livello di grigio f(x,y) di un pixel corrisponde all'altitudine del rilievo. In questo modo, una goccia d'acqua, che cade sul rilievo topografico, scorre lungo un percorso per raggiungere il bacino di raccolta, che corrisponde a un minimo locale. Cosí come in geografia ogni bacino idrografico é separato da quelli contigui dalla cosiddetta linea dello spartiacque, nell'elaborazione di immagini diverse linee di spartiacque rappresentano i contorni di oggetti. Allora, presa un'immagine di input, occorre elaborarla per ottenerne un'altra dove si riescano a idenificare i bacini di raccolta .
A tale scopo, sono stati implementati gli algoritmi illustrati in seguito.

2.3.1 Segmentazione watershed con la distance transform

Uno strumento ampiamente utilizzato come supporto per la *watershed segmentation* é la *distance transform*. Questa trasformazione agisce su immagini binarie, determinando per ciascun pixel la distanza dal pixel piú vicino, il cui valore sia diverso da 0. Chiaramente, ai pixel di valore 1 é associata una distanza nulla.
Presa una generica immagine di input, la si converte in immagine binaria e, se necessario, se ne calcola la complementare in modo da lasciare lo sfondo bianco. Successivamente la *distance transform* fornisce un'immagine nera, con zone bianche in concomitanza degli oggetti. Calcolata l'immagine in negativo di quest'ultimo output, si applica la *watershed transform*.
Cosí impostato, il metodo é efficace ma molte volte si incombe nell'*oversegmentation* e il vasto numero di regioni individuate rende addirittura l'output inutile; per contrastare questo problema sono stati sviluppati altri algoritmi.

2.3.2 Segmentazione watershed con il gradiente

L'intensitá del gradiente é spesso utilizzata per preprocessare immagini a toni di grigio, prima di segmentarla con la *watershed transform*: l'immagine che rappresenta il gradiente ha infatti pixel con alti valori laddove l'immagine originale presentava i contorni degli oggetti.
In realtá, cosí non si ottengono giá dei risultati soddisfacenti perché continuano ad esserci problemi di *oversegmentation*: smussare l'immagine del gradiente, prima di passare alla *watershed transform*, riduce sensibilmente questa complicazione.

2.3.3 Segmentazione watershed con marker-controll

Il metodo appena analizzato fallisce se l'immagine originale presenta oggetti sovrapposti o che si tocchino: questo é di certo uno dei massimi scogli per l'elaborazione di immagini digitali. Inoltre, applicare direttamente la*watershed*

transform all'immagine del gradiente riduce ma non elimina il problema dell'*oversegmentation*, perché l'immagine calcolata presenta rumore o altre irregolaritá.
Una soluzione, adottata da molti, consiste in una pre-elaborazione delle immagini che distingue oggetti in primo piano e aree di sfondo, evidenziandone i cosiddetti marcatori (ossia piccoli insiemi, connessi, di pixel). In particolare, si cercano i *marcatori interni*, dentro ciascun oggetto in primo piano, e i *marcatori esterni* nelle zone di sfondo, dove non compaiono oggetti. Riassumendo, l'algoritmo da applicare a una generica immagine di input segue dunque i seguenti passi:

1. si converte l'immagine originale in scala di grigi;
2. si utilizza il gradiente come funzione di segmentazione;
3. si individuano i marcatori interni e gli esterni;
4. si esegue la trasformata di watershed sull'immagine del gradiente.

Infine, esistono diversi metodi per individuare marcatori interni ed esterni, molti dei quali coinvolgono i filtri lineari (visti nel primo capitolo) e filtri non lineari. Non esiste un metodo migliore in assoluto, ma si determina quello piú opportuno in base alle caratterisiche dell'immagine da analizzare.

Capitolo 3

Risultati ottenuti

In questo capitolo si analizzano i risultati di alcuni metodi sopra citati, applicati a immagini in scala di grigio.

Gli algoritmi sono stati scritti col programma "MATLAB, The Language Of Technical Computing", versione 7.8.0 (R2009a). Nell'implementazione, si fa uso di alcune funzioni tratte dall'*Image Processing Toolbox (IPT)*, un pacchetto che fornisce un insieme di funzioni per la manipolazione delle immagini, la visione artificiale e l'elaborazione delle immagini.
La *DIPUM toolbox* é, invece, una collezione di circa settanta M-funzioni, tratte dal manuale "Digital Image Processing Using MATLAB" (si veda il riferimento [1, 1] nella bibliografia), a cui mi sono appoggiata nella fase di implementazione.

