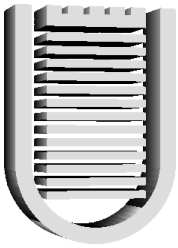
UNIVERSITÀ DEGLI STUDI DI ROMA
"TOR VERGATA"



FACOLTÀ DI INGEGNERIA

Dipartimento di Informatica, Sistemi e Produzione

Tesi di Laurea Specialistica in Ingegneria Informatica

RILEVAMENTO DI CONTORNI A PIÙ SCALE E DI CORPI IN SEQUENZE VIDEO

Relatore:
Ing. Francesco Martinelli

Laureando:
Emanuele Rodolà

Anno Accademico 2006/2007

I miei più sentiti ringraziamenti vanno al Prof. Tauraso per la straordinaria competenza, disponibilità ed entusiasmo, e al Prof. Martinelli per i consigli, la disponibilità e l'attenzione dimostrate

Indice

Introduzione

In questa tesi vengono descritti il progetto e la realizzazione software di due sistemi di visione: il primo consiste in un rilevatore di contorni per immagini, basato su risultati nell'ambito della fisiologia dell'occhio umano, e frutto di ricerca recente [1]; il secondo è un sistema di riconoscimento di corpi all'interno di sequenze video, basato esclusivamente su informazioni contenute nel moto stesso degli oggetti.

Il rilevamento di contorni è un'operazione fondamentale nel campo dell'*image processing* e della visione artificiale, ed è da sempre un fertile campo di ricerca. In letteratura sono stati proposti diversi rilevatori di bordi; tuttavia, questi agiscono reagendo a tutti i cambiamenti di luminosità nell'immagine oltre una certa soglia, indipendentemente dalla loro origine - che si trattino cioè di contorni appartenenti a oggetti o a piccoli dettagli. In questo lavoro di tesi viene affrontato il problema dell'estrazione dei contorni seguendo un approccio detto *non-contestuale*, che consente di isolare gli oggetti in una scena; per fare ciò, il rilevatore realizzato implementa una tecnica di *inibizione* dei dettagli motivata biologicamente, che fa uso delle informazioni sui contorni presenti a risoluzioni diverse dell'immagine di partenza. Il sistema di estrazione consente di ottenere ottimi risultati sia in termini di contorni rilevati che di dettagli soppressi; inoltre agisce meglio di quelli tradizionali (rilevatori di Canny, di Sobel, di Prewitt, ecc.) anche in presenza di rumore, e la realizzazione a monte di una fase di riduzione del rumore (al momento non presente) gioverebbe senza dubbio ulteriormente. Per questo motivo, il rilevatore in multiscala motivato biologicamente è da considerarsi senza dubbio tra quelli allo stato dell'arte.

Il rilevamento di persone all'interno di folle dense ha ricevuto relativamente poca attenzione nell'ambito della visione artificiale in quanto è, allo stesso tempo, un problema di segmentazione, riconoscimento e *tracking*. L'analisi automatica di folle e flussi di traffico è stata affrontata negli anni, sostanzialmente, con i medesimi modelli utilizzati per immagini statiche e con poche entità individuali. Queste tecniche sono diversamente efficaci e si basano sull'aspetto delle entità da riconoscere. In questa tesi viene ipotizzato che, all'interno di sequenze video, i moti indipendenti delle entità offrono già di loro un'ottima inizializzazione. Allo stato dell'arte, i sistemi che offrono risultati migliori (in termini di numero di entità correttamente tracciate) fanno uso di tecniche di modellazione dello sfondo per poter distinguere gli oggetti che si muovono all'interno di una scena dall'ambiente stesso. Gli oggetti (tipicamente persone) vengono dunque modellati con ellissoidi per ogni parte del corpo, oppure con delle *bounding box* che racchiudono ogni singola entità tracciata. Altri sistemi fanno uso di semplici modelli per parti del corpo che vengono ricercate all'interno della scena. Una volta individuate le parti, vengono imposti dei vincoli cinematici (e di altro tipo) su di esse, e vengono scartati gli "accorpamenti" meno probabili. Tuttavia, in ogni caso è noto il tipo di oggetto da riconoscere (indipendentemente dal fatto che

vengano utilizzati modelli o no - ad esempio basandosi semplicemente su alcune proprietà morfologiche). Il sistema realizzato in questa tesi è non supervisionato e basa il riconoscimento stesso sul solo suggerimento del moto: l'idea che sottintende questo ragionamento è il fatto che due feature vengano considerate parte dello stesso corpo solo se "abbastanza" vicine e se si spostano insieme rigidamente. Da questa idea è stata definita una misura di probabilità di moto coerente, ed è stato sviluppato un sistema per il riconoscimento di entità semoventi in sequenze video. Il vantaggio principale di questo approccio è che non fa uso di alcuna informazione sull'aspetto delle entità da tracciare. Gli esperimenti condotti con il software realizzato mostrano che il metodo dà ottimi risultati in termini di efficacia, risultati ulteriormente migliorabili se venisse integrata l'informazione sull'aspetto delle entità. Dunque, un rilevatore siffatto può agire da inizializzatore per un ampio spettro di applicazioni. Questo rilevatore di corpi è stato realizzato seguendo due approcci diversi; in entrambi i casi, i tempi di calcolo sono piuttosto alti anche per video brevi (comunque dipendentemente dalla complessità della scena e dai parametri impostati nel rilevatore). In conclusione, un sistema di questo tipo mal si presta a una ottimizzazione per il real-time.

La tesi è suddivisa in cinque capitoli. Nel capitolo 1 vengono dati i preliminari matematici necessari per una comprensione adeguata delle tecniche e dei metodi adottati. In particolare viene discussa la teoria delle *wavelet* e dell'*analisi multirisoluzione*, strumenti di analisi utilizzati in molte delle tecniche che si vedranno. Il capitolo 2 presenta alcuni metodi per il rilevamento di contorni in immagini a due dimensioni: vengono prima esposti sommariamente i rilevatori classici, introducendo il ben noto criterio di Canny; dopodiché vengono mostrate alcune tecniche per il rilevamento basate sull'utilizzo di wavelet e di spazi multiscala; infine, viene presentato un sistema basato sulla soppressione di *texture* e motivato da risultati sulla fisiologia umana, sistema di cui si è inoltre data un'implementazione C++. Di questo sistema vengono realizzate due varianti: una versione a risoluzione fissa, e una seconda che trae vantaggio da una rappresentazione in multiscala. Nel capitolo 3 si affronta il problema della *registrazione traslazionale* tra fotogrammi, che consiste in una forma particolare di *image matching* quando le immagini di riferimento sono prese in successione da una sequenza video. Poi viene mostrato un algoritmo veloce per il rilevamento di feature che comprende una fase di apprendimento. A partire dai risultati forniti da queste argomentazioni preliminari, vengono mostrate due tecniche originali per il rilevamento di corpi semoventi in sequenze video. Infine, vengono proposte alcune tecniche ibride per il rilevamento di corpi, che traggono beneficio dai risultati del capitolo 2. Il capitolo 4 è dedicato all'architettura e alla progettazione del software realizzato. Le applicazioni sviluppate sono indipendenti tra di loro e sono: 1) Un'applicazione per il rilevamento di contorni, basata sul capitolo 2.3. 2) Un'applicazione per il rilevamento di corpi in sequenze video, basata sul capitolo 3.3. 3) Un'applicazione per il rilevamento di feature in real-time, basata sul capitolo 3.2.1. Il capitolo 5 mostra i risultati di esperimenti effettuati con le applicazioni

realizzate; vengono dati tempi di calcolo, valutazioni qualitative e vengono fatte considerazioni sulla scelta giusta dei parametri e sui rapporti di efficienza dei sistemi. Infine, nel capitolo 6 vengono proposti sviluppi futuri e vengono date le dovute conclusioni.

Capitolo 1: Nozioni preliminari

In questo capitolo vengono rivisti alcuni importanti concetti matematici; dopodiché vengono introdotte la teoria delle wavelet, le trasformate e loro proprietà principali, e le applicazioni in contesti bidimensionali, d'interesse per lo sviluppo dei capitoli successivi. Di fatti, l'elaborazione di segnali in multiscala sta alla base di molte delle tecniche utilizzate per la progettazione e la realizzazione dei sistemi argomento di questa tesi.

Per ulteriori approfondimenti sui concetti esposti in questo capitolo, si rimanda a [7].

1.1 Analisi in tempo-frequenza

L'analisi di Fourier non è molto adatta quando si intende calcolare *localmente* i contenuti in frequenza di un segnale; inoltre, lo spettro di Fourier non fornisce nessuna informazione sulla localizzazione temporale delle frequenze.

In fig. 1 e fig. 2 vengono mostrati rispettivamente un segnale perturbato e il suo spettro di ampiezza.

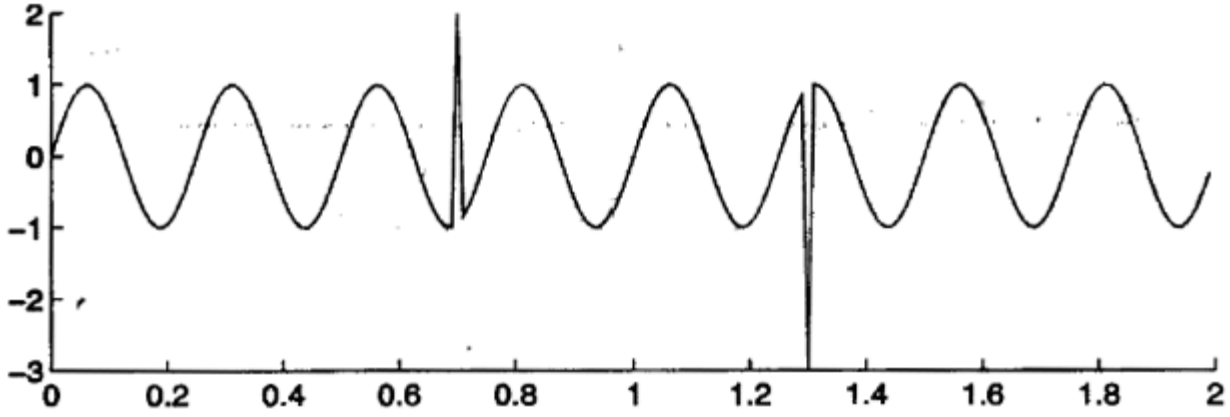


Fig. 1: Segnale sinusoidale perturbato.

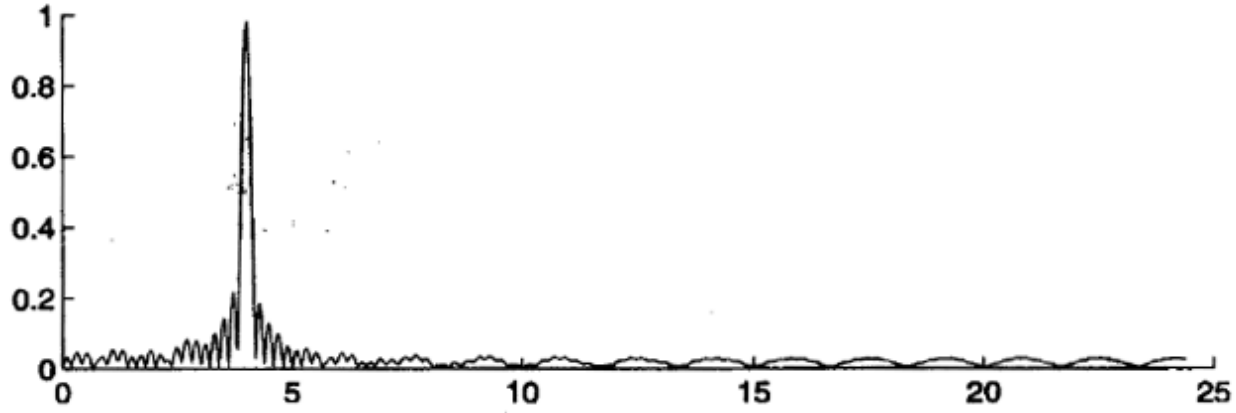


Fig. 2: Spettro di ampiezza del segnale perturbato.

Il segnale in esame è una semplice sinusoide con frequenza 4 Hz con delle perturbazioni di tipo impulsivo in due istanti di tempo. Dallo spettro di ampiezza si osserva un lobo centrato sulla frequenza 4 Hz. Questo lobo è presente al posto di una delta per via del fatto che la sinusoide analizzata non è infinita bensì troncata; inoltre, ai lati del lobo sono presenti delle perturbazioni dovute ai disturbi impulsivi, ma è evidente che tramite una rappresentazione siffatta non è possibile localizzare nel tempo questi disturbi.

In effetti, l'uso della trasformata di Fourier per la rappresentazione di caratteristiche locali di un segnale può risultare poco conveniente; ciò è dovuto al fatto che la trasformata consiste nel proiettare un segnale su una base di funzioni *globali* $e^{j\omega t}$ (nel senso che si estendono su tutto l'asse dei tempi). Si pensi ad esempio alla delta di Dirac: nella rappresentazione di Fourier, infinite armoniche devono sommarsi costruttivamente in $t=0$ e distruttivamente in tutti gli altri tempi: seni e coseni non sono adatti ad approssimare picchi e irregolarità improvvise!

In altre parole, il segnale $X(\omega)$, ottenuto come la trasformata di Fourier di un segnale $x(t)$, è *perfettamente locale in frequenza* ma *perfettamente globale nel tempo*. Vale a dire, il segnale non discrimina eventi che avvengono nel dominio del tempo, e quindi è una trasformata adatta ai soli segnali stazionari (segnali i cui contenuti in frequenza non cambiano nel tempo, ad esempio toni sostenuti).

In realtà, gran parte dei segnali fisici è di tipo non stazionario.

È chiaro dunque che servono nuovi strumenti per l'analisi locale e per l'analisi in *tempo-frequenza*.

Un primo approccio, piuttosto intuitivo, consiste nel *finestrare* la porzione di interesse del

segnale e applicare la trasformata di Fourier: questo significa selezionare tratti abbastanza corti da poterli trattare come stazionari. Occorre quindi inserire una dipendenza dal tempo nella trasformazione, rendendola locale operando su porzioni di $x(t)$. Questo metodo prende il nome di *Short Time Fourier Transform* (*STFT*), in letteratura noto anche come *Windowed Fourier Transform* (*WFT*).

Una *finestra* $\phi(t)$ è una funzione che vale zero al di fuori di un certo intervallo. Tipicamente (e didatticamente) viene presa in considerazione la finestra di tipo *rect*, così definita:

$rect(t/T)=1 \quad se \quad -T/2 \le t \le T/2$
$rect(t/T)=0 \quad altrimenti$

I suoi parametri caratteristici sono la posizione del centro t^* e la larghezza (o diametro) Δ_ϕ . Questi due parametri sono facilmente ottenibili da una coppia di equazioni [7]. Allo stesso modo, si può avere una finestra in frequenza $\hat{\phi}(\omega)$, con parametri ω^* e $\Delta_{\hat{\phi}}$; ad esempio, nel caso limite di una sinusoide (Fourier), si ha raggio infinito nel tempo e nullo in frequenza.

Vogliamo dunque finestrare una funzione attorno a un istante b e ottenere $f_b(t)=f(t)\phi(t-b)$, per poi trasformala in Fourier. La STFT di un segnale $f(t)$ è $G_\phi f(b,\xi)$, ed è così definita:

$$G_\phi f(b,\xi)=\int f(t)\overline{\phi_{b,\xi}(t)}\,dt$$

Dove la finestra ϕ è valutata nel punto (b,ξ) nel piano tempo-frequenza. La finestra genera una famiglia di finestre tramite traslazioni in tempo e frequenza:

$$\phi_{b,\xi}(t)=\phi(t-b)\,e^{j\xi t} \qquad \text{famiglia di finestre}$$

La trasformata è lineare e $G_\phi f(b,\xi)$ dà informazioni su $f(t)$ nella finestra temporale:

$$[t^*+b-\Delta_\phi,\ \ t^*+b+\Delta_\phi]$$

mentre in frequenza la finestra è nell'intervallo:

$$[\omega^*+\xi-\Delta_{\hat{\phi}},\ \ \omega^*+\xi+\Delta_{\hat{\phi}}]$$

Si noti che il prodotto $f(t)\phi(t-b)$ equivale a una convoluzione in frequenza, quindi lo

spettro di f nell'intorno di b risulta alterato dalla presenza della finestra.

La finestra in tempo-frequenza è dunque

$$[t^* + b - \Delta_\phi,\ t^* + b + \Delta_\phi] \times [\omega^* + \xi - \Delta_{\hat{\phi}},\ \omega^* + \xi + \Delta_{\hat{\phi}}]$$

La $\phi(t)$ può essere una funzione complessa e deve soddisfare la condizione:

$$\hat{\phi}(0) = \int \phi(t)\, dt \neq 0$$

In altre parole la finestra della STFT si comporta come un filtro passa-basso. In generale le proprietà della trasformata dipendono dalla scelta di $\phi(t)$, che potrà essere, ad esempio, una finestra di Hamming, di Blackman o altre dipendentemente dalle caratteristiche desiderate nel determinato dominio applicativo.

Ora è possibile apprezzare la differenza che c'è tra la STFT e la trasformata di Fourier. Nel secondo caso, $f(t)$ deve essere conosciuta nell'intero asse dei tempi prima di poter calcolare le componenti spettrali; la STFT invece necessita solo della porzione di segnale in cui la finestra è diversa da zero (*supporto* della finestra), cioè $G_\phi f(b, \xi)$ restituisce approssimativamente lo spettro di f vicino a $t = b$.

Quindi possiamo scrivere:

$$G_\phi f(b, \xi) = \langle f(t),\ \phi(t - b) e^{j \xi t} \rangle$$

La STFT di un segnale consiste cioè dei componenti dello stesso rispetto a questa nuova base nel piano tempo-frequenza.

1.1.1 Copertura del piano tempo-frequenza

La localizzazione in tempo e frequenza fornita dalla STFT è limitata da alcuni importanti risultati. Come visto, fissata una finestra $\phi(t)$ sono definite le indeterminazioni Δ_ϕ e $\Delta_{\hat{\phi}}$: due eventi distanti meno di Δ_ϕ o $\Delta_{\hat{\phi}}$ non sono più discriminati dalla STFT nel rispettivo dominio (e in questo senso si parla di *risoluzione*: una risoluzione bassa corrisponde a una discriminazione grossolana mentre una risoluzione alta corrisponde a una discriminazione precisa). I due raggi sono inversamente proporzionali: per un principio di indeterminazione analogo a quello di Heisenberg della fisica quantistica non è possibile infatti stimare con precisione arbitraria e simultaneamente i parametri temporali e frequenziali di un segnale.

Nell'ambito dei segnali il principio è noto come *principio di Heisenberg-Gabor*:

$$\Delta_\phi \Delta_{\hat{\phi}} \geq \frac{1}{2}$$

Il secondo risultato ha a che fare con il supporto del segnale nel tempo e della corrispettiva funzione in frequenza, e risiede nel fatto che, se $f(t)$ è un segnale a supporto compatto, la sua trasformata di Fourier non può essere nulla su un intero intervallo, e viceversa. Quindi è impossibile avere una funzione in L^2 che sia a supporto compatto nel tempo e nelle frequenze.

In particolare, questo significa che *non esiste un'analisi istantanea in frequenza* per segnali a energia finita. Da questi due fatti si evince che la localizzazione in tempo-frequenza è ottenibile solo su intervalli. Una localizzazione di questo tipo viene rappresentata con le cosiddette *celle di fase*, o *caselle di Heisenberg* (fig. 3).

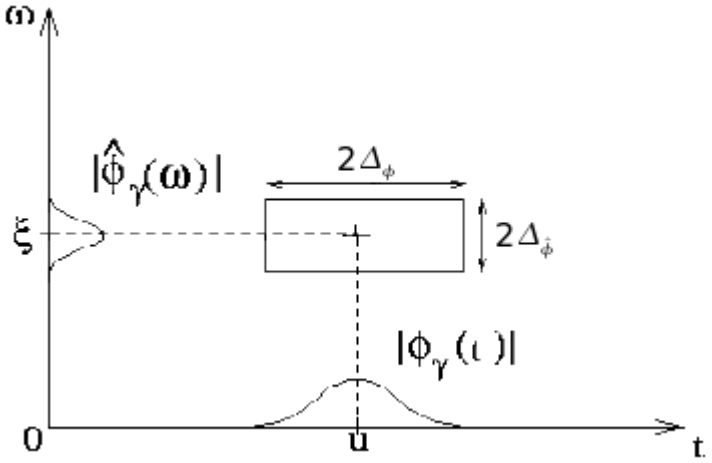


Fig. 3: Finestra in tempo-frequenza centrata in (u, ξ) *, rappresentata con una cella di fase.*

Il principio di indeterminazione impone un limite inferiore all'area coperta dalla finestra. Visualizzando la copertura delle finestre nella trasformata di Fourier sul piano tempo-frequenza, osserviamo una localizzazione ottimale in frequenza ma nulla nel tempo (fig. 4). Al contrario, la STFT fornisce una certa localizzazione nel tempo e in frequenza; il principale svantaggio risiede però nel fatto che i raggi Δ_ϕ e $\Delta_{\hat{\phi}}$ non dipendono dalla posizione della finestra sul piano: una volta scelta la funzione ϕ, la risoluzione in tempo-frequenza è fissa lungo l'intero processing. Il piano è ricoperto come in fig. 5.

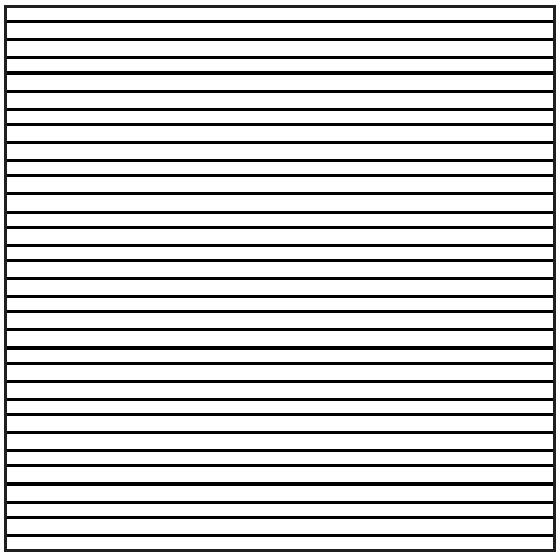


Fig. 4: Copertura del piano tempo-frequenza nella trasformata di Fourier: nel tempo (ascisse) la finestra ha raggio infinito mentre in frequenza (ordinate) la finestra ha raggio infinitesimo.

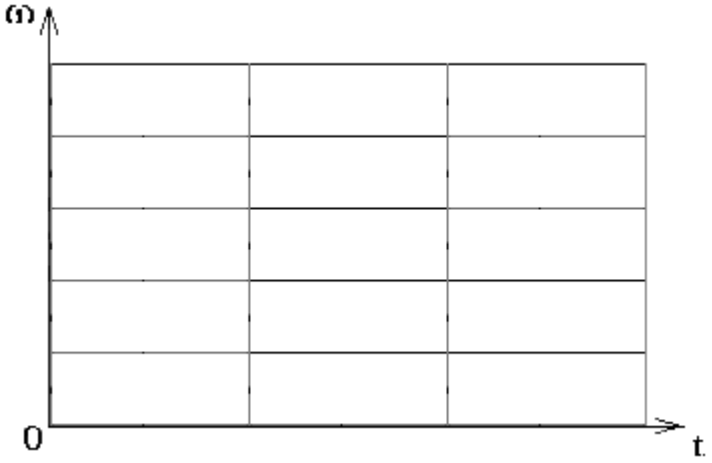


Fig. 5: Copertura del piano tempo-frequenza della STFT, la risoluzione è fissa una volta scelta la finestra.

Questo fatto ha molte implicazioni. Si consideri a titolo di esempio un segnale di tipo *chirp*:

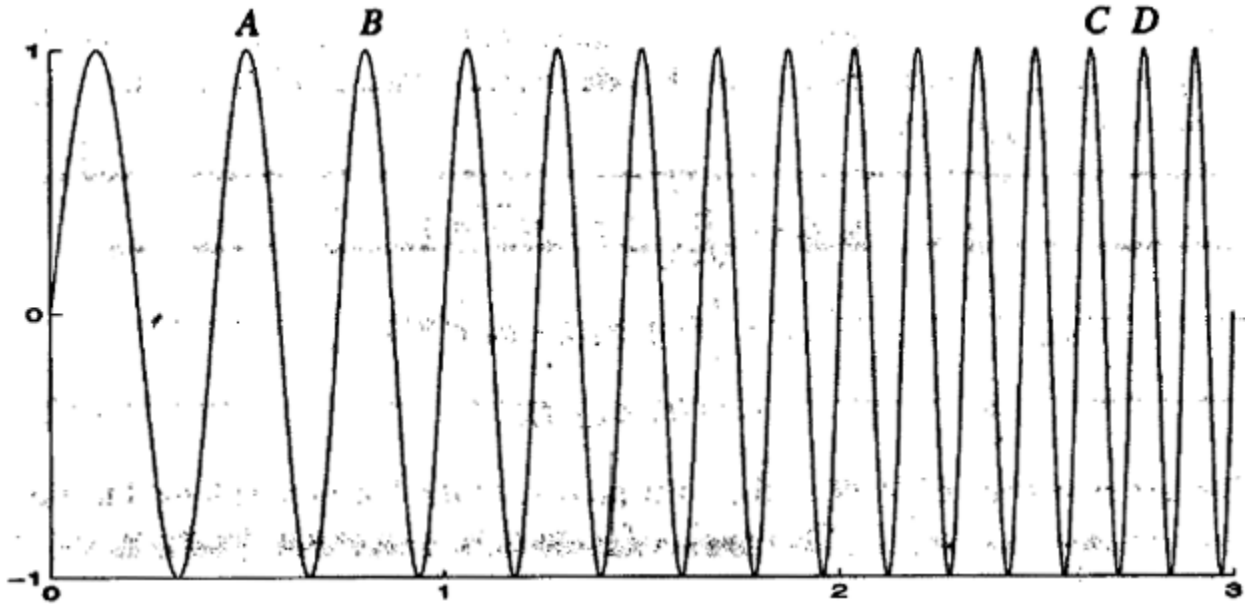


Fig. 6: Segnale di tipo chirp.

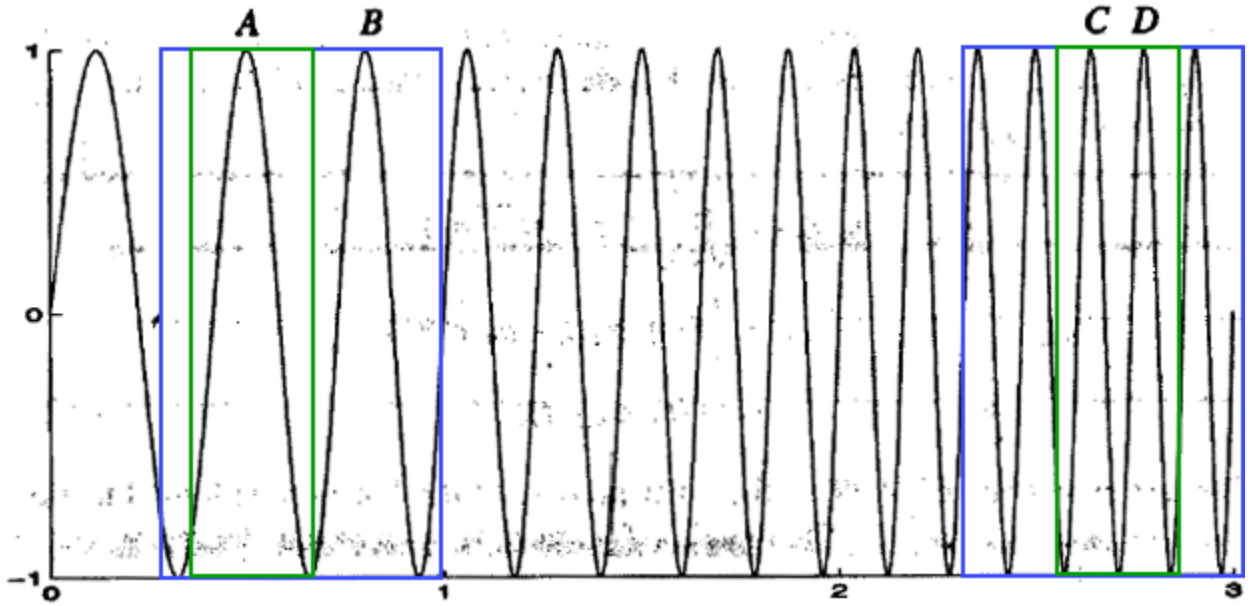


Fig. 7: Segnale chirp finestrato da una finestra larga (in blu) e una stretta (in verde).

Un segnale chirp è un tipo di segnale non-stazionario la cui frequenza aumenta nel tempo (secondo una legge lineare, quadratica o quant'altro). Scegliendo i parametri della finestra temporale $\phi(t)$ (ad esempio il parametro α nel caso di finestra Gaussiana) di modo tale

che Δ_ϕ sia circa uguale alla lunghezza AB , la STFT potrà risolvere meglio le basse frequenze di quelle alte; scegliendo invece Δ_ϕ circa pari a CD , le frequenze basse non saranno risolte correttamente. Quindi, con una finestra temporale ampia si ha una bassa risoluzione nel tempo e alcuni valori del segnale non possono essere risolti; tuttavia la risoluzione in frequenza è molto buona quindi le diverse frequenze sono ben distinguibili nel piano. Diminuendo l'ampiezza della finestra temporale, la risoluzione in frequenza peggiora ma la risoluzione nel tempo migliora.

Di seguito viene mostrato un segnale di esempio, la sua trasformata di Fourier, e il risultato dell'applicazione della STFT con tre finestre diverse ma di area uguale. La prima finestra è larga nel tempo e stretta in frequenza, la seconda finestra è stretta nel tempo ma larga in frequenza, la terza finestra ha $\Delta_\phi = \Delta_{\hat{\phi}}$. Il segnale è costruito come la somma di una sinusoide a 35 Hz, una chirp quadratica che inizia all'istante 0 con una frequenza di 25 Hz e termina dopo un secondo alla frequenza di 140 Hz, e un impulso di breve durata a circa 0.3 secondi.

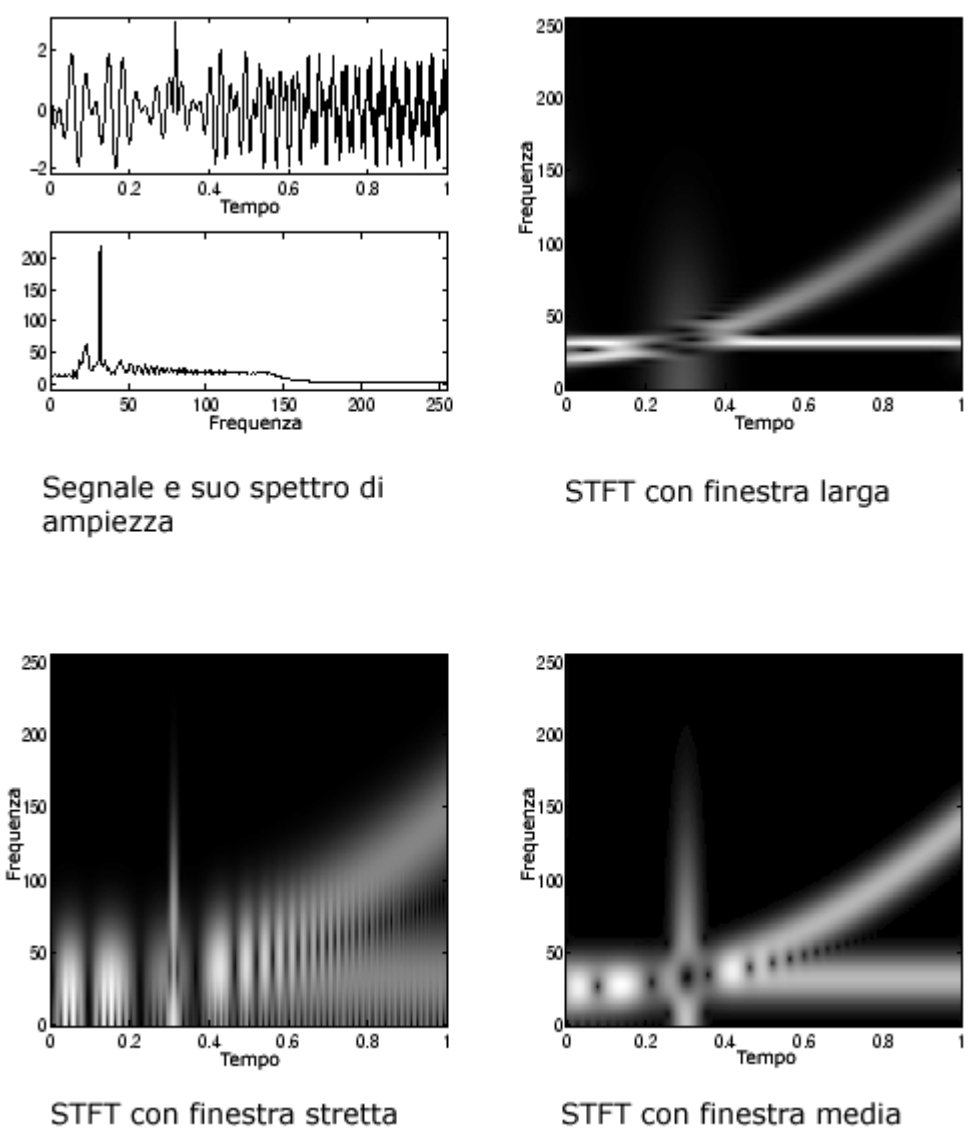


Fig. 8: Segnale di esempio e trasformata STFT con tre finestre diverse. Si nota che la finestra larga produce una buona risoluzione in frequenza ma bassa risoluzione nel tempo; la finestra stretta al contrario offre un'ottima risoluzione nel tempo ma piuttosto scarsa in frequenza; la finestra media offre risoluzioni decenti sia nel tempo che nella frequenza.

1.2 Analisi in tempo-scala

La STFT è solo uno dei tanti modi per effettuare l'analisi in tempo-frequenza dei segnali. Un'altra trasformata lineare utile per questo scopo è la trasformata wavelet continua (*CWT*, in letteratura nota anche come *IWT – Integral Wavelet Transform*).

Si è visto che la STFT fornisce una risoluzione fissa nel piano tempo-frequenza una volta scelta la finestra. Un obiettivo ideale è quello di ottenere una buona risoluzione in tempo-frequenza in posizione *arbitraria*. Per le applicazioni pratiche, è desiderabile una finestra che abbia Δ_ϕ crescente per le frequenze basse (per Heisenberg-Gabor ciò equivale a un raggio $\Delta_{\hat{\phi}}$ decrescente, perciò una buona risoluzione in frequenza) e Δ_ϕ decrescente spostandosi sulle frequenze alte (buona risoluzione nel tempo). In questo modo è possibile discriminare correttamente le discontinuità, e al contempo catturare le informazioni "di massima" del segnale, cioè quelle in bassa frequenza.

1.2.1 Trasformata Wavelet Continua

Per un segnale in L^2, la CWT è così definita:

$$W_\psi f(b,a) = \int_{-\infty}^{\infty} f(t)\,\overline{\psi_{b,a}(t)}\,dt \quad \text{, con} \quad \psi_{b,a}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right), \quad a>0$$

I parametri b ed a sono detti rispettivamente *parametro di traslazione* e *parametro di dilatazione o scala*. Il fattore $1/\sqrt{a}$ è un fattore di normalizzazione per avere $\|\psi_{b,a}\| = \|\psi\|$, cioè di modo tale che la funzione abbia la stessa energia a ogni livello di scala.

La funzione $\psi(t)$ è chiamata *wavelet madre*, o *di analisi*, e variandone i parametri di traslazione e scala si ottiene una famiglia di wavelet, proprio come nella STFT si otteneva una famiglia di finestre al variare dei parametri di traslazione in tempo e frequenza. La CWT si calcola dunque *traslando continuamente* una wavelet *scalabile continuamente* sul segnale in analisi, e calcolando la correlazione tra i due.

La $\psi(t)$ soddisfa la condizione:

$$\hat{\psi}(0) = \int \psi(t)\,dt = 0$$

cioè si comporta come un passa-banda e oscilla nel tempo (questa condizione è implicata dalla *condizione di ammissibilità*, vedi sez. 1.2.2), e si dice che la wavelet ha *momenti svanenti* di ordine m se

$$\int t^p \psi(t)\,dt = 0 \quad p = 0,\ldots,m-1$$

Spesso la wavelet viene ottenuta come derivata di una *funzione di smoothing* $\phi(t)$ (una funzione derivabile molte volte e che decade velocemente) in L^2. Se la funzione di smoothing ha n derivate continue e $\psi_p(t)$ è la derivata p-esima, allora $\psi_p(t)$ ha momenti svanenti di ordine p. In termini matematici:

$$\psi_p(t)=\frac{d^p\phi(t)}{dt^p},\quad p<n \;\Rightarrow\; \int_{-\infty}^{\infty} t^{p-1}\psi_p(t)\,dt=0$$

L'importanza della proprietà dei momenti svanenti verrà data, in maniera intuitiva, nel seguito.

