Università degli Studi di Brescia

Corso di

ROBOTICA

Prof. Riccardo Cassinis

Elaborato:

"Implementazione dell'algoritmo MSCD sull'edge-detection"

Studente:

ANDREOLI TIZIANO

mat. 026894

1. SOMMARIO

2. INTRODUZIONE

Nel campo dell'edge-detection pochi algoritmi sono in grado di elaborare immagini multi spettrali (ossia a colori). Uno dei primi algoritmi in grado di fare questo è l'MSCDA (Multi Spectral Contour Detection Algorithm) o *Algoritmo Multi Spettrale per il Riconoscimento di Contorni*.
L'elaborato consiste nell'implementazione dell'MSCDA e di un confronto con algoritmi classici come SOBEL, PREWITT e LoG (Laplacian of Gaussian).
Materiale utile è stato preso:

- dall'articolo *"Combined detection of intensity and chromatic contours in color images"* (Society of Photo-Optical Instrumentation Engineers, 1996) scritto da Andrea Baraldi e Flavio Parmiggiani;
- dal libro di testo *"Machine Vision"* per quanto riguarda gli algoritmi classici.

Nei prossimi paragrafi sarà data una descrizione della struttura dell'MSCD, per poi passare a trattare dell'implemetazione nel linguaggio C++.

3. STRUTTURA DELL'MSCDA

L'MSCDA usa una matrice bidimensionale di moduli di processo (Processing Modules). Ogni modulo consiste in un banco di filtri orientati che sono chiamati MSPE (Multi Spectral Processing Element, *Elementi base di Computazione Multi Spettrale*). Ogni MSPE calcola un valore di contrasto massimo da un campo recettivo caratterizzato da un orientamento, una forma ed una dimensione.
Il valore di contrasto è una combinazione di un contrasto cromatico e di un contrasto acromatico, calcolati separatamente. Bisogna chiarire che una variazione del contrasto acromatico riguarda un cambiamento nell'intensità di due pixel adiacenti dell'immagine, mentre una variazione cromatica del contrasto riguarda ad un cambiamento locale nelle componenti cromatiche dei pixel.
Ci sono due tipi di MSPE: uno a simmetria pari ed uno a simmetria dispari, legati l'uno all'altro dalla trasformata di Hilbert. Una coppia di MSPE (pari + dispari) coopera per estrarre da un punto dell'immagine un valore combinato di contrasto. L'immagine è convoluta separatamente con le risposte dei due filtri, ed i risultati sono combinati in una somma pitagorica non lineare. Dato che ogni MSPE estrae un valore massimo di contrasto, il risultato che si ottiene elaborando l'immagine originale con questa matrice di filtri è un'*"immagine di contrasto"*, che viene poi elaborata ulteriormente per estrarre i pixel candidati ad essere di contorno.
La dimensione dell'operatore MSCDA è di 3×3 o 5×5 pixel. Le dimensioni ridotte consentono di ottenere un rapporto segnale rumore elevato.

3.1. CONTRASTO

Il valore di contrasto (o più semplicemente contrasto) è definito come:

$$C(\mathbf{X},\mathbf{Y}) = 1 - VDM(\mathbf{X},\mathbf{Y})$$

Eq. 1

dove: **X** e **Y** sono due vettori multi spettrali ricavati dal campo recettivo dell'MSPE;
VDM è il grado di somiglianza vettoriale fra **X** e **Y** (vedi Cap. 8 - La metrica nvd).

4. FUNZIONAMENTO DEI FILTRI

4.1. FILTRO DISPARI (ODD-MSPE)

Data la maschera Mask1 di dimensione 3×3 o 5×5 pixel si considerano due campi recettivi indicati rispettivamente con Area1 e Area2 (vedi Tab. 1).
Questi due campi forniscono due valori medi indicati con $\mathbf{X_1}$ e $\mathbf{X_2}$. Notiamo che i pixel appartenenti alla fila centrale non sono interessati dalla computazione.

Mask1	–2	–1	0	1	2
2					
1	***A***	***R***	***E***	***A***	***1***
0					
–1	***A***	***R***	***E***	***A***	***2***
–2					

Tab. 1. *Mask1 e i campi recettivi Area1 ed Area2.*

Il valore di contrasto è indicato con C_{12} ed è definito, secondo l'Eq. 1 come:

$$C_{12} = 1 - VDM(\mathbf{X_1}, \mathbf{X_2})$$

Eq. 2

4.2. FILTRO PARI (EVEN-MSPE)

Data la maschera Mask2 di dimensione 3×3 o 5×5 pixel si considerano due campi recettivi ulteriori, indicati rispettivamente con Area3 e Area4 (vedi Tab. 2). Area4 non è segnato in quanto in realtà è la combinazione di Area1 ed Area2.
Questi due campi forniscono due valori medi indicati con $\mathbf{X_3}$ e $\mathbf{X_4}$. Notiamo che i pixel appartenenti alla fila centrale questa volta sono interessati dalla computazione.

Mask2	–2	–1	0	1	2
2					
1	***A***	***R***	***E***	***A***	***1***
0	***A***	***R***	***E***	***A***	***3***
–1	***A***	***R***	***E***	***A***	***2***
–2					

Tab. 2. *Mask2 e i campi recettivi Area3 ed Area4.*

Poiché i pixel della fila centrale sono interessati dalla computazione, si è deciso di dare un segno al valore del contrasto misurato.
Tale è indicato con C_{34} ed è definito come:

$$C_{34} = S_{34} \cdot NC_{34} ,$$

Eq. 3

dove

$$S_{34} = \begin{cases} +1 \text{ se } \mathbf{X_3} \geq \mathbf{X_4} \\ -1 \text{ altrimenti} \end{cases}$$
$$NC_{34} = 1 - VDM(\mathbf{X_3}, \mathbf{X_4})$$

Eq. 4

4.3. COMBINAZIONE DEI RISULTATI FORNITI DALLA COPPIA DI FILTRI

I risultati dei due caratterizzati da Mask1 e Mask2 vengono combinati nel filtro caratterizzato da una maschera Mask1+2.
La Mask1+2 produce un contrasto indicato con C_{1234} definito come:

$$C_{1234} = S_{1234} \cdot NC_{1234}$$

Eq. 5

dove

$$S_{1234} = S_{34}$$
$$NC_{1234} = 1 - VDM_{1234} .$$

Eq. 6

La Mask1+2:

- utilizza i quattro campi recettivi definiti nei due paragrafi precedenti;
- fornisce, dopo la convoluzione, quattro valori medi: $\mathbf{X_1}$, $\mathbf{X_2}$, $\mathbf{X_3}$ e $\mathbf{X_4}$.

Questi vengono combinati insieme mediane i coefficienti C_{12} e C_{34}. Infatti:

$$VDM_{1234} = 1 - \frac{MDM_{1234} \cdot ADM_{1234}}{2}$$
$$MDM_{1234} = \sqrt{MDM_{12}{}^2 + MDM_{34}{}^2}$$
$$ADM_{1234} = \sqrt{ADM_{12}{}^2 + ADM_{34}{}^2}$$

Eq. 7

Dato che MDM ed ADM variano tra 0 ed 1, abbiamo che MDM_{1234} e ADM_{1234} variano tra 0 e √2. Quindi VDM_{1234} varia fra 0 ed 1 e C_{1234} fra –1 ed 1. Infine C_{1234} ha lo stesso segno di C_{12}.

4.4. CAMPI RECETTIVI

Tre tipi di campi recettivi sono stati implementati:

1. Filtri *Rettangolari* (vedi Tab. 3 e Tab. 4): garantiscono un'alta larghezza di banda e il miglior riconoscimento nel caso di bordi a gradino; la loro dimensione è di 3x3 pixel;
2. Filtri di *Gabor* (vedi Tab. 5 e Tab. 6); la dimensione è di 5x5 pixel;
3. Filtri di *Gabor Scalati* (vedi Tab. 7 e Tab. 8); la dimensione è di 5x5 pixel.

Le maschere sono mostrate nelle tabelle seguenti:

Mask1	–2	–1	0	1	2
2					
1		–0.33	–0.33	–0.33	
0		0	0	0	
–1		0.33	0.33	0.33	
–2					

Tab. 3. *Mask1 rettangolare (MSPE dispari).*

Mask2	–2	–1	0	1	2
2					
1		–0.17	–0.17	–0.17	
0		0.33	0.33	0.33	
–1		–0.17	–0.17	–0.17	
–2					

Tab. 4. *Mask2 rettangolare (MSPE pari).*

Mask1	–2	–1	0	1	2
2	0	0	0.0001	0	0
1	0	–0.0043	–0.0317	–0.0043	0
0	0	0	0	0	0
–1	0	0.0043	0.0317	0.0043	0
–2	0	0	–0.0001	0	0

Tab. 5. *Mask1 di Gabor (MSPE dispari).*

Mask2	–2	–1	0	1	2
2	0	0	0.0001	0	0
1	0	–0.0108	–0.0801	–0.0108	0
0	0.0002	0.0861	0.6366	0.0861	0.0002
–1	0	–0.0108	–0.0801	–0.0108	0
–2	0	0	0.0001	0	0

Tab. 6. *Mask2 di Gabor (MSPE pari).*

Mask1	–2	–1	0	1	2
2	0	0	0.0025	0	0
1	0	–0.1067	–0.7866	–0.1067	0
0	0	0	0	0	0
–1	0	0.1067	0.7866	0.1067	0
–2	0	0	0.0025	0	0

Tab. 7. *Mask1 di Gabor Scalata (MSPE dispari).*

Mask2	–2	–1	0	1	2
2	0	0	0.0001	0	0
1	0	–0.0531	–0.3938	–0.0531	0
0	0.0002	0.1064	0.7865	0.1064	0.0002
–1	0	–0.0531	–0.3938	–0.0531	0
–2	0	0	0.0001	0	0

Tab. 8. *Mask2 di Gabor Scalata (MSPE pari).*

5. IMPLEMENTAZIONE

Per implementare l'algoritmo MSCD è stato utilizzato il *linguaggio C++* e come compilatore la versione 3.0 del *Borland C++ Builder*. In questo modo si sono evitati carichi di programmazione dal punto di vista dell'interfaccia.

5.1. LE IMMAGINI

Per memorizzare le immagini è stato utilizzato l'oggetto Timage, già definito nella libreria VCL del C++.
In tutto sono state utilizzate 6 immagini:

- l'immagine d'ingresso;
- i suoi 3 piani cromatici fondamentali R, G, B;
- il piano monocromatico;
- l'immagine "risultato" dell'elaborazione.

All'interno della classe Timage un'immagine è memorizzata come array bidimensionale (di dimensioni Larghezza × Altezza) di punti (pixel): ad ogni punto dell'immagine corrisponde il proprio colore (valore intero, più precisamente un tipo di dati enumerativo).

5.2. IL COLORE

Il colore di un pixel è memorizzato, come detto al paragrafo precedente, come valore intero (long int a 32 bit). Per essere più precisi, il colore è definito come tipo enumerativo.
La struttura della variabile colore è la seguente:

- Il byte più significativo (il byte 3) è "speciale" in quanto indica in che modo il programma deve ottimizzare i colori dell'immagine a seconda della palette scelta;
- Il byte successivo (byte 2) contiene la componente cromatica di blu del colore;
- Il byte successivo (byte 1) contiene la componente cromatica di verde;
- L'ultimo byte (byte 0) contiene la componente cromatica di rosso.

La struttura è mostrata anche nella Tab. 9, dove ogni byte è stato diviso in due "pezzi", o chunk, di 4 bit. In questo modo ogni chunk può essere espresso da una cifra esadecimale, rendendo quindi più facile l'espressione del colore. Infatti, per esempio:

`00|00|00|00` rappresenta il nero;
`00|00|00|FF` rappresenta il rosso puro;
`00|00|FF|00` rappresenta il verde puro;
`00|FF|00|00` rappresenta il blu puro;
`00|FF|FF|FF` rappresenta il bianco.

Byte 3		**Byte 2**		**Byte 1**		**Byte 0**	
Chunk 7	**Chunk 6**	**Chunk 5**	**Chunk 4**	**Chunk 3**	**Chunk 2**	**Chunk 1**	**Chunk 0**
Speciale		*Blue*		*Green*		*Red*	

Tab. 9. *Struttura del colore nell'array Pixels di Timage.*

Quindi per calcolare, data un'immagine "originale", i suoi quattro piani principali (ossia i tre cromatici più il quarto acromatico) si è usato il seguente algoritmo:

- per estrarre la componente rossa del punto si effettua l'AND logico bit a bit fra il colore ed il valore `000000FF`;
- per estrarre la componente verde del punto si effettua l'AND logico bit a bit fra il colore ed il valore `0000FF00`;
- per estrarre la componente blu del punto si effettua l'AND logico bit a bit fra il colore ed il valore `00FF0000`;
- per estrarre la componente acromatica (il livello di grigio) si calcola la media delle tre componenti cromatiche (tenendo conto del diverso peso dei valori, data la loro posizione diversa all'interno della variabile).

6. TEST DEGLI ALGORITMI CLASSICI

Il confronto è stato fatto con 3 algoritmi classici: SOBEL, PREWITT e LoG (vedi Cap. 9 - algoritmi classici).

6.1. PROVA 1

La prima immagine è formata da 5 linee non verticali e non orizzontali; tre sono colorate con i tre colori fondamentali, una è nera ed un'altra di un colore misto. I risultati sono mostrati nelle figure seguenti (threshold=100):

Fig. 1. *Immagine Originale.*

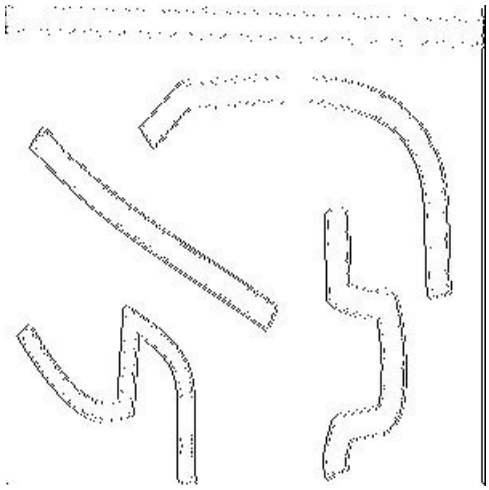


Fig. 2. *App. Filtro LoG.*

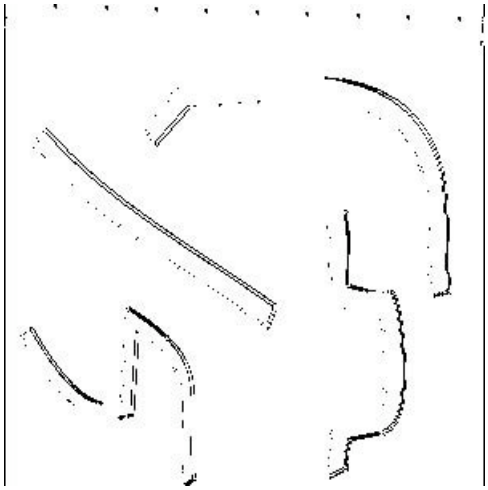


Fig. 3. *App. Filtro Sobel (Y).*

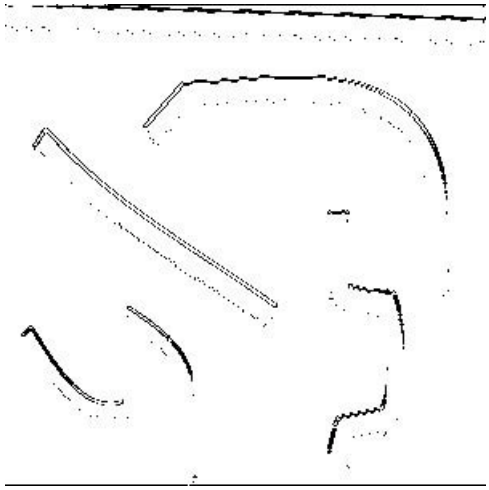


Fig. 4. *App. Filtro Sobel (X).*

(*continua…*)

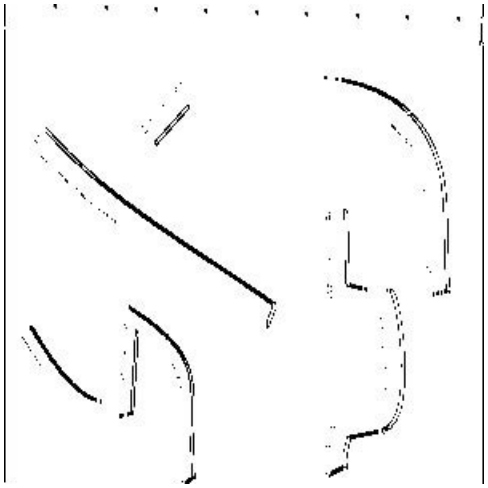


Fig. 5. *App. Filtro Prewitt (Y).*

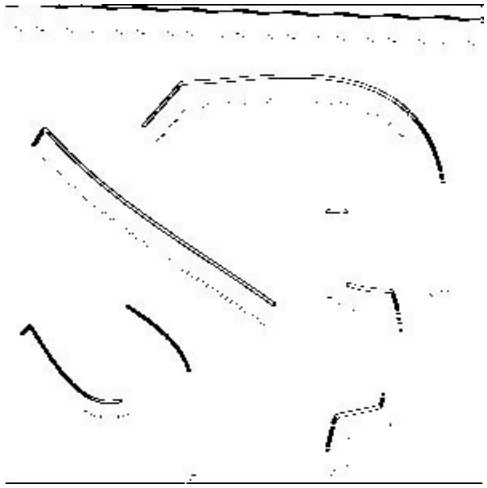


Fig. 6. *App. Filtro Prewitt (X).*

Tab. 10

Si nota che l'algoritmo LoG funziona bene, mentre gli altri no: Sobel e Prewitt non producono risultati "continui". Come si può notare dalle figure, alcune parti dei contorni non vengono rilevate, mentre l'LoG funziona molto bene: non ci sono interruzioni nei contorni rilevati (anche si sono presenti numerose sbavature). Inoltre si vede subito la differenza fra il filtraggio "verticale" ed il filtraggio "orizzontale": l'uno enfatizza una direzione di calcolo del gradiente a discapito dell'altra.