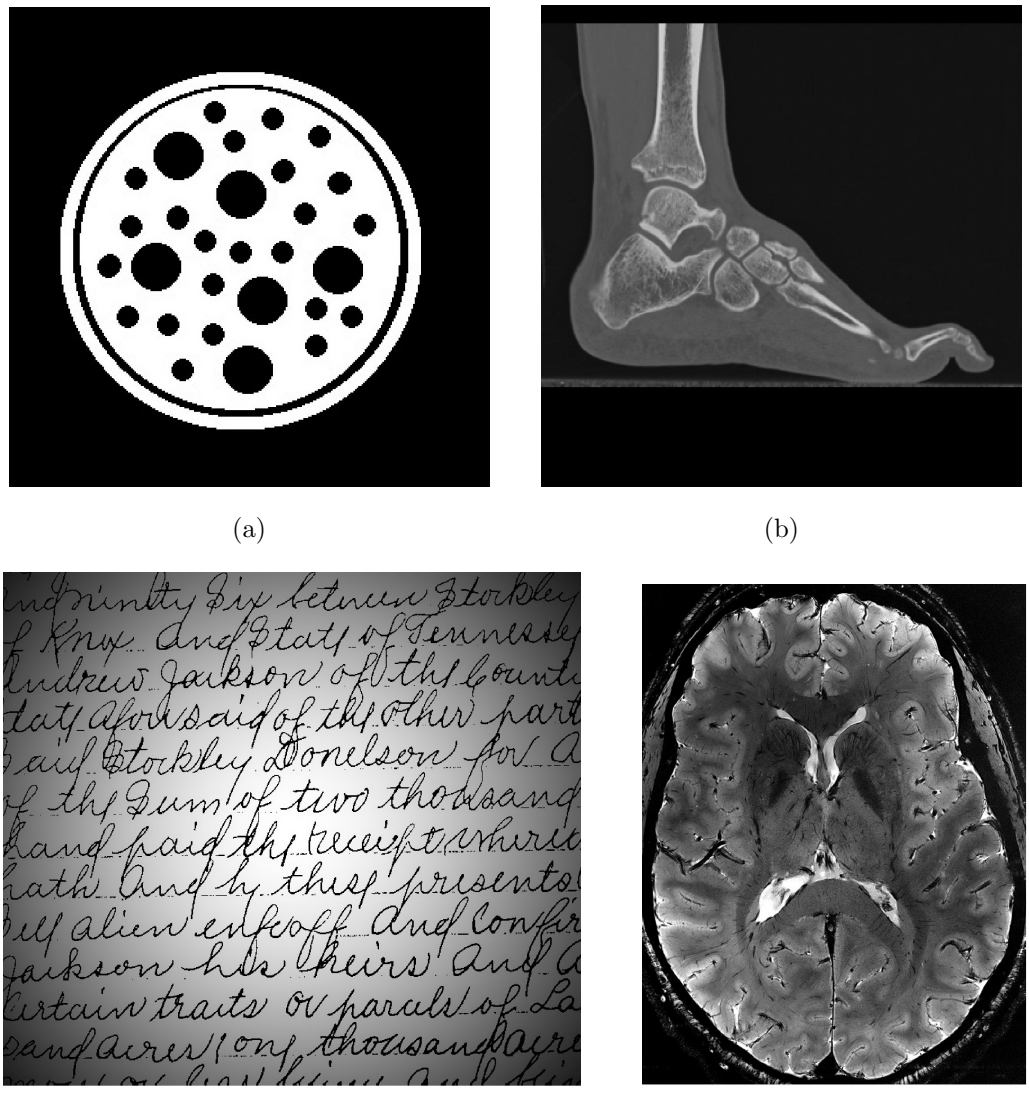
Nel corso del capitolo vengono mostrate immagini di output ottenute con lo stesso settaggio dei parametri, cosí da permettere il confronto tra i metodi stessi.
Alcune delle immagini utilizzate sono tratte da un archivio online, messo a disposizione dagli autori del testo sopra citato.

Un'ultima osservazione: per convenzione, in Matlab l'asse delle coordinate y é quello orizzontale, orientato da sinistra a destra, mentre l'asse x corrisponde al verticale, dall'alto al basso. In questo modo, l'immagine digitale

si presta meglio alla rappresentazione matriciale "riga per colonna".

Immagini campione

Queste che seguono sono le immagini su cui vengono testati gli algoritmi.



(a) (b)

(c) (d)

Figura 3.1: Immagini di input

L'immagine (a) é una semplificazione della visione al microscopio: gli elementi da segmentare sono distinti tra loro e perfettamente riconoscibili dallo sfondo, é un'immagine binaria e non é sporcata da rumore. Per questi motivi, rappresenta una buona immagine di input, dove i metodi daranno risultati piú che soddisfacenti.
L'immagine (b) é la tomografia assiale computerizzata di una caviglia: a vista, le ossa si distinguono abbastanza bene dal resto dell'immagine e la sagoma del piede si stacca nettamente dallo sfondo nero, che occupa una buona parte dell'intera immagine. Queste caratteristiche permetono di ottenere buoni risultati per l'individuazione dei contorni.
L'immagine (c) del testo é uno dei frequenti casi in cui l'illuminazione non uniforme ostacola l'elaborazione digitale. Inoltre sará interessante vedere come reagiscono gli operatori in presenza di linee sottili.
Infine, l'immagine (d) mostra una sezione di cervello umano, acquisita con una risonanza magnetica. Qui gli elementi sono molto meno distinguibili rispetto agli altri esempi e si vedrá come ció influenzi gli output.

3.1 Individuazione delle linee e dei contorni

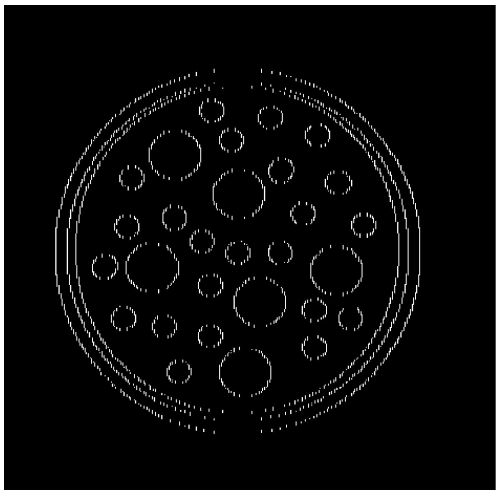
3.1.1 Filtraggio per il rilevamento di linee

Il seguente algoritmo legge in input `'image.jpg'` e ne esegue il filtraggio con la maschera `w`.

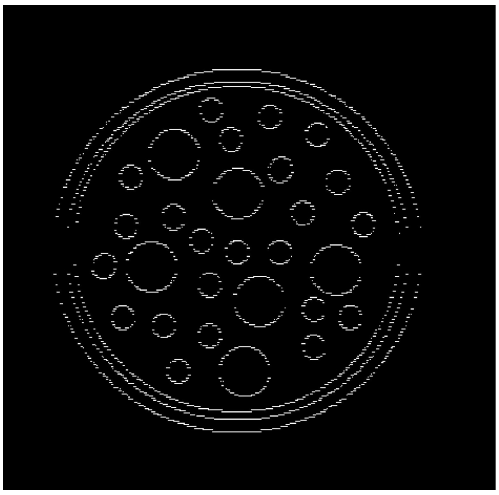
```
f= imread( 'image.jpg');
w = [ -1 2 -1; -1 2 -1; -1 2 -1];
g = imfilter ( double(f),w,'replicate' );
T = 0.3*max( abs( g(:) ) );
g = g>=T;
imshow( g )
```

Viene utilizzata la funzione `imfilter (f,w)` dell'IPT: prendendo `f` come immagine di input, restituisce l'immagine `g` delle stesse dimensioni. Suc-