Comportandosi come un passa-banda, la $\hat{\psi}(\omega)$ ha due centri e due raggi, ma per le applicazioni pratiche ci si restringe alle sole frequenze positive. Il principio di indeterminazione per le wavelet è nella forma:

$$\Delta_\psi \Delta_{\hat{\psi}}^{+} > \frac{1}{2}$$

E la finestra temporale è la seguente:

$$[at^*+b-a\,\Delta_\psi,\;\; at^*+b+a\,\Delta_\psi]$$

dove si vede che il raggio di $\psi(t)$ è espanso proporzionalmente al parametro di scala, e il centro di $\psi_{b,a}(t)$ è at^*+b, con t^* centro di ψ. Il diametro temporale è $2a\,\Delta_\psi$.
La finestra in frequenza è la seguente [7]:

$$[\frac{1}{a}(\omega_+^*-\Delta_{\hat{\psi}}^{+}),\;\; \frac{1}{a}(\omega_+^*+\Delta_{\hat{\psi}}^{+})]$$

quindi il diametro in frequenza è $\frac{2}{a}\Delta_{\hat{\psi}}^{+}$.

Si noti a questo punto che la CWT non fornisce un'analisi in tempo-frequenza ma in *tempo-scala*. D'altronde, si osserva che $1/a$ è una misura della frequenza perché riducendo a aumenta l'intervallo in frequenza. La rappresentazione in tempo-frequenza è dunque piuttosto indiretta, ma quantomeno intuitiva. Dalla definizione di $\psi_{b,a}(t)$ si vede che a scale basse (a piccole) la wavelet è una versione compressa della wavelet madre, e corrisponde alle frequenze alte; similmente, le scale alte (frequenze basse) rappresentano segnali che cambiano poco nel tempo, e le wavelet usate per l'approssimazione sono versioni dilatate della wavelet madre.

Ogni wavelet ha una relazione tra scala e frequenza, ad esempio per la wavelet di Morlet si

ha:

$$\psi_{morlet}(t) = e^{-\frac{t^2}{2}} \cos(5t)$$
$$frequency \approx \frac{5/2\pi}{scale}$$

In fig. 9 è mostrata la wavelet di Morlet e la sua rappresentazione in frequenza; si può notare come, dilatando la wavelet nel tempo, in frequenza si ottiene un restringimento (della metà) della banda passante: la wavelet diventa maggiormente selettiva sulle fluttuazioni.

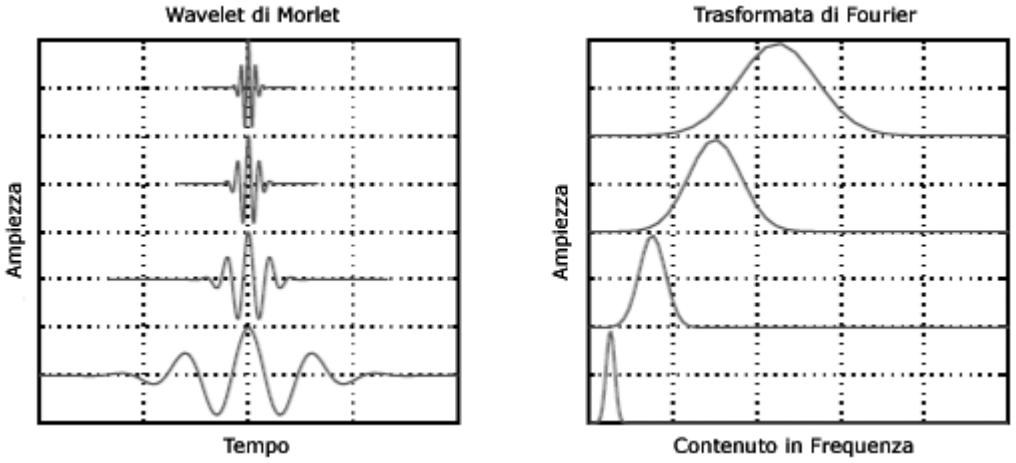


Fig. 9: Wavelet di Morlet a diverse scale e loro rappresentazione in frequenza.

A ogni scala dunque corrisponde un *intervallo* di frequenze.

Come visto, la finestra determinata dalla ψ ha un diametro temporale pari a $2a\,\Delta_\psi$ e un diametro in frequenza pari a $\frac{2}{a}\Delta_{\hat{\psi}}^{+}$. Per Heisenberg-Gabor si ha:

$$2a\,\Delta_\psi \frac{2}{a}\Delta_{\hat{\psi}}^{+} = 4\,\Delta_\psi \Delta_{\hat{\psi}}^{+} = cost.$$

Dalla fig. 10 si osserva che per frequenze alte ($1/a_2$ grande) la finestra in frequenza è ampia mentre la finestra nel tempo è piccola, e per frequenze basse vale il viceversa. Per un valore fissato di $1/a_0$ la finestra in tempo-frequenza è fissa, mentre con la STFT si aveva una finestra fissa indipendentemente dalla frequenza. Abbiamo cioè ottenuto una buona risoluzione in frequenza per ω basse (Δ_ψ crescente) e una buona risoluzione nel tempo per ω alte (Δ_ψ decrescente): la CWT si presta bene a rilevare anomalie di breve durata nei segnali, così come caratteristiche di lunga durata. Nelle applicazioni pratiche, si incontrano spesso segnali di

questo tipo. Ciò non significa chiaramente che questa trasformata possa rimpiazzare in tutti i casi la STFT o la trasformata di Fourier: se si deve analizzare, ad esempio, un tono musicale sostenuto ad alta frequenza, la trasformata wavelet non è molto adatta.

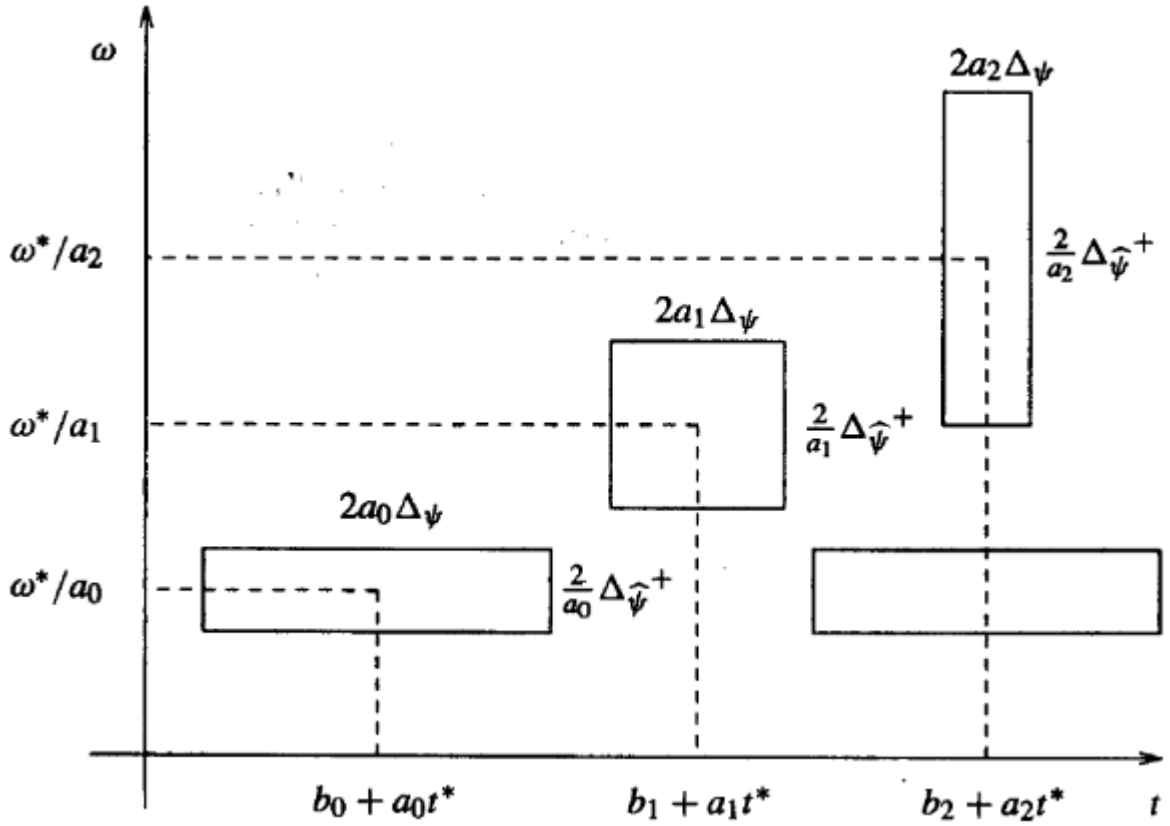


Fig. 10: Finestra di tipo wavelet. All'aumentare della frequenza (scale basse) la risoluzione nel tempo aumenta, mentre la risoluzione in frequenza diminuisce. Si ottiene quindi buona risoluzione in frequenza per gli eventi di lunga durata e buona risoluzione nel tempo per eventi di breve durata, come oscillazioni rapide e spike.

1.2.2 Trasformata inversa

La CWT ha una trasformata inversa univoca. Per ottenere il segnale originale a partire dalla sua CWT è necessario integrare sul piano definito da (b,a) .

$$f(t)=\frac{1}{C_\psi}\int_{-\infty}^{\infty} db \int_{-\infty}^{\infty} \frac{1}{a^2}[W_\psi f(b,a)]\psi_{b,a}(t)\,da$$

dove C_ψ è una costante che dipende dalla wavelet scelta:

$$C_\psi = \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega < \infty$$

Quest'ultima disuguaglianza è detta *condizione di ammissibilità*, e riduce la classe delle wavelet. L'inversa esiste solo se la condizione è soddisfatta; si noti che la condizione implica che $\hat{\psi}(0) = 0$.

Nell'anti-trasformata, l'integrale interno somma tutte le componenti della wavelet nella posizione b , e l'integrale esterno include tutte le locazioni sull'asse b . Più avanti si vedranno dei metodi di inversione della trasformata che più si prestano alla sintesi di segnali nelle applicazioni pratiche.

1.2.3 Lo zoo delle wavelet

Le wavelet sono infinite. Spesso le wavelet note non si adattano a tutti gli obiettivi quindi si rende necessaria la costruzione di wavelet ad-hoc (wavelet *adattate*) per una determinata applicazione. Tuttavia, la realizzazione di wavelet che soddisfino specifiche proprietà richiede l'uso della matematica in maniera piuttosto sofisticata, e non troverà qui una trattazione.

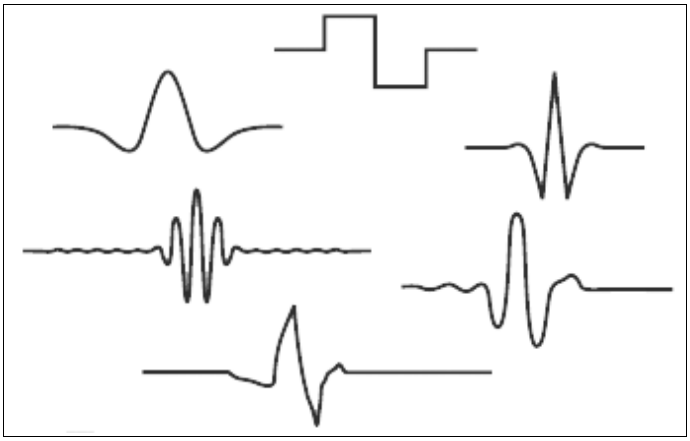


Fig. 11: Alcune wavelet, tra cui si distinguono Haar (in alto) e Daubechies (in basso).

1.3 Analisi Multi-Risoluzione

Nell'analisi multi-risoluzione (o *MRA*), una funzione è vista a diversi livelli di approssimazione o *risoluzioni*. L'idea è stata sviluppata da Meyer e Mallat. Applicando l'MRA è possibile dividere una funzione complicata in molte funzioni semplici e studiare queste funzioni separatamente.

Per ottenere un'analisi di questo tipo è necessario avere una funzione $\phi(t) \in L^2(\mathbb{R})$, chiamata *funzione di scaling* o *wavelet padre*, che generi una sequenza innestata $\{V_j\}$ di spazi:

$$\{0\} \subset ... \subset V_{-1} \subset V_0 \subset V_1 \subset ... \subset V_j \subset V_{j+1} \subset ... \subset L^2(\mathbb{R})$$

In una MRA i seguenti assiomi devono essere soddisfatti:

$\cup_{j=-\infty}^{\infty} V_j = L^2(\mathbb{R})$ *assioma di completezza*

e si dice anche che l'unione è *densa* in L^2; secondo questo assioma, a risoluzione infinita tutti i segnali di energia sono ricostruiti perfettamente.

Deve anche essere:

$\cap_{j=-\infty}^{\infty} V_j = \{\emptyset\}$ *assioma di separazione*

e cioè a risoluzione nulla, l'unico segnale di energia è il segnale costante zero.

La funzione di scaling deve soddisfare un'*equazione di dilatazione* (o raffinamento):

$$\phi(t) = \sum_k g_0[k] \phi(at - k)$$

per qualche $a > 0$ e coefficienti $\{g_0[k]\} \in \ell^2$. La funzione è rappresentata come una sovrapposizione di versioni traslate e scalate di se stessa, da cui il nome di funzione di *scaling*. Più precisamente, V_0 è generato da $\{\phi(\cdot - k) : k \in \mathbb{Z}\}$ e, in generale, V_s è generato da $\{\phi_{k,s} : k, s \in \mathbb{Z}\}$, con:

$$\phi_{k,s} = 2^{s/2} \phi(2^s t - k) \quad k \in \mathbb{Z}$$

Si hanno i risultati ovvi:

$$x(t)\in V_s \Leftrightarrow x(2t)\ \in V_{s+1}$$
$$x(t)\in V_s \Leftrightarrow x(t+2^{-s})\ \in V_s$$

Per ogni livello s, dal momento che V_s è un sottospazio di V_{s+1}, rimane dello spazio chiamato W_s, il quale combinato con V_s dà V_{s+1}. È cioè il suo complementare:

$$V_s \cap W_s = \{0\}$$
$$V_s \oplus W_s = V_{s+1}$$

L'ultima relazione prende il nome di *somma diretta*. Questo sottospazio W_s è chiamato *sottospazio delle wavelet*. I sottospazi $\{W_s\}$ sono generati da $\psi(t)\in L^2$, chiamata *wavelet*, nello stesso modo in cui $\{V_s\}$ è generato da $\phi(t)$. In altre parole, si hanno $x_s(t)\in V_s$ e $y_s(t)\in W_s$ che possono scriversi:

$$x_s(t) = \sum_k a_{k,s}\,\phi(2^s t - k)$$
$$y_s(t) = \sum_k w_{k,s}\,\psi(2^s t - k)$$

per delle sequenze $\{a_{k,s}\}, \{w_{k,s}\} \in \ell^2$.

Ora, si ha che:

$$V_1 = V_0 \oplus W_0$$
$$V_2 = V_1 \oplus W_1 = V_0 \oplus W_0 \oplus W_1$$
$$\dots$$
$$V_j = V_{j-1} \oplus W_{j-1} = V_0 \oplus \sum_{j=0}^{J-1} W_j$$
$$\dots$$
$$L^2(\mathbb{R}) = V_0 \oplus \sum_{j=0}^{\infty} W_j$$

La ricorsione può anche essere scritta per $j<0$:

$$V_0 = V_{-k} \oplus \sum W_j = \sum W_j$$

Per cui alla fine si ottiene:

$$L^2(\mathbb{R}) = \sum_{j=-\infty}^{\infty} W_j$$

Cioè per l'appunto le wavelet formano una base ortogonale per $L^2(\mathbb{R})$. L'inclusività della catena di sottospazi implica che ogni segnale a bassa risoluzione è anche un segnale ad alta risoluzione. Si osservi che gli spazi $\{V_s\}$ sono innestati mentre i $\{W_s\}$ sono mutuamente ortogonali. Se si ha anche che $W_s \perp A_s$ allora la decomposizione è detta *decomposizione ortogonale*.

Si noti anche che in generale, non è necessario che le funzioni di base in $\{V_s\}$ e quelle in $\{W_s\}$ siano ortogonali all'interno dello spazio, cioè in generale:

$$\int_{-\infty}^{\infty} \phi(t)\phi(t-\ell)\,dt \neq 0$$
$$\int_{-\infty}^{\infty} \psi(t)\psi(t-\ell)\,dt \neq 0$$

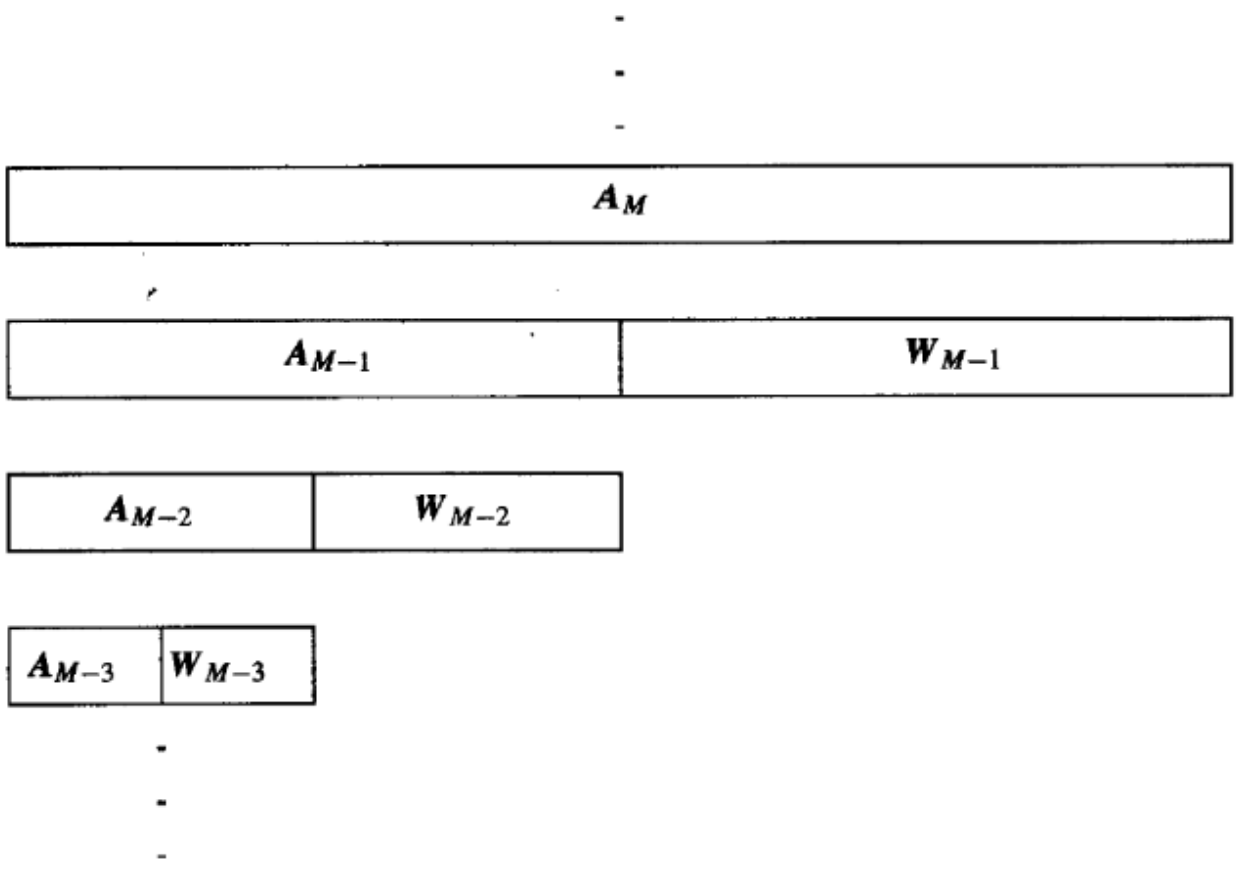


Fig. 12: Decomposizione in cascata degli spazi delle funzioni di scaling (qui indicati con A_i).

Le funzioni di scaling e le wavelet nella stessa scala sono correlate alle funzioni di scaling della scala successiva (più dettagliata), tramite le *relazioni two-scale*. Cioè la relazione tra V_j, W_j con V_{j+1} è governata da:

$$\phi(2^j t)=\sum_k g_0[k]\phi(2^{j+1}t-k)$$
$$\psi(2^j t)=\sum_k g_1[k]\phi(2^{j+1}t-k)$$

Ad esempio, nella base di Haar $\psi(t)=\phi(2t)-\phi(2t-1)$.

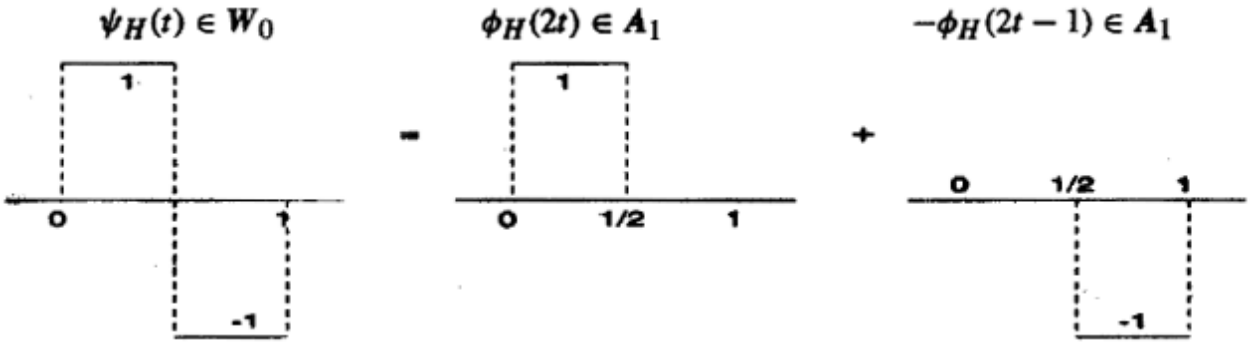


Fig. 13: Relazione two-scale per la wavelet di Haar a risoluzione $j=0$ *, si ha* $g_1[0]=-g_1[1]=1$.

Le *relazioni di decomposizione* permettono invece di ottenere la funzione di scaling a una scala qualsiasi in termini di funzione di scaling e wavelet alla scala più bassa. Esistono cioè due sequenze $\{h_0[k]\}$ e $\{h_1[k]\}$ in ℓ^2 tali che:

$$\phi(2^{j+1}t-\ell)=\sum_k \{h_0[2k-\ell]\phi(2^j t-k)+h_1[2k-\ell]\psi(2^j t-k)\} \quad \text{per ogni } \ell\in\mathbb{Z}$$

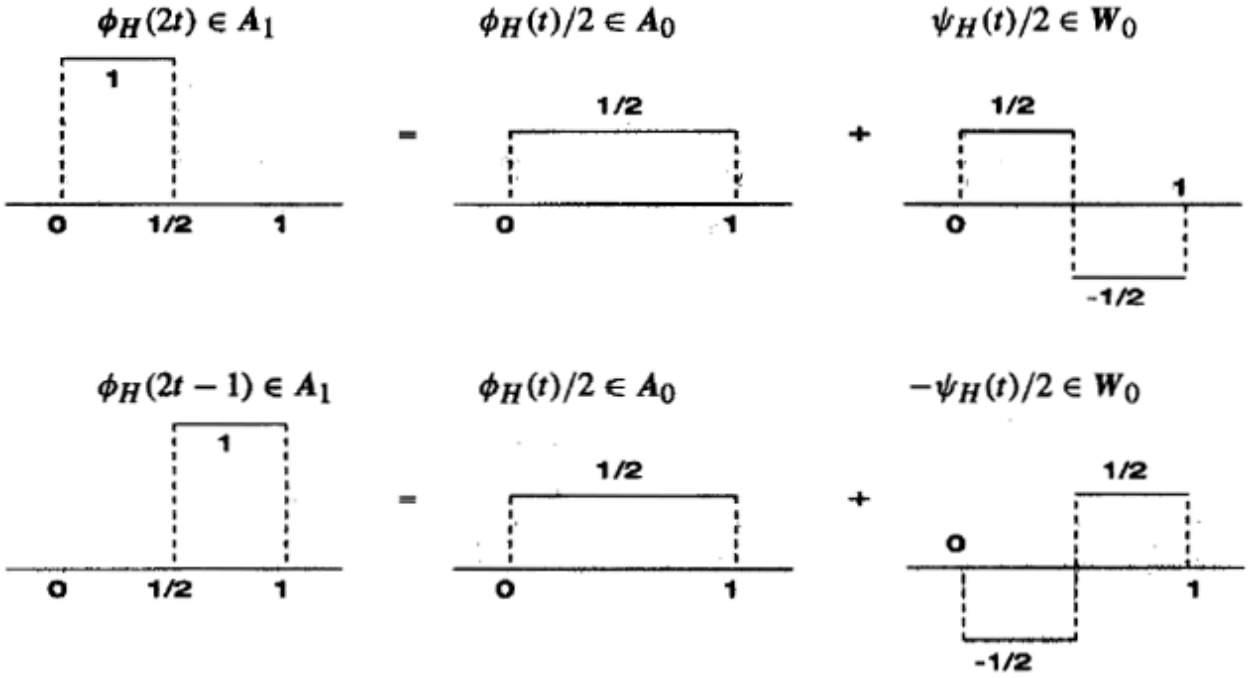


Fig. 14: Relazione di decomposizione nel caso di Haar. Si ha $h_0[0]=h_1[0]=1/2$ *(prima riga) e* $h_0[-1]=-h_1[-1]=1/2$ *(seconda riga).*

1.4 Trasformata Wavelet Discreta

La trasformata wavelet continua si rivela un ottimo strumento per l'analisi in tempo-frequenza di segnali non-stazionari; inoltre, il vasto campo di applicazioni che ne derivano motiva una ricerca costante in questo ambito relativamente nuovo. Tuttavia, la CWT comporta una serie di svantaggi non da poco di cui è necessario tenere conto. Per cominciare, la valutazione dell'integrale di analisi risulta in un calcolo troppo oneroso, e spesso nell'ambito del signal processing l'efficienza e la rapidità di calcolo sono due requisiti imprescindibili. Inoltre, la trasformazione produce valori su scale continue e traslazioni continue; Daubechies ha dimostrato che con una scelta opportuna e minimale dei parametri di traslazione e di scala, il segnale può essere ricostruito perfettamente, il che comporta un certo grado di ridondanza presente nella CWT. Risulta chiaro che una rivalutazione dei metodi di calcolo, e della matematica che ne è alla base, si rende imperativa.

Per superare queste difficoltà, la CWT subisce un particolare processo di discretizzazione che dà vita alla cosiddetta *trasformata wavelet discreta*, o *DWT*.

Chiariamo che per *ridondanza* si intende il fatto che molti dei valori prodotti dalla trasformata non sono in realtà necessari alla ricostruzione del segnale. Consideriamo, a titolo di esempio, un segnale chirp; per poterlo ricostruire fedelmente a partire dai suoi campioni tramite Fourier è necessario campionarlo a una frequenza doppia della sua banda massima (Nyquist). Questa frequenza di campionamento sarebbe eccessiva per le parti in bassa frequenza, cioè si avrebbero tanti campioni non necessari alla ricostruzione perfetta: campioni, dunque, ridondanti. È chiaro anche che nel caso della CWT le funzioni di base sono versioni traslate e scalate in maniera continua di loro stesse; è evidente che una base così formata non può essere veramente ortogonale.

1.4.1 Analisi e Sintesi

Come detto, la CWT genera informazioni ridondanti sul segnale nel piano tempo-scala. Si dimostra che con una particolare scelta dei parametri b e a, la trasformata perde la ridondanza e non solo, il calcolo diventa molto più efficiente. La DWT mantiene infatti abbastanza informazioni sul segnale di modo tale che sia possibile ricostruirlo in maniera perfetta a partire da alcuni coefficienti analoghi ai coefficienti presenti nella serie di Fourier.

Riscriviamo l'espressione della CWT per un segnale $x(t)$:

$$W_{\psi}x(b,a)=\frac{1}{\sqrt{a}}\int_{-\infty}^{\infty}x(t)\overline{\psi(\frac{t-b}{a})}dt$$

La trasformata wavelet discreta consiste essenzialmente nel prendere i parametri b e a nella forma seguente (e dunque *non* corrisponde a una discretizzazione dei tempi e delle frequenze):

$$a\rightarrow 2^{-s} \qquad\qquad b\rightarrow k2^{-s}$$

per $k,s\in\mathbb{Z}$.

La discretizzazione del parametro di scala viene dunque effettuata su una scala logaritmica: scegliendo la base 2 si parla di campionamento *diadico*. In fig. 15 viene mostrata una rappresentazione grafica del piano tempo-frequenza (e non del piano tempo-scala, in quanto in figura è $1/a$ a variare sulle ordinate) campionato seguendo questo procedimento diadico: ad esempio, per $1/a=8$ si ha $s=3$ e il parametro di traslazione assume valori multipli di $1/8$; per $s=2$, b varia a multipli di $1/4$, e così via. Chiaramente se si fosse disegnato il piano con a che varia sulle ordinate, si sarebbe ottenuta una versione ribaltata del

grafico.

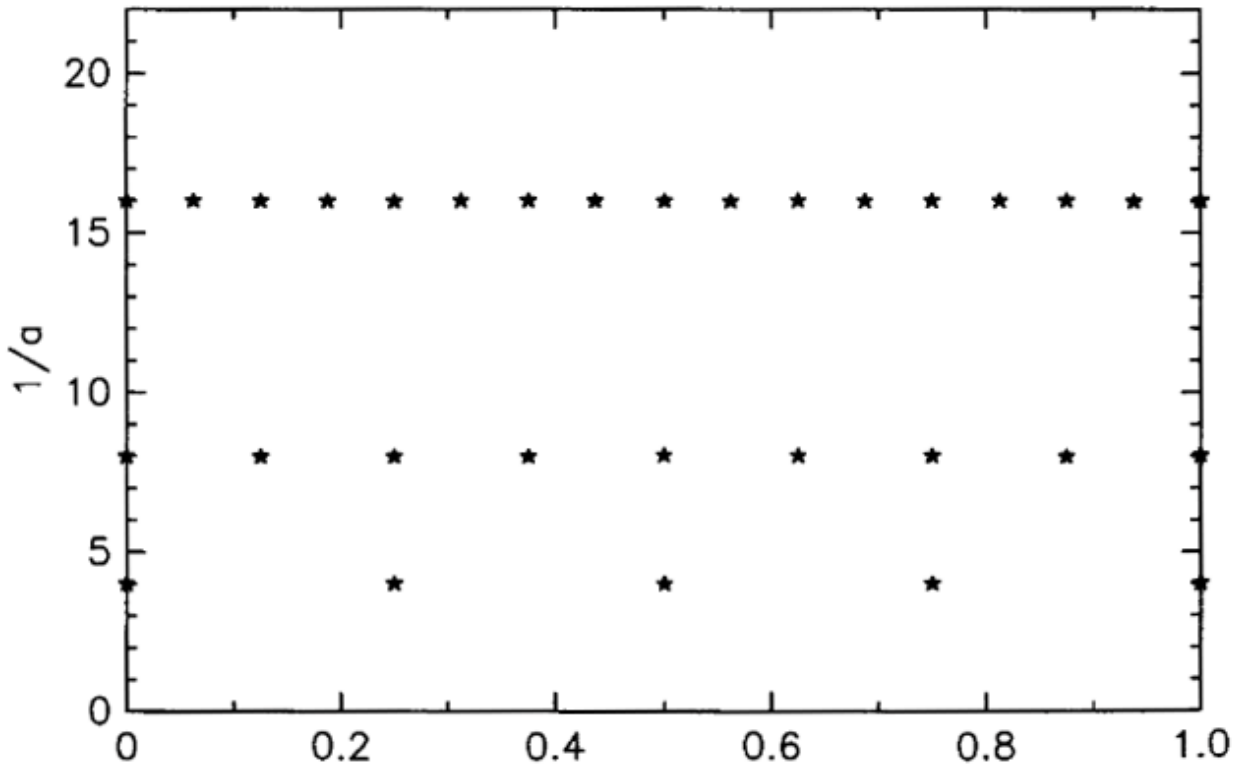


Fig. 15: Campionamento diadico del piano tempo-frequenza.

È ragionevole a questo punto chiedersi se, con questa scelta di discretizzare a e b in maniera diadica, sia possibile ricostruire il segnale in maniera perfetta a partire dalla sua trasformata. Daubechies dimostra che è possibile, a patto che vengano soddisfatte alcune condizioni di tipo energetico; tuttavia questi dettagli non verranno qui discussi.

Con questa forma di campionamento (anche chiamato *campionamento critico*), la CWT diventa:

$$W_\psi f(k2^{-s},2^{-s}) = 2^{s/2}\int_{-\infty}^{\infty} f(t)\overline{\psi(\frac{t-k2^{-s}}{2^{-s}})}\,dt = 2^{s/2}\int_{-\infty}^{\infty} f(t)\overline{\psi(2^s t-k)}\,dt$$

che per k e s fissati, rappresenta un valore nel punto diadico $(k/2^s,1/2^s)$ sul piano tempo-frequenza. La DWT genera dunque un insieme sparso di valori sul piano, valori che rappresentano la correlazione tra il segnale in analisi e la wavelet.

Nel caso di sequenza numerica, ad esempio ottenuta campionando $f(t)$ con intervallo di

campionamento pari a 1 ($f(nT)=f(n)$):

$$W_\psi f(k2^{-s},2^s) \approx 2^{s/2}\sum_n f(n)\psi(2^s n-k)$$

Cioè la wavelet madre è traslata per numeri interi e scalata a potenze di 2.

A questo punto si può apprezzare il vantaggio principale della trasformata wavelet rispetto alla STFT. La copertura del piano tempo-frequenza è mostrata di seguito:

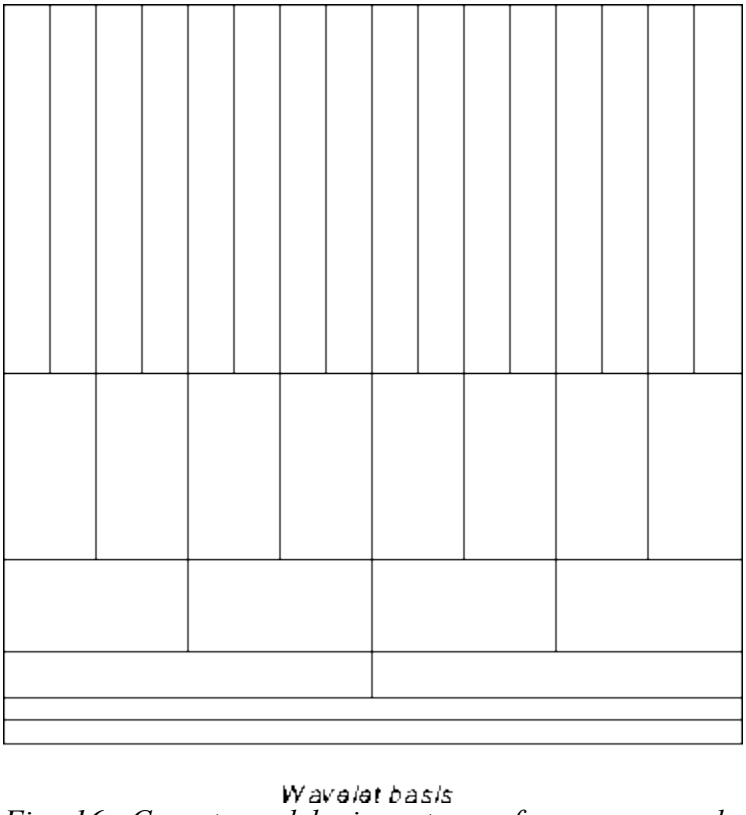


Fig. 16: Copertura del piano tempo-frequenza con la base di wavelet. La risoluzione cambia al variare della scala.

A frequenze basse si ha una buona risoluzione in frequenza ma bassa risoluzione nel tempo, e le wavelet catturano fenomeni di lunga durata. Ad alte frequenze la wavelet ha un supporto piccolo e la risoluzione nel tempo migliora notevolmente, rendendo possibile il rilevamento di dettagli come cuspidi o salti o in generale fenomeni di breve durata, mentre la risoluzione in frequenza peggiora. In particolare la risoluzione varia a fattori di 2: ogni livello di dettaglio ha la metà della risoluzione nel tempo rispetto al livello precedente, e il doppio della risoluzione in

frequenza.

Chiaramente per il principio di incertezza di Heisenberg non si può avere una localizzazione veramente istantanea, ma possiamo solo sapere quali bande di frequenza esistono a quali intervalli di tempo.