6.2. PROVA 2

La successiva immagine provata è simile alla precedente, ma presenta delle sfumature di colore: anche qui le linee sono cinque: una bianca, una rossa, una blu, una verde, una grigia. I risultati (tralasciando le immagini intermedie, threshold=100) sono:

Fig. 7. *Immagine Originale.*

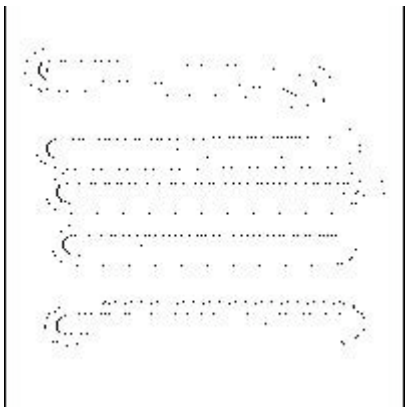


Fig. 8. *App. Filtro LoG.*

(*continua…*)

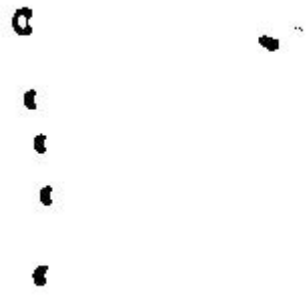


Fig. 9. *App. Filtro Sobel (Y).*

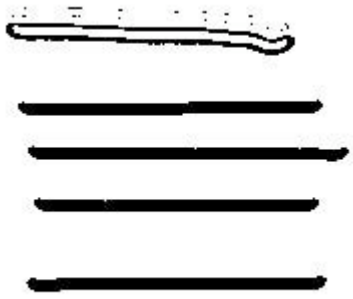


Fig. 10. *App. Filtro Sobel (X).*

Fig. 11. *App. Filtro Prewitt (Y).*

Fig. 12. *App. Filtro Prewitt (X).*

Tab. 11

Qui si notano alcune cose: innanzi tutto gli algoritmi funzionano non in modo efficiente, in quanto non riconoscono bene le linee. L'unico che raggiunge risultati più soddisfacenti è l'LoG, che riconosce le linee (anche se l'effetto sfumato ai bordi crea un contorno attorno ad esse. In ogni caso il risultato non è buono, soprattutto nel caso dei filtraggi verticali, dove Sobel e Prewitt non riconoscono praticamente nulla.

6.3. PROVA 3

La terza prova è stata fatta con un'immagine più complessa ma con poche variazioni di colore: i risultati sono stati (threshold=100):

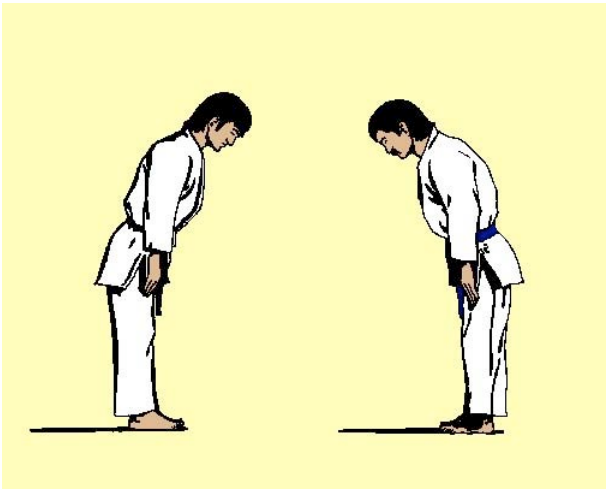


Fig. 13. *Immagine Originale.*

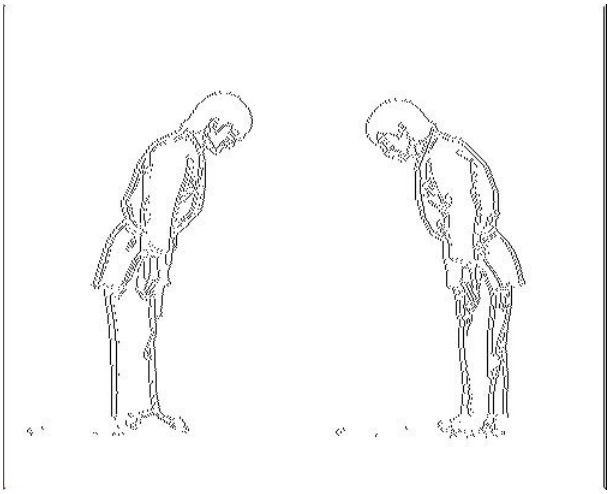


Fig. 14. *App. Filtro LoG.*

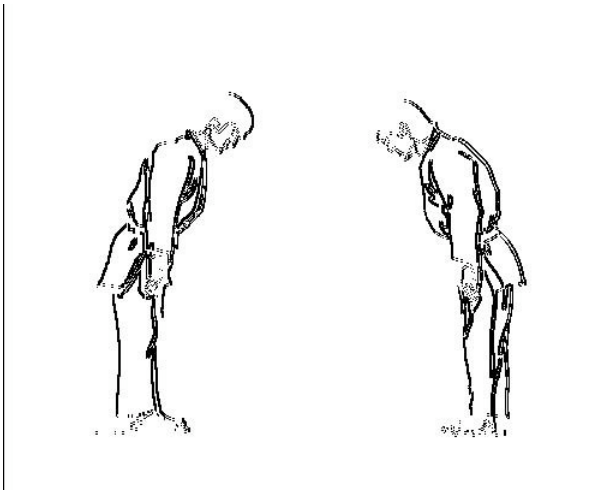


Fig. 15. *App. Filtro Sobel (Y).*

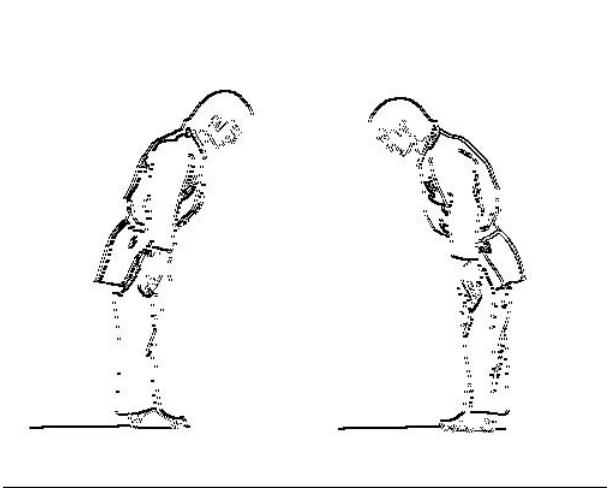


Fig. 16. *App. Filtro Sobel (X).*

(*continua…*)

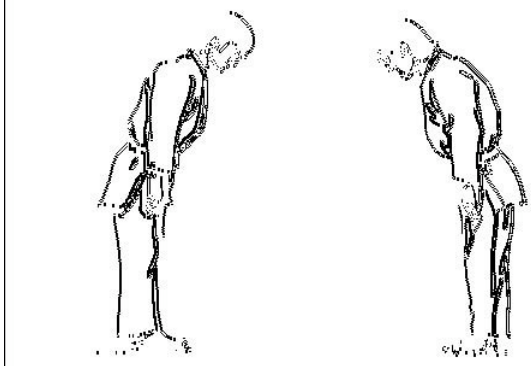


Fig. 17. *App. Filtro Prewitt (Y).*

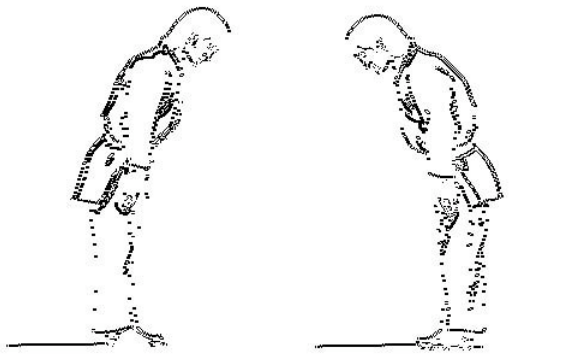


Fig. 18. *App. Filtro Prewitt (X).*

Tab. 12

Poiché quello che conta sono le variazioni d'intensità, le considerazioni sono le stesse fatte al §6.1.

6.4. PROVA 4

La seguente prova è stata effettuata sull'immagine più complessa, cioè quella di un paesaggio. I risultati sono stati (threshold=100):

Fig. 19. *Immagine Originale.*

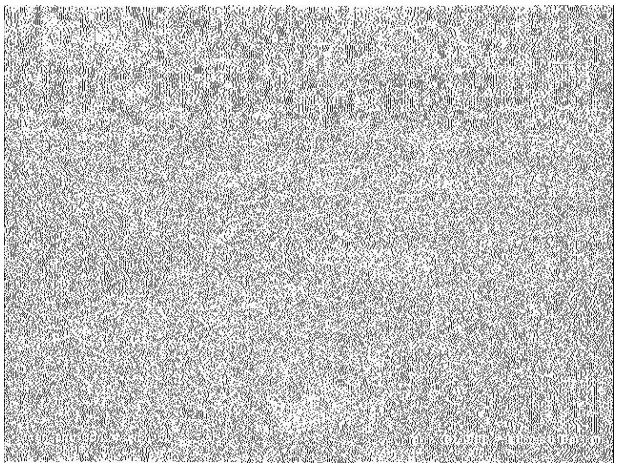


Fig. 20. *App. Filtro LoG.*

(*continua…*)

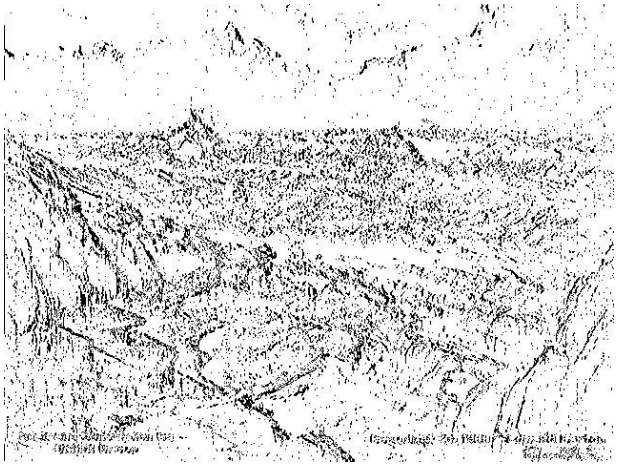


Fig. 21. *App. Filtro Sobel (Y).*

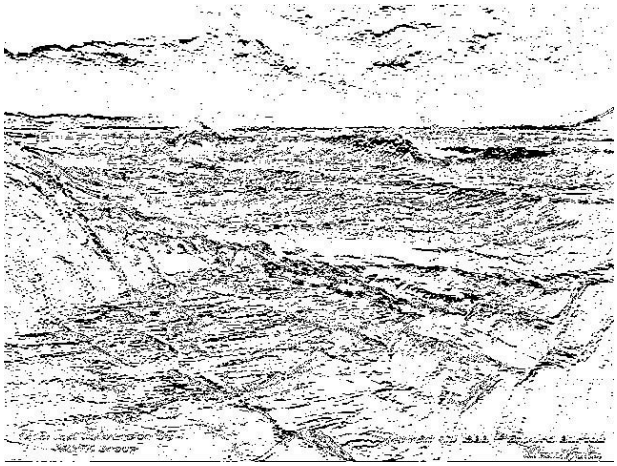


Fig. 22. *App. Filtro Sobel (Y).*

Fig. 23. *App. Filtro Prewitt (Y).*

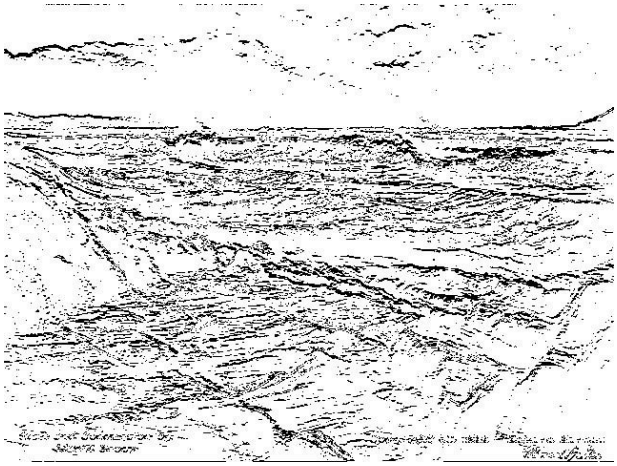


Fig. 24. *App. Filtro Prewitt (X).*

Tab. 13

Si nota come in questo caso gli algoritmi di Sobel e Prewitt danno risultati più efficienti dell'LoG.