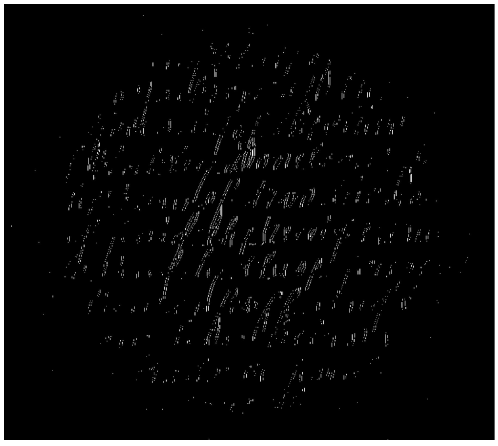
cessivamente si identificano i punti di interesse controllando se $\mathtt{g} \geq \mathtt{T}$: cosí facendo si trasforma l'output in immagine binaria.
Uilizzando i filtri visti nelle tabelle 1.9, si ottengono i risultati mostrati nelle figure 3.2.



(a) Filtraggio orizzontale



(b) Filtraggio verticale



(c) Filtraggio orizzontale

(d) Filtraggio orizzontale

Figura 3.2: Output del filtraggio

Si consideri il primo output, dove sono state rilevate circonferenze, aperte nei tratti in alto e in basso. Questo risultato é dovuto al fatto che l'algoritmo

cerca le variazioni di intensitá nella direzione orizzontale: i bordi con tale orientazione non sono dunque rilevati come contorni, poiché intesi come aree a intensitá costante. Analogamente succede col filtro per la direzione verticale, dove i contorni estratti sono aperti nelle estremitá laterali, in concomitanza dei tratti verticali.
Le figure (c) e (d) mostrano gli output del filtraggio orizzontale, su immagini in scala di grigio. I risultati sono meno nitidi poiché le variazioni tra i toni sono meno accentuate: l'immagine (c), infatti, é piú chiara nell'area centrale, dove la scrittura spiccava meglio dallo sfondo.
Si osserva complessivamente che il filtraggio ottiene buoni risulati, limitamente alle sinol direzioni.

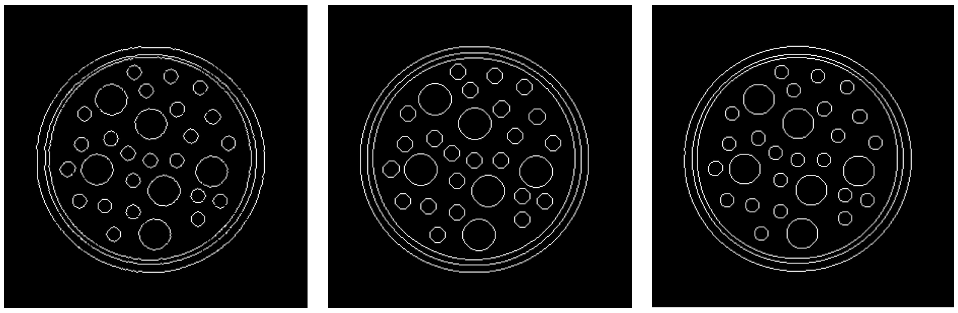
3.1.2 Operatori per l'estrazione dei contorni

L'*Image Processing Toolbox* contiene la funzione `edge (f, 'method')` che esegue il calcolo delle derivate, secondo il criterio espresso in *'method'*.

```
f= imread( 'image.jpg');
[g_sobel_default, ts] = edge ( f,'sobel' );
[g_log_default, tlog] = edge ( f,'log' );
[g_canny_default, tc] = edge ( f,'canny' );
```

La funzione `edge` restituisce un'immagine binaria `g`, i cui pixel bianchi costituiscono i contorni degli oggetti. L'altro elemento di output é un parametro numerico, che indica il valore di sogliatura usato dalla funzione per ottenere risultati migliori dal gradiente.

Si considerino le immagini mostrate in figura 3.3.
Paragonando questi output ai precedenti, si apprezza immediatamente che i contorni sono linee chiuse. Inoltre, utilizzando l'operatore di Sobel, l'LoG e quello di Canny, si osserva un ulteriore, graduale miglioramento nella detenzione dei bordi: i cerchi si fanno man mano piú esatti e quelli esterni anche piú concentrici e distinti.



(a) Sobel (b) LoG (c) Canny

Figura 3.3: Output di 'edge'

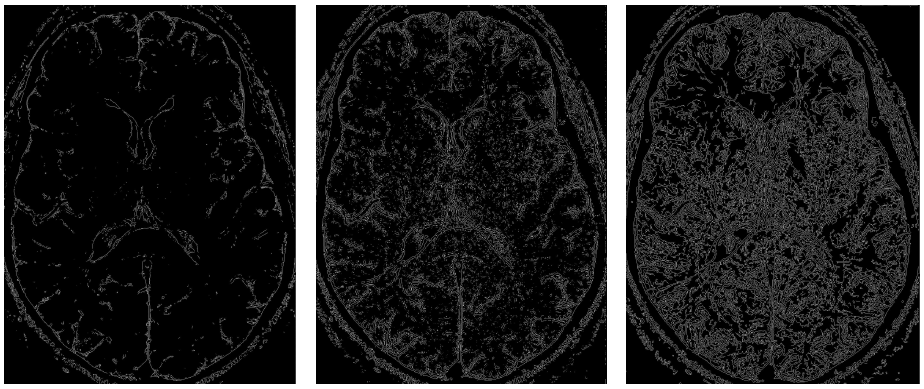
É interessante analizzare questi operatori anche con le altre immagini, dove gli oggetti non sono nettamente contrastati dallo sfondo.
Con la TAC al piede si ottengono i seguenti output:

(a) Sobel (b) LoG (c) Canny

Figura 3.4: Output di 'edge'

Si osserva che gli algoritmi LoG e, in particolar modo, quello di Canny rilevano troppe linee, alcune delle quali non identificano alcun bordo chiuso e sporcano l'immagine elaborata: questo comportamento é dovuto alle sfumature di grigio che caratterizzano i tessuti molli del piede.

Il fenomeno dell'*oversegmentation* si amplifica nell'immagine della risonanza magnetica con cui l'operatore di Canny produce una segmentazione decisamente fitta. A tal proposito, si vedano le immagini della figura 3.5.