Potrebbe sembrare a questo punto che, di fronte alla DWT, la CWT perda di utilità. In realtà le due trasformate si adattano a compiti diversi a e in ambiti diversi. Un'analisi di tipo continuo risulta più facile da interpretare proprio grazie alla ridondanza che rende le informazioni più visibili. Quindi usando la CWT si guadagna in "leggibilità" e facilità di interpretazione, ma si perde in termini di costi computazionali e di spazio. Al contrario la DWT si presta molto ad applicazioni di elaborazione dei segnali piuttosto che di analisi in tempo-frequenza, grazie anche all'efficienza dell'algoritmo che ne permette la realizzazione (nella sezione seguente).

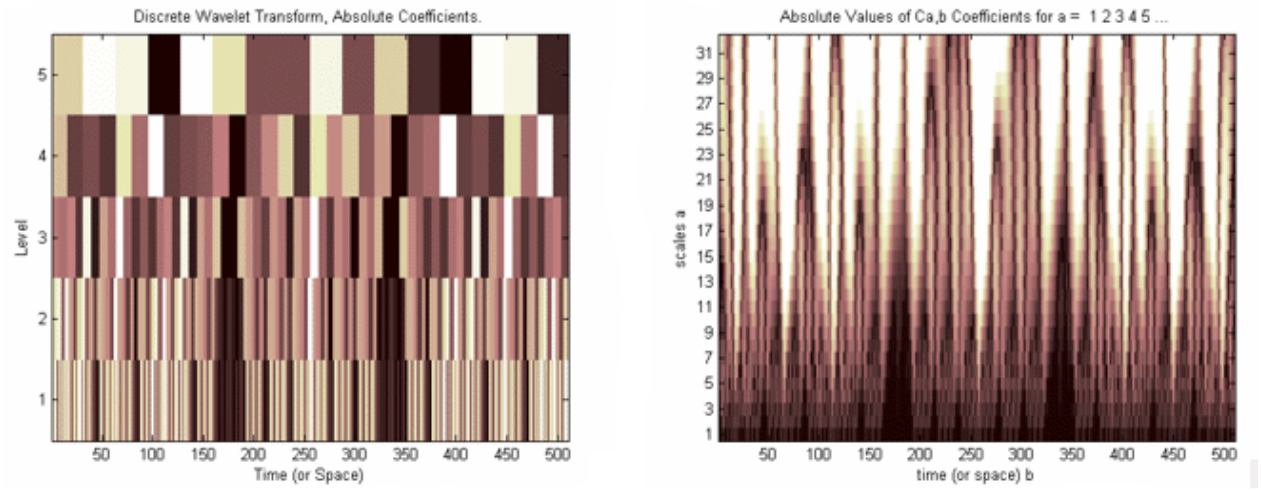


Fig. 17: La DWT contiene tutte le informazioni necessarie per una perfetta ricostruzione del segnale, ma la sua interpretazione ne risulta difficoltosa. Le ridondanze della CWT facilitano invece l'interpretazione.

Esiste una trasformata inversa univoca per la DWT (trasformata di sintesi) tale che la funzione originale possa essere ricostruita perfettamente dalle sue componenti alle diverse scale. L'algoritmo di ricostruzione è basato sulle relazioni two-scale della funzione di scaling e della wavelet. In questa sede non siamo molto interessati alla sintesi di segnali a partire dalla loro trasformata wavelet, ma un'esposizione sommaria ne viene data in sez. 1.4.2.2.

1.4.2 Fast Wavelet Transform

Il calcolo della DWT può essere effettuato con un procedimento piuttosto semplice. Alla fine degli anni '80 Stephane Mallat collaborò con Yves Meyer nello sviluppo di una costruzione di tipo MRA per wavelet a supporto compatto, il che rese l'implementazione delle wavelet pratica per applicazioni ingegneristiche. Mallat dimostrò l'equivalenza tra le basi di wavelet e un certo tipo di filtri numerici. Questo procedimento prende oggi il nome di *Fast Wavelet Transform*, e può essere considerato un algoritmo veloce alla stregua degli algoritmi FFT per il calcolo della trasformata discreta di Fourier.

L'idea principale dell'algoritmo è quella di associare a funzione di scaling e wavelet due filtri numerici, chiamati rispettivamente filtro di *analisi* e filtro di *sintesi*, da applicare al segnale in ingresso. In questo modo se ne ottiene la decomposizione secondo i paradigmi dell'MRA. Sebbene inerentemente la funzione è la medesima svolta dalla DWT, l'obiettivo dell'algoritmo di Mallat è quello di separare le componenti in bassa ed alta frequenza del segnale affinché possano poi essere processate da diversi algoritmi di DSP.

1.4.2.1 Algoritmo di decomposizione

Esiste tutta un'area di studio dei segnali che tratta la rappresentazione di funzioni a più frequenze di campionamento (e si parla di rappresentazione *multirate*). Due meccanismi tipici per cambiare la frequenza di campionamento sono la *decimazione* e l'*interpolazione*.

Un decimatore a M punti, applicato a un segnale in ingresso, mantiene un solo campione ogni M e viene rappresentato dal seguente diagramma:

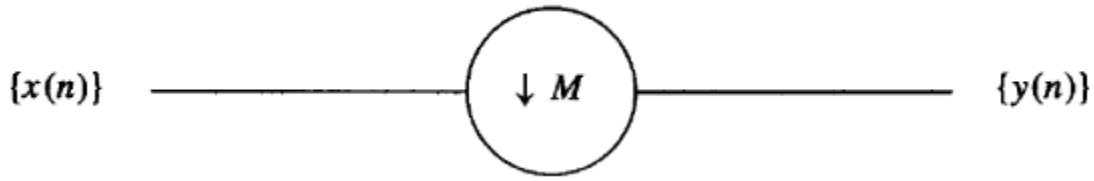


Fig. 18: Decimatore a M punti.

Nel tempo, la decimazione a M punti di un sequenza $x(n)$ è data da

$$y(n)=x(nM) \quad n\in\mathbb{Z}$$

Si noti che il segnale decimato $y(n)$ è una versione contratta di $x(n)$. Lo spettro del segnale decimato risulta espanso di un fattore M rispetto alla versione non decimata, e l'ampiezza ridotta di un fattore $1/M$. Un decimatore chiaramente può introdurre *aliasing* se la banda spettrale del segnale in input è maggiore di π/M (cioè se $|w|>\pi/M$); a ogni modo, per come è progettato l'algoritmo di ricostruzione, l'aliasing introdotto in questo modo può essere eliminato [7].

Per *interpolazione* si intende invece la procedura di inserire punti in una sequenza per aumentare il tasso di campionamento; questi campioni aggiuntivi sono in genere scelti nulli.

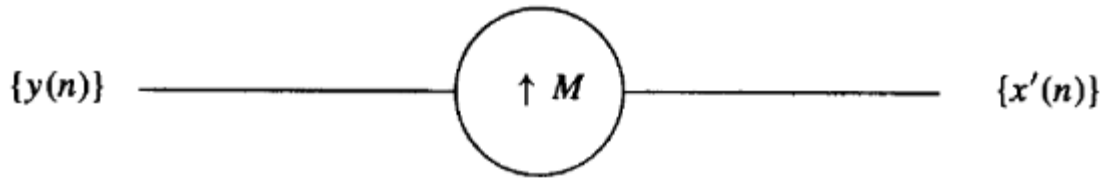


Fig. 19: Interpolatore a M punti.

Seguendo un ragionamento analogo a quanto fatto col decimatore, si osserva che lo spettro del segnale interpolato, essendo questo una versione *espansa* del segnale originale, viene compresso di un fattore M sull'asse delle frequenze. Al contrario del decimatore, non c'è ora pericolo di aliasing dal momento che lo spettro in output è più stretto di quello in input. In forma matriciale, si ottiene che la matrice di interpolazione è l'inversa (e per ortogonalità è dunque la trasposta) della matrice di decimazione.

La decimazione e l'interpolazione investono un ruolo piuttosto importante nello sviluppo di algoritmi efficienti per la decomposizione e la ricostruzione di segnali tramite funzioni di scaling e wavelet.

In particolare le operazioni di convoluzione seguita da decimazione e di interpolazione seguita da convoluzione hanno un ruolo principe nel cosiddetto *algoritmo piramidale di Mallat*.

1. Convoluzione seguita da decimazione:

$$\{h(n)*x(n)\}_{\downarrow 2}=\{\sum_k x(k)h(n-k)\}_{\downarrow 2}=\sum_k x(k)h(2n-k)$$

2. Interpolazione seguita da convoluzione:

$$\{g(n)*[x(n)]_{\uparrow 2}\}=\{g(n)*x(\frac{n}{2})\}=\sum_{l} x(\ell)g(n-2\ell)$$

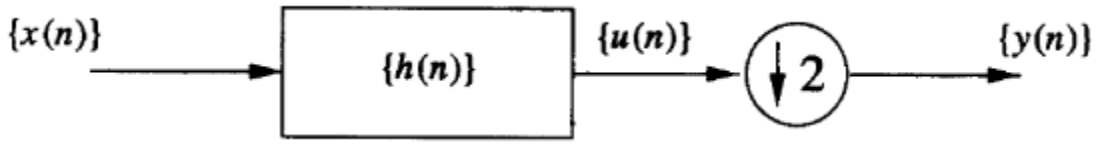


Fig. 20: Convoluzione seguita da decimazione.

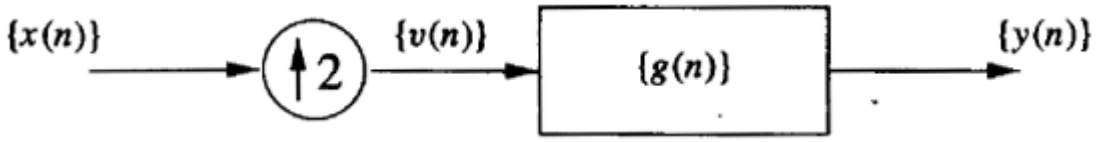


Fig. 21: Interpolazione seguita da convoluzione.

Come detto, l'algoritmo di decomposizione separa un segnale in componenti a scale diverse. Dato un segnale di energia, la sua decomposizione ha inizio con il mapping all'interno di un sottospazio V_M ad alta risoluzione:

$$x_M=\sum_k a_{k,M}\phi(2^M t-k) \in V_M$$

Dal momento che

$$V_M=\sum W_{M-n}+V_{M-N}$$

allora si può scrivere

$$x_M(t)=\sum_{n=1}^{N} y_{M-n}+x_{M-N}$$
$$x_s(t)\in V_s \Rightarrow x_s(t)=\sum_k a_{k,s}\phi_{k,s}(t)$$
$$y_s(t)\in W_s \Rightarrow y_s(t)=\sum_k w_{k,s}\psi_{k,s}(t)$$

dove x_{M-N} è la versione più grossolana di $x_M(t)$, e i segnali y_{M-n} sono i segnali *di dettaglio*. I coefficienti di approssimazione $\{a_{k,M-1}\}$ e i coefficienti wavelet $\{w_{k,M-1}\}$

possono essere calcolati a partire da $\{a_{k,M}\}$. Infatti, a partire da

$$\sum_k a_{k,s+1}\phi_{k,s+1}(t)=\sum_k a_{k,s}\phi_{k,s}(t)+\sum_k w_{k,s}\psi_{k,s}(t)$$

e utilizzando la relazione di decomposizione (sez. 1.3) ed alcuni passaggi [7] si ottiene:

$$a_{k,s}=\sum_\ell h_0[2k-\ell]a_{\ell,s+1}$$
$$w_{k,s}=\sum_\ell h_1[2k-\ell]a_{\ell,s+1}$$

Questo processo viene ripetuto, cioè i coefficienti del prossimo livello $M-2$ vengono ottenuti $\{a_{k,M-1}\}$, e così via fino allo spazio di approssimazione desiderato. Ogni wavelet è dunque univocamente caratterizzata dai coefficienti dei filtri che compaiono nelle relazioni di scaling.

È interessante osservare che le due equazioni corrispondono all'applicazione di un decimatore a 2 punti seguito di una convoluzione. Queste due formule mettono in relazione *i coefficienti* delle funzioni di scaling e delle wavelet a una scala generica con quelli della scala successiva. Adottando la notazione

$$\boldsymbol{a}_s:=\{a_{k,s}\} \qquad \boldsymbol{w}_s:=\{w_{k,s}\} \qquad \boldsymbol{h}_0:=\{h_0[k]\} \qquad \boldsymbol{h}_1:=\{h_1[k]\}$$

si può ora mostrare l'algoritmo (in fig. 22).

I segnali $\boldsymbol{h}_0$ e $\boldsymbol{h}_1$ prendono il nome di *filtri di analisi* (la coppia è anche chiamata *quadrature mirror filter*). In particolare, il filtro $\boldsymbol{h}_0$ è di tipo passa-basso e produce una versione a più bassa risoluzione del segnale in ingresso: corrisponde cioè ai coefficienti di una funzione di scaling. Per il Teorema del Campionamento, un segnale filtrato da un passa-basso può essere rappresentato dalla metà dei suoi campioni, in quanto la sua banda massima è stata dimezzata: per dimezzare il numero dei campioni si usa appunto un decimatore a 2 punti. Il filtro $\boldsymbol{h}_1$ è invece di tipo passa-alto e produce il dettaglio perso nel passaggio da $\boldsymbol{a}_s$ a $\boldsymbol{a}_{s+1}$: corrisponde quindi ai coefficienti di una wavelet. Si dimostra che il numero dei coefficienti del filtro è in relazione al numero di momenti svanenti della wavelet. L'algoritmo di decomposizione può essere applicato ricorsivamente al segnale $\boldsymbol{a}_{s+1}$, e ancora su $\boldsymbol{a}_{s+2}$, e così via fino alla risoluzione desiderata o fino a quando il segnale consiste di un numero troppo piccolo di valori. All'avanzare dei livelli di trasformazione, il filtro $\boldsymbol{h}_0$ è sempre più selettivo sulle frequenze basse; allo stesso modo, $\boldsymbol{h}_1$ restringe sempre più la banda passante. La DWT del segnale originale è quindi composta dai campioni dell'ultima sequenza $\boldsymbol{a}_\ell$ concatenati con i diversi

segnali di dettaglio, ordinati per risoluzione crescente.

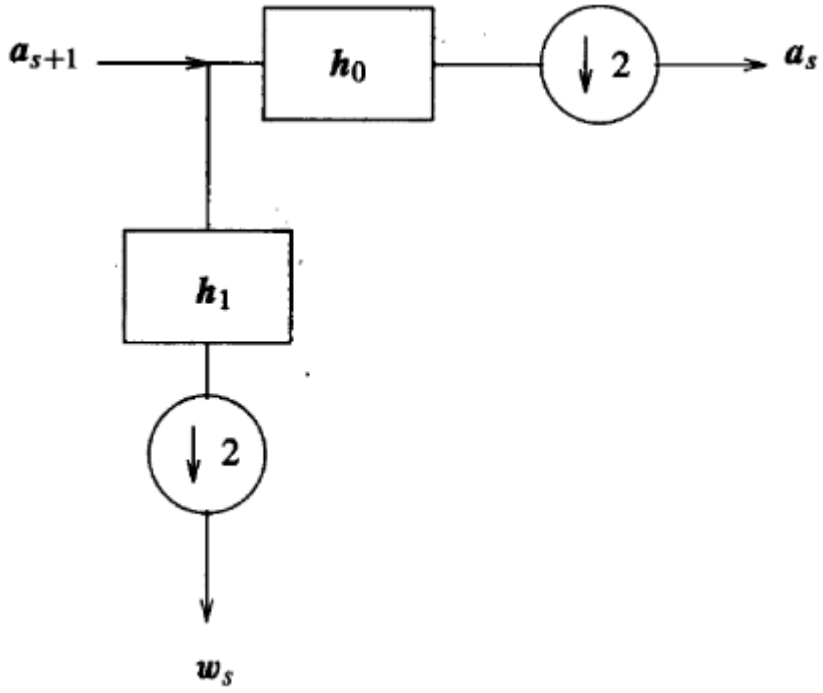


Fig. 22: Singolo livello di decomposizione secondo l'algoritmo di Mallat.

Fig. 23: DWT di un segnale con risoluzione iniziale max e risoluzione finale 0.

In letteratura l'analisi dei segnali seguendo questo approccio è denominata anche *subband coding*. Procedere nella ricorsione corrisponde, in questa visione "elettronica" alla multi-risoluzione, alla progettazione di un *banco di filtri*, un insieme di filtri che separano le componenti frequenziali di un segnale in segnali separati, ognuno con un sottoinsieme delle frequenze. Combinando le bande passanti dei filtri, secondo la multi-resolution analysis, si ottiene una copertura completa dell'intervallo delle frequenze, e i filtri quindi sono complementari. Per fare un paragone, la trasformata di Fourier può essere considerata un banco

di filtri “speciale”: separa il segnale d'ingresso in tanti segnali quante sono le armoniche che lo compongono.

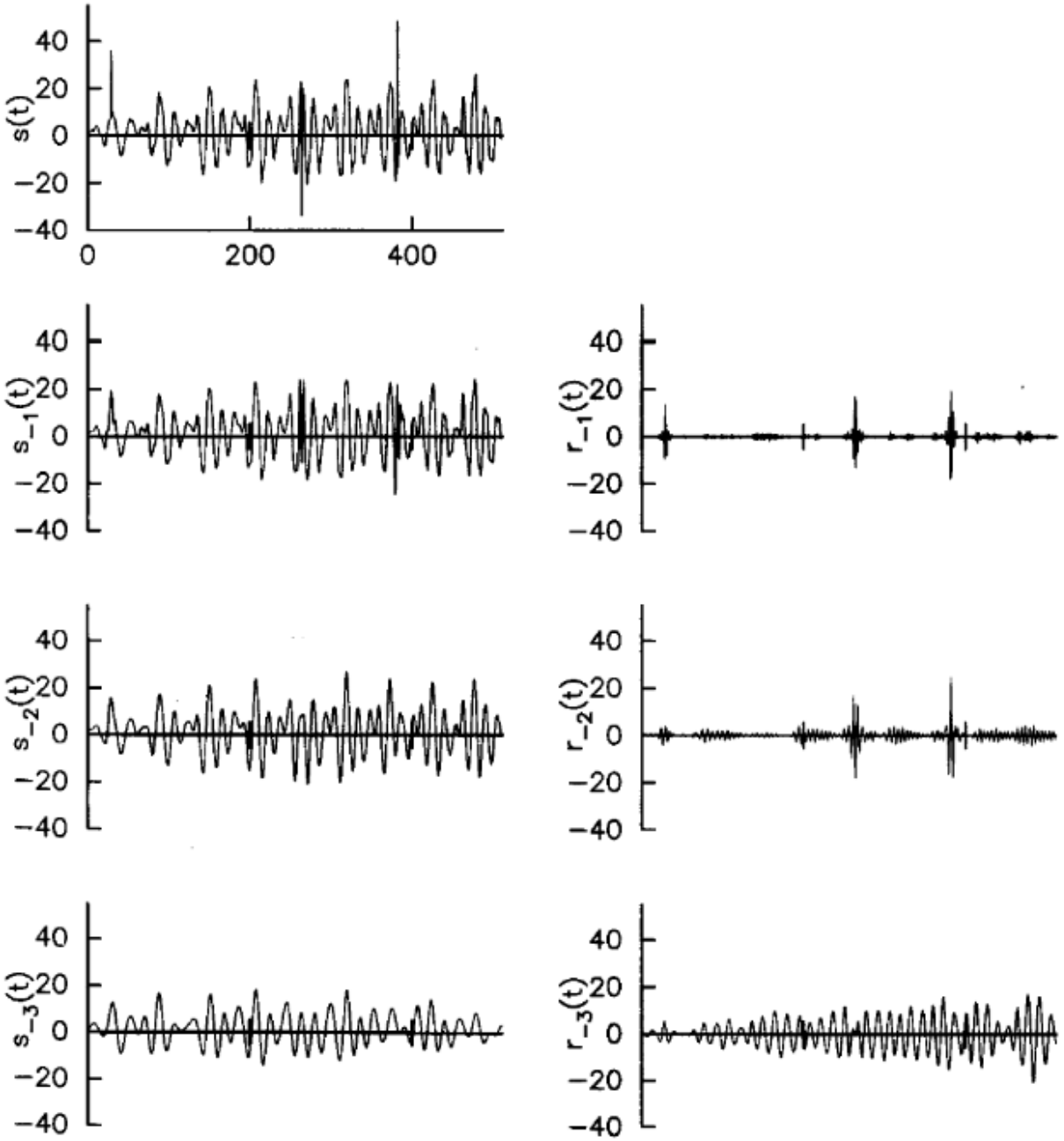


Fig. 24: Decomposizione a 3 livelli di un segnale. Ogni riga è uno stadio della decomposizione e contiene segnale approssimato e segnale di dettaglio.

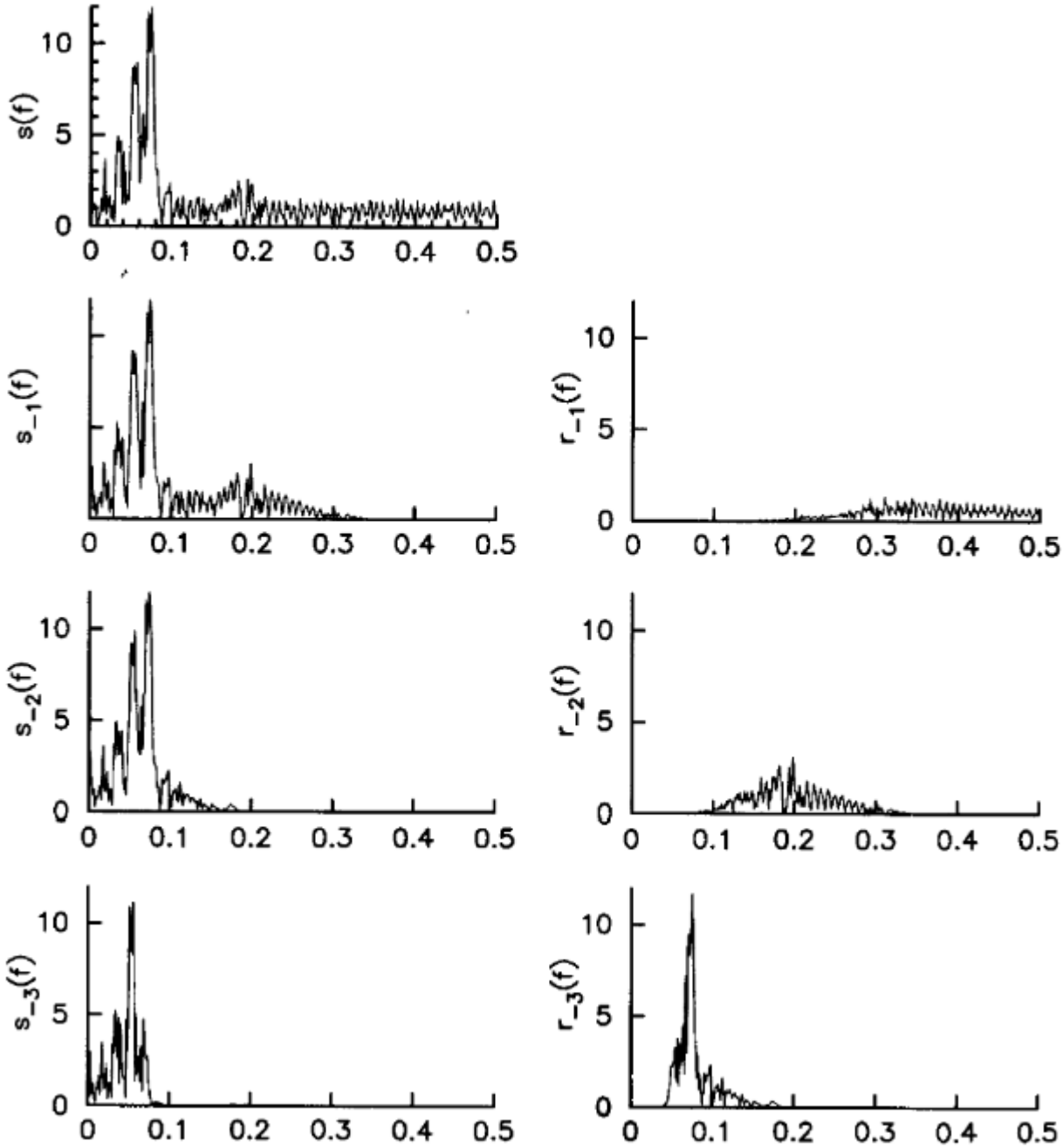


Fig. 25: Spettro d'ampiezza ai vari stadi della decomposizione, sono evidenti le caratteristiche di passa-basso e passa-banda delle funzioni di scaling e delle wavelet.

A ogni stadio, l'algoritmo è applicato a metà dei campioni rispetto allo stadio precedente. Quindi se la sequenza iniziale è lunga $n = 2^m$, il numero di operazioni totale è al più:

$$2^m + \frac{2^m}{2} + \frac{2^m}{4} + \ldots + \frac{2^m}{2^{m-1}} = 2^m \cdot \sum_{i=0}^{m-1} 2^{-i} = 2^m \frac{1-2^{-m}}{½} = 2 \cdot 2^m - 2 = 2 \cdot n - 2$$

quindi questo procedimento ha una complessità di $O(n)$. Tuttavia, il vantaggio principale di questi algoritmi non è l'efficienza, bensì il fatto che, dal momento che le wavelet sono univocamente caratterizzate dai coefficienti dei filtri, *non è nemmeno necessario specificare esplicitamente le wavelet*! Ciò risulta anche di notevole utilizzo nei numerosi casi in cui la wavelet o la funzione di scaling non siano note in forme chiusa (come è il caso della famiglia delle Daubechies). Tuttavia, *non tutte le wavelet* possono essere espresse sotto forma di filtri.

1.4.2.2 Algoritmo di ricostruzione

Nella sezione precedente si è mostrato come sia possibile effettuare la trasformata wavelet discreta di un segnale utilizzando una coppia di filtri di analisi. I due filtri corrispondono a una rappresentazione numerica della funzione di scaling, che opera come passa-basso, e della wavelet, che agisce da passa-banda, seguiti da una forma di downsampling che prende il nome di "decimazione a 2 punti". Il risultato di questo algoritmo è una coppia di segnali: il segnale ottenuto tramite il primo filtro è una versione a bassa risoluzione del segnale in ingresso; di contro, il secondo segnale rappresenta il dettaglio "perso" nell'applicazione del primo filtro. L'algoritmo può essere applicato ricorsivamente sul segnale a bassa risoluzione, fino alla risoluzione desiderata o comunque fino a quando il numero di campioni disponibili lo consente.

Esiste una trasformata di sintesi che permette di recuperare perfettamente la funzione originale a partire dai componenti nelle diverse scale. L'algoritmo di ricostruzione è ancora basato sulle relazioni two-scale, e in maniera analoga a prima si ottiene una corrispondenza con le equazioni relative all'operazione di interpolazione seguita da convoluzione.

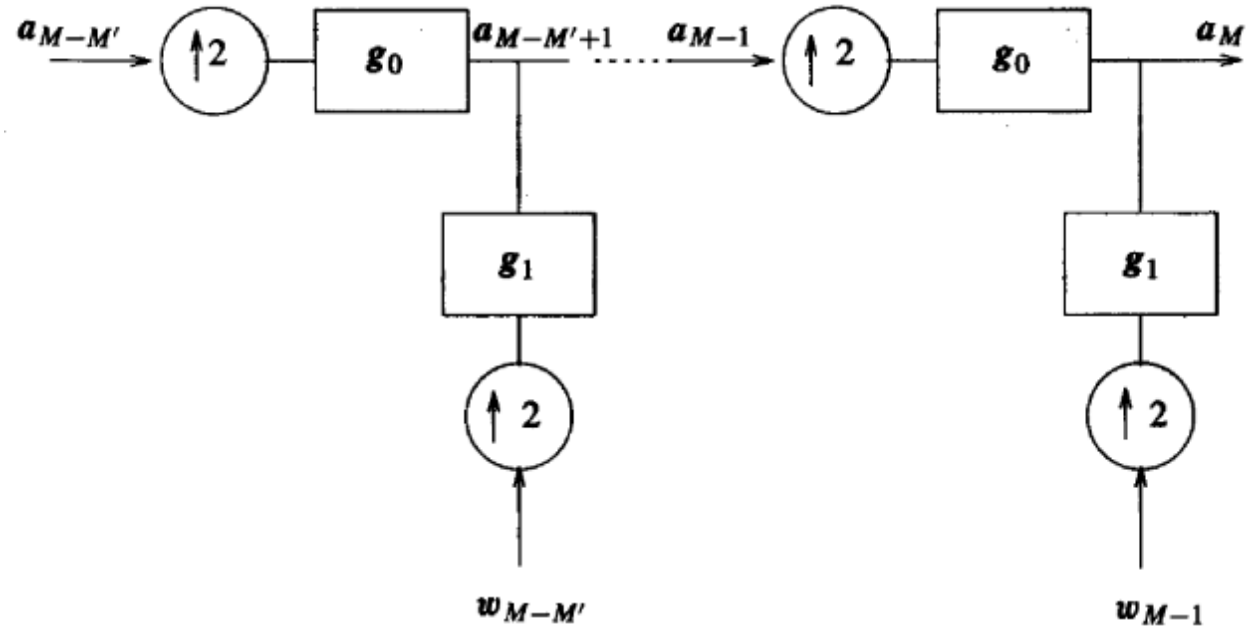


Fig. 26: Ricostruzione del segnale a_M *a partire dai coefficienti delle funzioni di scaling e delle wavelet.*

In particolare, i filtri di sintesi formano un banco di filtri la cui rappresentazione in forma di matrice corrisponde all'inversione dei filtri di analisi. Per maggiori dettagli si rimanda a [7].

1.5 Applicazioni

Le wavelet e l'analisi in multi-risoluzione hanno trovato molte applicazioni in campi diversi, e a tutt'oggi vengono sfruttate in modi sempre diversi e originali. Gli esperti del campo sostengono che la teoria è in fase di rifinitura e che la vera forza delle wavelet risiede proprio nel vasto contributo effettivo e potenziale che possono dare in domini applicativi tanto diversi.

Vengono di seguito mostrate sommariamente le applicazioni che storicamente hanno trovato immediato vantaggio nell'uso delle wavelet: la *riduzione di rumore* e la *compressione*.

1.5.1 Riduzione del rumore

Sia f un segnale affetto da rumore di tipo Gaussiano, come in fig. 27a.

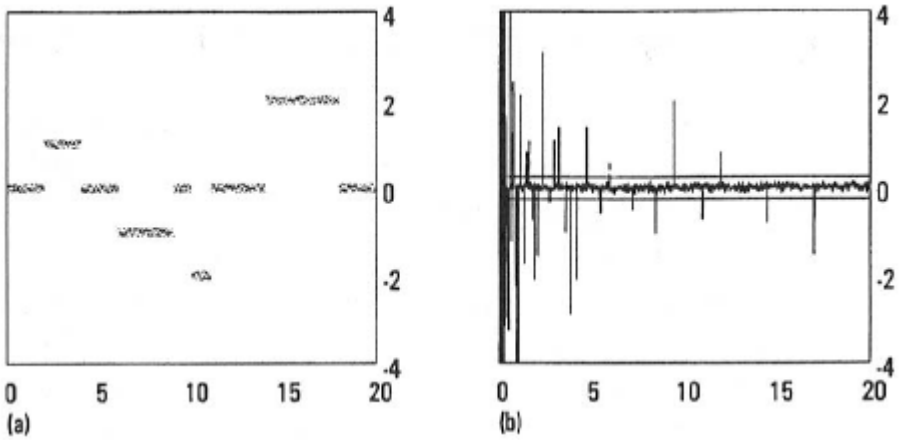


Fig. 27: Segnale affetto da rumore (a) e sua trasformata wavelet (b). Nel grafico della trasformata è possibile vedere la soglia.

È ragionevole scegliere una soglia T al di sotto della quale annullare i valori della trasformata; antitrasformando si ottiene il segnale depurato, in parte, del rumore:

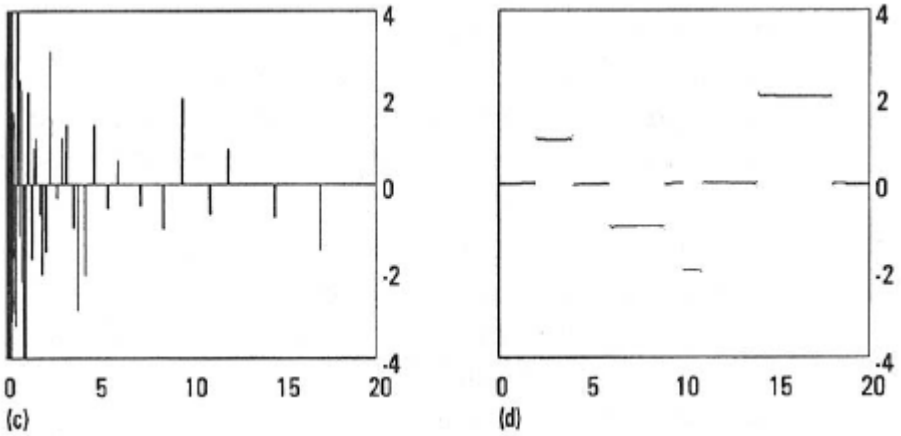


Fig. 28: Segnale trasformato dopo l'applicazione della soglia (c) e risultato della ricostruzione (d).

È chiaro che così facendo è possibile che alcuni valori di fluttuazione vengano sopraffatti dal rumore e scambiati per tali. Dunque più la trasformata (la wavelet) scelta localizza l'energia del segnale in valori sopra la soglia, migliore sarà la rimozione del rumore. Nel caso in cui il rumore non sia di tipo Gaussiano ma di tipo "pop", cioè presente in forma di picchi improvvisi, è possibile ad esempio usare, al posto di una semplice soglia, delle *bande di accettazione* definite per ogni sotto-segnale di fluttuazione, in maniera tale che i valori in banda vengano mantenuti, e

quelli al di fuori vengano azzerati. Un fatto interessante è che il valore della soglia può essere determinato in maniera automatica a partire da considerazioni sul tipo di rumore, senza nessuna conoscenza a priori del segnale affetto [7].

Il meccanismo mostrato sopra prende il nome di *sogliatura forte*. Esso consiste nell'applicazione di una funzione non continua:

$$H(x)=x \quad se \quad |x|\geq T \quad ; \quad H(x)=0 \quad se \quad |x|<T$$

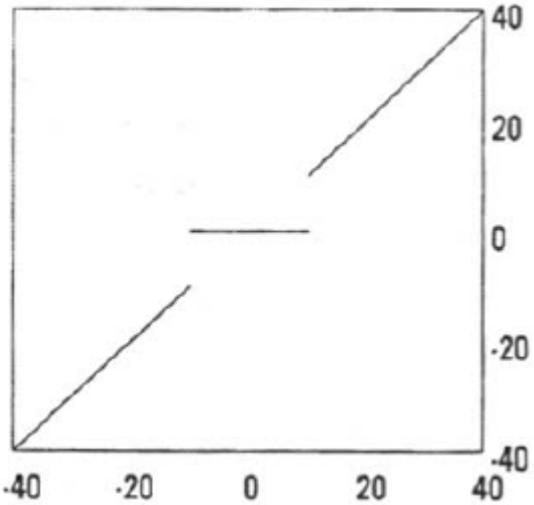


Fig. 29: Funzione di sogliatura forte, sono evidenti le discontinuità.

Questa funzione esagera le differenze tra valori vicini a T : se un valore della trasformata è leggermente inferiore a T , viene azzerato, mentre se è solo poco più grande di T viene mantenuto. Si possono ottenere risultati migliori adottando una funzione di soglia continua, che non discrimini in maniera così definitiva valori "buoni" da valori "cattivi". Questo secondo metodo prende il nome di *sogliatura morbida* e consiste nell'applicazione di una funzione di soglia più *smooth*, ad esempio:

$$S(x)=x \quad se \quad |x|\geq T \quad ; \quad S(x)=2\mathrm{x}-T \quad se \quad T/2\leq x<T$$
$$S(x)=T+2\mathrm{x} \quad se \quad -T<x\leq -T/2 \quad ; \quad S(x)=0 \quad se \quad |x|<T/2$$

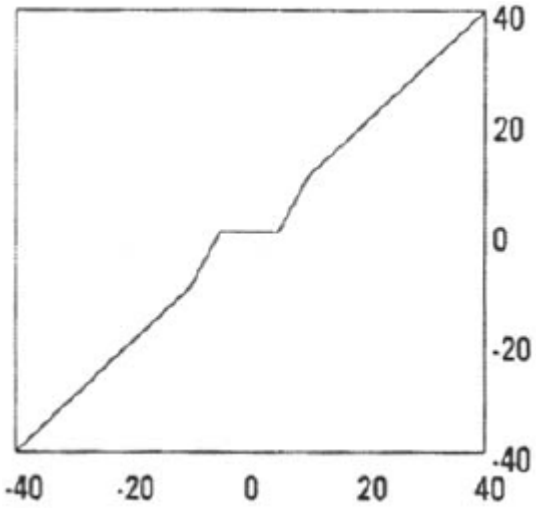


Fig. 30: Funzione di sogliatura morbida.