7. TEST DELL'MSCDA

7.1. PROVA 1

L'MSCDA, come già detto, è stato implementato usando tre tipi di maschere: i risultati ottenuti sulle prove, pertanto, confronteranno le prestazioni sia fra le maschere, sia fra i vari algoritmi.
I risultati sono:

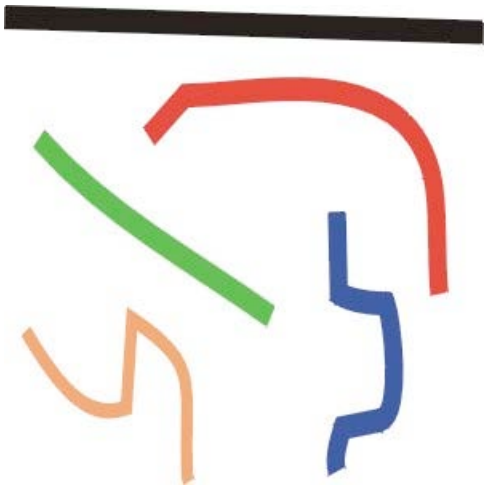


Fig. 25. *Immagine Originale.*

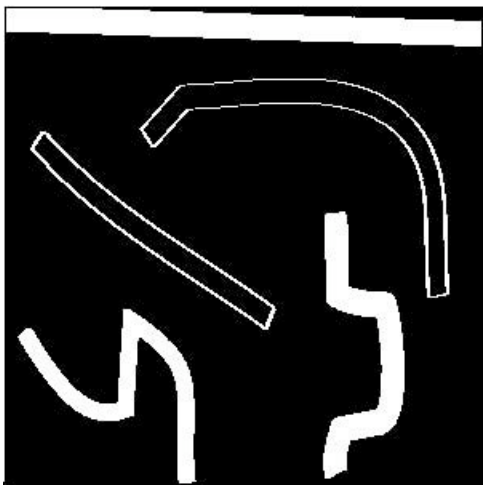


Fig. 26. *MSCD con filtro BOX (3×3).*

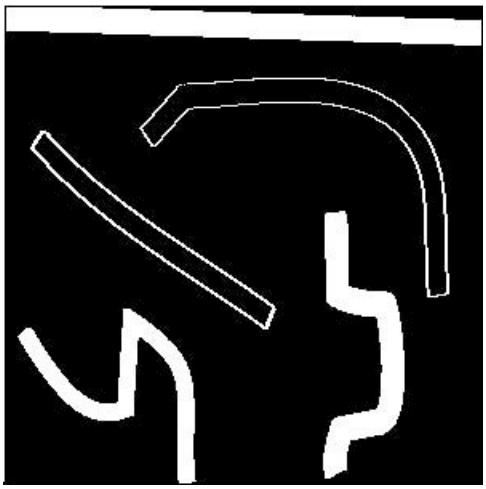


Fig. 27. *MSCD con filtro di GABOR (5×5).*

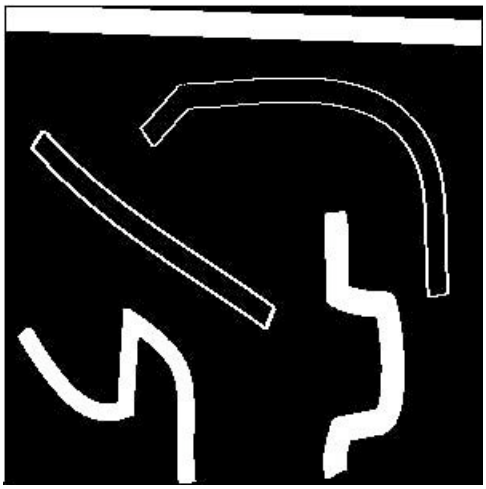


Fig. 28. *MSCD con filtro S-GABOR (5×5).*

Tab. 14

Si nota come i contorni sono rilevati in modo ottimo da tutti e tre i campi recettivi. In questo caso, quindi, il metodo MSCD si dimostra il migliore fra tutti.

7.2. PROVA 2

I risultati sono:

Fig. 29. *Immagine Originale.*

Fig. 30. *MSCD con filtro BOX (3×3).*

Fig. 31. *MSCD con filtro di GABOR (5×5).*

Fig. 32. *MSCD con filtro S-GABOR (5×5).*

Tab. 15

L'algoritmo MSCD è molto efficiente rispetto ai precedenti, in quanto riconosce perfettamente le linee, quasi "non curandosi" del bordo sfumato: per lui tutto fa parte della linea. Si nota come i bordi siano accentuati, cosicché alla fine i bordi delle linee risultano molto spessi.

7.3. PROVA 3

I risultati sono:

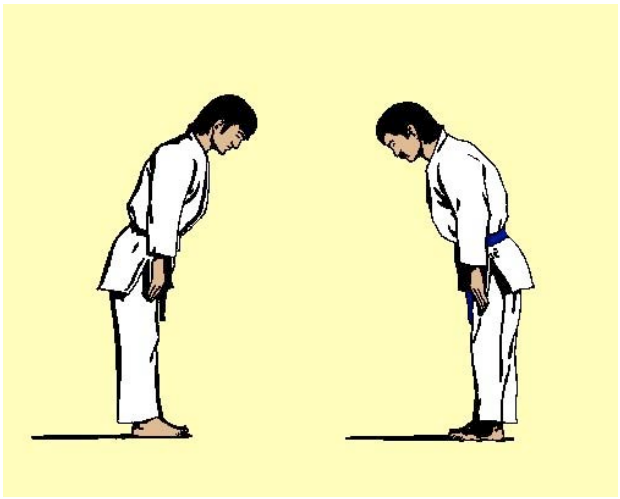


Fig. 33. *Immagine Originale.*

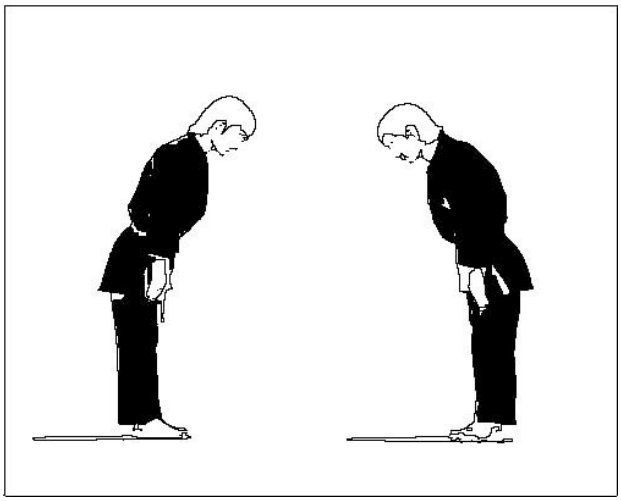


Fig. 34. *MSCD con filtro BOX (3×3).*

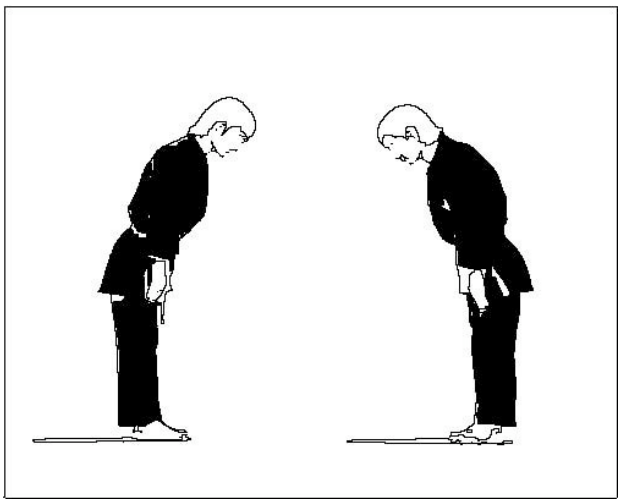


Fig. 35. *MSCD con filtro di GABOR (5×5).*

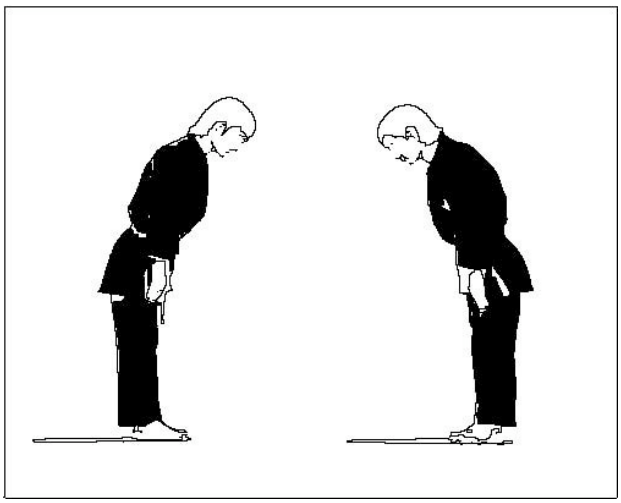


Fig. 36. *MSCD con filtro S-GABOR (5×5).*

Tab. 16

Le immagini finale sono le stesse in tutti e tre i casi esaminati. Si nota che i contorni sono rilevati in modo più efficiente che negli altri algoritmi.
Una nota riguarda, nel caso dell'immagine, il kimono: l'MSCD lo colora tutto di bianco, tranne alcune parti, ma vengono tolte tutte le pieghe interne.

7.4. PROVA 4

I risultati sono stati:

Fig. 37. *Immagine Originale.*

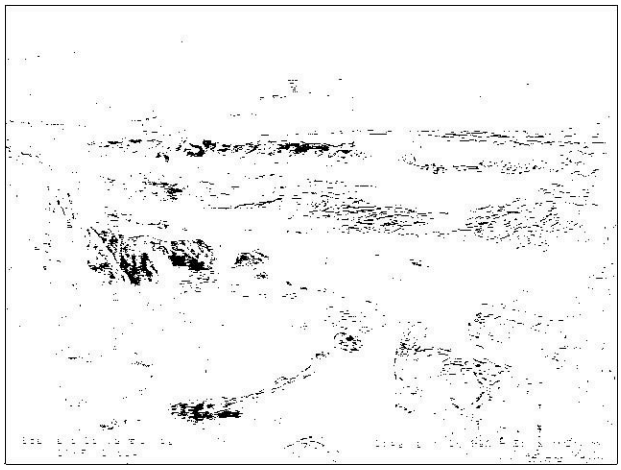


Fig. 38. *MSCD con filtro BOX (3×3).*

Fig. 39. *MSCD con filtro di GABOR (5×5).*

Fig. 40. *MSCD con filtro S-GABOR (5×5).*

Tab. 17

I risultati sono simili agli algoritmi di Sobel e Prewitt, anche se si osserva che l'MSCD toglie automaticamente tutti i passaggi graduali di colore, lasciando solo una zona uniforme.

Nel complesso, quindi, l'MSCD si comporta in modo soddisfacente in tutti i casi, riconoscendo i contorni con un'affidabilità più elevata che negli altri algoritmi.

8. LA METRICA NVD

8.1. NVD – DISTANZA NORMALIZZATA FRA VETTORI

La NVD (o *normalized vector distance*) costituisce una relazione biunivoca fra uno spazio bidimensionale S×S e l'intervallo [0,1]:

$$\text{NVD: } \boldsymbol{S} \times \boldsymbol{S} \leftrightarrow [0,1]$$

La matrice NVD fornisce una misura di distanza (contrasto) normalizzata fra due vettori. L'aspetto interessante di questa metrica è che si può applicare sia ad un vettore – nel caso specifico quindi è in grado di elaborare informazioni cromatiche – sia ad uno scalare, che rappresenta un caso particolare di vettore (quindi può gestire anche l'informazione acromatica).
Innanzi tutto per *vettore cromatico* si intende la terna (r,g,b) delle componenti fondamentali del colore di un punto di un'immagine. Per *vettore acromatico* invece si intende un vettore ad una sola componente, quindi un semplice scalare, che rappresenta l'intensità (o il livello di grigio) del punto di un'immagine.
Una volta definito cosa si intende per vettore, il valore NVD è dato da:

$$NVD(\mathbf{X},\mathbf{Y}) = 1 - VDM(\mathbf{X},\mathbf{Y}),$$

Eq. 8

dove VDM è il grado vettoriale di somiglianza fra **X** e **Y** (vedi prossimo paragrafo). Se NVD = 0 allora **X** e **Y** sono due vettori uguali, se NVD = 1 **X** e **Y** sono diversi al 100%. Si dimostra che l'NVD è commutativa, ossia NVD(**X**,**Y**) = NVD(**Y**,**X**).

8.2. VDM – GRADO DI SOMIGLIANZA FRA VETTORI

Dati due vettori **X** e **Y**, essi sono detti uguali se hanno:

1. lo stesso modulo;
2. la stessa direzione e lo stesso verso.

La VDM (*vector degree of match*) si propone di combinare queste due condizioni per fornire una misura normalizzata della somiglianza fra **X** e **Y**.
Innanzitutto si prendano **X** e **Y** . Si definisce grado di somiglianza dei moduli MDM (*module degree of match*):

$$MDM = \min\left(\frac{|\mathbf{X}|}{|\mathbf{Y}|}, \frac{|\mathbf{Y}|}{|\mathbf{X}|}\right),$$

Eq. 9

dove |X| è il modulo del vettore **X**, mentre |Y| è il modulo del vettore **Y**.

MDM varia tra 0 ed 1, e MDM = 0 se i due vettori hanno lo stesso modulo.

Ora si definisce il coefficiente γ:

$$\gamma = \cos\alpha = \frac{\mathbf{X} \circ \mathbf{Y}}{|\mathbf{X}| \cdot |\mathbf{Y}|},$$

Eq. 10

dove **T**∘**X** è il prodotto scalare fra **X** e **Y**. Si nota che γ varia da –1 a 1.

Dalla Eq. 10 ricaviamo che α=arcos(γ) dove α varia tra 0 e π.

Per ottenere un valore normalizzato fra 0 ed 1 viene definito il grado di somiglianza dell'angolo ADM (*angle degree of match*):

$$ADM = \frac{\pi - \alpha}{\pi}.$$

Eq. 11

A questo punto la definizione di VMD è:

$$VDM(\mathbf{X}, \mathbf{Y}) = MDM(\mathbf{X}, \mathbf{Y}) \cdot ADM(\mathbf{X}, \mathbf{Y})$$

Eq. 12

9. ALGORITMI CLASSICI

9.1. L'ALGORITMO DI SOBEL

Questo algoritmo si basa sul calcolo del gradiente attorno ad un punto in una particolare direzione, ossia un possibile contorno è presente dove si ha un cambiamento nell'intensità dell'immagine. L'algoritmo di Sobel consiste nella convoluzione di quest'ultima con una delle due maschere in Tab. 18 e Tab. 19. Si nota che Sobel pone l'attenzione sui pixel che sono più vicini al centro della maschera.

Sobel X	–1	0	1
1	–1	0	1
0	–2	0	2
–1	–1	0	1

Tab. 18. *Maschera di Sobel che enfatizza le variazioni lungo l'asse X.*

Sobel Y	–1	0	1
1	1	2	1
0	0	0	0
–1	–1	–2	–1

Tab. 19. *Maschera di Sobel che enfatizza le variazioni lungo l'asse Y.*

9.2. L'ALGORITMO DI PREWITT

L'algoritmo di Prewitt è molto simile a quello di Sobel, con la differenza che non pone l'attenzione sui pixel vicini al contro della maschera. Le maschere sono mostrate nelle Tab. 20 e Tab. 21.

Prew. X	–1	0	1
1	–1	0	1
0	–1	0	1
–1	–1	0	1

Tab. 20. *Maschera di Prewitt che enfatizza le variazioni lungo l'asse X.*

Prew. Y	–1	0	1
1	1	1	1
0	0	0	0
–1	–1	–1	–1

Tab. 21. *Maschera di Prewitt che enfatizza le variazioni lungo l'asse Y.*

9.3. L'ALGORITMO LoG

L'algoritmo Laplaciano del Gaussiano (*Laplacian Of Gaussian*) combina un filtraggio gaussiano con il laplaciano per il riconoscimento dei contorni: il filtraggio consente di ridurre il rumore presente nell'immagine, mentre il laplaciano consiste nel calcolo della derivata seconda.
La tecnica di riconoscimento dei contorni è la *zero-crossing*: è rilevato un contorno quando la derivata seconda attraversa lo zero.
Il filtraggio viene combinato con il calcolo del laplaciano in due modi:

- prima si convolve l'immagine con un filtro gaussiano, poi si calcola il laplaciano del risultato;
- oppure si convolve l'immagine con un filtro che rappresenta il laplaciano del filtro gaussiano.