(a) Sobel (b) LoG (c) Canny

Figura 3.5: Output di 'edge'

3.2 Risultati della segmentazione

3.2.1 Metodi di sogliatura

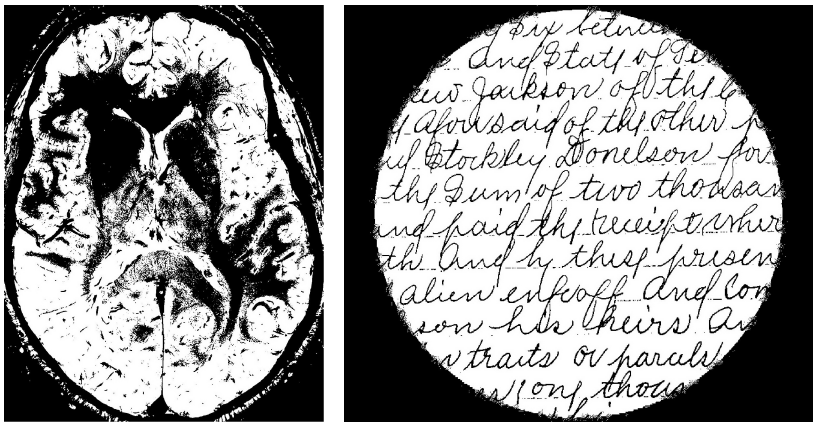
Applicando un algoritmo di sogliatura ad immagini binarie, si ottengono output idenici alle immagini di input, quindi ora si analizzano i risultati conseguiti da immagini in scala di grigio.

L'algoritmo per la sogliatura globale é molto semplice.

```
f= imread( 'image.jpg');
T = 0.5*( double(min(f(:))) + double(max(f(:))) );
done = false;
while ~done
    g = f>=T;
    Tnext = 0.5*( mean(f(g)) + mean(f(~g)) );
    done = abs( T-Tnext )<0.5;
    T = Tnext;
end
```

Letto l'input, si parte calcolando il valore `T` di soglia, come valor medio tra il minimo e il massimo valore assunto nell'immagine `f`, successivamente il cilclo

`while` raffina la scelta di `T`, procedendo come spiegato nel paragrafo 2.1.1. Eccone i risultati:



(a) (b)

Figura 3.6: Output della sogliaura

L'immagine (a) del cervello mostra un risultato discreto di binarizzazione: le anse del cervello sono abbastanza nitide nella parte superiore, mentre nella fascia inferiore i loro bordi sono solo accennati. Inoltre, é stato individuato bene il settore centrale della corteccia, a discapito della nitidezza del tronco nervoso che appare infatti punteggiato.
Si consideri ora l'immagine riportata (b): il rumore che compariva ai margini dell'immagine ha prodotto una zona completamente nera nell'output. Al contrario, i toni di grigio chiaro che permeavano la zona piú luminosa del testo, sono stati annullati e resi con il bianco che si cercava come sfondo per il testo. In particolare, in questo cerchio centrale, il testo é ovunque ben leggibile: il riconoscimento delle frasi é completo in ogni direzione, le linee scritte sono rese con il giusto spessore e, addiritura, le righe orizzontali del foglio sono quasi impercepibili.

L'algoritmo di sogliatura cosí impostato fallisce con l'esempio del piede. Si considerino le immagini riportate nella figura 3.7. L'immagine originale presenta una vasta zona nera per lo sfondo e qualche area quasi bianca per

(a) T=35.60 (b) T=102 (c) T=60

Figura 3.7: Output della sogliaura con diversi valori della soglia T

le ossa, quindi il primo valore calcolato per `T` corrisponde a un tono grigio intermedio (per precisione pari a 102), che scinde l'immagine in due regioni fortemente contrastanti. Nel ciclo `while` si calcolano dunque valori medi μ_1 e μ_2 molto distanti tra loro, che abbassano il `Tnext` a un valore molto lontano dalla prima soglia. Solo dopo molte iterazioni il ciclo termina, settando `T=35.60`, che corrisponde a una tonalitá di grigio piú scura di tutta l'area del piede; questo fa sí che la sagoma diventi un'unica regione bianca, come mostrato nell'immagine (a).
In realtá, si vede che la soglia `T=102` restituisce una migliore binarizzazione dell'immagine di input: si distinguono infatti le ossa, anche se si perde la sagoma del piede. Se si ripete l'algoritmo abbassando i valori della soglia, si vede che riappare gradualmente la forma del piede ma, nello stesso tempo, si perdono i contorni delle ossa, con l'espandersi dell'area bianca. L'immagine (c) presenta l'output ottenuto settando `T=60`.

3.2.2 Metodi region-based

Segmentazione region-growing

All'interno della *DIPUM toolbox*, si trova la funzione `regiongrow` che esegue la procedura di segmentazione dell'immagine data. Tale funzione é definita come segue:

```
function [g, NR, SI, TI] = regiongrow(f, S, T)
```

```
f = double(f);
if numel(S) == 1
   SI = f == S;
   S1 = S;
else
   SI = bwmorph(S, 'shrink', Inf);
   J = find(SI);
   S1 = f(J);
end
TI = false(size(f));
for K = 1:length(S1)
   seedvalue = S1(K);
   S = abs(f - seedvalue) <= T;
   TI = TI | S;
end
[g, NR] = bwlabel(imreconstruct(SI, TI));
```

Se `f` é la solita immagine di input, i parametri `s` e `t` sono invece i valori che determinano rispettivamente l'intensitá dei semi e la massima differenza tollerabile tra il valore del pixel in esame e quello del seme. Come output, si ha l'immagine segmentata `g`, il numero `nr` di regioni individuate, l'immagine finale `si` coi semi e quella `ti` contenente i pixel che soddisfano la sogliatura. Ad esempio

```
f= imread( 'image.jpg');
s = 60;
t= 75;
[g, nr,si,ti] = regiongrow(f,s,t);
```

restituisce le immagini mostrate nella figura 3.8, dove lo sfondo viene inteso come elemento importante dell'immagine, dato che il numero di semi fissato era molto alto.