Un metodo alternativo (rispetto alla sogliatura) per distinguere i valori buoni da quelli affetti da rumore è dato dal principio secondo il quale se dei valori significativi si trovano nelle stesse posizioni relative, a diversi livelli di fluttuazione, allora probabilmente non si tratta di rumore.

Gli algoritmi per la riduzione di rumore basati sulle wavelet possono essere applicati anche a parti di segnali, cioè possono lavorare in maniera locale; ad esempio se un'immagine (capitolo 1.6) è affetta da rumore solamente su una porzione, o su una banda, basta applicare uno dei metodi visti prima solo su quell'intervallo, e dunque antitrasformare.

1.5.2 Compressione

La compressione è probabilmente un'applicazione ovvia della trasformata wavelet. In sez. 1.4.2.1 si è mostrato un algoritmo veloce per la decomposizione di un segnale in sue versioni approssimate fino al livello di risoluzione desiderato. È chiaro che questo procedimento può essere anche visto come un modo per comprimere un segnale (nel senso della teoria dell'informazione) secondo un approccio con perdita (*lossy compression*). Algoritmi di questo tipo sono di pronto utilizzo in ambito audio-video. Mentre la trasformata wavelet bidimensionale (vedi sezione successiva) ha uno dei suoi storici utilizzi nella compressione delle impronte digitali, e sull'analisi multirisoluzione si basa lo standard JPEG2000, ad oggi risultati sperimentali mostrano che la trasformata wavelet non è di particolare vantaggio nella compressione di segnali audio. Tuttavia, è di recente concepimento un codec video sperimentale, chiamato SNOW [31], che permette di ottenere ottima qualità con bitrate molto bassi.

1.6 Trasformata Wavelet a due dimensioni

Nel campo della Machine Vision, diversi risultati sperimentali e teorici mostrano come possa essere tratto particolare vantaggio dall'uso di wavelet e analisi multirisoluzione.

Sia $f(m,n)$ un segnale numerico bidimensionale (immagine). Esso è detto *separabile* se vale l'identità:

$$x(m,n)=x_1(m)x_2(n)$$

Definiamo il prodotto scalare tra immagini in questo modo:

$$f\cdot g=f_{1,1}g_{1,1}+f_{1,2}g_{1,2}+\ldots+f_{N,M}g_{N,M} \qquad \text{dove} \quad f_{i,k}=f(i,k) \ .$$

I diversi livelli di trasformazione possono essere calcolati, analogamente a quanto accadeva a una dimensione, usando prodotti scalari con immagini di scaling e wavelet bidimensionali.

Per una funzione di scaling ϕ e la corrispettiva wavelet ψ, costruiamo tre diverse wavelet bidimensionali e una funzione di approssimazione (di scaling) usando l'approccio del prodotto tensore, per il generico livello di trasformazione ℓ .

La funzione di scaling 2-D è una funzione separabile ed è così definita:

$$\phi(x,y)=\phi(x)\phi(y)$$

e dal momento che sia $\phi(x)$ che $\phi(y)$ soddisfano l'equazione di dilatazione (sez. 1.3)

$$\phi(x)=\sqrt{2}\sum_k g_0[k]\phi(2x-k)$$

allora possiamo scrivere

$$\phi(x,y)=\phi(x)\phi(y)=2\sum_{k,\ell} h[k,\ell]\phi(2x-k)\phi(2y-\ell)$$

dove $h[k,\ell]=h[k]\cdot h[\ell]$ e $h[k]$ è una ridenominazione più comoda per $g_0[k]$.

Allo stesso modo possiamo definire una wavelet $\psi^{[h]}(x,y)$ in questo modo:

$$\psi^{[h]}(x,y)=\phi(x)\psi(y)=2\sum_{k,\ell} g_0[k]\phi(2x-k)g_1[\ell]\psi(2y-\ell)$$

e chiamando per comodità $g_0[k]=h[k]$ e $g_1[k]=g[k]$ possiamo scrivere:

$$\psi^{[h]}(x,y)=\phi(x)\psi(y)=2\sum_{k,\ell} g[k,\ell]\phi(2x-k)\psi(2y-\ell)$$

con $g[k,\ell]=h[k]\cdot g[\ell]$.

Quindi in generale per il caso bidimensionale si definisce la funzione di scaling:

$$\Phi_{i,j}(x,y)=2^{\ell}\phi(2^{\ell}x-i)\phi(2^{\ell}y-j)$$

dove i e j indicano le traslazioni orizzontali e verticali, e le *tre* wavelet

$$\begin{aligned}\Psi^{[h]}_{i,j}(x,y)&=2^{\ell}\phi(2^{\ell}x-i)\psi(2^{\ell}y-j)\\ \Psi^{[v]}_{i,j}(x,y)&=2^{\ell}\psi(2^{\ell}x-i)\phi(2^{\ell}y-j)\\ \Psi^{[d]}_{i,j}(x,y)&=2^{\ell}\psi(2^{\ell}x-i)\psi(2^{\ell}y-j)\end{aligned}$$

che sono wavelet perché soddisfano

$$\int\int \Psi^{[t]}_{i,j}(x,y)dx\,dy=0 \qquad \text{per} \quad t=[h],[v],[d]$$

Il significato dei simboli $[h]$, $[v]$ e $[d]$ sarà chiaro tra breve. Ognuna di queste wavelet e la funzione di scaling occupano una diversa porzione del piano (fig. 32). In fig. 31 è mostrata la wavelet $\Psi^{[h]}_{2,3}(x,y)$, che produce il valore in coordinate $(2,3)$ per la sotto-immagine h (vedere appresso), tramite il prodotto scalare tra le due funzioni.

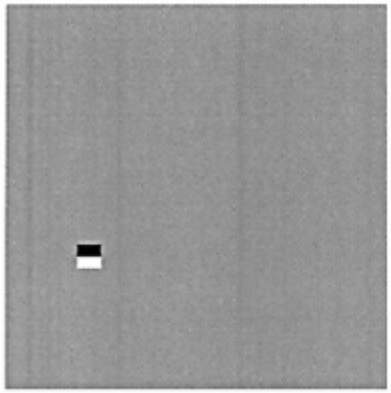


Fig. 31: Wavelet di Haar $\Psi^{[h]}_{2,3}(x,y)$ *al livello 2. I pixel neri indicano un valore negativo, i pixel bianchi un valore positivo, i pixel grigi zero.*

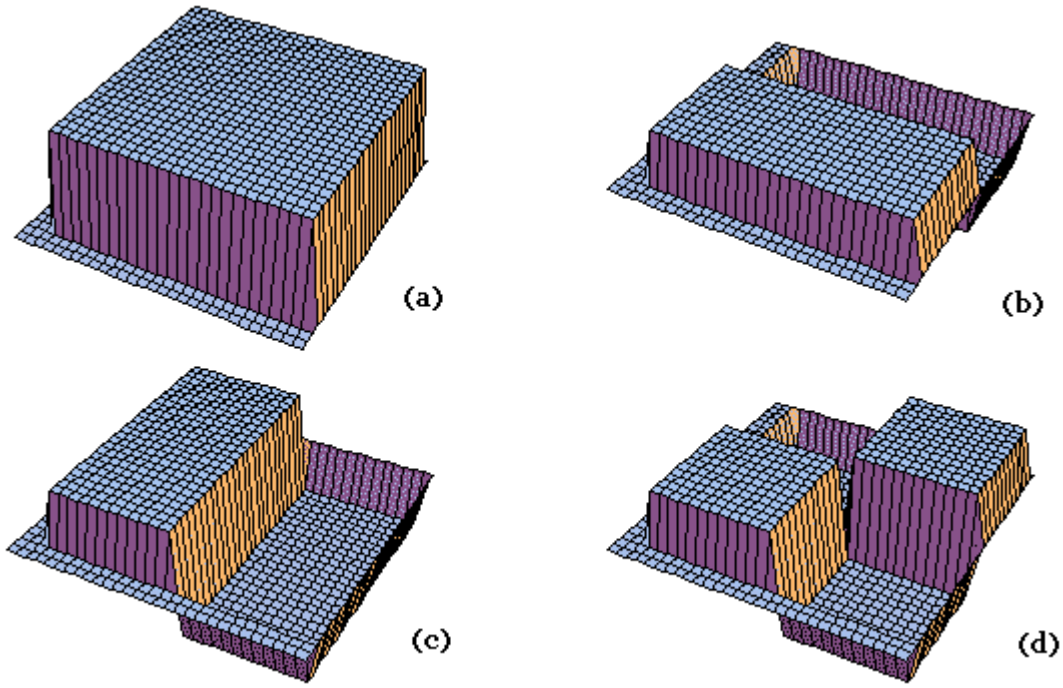


Fig. 32: Funzione di scaling e wavelets di Haar a due dimensioni. (a) $\Phi_{i,j}(x,y)$ *(b)* $\Psi^{[h]}_{i,j}(x,y)$ *(c)* $\Psi^{[v]}_{i,j}(x,y)$ *(d)* $\Psi^{[d]}_{i,j}(x,y)$.

Come risultato, l'estensione a due dimensioni degli algoritmi di decomposizione corrisponde all'algoritmo a una dimensione applicato alle due direzioni dell'immagine. Data un'immagine di

input $c^j(m,n)$ alla risoluzione j e di dimensioni $N \times N$, possiamo processarla lungo la direzione orizzontale: l'immagine viene decomposta riga per riga e, a causa del processo di decimazione, le due immagini risultanti sono larghe la metà.

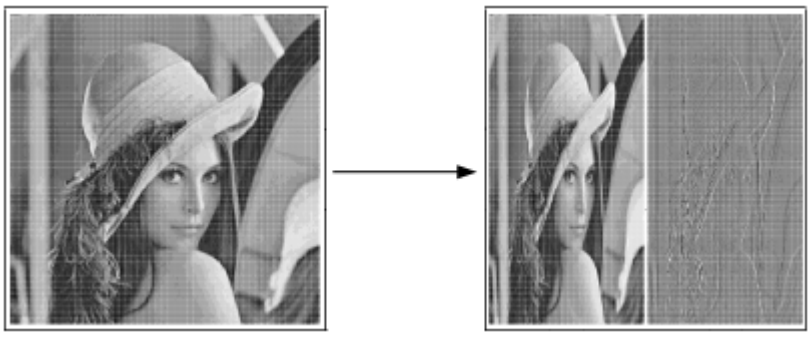


Fig. 33: Decomposizione riga per riga di Lena.

Le matrici che rappresentano le immagini vengono poi trasposte, e processate di nuovo riga per riga per ottenere quattro immagini quadrate $\frac{N}{2} \times \frac{N}{2}$, a risoluzione $j-1$.

$$a^{j-1}(m,n) \qquad h^{j-1}(m,n) \qquad v^{j-1}(m,n) \qquad d^{j-1}(m,n)$$

La procedura di trasposizione e elaborazione riga per riga equivale a un'elaborazione colonna per colonna. La trasformazione complessiva corrisponde al mapping

$$c^j \rightarrow \begin{pmatrix} h^{j-1} & d^{j-1} \\ a^{j-1} & v^{j-1} \end{pmatrix}$$

in cui le sotto-immagini h^{j-1}, d^{j-1} e v^{j-1} sono immagini di dettaglio e a^{j-1} è una versione a bassa risoluzione della c^j.

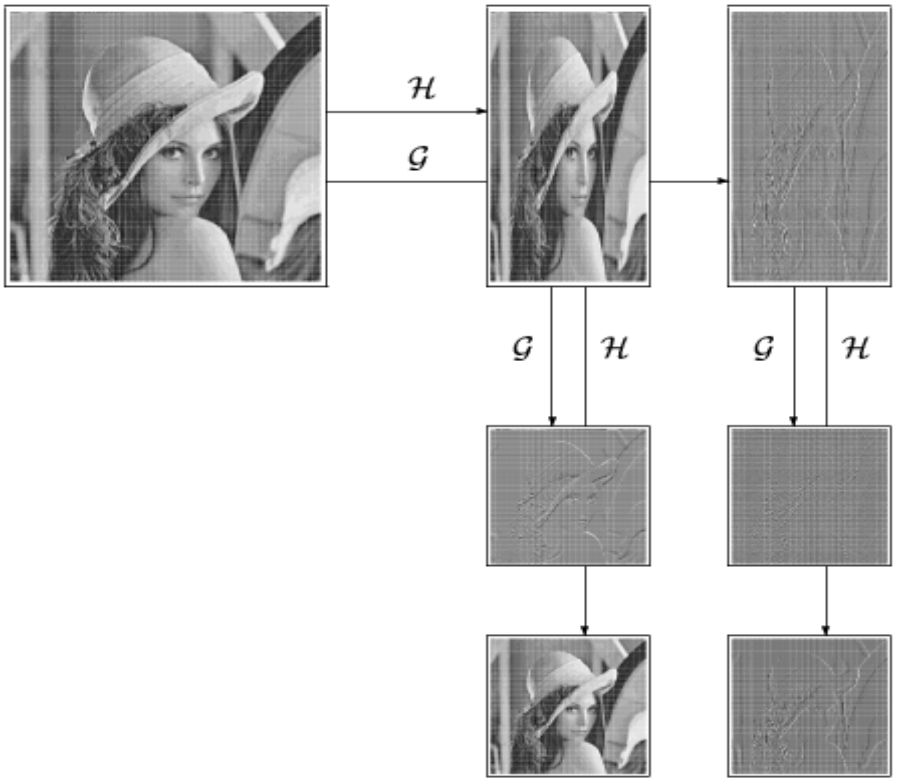


Fig. 34: I valori delle 3 sotto-immagini di dettaglio sono molto più piccoli di quelli della sotto-immagine a bassa risoluzione, quindi nella pratica per visualizzarle si ricorre all'utilizzo di una scala logaritmica.

In particolare le quattro sotto-immagini sono così formate:

h^{j-1}	Ottenuta calcolando i coefficienti di approssimazione sulle righe e le fluttuazioni sulle colonne, che saranno valori prossimi a zero: i *bordi orizzontali* vengono rilevati dalle fluttuazioni mentre i bordi verticali scompaiono.
v^{j-1}	Ottenuta calcolando le fluttuazioni sulle righe e operando la media mobile sulle colonne: rileva le *fluttuazioni verticali*.
d^{j-1}	Le fluttuazioni sono calcolate sulle righe e sulle colonne, quindi vengono esaltate le *caratteristiche diagonali*.
a^{j-1}	Versione a bassa risoluzione dell'immagine di partenza, ottenuta calcolando le medie mobili sulle righe e poi sulle colonne.

Questa procedura può essere applicata ricorsivamente sulle immagini $a^{\ell}(m,n)$, ed è

immediato osservare che il numero totale (comprensivo delle sotto-immagini di dettaglio) di coefficienti dopo la decomposizione sarà *sempre* uguale al numero iniziale di coefficienti, N^2 . Inoltre, il principio di conservazione dell'energia continua a essere soddisfatto e ad ogni ricorsione l'energia è soggetta a rilocazione [7].

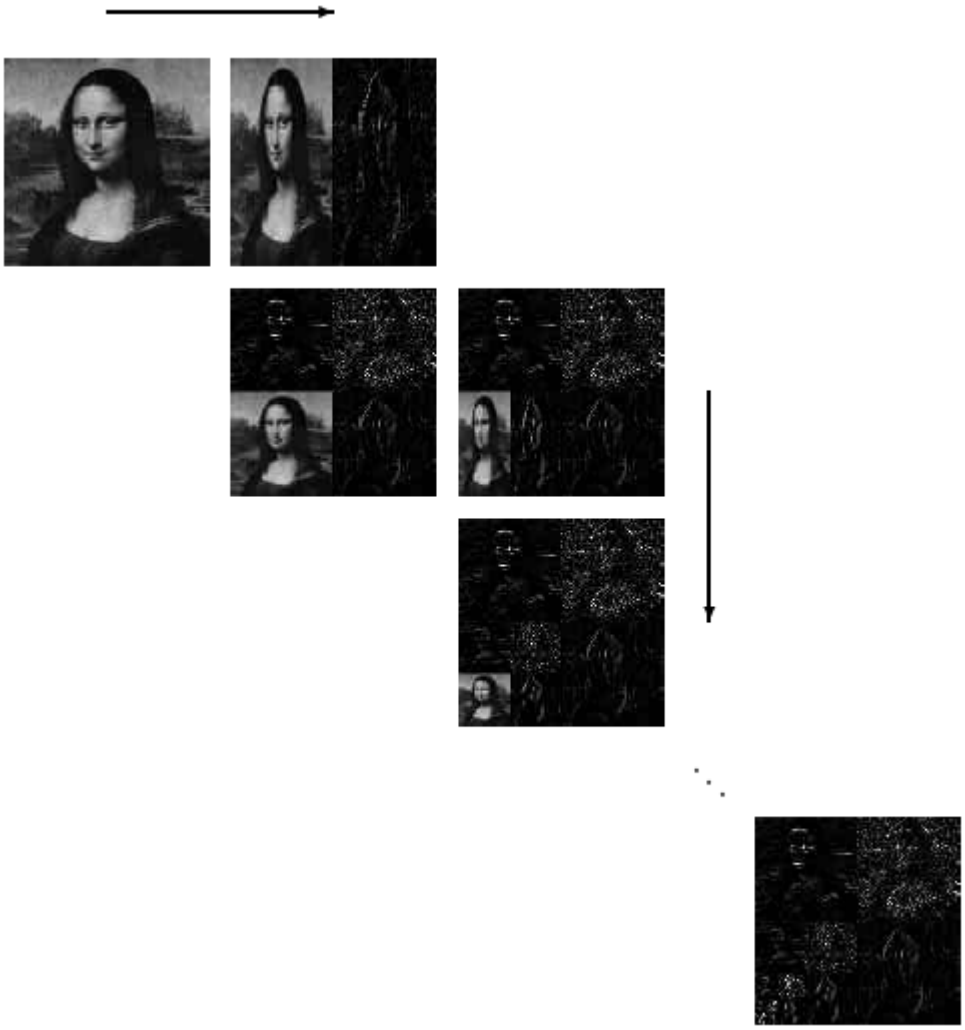


Fig. 35: Decomposizione non-standard di un'immagine.

Se, al termine della decomposizione, i coefficienti non vengono elaborati, l'immagine originale può essere ri-sintetizzata perfettamente tramite l'algoritmo di ricostruzione. La procedura è esattamente la stessa di quella di analisi, con la differenza che vengono usati i filtri di sintesi $\{g_0[k]\}$ e $\{g_1[k]\}$ al posto di quelli di analisi $\{h_0[k]\}$ e $\{h_1[k]\}$.

Questo procedimento è noto in letteratura come algoritmo di *decomposizione non-standard*

per immagini, ed è il metodo più largamente impiegato. Un secondo metodo di decomposizione è chiamato *decomposizione standard*, e consiste nell'applicare la trasformata wavelet monodimensionale riga per riga, *fino al livello di risoluzione desiderato*.

Fig. 36: Decomposizione standard: la trasformata viene prima applicata riga per riga fino alla risoluzione desiderata.

Dopodiché viene applicata la trasformata monodimensionale sulle colonne; secondo la notazione seguita finora, la trasformazione sulle colonne viene fatta ancora dal basso verso l'alto. I valori che risultano da questa operazione sono quasi tutti coefficienti di dettaglio, tranne una piccola zona che rappresenta il segnale mediato, e che si trova nel quadrante in basso a sinistra nell'immagine finale.

La decomposizione non-standard risulta leggermente più efficiente della seconda perché a ogni livello di trasformazione vengono elaborati un quarto (piuttosto che la metà, come nel secondo caso) dei punti dell'immagine del livello precedente. In particolare per un'immagine grande $M \times M$, la decomposizione standard richiede $4(m^2 - m)$ assegnazioni, e la non-standard ne richiede $(8/3)(m^2 - 1)$.

Si osservi che i due metodi di decomposizione forniscono coefficienti che vanno fatti corrispondere a due insiemi *diversi* di funzioni di base, uno per ognuno dei due metodi; la base relativa alla decomposizione non-standard è stata mostrata a inizio sezione. Chiaramente i due metodi danno risultati diversi, quindi la preferenza di uno dei due è dettata dal dominio applicativo.

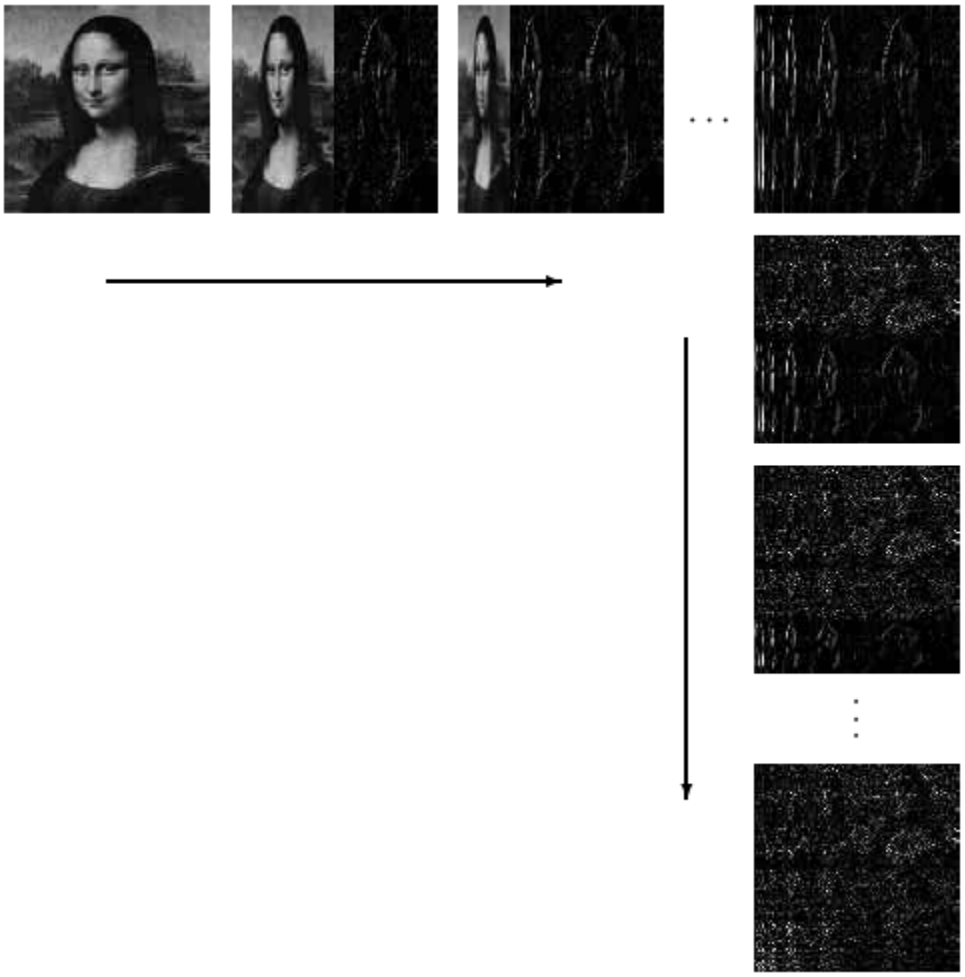


Fig. 37: Decomposizione standard di un'immagine.

Capitolo 2: Rilevamento dei contorni

Nell'ambito dell'analisi di immagini, il rilevamento dei bordi è una delle operazioni maggiormente utilizzate, e con ogni probabilità in letteratura sono presenti più algoritmi in merito rispetto a qualsiasi altro argomento.

Per *bordo* si intende il confine tra un oggetto e lo sfondo, o il confine tra oggetti sovrapposti. Questo significa che *se* è possibile identificare in maniera accurata tutti i contorni in un'immagine, allora tutti gli oggetti contenuti in essa possono essere localizzati, e proprietà di base come area, perimetro e forma possono essere calcolate.

Esistono diverse possibili definizioni per i *tipi* di bordo, ognuna delle quali si applica in circostanze specifiche. Una delle definizioni più comunemente usate è quella di bordo a *gradino ideale*, mostrato in fig. 38b. In questo caso il bordo è semplicemente un cambiamento di intensità (in una scala di livelli di grigio) che occorre in una posizione specifica. Più grande è il cambiamento di intensità, più facilmente sarà localizzabile il bordo; è chiaro che situazioni ideali di questo tipo sono molto improbabili nella pratica.

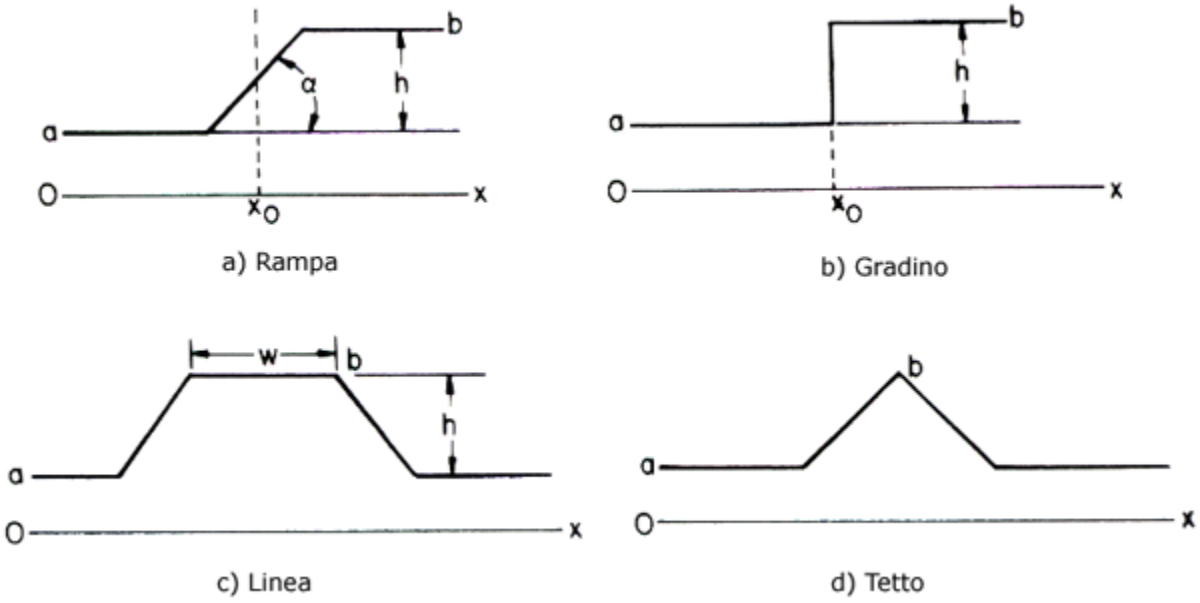


Fig. 38: Tipi di bordo

Vi sono tuttavia delle complicazioni. La prima è dovuta al processo di digitalizzazione: è poco

probabile che un'immagine venga campionata in maniera tale che tutti i bordi contenuti in essa corrispondano esattamente a un bordo netto di 1 o 2 pixel. In realtà, il cambiamento di intensità può estendersi su un numero non specificato di pixel, e in questa situazione la posizione effettiva del bordo può essere considerata essere il centro della rampa che porta l'intensità da un livello basso a uno alto (chiaramente la rampa menzionata è in senso matematico dal momento che dopo il campionamento si ottiene una "scalinata"). La seconda complicazione è rappresentata dal rumore, che può essere causato da molti fattori, tra cui il tipo del dispositivo di acquisizione, la lente utilizzata, i cambiamenti di luminosità, il movimento, la temperatura, gli effetti atmosferici, la polvere, o altro. A causa del rumore diventa altamente improbabile che due pixel che *dovrebbero* corrispondere alla stessa intensità di grigio, abbiano lo stesso livello nell'immagine. Ed essendo il rumore un fenomeno casuale, esso può essere caratterizzato solo statisticamente, e il risultato della sua azione sulle immagini è che di pixel in pixel si produce una variazione casuale del livello di intensità, cosicché le linee sinuose e le rampe che costituirebbero bordi *ideali* non vengono mai incontrati nella realtà.

2.1 Rilevatori classici

Esistono essenzialmente tre tipi comuni di operatori per la localizzazione dei bordi. Il primo tipo racchiude gli *operatori derivativi*, operatori progettati per identificare le locazioni in cui vi sono grandi cambiamenti di intensità. Il secondo tipo richiama gli schemi di *template-matching*, dove i contorni sono modellati da piccole immagini che esibiscono le proprietà astratte di bordi ideali. Infine vi sono gli operatori che fanno uso di un modello matematico per i contorni e per il rumore. Nelle sezioni successive verranno presentati alcuni tra gli operatori più comuni.

Tipicamente un task di estrazione dei bordi è formato da tre fasi principali e una fase opzionale:

1. Calcolo dell'intensità e dell'orientamento dei bordi, o *gradiente*
2. Rilevamento dei massimi locali
3. Binarizzazione o sogliatura
4. Miglioramento tramite elaborazione morfologica (opzionale)

Nel seguito verranno mostrati diversi metodi per svolgere ciascuno di questi passi. Il calcolo dell'intensità (*modulo*) e orientamento (*fase*) del gradiente viene effettuato mediante l'utilizzo di uno degli operatori spiegati nelle sezioni seguenti (fase 1). Dopodiché, si tratta di costruire l'immagine binaria che contenga la mappa finale dei contorni. Il modo più semplice e immediato per fare questo è applicare un semplice algoritmo di sogliatura (*thresholding*) marcando di bianco

ciò che è considerato *bordo* e lasciando in nero ciò che è considerato *sfondo*. Questa procedura però non dà ottimi risultati; tenendo presente il fatto che, come risultato dal calcolo del gradiente, i pixel di contorno hanno associata una direzione, occorre tenerne conto durante la fase di sogliatura. In particolare, quello che deve essere fatto è sopprimere i valori non-massimi presenti nel gradiente percorrendo i contorni nella direzione dello stesso (fase 2). Il risultato di questa operazione è che solo i massimi locali del gradiente vengono mantenuti: in altre parole, si è applicata una procedura di *thinning* dei bordi grossolani forniti dal gradiente, e si sono ottenuti contorni spessi al più *un* pixel. La loro posizione e contiguità dipendono ovviamente dall'algoritmo di estrazione. Dopo questa operazione, è possibile applicare un algoritmo di thresholding e finalmente costruire la mappa binaria finale (fase 3).

Nel seguito verranno anzitutto considerati i rilevatori classici; poi, nel capitolo 2.3, verrà presentato il sistema di estrazione dei contorni adottato da una delle applicazioni software realizzate per il lavoro di tesi.

2.1.1 Operatori derivativi

Dal momento che un bordo è definito da un certo cambiamento di intensità, è chiaro che un operatore sensibile a questo tipo di variazioni agirà come rilevatore. Ciò è esattamente quanto svolto da un derivatore; di fatti un'interpretazione di una derivata è il tasso di cambiamento dei valori di una funzione, e nelle immagini questo tasso è grande vicino ai bordi e piccolo in aree quasi costanti. Dal momento che le immagini sono segnali numerici a due dimensioni, è importante considerare i cambiamenti di livello in più direzioni. Per questa ragione, vengono usate le derivate parziali dell'immagine rispetto alle direzioni principali definite dalla x e dalla y. Una stima della direzione effettiva dei contorni può essere quindi ottenuta usando le derivate in x e in y come *componenti* della direzione effettiva lungo gli assi, e calcolando la somma vettoriale.

L'operatore di cui si parla è per l'appunto il gradiente, che è un campo vettoriale, qui definito per un'immagine $A(x, y)$:

$$\nabla A(x, y) = \left(\frac{\delta A}{\delta x}, \frac{\delta A}{\delta y} \right)$$

per cui ogni pixel è descritto da un vettore e in quanto tale contiene informazioni sulla sua intensità e sulla sua direzione. Matematicamente, queste due quantità sono date da:

$$G_{mag} = \sqrt{(\delta A / \delta x)^2 + (\delta A / \delta y)^2}$$

$$G_{dir} = atan2\left(\frac{\delta A}{\delta y}, \frac{\delta A}{\delta x}\right)$$

Ma poiché un'immagine è un segnale numerico, per poterla derivare occorre utilizzare risultati noti nell'ambito dell'elaborazione numerica dei segnali; nel seguito ne vengono mostrati alcuni.

2.1.2 Operatori basati su template

Come accennato, l'idea che sta dietro a questo tipo di rilevamento è quella di utilizzare un piccolo template (nella fattispecie, una piccola immagine) come modello di un bordo, e cercare all'interno dell'immagine da analizzare le regioni in cui si osserva una certa similarità con il template stesso (da cui *template-matching*).

Spesso il template è dato sotto forma di un "modellino" delle variazioni di intensità dei bordi, o come approssimazione di un derivatore. Di rilevatori di questo tipo ne esistono molti, e a dispetto dei loro svantaggi vengono tuttora utilizzati in diverse applicazioni grazie alla rapidità di calcolo che comportano. Tra i più comuni rilevatori troviamo l'operatore di Sobel.

Questo operatore consiste di una coppia di maschere di convoluzione. L'applicazione delle maschere equivale a una combinazione lineare dei pixel coperti dal loro supporto; di conseguenza, il calcolo delle immagini risultanti (cioè il gradiente orizzontale e il gradiente verticale, in altre parole la ripidità nelle due direzioni) avviene per il tramite di convoluzioni.

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

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

Un modo per avere un'intuizione di questi template, è vederli come un'approssimazione degli

operatori di derivazione calcolati nel pixel che corrisponde al centro della maschera. La componente x dell'operatore di Sobel è data dalla prima maschera, mentre la componente y è data dalla seconda: considerando queste componenti come componenti del gradiente è possibile ottenere modulo e fase per ogni pixel di bordo usando le relazioni viste nella sezione precedente.

Una volta ottenuto il modulo del gradiente, tipicamente si applica una forma di sogliatura per scartare i pixel troppo deboli (ad esempio dovuti al rumore) e considerare tutto il resto come contorno. Si osservi che la valutazione del gradiente richiede il calcolo di due quadrati e di una radice quadrata, operazioni piuttosto lente; per questo motivo, non è raro vedere utilizzata direttamente la somma dei moduli $|\delta A/\delta x|+|\delta A/\delta y|$ come approssimazione.

Gli operatori basati su template danno risultati migliori degli operatori derivativi, ma sono comunque troppo sensibili al rumore. Un modo per alleviare il problema consiste nell'espansione delle maschere utilizzate, ma una volta scelta una maschera, questa è di dimensione fissa lungo l'intero processing.

2.1.3 Rilevatore di Canny

Nel 1986, John Canny ha definito un insieme di obiettivi che un rilevatore di bordi ideale debba raggiungere, e ha descritto un metodo ottimale per perseguirli. I tre obiettivi principali sono i seguenti:

1. Un rilevatore dovrebbe rilevare *tutti e soli* i bordi (buon *rilevamento*)
2. La distanza tra i pixel di bordi identificati e i bordi effettivi dovrebbe essere la più piccola possibile (buona *localizzazione*)
3. Un rilevatore dovrebbe rilevare ogni bordo una sola volta (*risposta* singola)

Il rilevatore di Canny è un filtro il cui compito è quello di attenuare (ammorbidire) il rumore *e* di localizzare i bordi. Il problema è la ricerca di un filtro tale che ottimizzi i tre criteri sopra enunciati (l'espressione matematica dei tre criteri è piuttosto complicata e non è d'interesse in questa sede, per cui si rimanda a [13]). Per la complessità dei calcoli, non è stata trovata una soluzione analitica che ottimizzi i tre criteri, ma Canny ha fornito un'approssimazione efficiente del filtro sotto forma di *derivata prima di una Gaussiana*. Effettuando la convoluzione tra l'immagine di partenza e questo filtro, si ottiene l'immagine dei contorni. Uno dei grandi vantaggi

di questa procedura risiede anche nei tempi di calcolo: realizzare una convoluzione è semplice ma computazionalmente costoso, anche passando al dominio delle frequenze. Però, essendo la bivariata Gaussiana una funzione separabile, la convoluzione a due dimensioni può essere separata in due convoluzioni con due Gaussiane monodimensionali, una lungo le x e la seconda lungo le y .

Il metodo ottimale di Canny si svolge in una serie di passi. La prima fase richiede lo *smoothing* dell'immagine tramite un filtro Gaussiano: in questo modo si sfumano i contorni e si diminuisce l'influenza del rumore. Ciò è seguito dal calcolo del gradiente effettuando la convoluzione con la derivata della Gaussiana nelle direzioni orizzontale e verticale. Dopo aver soppresso i non-massimi come descritto nelle sezioni precedenti, occorre applicare una funzione di sogliatura: Canny suggerisce un nuovo meccanismo di thresholding, chiamato *isteresi*: questo schema fa uso di un limite superiore T_h e un limite inferiore T_ℓ . Se un pixel ha un valore di intensità maggiore di T_h , allora viene marcato come pixel di bordo; poi, ogni pixel connesso a questo viene valutato e se ha intensità maggiore di T_ℓ viene anche marcato come pixel di bordo, altrimenti viene scartato.