Si è scelta la seconda tecnica e la maschera usata per la convoluzione è riportata nella Tab. 22 (la dimensione è di 5×5 pixel):

LoG	–2	–1	0	1	2
2	0	0	–1	0	0
1	0	–1	–2	–1	0
0	–1	–2	16	–2	–1
–1	0	–1	–2	–1	0
–2	0	0	–1	0	0

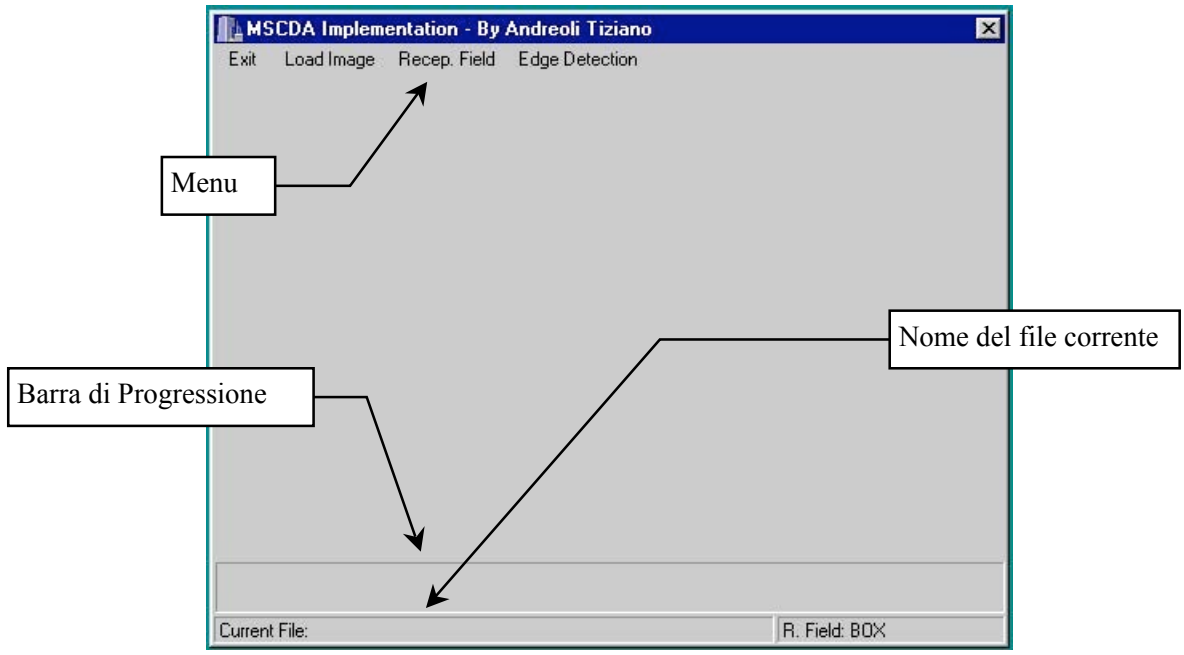
Tab. 22. *Maschera LoG.*

10. IL PROGRAMMA

Passiamo ora a descrivere brevemente il funzionamento del programma scritto per effettuare il test dell'algoritmo MSCD.

10.1. ESECUZIONE

Quando si esegue il programma, compare la finestra di dialogo principale:



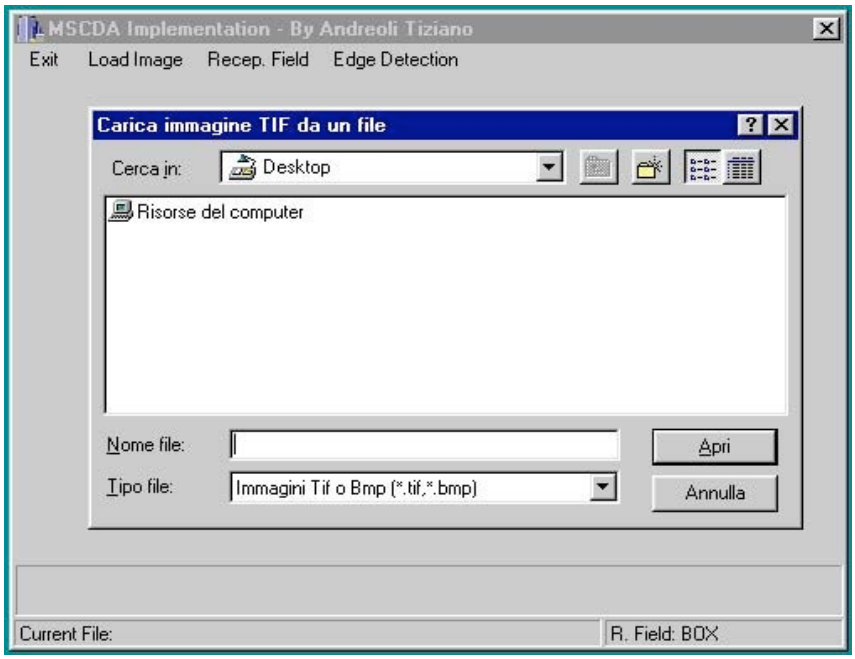
Sono presenti, all'inizio, quattro voci:

1. *Exit*: per terminare il programma;
2. *Load Image*: per caricare l'immagine da analizzare da file;
3. *Recep. Field*: permette di definire il tipo di campo recettivo;
4. *Edge Detection*: esegue il riconoscimento dei bordi.
5. *View Plan* (nascosta): vedi §10.4;
6. *Result* (nascosta)*:* vedi §10.6;

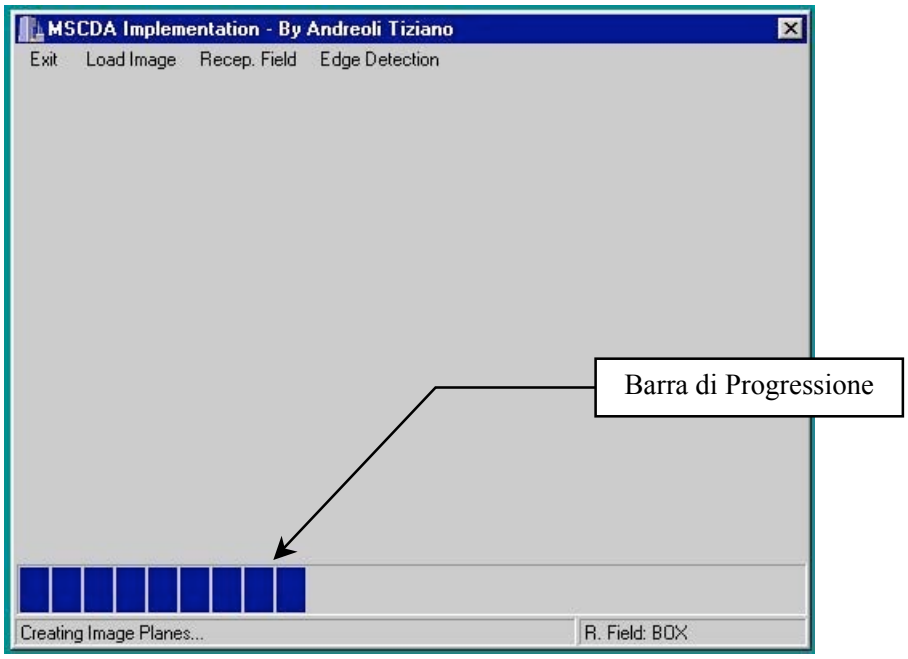
E' bene notare che l'esecuzione dell'*Edge Detection* non avrà effetto finchè un'immagine non sarà presente in memoria.
Passiamo ad analizzare le quattro voci in dettaglio: la prima, vista l'ovvietà della funzione svolta, verrà tralasciata.

10.2. VOCE "LOAD IMAGE"

Selezionando questa voce, comparirà la finestra necessaria a specificare il nome del file da caricare:



Una volta selezionato il file da caricare, basta cliccare sul pulsante *Apri* o premere *Enter* per proseguire. A questo punto il programma carica l'immagine e crea i quattro piani necessari alla sua elaborazione: mentre fa questo, una barra progressiva informa l'utente di quanta percentuale del processo totale sia stata completata, come appare dalla figura:



10.3. FORMATO DEI FILE

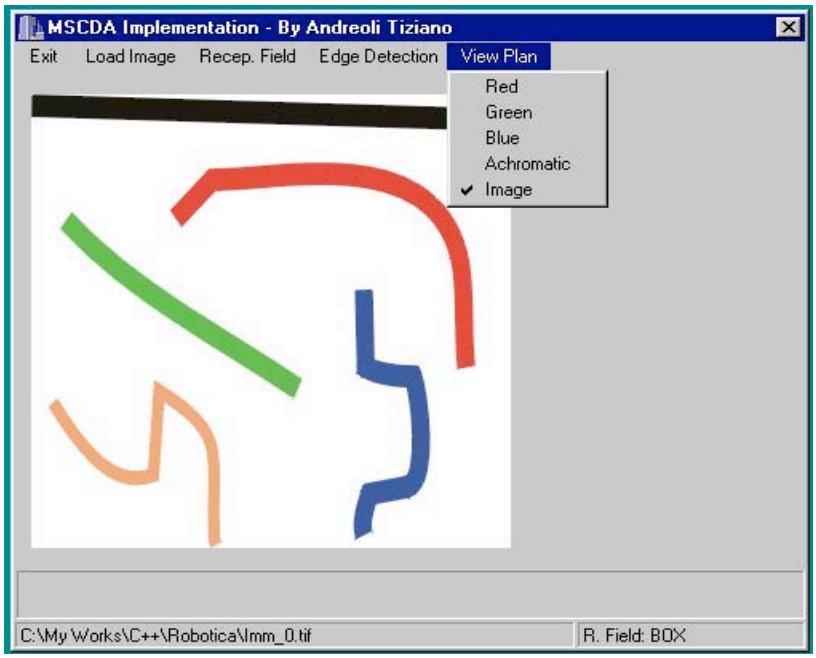
Il programma è stato sviluppato per supportare i seguenti formati di files:

- Windows Bitmap, BMP–Files (*true color* e *256 colori)*;
- TIFF–Files (*Ver. 6.0*, *Packbits*, *LZW*, *CCITT G.3* e *G.4*);

La libreria utilizzata per la lettura dei file TIFF, in realtà, supporta anche i Windows Graphics Metafile (WMF), i file PCX ed i file compressi JPEG. Volendo anche questa funzionalità basterà modificare leggermente il codice.

10.4. VOCE "VIEW PLAN"

Una volta caricata un'immagine, compare una voce di menu chiamata *View Plan*:



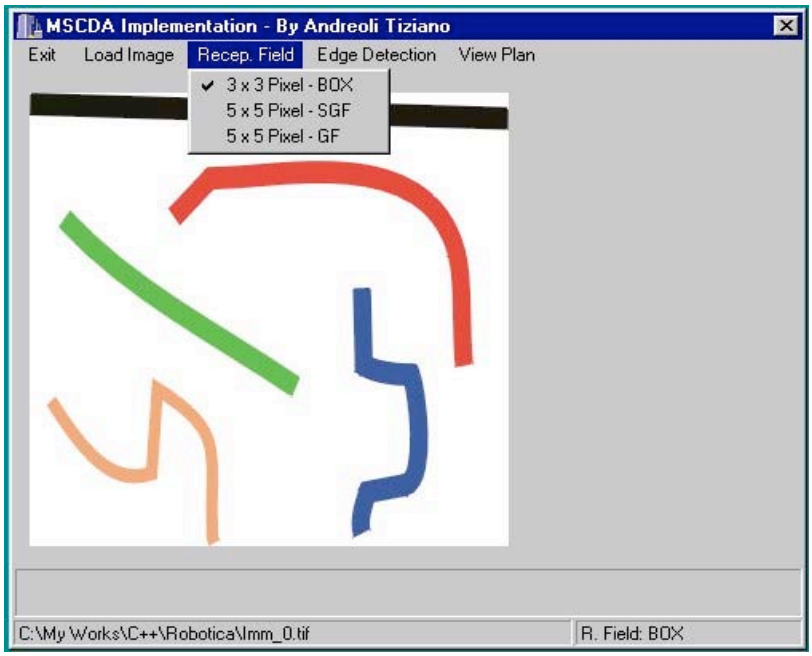
Questa voce permette di vedere i vari piani dell'immagine:

1. *Red*: il piano fondamentale del rosso;
2. *Green*: il piano fondamentale del verde;
3. *Blue*: il piano fondamentale del blu;
4. *Achromatic*: il piano monocromatico;
5. *Image*: l'immagine.

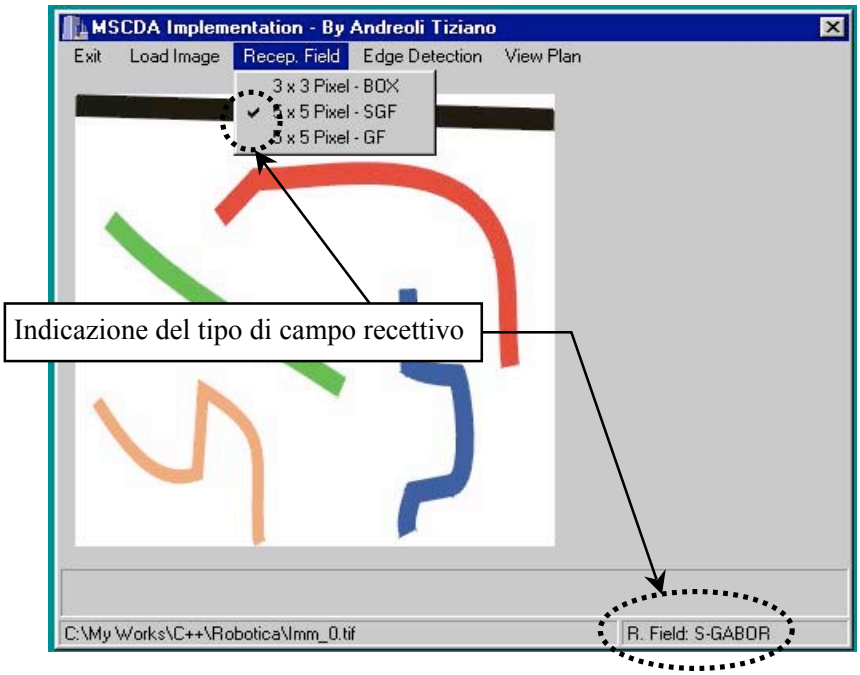
Un segno (checklist) compare sul piano che si sta visualizzando al momento.