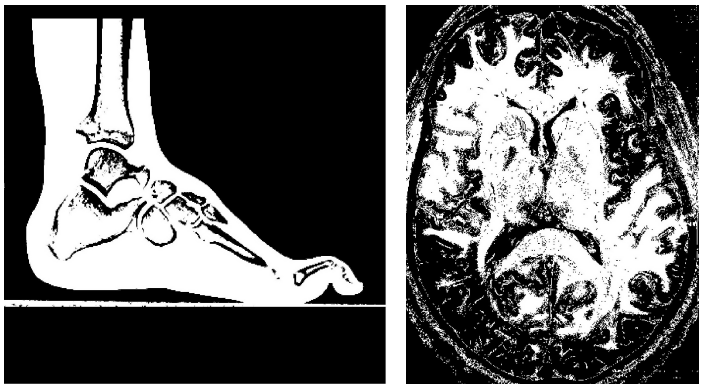
(a) (b)

Figura 3.8: Output della region growing con t=75

Se, invece, si setta

```
t = 35;
```

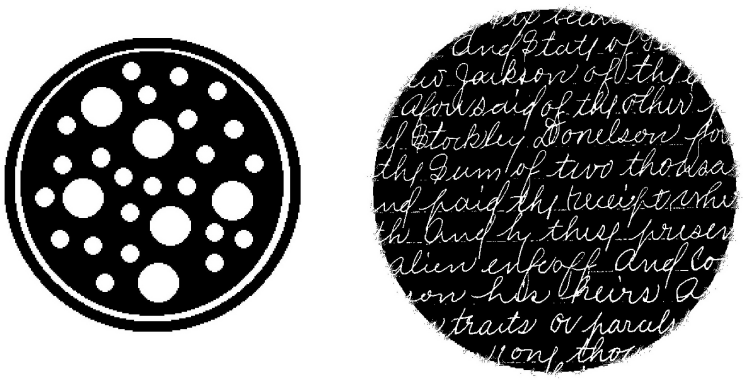
gli output sono molto piú validi



(a) (b)

Figura 3.9: Output della region growing con t=35

Nel piede ora si distinguono le ossa, grazie a una buona identificazione degli strati corticali esterni, e la forma del piede si impone sullo sfondo nero. Anche nell' esempio (b) l'immagine emerge dallo sfondo scuro, ma le anse celebrali ancora non sono molto nitide, soprattutto nella bassa parte sinistra.



(a) (b)

Figura 3.10: Output della region growing con t=75

Si considerino gli output della figura 3.10. L'immagine (a) non differisce dall'originale: essendo giá un'immagine binaria, l'individuazione dei semi non crea problemi. Un caso interessante é quello fornito dall'output (b): tralasciando il solito problema ai margini, si apprezza la leggibilitá del testo. Sullo sfondo nero, si distingue bene la calligrafia: il tratto é continuo, rappresentato con un opportuno spessore e non appaiono inutili macchie bianche.

Segmentazione splitting and merging

Sempre nell'archivio *DIPUM toolbox* si trova la funzione `splitmerge(f, mindim, fun)`, che provvede a segmentare un'immagine seguendo il metodo esplicato nel paragrafo 2.2.2. Eccone il testo:

```
function g = splitmerge(f, mindim, fun)


q = 2^nextpow2(max(size(f)));
[m n]=size(f);
f = padarray(f, [q-m, q-n], 'post');
s = qtdecomp(f, @split_test, mindim, fun);
lmax = full(max(s(:)));
g=zeros(size(f));
marker = zeros(size(f));
```

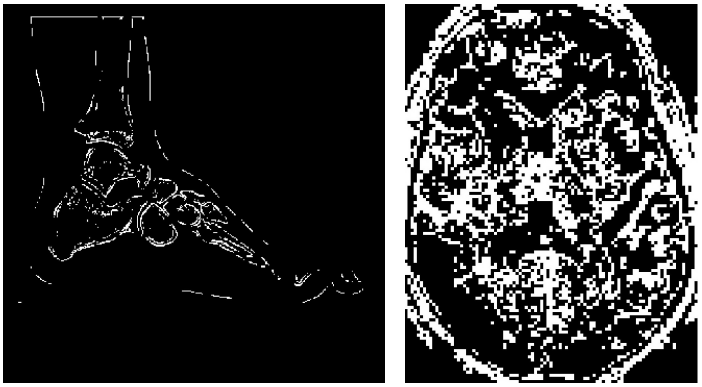
```
for k = 1:lmax
        [vals, r, c] = qtgetblk(f, s,k);
        if ~isempty(vals)
        for i= 1:length(r)
               xlow = r(i);
               ylow = c(i);
               xhigh = xlow + k -1;
               yhigh = ylow + k -1;
               region = f(xlow:xhigh, ylow:yhigh);
               flag = feval(fun, region);
               if flag
                   g(xlow:xhigh, ylow:yhigh) = 1;
                   marker(xlow, ylow) = 1;
               end
        end
        end
end
g = bwlabel(imreconstruct(marker,g));
g = g(1:m,1:n);
```

La variabile `mindim` deve essere una potenza del 2, perché corrisponde alla minima dimensione consentita per un quadrante. Invece `fun` é la funzione che corrisponde al predicato *P(R)* su cui si basa l'intera procedura. Si definisce tale funzione con

```
function flag = predicate (region)
sd = std2(region);
m = mean2(region);
flag = (sd>10) & (m>0) & (m<125);
```

in modo che restituisca `flag = TRUE` se le intensitá dei pixel contenuti in `region` hanno $\sigma > 10$ e la loro media abbia valore `m`$\in(0, 125)$.
La figura 3.11 riporta le immagini di output elaborate dal comando

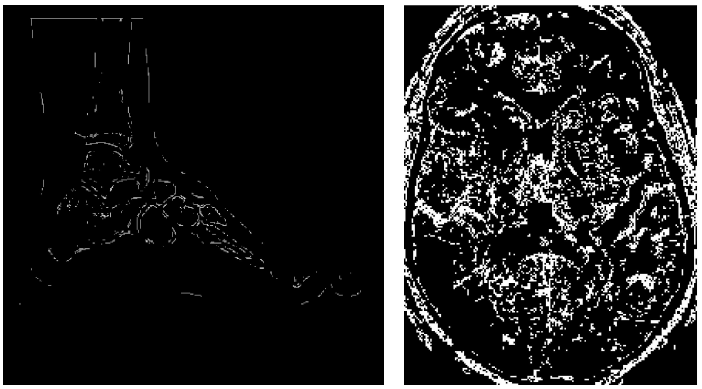
```
g=splitmerge (f,4,@predicate);
```



(a) (b)

Figura 3.11: Output della splitting and merging con mindim=4

Questi risultati non sono certo dei migliori. Nella TAC (a) si percepiscono le forme del piede e delle ossa, ma i contorni sono frastagliati e aperti. Anche nella seconda immagine il risultato finale é confuso, poiché l'output é fortemente "piastrellato" e le regioni non sono delimitate da linee morbide.
Questo problema si limita riducendo `mindim`; fissandolo semplicmente a `2`, si ottengono le immagini della figura 3.12.