Le prestazioni dell'algoritmo di Canny dipendono pesantemente dalla deviazione standard σ del filtro Gaussiano e dai valori di soglia adottati nell'isteresi. Il valore σ controlla anche la dimensione del filtro; a un numero alto corrisponde un filtro di dimensioni grandi, il che implica più *blurring* (necessario per immagini rumorose) e bordi più grossolani. Ovviamente in questo modo la localizzazione dei bordi è meno accurata. A valori piccoli di σ d'altra parte corrispondono sia bordi più fini e meglio localizzati, che una maggiore sensitività al rumore. Dipendentemente dal livello di rumore quindi l'utilizzatore può scegliere la dimensione più adatta per il filtro.

Nelle sezioni successive vengono presentati dei sistemi di estrazione dei contorni che traggono vantaggio dal rilevamento dei bordi a risoluzioni multiple. In questi sistemi il parametro σ viene in altre parole fatto variare lungo una sequenza particolare di valori (valori detti *diadici*), il che risulta in versioni dell'immagine a diverse risoluzioni. Combinando le informazioni sui massimi a risoluzioni differenti, è possibile ottenere una classificazione migliore dei bordi veri dell'immagine. Chiaramente, in questi sistemi, i tempi di calcolo aumentano considerevolmente, per cui spesso in domini real-time si preferiscono realizzazioni hardware a soluzioni software [14].

2.2 Rilevamento basato su wavelet

In sez. 2.1 si è parlato brevemente delle tecniche tradizionali per il rilevamento dei bordi. Questo task può trarre ulteriori vantaggi da una rappresentazione a più scale: versioni a

risoluzioni diverse dell'immagine in analisi possono essere impiegate con profitto per descrivere la varietà di strutture di bordi presenti. Queste descrizioni in multiscala possono essere dunque sintetizzate per la costruzione di una mappa dei contorni.

La trasformata wavelet (sez. 1.4) consente implicitamente di effettuare l'analisi multirisoluzione di segnali, e nel corso degli anni sono state proposte diverse tecniche per il rilevamento dei bordi basate sulle proprietà delle wavelet. In particolare, in un dominio DWT, si osserva che le strutture di interesse (bordi e contorni) vengono preservate al variare della risoluzione, mentre il rumore presente decresce rapidamente lungo le scale.

L'approccio di base che viene adottato e che sfrutta una rappresentazione siffatta consiste nel moltiplicare i risultati ottenuti in ciascuna scala per affilare i bordi e al contempo diluire il rumore. Però, questo semplice approccio può dare risultati non sempre soddisfacenti; nel seguito vengono esposte alcune tecniche alternative per il rilevamento motivato dalle wavelet e dall'analisi multirisoluzione.

2.2.1 Metodo di Mallat

Il metodo di Mallat è l'approccio tradizionale al riconoscimento e l'estrazione dei contorni tramite wavelet. I passi principali sono evidenziati nel diagramma seguente:

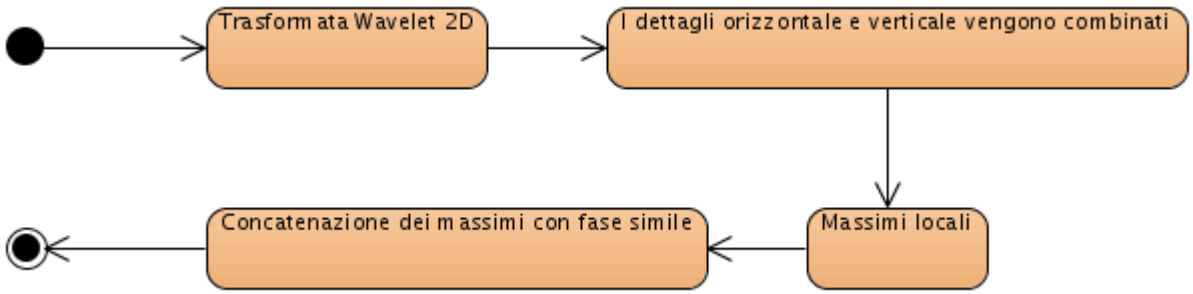


Fig. 39: Diagramma di flusso del metodo di Mallat

L'algoritmo procede come segue. Una volta effettuata la trasformazione wavelet dell'immagine (usando come wavelet la derivata di una bivariata Gaussiana, ricalcando quanto proposto da Canny), si ottiene come noto un insieme di 4 immagini, che corrispondono ai bordi orizzontali, ai bordi verticali, e ai bordi diagonali dell'immagine di partenza, insieme a una sua versione a risoluzione dimezzata (sez. 1.6). La decomposizione può essere applicata per il numero di livelli desiderato, ottenendo in questo modo un'analisi in multiscala dell'immagine. A

questo punto solo i dettagli orizzontale e verticale sono presi in considerazione: l'immagine del modulo massimo alla scala $s=2^j$ (indicato con $M_s f(x,y)$) combina le due sotto-immagini in questo modo:

$$M_s f(x,y)=\sqrt{|W_s^1 f(x,y)|^2+|W_s^2 f(x,y)|^2}$$

dove $W_s^1 f(x,y)$ e $W_s^2 f(x,y)$ sono le componenti della trasformata wavelet di $f(x,y)$ alla scala s, mentre la fase ($A_s f(x,y)$) viene calcolata usando:

$$A_s f(x,y)=atan2\left(W_s^2 f(x,y), W_s^1 f(x,y)\right)$$

Un metodo alternativo, che dà tra l'altro risultati migliori [6], prevede il calcolo della DWT inversa (sez. 1.4.2.2) a partire dalle sotto-immagini orizzontale, verticale e diagonale.

Infine, per ottenere una mappa binaria dei contorni (con bordi spessi al più 1 pixel), vengono utilizzate le informazioni dal modulo e dalla fase in questo modo: un pixel è considerato pixel di bordo se il suo modulo massimo è maggiore dei due pixel vicini lungo la direzione del gradiente in quel punto. D'altro canto, un pixel ha solamente 8 vicini *immediati*: per sopperire a questa limitazione, gli angoli da 0 a 2π vengono divisi in sezioni di 45° come da figura:

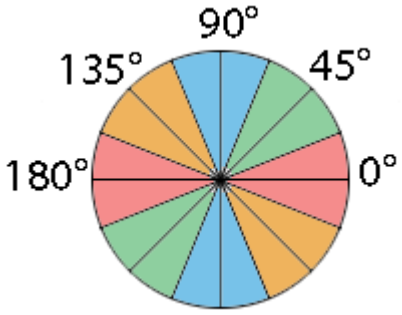


Fig. 40: Gli angoli compresi in ogni zona vengono approssimati a un multiplo di 45°, per un totale di 4 orientamenti diversi

Per rilevare solo le caratteristiche più salienti è pensabile marcare come pixel di bordo quei

pixel che, oltre a essere massimi locali, hanno un'intensità superiore a una certa soglia. Ora a partire dall'immagine formata da questi massimi locali, possono essere costruite delle “catene” che definiscano i contorni finali: per ogni massimo locale, i punti vicini che abbiano una fase molto simile (anche se di modulo piuttosto basso) vengono collegati ad esso.

Questo metodo è relativamente libero dall'influenza del rumore e dà risultati soddisfacenti [11]: ad esempio, l'operatore di Sobel, o altri simili come quello di Laplace, utilizzando finestre piccole, sono influenzati in misura molto maggiore dal rumore di tipo “sale e pepe” (un tipo di rumore che occorre sotto forma di pixel bianchi e neri sparsi in maniera casuale, e molto comune nelle immagini). Inoltre, la proprietà di scaling delle wavelet offre la possibilità di rilevare bordi a scale diverse; le tecniche convenzionali (sez. 2.1) non hanno informazioni sulla scala a cui operano dal momento che consistono di operatori di dimensioni fisse, mentre la trasformata wavelet discreta fornisce un'analisi multirisoluzione per definizione. In più gli effetti del rumore vengono dispersi, grazie alla proprietà di compattezza dell'energia [7], lungo le scale e su diversi coefficienti, mentre i bordi rimangono codificati da componenti singole (si veda anche la sezione successiva). Si osservi però che il metodo presentato non prevede una sintesi finale a partire dai contorni estratti a diverse risoluzioni. È semplicemente un *modo* per rilevare i contorni a partire dalle informazioni fornite dai coefficienti wavelet a seguito di una trasformazione, per una scala fissa. È chiaro che a diverse risoluzioni si otterranno (anche dipendentemente dal tipo di immagine) bordi differenti: negli anni sono state proposte diverse soluzioni per sintetizzare una mappa finale dei contorni a partire dalle informazioni a scale diverse. Nel seguito verranno presentati alcuni di questi approcci.

2.2.2 WMC

Questo metodo (per esteso *Wavelet maxima chain edge detection method*) è un esempio piuttosto semplice e diretto di come utilizzare le informazioni prese da scale diverse per ottenere un'unica mappa dei contorni.

All'immagine in ingresso viene applicata una versione modificata della trasformata wavelet discreta nota come *Redundant Wavelet Transform* (RWT) o *Stationary Wavelet Transform* (SWT): il segnale non viene sottoposto a downsampling lungo la decomposizione (sez. 1.4.2.1) e quindi durante l'intero processing vengono mantenute molte informazioni ridondanti, da cui risulta che la trasformata finale contiene il doppio dei campioni in ingresso. Questa trasformata viene applicata all'immagine fino al livello desiderato, usando wavelet con pochi momenti svanenti (ad esempio Haar) in modo tale da rilevare le transizioni nette dei livelli di intensità. A ogni scala, viene calcolato modulo e fase del gradiente come in sez. 2.1.1; dopodiché vengono marcati come pixel di bordo i punti che sono massimi locali lungo la direzione del gradiente, così

a ogni scala. I bordi sono poi formati concatenando i massimi *vicini* (secondo un criterio di vicinanza, ad esempio entro una massima distanza di 2 pixel) con ampiezza e fase simili. Ora vengono presi in considerazione i risultati ottenuti a ogni scala: i massimi la cui ampiezza rimane relativamente costante lungo tutte le scale considerate sono mantenuti, mentre gli altri vengono scartati.

Il metodo WMC è piuttosto rapido e accurato, e trova molte applicazioni in ambito medico, ad esempio per il rilevamento di nuclei, globuli e corpuscoli [12], ma richiede una certa interazione umana; per questo motivo, l'impiego del WMC non è indicato in molte altre applicazioni, dove è necessaria una localizzazione più precisa dei bordi vicini e dove è preferito un approccio non supervisionato al rilevamento.

Un diagramma di flusso del metodo presente permette di averne una visione più immediata:

Fig. 41: Diagramma di flusso del metodo WMC

2.2.3 Moltiplicazione tra scale

Zhang e Bao [6] propongono uno schema di rilevamento basato sulla trasformata wavelet discreta (DWT) e la moltiplicazione tra scale. Dal momento che esistono molte similarità spaziali tra sotto-bande di wavelet, viene definita una funzione di prodotto tra scale come la moltiplicazione dei coefficienti wavelet di due scale adiacenti, in modo da amplificare i contorni e attenuare il rumore. Dopodiché i bordi vengono determinati come massimi locali del prodotto, differentemente da quanto fatto in tecniche simili dove il prodotto tra scale viene effettuato *dopo* la fase di binarizzazione. Una volta svolto il prodotto, bordi e rumore possono essere distinti più facilmente e dunque si può utilizzare un metodo di sogliatura a livello singolo (differentemente dall'isteresi di Canny) per sopprimere il rumore. Questo procedimento permette di ottenere

risultati di gran lunga migliori di quelli ottenibili da una delle due scale separatamente, in particolar modo per quanto riguarda la localizzazione.

La DWT può qui essere progettata come un rilevatore in multiscala equivalente a quello di Canny. Sia $\phi(x)$ una funzione di smoothing più volte derivabile e sia definita la wavelet $\psi(x)$ come sua derivata prima:

$$\psi(x)=\frac{d\,\phi(x)}{dx}$$

Allora la DWT di un segnale $f(x)$ alla scala 2^j e alla posizione x (sez. 1.4) è:

$$W_j f(x)=f*\psi_j(x)=f*\left(2^j\frac{d\,\phi_j}{dx}\right)(x)=2^j\frac{d}{dx}(f*\phi_j)(x)$$

La $W_j f(x)$ è proporzionale alla derivata prima di $f(x)$, "ammorbidita" dalla funzione $\phi_j(x)$. Quando la $\phi(x)$ è una Gaussiana, la determinazione degli estremi locali di $W_j f(x)$ è equivalente al rilevamento di Canny. Gli autori di [6] prendono come funzione di smoothing una spline cubica, ottenendo così come wavelet una spline quadratica. Le due funzioni approssimano rispettivamente in maniera molto vicina una Gaussiana e la sua derivata prima. Dunque anche in questo caso la DWT è equivalente al rilevamento di Canny.

A due dimensioni sono necessarie due wavelet, una per la direzione orizzontale e l'altra per la direzione verticale. Con $\phi(x,y)$ funzione di smoothing a due dimensioni (integra a 1 e converge a 0 all'infinito), si ha:

$$\psi^1(x,y)=\frac{\delta\,\phi(x,y)}{\delta\,x}\qquad\qquad\psi^2(x,y)=\frac{\delta\,\phi(x,y)}{\delta\,y}$$

per cui la trasformata wavelet di $f(x,y)$ alla scala 2^j e posizione (x,y) ha due componenti:

$$W_j^1 f(x,y)=f*\psi_j^1(x,y)\qquad\qquad W_j^2 f(x,y)=f*\psi_j^2(x,y)$$

Il prodotto tra scale adiacenti per la funzione $f(x)$ è definito dunque in questo modo (lungo una sequenza diadica):

$$P_j^f(x) = W_j f(x) W_{j+1} f(x)$$

dove il pedice j indica la scala. Un risultato importante legato all'analisi multirisoluzione e alle proprietà delle wavelet [6] risiede nel fatto che i salti (le discontinuità) presenti in un segnale tendono a propagarsi lungo le scale, mentre il rumore di tipo Gaussiano si dimezza (circa) nel passare da una scala a un'altra più bassa. Quindi moltiplicando direttamente le DWT a scale adiacenti si ottiene come risultato una forma di "diluizione" del rumore.

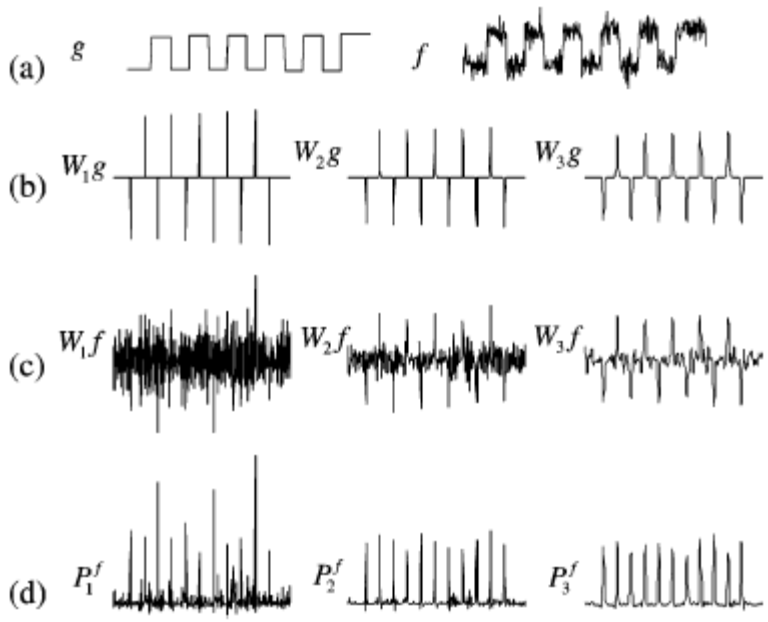


Fig. 42: a) Onda quadra e sua versione affetta da rumore b) Trasformate wavelet dell'onda quadra c) Trasformate wavelet dell'onda affetta da rumore d) Prodotti tra trasformate a scale adiacenti dell'onda affetta da rumore

Consideriamo un'onda quadra affetta da rumore (fig. 42a); applicando la trasformata wavelet più volte, si ottengono i segnali in fig. 42b. A scale piccole, le posizioni dei gradini sono ben localizzate, ma sono praticamente sommerse dal rumore; a scale grandi d'altro canto, il rapporto segnale-rumore migliora e i bordi possono essere rilevati più facilmente, anche se localizzati con accuratezza minore (fig. 42c). In fig. 42d si vede che il prodotto tra due DWT adiacenti permette di godere dei vantaggi di entrambe le rappresentazioni.

Una volta ottenuto il prodotto, bordi (i massimi locali di P_j^f) e rumore possono essere distinti più agevolmente; applicando una soglia scelta in maniera appropriata si potrebbe arrivare ai risultati desiderati. Dal momento che un bordo significativo in x_0 compare in entrambe le scale con lo stesso segno, $P_j^f(x_0)$ sarà un valore non-negativo. Se $P_j^f(x)$ è minore di zero, il punto può essere considerato rumore e quindi può essere filtrato. In [6] viene proposta una funzione di sogliatura che dipende dalla norma delle due wavelet alle scale j e $j+1$, e da una costante moltiplicativa fissata; in questo modo diminuisce il numero di parametri di cui tenere conto e si favorisce una realizzazione *non supervisionata* dell'algoritmo di rilevamento.

A due dimensioni, il prodotto tra scale va definito sulle ascisse e sulle ordinate:

$$P_j^{f,1}(x,y) = W_j^1 f(x,y) \cdot W_{j+1}^1 f(x,y)$$
$$P_j^{f,2}(x,y) = W_j^2 f(x,y) \cdot W_{j+1}^2 f(x,y)$$

Dopo aver impostato a zero i punti con $P_j^{f,1}(x,y) < 0$ o $P_j^{f,2}(x,y) < 0$, il modulo e l'angolo per ogni punto (x,y) sono definiti come:

$$M_j f(x,y) = \sqrt{P_j^{f,1}(x,y) + P_j^{f,2}(x,y)}$$
$$A_j f(x,y) = atan\left(\frac{sgn(W_j^2 f(x,y)) \cdot \sqrt{P_j^{f,2}(x,y)}}{sgn(W_j^1 f(x,y)) \cdot \sqrt{P_j^{f,1}(x,y)}} \right)$$

Ora un punto di bordo è preso come massimo locale di $M_j f(x,y)$ nella direzione del gradiente (data da $A_j f(x,y)$). La mappa $M_j f(x,y)$ va poi sottoposta a sogliatura per eliminare il rumore, adottando una specifica funzione di soglia [6].

Un rilevamento di questo tipo esibisce ottime prestazioni. La moltiplicazione tra scale permette di migliorare l'accuratezza di localizzazione mantenendo un'alta efficienza per il rilevamento. Inoltre, la localizzazione di bordi vicini permette di avere, con la giusta scelta delle scale di partenza e di arrivo, buonissimi risultati.

Vediamo una visione schematica di questo metodo, come fatto per i metodi precedenti:

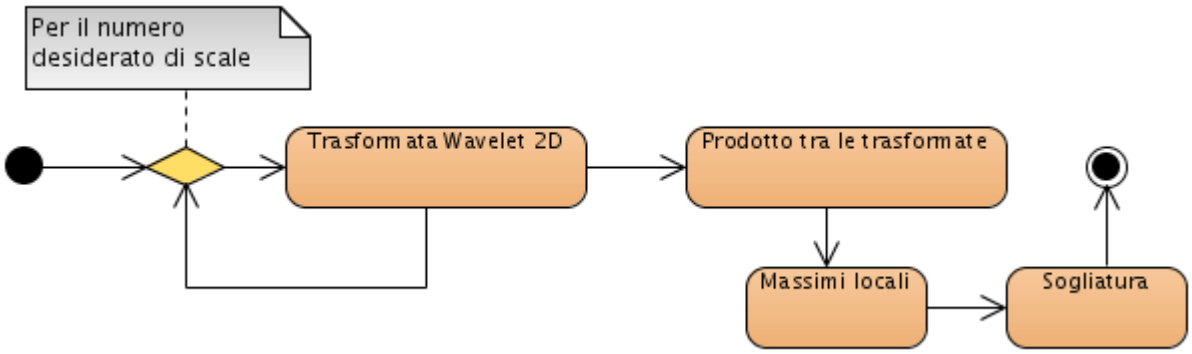


Fig. 43: Diagramma di flusso del metodo di moltiplicazione tra scale

2.2.4 Tracking dei contorni

In [8] viene proposto un approccio di estrazione dei bordi che combina una forma di rilevamento basata su gradiente e un metodo di *tracking* in multirisoluzione basato sulle wavelet.

L'immagine di partenza viene anzitutto decomposta in bande di frequenza, secondo i paradigmi dell'MRA (sez. 1.3). Dalla versione ad alta risoluzione vengono estratti dunque i bordi tramite un filtro basato sul contesto; infine, un componente chiamato *tracker* rifinisce i bordi rilevati basandosi sulle informazioni dei gradienti alle diverse scale.

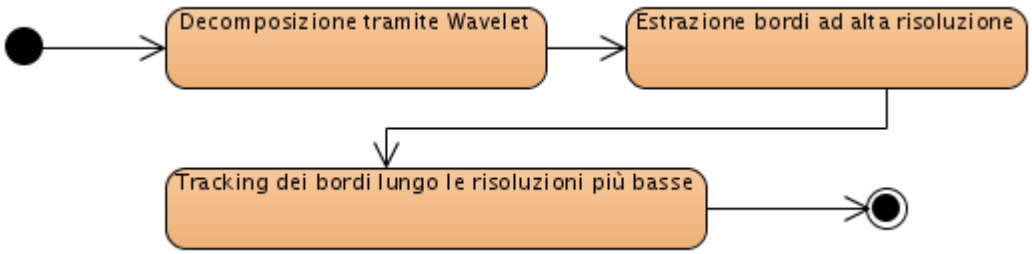


Fig. 44: Diagramma di flusso del metodo di tracking dei contorni

Il rilevatore opera estraendo in primo luogo i bordi dal gradiente ad alta risoluzione; l'immagine risultante viene quindi migliorata adottando due strategie. La prima strategia consiste nell'applicazione di un *filtro contestuale*: una finestra quadrata (es. 7x7) viene fatta scorrere su ogni pixel; se il pixel è un massimo locale e il suo valore *relativo* è più grande di un livello fissato, allora è marcato come punto di bordo. Il valore relativo corrisponde alla massima

differenza di intensità tra il pixel considerato e gli altri pixel nella finestra; un pixel marcato come punto di bordo è chiaramente un massimo locale lungo *almeno* una direzione. Ovviamente un'elaborazione di questo tipo dipende pesantemente dal valore di soglia scelto, tra l'altro difficile da determinare in maniera automatica. Per ottenere risultati migliori si può adottare una strategia a due livelli: vengono utilizzate due soglie T_ℓ e T_h (con $T_\ell < T_h$) per generare due mappe di contorni, I_ℓ e I_h. I_ℓ conterrà molti bordi non significativi, cioè bordi erroneamente marcati come tali, mentre I_h conterrà relativamente pochi contorni (chiaramente I_ℓ è sempre contenuta in I_h). A questo punto I_ℓ e I_h possono essere combinate per ottenere un insieme di bordi più ragionevole: la strategia adottata consiste nel comporre il risultato prendendo i segmenti di contorni di I_ℓ che hanno almeno un punto in I_h. Per ridurre il numero di parametri si può esprimere T_ℓ in funzione di T_h (o viceversa), ad esempio ponendo $T_\ell = 0.4\,T_h$ [8].

Questo tipo di rilevatore non riesce comunque a garantire in tutti i casi buoni risultati: c'è sempre una dipendenza da valori di soglia, da cui dipende ovviamente a sua volta la mappa dei contorni prodotta. L'idea è quindi quella di usare soglie piuttosto alte per prendere pochi (ma praticamente *sicuri*) contorni, e recuperare i restanti tramite un metodo di *tracking* in multiscala (appresso mostrato).

Come noto, le informazioni sui bordi vengono propagate tra le scale; bordi spezzati a scale basse (alta risoluzione) potrebbero essere invece connessi (contigui) a scale alte. Perciò, una volta estratti dei bordi “significativi” tramite il filtro contestuale, si può pensare di connettere frammenti di contorni, e costruirne di nuovi, utilizzando in maniera appropriata le informazioni presenti a scale diverse. Nell'ambito della Machine Vision il concetto di *tracking* si riferisce all'”inseguimento” di determinate caratteristiche, pixel, entità lungo percorsi nello spazio, nel tempo, oppure, come questo è il caso, lungo scale diverse. Prima di definire un metodo di tracking occorre definire le entità da inseguire: qui le feature prescelte sono i punti terminali dei contorni estratti nella prima fase. Si vuole cioè osservare l'evoluzione dei bordi lungo una sequenza diadica di risoluzioni, e costruire nuovi contorni in seguito a “scoperte” fatte in versioni via via più grossolane dell'immagine di partenza.

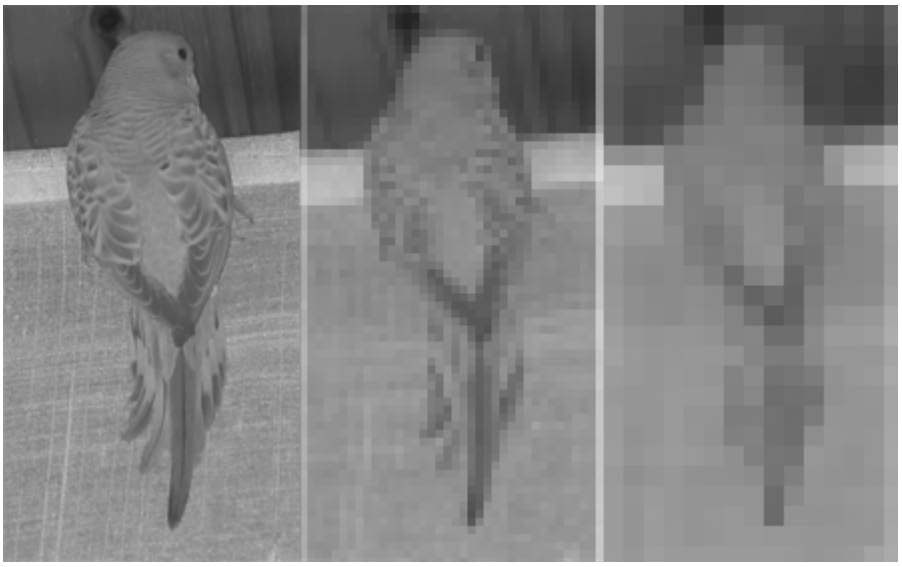


Fig. 45: Immagine a risoluzione decrescente. I contorni rilevati ad alta risoluzione vengono via via cercati nelle versioni più grossolane dell'immagine.

A partire dalla mappa dei contorni risultante da una sogliatura alta, i punti che hanno esattamente *un* vicino (nelle 8 possibili direzioni) sono marcati come end-point. Per rapidità di calcolo è anche pensabile tracciare solo quei punti che hanno un gradiente elevato.

Sia ora p_i^ℓ un punto nel gradiente alla scala ℓ. $P(p_i^\ell)$ è il "padre" di p_i^ℓ alla scala $\ell+1$ (più grossolana), e sia p_j^ℓ un punto adiacente a p_i^ℓ. Se p_i^ℓ è un massimo locale, allora definiamo $M(p_i^\ell)=1$, altrimenti $M(p_i^\ell)=0$. L'indicatore $C(p_i^\ell, p_j^\ell)$ dice invece se p_j^ℓ è stato selezionato per connettere p_i^ℓ dopo il tracking:

$$C(p_i^\ell, p_j^\ell) = M(p_i^\ell) \wedge M(P(p_i^\ell)) \wedge M(p_j^\ell) \wedge M(P(p_j^\ell))$$

Il metodo di tracking funziona "predicendo" bordi possibili man mano che si scende con la risoluzione. Assumiamo che due punti di massimo p_i^ℓ e p_k^ℓ non siano adiacenti. Se i loro corrispondenti alla scala $\ell+1$ sono anche dei massimi e sono adiacenti, allora assumiamo che alla scala ℓ esista un percorso tra p_i^ℓ e p_k^ℓ. Indichiamo questa evenienza con $B(p_i^\ell, p_k^\ell)=1$; in particolare definiamo:

$B(p_i^\ell, p_k^\ell)=C(p_i^\ell, p_k^\ell)$ se p_k^ℓ è adiacente a p_i^ℓ

$B(p_i^\ell, p_k^\ell)=M(p_i^\ell)\wedge M(p_k^\ell)\wedge B(P(p_i^\ell), P(p_k^\ell))$ altrimenti

Il tracking può essere fatto partire dalla scala a risoluzione migliore, dove tutti i contorni sono presenti (anche quelli non desiderati), ricordando comunque che è preferibile usare valori alti per le soglie T_ℓ e T_h nella strategia di estrazione iniziale dei contorni. Dopo aver valutato le feature iniziali (gli end-point), per ognuna di esse viene valutato l'indicatore $B(.)$ in una finestra quadrata centrata su di esso. Ad esempio, per il punto p_i^ℓ si controlla il punto adiacente p_j^ℓ ; se $C(p_i^\ell, p_j^\ell)=0$ allora si effettua la ricerca alle scale più alte, fino a quando eventualmente non si trova un punto p_k^ℓ tale che $B(p_i^\ell, p_k^\ell)=1$. Quando questo accade, il punto adiacente a p_i^ℓ nel percorso tra p_i^ℓ e p_k^ℓ viene selezionato per connettere p_i^ℓ (cioè l'indicatore $C(.)$ viene posto a 1).

Nella fase di edge tracking, se le feature iniziali si perdono o vengono fuse tra di loro nelle immagini a bassa risoluzione, il tracciamento fallisce. Per questo motivo conviene in genere partire da un'immagine ad alta risoluzione, e poi salendo coi livelli inseguire i bordi per recuperare quelli persi per via della sogliatura: in questo modo si evita l'influenza del rumore e si risolvono i problemi dei contorni spezzati.

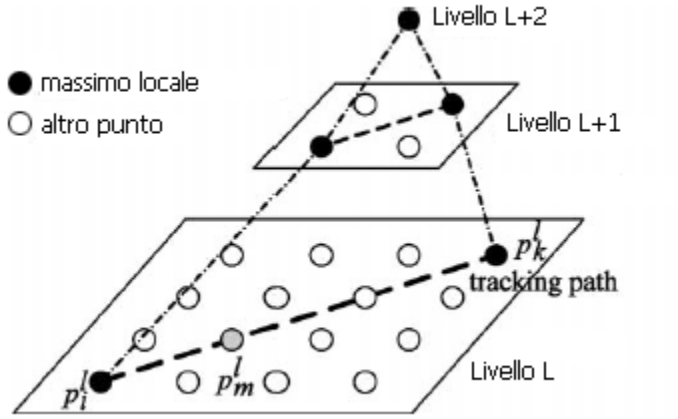


Fig. 46: Tracking dei contorni in multiscala

Questo sistema di rilevamento dei bordi (filtro contestuale + edge tracking) può essere adattato per diverse applicazioni semplicemente variando le soglie T_ℓ e T_h . Gli autori di [8] hanno però ottenuto cattivi risultati nel rilevamento e nella localizzazione precisa di bordi vicini (si osservi la fig. 47).

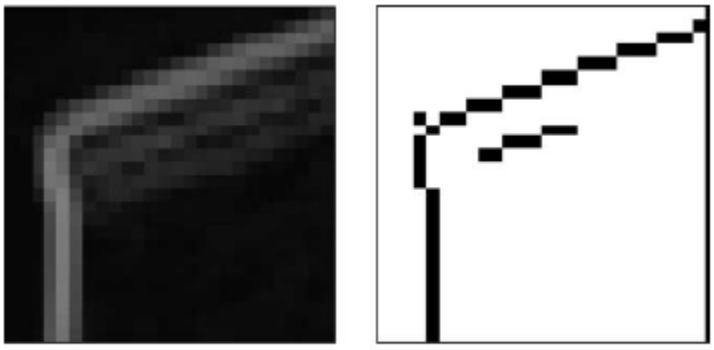


Fig. 47: a) Particolare del gradiente di un'immagine b) I bordi vicini non vengono rilevati correttamente

2.3 Rilevamento in multirisoluzione con soppressione delle texture

In questa sezione viene presentato un sistema per il rilevamento di contorni che trae beneficio da alcuni risultati sulla fisiologia umana, grazie ai quali è possibile la messa a punto di un meccanismo efficiente di inibizione di aree ricche di dettaglio comunemente chiamate *texture*. Il sistema mostrato di seguito è stato, nell'ambito di questa tesi, realizzato come applicazione C/C++. Nel capitolo 5 vengono inoltre dati i risultati di esperimenti condotti su una serie di immagini, per valutarne l'efficienza quanto l'efficacia in diversi domini applicativi.

2.3.1 Rilevamento a risoluzione fissa

Sia $I(x, y)$ un segnale numerico rappresentante l'intensità di un'immagine di partenza (in generale affetta da una qualche forma di rumore). Vogliamo ottenere il segnale $b_\sigma(x, y)$, che rappresenti in forma binaria i contorni di $I(x, y)$. L'estrazione dei contorni consiste nell'applicazione di una serie di passi, che ricalcano quelli visti nelle sezioni precedenti, con l'aggiunta di una fase di *soppressione delle texture* di cui si parlerà nelle sezioni a seguire.

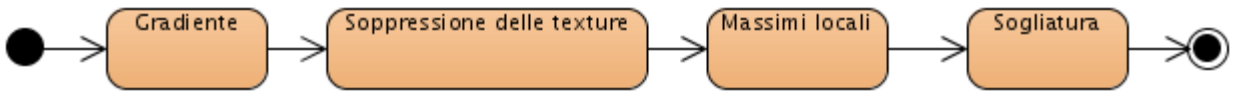


Fig. 48: Diagramma di flusso del rilevamento a risoluzione fissa

Si noti che viene usato il pedice σ per indicare la dipendenza delle quantità e degli operatori definiti appresso dal parametro di scala (risoluzione) σ .

2.3.1.1 Calcolo del gradiente

Il primo passo consiste nel calcolo del gradiente dell'immagine di partenza $I(x,y)$. Quando si usa il metodo delle differenze con finestre piccole (sez. 2.1), il gradiente è suscettibile al rumore e agli effetti di discretizzazione. Per diminuire queste influenze, si usa prima applicare qualche tipo di smoothing; procediamo come proposto da Canny (sez. 2.1.3):

$$\nabla_\sigma I(x,y) = \nabla\{I * g_\sigma\}(x,y) = \begin{bmatrix} \{I * \frac{\delta g_\sigma}{\delta x}\}(x,y) \\ \{I * \frac{\delta g_\sigma}{\delta y}\}(x,y) \end{bmatrix}$$

dove l'immagine è convoluita con le derivate parziali di g_σ , una bivariata Gaussiana analiticamente ben definita e grazie alla quale non è necessario calcolare le differenze parziali:

$$g_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

$$\frac{\delta g_\sigma(x,y)}{\delta x} = -\frac{x}{2\pi\sigma^4} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \qquad \frac{\delta g_\sigma(x,y)}{\delta y} = -\frac{y}{2\pi\sigma^4} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Ora modulo e fase del gradiente sono date dalle relazioni note. All'aumentare del valore di σ , cioè per scale più alte, si ottengono versioni via via più sfumate del modulo del gradiente.