10.5. VOCE "RECEP. FIELD"

Permette di scegliere il tipo di campo recettivo che l'algoritmo MSCD userà durante la prima fase della computazione:

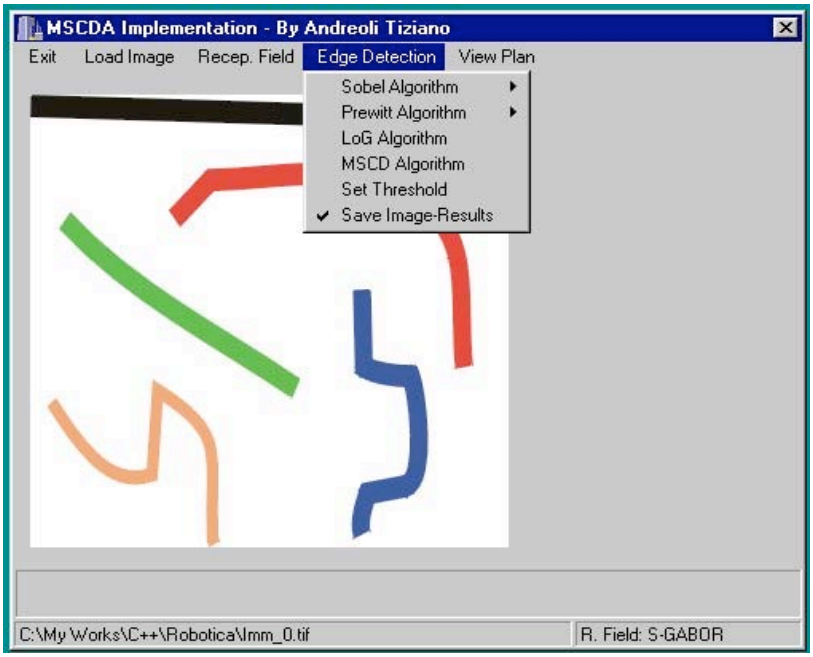


Le voci sono tre, corrispondenti ai campi recettivi implementati. Anche qui un segno indica il campo che si sta utilizzando al momento.
Facciamo notare anche che nell'angolo in basso a destra della finestra, compare una PanelBar con un campo chiamato *R. Field*: riporta il tipo di campo recettivo scelto. Infatti, se per esempio si clicca sulla voce *5x5 Pixel – SGF*, questa voce cambia:



10.6. VOCE "EDGE DETECTION"

Questa voce ha effetto solo se un'immagine è in memoria:



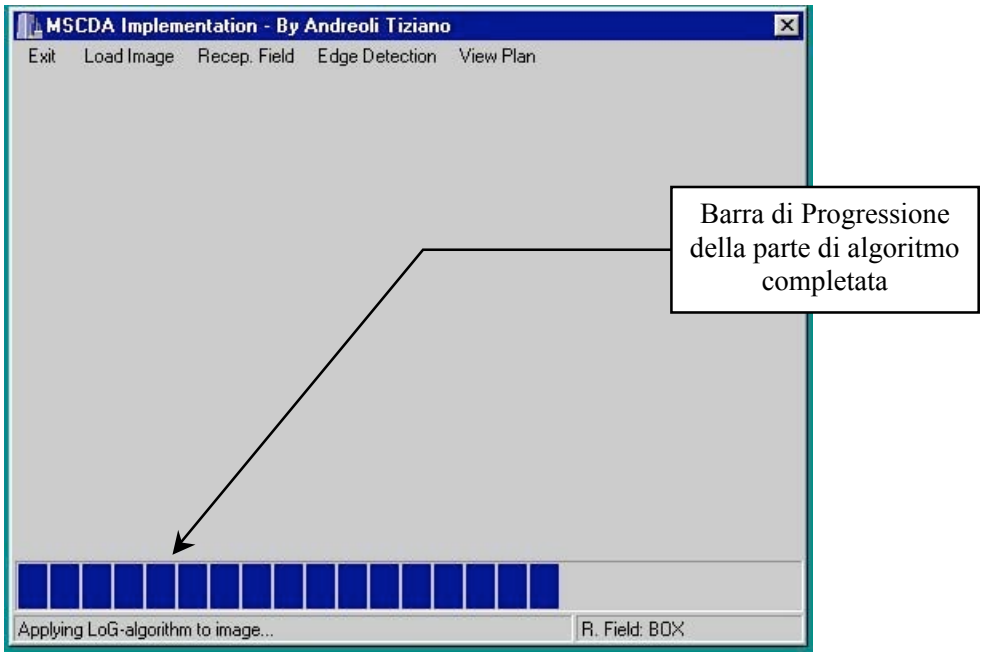
In ogni caso, le voci possibili sono sei:

1. *Sobel Algorithm*: applica l'algoritmo di Sobel (due sottovoci permettono di scegliere la direzione del gradiente);
2. *Prewitt Algorithm*: applica l'algoritmo di Prewitt (due sottovoci permettono di scegliere la direzione del gradiente);
3. *LoG Algorithm*: applica il Laplaciano del Gaussiano;
4. *MSCD Algorithm*: applica l'algoritmo MSCD;
5. *Set Threshold*: imposta la soglia per i primi 3 algoritmi;
6. *Save Image-Results*: indica se il risultato della computazione (ossia l'immagine d'uscita) sarà salvato su disco.

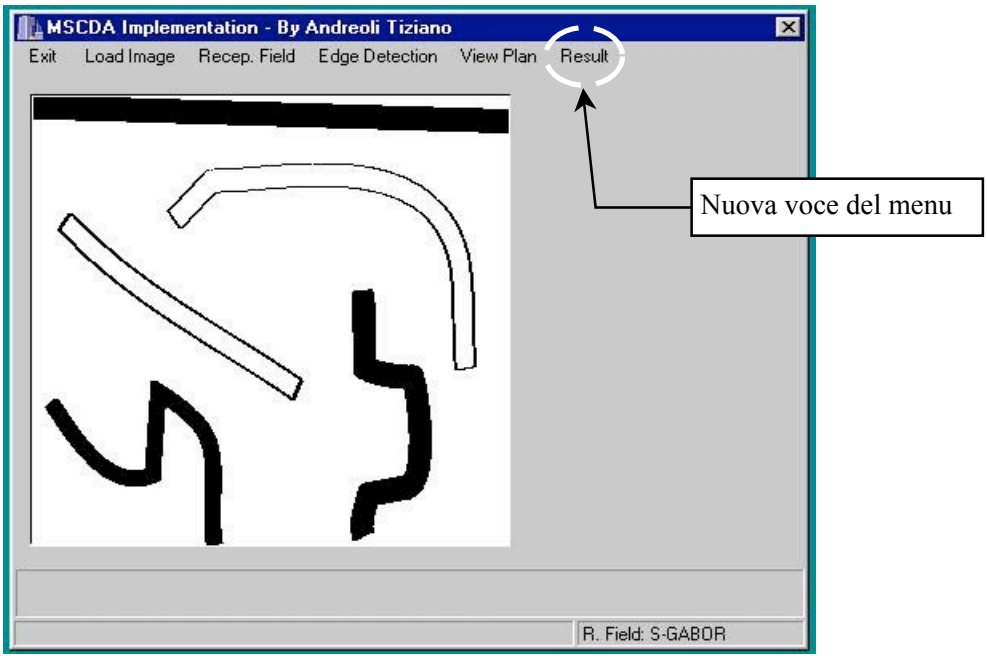
Le immagini di uscita vengono salvate in questo modo: è mantenuto il nome originale, ma gli vengono aggiunti alcuni caratteri:

- "**_SOBH**" per i risultati della voce 1 – filtraggio orizzontale;
- "**_SOBV**" per i risultati della voce 1 – filtraggio verticale;
- "**_PREH**" per i risultati della voce 2 – filtraggio orizzontale;
- "**_PREV**" per i risultati della voce 2 – filtraggio verticale;
- "**_LOG**" per i risultati della voce 3;
- "**_MSCD_3B**" per risultati della voce 4 – r. field di tipo BOX;
- "**_MSCD_5S**" per risultati della voce 4 – r. field di tipo S–GABOR;
- "**_MSCD_5G**" per risultati della voce 4 – r. field di tipo GABOR.

Anche qui una barra di progressione indica la parte di algoritmo completata:



Una volta completato l'algoritmo compare la settima voce del menu: *Result*.



Mediante questa voce è possibile visualizzare l'ultimo risultato ottenuto.

11. IL LISTATO (FILE UNIT1.CPP)

Di seguito viene riportato il listato del programma nel linguaggio C++ Builder.

11.1. DICHIARAZIONI GLOBALI

```
//---------------------------------------------------------------------------
#define X_1 0
#define X_2 1
#define X_3 2
#define X_4 3
#define CTHMAX 0.000001 // Soglia massima per il coefficiente C1234
#define CTHMIN 0.0      // Soglia minima per il coefficiente C1234
#include <vcl.h>
#include <dir.h>
#include <math.h>
#include <stdlib.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "ImageEn"
#pragma link "ImageEnView"
#pragma resource "*.dfm"

TForm1 *Form1;

//*****************************
//* DEFINIZIONE TIPI DI DATI *
//*****************************
typedef double vect_x[4][4];
typedef double maschera[5][5];

//***********************************
//* DICHIARAZIONE PROTOTIPI FUNZIONI *
//***********************************
TColor calcola_rect   (int m[][5],int,int,int,int,TCanvas *,int,int);
void   my_conv        (TImage *,int m[][5],TImage *,char *);
void   comp_soglia    (TImage *);
void   comp_soglia_log(TImage *);
void   mscd_algorithm (TImage *);

//---------------------------------------------------------------------------

bool l_image,     // Dice se cìè un'immagine in memoria
     comp_image; // Dice se all'immagnie è stato applicato un algoritmo
char *nomefile;  // Contiene il nome del file dell'immagine
int dim_cell,    // Specifica il tipo di algoritmo selezionato
    th=100,      // Soglia per gli algoritmi classici
    h,w,         // Altezza e Larghezza dell'immagine
    mas[5][5];   // Maschera per la convoluzione

//---------------------------------------------------------------------------
```

11.2. PROGRAMMAZIONE DEGLI EVENTI

```
__fastcall TForm1::TForm1(TComponent* Owner)
   : TForm(Owner)
{
}

//---------------------------------------------------------------------------

void __fastcall TForm1::Exit1Click(TObject *Sender)
{
 delete(nomefile);
 Close();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::N3x3Pixel1Click(TObject *Sender)
{
 if (N3x3Pixel1->Checked==false)
  {
   N3x3Pixel1  ->Checked=true;
   N5x5Pixel1  ->Checked=false;
   N5x5PixelGF1->Checked=false;
   dim_cell=3;
   StatusBar1->Panels->Items[1]->Text="R. Field: BOX";
  }
}

//---------------------------------------------------------------------------

void __fastcall TForm1::N5x5Pixel1Click(TObject *Sender)
{
 if (N5x5Pixel1->Checked==false)
  {
   N5x5Pixel1  ->Checked=true;
   N3x3Pixel1  ->Checked=false;
   N5x5PixelGF1->Checked=false;
   dim_cell=5;
   StatusBar1->Panels->Items[1]->Text="R. Field: S-GABOR";
  }
}

//---------------------------------------------------------------------------

void __fastcall TForm1::N5x5PixelGF1Click(TObject *Sender)
{
 if (N5x5PixelGF1->Checked==false)
  {
   N5x5Pixel1  ->Checked=false;
   N3x3Pixel1  ->Checked=false;
   N5x5PixelGF1->Checked=false;
   dim_cell=-5;
   StatusBar1->Panels->Items[1]->Text="R. Field: GABOR";
  }
}

//---------------------------------------------------------------------------

void __fastcall TForm1::FormResize(TObject *Sender)
{
 StatusBar1->Panels->Items[0]->Width=Width-150;
}

//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Immagine1Click(TObject *Sender)
{
 Red1       ->Checked=false;
 Green1     ->Checked=false;
 Blue1      ->Checked=false;
 Achromatic1->Checked=false;
 Immagine1  ->Checked=true;

 Image1->Visible=true;
 ImageR->Visible=false;
 ImageG->Visible=false;
 ImageB->Visible=false;
 ImageA->Visible=false;
 if (comp_image==true) Image_out->Visible=false;
}

//---------------------------------------------------------------------------

void __fastcall TForm1::SaveImageResults1Click(TObject *Sender)
{
 if (SaveImageResults1->Checked==true) SaveImageResults1->Checked=false;
 else                                  SaveImageResults1->Checked=true;
}

//---------------------------------------------------------------------------

void __fastcall TForm1::Achromatic1Click(TObject *Sender)
{
 Red1       ->Checked=false;
 Green1     ->Checked=false;
 Blue1      ->Checked=false;
 Achromatic1->Checked=true;
 Immagine1  ->Checked=false;

 Image1->Visible=false;
 ImageR->Visible=false;
 ImageG->Visible=false;
 ImageB->Visible=false;
 ImageA->Visible=true;
 if (comp_image==true) Image_out->Visible=false;
}

//---------------------------------------------------------------------------

void __fastcall TForm1::Red1Click(TObject *Sender)
{
 Red1       ->Checked=true;
 Green1     ->Checked=false;
 Blue1      ->Checked=false;
 Achromatic1->Checked=false;
 Immagine1  ->Checked=false;

 Image1->Visible=false;
 ImageR->Visible=true;
 ImageG->Visible=false;
 ImageB->Visible=false;
 ImageA->Visible=false;
 if (comp_image==true) Image_out->Visible=false;
}

//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Green1Click(TObject *Sender)
{
 Red1       ->Checked=false;
 Green1     ->Checked=true;
 Blue1      ->Checked=false;
 Achromatic1->Checked=false;
 Immagine1  ->Checked=false;

 Image1->Visible=false;
 ImageR->Visible=false;
 ImageG->Visible=true;
 ImageB->Visible=false;
 ImageA->Visible=false;
 if (comp_image==true) Image_out->Visible=false;
}

//---------------------------------------------------------------------------

void __fastcall TForm1::Blue1Click(TObject *Sender)
{
 Red1       ->Checked=false;
 Green1     ->Checked=false;
 Blue1      ->Checked=true;
 Achromatic1->Checked=false;
 Immagine1  ->Checked=false;

 Image1->Visible=false;
 ImageR->Visible=false;
 ImageG->Visible=false;
 ImageB->Visible=true;
 ImageA->Visible=false;
 if (comp_image==true) Image_out->Visible=false;
}

//---------------------------------------------------------------------------

void __fastcall TForm1::Risultato1Click(TObject *Sender)
{
 if (comp_image==true)
  {
   Red1       ->Checked=false;
   Green1     ->Checked=false;
   Blue1      ->Checked=false;
   Achromatic1->Checked=false;
   Immagine1  ->Checked=false;

   Image1->Visible=false;
   ImageR->Visible=false;
   ImageG->Visible=false;
   ImageB->Visible=false;
   ImageA->Visible=false;
   Image_out->Visible=true;
  }
}

//---------------------------------------------------------------------------

void __fastcall TForm1::ImpostaSoglia1Click(TObject *Sender)
{
 AnsiString stringa;

 stringa=IntToStr(th);
 stringa=InputBox("Change Threshold (0 - 255)", "New value:",stringa);
 th=stringa.ToInt();
}

//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Carica1Click(TObject *Sender)
{
 char salva[200]; // Stringa temporanea

 OpenDialog1->Title="Carica immagine TIF da un file";
 OpenDialog1->DefaultExt="*.tif";
 OpenDialog1->Filter="Immagini Tif o Bmp (*.tif,*.bmp)|*.TIF;*.BMP";
 if (OpenDialog1->Execute())
  {
   if (l_image==false) l_image=true;
   else
    {
     Image1->Visible=false;
     ImageR->Visible=false;
     ImageG->Visible=false;
     ImageB->Visible=false;
     ImageA->Visible=false;
     Image_out->Visible=false;

     nomefile=OpenDialog1->FileName.c_str();
     strcpy(salva,"Current File: ");
     strcat(salva,nomefile);
     StatusBar1->Panels->Items[0]->Text=salva;
     My_Creapiani_image();
     Immagine1Click(Sender);
     Risultato1->Visible=false;
     comp_image=false;
/*
     // ISTRUZIONI PER LA FASE DI TEST - PARTE I
     S1Click(Sender);
     Dx1Click(Sender);
     Sx1Click(Sender);
     Dx2Click(Sender);
     AlgoritmoLAPLACIANO1Click(Sender);
*/
/*
     // ISTRUZIONI PER LA FASE DI TEST - PARTE II
     N3x3Pixel1Click(Sender);
     AlgoritmoMSCD1Click(Sender);
     N5x5Pixel1Click(Sender);
     AlgoritmoMSCD1Click(Sender);
     N5x5PixelGF1Click(Sender);
     AlgoritmoMSCD1Click(Sender);
*/
  }
}

//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 nomefile=new char[100];
 comp_image=false;

 Width=500; Height=400;

 // Nasconde la voce del menu
 VediPiano1       ->Visible=false;
 Risultato1       ->Visible=false;
 N3x3Pixel1       ->Checked=true;
 N5x5Pixel1       ->Checked=false;
 N5x5PixelGF1     ->Checked=false;
 SaveImageResults1->Checked=true;

 // Imposta le immagini
 ImageEn1->Visible=false;
```