(a) (b)

Figura 3.12: Output della splitting and merging con mindim=2

Se nell'immagine di sinistra la segmentazione si é ridotta al punto di essere quasi indiscernibile, nell'output di destra si apprezza qualche miglioramento.

3.2.3 Metodi con la trasformata Watershed

Segmentazione con la distance transform

Il seguente algoritmo esegue la segmentazione di un'immagine, seguendo il metodo descritto nel paragrafo 2.3.1.

```
f= imread( 'image.jpg');
g = im2bw(f,graythresh(f));
gc = ~g;
D = bwdist(gc);
L = watershed(-D);
w = L==0;
g2 = g&~w;
```

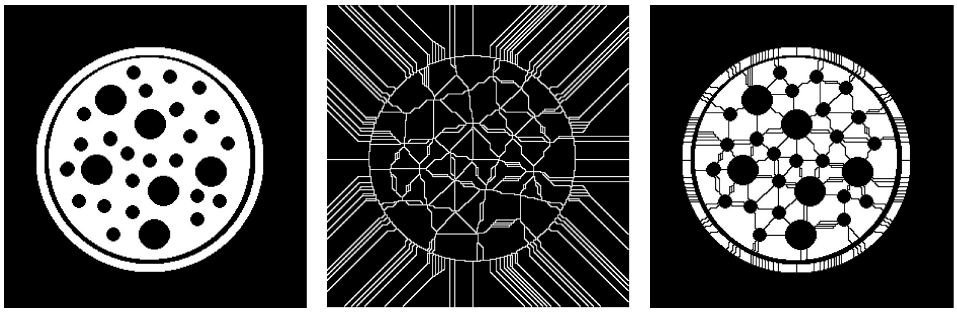
Acquisita l'immagine di input, la si converte in binaria tramite la sogliatura col valore `graythresh(f)`. Successivamente si applica la *distance transform* (con la funzione `D = bwdist(gc)`) all'immagine `gc`, che é la complementare di quella sogliata. Calcolata la *watershed transform* all'immagine `-D`, si determinano le linee "spartiacque" e le si sovrappone all'input, ottenendo i risultati riportati nella figura 3.13.

Come prevedibile, la funzione `watershed` ha prodotto troppe linee di cresta.

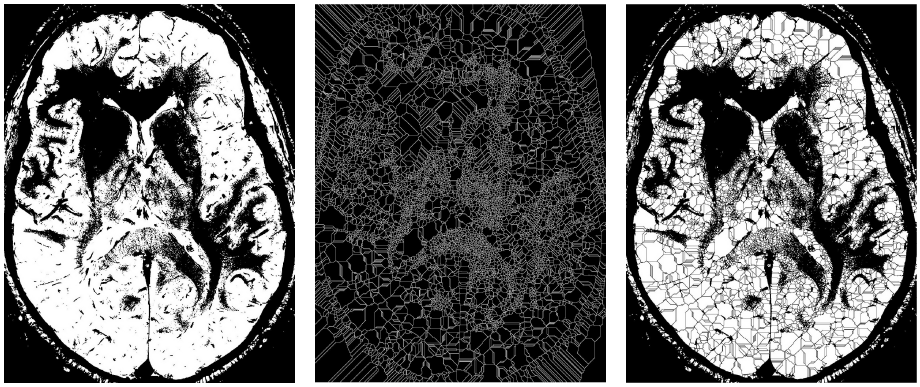
Segmentazione con il gradiente

Per ridurre l'effetto dell'*oversegmentation* si puó provare ad applicare la trasformazione *watershed* all'immagine del gradiente. Il seguente algoritmo determina il gradiente con le approssimazioni di Sobel e, infine, calcola la segmentazione.

```
f= imread( 'image.jpg');
h = fspecial('sobel');
```