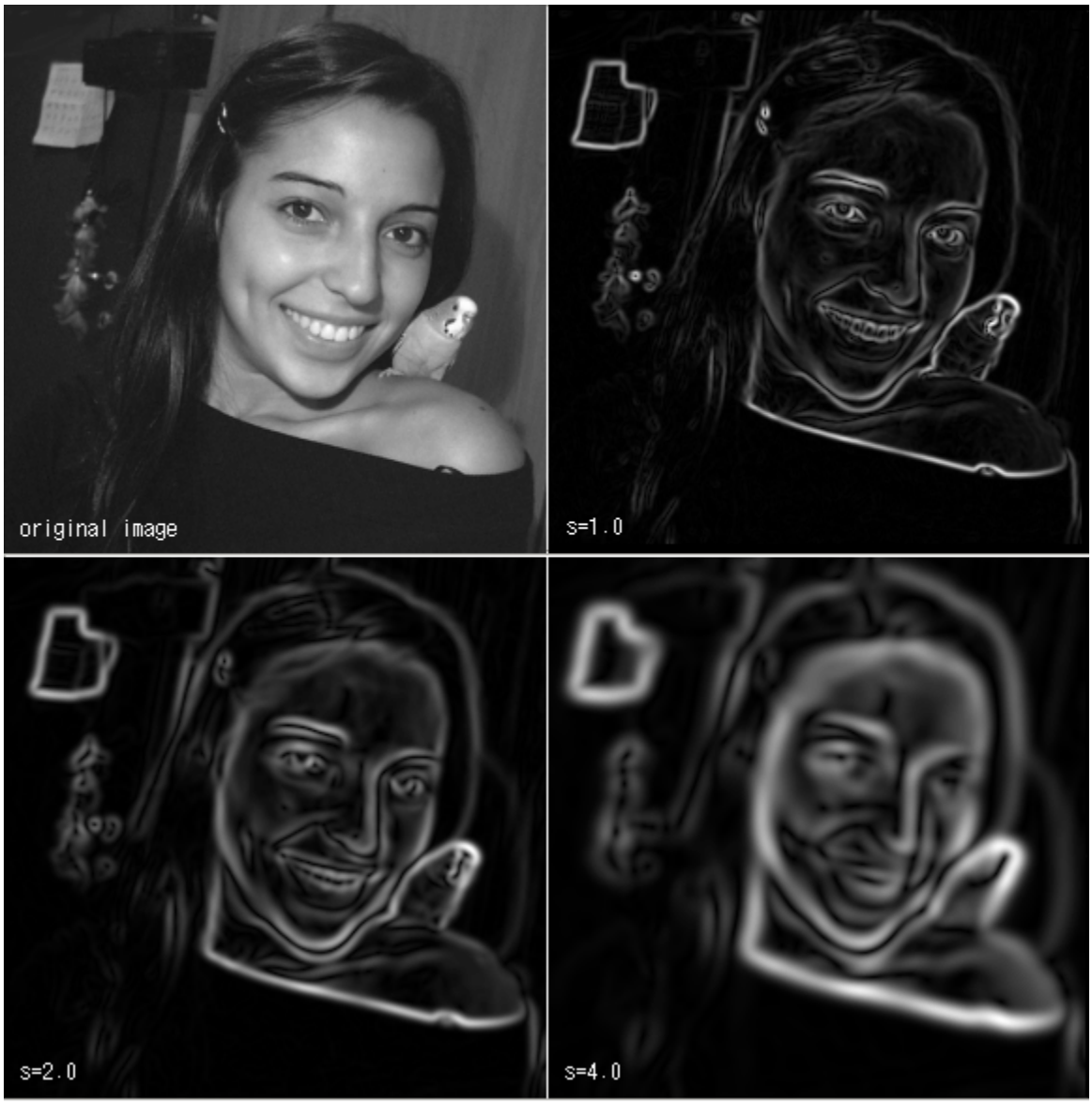


Fig. 49: Immagine di partenza e modulo del gradiente al variare del parametro di scala σ

2.3.1.2 Soppressione delle texture

I rilevatori discussi nelle sezioni precedenti non fanno alcuna differenza tra i vari tipi di bordi che possono comparire in un'immagine; in particolare, i bordi dovuti a texture non vengono in alcun modo distinti dai *contorni* degli oggetti e delle regioni dell'immagine (ad esempio la linea dell'orizzonte che separa la terra dal cielo). Per questo motivo, i rilevatori di questo tipo vengono

anche chiamati rilevatori generici o *non-contestuali*. Al contrario, operatori che agiscono in maniera selettiva sui bordi di interesse nel contesto di un task specifico, vengono chiamati rilevatori *contestuali*.

Negli ultimi anni sono stati proposti diversi metodi di estrazione dei contorni basati sullo studio del sistema visivo umano (Human Visual System, o HVS). L'HVS impiega speciali meccanismi per differenziare tra bordi isolati (come contorni di oggetti o confini tra regioni) e gruppi o cumuli di bordi (come quelli delle texture). Vari studi hanno mostrato che la percezione di uno stimolo orientato (ad esempio una linea) può essere influenzata dalla presenza di altri stimoli orientati nelle sue vicinanze; questa influenza può manifestarsi in diversi modi e riduce in generale la visibilità del contorno d'interesse (fig. 50). Altri studi psicofisici sul sistema visivo hanno mostrato che la percezione delle immagini può essere divisa in due fasi successive: la fase *pre-attentiva* e la fase *attentiva*. Nella prima fase, che dura per i primi 1-3 decimi di secondo dopo la proiezione dell'immagine sulla retina, vengono percepite le informazioni a bassa risoluzione, mentre nella seconda fase vengono identificati i dettagli. Alcuni esperimenti indicano che i dati visuali vengono elaborati separatamente nelle diverse bande di frequenza. Dunque si può assumere che l'immagine retinica venga decomposta tramite filtri passa-banda, costituendo così un modello multicanale. Questi studi suggeriscono la realizzazione di un sistema di rilevamento dei contorni basato su un framework multirisoluzione [1].

Fig. 50: Un contorno isolato (a) è più visibile dello stesso contorno sommerso da texture (b)

D'ora in poi verrà utilizzato il termine “contorno” per riferirsi a una linea che delimiti un oggetto (o parte di esso) all'interno di una scena. Con il termine “bordo” si intenderà invece la

generica variazione locale non trascurabile di luminosità.

Seguendo [1], ho realizzato (tramite un'implementazione C/C++) un meccanismo di inibizione che tiene conto, per ogni punto, dell'influenza contestuale delle texture che lo circondano. In particolare ho implementato due diverse soluzioni: la prima porta a una forma di auto-inibizione dei contorni effettivi degli oggetti presenti nell'immagine, ma gode di tempi di esecuzione piuttosto bassi. La seconda soluzione vince il problema dell'auto-inibizione al costo però di tempi di calcolo di molto superiori. Dipendentemente dal dominio applicativo è dunque preferibile adottare una soluzione rispetto all'altra. I due schemi di inibizione vengono mostrati nel seguito.

2.3.1.2.1 Schema base

L'idea di fondo dello schema di soppressione è quella di riconoscere la presenza di texture e di applicare una forma di inibizione delle stesse, in modo tale che vengano messi in risalto i soli contorni degli oggetti. Per fare questo è necessario, per ogni punto dell'immagine, tenere conto dell'influenza del contesto su di esso. Ciò può essere fatto, seguendo un approccio biologicamente motivato [1], calcolando un cosiddetto *termine di inibizione*, una quantità che si desidera grande in aree coperte da texture, e piccola in corrispondenza dei contorni delle entità sottoposte all'estrazione.

Sia $M_\sigma(x,y)$ l'ampiezza del gradiente, $I(x,y)$ l'immagine di partenza e g_σ una gaussiana come da sez. 2.3.1.1:

$$M_\sigma(x,y)=|\nabla_\sigma I(x,y)|=\sqrt{\left[\left\{I*\frac{\delta g_\sigma}{\delta x}\right\}(x,y)\right]^2+\left[\left\{I*\frac{\delta g_\sigma}{\delta y}\right\}(x,y)\right]^2}$$

Il termine di inibizione è una funzione $t_\sigma(x,y)$ definita come la media locale pesata di $M_\sigma(x,y)$ su una regione anulare intorno a ogni pixel, regione chiamata *suppression surround* [16]:

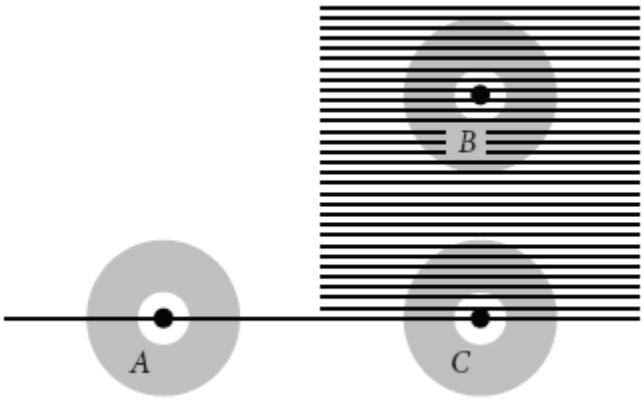


Fig. 51: A) Pixel circondato da una regione anulare su un contorno isolato B) Il pixel è in una regione ricoperta da texture C) Qui il pixel è posto sul bordo di una regione ricoperta da texture

Il calcolo del termine è effettuato tramite la convoluzione di $M_\sigma(x,y)$ con una funzione di pesatura $w_\sigma(x,y)$. La $w_\sigma(x,y)$ è definita come differenza di due Gaussiane concentriche, di cui viene dunque presa la rettificazione a mezza onda, e di cui infine si effettua una normalizzazione L^1.

$$DoG_\sigma(x,y)=|g_{k\sigma}(x,y)-g_\sigma(x,y)|^+$$

$$w_\sigma(x,y)=\frac{DoG_\sigma(x,y)}{\|DoG_\sigma(x,y)\|_1}=\frac{DoG_\sigma(x,y)}{\int\int_{R^2} DoG_\sigma(x,y)\,dx\,dy}$$

$$t_\sigma(x,y)=\{M_\sigma * w_\sigma\}(x,y)$$

Il parametro *k* nella definizione di $DoG_\sigma(x,y)$ misura il rapporto tra le deviazioni standard delle due Gaussiane. Variando questo rapporto, cambia l'estensione della regione anulare definita dal supporto della funzione di pesatura. In figura è possibile vedere alcuni esempi:

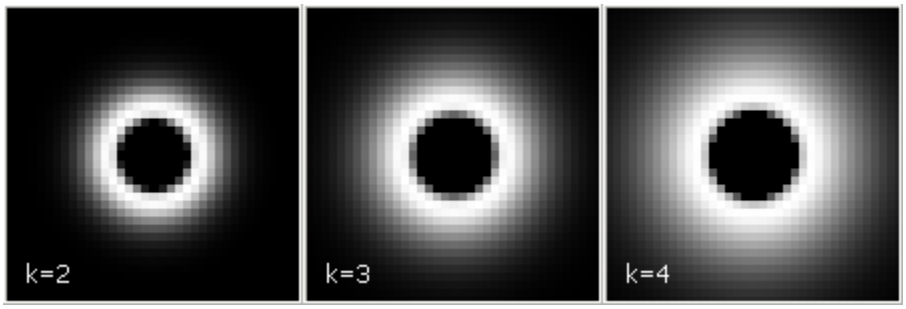


Fig. 52: Funzione di pesatura al variare del parametro k

La rettificazione a mezza onda è così definita:

$$|\psi|^{+} = \psi \quad se\, \psi \geq 0 \quad , \quad 0 \quad altrimenti$$

La regione centrale, esclusa dal calcolo del termine di inibizione, ha raggio ρ dato da:

$$\rho(k) = 2\sigma\sqrt{\frac{ln(k)}{1-1/k^2}}$$

che è una funzione che varia lentamente al variare di *k*. A distanze grandi dal centro dell'anello, il contributo dei punti può essere trascurato in quanto di intensità prossima a zero, e quindi è lecito definire il raggio dell'intorno anulare; questo risulta essere pari a circa *k* volte ρ [1]. Il parametro *k* dunque è da tenere in considerazione negli esperimenti, anche se è stato osservato non influire pesantemente sui risultati finali.

Calcolando in questo modo il termine di inibizione, si otterranno valori grandi per quei punti i cui dintorni presentano bordi multipli (fig. 51B), e sarà altresì piccolo per quei punti che giacciono su bordi isolati (fig. 51A). A questo punto per sopprimere le texture dall'immagine originale non resta che sottrarre il termine di inibizione dal modulo del gradiente; così facendo si lasciano relativamente inalterati i contorni isolati. Il termine di inibizione viene opportunamente pesato da un coefficiente di intensità di inibizione, che specifica per l'appunto la misura in cui l'inibizione apportata dal segnale t_σ grava sull'immagine. Il risultato è l'immagine $c_\sigma(x, y)$

$$c_\sigma(x, y) = |M_\sigma(x, y) - \alpha t_\sigma(x, y)|^{+}$$

Dipendentemente dal valore di α , il termine di inibizione può sopprimere le texture in parte o completamente. Ovviamente, per valori molti elevati di questo parametro, i contorni possono anche essere soggetti a inibizione. La scelta ideale di questo parametro è quella che massimizza la soppressione delle texture e minimizza l'inibizione parziale dei contorni veri. Si osservi che questa fase di rilevamento distingue il sistema presente dai tradizionali operatori non-contestuali: mentre questi agiscono in genere *migliorando* i contorni di interesse, la fase di soppressione contribuisce inibendo l'influenza dei bordi dovuti a texture.

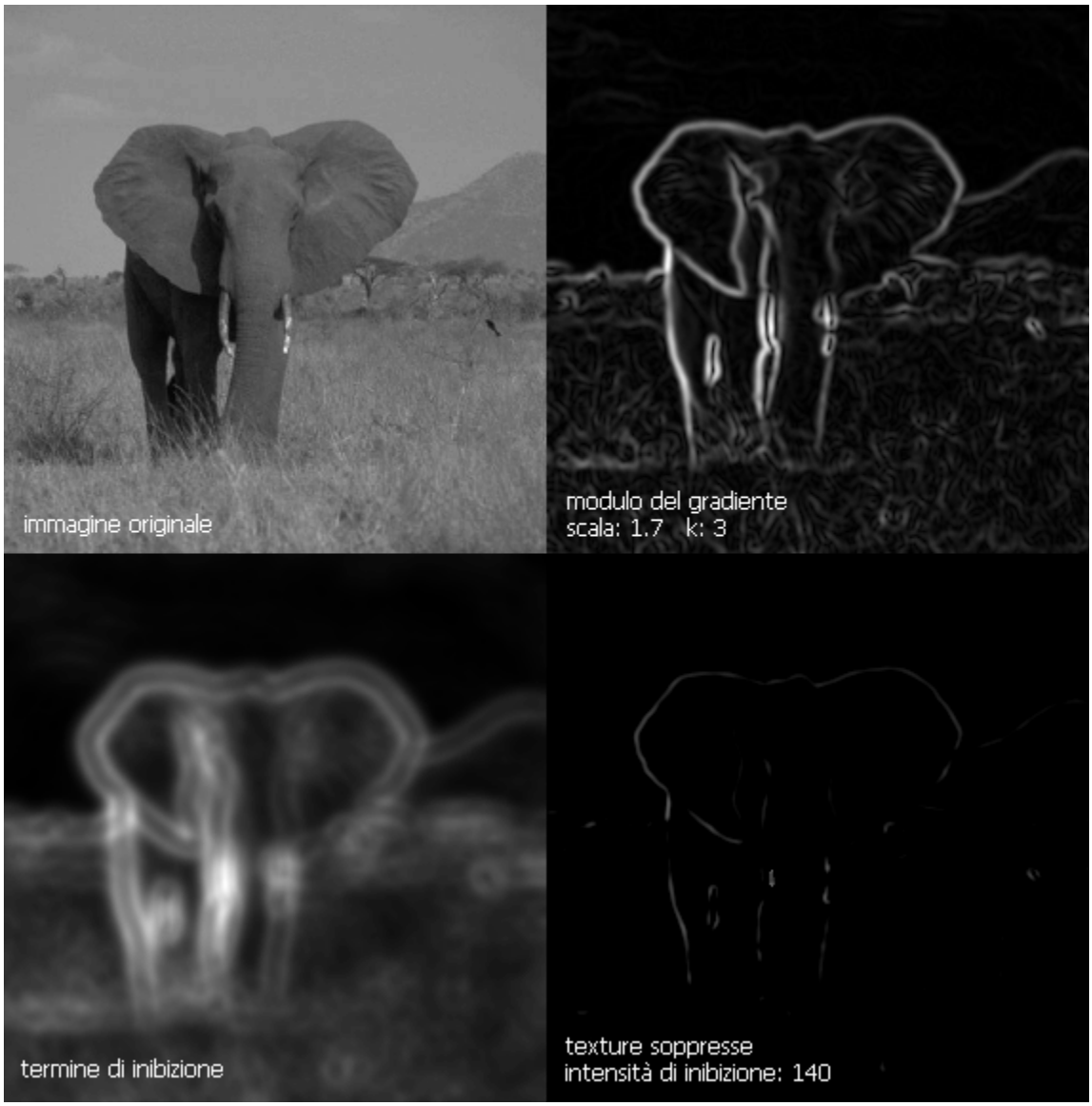


Fig. 53: Esempio di applicazione dello schema base di soppressione delle texture: il termine di inibizione viene applicato al modulo del gradiente per ottenere l'immagine in basso a destra

Purtroppo, questo schema di base soffre di uno svantaggio considerevole: anche se piccolo, il termine di inibizione non è mai nullo sui bordi isolati perché parti del bordo stesso vengono coperte dal supporto della funzione di pesatura. Possiamo chiamare questo fenomeno *auto-inibizione*. Inoltre, i bordi di oggetti ricoperti di texture, come in fig. 51C, vengono anch'essi inibiti in misura considerevole.

2.3.1.2.2 Schema migliorato

Questo secondo meccanismo di inibizione permette di vincere gli svantaggi presenti nel primo. L'intuizione che permette di fare ciò è semplice: evitando che la regione anulare copra lateralmente il bordo, si annulla l'auto-inibizione. Per fare questo basta escludere dalla funzione di pesatura una banda orientata lungo il bordo, come mostrato in figura 54.

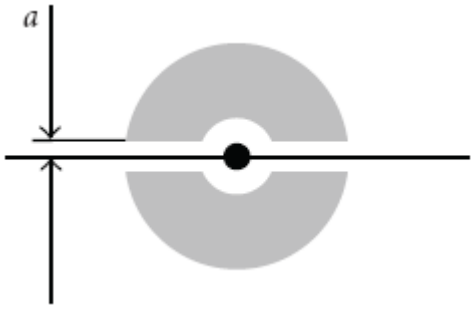


Fig. 54: Funzione di pesatura da cui viene esclusa una banda centrale larga 2a

Ora la funzione di pesatura consiste di due semi-anelli:

$$DoG^{+}_{\sigma,\phi}(x,y) = DoG_{\sigma}(x,y) \cdot U(x\cos\phi + y\sin\phi - a)$$

$$DoG^{-}_{\sigma,\phi}(x,y) = DoG_{\sigma}(x,y) \cdot U(a - x\cos\phi - y\sin\phi)$$

$$w^{\pm}_{\sigma,\phi}(x,y) = \frac{DoG^{\pm}_{\sigma,\phi}(x,y)}{\|DoG^{\pm}_{\sigma,\phi}(x,y)\|_1}$$

Nelle relazioni scritte sopra, *a* indica metà della larghezza della banda esclusa, $\phi \in [0,\pi)$ è un orientamento generico e *U* è la funzione gradino unitario, cioè vale 1 se l'argomento è maggiore o uguale a 0 e vale 0 altrimenti.

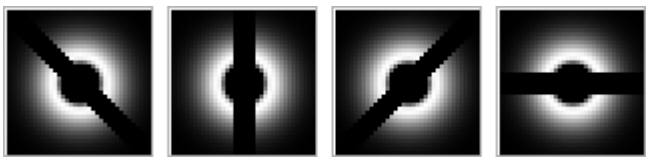


Fig. 55: Funzione di pesatura per lo schema migliorato, con 4 orientamenti della banda centrale

Il termine di inibizione viene a questo punto calcolato in questo modo:

1. Per ogni orientamento ϕ , viene effettuata la convoluzione tra il modulo del gradiente e i due semi-anelli:

   $$t_\sigma^{\pm}(x,y) = M_\sigma(x,y) * w_{\sigma,\phi}^{\pm}(x,y)$$

   Si forma dunque un'immagine prendendo per ogni pixel il minimo tra $t_\sigma^{+}(x,y)$ e $t_\sigma^{-}(x,y)$; questa scelta è motivata fisiologicamente e si rimanda a [1]. Si ottengono in questo modo tanti termini di inibizione (uno per orientamento) orientati in maniera differente.

2. I termini di inibizione vengono composti in questo modo: per ogni pixel, si prende il valore corrispondente dal termine di inibizione che è orientato approssimativamente come la fase del gradiente per quel pixel. Prima di costruire questa nuova immagine, però, la fase del gradiente viene approssimata con i 4 orientamenti principali (come in sez. 2.2.1)

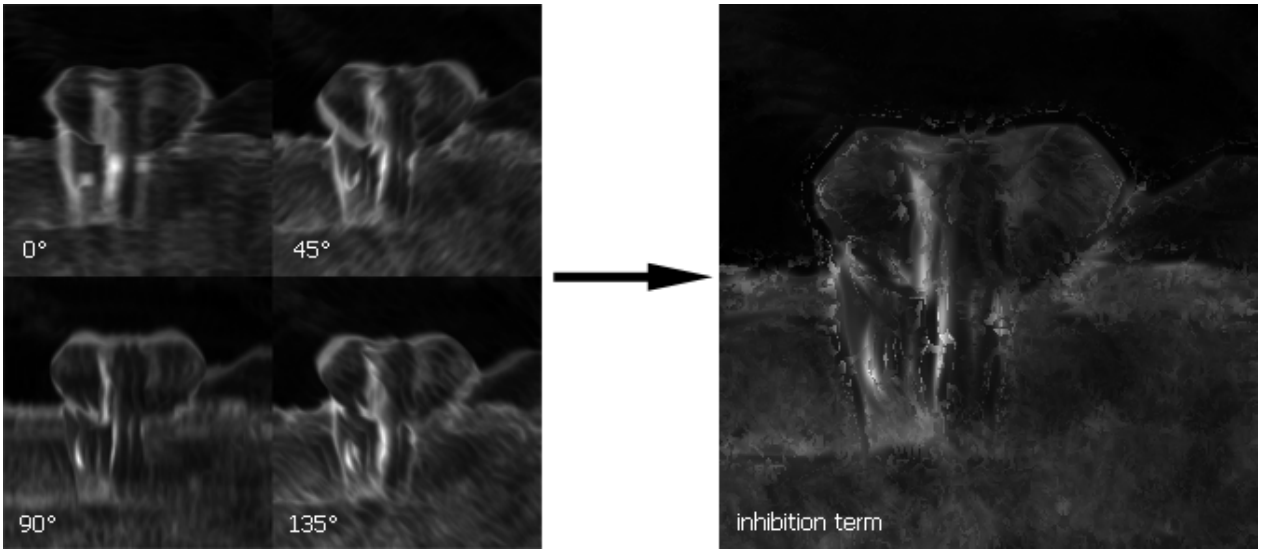


Fig. 56: Nello schema migliorato, il termine di inibizione viene composto a partire da più termini diversamente orientati

Il risultato di questi passi è un singolo termine di inibizione in cui viene tenuto conto, in ogni punto, dell'orientamento del bordo su cui il punto giace; ciò corrisponde al calcolo di una derivata direzionale dell'ampiezza del gradiente nella direzione del gradiente. Per motivi di costi computazionali, vengono presi in considerazione solo 4 orientamenti: 0 , $\pi/4$, $\pi/2$, $\pi 3/2$. Tuttavia, sperimentalmente si osserva che aumentando gli orientamenti di cui tenere

conto, le prestazioni non subiscono variazioni considerevoli.

Il parametro *a*, come detto, controlla l'ampiezza della banda esclusa e costituisce dunque un nuovo parametro del sistema.

L'intensità dei bordi può essere calcolata in maniera analoga a quanto fatto con il primo schema (per distinguere i termini di inibizione relativi ai due schemi, il secondo viene chiamato T_σ):

$$c_\sigma(x,y)=|M_\sigma(x,y)-\alpha T_\sigma(x,y)|^+$$

Dal momento che lo schema migliorato non soffre del problema dell'auto-inibizione, il parametro α (che misura l'intensità dell'inibizione) può essere impostato a valori più alti, portando in questo modo a una soppressione migliore delle texture mantenendo allo stesso tempo integri i contorni, anche quelli più deboli.

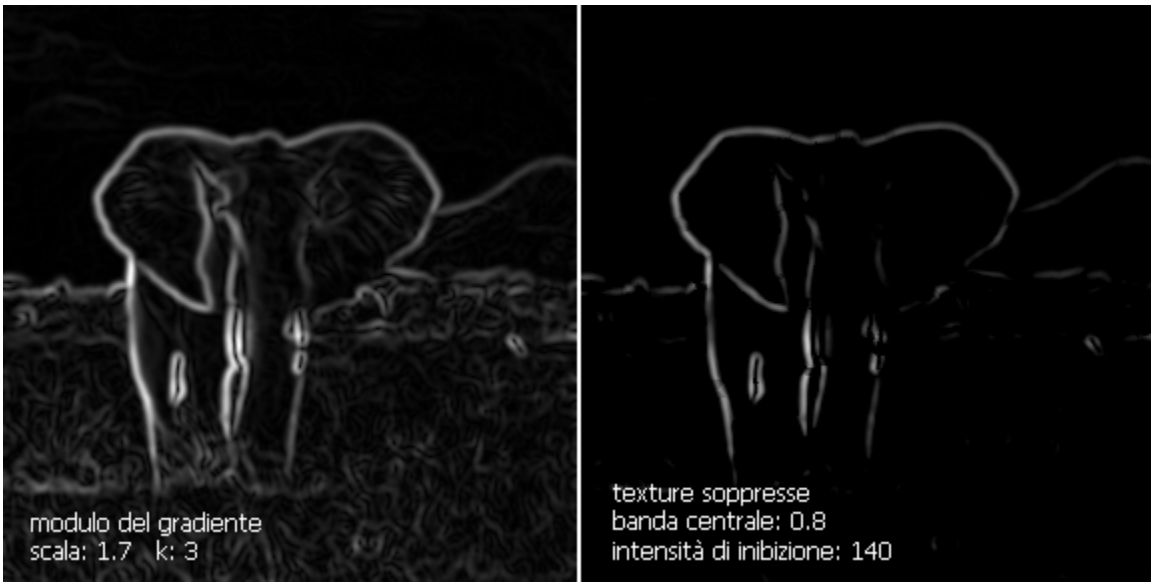


Fig. 57: Esempio di applicazione del secondo schema di soppressione delle texture. I parametri utilizzati sono gli stessi di fig. 53, ma si ottengono contorni migliori.

2.3.1.3 Binarizzazione

L'ultima fase del processo di estrazione dei contorni è la binarizzazione. Tipicamente in un task di questo tipo i passi necessari per ottenere la mappa finale sono la ritenzione dei massimi

locali e la sogliatura (sez. 2.1).

Data la fase del gradiente e l'immagine di intensità dei bordi risultante dalle fasi precedenti (segnale $c_\sigma(x, y)$), la procedura di soppressione dei non-massimi consiste nella ricerca dei massimi locali nella direzione del gradiente. Prima viene inizializzata una nuova immagine, i cui pixel sono impostati al valore di sfondo (zero). Dopodiché, per ogni punto in $c_\sigma(x, y)$, si considerano i due punti vicini lungo la linea del gradiente (si ricordi che la fase del gradiente è stata preventivamente approssimata lungo i quattro orientamenti 0, 45°, 90°, 135°): se il pixel centrale ha un'intensità maggiore di quella di entrambi i vicini, alle stesse coordinate della nuova immagine viene disegnato un pixel della stessa intensità (e *non* un pixel di primo piano, dal momento che l'immagine in formazione non è ancora la mappa binaria). Questo processo assottiglia i bordi dell'immagine $c_\sigma(x, y)$ in contorni candidati spessi al più un pixel, e lo fa conservando i massimi locali lungo i bordi.

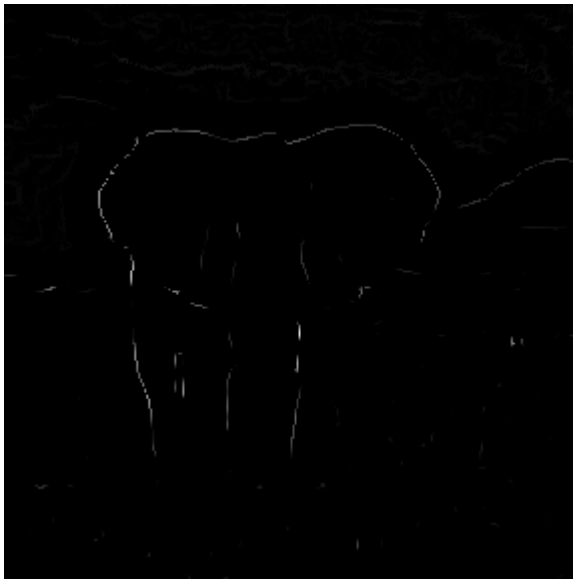


Fig. 58: Immagine dei massimi locali; il contrasto è stato migliorato per una migliore visibilità

A questo punto si può applicare una procedura per la costruzione della mappa binaria finale, che sarà il risultato ultimo prodotto dal rilevatore. Questo secondo passaggio viene effettuato tipicamente scegliendo (anche in maniera automatica) una soglia sui valori di intensità al di sotto della quale scartare pixel -che dunque *non* vengono marcati come bordi- e al di sopra della quale marcarli come tali. Applicate queste marcature si costruisce un'immagine binaria in cui sono distinguibili i contorni finali dell'immagine. Questa tecnica prende anche il nome di *hard thresholding* (sez. 1.5.1), per via della scelta della funzione di thresholding che discrimina in

maniera troppo marcata due pixel vicini in egual misura al valore di soglia, con uno che vi tende dal basso e l'altro dall'alto.

In immagini che contengono texture, questo metodo tradizionale non dà però buoni risultati: è più che probabile che i bordi dovuti a texture siano più intensi di "pezzi" di veri contorni, anche in seguito a inibizione (che è un processo di attenuazione e *non* di risalto). Una forma di isteresi è anche stata applicata in [16] a seguito del processo di inibizione, ma ancora con risultati poco soddisfacenti. Per questo motivo, il software realizzato in questa sede adotta un algoritmo di thresholding ad-hoc che tiene conto dell'azione svolta dal sistema di inibizione delle texture mostrato poc'anzi.

Questo nuovo algoritmo di sogliatura è basato sull'osservazione che i contorni degli oggetti danno vita, in genere, a contorni *lunghi e contigui* di pixel non nulli, mentre i bordi dovuti a texture -soprattutto dopo l'inibizione- consistono di componenti relativamente piccoli e discontinui.

L'algoritmo prevede in primo luogo la ricerca dei cosiddetti *componenti connessi*, a partire dall'immagine dei massimi locali. Per comprendere la nozione di componente connesso è necessario anzitutto definire in maniera precisa quando un pixel può considerarsi un *vicino* di un altro pixel. Questa definizione dipende in realtà dall'applicazione e dalle scelte progettuali; non esiste una scelta che funziona al meglio in tutti i casi. Una definizione prevede che i vicini di un pixel *p* siano quei pixel che lo affiancano lungo le direzioni orizzontale e verticale. Ce ne sono cioè quattro:

	1	
4	**p**	2
	3	

Uno schema di questo tipo prende il nome di *four-connected neighborhood*. Un'altra definizione comune, che è quella peraltro utilizzata nel presente lavoro, racchiude l'insieme di pixel che condividono un bordo o un angolo con *p*; ce ne sono otto (di seguito viene riportata la numerazione classica):

8	1	2
7	**p**	3
6	5	4

Questo schema è chiamato *eight-connected neighborhood*. Un *componente connesso* è un

insieme di pixel vicini tra loro secondo la definizione adottata.

Per la ricerca dei componenti connessi ho adottato la tecnica della *union-find* [5], opportunamente modificata. Questo algoritmo è progettato per poter formare in maniera rapida l'unione di due insiemi, e trovare velocemente l'insieme che contiene un dato elemento. Il primo passo consiste nel creare un nodo per ogni pixel non-nullo dell'immagine di partenza; l'algoritmo consiste essenzialmente nella costruzione di alberi, quindi è prevista una funzione che, dato un nodo, dica quale sia la sua root. Chiamiamo questa funzione $nodes(k)$ (seguendo la notazione di [5]), che dato un nodo *k* restituisce un puntatore al nodo di root. Un nodo che è root di se stesso ha ovviamente $nodes(k)=k$. Ora l'immagine viene scandita per righe, e per ogni pixel vengono considerati i suoi otto vicini (nell'ordine mostrato in figura) per verificare che possano essere connessi al pixel corrente (indicato con **p**).

1	2	3
8	**p**	4
7	6	5

La verifica e la costruzione degli alberi viene fatta in questo modo (pseudo-codice):

```
cur_node:     il pixel corrente
other_node:   il vicino considerato
outlier:      un nodo non connesso ad altri
other_tree:   l'albero che contiene other_node

if other_node is outlier then
        if !connected(cur_node,other_node) then
                other_node.root → cur_node.root
        endif
else
        if !connected(cur_node,other_node) then
                if cur_node is outlier then
```

```
                cur_node.root → other_node.root
            else
                other_tree.root → cur_node.root
            endif
        endif
endif
```

Si noti che l'effettiva costruzione e unione degli alberi consiste in semplici assegnamenti di puntatori in stile C (simbolo →), quindi l'algoritmo è molto veloce dal momento che non vengono effettivamente manipolati o spostati dati. Al termine dell'algoritmo, si sono ottenuti tanti alberi quanti sono i componenti eight-connected nell'immagine.

Passiamo ora alla fase di thresholding. L'intuizione che sta alla base di questo nuovo algoritmo è la seguente: occorre tenere conto della *lunghezza* dei componenti connessi, insieme all'intensità dei pixel che li costituiscono, e in base a queste misurazioni scegliere quali componenti tenere e quali no. Dal momento che sperabilmente (con la giusta scelta dei parametri) la fase di soppressione delle texture ha inibito i bordi non desiderati, possiamo aspettarci che le porzioni di texture sfuggiti alla fase di inibizione appaiano sotto forma di piccoli componenti sconnessi, mentre i contorni effettivi degli oggetti siano componenti connessi contigui e abbastanza lunghi. In base a ciò, è possibile attuare una forma di sogliatura che ne tenga conto.

Si osservi che un algoritmo siffatto darebbe pessimi risultati senza l'applicazione preventiva della soppressione delle texture, fase che risulta quindi essenziale. Infatti in assenza di essa l'immagine risultante includerebbe molte componenti intricate dovute alle texture quanto alle regioni rumorose dell'immagine di partenza (fig. 59). Tali componenti, sebbene possano avere valori bassi di intensità (risultando, nell'immagine di partenza, difficilmente percepibili a occhio nudo), formano bordi contigui molto lunghi e non possono dunque essere eliminate con facilità dall'algoritmo di sogliatura. Al contrario, applicando l'inibizione si spezzano queste componenti in piccoli cumuli, annullandone alcuni, e dunque quelli rimanenti possono essere rimossi con successo tramite la sogliatura.

Fig. 59: L'immagine di sinistra risulta dall'applicazione dell'algoritmo di sogliatura senza previa soppressione delle texture. Si possono notare molte componenti intricate dovute al rumore. L'immagine di destra risulta invece dall'applicazione dello schema di soppressione delle texture, seguito da sogliatura.

Per proseguire occorre introdurre uno strumento importante nell'ambito dell'elaborazione morfologica delle immagini. In questo campo si studiano tipi di elaborazione che prevedono la trasformazione della struttura spaziale delle immagini per il tramite di particolari operatori. Questi operatori vengono chiamati *elementi strutturanti* e sono generalmente rappresentati con delle piccole maschere che vengono fatte scorrere sull'immagine binaria da modificare. Se il pattern (a valori binari) della maschera combacia (*hit*) con lo stato del pixel coperto dalla maschera, un pixel di output in corrispondenza spaziale con il pixel centrale della maschera viene impostato al valore binario desiderato. Altrimenti, in caso di *miss*, il pixel di output viene impostato all'opposto valore binario. L'operazione di *dilatazione* in particolare, assieme all'erosione, è una operazione di base nell'area della morfologia matematica. Si supponga che l'origine di un elemento strutturante 3x3 sia il suo centro; per calcolare la dilatazione morfologica di un'immagine binaria data in input, viene fatto scorrere il centro del quadrato su ogni pixel e vengono considerati i pixel coperti dall'elemento strutturante. Se almeno un pixel del quadrato coincide con un pixel posto a 1 nella regione coperta dell'immagine di input, allora il pixel corrente viene impostato anch'esso a 1. In caso contrario, il pixel mantiene il suo valore. L'effetto di questa operazione è che i contorni si dilatano mentre i buchi si restringono.

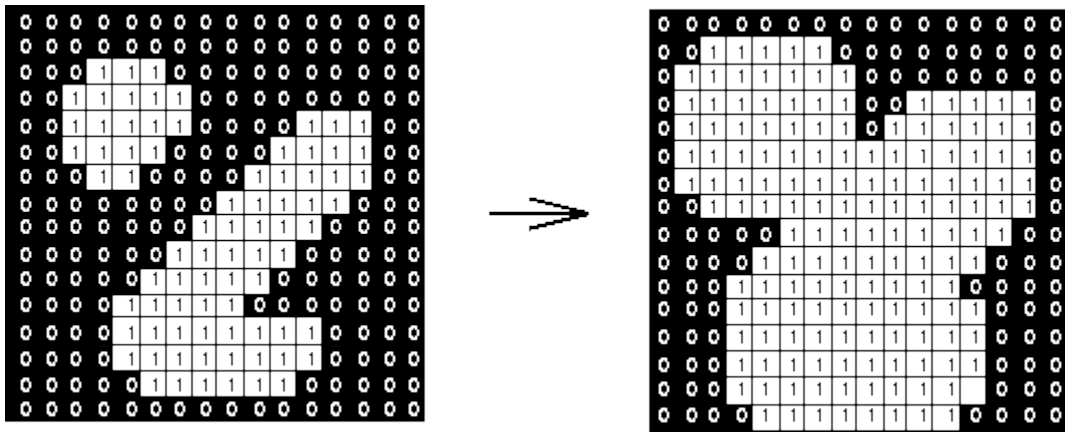


Fig. 60: Dilatazione morfologica

Per applicare questa sogliatura su misura possiamo ora definire le quantità su cui agire. Avendo ottenuti i componenti connessi, viene loro applicata una dilatazione morfologica con un quadrato q_3 di dimensioni 3x3 come elemento strutturante, e vengono così ottenuti i componenti dilatati D_k .

$$D_k = C_k \oplus q_3$$

Per ogni componente C_k viene introdotta una quantità G_k , chiamata *peso globale del contorno*, definita come la somma dei valori di $c_\sigma(x,y)$ sui componenti dilatati D_k :

$$G_k = \sum_{(x,y)\in D_k} c_\sigma(x,y)$$

Questo valore è tanto più grande quanto più lunghi sono i componenti connessi e quanto più intensi sono i bordi che li compongono. Chiaramente alcuni termini della sommatoria potranno essere nulli dal momento che non è detto che in corrispondenza di tutti i pixel dei D_k si abbia $c_\sigma(x,y) \neq 0$. Ora possiamo applicare una soglia ai pesi G_k : la mappa binaria $b_\sigma(x,y)$ è costruita impostando in primo piano il valore dei pixel dei vari C_k i cui pesi G_k superano un certo G_{min} .