(continua…)

```
 Image1   ->Top=15;  Image1   ->Left=10;  Image1   ->AutoSize=true;
 ImageR   ->Top=15;  ImageR   ->Left=10;  ImageR   ->AutoSize=true;
 ImageG   ->Top=15;  ImageG   ->Left=10;  ImageG   ->AutoSize=true;
 ImageB   ->Top=15;  ImageB   ->Left=10;  ImageB   ->AutoSize=true;
 ImageA   ->Top=15;  ImageA   ->Left=10;  ImageA   ->AutoSize=true;
 Image_out->Top=15;  Image_out->Left=10;  Image_out->AutoSize=true;

 Image1->Visible=false; ImageR->Visible=false; ImageG->Visible=false;
 ImageB->Visible=false; ImageA->Visible=false; Image_out->Visible=false;

 dim_cell=3;
 for (int i=0; i<100; i++) nomefile[i]=' ';
 nomefile[0]='\0';
 StatusBar1->Panels->Items[0]->Width=Width-150;
 StatusBar1->Panels->Items[0]->Text=strcat("Current File: ",nomefile);
 StatusBar1->Panels->Items[1]->Text="R. Field: BOX";
 Caption="MSCDA Implementation - By Andreoli Tiziano";

 ProgressBarR->Top =Height-ProgressBarR->Height-63;
 ProgressBarR->Left=1;
 ProgressBarR->Width=Width-7;
 ProgressBarR->Position=0;

 StatusBar1->Visible=true;
}

//---------------------------------------------------------------------------
void __fastcall TForm1::AlgoritmoLAPLACIANO1Click(Tobject
                                                  *Sender)
{
 char s[_MAX_PATH];

 // Costruisce il nome dell'immagine di uscita da salvare
 strcpy(s,"\0");
 if (SaveImageResults1->Checked==true)
  {
   char drive[_MAX_DRIVE],dir[_MAX_DIR],filen[_MAX_FNAME],ext[_MAX_EXT];

   _splitpath(nomefile,drive,dir,filen,ext);
   strcat(filen,"_LOG"); // Appende la sequenza particolare
   _makepath(s,drive,dir,filen,".bmp");
  }
 else s[0]='\0';

 if (l_image==true)
  {
   Image1->Visible=false; ImageR->Visible=false; ImageG->Visible=false;
   ImageB->Visible=false; ImageA->Visible=false; Image_out->Visible=false;

   // Definisco la maschera
   mas[0][0]= 0; mas[0][1]= 0; mas[0][2]=-1; mas[0][3]= 0; mas[0][4]= 0;
   mas[1][0]= 0; mas[1][1]=-1; mas[1][2]=-2; mas[1][3]=-1; mas[1][4]= 0;
   mas[2][0]=-1; mas[2][1]=-2; mas[2][2]=16; mas[2][3]=-2; mas[2][4]=-1;
   mas[3][0]= 0; mas[3][1]=-1; mas[3][2]=-2; mas[3][3]=-1; mas[3][4]= 0;
   mas[4][0]= 0; mas[4][1]= 0; mas[4][2]=-1; mas[4][3]= 0; mas[4][4]= 0;
   my_conv(ImageA,mas,Image_out,"Applying LoG-algorithm to image...");
   comp_soglia_log(Image_out);

   Risultato1->Visible=true;
   Image_out->Visible=true;
   comp_image=true;

   if (s[0]!='\0') Image_out->Picture->SaveToFile(s);
   ProgressBarR->Position=0;
  }
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::S1Click(TObject *Sender)
{
 char s[_MAX_PATH];

 // Costruisce il nome dell'immagine di uscita da salvare
 strcpy(s,"\0");
 if (SaveImageResults1->Checked==true)
  {
   char drive[_MAX_DRIVE],dir[_MAX_DIR],filen[_MAX_FNAME],ext[_MAX_EXT];

   _splitpath(nomefile,drive,dir,filen,ext);
   strcat(filen,"_SOBV"); // Appende la sequenza particolare
   _makepath(s,drive,dir,filen,".bmp");
  }
 else s[0]='\0';

 if (l_image==true)
  {
   Image1->Visible=false; ImageR->Visible=false; ImageG->Visible=false;
   ImageB->Visible=false; ImageA->Visible=false; Image_out->Visible=false;

   // Definisco la maschera
   mas[0][0]=0; mas[0][1]= 0; mas[0][2]=0; mas[0][3]=0; mas[0][4]=0;
   mas[1][0]=0; mas[1][1]=-1; mas[1][2]=0; mas[1][3]=1; mas[1][4]=0;
   mas[2][0]=0; mas[2][1]=-2; mas[2][2]=0; mas[2][3]=2; mas[2][4]=0;
   mas[3][0]=0; mas[3][1]=-1; mas[3][2]=0; mas[3][3]=1; mas[3][4]=0;
   mas[4][0]=0; mas[4][1]= 0; mas[4][2]=0; mas[4][3]=0; mas[4][4]=0;
   my_conv(ImageA,mas,Image_out,"Applying SOBEL-algorithm to image - Vertical
                                                                    Filtering...");
   comp_soglia(Image_out);

   Risultato1->Visible=true;
   Image_out->Visible=true;
   comp_image=true;
   if (s[0]!='\0') Image_out->Picture->SaveToFile(s);
   ProgressBarR->Position=0;
 }
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Dx1Click(TObject *Sender)
{
 char s[_MAX_PATH];

 // Costruisce il nome dell'immagine di uscita da salvare
 strcpy(s,"\0");
 if (SaveImageResults1->Checked==true)
  {
   char drive[_MAX_DRIVE],dir[_MAX_DIR],filen[_MAX_FNAME],ext[_MAX_EXT];

   _splitpath(nomefile,drive,dir,filen,ext);
   strcat(filen,"_SOBH"); // Appende la sequenza particolare
   _makepath(s,drive,dir,filen,".bmp");
  }
 else s[0]='\0';
 if (l_image==true)
  {
   Image1->Visible=false; ImageR->Visible=false; ImageG->Visible=false;
   ImageB->Visible=false; ImageA->Visible=false; Image_out->Visible=false;
   // Definisco la maschera
   mas[0][0]=0; mas[0][1]= 0; mas[0][2]= 0; mas[0][3]= 0; mas[0][4]=0;
   mas[1][0]=0; mas[1][1]= 1; mas[1][2]= 2; mas[1][3]= 1; mas[1][4]=0;
   mas[2][0]=0; mas[2][1]= 0; mas[2][2]= 0; mas[2][3]= 0; mas[2][4]=0;
   mas[3][0]=0; mas[3][1]=-1; mas[3][2]=-2; mas[3][3]=-1; mas[3][4]=0;
   mas[4][0]=0; mas[4][1]= 0; mas[4][2]= 0; mas[4][3]= 0; mas[4][4]=0;

  (continua…)
```

```
   my_conv(ImageA,mas,Image_out,"Applying SOBEL-algorithm to image - Horizontal
                                                                    Filtering...");
   comp_soglia(Image_out);
   Risultato1->Visible=true;
   Image_out->Visible=true;
   comp_image=true;

   if (s[0]!='\0') Image_out->Picture->SaveToFile(s);
   ProgressBarR->Position=0;
  }
}

//---------------------------------------------------------------------------

void __fastcall TForm1::Sx1Click(TObject *Sender)
{
 char s[_MAX_PATH];

 // Costruisce il nome dell'immagine di uscita da salvare
 strcpy(s,"\0");
 if (SaveImageResults1->Checked==true)
  {
   char drive[_MAX_DRIVE],dir[_MAX_DIR],filen[_MAX_FNAME],ext[_MAX_EXT];

   _splitpath(nomefile,drive,dir,filen,ext);
   strcat(filen,"_PREV"); // Appende la sequenza particolare
   _makepath(s,drive,dir,filen,".bmp");
  }
 else s[0]='\0';

 if (l_image==true)
  {
   Image1->Visible=false; ImageR->Visible=false; ImageG->Visible=false;
   ImageB->Visible=false; ImageA->Visible=false; Image_out->Visible=false;

   // Definisco la maschera
   mas[0][0]=0; mas[0][1]= 0; mas[0][2]=0; mas[0][3]=0; mas[0][4]=0;
   mas[1][0]=0; mas[1][1]=-1; mas[1][2]=0; mas[1][3]=1; mas[1][4]=0;
   mas[2][0]=0; mas[2][1]=-1; mas[2][2]=0; mas[2][3]=1; mas[2][4]=0;
   mas[3][0]=0; mas[3][1]=-1; mas[3][2]=0; mas[3][3]=1; mas[3][4]=0;
   mas[4][0]=0; mas[4][1]= 0; mas[4][2]=0; mas[4][3]=0; mas[4][4]=0;
   my_conv(ImageA,mas,Image_out,"Applying PREWITT-algorithm to image - Vertical
                                                                    Filtering...");
   comp_soglia(Image_out);

   Risultato1->Visible=true;
   Image_out->Visible=true;
   comp_image=true;

   if (s[0]!='\0') Image_out->Picture->SaveToFile(s);
   ProgressBarR->Position=0;
  }
}

//---------------------------------------------------------------------------

void __fastcall TForm1::Dx2Click(TObject *Sender)
{
 char s[_MAX_PATH];

 // Costruisce il nome dell'immagine di uscita da salvare
 strcpy(s,"\0");

 if (SaveImageResults1->Checked==true)
  {
   char drive[_MAX_DRIVE],dir[_MAX_DIR],filen[_MAX_FNAME],ext[_MAX_EXT];

   (continua…)
```

```
   _splitpath(nomefile,drive,dir,filen,ext);
   strcat(filen,"_PREH"); // Appende la sequenza particolare
   _makepath(s,drive,dir,filen,".bmp");
  }
 else s[0]='\0';

 if (l_image==true)
  {
   Image1->Visible=false; ImageR->Visible=false; ImageG->Visible=false;
   ImageB->Visible=false; ImageA->Visible=false; Image_out->Visible=false;

   // Definisco la maschera
   mas[0][0]=0; mas[0][1]= 0; mas[0][2]= 0; mas[0][3]= 0; mas[0][4]=0;
   mas[1][0]=0; mas[1][1]= 1; mas[1][2]= 1; mas[1][3]= 1; mas[1][4]=0;
   mas[2][0]=0; mas[2][1]= 0; mas[2][2]= 0; mas[2][3]= 0; mas[2][4]=0;
   mas[3][0]=0; mas[3][1]=-1; mas[3][2]=-1; mas[3][3]=-1; mas[3][4]=0;
   mas[4][0]=0; mas[4][1]= 0; mas[4][2]= 0; mas[4][3]= 0; mas[4][4]=0;
   my_conv(ImageA,mas,Image_out,"Applying PREWITT-algorithm to image - Horizontal
                                                                    Filtering...");
   comp_soglia(Image_out);

   Risultato1->Visible=true;
   Image_out->Visible=true;
   comp_image=true;

   if (s[0]!='\0') Image_out->Picture->SaveToFile(s);
   ProgressBarR->Position=0;
  }
}

//---------------------------------------------------------------------------

void __fastcall TForm1::AlgoritmoMSCD1Click(TObject *Sender)
{
 char s[_MAX_PATH];

 // Costruisce il nome dell'immagine di uscita da salvare
 strcpy(s,"\0");
 if (SaveImageResults1->Checked==true)
  {
   char drive[_MAX_DRIVE],dir[_MAX_DIR],filen[_MAX_FNAME],ext[_MAX_EXT];

   _splitpath(nomefile,drive,dir,filen,ext);
   strcat(filen,"_MSCD"); // Appende la sequenza particolare
   if (dim_cell== 3) strcat(filen,"_3b");
   if (dim_cell== 5) strcat(filen,"_5s");
   if (dim_cell==-5) strcat(filen,"_5g");
   _makepath(s,drive,dir,filen,".bmp");
  }
 else s[0]='\0';

 if (l_image==true)
  {
   Image1->Visible=false; ImageR->Visible=false; ImageG->Visible=false;
   ImageB->Visible=false; ImageA->Visible=false; Image_out->Visible=false;

   mscd_algorithm(Image_out);
   comp_image=true;
   Risultato1->Visible=true;
   Image_out->Visible=true;

   if (s[0]!='\0') Image_out->Picture->SaveToFile(s);
   ProgressBarR->Position=0;
  }
}

//---------------------------------------------------------------------------
```

11.3. PROCEDURE AGGIUNTIVE

```
//****************
//* FUNZIONI MIE *
//****************

// Data l'immagine in ingresso, crea i 3 piani R, G, B più il quarto acromatico
void __fastcall TForm1::My_Creapiani_image(void)
{
 char salva[200];                  // Update per il testo nella panelbar
 long int aux,auxr,auxg,auxb;      // Componenti per i colori
 int j,k;                          // Contatori

 // Salva la vecchia stringa della PanelBar
 strcpy(salva,nomefile);
 StatusBar1->Panels->Items[0]->Text="Loading image from file...";
 ProgressBarR->Visible=true;

 // Carica l'immagine da file
 if (nomefile[strlen(nomefile)-1]=='p' || nomefile[strlen(nomefile)-1]=='P')
  {
   // L'immagine è di tipo Bitmap BMP
   Image1->Picture->LoadFromFile(nomefile); ImageR->Picture->LoadFromFile(nomefile);
   ImageG->Picture->LoadFromFile(nomefile); ImageB->Picture->LoadFromFile(nomefile);
   ImageA->Picture->LoadFromFile(nomefile); Image_out->Picture->LoadFromFile(nomefile);
  }
 else
  {
   // L'immagine è di tipo TIFF
   ImageEn1->LoadFromFile(nomefile);

   Image1->Picture->Bitmap->Assign(ImageEn1->Bitmap);
   ImageR->Picture->Bitmap->Assign(ImageEn1->Bitmap);
   ImageG->Picture->Bitmap->Assign(ImageEn1->Bitmap);
   ImageB->Picture->Bitmap->Assign(ImageEn1->Bitmap);
   ImageA->Picture->Bitmap->Assign(ImageEn1->Bitmap);
   Image_out->Picture->Bitmap->Assign(ImageEn1->Bitmap);
  }

 h=Image1->Height; w=Image1->Width;

 // Imposta e crea i 4 piani...
 ImageR->Width =w; ImageG->Width =w; ImageB->Width =w; ImageA->Width =w; Image_out->Width =w;
 ImageR->Height=h; ImageG->Height=h; ImageB->Height=h; ImageA->Height=h; Image_out->Height=h;

 // Imposta la barra
 ProgressBarR->Max=h;
 ProgressBarR->Min=0;
 ProgressBarR->Position=0;
 ProgressBarR->Step=1;
 StatusBar1->Panels->Items[0]->Text="Creating Image Planes...";

 // Crea i piani
 for(j=0; j<h; j++)
  {
   for(k=0; k<w; k++)
    {
     // Componente R
     aux=ImageR->Picture->Bitmap->Canvas->Pixels[k][j];
     aux &= 0x000000FF;
     auxr = aux;
     ImageR->Picture->Bitmap->Canvas->Pixels[k][j]=(TColor)aux;

     (continua…)
```

```
     // Componente G
     aux=ImageG->Picture->Bitmap->Canvas->Pixels[k][j];
     aux &= 0x0000FF00;
     auxg = aux >> 8;
     ImageG->Picture->Bitmap->Canvas->Pixels[k][j]=(TColor)aux;

     // Componente B
     aux=ImageB->Picture->Bitmap->Canvas->Pixels[k][j];
     aux &= 0x00FF0000;
     auxb = aux >> 16;
     ImageB->Picture->Bitmap->Canvas->Pixels[k][j]=(TColor)aux;

     aux=(auxr+auxg+auxb)/3.0; // Determino la componente media
     ImageA->Picture->Bitmap->Canvas->Pixels[k][j]=(TColor)((aux<<16)+(aux<<8)+aux);
    }
   ProgressBarR->StepIt();
  }

 VediPiano1->Visible=true; // Crea la nuova voce del menu

 // Resiza la finestra principale se l'immagine supera i margini
 if ((Image1->Width )>(ClientWidth -30)) Width =Image1->Width+100;
 if ((Image1->Height)>(ClientHeight-60)) Height=Image1->Height+125;

 ProgressBarR->Visible=false;
 ProgressBarR->Top =Height-ProgressBarR->Height-63;
 ProgressBarR->Position=0;
 ProgressBarR->Left=1;
 ProgressBarR->Width=Width-7;
 StatusBar1->Panels->Items[0]->Text=salva;
}

//---------------------------------------------------------------------------

// Esegue la convoluzione fra l'immagine puntata da pim e la maschera mask; pone il
// risultato in pimo e usa come testo esplicativo nella panelbar "stringa".
void my_conv(TImage *pim,int mask[][5],TImage *pimo,
                                                    char *stringa)
{
 char salva[200];   // Update per il testo nella panelbar
 TCanvas *pc,*pout; // Puntatori alle immagini
 int i,j;           // Contatori

 pc  =pim ->Picture->Bitmap->Canvas;
 pout=pimo->Picture->Bitmap->Canvas;

 // Imposta la barra
 Form1->ProgressBarR->Max=w*h;
 Form1->ProgressBarR->Position=0;
 Form1->ProgressBarR->Min=0;
 Form1->ProgressBarR->Step=1;

 // Salva la vecchia stringa della PanelBar
 strcpy(salva,nomefile);
 Form1->StatusBar1->Panels->Items[0]->Text=stringa;

 //**************************
 //* Ciclo di processing... *
 //**************************
 // Caselle Speciali
 pout->Pixels[  0][  0]=calcola_rect(mask,2,4,2,4,pc,  0,  0);
 pout->Pixels[  0][  1]=calcola_rect(mask,1,4,2,4,pc,  0,  1);
 pout->Pixels[  0][h-2]=calcola_rect(mask,2,4,0,3,pc,  0,h-2);
 pout->Pixels[  0][h-1]=calcola_rect(mask,2,4,0,2,pc,  0,h-1);
 Form1->ProgressBarR->StepBy(4);
```