(a) Distance transform (b) Watershed ridgelines (c) Output finale

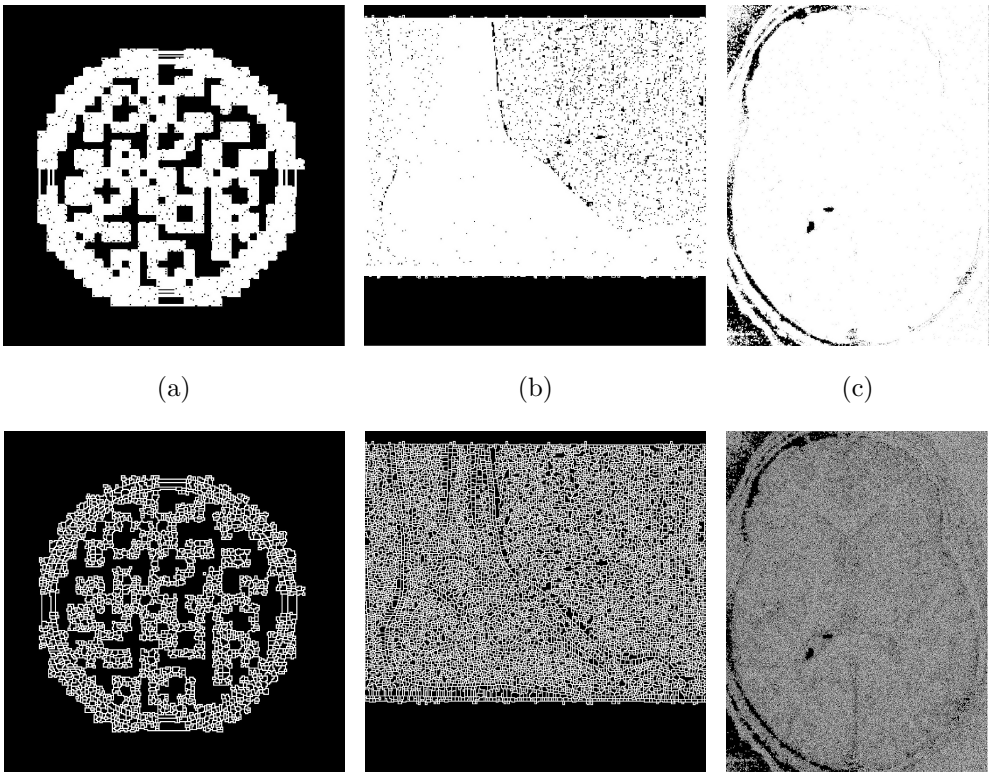


(d) Distance transform (e) Watershed ridgelines (f) Output finale

Figura 3.13: Output della segmentazione watershed con la distance transform

```
fd = double(f);
g = sqrt ( imfilter(fd,h,'replicate').^2 +
          imfilter(fd,h','replicate').^2 );
L = watershed(g);
wr = L==0;
```

La figura 3.14 riporta i risultati di questi passaggi.
Le immagini (a), (b) e (c) mostrano i gradienti, le altre presentano le *ridgelines* individuate. I risultati ottenuti sono totalmente insoddisfacenti: nel caso piú semplice, (d), le circonferenze piú piccole sono rappresenate da un groviglio di quadratini, nel secondo esempio si intravede solo la sagoma del piede e di poche ossa, mentre nell' output (f) non si distingono elementi all'interno.



(a) (b) (c)

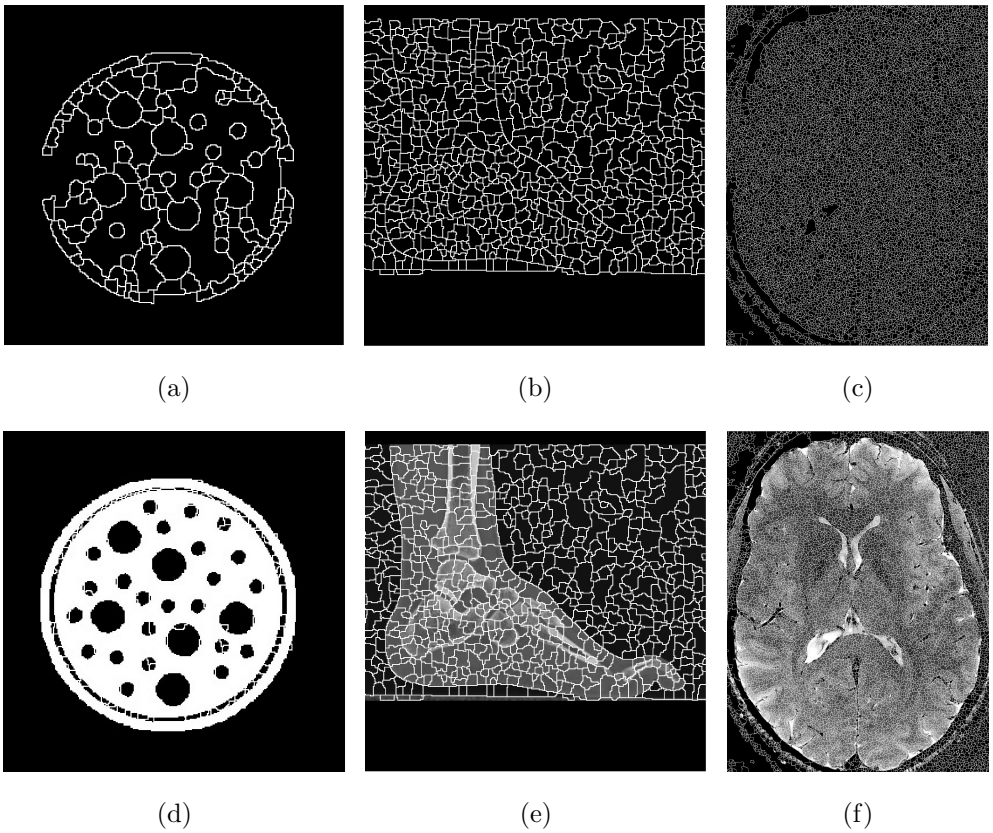
(d) (e) (f)

Figura 3.14: Immagini dei gradienti e output della segmentazione con le ridgelines

Come giá spiegato nel paragrafo 2.3.2, conviene smussare l'immagine del gradiente, aggiungendo all'algoritmo i seguenti passaggi.

```
g2 = imclose ( imopen(g,ones(3,3)),ones(3,3) );
L2 = watershed(g2);
wr2 = L2==0;
f2 = f;
f2(wr2) = 255;
```

In particolare, l'ultimo comando permette di visualizzare l'immagine originaria con le linee bianche sovrapposte. Ad ogni modo, gli output mostrati nella figura 3.15 sono migliori, ma non ancora ottimi.



(a) (b) (c)

(d) (e) (f)

Figura 3.15: Output della segmentazione e sovrapposizioni con l'input

Nelle immagini (a) e (d) si vede un netto miglioramento, mentre con gli input piú complicati la correzione continua a non fornire i risultati desiderati. Chiaramente, il problema dell'*oversegmentation* é proporzionale alla difficoltá di segmentazione dell'immagine.

Segmentazione con i marcatori

Questa procedura é stata sviluppata per ridurre l'*oversegmentation* dei risultati precedenti. Applicando la funzione

```
rm = imregionalmin (g);
```

dell'IPT all'immagine del gradiente precedentemente calcolata, si evidenziano tutti i punti di minimo locale dell'immagine. Se si considera il caso "intermedio" della TAC, per esempio, si vede bene che l'abbondante presenza di

questi punti produce tanti piccoli e fitti "bacini di raccolta", che evidenziano zone di irrelevante importanza.