$b_\sigma(x,y) = 1$ se $(x,y) \in \bigcup_{G_k > G_{min}} C_k$

$b_\sigma(x,y) = 0$ altrimenti

In questo modo è possibile estrarre correttamente anche contorni formati da segmenti deboli, e risanare le giunzioni tra contorni soppresse dal meccanismo di inibizione. In fig. 61 si vede il risultato dell'applicazione di questo tipo di thresholding confrontato con la tecnica tradizionale. Il parametro G_{min} può essere regolato in base al tipo di oggetto che si sta analizzando e dovrebbe essere in qualche misura proporzionale alla dimensione lineare dell'immagine. Nella pratica il valore viene determinato empiricamente; una scelta automatica del parametro costituirebbe un utile miglioramento dell'algoritmo.

Fig. 61: Sulla sinistra, la mappa binaria ottenuta con la tecnica di thresholding tradizionale. Sulla destra, il risultato dell'applicazione del metodo di sogliatura proposto.

2.3.2 Rilevamento in multirisoluzione

In sez. 1.3 si è parlato di analisi multirisoluzione e si è visto che le scale più grossolane contengono solo la morfologia più generale di un'immagine, mentre i dettagli in alta frequenza (come le texture) scompaiono. Si osservi la fig. 62. Ad alta risoluzione, i contorni vengono rilevati nelle posizioni esatte e le giunzioni tra bordi sono preservate, ma è anche presente una certa quantità di texture. A una scala più grossolana, e cioè al diminuire della risoluzione, le texture scompaiono ma i contorni traslano dalla loro posizione corretta (in particolar modo in prossimità di curve), e alcune giunzioni vengono distrutte.

Fig. 62: L'immagine sulla sinistra è ottenuta a partire dalla risoluzione più fine: i contorni si trovano nelle posizioni esatte, ma sono presenti anche molti dettagli. L'immagine sulla destra è stata invece ottenuta a partire da una risoluzione più bassa ($\sigma = 5$): i contorni sono contigui e i dettagli scompaiono, ma la posizione dei bordi è inesatta.

Un risultato ideale dovrebbe prendere i vantaggi dalle due situazioni: è desiderabile cioè mantenere i contorni degli oggetti nelle loro posizioni esatte, mantenere le giunzioni e allo stesso tempo sopprimere più texture possibili. Per fare questo è pensabile sovrapporre le due immagini binarie, e selezionare dalla mappa ad alta risoluzione solo quei pixel di primo piano che siano "abbastanza vicini" a quelli di risoluzione più bassa.

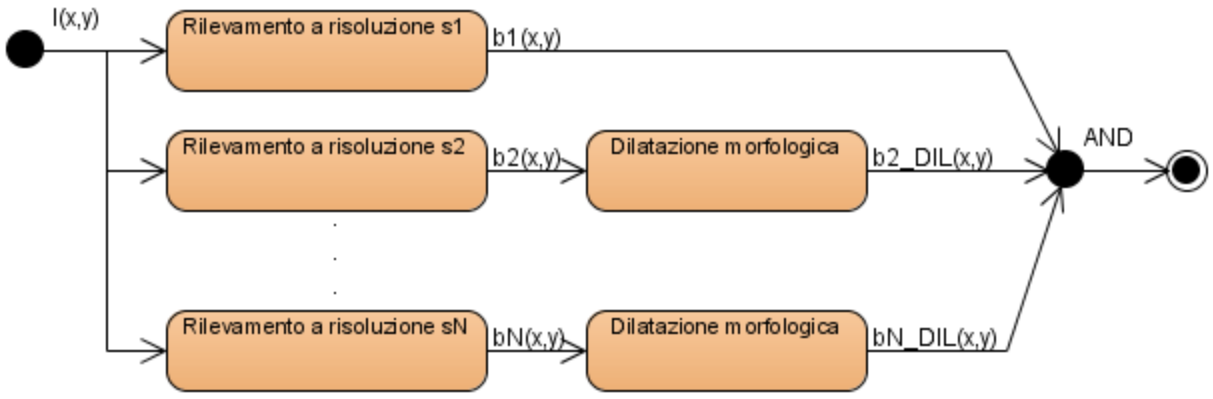


Fig. 63: Diagramma del metodo di rilevamento in multirisoluzione

In pratica, prima di sovrapporre le due immagini viene applicata la dilatazione morfologica alla mappa a bassa risoluzione (fig. 64). I contorni degli oggetti, che sono ben localizzati e dettagliati nella prima mappa, sono contenuti nella dilatazione; al contrario, gran parte delle texture non sono presenti a risoluzione più bassa e quindi un'operazione di AND logico tra le due mappe dà i risultati sperati.

Fig. 64: Sovrapposizione tra una mappa binaria e un'altra a risoluzione più bassa ma opportunamente dilatata

L'elemento strutturante adoperato per questa operazione è un disco di raggio 3σ ; è stato scelto un disco invece di un quadrato per formare contorni più morbidi a seguito della dilatazione. La dilatazione morfologica permette di ricostruire le giunzioni perse all'aumentare della scala, e compensa le traslazioni dei contorni. Naturalmente, un procedimento di questo tipo può essere ripetuto il numero di volte desiderato. Il rilevatore multiscala funziona in questo modo:

1. Per ogni scala ($k=1,\dots,N$), vengono calcolate le mappe binarie dei contorni $b_k(x,y)$ (seguendo il procedimento a risoluzione fissa visto in sez. 2.3.1), ottenendo così quella che viene chiamata rappresentazione dei bordi in multiscala, o *MER* [12]
2. Viene applicata la dilatazione morfologica a tutte le mappe, tranne che a quella ad alta risoluzione: $b_{k,DIL}=b_k \oplus D_{3\sigma}$ per $k=2,\dots,N$
3. Le mappe vengono messe in AND logico per ottenere l'immagine finale: $b_{out}=b_1(x,y) \cdot \prod_{k=2}^{N} b_{k,DIL}(x,y)$

Si noti che al passo 3 si è effettuata la moltiplicazione tra scale *dopo* aver ottenuto le mappe binarie (come avviene tipicamente coi rilevatori che agiscono in multiscala), e non *prima* come invece avviene ad esempio nel rilevatore di Zhang e Bao (sez. 2.2.3).

Si osservi inoltre che questo procedimento in multiscala deriva direttamente da una caratteristica del sistema visivo umano, secondo la quale l'informazione visiva viene elaborata separatamente in bande di frequenza distinte [1].

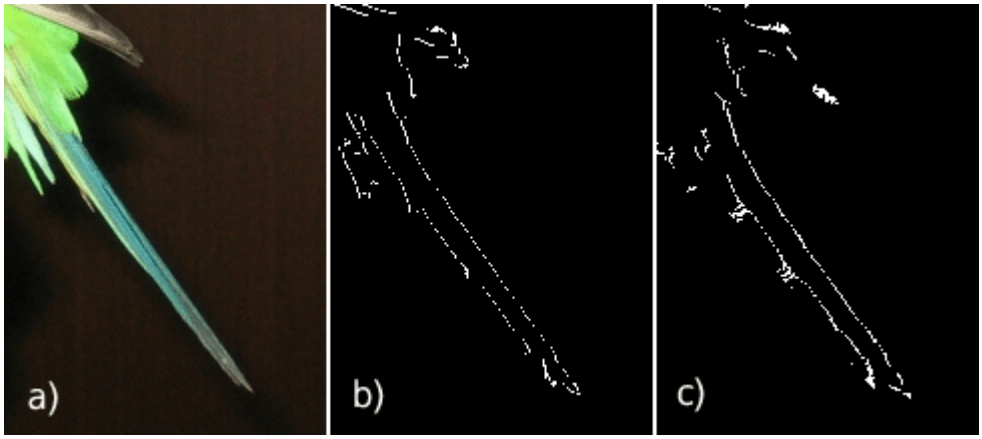


Fig. 65: L'immagine originale (a) viene analizzata con lo schema migliorato a risoluzione fissa (b) e con lo schema in multirisoluzione (c). In c) è possibile osservare bordi più solidi.

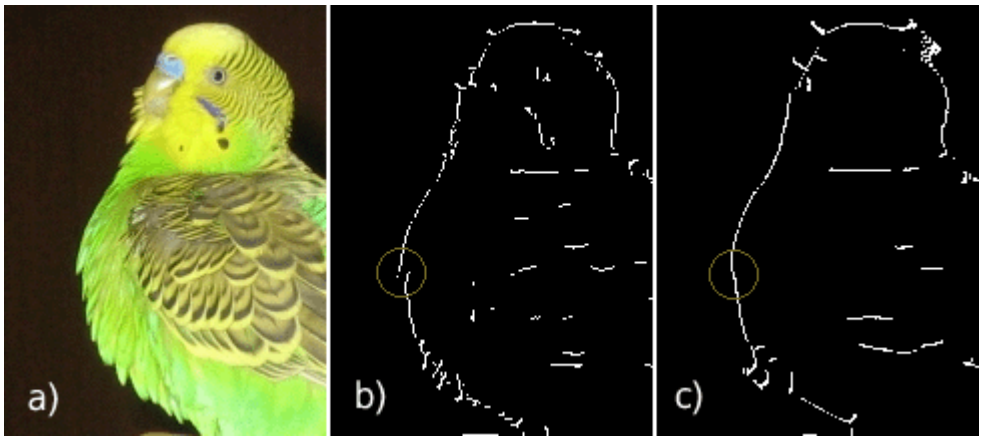


Fig. 66: La mappa risultante dal rilevamento a più scale (c) presenta meno dettagli, contorni più solidi e giunzioni ripristinate (cerchiate nell'immagine).

Capitolo 3: Rilevamento e tracking di corpi in sequenze video

Il tracciamento o inseguimento nel tempo (o, più comunemente, *tracking*) di corpi (o entità, nell'accezione più generale del termine) consiste nel trovare corrispondenze tra oggetti in frame consecutivi; le difficoltà di questo task risiedono tipicamente nella complessità della scena e degli oggetti tracciati. L'analisi della corrispondenza è spesso supportata da forme di predizione: basandosi su oggetti rilevati in precedenza, è possibile predire con un certo grado di affidabilità l'aspetto e la posizione degli oggetti nei frame successivi, prima ancora di rilevarli; una volta rilevati, vengono dunque confrontati (con una qualche metrica) con le loro versioni "predette". Questa operazione introduce una regione di interesse sia nello spazio dell'immagine che nello spazio degli stati, e dunque riduce il fabbisogno generale di calcoli. La predizione dei vari parametri sullo stato è basata su un modello di come questi parametri evolvono nel tempo; un modello tipico prende in considerazione velocità e accelerazione degli oggetti. Più avanti verrà spiegato come il software costruito in questa sede affronti il problema del tracking facendo uso di un modello di questo tipo.

In generale, una qualsiasi funzione di tre variabili $I(x, y, t)$, dove le variabili di spazio x e y così come la variabile tempo t, sono discrete e limitate, può rappresentare una sequenza di immagini. Tipicamente però, immagini "catturate" a istanti di tempo vicini sono fortemente correlate tra di loro, poiché si riferiscono alla stessa scena ripresa da punti di vista che differiscono solo leggermente. Di solito questa correlazione si esprime in altri termini, dicendo che vi sono gruppi di punti che si spostano in un flusso numerabile di immagini. Formalmente, questo significa che la funzione $I(x, y, t)$ non è arbitraria, bensì soddisfa la seguente proprietà:

$$I(x, y, t+\tau) = I(x-\psi, y-\nu, t)$$

Cioè, un'immagine presa all'istante $t+\tau$ può essere ottenuta spostando ogni punto dell'immagine corrente (riferita al tempo t) di una certa quantità. La misura di questo spostamento $d = (\psi, \nu)$ viene chiamata *dislocazione* del punto (x, y) tra gli istanti temporali t e $t+\tau$, ed è in generale una funzione di x, y, t e τ.

Tuttavia, questa proprietà viene violata in molte situazioni. Ad esempio, nelle aree dell'immagine in cui sono presenti corpi occludenti, i punti che vi si avvicinano cominciano a scomparire e ricomparire al progredire del tempo. Al di là di questo, anche senza l'assunzione di

illuminazione costante, nelle posizioni in cui l'intensità dell'immagine cambia repentinamente al variare di x e di y, il punto della variazione rimane ben definito a dispetto di variazioni piccole della luminosità nel suo intorno.

Nel corso degli anni sono state proposte diverse tecniche e sono stati realizzati diversi sistemi con lo scopo di tracciare il movimento di entità di vario tipo (ad esempio, persone) all'interno di scene video in condizioni non ottimali. In questo ambito vengono fatte varie assunzioni associate alla cattura del movimento (ad esempio, poco fa è stato fatto cenno all'assunzione di illuminazione costante); le assunzioni tipiche si possono raggruppare in due classi: *assunzioni sul movimento* e *assunzioni sull'aspetto*. La prima classe riguarda vincoli e restrizioni sui movimenti dell'oggetto e/o della telecamera; la seconda classe riguarda invece l'aspetto dell'oggetto e dell'ambiente in cui si muove. In tabella vengono riportate le assunzioni più tipiche.

Assunzioni legate al movimento	***Assunzioni legate all'aspetto***
1. L'oggetto rimane all'interno dell'area di interesse	**Ambiente**
2. La telecamera sta ferma o segue un movimento costante	1. Illuminazione costante
3. C'è una sola entità per volta nell'area di interesse	2. Sfondo statico
4. Il movimento è parallelo al piano della telecamera	3. Sfondo uniforme
5. Assenza di occlusioni	4. I parametri della telecamera sono noti
6. I movimenti sono lenti e continui	5. L'hardware è dedicato
7. Il tipo di movimento dell'oggetto è noto e ben caratterizzabile	**Oggetto**
	1. La posizione iniziale è nota
	2. L'oggetto è noto
	3. L'oggetto presenta caratteristiche tracciabili
	4. L'oggetto ha dimensioni prevedibili

3.1 Il problema della registrazione

Il problema della *registrazione traslazionale di immagini* può essere caratterizzato come segue: date due funzioni $F(x)$ e $G(x)$, che danno i valori rispettivi dei pixel a ogni locazione x (un vettore) in due immagini, vogliamo trovare il vettore h di disparità che minimizzi una qualche misura della differenza tra $F(x+h)$ e $G(x)$, con x compreso in una regione di interesse R.

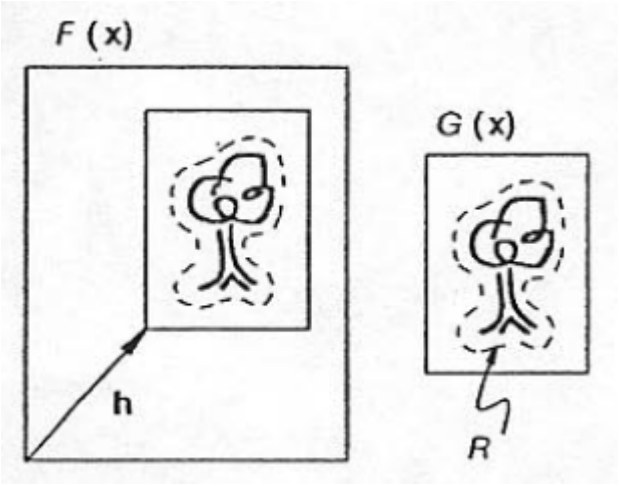


Fig. 67: Il problema della registrazione

Misure tipiche di differenza sono la norma L^1 e la norma L^2:

$$Norma\ \ L^1 = \sum_{x \in R} |F(x+h) - G(x)|$$

$$Norma\ \ L^2 = \left(\sum_{x \in R} [F(x+h) - G(x)]^2\right)^{1/2}$$

Una tecnica ovvia per la registrazione di due immagini consiste nel calcolare la differenza tra di esse per ogni possibile valore del vettore h, che coincide con una ricerca esaustiva. Ovviamente questa tecnica richiede molto tempo; Lucas e Kanade hanno presentato un algoritmo

efficiente che, a partire da una stima iniziale di h, usa l'intensità del gradiente spaziale in ogni punto per modificare il valore di h e portare a un match migliore [18]. Se l'iterazione converge, lo fa in un tempo logaritmico (contro la complessità quadratica della ricerca esaustiva). Questa tecnica trae vantaggio dal fatto che in molte applicazioni, come detto, le due immagini si trovano già in uno stato di registrazione approssimativa (ad esempio due frame successivi in una sequenza video).

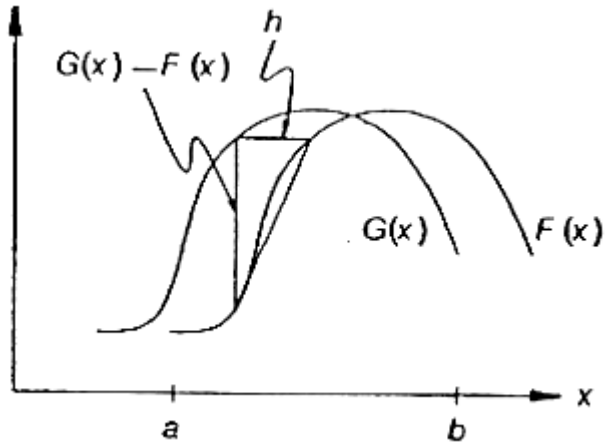


Fig. 68: Due curve da registrare

3.2 Selezione e rilevamento delle feature

Pur avendo un metodo molto efficiente (Lucas e Kanade) per risolvere il problema della registrazione, non siamo ancora in grado di dare una risposta a queste domande: come selezionare le feature, e come tracciarle appropriatamente di frame in frame anche in condizioni non ottimali (presenza di rumore, variazioni di illuminazione, corpi occludenti, ecc.). Per prima cosa, bisogna tenere conto che grazie allo spostamento minimo tra frame, l'illuminazione delle due immagini coinvolte differisce a meno di un termine residuo che dipende quasi linearmente dal vettore di traslazione ottenuto col metodo di Lucas-Kanade [24]. Inoltre, non esiste chiaramente un solo metodo per estrarre delle feature da tracciare all'interno di una sequenza di frame. In letteratura sono state proposte molte definizioni di cosa è una "buona feature", definizioni basate sulla nozione aprioristica di caratteristica "significativa" o "d'interesse". In contrasto con ciò, Tomasi e Kanade hanno sviluppato un criterio di selezione ottimale che tenga conto dell'algoritmo stesso

adottato per la registrazione: in altre parole, una feature è una buona feature se può essere ben inseguita.

Purtroppo però, un pixel *isolato* non può essere tracciato in una sequenza video, a meno che non abbia un valore di luminosità *molto* diverso da quello dei suoi vicini. Di fatto, il valore del pixel può variare diversamente a causa del rumore, e può essere altresì confuso con i pixel adiacenti. Come conseguenza, è spesso difficile o persino impossibile determinare la nuova posizione del pixel al susseguirsi dei frame. Per questo motivo, quello che si fa tipicamente è inseguire *finestre* di pixel, e nel fare questo si cercano finestre che contengano un certo quantitativo di texture. Sfortunatamente, anche in questo caso i pixel contenuti in una finestra possono comportarsi in maniera differente, e si può comunque incorrere in zone che presentano occlusioni.

Queste ed altre argomentazioni dovrebbero dare un'idea dei problemi che devono essere affrontati nell'approcciare un task di *body tracking* in sequenze video. Chiaramente esistono molte tecniche e metodi di varia efficacia che rispondono in misura diversa alle domande che sono state poste. Nell'ambito di questa tesi è stato sviluppato, basandosi su quanto realizzato in [29] e modificandolo pesantemente (dandone inoltre un'implementazione C/C++), un sistema di tracking di entità pensato per agire offline (cioè *non* in tempo reale) e fondato sul concetto di *traiettoria* e *spostamento coerente* di punti. In sez. 3.3 sono presenti tutti i dettagli.

3.2.1 Machine Learning per il rilevamento di angoli

Nell'ambito di questa tesi, al fine di favorire un'eventuale estensione futura delle applicazioni affinché possano migrare verso una realizzazione real-time (o quasi, vedere capitoli 5 e 6), è stato realizzato uno dei rilevatori più veloci attualmente noti, sacrificando in parte l'efficacia ottenibile ad esempio con il metodo di Tomasi-Kanade (cui si è accennato nella sezione precedente, per dettagli vedere [24]). Questo algoritmo si chiama *FAST* e trae vantaggio da una fase di apprendimento (*machine learning*) per ottenere un grosso aumento di velocità.

Gran parte degli algoritmi di rilevamento di feature funziona calcolando una funzione di risposta C agli *angoli* lungo l'immagine. I pixel che, oltre a essere massimi locali, superano una certa soglia di intensità, vengono dunque mantenuti. Moravec utilizza una funzione di smoothing per ridurre l'influenza del rumore sui singoli frame, e dunque calcola la somma delle differenze quadratiche (*SSD*) tra una finestra centrata in un angolo candidato e diverse finestre traslate di poco dalla prima, lungo più direzioni. C è dunque presa come la SSD più piccola così ottenuta, il che assicura che gli angoli estratti siano quelle locazioni che cambiano massimamente con le traslazioni. Altri rilevatori estendono il sistema di Moravec prendendo

versioni modificate della SSD e calcolando in maniera diversa la risposta C. Un'altra classe di rilevatori esamina l'immagine in piccoli blocchi per verificare che questi "somiglino" ad angoli. I rilevatori di questo tipo non richiedono alcuna forma di smoothing per ridurre il rumore e conseguentemente sono piuttosto efficienti; purtroppo tendono però a dare cattive prestazioni con immagini che esibiscono pochi contenuti in alta frequenza, come le immagini sfocate. Il rilevatore FAST appartiene a questa categoria.

Fig. 69: In questo frame gli angoli non vengono rilevati per la mano in basso, che si muove velocemente. Cambiando i parametri è possibile rilevare la mano ma si aumenta anche la sensibilità al rumore.

FAST è un acrostico che sta per *Features from Accelerated Segment Test*. Il *segment test* è un criterio che, per ogni angolo candidato p, prende in considerazione una circonferenza (secondo Bresenham) di 16 pixel centrata in esso. Il candidato è classificato come angolo se esiste un arco di n pixel contigui sulla circonferenza che siano tutti *più luminosi* del pixel centrale sommato a una certa soglia t, oppure che siano tutti *più scuri* del pixel centrale meno una certa soglia t.

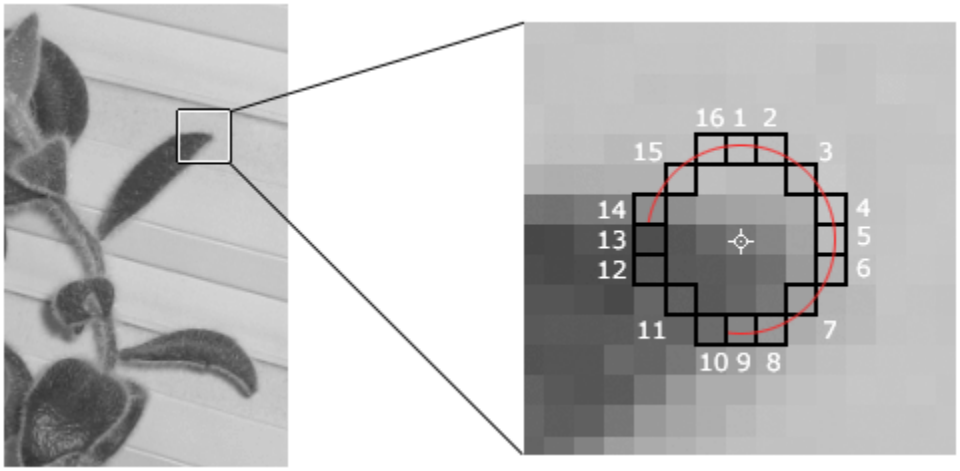


Fig. 70: Segment test con $n=12$. Il pixel centrale è un angolo candidato, i 12 pixel toccati dall'arco in rosso sono più luminosi del pixel centrale oltre una certa soglia.

La lunghezza dell'arco di circonferenza da considerare è un parametro, e per avere risultati ammissibili può essere fatto variare da 9 a 12 [27]; tuttavia, nel seguito fisseremo $n=12$, perché sperimentalmente si osserva che con questo valore è possibile escludere un gran numero di non-angoli.

Il test veloce comincia esaminando per prima cosa solo i quattro pixel 1,5,9 e 13 (le quattro direzioni cardinali). Se p è un angolo allora almeno 3 di questi devono essere più luminosi di I_p+t o più scuri di I_p-t , quindi se nessuno dei due casi è verificato p non può essere un angolo e viene immediatamente scartato. Altrimenti, si può procedere ad esaminare i pixel rimanenti lungo la circonferenza. Questo rilevatore di base esibisce già alte prestazioni, ma soffre di diverse debolezze:

1. Il test non può essere generalizzato bene per $n<12$
2. La scelta e l'ordinamento dei pixel per il test veloce contiene assunzioni implicite riguardo la distribuzione dell'aspetto delle feature
3. La conoscenza ottenuta dai test sui quattro punti cardinali non viene riutilizzata
4. Vengono rilevati angoli multipli adiacenti tra di loro

I primi tre punti di questo elenco possono essere affrontati mediante l'uso di un algoritmo di

apprendimento.

Il processo opera in due fasi. Per costruire un rilevatore di angoli per un dato n, prima gli angoli vengono rilevati da un insieme di immagini (preferibilmente dal dominio applicativo di interesse) usando il segment test per n e una soglia t conveniente. Questo test usa un algoritmo lento in cui per ogni pixel vengono testate tutte le 16 locazioni nella circonferenza che lo circonda.

Per ogni locazione sul cerchio $x \in \{1..16\}$, il pixel in quella posizione relativa a p (che denotiamo con $p \to x$) può essere in uno dei tre stati:

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t \quad (più \;\; scuro) \\ s, & I_p - t < I_{p\to x} < I_p + t \quad (simile) \\ b, & I_p + t \le I_{p\to x} \quad (più\, luminoso) \end{cases}$$

Scegliendo una x e calcolando $S_{p\to x}$ per tutti i punti p di tutte le immagini di training, si ottiene il partizionamento dell'insieme P (tutti i pixel di tutte le immagini) in tre sottoinsiemi P_d, P_s e P_b, dove ogni p viene assegnato a $P_{S_{p\to x}}$.

Sia ora K_p una variabile booleana che è vera se p è un angolo e falsa altrimenti. La seconda fase del processo inizia ora, selezionando la x con il maggior contenuto informativo riguardo il fatto che il pixel candidato sia o no un angolo, informazione misurata dall'entropia di K_p [27]. L'entropia di K per l'insieme P è:

$$H(P) = (c + \bar{c}) \log_2 (c + \bar{c}) - c \log_2 c - \bar{c} \log_2 \bar{c}$$

dove $c = |\{p | K_p \, è \, vera\}|$ (numero degli angoli)

e $\bar{c} = |\{p | K_p \, è \, falsa\}|$ (numero dei non-angoli)

La scelta di x dunque porta al guadagno di informazione:

$$H(P) - H(P_d) - H(P_s) - H(P_b)$$

Avendo ora selezionato la x che porta al maggior contenuto informativo, il processo viene

applicato ricorsivamente sui tre sottoinsiemi: x_b viene selezionata per partizionare P_b in $P_{b,d}$, $P_{b,s}$, $P_{b,b}$, x_s viene selezionata per partizionare P_s in $P_{s,d}$, $P_{s,s}$, $P_{s,b}$ e così via, dove ogni x è scelta come la x che produce più informazione sull'insieme a cui è applicata. Il processo termina quando l'entropia di un sottoinsieme è nulla, il che significa che tutti i punti p del sottoinsieme hanno lo stesso valore di K_p (cioè sono tutti angoli o tutti non-angoli). Questo avverrà sicuramente, dal momento che K è una funzione esatta dei dati di learning.

Il processo crea un albero decisionale che può classificare correttamente tutti gli angoli visti nel training set e dunque (con buona approssimazione) incorpora correttamente le regole del rilevatore di angoli. Questo albero decisionale può essere allora convertito in codice (nella fattispecie codice C), creando una lunga sequenza di istruzioni condizionali innestate che rappresentano cioè il rilevatore stesso.

Si noti che, poiché i dati non coprono completamente tutti gli angoli possibili, il rilevatore così ottenuto non è precisamente uguale al rilevatore di tipo segment test. D'altro canto, tutti i rilevatori di angoli hanno un certo grado di euristica, e un rilevatore siffatto è semplicemente un'euristica leggermente diversa dal segment test.

Resta comunque da affrontare il quarto punto degli svantaggi esposti dal rilevatore FAST (vengono rilevati molti angoli tra di loro adiacenti). Dal momento che il segment test non calcola una funzione di risposta C, non è possibile applicare direttamente una soglia sugli angoli rilevati; di conseguenza, deve essere definita una *funzione di score*, V, a cui poi applicare una forma di soppressione dei non-massimi, in cui gli angoli che hanno dei vicini con una V più alta vengono soppressi.

Ci sono diverse definizioni intuitive per V:

1. Il massimo valore di n per il quale p rimane un angolo
2. Il massimo valore di t per il quale p rimane un angolo
3. La somma della differenza assoluta tra i pixel nell'arco contiguo e il pixel centrale

Adottare una delle prime due definizioni potrebbe portare a più pixel che condividono lo stesso valore. Rimane la definizione 3. Per velocità di calcolo, ne viene data una versione leggermente modificata:

$$V = max\left(\sum_{x \in S_{luminosi}} |I_{p \to x} - I_p| - t, \sum_{x \in S_{scuri}} |I_p - I_{p \to x}| - t\right)$$

con $S_{luminosi}=\{x|I_{p\to x}\geq I_p+t\}$ e $S_{scuri}=\{x|I_{p\to x}\leq I_p-t\}$

Il rilevatore FAST offre già nella sua versione base (senza la parte di apprendimento) prestazioni più elevate dei rilevatori classici, e in seguito all'applicazione dell'algoritmo di apprendimento raggiunge anche velocità doppie [27]. Ciò lo rende un rilevatore adatto ad applicazioni real-time. Tuttavia, FAST non è un rilevatore robusto al rumore, fatto che non è comunque inatteso: dal momento che l'alta velocità raggiungibile deriva da un'analisi fatta sul minor numero possibile di pixel, la capacità di inibire l'influenza del rumore risulta di molto ridotta. Un secondo svantaggio risiede inoltre nella dipendenza, per n fisso, da un parametro di soglia. In sez. 3.3 verrà comunque presentato un sistema che fa uso di questo rilevatore di angoli per tracciare corpi in sequenze video, e verranno dati e discussi alcuni risultati sperimentali.

3.3 Tecniche per il rilevamento dei corpi

Il rilevamento di persone in folle dense non ha mai ricevuto molta attenzione poiché è allo stesso tempo un problema di segmentazione, riconoscimento e tracking. L'analisi automatizzata di folle è stata approcciata il più delle volte usando gli stessi modelli che si applicano ad immagini contenenti poche entità individuali. Le tecniche già esistenti si adattano diversamente a situazioni che prevedono un maggior numero di entità, ma generalmente non danno buoni risultati in presenza di folle piuttosto dense; in questi casi infatti ci sono così tanti individui che tecniche tradizionali come l'eliminazione dello sfondo falliscono del tutto nel cercare dei confini significativi tra le entità. In situazioni di alta densità, inoltre, hanno luogo più di frequente fenomeni svantaggiosi come occlusioni e variazioni di luminosità.

Brostow e Cipolla hanno ipotizzato invece che, in sequenze video, il solo fatto che entità separate si spostino in maniera indipendente può già offrire un'efficiente inizializzazione per il tracciamento in folle dense [29]. La premessa centrale dell'algoritmo è che una coppia di punti che si spostano *insieme* fa probabilmente parte dello stesso individuo. Questa assunzione è valida in diversi scenari e domini applicativi e può essere adottata con successo anche nel tracciamento di oggetti di diverso tipo: formiche, macchine, persone, mani.

In questa sede, il problema del tracking dei corpi è stato affrontato seguendo due approcci, il secondo dei quali trae l'idea del moto rigido da [29], dandone però uno svolgimento differente e diversamente ottimizzato.

3.3.1 Flusso Ottico

Il flusso ottico è un concetto che approssima quello del moto di oggetti all'interno di una rappresentazione visuale. Esso può essere cioè visto come una rappresentazione del moto apparente del mondo proiettato sul piano dell'immagine di una telecamera in movimento, e viene in genere rappresentato da un campo di velocità a due dimensioni. Più precisamente, il flusso ottico può risultare dal movimento della telecamera oppure dal movimento di oggetti all'interno della scena.

In termini semplici, il flusso ottico "insegue" tutti i pixel di un fotogramma nel fotogramma successivo. L'output è un insieme di vettori, uno per ogni pixel della scena. La tecnica consiste nel confrontare due frame e provare a mappare tutti i pixel del primo frame nel successivo, basandosi sul loro valore di intensità. Se un oggetto, ad esempio una sfera, perfettamente liscia, luminosa e ferma ruota sul proprio asse, non è percepibile alcun moto (cioè non c'è nessuno *moto apparente*), e perciò il flusso ottico non dà alcun risultato. Il problema di base risiede nell'assunzione di luminosità costante, secondo la quale i valori di luminosità di ogni pixel dell'immagine rimangono costanti nel tempo, anche se la loro posizione nello spazio può cambiare; il vero moto fisico degli oggetti nel mondo invece non sempre si riflette in cambiamenti di luminosità nelle immagini corrispondenti, così come i cambiamenti di luminosità non sempre sono dovuti al moto degli oggetti.

Gli algoritmi che si utilizzano di norma per stimare il flusso ottico si basano su questa assunzione e la sfruttano in diversi modi per calcolare un campo vettoriale di velocità che descrive le componenti orizzontale e verticale del moto per tutti i pixel nell'immagine. Il flusso ottico è difficile da calcolare, principalmente per due motivi: prima di tutto, il flusso ottico è *ambiguo* in zone dell'immagine pressoché omogenee (come nell'esempio della sfera fatto poco fa), poiché l'assunzione di luminosità costante viene continuamente soddisfatta in condizioni di moto. Inoltre, nelle situazioni reali questa assunzione è violata in una serie di istanze, anzitutto perché l'illuminazione non è detto che sia davvero sempre costante, e inoltre nei casi di moti non rigidi, di ombre proiettate, di problemi di trasparenza, riflessi, e sui bordi degli oggetti. Tipicamente queste "violazioni" vengono affrontate con approcci statistici che possono ad esempio arricchire l'assunzione di base tenendo conto dei diversi modi in cui l'illuminazione può cambiare all'interno di una sequenza video.

In generale ci sono tre algoritmi per il calcolo del flusso ottico:

1. *Block-matching*
2. Correlazione basata in frequenza

3. Algoritmi basati sul gradiente

Spesso questi algoritmi sono arricchiti dall'utilizzo di tecniche di più alto livello.

Il primo algoritmo considera una regione del frame di partenza, che viene confrontata con regioni della stessa dimensione nel secondo frame fino a quando viene trovato un vettore che minimizzi un certo criterio di errore (tipicamente il modulo della differenza). Questo approccio è abbastanza rapido ma non dà ottimi risultati (viene ad esempio utilizzato come base per la compressione MPEG).

Il secondo metodo è migliore: il suo funzionamento è simile a quello del *block-matching*, ma i blocchi di pixel vengono confrontati nel dominio delle frequenze, il che porta a risultati più accurati. È un algoritmo abbastanza complicato dal momento che richiede molte FFT e prodotti tra di esse, ma si presta bene a una realizzazione hardware.