(continua…)

```
 pout->Pixels[  1][  0]=calcola_rect(mask,1,4,2,4,pc,  1,  0);
 pout->Pixels[  1][  1]=calcola_rect(mask,1,4,1,4,pc,  1,  1);
 pout->Pixels[  1][h-2]=calcola_rect(mask,1,4,0,3,pc,  1,h-2);
 pout->Pixels[  1][h-1]=calcola_rect(mask,1,4,0,2,pc,  1,h-1);
 Form1->ProgressBarR->StepBy(4);
 pout->Pixels[w-1][  0]=calcola_rect(mask,0,2,2,4,pc,w-1,  0);
 pout->Pixels[w-1][  1]=calcola_rect(mask,0,2,1,4,pc,w-1,  1);
 pout->Pixels[w-1][h-2]=calcola_rect(mask,0,2,0,3,pc,w-1,h-2);
 pout->Pixels[w-1][h-1]=calcola_rect(mask,0,2,0,2,pc,w-1,h-1);
 Form1->ProgressBarR->StepBy(4);
 pout->Pixels[w-2][  0]=calcola_rect(mask,0,3,2,4,pc,w-2,  0);
 pout->Pixels[w-2][  1]=calcola_rect(mask,0,3,2,3,pc,w-2,  1);
 pout->Pixels[w-2][h-2]=calcola_rect(mask,0,3,0,3,pc,w-2,h-2);
 pout->Pixels[w-2][h-1]=calcola_rect(mask,0,3,0,2,pc,w-2,h-1);
 Form1->ProgressBarR->StepBy(4);

 // Righe speciali
 for(i=2; i<w-2; i++)
  {
   pout->Pixels[i][  0]=calcola_rect(mask,0,4,2,4,pc,i,  0); // Riga 0
   pout->Pixels[i][  1]=calcola_rect(mask,0,4,1,4,pc,i,  1); // Riga 1
   pout->Pixels[i][h-2]=calcola_rect(mask,0,4,0,3,pc,i,h-2); // Riga h-2
   pout->Pixels[i][h-1]=calcola_rect(mask,0,4,0,2,pc,i,h-1); // Riga h-1
   Form1->ProgressBarR->StepBy(4);
  }

 // Colonne speciali
 for(i=2; i<h-2; i++)
  {
   pout->Pixels[  0][i]=calcola_rect(mask,2,4,0,4,pc,  0,i); // Colonna 0
   pout->Pixels[  1][i]=calcola_rect(mask,1,4,0,4,pc,  1,i); // Colonna 1
   pout->Pixels[w-2][i]=calcola_rect(mask,0,3,0,4,pc,w-2,i); // Colonna w-2
   pout->Pixels[w-1][i]=calcola_rect(mask,0,2,0,4,pc,w-1,i); // Colonna w-1
   Form1->ProgressBarR->StepBy(4);
  }

 // Parte Interna
 for(i=2; i<w-2; i++)
  {
   for(j=2; j<h-2; j++) pout->Pixels[i][j]=calcola_rect(mask,0,4,0,4,pc,i,j);
   Form1->ProgressBarR->StepBy(h-4);
  }

 Form1->StatusBar1->Panels->Items[0]->Text=salva;
}

//---------------------------------------------------------------------------

// Calcola la somma dei valori dei pixel attorno al pixel "(x,y)" moltiplicati per i
// rispettivi valori (pesi) della maschera "m". La maschera è centrata in "(x,y)".
// L'immagine è puntata da "pci" mentre "am,bm,cm,dm" sono gli estremi del rettangolo.
TColor calcola_rect(int m[][5],int am, int cm,int bm,
                                    int dm,TCanvas *pci,int x,int y)
{
 int i,k,aux; // Contatori e variabile ausiliaria

 aux=0;
 for(k=bm; k<=dm; k++)
   for(i=am; i<=cm; i++) aux+=m[k][i]*pci->Pixels[x-(2-i)][y-(2-k)];
 return ((TColor)aux);
}

//---------------------------------------------------------------------------
```

```
// Trova i bordi con il metodo del thresholding
void comp_soglia(TImage *pi)
{
 char salva[200]; // Update per il testo nella panelbar
 long int thre;   // Soglia
 TCanvas *pc;     // Puntatori alle immagini
 int i,k;         // Contatori

 strcpy(salva,nomefile);

 Form1->ProgressBarR->Max=w;
 Form1->ProgressBarR->Position=0;
 Form1->ProgressBarR->Min=0;
 Form1->ProgressBarR->Step=1;
 Form1->StatusBar1->Panels->Items[0]->Text="Finding Edges...";

 pc=pi->Picture->Bitmap->Canvas;
 thre=((th<<16)+(th<<8)+th);
 for(i=0; i<w; i++)
  {
   for(k=0; k<h; k++)
     if ((pc->Pixels[i][k])>((TColor)thre)) pc->Pixels[i][k]=(TColor)0x00000000;
     else                                   pc->Pixels[i][k]=(TColor)0x00FFFFFF;
   Form1->ProgressBarR->StepIt();
  }

 Form1->StatusBar1->Panels->Items[0]->Text=salva;
}

//---------------------------------------------------------------------------

// Trova i bordi con il metodo dello zero crossing
void comp_soglia_log(TImage *pi)
{
 char salva[200]; // Update per il testo nella panelbar
 TCanvas *pc;     // Puntatori alle immagini
 int i,k;         // Contatori

 strcpy(salva,nomefile);

 Form1->ProgressBarR->Max=w; Form1->ProgressBarR->Position=0;
 Form1->ProgressBarR->Min=0; Form1->ProgressBarR->Step=1;
 Form1->StatusBar1->Panels->Items[0]->Text="Finding Edges...";

 pc=pi->Picture->Bitmap->Canvas;
 for(i=0; i<w; i++)
  {
   for(k=0; k<h; k++)
    {
     if (i>0)
      {
       // Se c'è uno zero-crossing
       if ((pc->Pixels[i][k]>=0) && (pc->Pixels[i+1][k]<0) ||
           (pc->Pixels[i][k]<=0) && (pc->Pixels[i+1][k]>0))
         pc->Pixels[i][k]=(TColor)0x00000000;
       else
         pc->Pixels[i][k]=(TColor)0x00FFFFFF;
      }
    }
   Form1->ProgressBarR->StepIt();
  }

 Form1->StatusBar1->Panels->Items[0]->Text=salva;
}

//---------------------------------------------------------------------------
```

```
//*******************
//* MSCD - FUNZIONI *
//*******************

// Calcola l'MDM per due vettori
double comp_mdm(double v1[],double v2[])
{
 double m1,m2;

 m1=sqrt(v1[0]*v1[0]+v1[1]*v1[1]+v1[2]*v1[2]); // Calcola il modulo di X1
 m2=sqrt(v2[0]*v2[0]+v2[1]*v2[1]+v2[2]*v2[2]); // Calcola il modulo di X2

 if (m2==0.0) // Se uno dei due è il vettor nullo
  {
   if (m1==0.0) return 1.0; // I due vettori sono uguali
   else         return 0.0; // I due vettori sono diversi al 100%
  }
 else
  {
   if (m1<m2) return m1/m2;
   else       return m2/m1;
  }
}

//---------------------------------------------------------------------------

// Calcola l'ADM per due vettori
double comp_adm(double v1[],double v2[])
{
 double pscal,m1,m2,x;

 pscal=v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2];    // Calcola il prodotto scalare
 m1=sqrt(v1[0]*v1[0]+v1[1]*v1[1]+v1[2]*v1[2]); //    "     " modulo di X1
 m2=sqrt(v2[0]*v2[0]+v2[1]*v2[1]+v2[2]*v2[2]); //    "     "    "   "  X2

 if (m2==0.0 || m1==0.0) x=0.0;
 else                    x=pscal/(m1*m2);

 return ((3.14159265358979-acos(x))/3.14159265358979);
}

//---------------------------------------------------------------------------

// Calcola il VDM per due vettori
double comp_vdm(double v1[],double v2[])
{
 return comp_mdm(v1,v2)*comp_adm(v1,v2);
}

//---------------------------------------------------------------------------

// Funzione che confronta due array "v1" e "v2" e ritorna:
// * +1 se v1 >= v2;
// * -1 se v1 < v2;
int maggiore_di(double v1[],double v2[])
{
 double m1,m2,cv;
 bool e;
 int i,s;

 s=1;
 m1=sqrt(v1[0]*v1[0]+v1[1]*v1[1]+v1[2]*v1[2]); // Modulo di V1
 m2=sqrt(v2[0]*v2[0]+v2[1]*v2[1]+v2[2]*v2[2]); // Modulo di V2
 cv=1.0-comp_vdm(v1,v2);
```

(continua…)

```
 if (cv>0) s=1;
 else
  {
   if (m1==m2) s=-1;
   else
    {
     if (m1<m2) s=-1;
     else
      {
       if (m1>m2)
        {
         s=+1;
         e=true;
         for(i=0; i<=X_4; i++)
           if (v1[i]<v2[i]) e=false;
         if (e==false) s=-1;
        }
      }
    }
  }
 return s;
}

//-------------------------------------------------------------------------------

// Controlla se tutti i pixels vicini a "(x,y)" hanno il coefficiente C1234=0, dato
// l'array bidimensionale "c": in tal caso "(x,y)" è un pixel speciale.
bool controlla_pixel(double **c,int x,int y)
{
 // E' uno dei quattro angoli
 if (x==0 && y==0)
   return
    ((c[y  ][x+1]>=CTHMIN && c[y  ][x+1]<=CTHMAX) &&
     (c[y+1][  x]>=CTHMIN && c[y+1][  x]<=CTHMAX) &&
     (c[y+1][x+1]>=CTHMIN && c[y+1][x+1]<=CTHMAX));

 if (x==0 && y==h-1)
   return
    ((c[y-1][  x]>=CTHMIN && c[y-1][  x]<=CTHMAX) &&
     (c[y-1][x+1]>=CTHMIN && c[y-1][x+1]<=CTHMAX) &&
     (c[y  ][x+1]>=CTHMIN && c[y  ][x+1]<=CTHMAX));

 if (x==w-1 && y==0)
   return
    ((c[y  ][x-1]>=CTHMIN && c[y  ][x-1]<=CTHMAX) &&
     (c[y+1][x-1]>=CTHMIN && c[y+1][x-1]<=CTHMAX) &&
     (c[y+1][  x]>=CTHMIN && c[y+1][  x]<=CTHMAX));

 if (x==w-1 && y==h-1)
   return
    ((c[y-1][x-1]>=CTHMIN && c[y-1][x-1]<=CTHMAX) &&
     (c[y-1][  x]>=CTHMIN && c[y-1][  x]<=CTHMAX) &&
     (c[y  ][x-1]>=CTHMIN && c[y  ][x-1]<=CTHMAX));

 // E' un pixel in posizione normale
 if (x>0 && y>0 && (x<w-1) && (y<h-1))
   return
    ((c[y-1][x-1]>=CTHMIN && c[y-1][x-1]<=CTHMAX) &&
     (c[y-1][  x]>=CTHMIN && c[y-1][  x]<=CTHMAX) &&
     (c[y-1][x+1]>=CTHMIN && c[y-1][x+1]<=CTHMAX) &&
     (c[y  ][x-1]>=CTHMIN && c[y  ][x-1]<=CTHMAX) &&
     (c[y  ][x+1]>=CTHMIN && c[y  ][x+1]<=CTHMAX) &&
     (c[y+1][x-1]>=CTHMIN && c[y+1][x-1]<=CTHMAX) &&
     (c[y+1][  x]>=CTHMIN && c[y+1][  x]<=CTHMAX) &&
     (c[y+1][x+1]>=CTHMIN && c[y+1][x+1]<=CTHMAX));
```

(continua…)

```
 else // E' un pixel di frontiera
  {
   // Pixel sulla prima colonna
   if (x==0)
     return
      ((c[y-1][  x]>=CTHMIN && c[y-1][  x]<=CTHMAX) &&
       (c[y-1][x+1]>=CTHMIN && c[y-1][x+1]<=CTHMAX) &&
       (c[y  ][x+1]>=CTHMIN && c[y  ][x+1]<=CTHMAX) &&
       (c[y+1][  x]>=CTHMIN && c[y+1][  x]<=CTHMAX) &&
       (c[y+1][x+1]>=CTHMIN && c[y+1][x+1]<=CTHMAX));

   // Pixel sulla prima riga
   if (y==0)
     return
      ((c[y  ][x-1]>=CTHMIN && c[y  ][x-1]<=CTHMAX) &&
       (c[y  ][x+1]>=CTHMIN && c[y  ][x+1]<=CTHMAX) &&
       (c[y+1][x-1]>=CTHMIN && c[y+1][x-1]<=CTHMAX) &&
       (c[y+1][  x]>=CTHMIN && c[y+1][  x]<=CTHMAX) &&
       (c[y+1][x+1]>=CTHMIN && c[y+1][x+1]<=CTHMAX));

   // Pixel sull'ultima colonna
   if (x==w-1)
     return
      ((c[y-1][x-1]>=CTHMIN && c[y-1][x-1]<=CTHMAX) &&
       (c[y-1][  x]>=CTHMIN && c[y-1][  x]<=CTHMAX) &&
       (c[y  ][x-1]>=CTHMIN && c[y  ][x-1]<=CTHMAX) &&
       (c[y+1][x-1]>=CTHMIN && c[y+1][x-1]<=CTHMAX) &&
       (c[y+1][  x]>=CTHMIN && c[y+1][  x]<=CTHMAX));

   // Pixel sull'ultima riga
   if (y==h-1)
     return
      ((c[y-1][x-1]>=CTHMIN && c[y-1][x-1]<=CTHMAX) &&
       (c[y-1][  x]>=CTHMIN && c[y-1][  x]<=CTHMAX) &&
       (c[y-1][x+1]>=CTHMIN && c[y-1][x+1]<=CTHMAX) &&
       (c[y  ][x-1]>=CTHMIN && c[y  ][x-1]<=CTHMAX) &&
       (c[y  ][x+1]>=CTHMIN && c[y  ][x+1]<=CTHMAX));
  }
 return false;
}

//---------------------------------------------------------------------------

// Simile alla 'calcola_rect' ma qui utilizza i campi recettivi; le immagini sono
// puntate da "pir", "pig", "pib"; "mo" ed "me" sono le due maschere Mask1 e Mask2,
// "a,b,c,d" sono gli estremi del rettangolo e "v" è il vettore dei valori medi
// forniti (X1,X2,X3,X4) dove gli X sono vettori di 3 componenti (r,g,b).
void calcola_rect_special_a(TImage *pir,TImage *pig,
                            Timage *pib,maschera mo,maschera me,int a,
                              int c,int b,int d,int x,int y,vect_x *v)
{
 TCanvas *pcr,*pcg,*pcb;
 int i;

 // Inizializzazioni
 *v[X_1][0] = *v[X_1][1] = *v[X_1][2] = *v[X_1][3] = 0.0;
 *v[X_2][0] = *v[X_2][1] = *v[X_2][2] = *v[X_2][3] = 0.0;
 *v[X_3][0] = *v[X_3][1] = *v[X_3][2] = *v[X_3][3] = 0.0;
 *v[X_4][0] = *v[X_4][1] = *v[X_4][2] = *v[X_4][3] = 0.0;

 pcr=pir->Picture->Bitmap->Canvas;
 pcg=pig->Picture->Bitmap->Canvas;
 pcb=pib->Picture->Bitmap->Canvas;
```