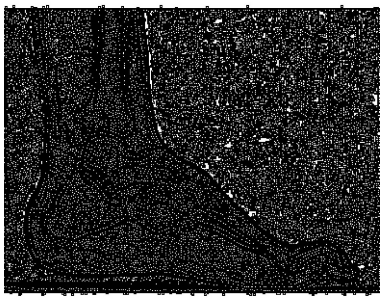


Figura 3.16: Regional minima dell'input

La funzione `imextendedmin (f, h)` dell'*IPT* elimina i minimi in eccesso, determinando gli insiemi di punti, il cui valore é inferiore a quello dei pixel circostanti di una certa quantitá `h`. Dunque la sintassi

```
im = imextendedmin(f,2);
```

determina i marcatori interni dell'immagine originale.
Successivamente si cercano i marcatori esterni, identificandoli come punti intermedi tra i minimi: per farlo, si calcola la *watershed transform* della trasformata delle distanze, applicata all'immagine coi marcatori interni.

```
Lim = watershed ( bwdist(im) );
em = Lim==0;
```

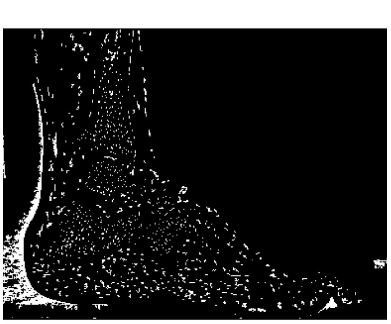
Ottenuti tutti i marcatori, li si utilizza per modificare il gradiente, con la procedura della *minima imposition*, implementata dal comando

```
g2 = imimposemin ( g, im | em );
```

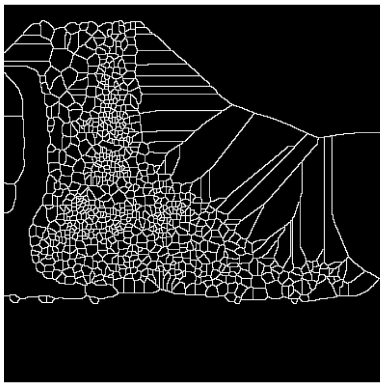
dove `im|em` é un'immagine binaria, i cui pixel di sfondo marcano le posizioni dei minimi locali dell'immagine di output. Ora si esegue la trasformata sul nuovo gradiente `g2` e si predispone l'output con l'immagine originale sovrastata dalle righe ottenute.

```
L2 = watershed(g2);
f2 = f;
f2( L2==0 ) = 255;
```

Eccone i risultati grafici:



(a) Marcatori interni



(b) Marcatori esterni

(c) Gradiente col minima imposition

(d) Risultato finale

Figura 3.17: Output della segmentazione e sovrapposizione con l'input

La segmentazione con la trasformata *watershed*, cosí articolata, fornisce risultati migliori. Dallo sfondo spariscono quasi tutte le *ridgelines*, che non

corrispondevano ad alcun elemento nell'immagine, mentre restano numerose quelle nell'area interna al piede. In veritá, qui non determinano con esattezza la forma delle ossa, né tantomeno si concentrano in prossimitá dei contorni.

Conclusioni

In questa dissertazione si sono analizzate diverse tecniche di estrazione dei contorni e di segmentazione delle immagini digitali a toni di grigio.

Il filtraggio con una maschera rileva molto bene i bordi degli oggetti e non é particolarmente sensibile al rumore delle immagini, peró lascia molto spesso contorni aperti ed incompleti, cosí come non si presta particolarmente per immagini piene di linee sottili. Al contrario, operatori come quello di Sobel, di Canny e l'LoG sono maggiormente influenzabili dal rumore e, per questo, hanno piú frequentemente problemi di *oversegmentation*. D'altra parte, restituiscono contorni molto precisi, chiusi e nitidi.

Per quanto riguarda le tecniche di segmentazione, il metodo della sogliatura é quello computazionalmente piú semplice e veloce, che permette di ottenere risultati apprezzabili. Eventualmente settando un confacente valore di soglia, si possono ricavare output decorosi anche da immagini "difficili", con molte sfumature di grigio.
La segmentazione *region-based* é generalmente poco sensibile al rumore dell'input. Inoltre, é sufficiente settare accuratamente i due parametri che caratterizzano l'algoritmo della *region growing*, per ottenere output significativi dove le linee siano ben definite e le sfumature di grigio ben distinte. Meno comprensibili sono i risultati del metodo *splitting and merging*, poiché abbassando il parametro di soglia si ottiene un'immagine piú nitida ma meno contrastata e troppo scura.

Infine, la segmentazione con la trasformata *watershed* ha evidenziato il previsto problema dell'*oversegmentation*, che é stato in parte ridotto, passo a passo, con i metodi successivamente elaborati.

Comunque, il problema della segmentazione non ha una soluzione che sia in assoluto preferibile alle altre, perché dipende dalle caratteristiche di ogni singolo input. Infatti, se le immagini binarie sono facili da elaborare, sia quelle ricche di sfumature (come la risonanza magnetica del cervello) che quelle costiuite da tratti e linee sotitli (come quella del testo) comportano maggiori difficoltá, anche di diversa natura.

Bibliografia

[1] Gonzalez, Woods and Eddins, *Digital Image Processing Using MATLAB*, Prentice Hall, 1st edition, 2004.

[2] http://www.imageprocessingplace.com/

[3] The MathWorks, *Image Processing Toolbox, User's guide, Version 4.*

[4] John F. Canny, "A Computational Approch for Edge Detection", in *IEEE Transactions Pattern Analysis and Machine Intelligence*, IEEE Computer Society, pp. 679-698, 1986.

[5] Nobuyuki Otsu, "A Threshold Selection Method from Gay-Lvel Histgrams", in *IEEE Transactions on Systems, Man, and Cybernetics*, IEEE Computer Society, pp.62-66, 1979.

[6] Harry L. Van Trees, *Detection, Estimation and Modulation Theory, part I*, Wiley-Interscience Pubblication, 1968.

[7] Luc Vincent and Pierre Soille, "Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations", in *IEEE Transactions Pattern Analysis and Machine Intelligence*, IEEE Computer Society, pp. 583-598, 1991.