(continua…)

```
 if (((c-a)==4) && ((d-b)==4)) // E' una casella normale
  {
   for(i=a; i<=c; i++) // Calcola i vettori X
    {
     // Componente R
     *v[X_1][0]+=mo[1][i]*((double)(pcr->Pixels[x-(2-i)][y-1] & 0x000000FF));
     *v[X_2][0]+=mo[2][i]*((double)(pcr->Pixels[x-(2-i)][y+1] & 0x000000FF));
     *v[X_3][0]+=mo[3][i]*((double)(pcr->Pixels[x-(2-i)][y  ] & 0x000000FF));

     // Componente G
     *v[X_1][1]+=mo[1][i]*((double)((pcg->Pixels[x-(2-i)][y-1] & 0x0000FF00)>>8));
     *v[X_2][1]+=mo[2][i]*((double)((pcg->Pixels[x-(2-i)][y+1] & 0x0000FF00)>>8));
     *v[X_3][1]+=mo[3][i]*((double)((pcg->Pixels[x-(2-i)][y  ] & 0x0000FF00)>>8));

     // Componente B
     *v[X_1][2]+=mo[1][i]*((double)((pcb->Pixels[x-(2-i)][y-1] & 0x00FF0000)>>16));
     *v[X_2][2]+=mo[2][i]*((double)((pcb->Pixels[x-(2-i)][y+1] & 0x00FF0000)>>16));
     *v[X_3][2]+=mo[3][i]*((double)((pcb->Pixels[x-(2-i)][y  ] & 0x00FF0000)>>16));
    }
  }
 else // E' una casella speciale
  {
   if (b==2) // Non c'è Area1
    {
     for(i=a; i<=c; i++) // Calcola i vettori X
      {
       // Componente R
       *v[X_2][0]+=mo[2][i]*((double)(pcr->Pixels[x-(2-i)][y+1] & 0x000000FF));
       *v[X_3][0]+=mo[3][i]*((double)(pcr->Pixels[x-(2-i)][y  ] & 0x000000FF));

       // Componente G
       *v[X_2][1]+=mo[2][i]*((double)((pcg->Pixels[x-(2-i)][y+1] & 0x0000FF00)>>8));
       *v[X_3][1]+=mo[3][i]*((double)((pcg->Pixels[x-(2-i)][y  ] & 0x0000FF00)>>8));

       // Componente B
       *v[X_2][2]+=mo[2][i]*((double)((pcb->Pixels[x-(2-i)][y+1]& 0x00FF0000)>>16));
       *v[X_3][2]+=mo[3][i]*((double)((pcb->Pixels[x-(2-i)][y  ]& 0x00FF0000)>>16));
      }
    }
   else
    {
     if (d==2) // Non c'è Area2
      {
       for(i=a; i<=c; i++) // Calcola i vettori X
        {
         // Componente R
         *v[X_1][0]+=mo[1][i]*((double)(pcr->Pixels[x-(2-i)][y-1] & 0x000000FF));
         *v[X_3][0]+=mo[3][i]*((double)(pcr->Pixels[x-(2-i)][y  ] & 0x000000FF));

         // Componente G
         *v[X_1][1]+=mo[1][i]*((double)((pcg->Pixels[x-(2-i)][y-1]&0x0000FF00)>>8));
         *v[X_3][1]+=mo[3][i]*((double)((pcg->Pixels[x-(2-i)][y  ]&0x0000FF00)>>8));

         // Componente B
         *v[X_1][2]+=mo[1][i]*((double)((pcb->Pixels[x-(2-i)][y-1]&0x00FF0000)>>16));
         *v[X_3][2]+=mo[3][i]*((double)((pcb->Pixels[x-(2-i)][y  ]&0x00FF0000)>>16));
        }
      }
    }
  }
}

//-----------------------------------------------------------------------------
```

```
// Calcola i vettori X1, X2, X3 dell'immagine data dai 3 piani R,G,B; "ms1" e "ms2"
// sono le due maschere Mask1 e Mask2, "(x,y)" sono le coordinate del punto mentre
// vout è il vettore di 4 elementi contenente X1, X2, X3, X4.
void comp_vectors(maschera *ms1,maschera *ms2,int x,int y,
                                                    vect_x *vout)
{
 int a,b,c,d;

 if (x>=2 && x<=w-2)
  {
   if (y>=2 && y<=h-2) { a=0; c=4; b=0; d=4; }
   else
    {
     if (y==0)   { a=0; c=4; b=2; d=4; }
     if (y==1)   { a=0; c=4; b=1; d=4; }
     if (y==h-2) { a=0; c=4; b=0; d=2; }
     if (y==h-1) { a=0; c=4; b=0; d=3; }
    }
  }
 else
  {
   if (y>=2 && y<=h-2)
    {
     if (x==  0) { a=2; c=4; b=0; d=4; }
     if (x==  1) { a=1; c=4; b=0; d=4; }
     if (x==w-2) { a=0; c=3; b=0; d=4; }
     if (x==w-1) { a=0; c=2; b=0; d=4; }
    }
   else
    {
     if (x==  0 && y==  0) { a=2; c=4; b=2; d=4; }
     if (x==  0 && y==  1) { a=1; c=4; b=2; d=4; }
     if (x==  0 && y==h-1) { a=2; c=4; b=0; d=2; }
     if (x==  0 && y==h-2) { a=2; c=4; b=0; d=3; }

     if (x==  1 && y==  0) { a=1; c=4; b=2; d=4; }
     if (x==  1 && y==  1) { a=1; c=4; b=1; d=4; }
     if (x==  1 && y==h-1) { a=1; c=4; b=0; d=2; }
     if (x==  1 && y==h-2) { a=1; c=4; b=0; d=3; }

     if (x==w-1 && y==  0) { a=0; c=2; b=2; d=4; }
     if (x==w-1 && y==  1) { a=0; c=2; b=1; d=4; }
     if (x==w-1 && y==h-2) { a=0; c=2; b=0; d=3; }
     if (x==w-1 && y==h-1) { a=0; c=2; b=0; d=2; }

     if (x==w-2 && y==  0) { a=0; c=3; b=2; d=4; }
     if (x==w-2 && y==  1) { a=0; c=3; b=2; d=3; }
     if (x==w-2 && y==h-2) { a=0; c=3; b=0; d=3; }
     if (x==w-2 && y==h-1) { a=0; c=3; b=0; d=2; }
    }
  }
 calcola_rect_special_a(Form1->ImageR,Form1->ImageG,Form1->ImageB,
                                                        *ms1,*ms2,a,c,b,d,x,y,vout);
}

//---------------------------------------------------------------------------
```

```
// Applica l'algoritmo MSCD
void mscd_algorithm(TImage *pi)
{
 char salva[200];              // Update per il testo nella panelbar
 TCanvas *pc;                  // Puntatore all'immagine
 int i,k;                      // Contatori
 double **c1234,s34,mdm1234,   // Vari coefficienti...
        adm1234,vdm1234;
 vect_x vo;                    // ...e vettore degli X
 maschera mask1,mask2;         // Maschere

 // Crea l'array dei coefficienti
 c1234=new double*[h];
 for(i=0; i<h; i++) c1234[i]=new double[w];

 for(i=0; i<4; i++)
   for(k=0; k<4; k++) vo[k][i]=0.0;
 pc=pi->Picture->Bitmap->Canvas;

 // Imposta la barra
 Form1->ProgressBarR->Max=w*h;
 Form1->ProgressBarR->Min=0;
 Form1->ProgressBarR->Position=0;
 Form1->ProgressBarR->Step=1;
 strcpy(salva,nomefile);
 Form1->StatusBar1->Panels->Items[0]->Text="Applying MSCD Algorithm to image -
                                                          Building C-Matrix...";

 if (dim_cell==3) // Uso il filtro 3x3-BOX
  {
   mask1[0][0]=.0; mask1[0][1]= .0 ; mask1[0][2]= .0 ; mask1[0][3]= .0 ; mask1[0][4]=.0;
   mask1[1][0]=.0; mask1[1][1]=-.33; mask1[1][2]=-.33; mask1[1][3]=-.33; mask1[1][4]=.0;
   mask1[2][0]=.0; mask1[2][1]= .0 ; mask1[2][2]= .0 ; mask1[2][3]= .0 ; mask1[2][4]=.0;
   mask1[3][0]=.0; mask1[3][1]= .33; mask1[3][2]= .33; mask1[3][3]= .33; mask1[3][4]=.0;
   mask1[4][0]=.0; mask1[4][1]= .0 ; mask1[4][2]= .0 ; mask1[4][3]= .0 ; mask1[4][4]=.0;

   mask2[0][0]=.0; mask2[0][1]= .0 ; mask2[0][2]= .0 ; mask2[0][3]= .0 ; mask2[0][4]=.0;
   mask2[1][0]=.0; mask2[1][1]=-.17; mask2[1][2]=-.17; mask2[1][3]=-.17; mask2[1][4]=.0;
   mask2[2][0]=.0; mask2[2][1]= .33; mask2[2][2]= .33; mask2[2][3]= .33; mask2[2][4]=.0;
   mask2[3][0]=.0; mask2[3][1]=-.17; mask2[3][2]=-.17; mask2[3][3]=-.17; mask2[3][4]=.0;
   mask2[4][0]=.0; mask2[4][1]= .0 ; mask2[4][2]= .0 ; mask2[4][3]= .0 ; mask2[4][4]=.0;
  }
 if (dim_cell==5) // Uso un filtro 5x5-SGF
  {
   mask1[0][0]=.0; mask1[0][1]= .0   ; mask1[0][2]= .0025; mask1[0][3]= .0   ; mask1[0][4]=.0;
   mask1[1][0]=.0; mask1[1][1]=-.1067; mask1[1][2]=-.7866; mask1[1][3]=-.1067; mask1[1][4]=.0;
   mask1[2][0]=.0; mask1[2][1]= .0   ; mask1[2][2]= .0   ; mask1[2][3]= .0   ; mask1[2][4]=.0;
   mask1[3][0]=.0; mask1[3][1]= .1067; mask1[3][2]= .7866; mask1[3][3]= .1067; mask1[3][4]=.0;
   mask1[4][0]=.0; mask1[4][1]= .0   ; mask1[4][2]=-.0025; mask1[4][3]= .0   ; mask1[4][4]=.0;

   mask2[0][0]=.0   ;mask2[0][1]= .0   ;mask2[0][2]= .0001;mask2[0][3]= .0   ;mask2[0][4]=.0   ;
   mask2[1][0]=.0   ;mask2[1][1]=-.0531;mask2[1][2]=-.3938;mask2[1][3]=-.0531;mask2[1][4]=.0   ;
   mask2[2][0]=.0002;mask2[2][1]= .1064;mask2[2][2]= .7865;mask2[2][3]= .1064;mask2[2][4]=.0002;
   mask2[3][0]=.0   ;mask2[3][1]=-.0531;mask2[3][2]=-.3938;mask2[3][3]=-.0531;mask2[3][4]=.0   ;
   mask2[4][0]=.0   ;mask2[4][1]= .0   ;mask2[4][2]= .0001;mask2[4][3]= .0   ;mask2[4][4]=.0   ;
  }
 if (dim_cell==-5) // Uso un filtro 5x5-GF
  {
   mask1[0][0]=.0; mask1[0][1]= .0   ; mask1[0][2]= .0001; mask1[0][3]= .0   ; mask1[0][4]=.0;
   mask1[1][0]=.0; mask1[1][1]=-.0043; mask1[1][2]=-.0317; mask1[1][3]=-.0043; mask1[1][4]=.0;
   mask1[2][0]=.0; mask1[2][1]= .0   ; mask1[2][2]= .0   ; mask1[2][3]= .0   ; mask1[2][4]=.0;
   mask1[3][0]=.0; mask1[3][1]= .0043; mask1[3][2]= .0317; mask1[3][3]= .0043; mask1[3][4]=.0;
   mask1[4][0]=.0; mask1[4][1]= .0   ; mask1[4][2]=-.0001; mask1[4][3]= .0   ; mask1[4][4]=.0;

   mask2[0][0]=.0   ;mask2[0][1]= .0   ;mask2[0][2]= .0001;mask2[0][3]= .0   ;mask2[0][4]=.0   ;
   mask2[1][0]=.0   ;mask2[1][1]=-.0108;mask2[1][2]=-.0801;mask2[1][3]=-.0108;mask2[1][4]=.0   ;
   mask2[2][0]=.0002;mask2[2][1]= .0861;mask2[2][2]= .6366;mask2[2][3]= .0861;mask2[2][4]=.0002;
   mask2[3][0]=.0   ;mask2[3][1]=-.0108;mask2[3][2]=-.0801;mask2[3][3]=-.0108;mask2[3][4]=.0   ;
   mask2[4][0]=.0   ;mask2[4][1]= .0   ;mask2[4][2]= .0001;mask2[4][3]= .0   ;mask2[4][4]=.0   ;
  }
```

(continua…)

```
 // Calcola tutti i coefficienti C1234
 for(i=0; i<h; i++)
  {
   for(k=0; k<w; k++)
    {
     comp_vectors(&mask1,&mask2,k,i,&vo);
     vo[X_4][0]=vo[X_1][0]*0.5+vo[X_2][0]*0.5;
     vo[X_4][1]=vo[X_1][1]*0.5+vo[X_2][1]*0.5;
     vo[X_4][2]=vo[X_1][2]*0.5+vo[X_2][2]*0.5;

     // Calcolo C34
     s34=maggiore_di(vo[X_3],vo[X_4]);

     // Calcolo C1234
     mdm1234=sqrt(pow(comp_mdm(vo[X_1],vo[X_2]),2)+pow(comp_mdm(vo[X_3],vo[X_4]),2));
     adm1234=sqrt(pow(comp_adm(vo[X_1],vo[X_2]),2)+pow(comp_adm(vo[X_3],vo[X_4]),2));
     vdm1234=(double)mdm1234*adm1234;
     c1234[i][k]=(double)(s34*(1.0-vdm1234));
    }
   Form1->ProgressBarR->StepBy(w);
  }

 // Imposta la barra
 Form1->ProgressBarR->Position=0;
 Form1->ProgressBarR->Min=0;
 Form1->StatusBar1->Panels->Items[0]->Text="Applying MSCD Algorithm to image - Finding
                                                                              Edges...";

 // Trova i bordi
 for(i=0; i<h; i++)
  {
   for(k=0; k<w; k++)
    {
     if (c1234[i][k]<0.0) pc->Pixels[k][i]=(TColor)0x00FFFFFF;
     else
      {
       if (controlla_pixel(c1234,k,i)==true) pc->Pixels[k][i]=(TColor)0x00FFFFFF;
       else                                  pc->Pixels[k][i]=(TColor)0x00000000;
      }
    }
   Form1->ProgressBarR->StepBy(w);
  }

 // Libera l'array
 for(i=0; i<h; i++) delete c1234[i];
 delete c1234;

 Form1->StatusBar1->Panels->Items[0]->Text=salva;
}
```