Il terzo algoritmo dà i risultati migliori. L'idea è quella di prendere due blocchi di pixel da frame adiacenti e usare l'informazione del gradiente per procedere iterativamente a trovare una soluzione in cui i pixel dei due blocchi sono allineati. Tipicamente questo metodo è implementato secondo un approccio gerarchico (in multirisoluzione) secondo il quale il matching viene prima effettuato a scale alte (risoluzioni basse) per poi procedere con le scale più basse. Questo metodo può essere computazionalmente costoso dal momento che ogni blocco a ogni risoluzione richiede un certo numero di iterazioni per raggiungere la convergenza.

In questa tesi viene utilizzata un'implementazione del flusso ottico basata sull'algoritmo di registrazione di Lucas e Kanade (sez. 3.1) per il tracking di feature tra frame consecutivi di una sequenza video. Le due tecniche di tracking di corpi che verranno discusse nel seguito si basano pesantemente sul calcolo del flusso ottico per feature e per gruppi di feature.

3.3.2 Clusters Tracking

Come detto, nell'ambito di questa tesi sono stati realizzati due approcci differenti al problema del tracking di corpi (o, più genericamente, *entità*) all'interno di sequenze video.

Questo primo approccio consiste nell'inseguimento di gruppi di feature individuati nella sequenza video. Il metodo può essere denominato *clusters tracking*, e funziona come descritto di seguito.

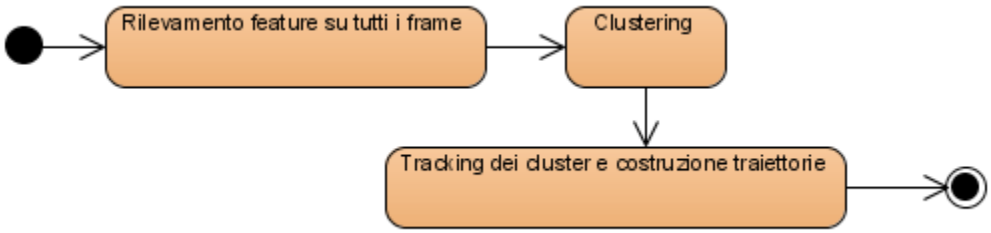


Fig. 71: Metodo di "clusters tracking"

Come primo passo vengono *rilevate le feature* di Rosten-Drummond per tutto il video, secondo i parametri specificati dall'utente (per dettagli, vedere il capitolo 3.2.1). Dopodiché, per ogni frame vengono considerate le feature rilevate, le quali vengono *raggruppate in diversi cluster* secondo un certo algoritmo. Le tecniche di clustering si raggruppano tipicamente in due categorie: tecniche *gerarchiche* e tecniche *non gerarchiche*. Il primo gruppo comprende quelle tecniche in cui i dati di input vengono raggruppati in cluster a passaggi successivi, fino a quando si è ottenuto il numero finale di cluster (numero che *non* è deciso a priori). La seconda categoria comprende invece le tecniche che, partendo da un partizionamento iniziale corrispondente al numero desiderato di cluster, procedono con la rilocazione dei punti tra i cluster di modo che un dato criterio sia ottimizzato. Per questa fase, il software sviluppato in questa sede implementa un algoritmo denominato *Minimal Spanning Tree* (o MST, [15]), che fa parte del primo gruppo. L'algoritmo comincia considerando ogni singola feature come un cluster a sé stante. Poi, i due cluster più vicini (secondo la semplice distanza euclidea tra i centroidi degli stessi) vengono fusi in un singolo cluster. Il processo continua fino a quando tutti i punti appartengono allo stesso cluster, oppure fino a quando si è raggiunto un certo numero di cluster finale, determinato in base a un criterio fissato. Questo criterio tiene conto del fatto che due cluster, per essere considerati *vicini* e dunque per poter essere fusi insieme, non possono essere distanti più di una certa soglia specificata dall'utente, e dipendente dal tipo e dalla dimensione degli oggetti che si vogliono identificare. Chiaramente, per natura stessa dell'algoritmo, il numero finale di cluster per ogni frame non è noto a priori, quindi non è detto che passando da un frame al successivo i cluster del primo vengano ritrovati (mantenuti) nel secondo.

Procedendo con il tracking dei corpi, una volta “clusterizzato” ogni frame di feature, il video viene ripercorso a partire dal tempo zero e ogni cluster viene inseguito nei frame successivi, fino a quando il video termina oppure fino a quando il cluster viene "perso". *Per ogni cluster inseguito viene creata una traiettoria*, che ne rappresenta il moto nel tempo. Come accennato, per il tracking vengono considerati solo i frame successivi a quello del cluster corrente. Questa scelta è motivata dal fatto che tutti i cluster precedenti sono già stati analizzati e in quanto tali fanno già parte di qualche traiettoria; quindi è inutile cercare il cluster corrente nei frame passati perché se

ci fosse (evento del tutto probabile, anzi benvenuto), sarebbe già parte della traiettoria di qualche cluster precedente. Il tracking di per sé è effettuato tramite un'implementazione del flusso ottico secondo l'algoritmo iterativo di Lucas-Kanade (sez. 3.1). Il successo di questa fase è determinata da quanti punti è stato possibile inseguire da un frame all'altro: se, ad esempio, di un cluster di 20 feature, solo 2 ne sono state "ritrovate" nel frame successivo, allora questo cluster viene considerato "perso" e la sua traiettoria termina col frame corrente. Se il cluster invece non è andato perso, allora occorre verificare che nel secondo frame esista un cluster (ottenuto all'inizio del processo, durante la fase di clustering dei frame) che ricopra totalmente i punti inseguiti correttamente col flusso ottico. A questo punto si distinguono i due casi:

1. Esiste effettivamente un cluster nel secondo frame che ricopra totalmente il cluster inseguito dal primo frame;
2. Tale cluster ricoprente non esiste: il primo cluster è stato inseguito con successo dal flusso ottico, ma nel passaggio da un frame all'altro il primo cluster ha "disperso" le sue feature, alcune delle quali sono state accorpate a un cluster differente.

Nel primo caso è possibile continuare con il tracking del cluster, procedendo con la costruzione della sua traiettoria. Nella seconda evenienza invece, sebbene l'applicazione del flusso ottico abbia avuto successo, il cluster di partenza è considerato perso in quanto non è più possibile stabilire con una certa affidabilità il suo moto nel tempo. Quindi la costruzione di una traiettoria per il cluster corrente ha termine.

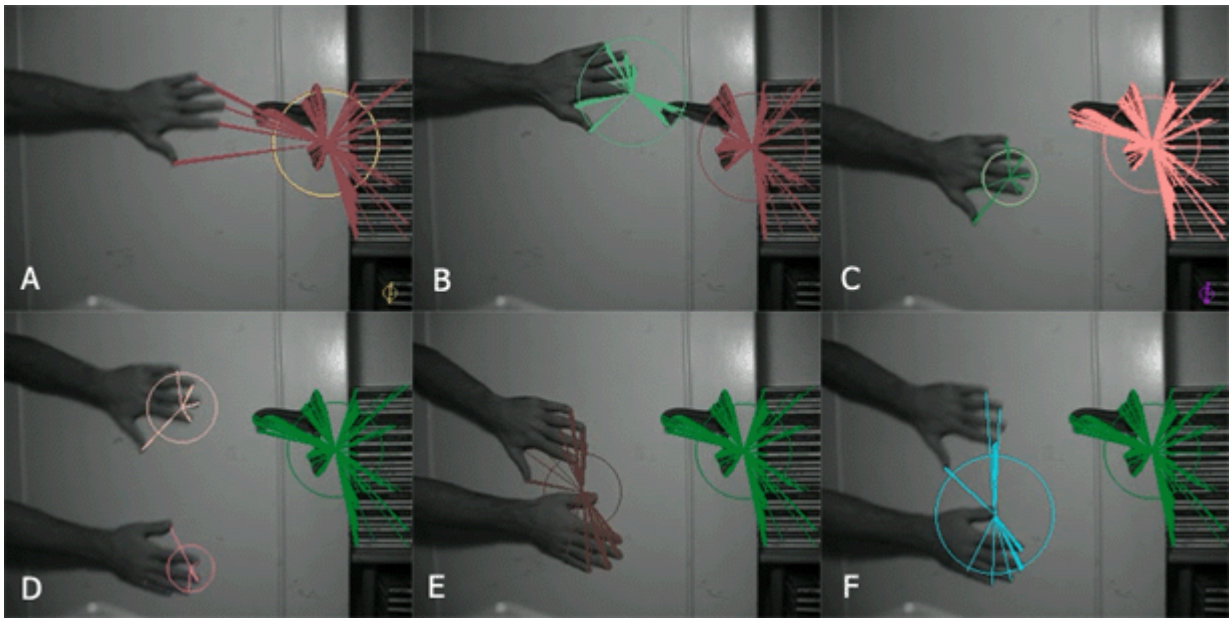


Fig. 72: Il metodo di tracking dei cluster permette di individuare corpi piuttosto stabili, ma fallisce quando le traiettorie si avvicinano tra di loro

Una volta create le prime traiettorie, il processo di costruzione delle stesse si snellisce sempre di più in quanto, man mano che si procede con il tracking dei cluster, questi vengono prima cercati all'interno della lista delle traiettorie già esistenti, e se presenti in lista non vengono più inseguiti (dal momento che questa operazione è già stata svolta partendo da un cluster precedente). Si osservi che se si proseguisse con il tracking dei cluster *già presenti* in una traiettoria, si commetterebbe un errore, in quanto si darebbe vita possibilmente a traiettorie duplicate (poco probabile) o a sotto-traiettorie di traiettorie già rilevate (evento certo).

Al termine dell'intero processo di *clusters tracking* si ha una lista di traiettorie, ciascuna rappresentante il moto di un singolo cluster lungo la sequenza video.

In fig. 72 è possibile visionare alcuni fotogrammi tratti da un video risultante dall'applicazione del metodo corrente. La distanza massima tra cluster è stata impostata a 70 pixel. I frame mostrano la debolezza principale del metodo: sebbene i corpi individuati siano relativamente stabili (l'aggiunta di feature a un cluster non invalida il cluster, bensì lo popola), quando due corpi si avvicinano troppo (cioè quando i centroidi sono distanti meno di 70 pixel) vengono accorpati in un nuovo cluster, che vive fin tanto che i cluster originari stanno vicini (fotogrammi A, E e F). Se, invece, durante il video i cluster non si avvicinano mai troppo, il metodo dà risultati soddisfacenti ed è piuttosto robusto al rumore (frame B, C e D).

3.3.3 Rilevamento di moto coerente

Il secondo procedimento può prendere il nome di *coherent motion detection*. Questo metodo prende da [29] l'idea di base secondo la quale due feature possono essere considerate parte dello stesso corpo *se si spostano insieme*.

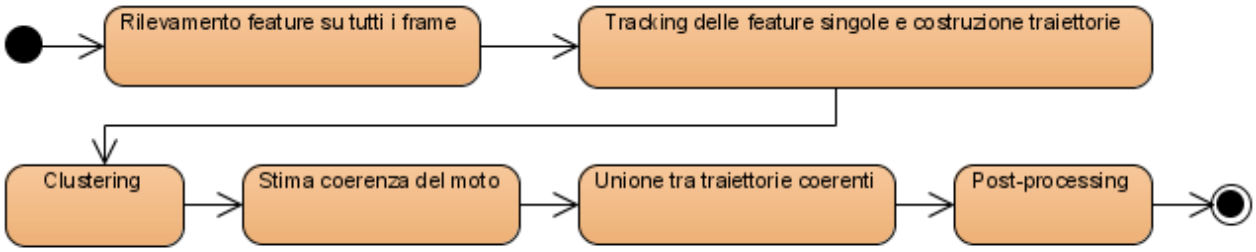


Fig. 73: Schema per il metodo di rilevamento di moto coerente

Una feature puntiforme x percorre una traiettoria X quando tracciata nel tempo. Differentemente dal primo metodo (nel capitolo precedente), dopo aver rilevato gli angoli di Rosten-Drummond per l'intero video, il rilevamento ha inizio con il tracking *delle singole feature* per ogni frame, senza alcun tipo di clustering spaziale: per ogni feature x_i del primo frame della sequenza viene da subito inizializzata una traiettoria. Ognuna di queste feature di partenza prende il nome di *seed*, e verrà tracciata indipendentemente dalle altre lungo il video; è chiaro che durante l'elaborazione della sequenza di fotogrammi potranno crearsi nuovi seed, evenienza che avrà luogo quando si incontreranno nuove feature non facenti parte di alcuna traiettoria preesistente. Una volta inizializzate le traiettorie X_i, ha inizio la fase fondamentale dell'algoritmo, che consiste nel calcolo del flusso ottico per ogni coppia di frame adiacenti nel video, fino all'ultimo fotogramma. Procedendo con la stima dei vettori di velocità delle x_i, le diverse traiettorie prendono forma, e laddove una feature risulta "persa" (cioè il flusso ottico non rileva più moto), la sua traiettoria risulta completata. Come detto, al procedere dell'elaborazione è possibile incontrare nuove feature che non fanno parte di alcuna traiettoria esistente, e che quindi vengono trattate come nuovi seed, inizializzatori di nuove traiettorie.

Ora, l'idea che ci permette di asserire l'appartenenza di due feature allo stesso corpo è la seguente: se le traiettorie delle due feature si sovrappongono nel tempo (cioè le due feature esistono negli stessi intervalli di tempo lungo il video), e se queste traiettorie sono circa uguali tra loro nell'arco della loro esistenza, allora vuol dire che descrivono lo stesso movimento, e probabilmente si riferiscono perciò allo stesso corpo. D'altro canto, questa affermazione ha senso

solo se le due feature sono comunque abbastanza vicine tra loro: potrebbe essere il caso infatti che due corpi distinti seguano casualmente la stessa traiettoria negli stessi istanti, ma in posizioni della scena lontane tra di loro (fig. 74).

Fig. 74: Autovetture che corrono su un'autostrada. Questo è un tipico esempio in cui è possibile avere corpi distinti che si spostano insieme rigidamente lungo l'intera traiettoria.

Per questo motivo, prima di cercare similarità tra le traiettorie, viene effettuata una *clusterizzazione delle feature* per ogni frame, che tenga conto della massima copertura spaziale che un corpo può avere dipendentemente dal tipo di target per il rilevamento. In questo modo non solo diminuiscono di molto i tempi di calcolo, ma anche la precisione dell'algoritmo migliora in quanto vengono eliminati del tutto falsi accoppiamenti.

A questo punto *viene stimata la coerenza del moto* tra tutte le feature potenzialmente accoppiabili, quelle feature cioè *che coesistano nel tempo* e che inoltre *facciano parte dello stesso cluster* il più a lungo possibile: infatti, se due feature appartengono allo stesso cluster nei primi 10 frame, non è detto che la cosa sarà vera anche per il resto del video. Il rumore, ad esempio, o gli improvvisi cambiamenti di luminosità, possono dare luogo a nuove feature e eliminarne di altre (un fascio di luce su un campo d'erba può nascondere le texture!). O ancora, può essere il caso che due corpi che si spostano insieme a distanza ravvicinata per molto tempo, poi si separino. Quindi due feature sono considerate parte dello stesso cluster solo se lo condividono per la maggior parte della loro esistenza, fatto espresso da un parametro di sensibilità lasciato all'utente e che si auspica il più alto possibile (ad esempio, il 95% del tempo).

La stima della coerenza del moto viene effettuata calcolando la distanza euclidea (D) tra due feature per ogni istante in cui queste coesistono, e dunque prendendo la varianza di questa misura, utilizzando la formula:

$$Var\{D\} = \frac{\sum_{j=0}^{N} (\bar{D} - D_j)^2}{N}$$ con N pari al numero di frame in cui le feature coesistono

A partire dalla varianza viene dunque calcolata la *probabilità di moto rigido* per la coppia di feature:

$$Q = \frac{1}{1 + Var\{D\}}$$

Quantità chiaramente compresa tra 0^+ e 1. Idealmente, due feature che si spostano in un corpo rigido comportano $Q = 1$, in quanto restano sempre a una distanza costante. In realtà, questo è vero solo per movimenti paralleli al piano dell'immagine, ma è stato dimostrato essere comunque un'approssimazione accettabile, in parte perché la scala prospettica varia poco quando la sovrapposizione di traiettorie è più breve. Se la Q è superiore a una certa soglia (definita dall'utente) allora le due feature possono essere considerate caratterizzanti lo stesso corpo, e le loro traiettorie possono essere "fuse" insieme: il risultato di questa operazione è la creazione di una traiettoria per un *cluster* di punti, formato dall'unione delle due feature. Procedendo con queste unioni si giungerà al caso in cui due traiettorie già fuse tra di loro dovranno essere fuse ad un'ulteriore traiettoria (il cui moto risultava coerente con una delle due, se non entrambe), formando in questo modo cluster sempre più "corposi", e il cui centroide si muove seguendo una traiettoria altamente simile alle traiettorie di *tutte* le feature che formano il cluster.

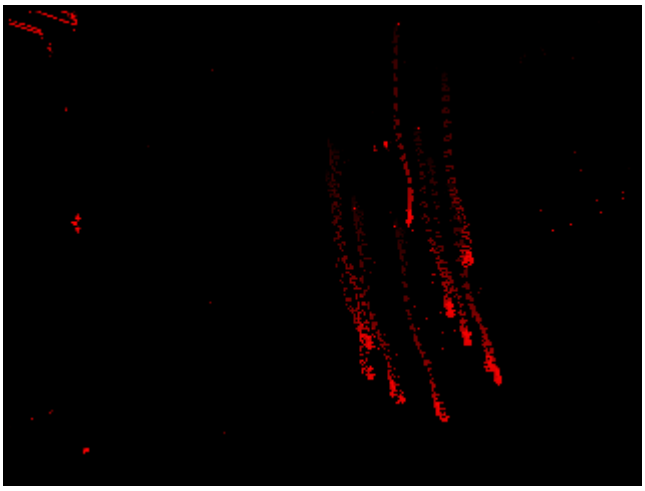


Fig. 75: Traiettorie per un video di esempio in cui una mano si sposta dall'alto verso il basso; le altre traiettorie sono dovute al rumore.

Bisogna osservare che, al termine di questa fase, è anche possibile che vi siano feature non ancora accoppiate: questa evenienza è da attribuire principalmente alla presenza di feature dovute al rumore o ad oggetti statici sottoposti a diverse condizioni di illuminazione.

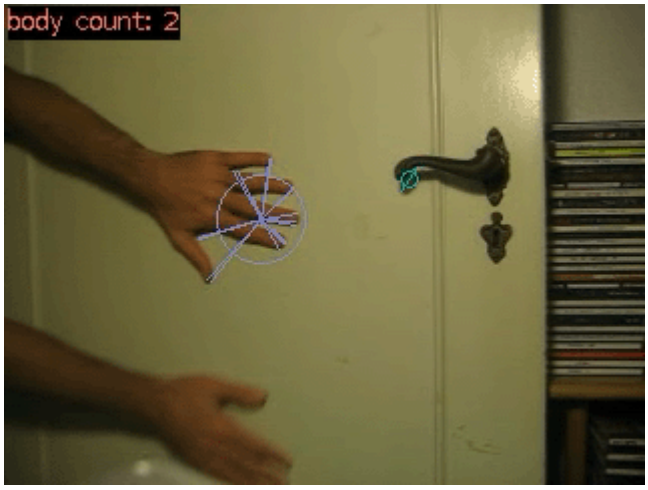


Fig. 76: In questo fotogramma vengono rilevati due corpi, uno dei quali esiste per soli 3 frame in corrispondenza della maniglia della porta.

Il rumore dovuto ai cambiamenti di illuminazione è anche responsabile di curiosi artifici che possono dare vita a entità semoventi in quei casi in cui, fortuitamente, delle *noise features*

abbiano un moto coerente tra di loro. Per questi ed altri motivi è necessaria una forma di *post-elaborazione* dei risultati che tenga conto di eventualità di questo tipo e che ne inibisca l'azione il più possibile.

Di seguito vengono elencate, nell'ordine in cui vengono applicate, le strategie principali di post-processing adottate dal software realizzato (sez. 4.3):

1. **Fusione delle traiettorie vicine**

   Se due traiettorie non mostrano tra di esse coerenza del moto, ma per molto tempo le loro feature sono *molto* vicine, allora vengono fuse: questa operazione permette di irrobustire il rilevamento di corpi quando sottoposti a rumore o variazioni di illuminazione (ad esempio quando l'ombra di un corpo viene proiettata su di un altro)

2. **Giunzione delle traiettorie spezzate e gestione delle entrate/uscite**

   I corpi non possono crearsi dal nulla, né svanire nel nulla: se una nuova traiettoria parte dal centro della scena (cioè non dai suoi confini, entro fissati limiti di tolleranza), o termina al centro di una scena, allora viene cercata una traiettoria "vicina" ad essa nello spazio e nel tempo: se, ad esempio, un frame dopo che una traiettoria termina nel centro di una scena, circa in quella posizione ne nasce un'altra, allora le due traiettorie vengono fuse in una unica. Questo fatto può accadere quando un corpo si muove troppo velocemente: in questo caso infatti il corpo risulta sfocato e il rilevatore di Rosten-Drummond non si comporta bene e perde degli angoli (sez. 3.2.1). Può inoltre avere luogo sempre in presenza di rumore, che si manifesta sotto forma di traiettorie di durata molto breve, che nascono e muoiono in posizioni arbitrarie della scena. Si ricordi inoltre che per un rilevatore basato unicamente su feature (e quindi *non* sull'aspetto), oggetti relativamente privi di texture rappresentano un problema: una mano aperta ha pochi angoli, precisamente le 5 punte delle dita e le 4 giunzioni tra le stesse; quindi, una mano che si sposta velocemente può essere difficile da tracciare.

   La vicinanza nello spazio è limitata da un valore di tolleranza specificato dall'utente, e chiaramente dipendente dal tipo degli oggetti che si intende rilevare e dalle caratteristiche geometriche della scena (angolo dell'inquadratura, zoom, etc.). La prossimità nel tempo dipende anche da una certa soglia, che determina il numero di frame successivi/precedenti in cui andare a cercare una traiettoria che nasce quando quella corrente muore, o che muore quando quella corrente nasce.

   Si noti che questa forma di post-elaborazione assume che le entrate e le uscite avvengano ai confini dell'immagine, cosa ovviamente non sempre vera e che andrebbe gestita adeguatamente.

3. **Eliminazione delle brevi traiettorie**

   Come detto in precedenza, le feature dovute a rumore danno luogo tipicamente a traiettorie molto brevi (da 1 a 3 frame di durata in media), e in quanto tali è difficile (anche se può accadere) stabilire relazioni di rigidità del moto con esse; di conseguenza il rumore risulta praticamente sempre isolato. Questa forma di auto-filtraggio può essere sfruttata semplicemente eliminando tutte le traiettorie brevi. Nell'operare questa cancellazione va tenuto conto dei seguenti fatti: una traiettoria breve può in realtà rappresentare un corpo che entra e subito esce dalla scena, oppure può essere parte di una traiettoria lunga *spezzata* ad esempio dal *blurring* (dovuto a un movimento veloce, vedere il secondo punto). Nel primo caso, occorre prestare attenzione alla definizione di *breve*, scegliendo un valore adeguato per il parametro che stabilisce il numero massimo di frame per una traiettoria di questo tipo. Nel secondo caso invece, la speranza è che l'ottimizzazione enunciata al punto 2 sia stata sufficiente a ricollegare le traiettorie spezzate. Altrimenti, quello che si ottiene è che un corpo che in realtà percorre un lungo percorso venga identificato come *più corpi* che nascono e muoiono in coda, lungo una certa traiettoria.

4. **Cancellazione dei corpi immobili**

   A questo punto non è certo che il sistema abbia eliminato del tutto il rumore: sebbene le tre ottimizzazioni precedenti riducano notevolmente la sua influenza sul rilevamento dei corpi (influenza misurabile con buona approssimazione dalla presenza di falsi positivi), nessuna di esse ha affrontato direttamente il caso di rumore consistente circoscritto a regioni precise dell'immagine. Ad esempio, un panno o un telo su cui si spostano dei corpi pesanti è costantemente sottoposto a deformazioni e sgualciture: una sgualcitura può essere facilmente rilevata da Rosten-Drummond e può facilmente essere considerata come un corpo a sé stante; di conseguenza, in corrispondenza di essa verrà rilevata un'entità indipendente, la cui posizione rimane pressoché costante nel tempo. Allo stesso modo, il rumore casuale potrebbe persistere per più di un frame dando vita a traiettorie brevi e quasi immobili. Per questi motivi l'ultimo passo della fase di post-elaborazione consiste nell'eliminazione dei corpi immobili, rappresentati da traiettorie quasi statiche e il cui massimo spostamento può essere fissato dall'utente tramite un parametro (ad esempio, se una traiettoria ha uno spostamento massimo di 5 pixel allora può essere considerata statica). È da tenere a mente che, sebbene questo procedimento potrebbe portare a falsi negativi, eliminando ad esempio persone sedute in un task di rilevamento di individui, il sistema stesso è per sua natura concepito per identificare entità utilizzando il solo suggerimento del moto, ed è pertanto inevitabile e anzi implicito: un rilevatore siffatto *non può* rilevare oggetti fermi, e dunque questi vanno eliminati.

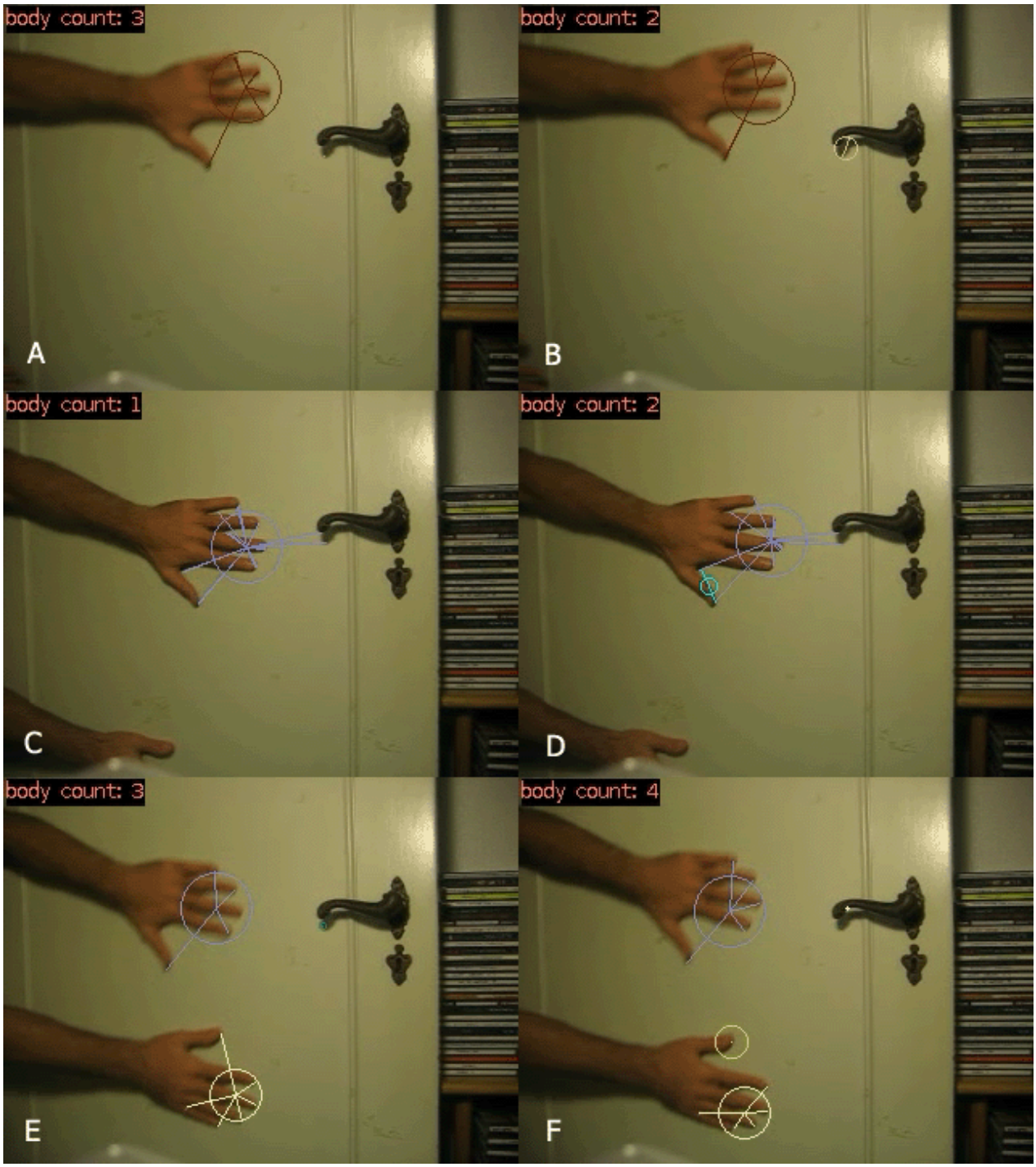


Fig. 77: In un'applicazione di conteggio dei corpi, il rumore e gli oggetti statici sottoposti a diverse condizioni di illuminazione provocano oscillazioni del conteggio. Ogni riga mostra due frame successivi tratti dallo stesso video.

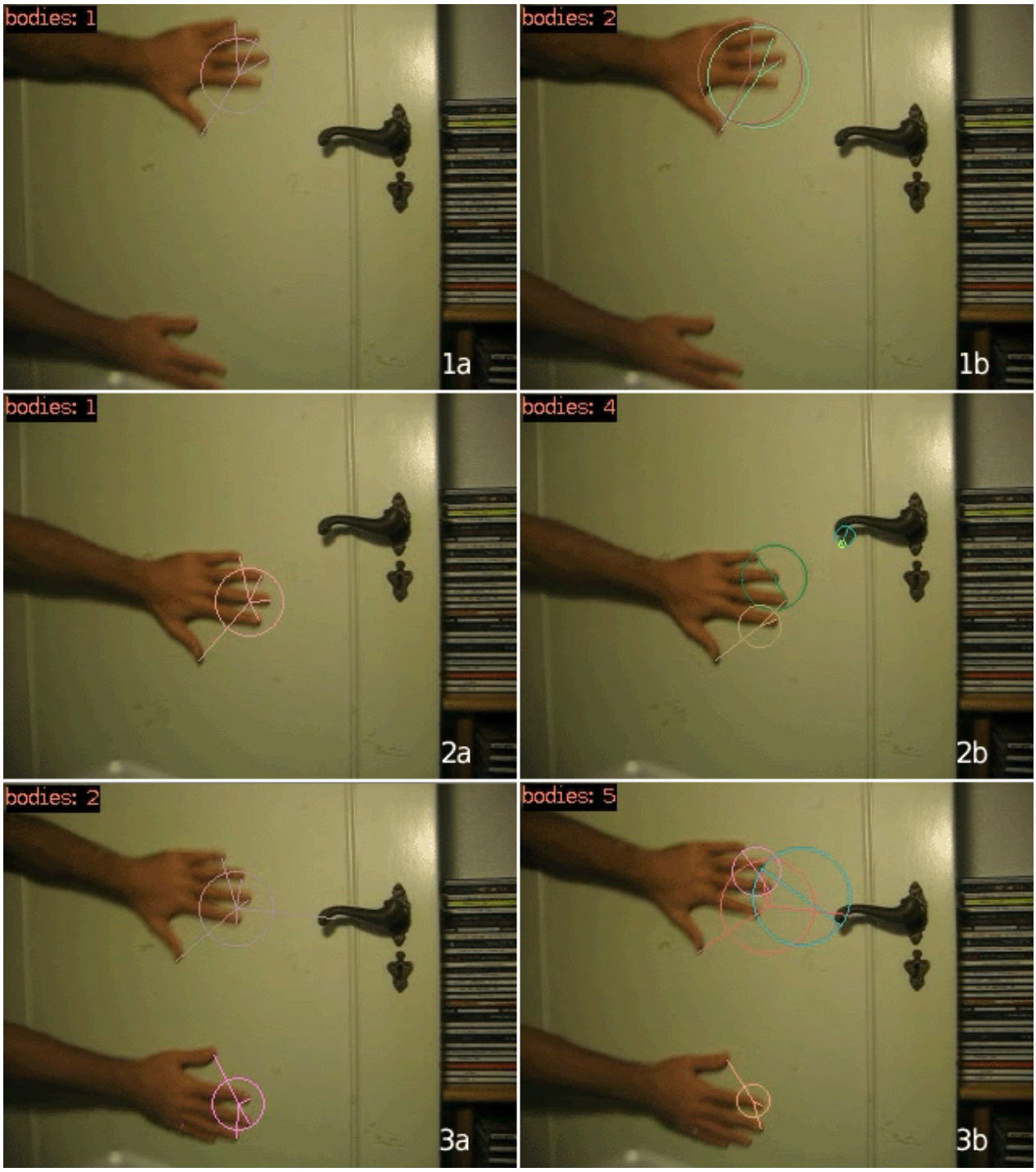


Fig. 78: La fase di post-elaborazione migliora di molto i risultati. Ogni riga mostra il risultato del sistema di tracking con (colonna sinistra) e senza (colonna destra) applicazione dei passi di post-processing, in un'applicazione di conteggio automatico dei corpi.

La fase di post-elaborazione migliora di molto i risultati (vedere fig. 77). In un'applicazione di conteggio di corpi frame per frame all'interno di un videoclip, il numero è esatto a meno di

piccole oscillazioni. Queste sono dovute principalmente ai seguenti motivi:

- Le feature di Rosten-Drummond sono pensate per un rilevamento in real-time e pertanto non sono ben tracciabili (sez. 3.2.1), inoltre sono troppo sensibili al rumore
- I corpi effettivi sono più piccoli o più grandi di quanto previsto dalla parametrizzazione corrente, o in generale i parametri sono stati scelti male
- Il metodo è inerentemente poco preciso dal momento che non viene considerato per nulla l'aspetto delle entità tracciate

I limiti di questo approccio basato unicamente sul moto non sono inattesi. Anzitutto, se un individuo o un oggetto semovente è camuffato in maniera tale che non siano facilmente identificabili delle feature da tracciare (ad esempio nel caso di un uomo che indossa una maglietta monocolore), allora la fase di clustering non produrrà alcun gruppo significativo ignorando sostanzialmente il corpo. Inoltre, dato che due feature sono classificate come appartenenti allo stesso corpo solo se si spostano insieme rigidamente, si hanno frequenti falsi positivi quando ad esempio si ha a che fare con scene in cui delle persone camminano portando in mano giornali, o o altri oggetti, che vengono identificati come oggetti separati in quanto dotati di movimento rigido indipendente dallo spostamento del corpo umano (in quanto, per via del movimento delle braccia, oscillano rispetto ad esso). Falsi negativi vengono invece rilevati quando due (o più) corpi si spostano fianco a fianco e con lo stesso passo. Con questo metodo, non c'è da aspettarsi buoni risultati da sequenze video che, ad esempio, riprendono una truppa che marcia. Infine, si tenga a mente che mentre l'algoritmo usa feature tracciate su sequenze di frame, il clustering e il rilevamento di individui di per sé avvengono indipendentemente su ogni frame, e cioè il sistema *non* traccia le entità.

In conclusione, sarebbe opportuno considerare questa tecnica per fornire una buona inizializzazione a un rilevatore "ibrido" che tenga conto anche dell'aspetto delle entità da tracciare, considerazioni da cui si può trarre sicuramente gran beneficio. Nella sezione successiva vengono proposte alcune soluzioni di questo tipo, tenendo conto, in particolare, di quanto esposto nel capitolo 2.3 e in quello presente.

3.4 Rilevamento di persone basato sull'aspetto

Molti rilevatori di corpi all'interno di sequenze video fanno uso esclusivamente di

informazioni sull'*aspetto* delle entità da inseguire; alcuni di questi approcci comprendono una fase di apprendimento in fase di esecuzione, di modo tale che il rilevatore sia in grado di volta in volta di "imparare" gli oggetti che analizza, altri rilevatori invece vengono dotati di conoscenza pregressa (ad esempio fanno uso di modelli di base per gli oggetti da rilevare) in base alla quale fare poi delle decisioni in fase di rilevamento.

In questa sezione viene prima mostrato brevemente un sistema di rilevamento di corpi esclusivamente basato sull'aspetto; quindi, vengono proposte delle soluzioni che utilizzano congiuntamente le informazioni sull'aspetto e le tecniche di feature tracking presentate in precedenza.

3.4.1 Rilevamento basato sulla simmetria verticale

A titolo di esempio, viene esposto di seguito un approccio interessante, adottato dal sistema realizzato per il veicolo ARGO [28]: questo sistema sfrutta le caratteristiche morfologiche e la forte simmetria verticale del corpo umano per il rilevamento e il riconoscimento di pedoni. Il metodo si è rivelato robusto alle diverse pose e posizioni assumibili dai pedoni, così come ai vestiti indossati.

L' ipotesi principale su cui si basa l'algoritmo è che una persona può essere caratterizzata da:

- principalmente bordi verticali, con una forte simmetria rispetto all'asse verticale
- dimensioni e rapporti di aspetto entro vincoli specifici
- posizione in una specifica regione dell'immagine (un pedone non può trovarsi, ad esempio, sopra a un lampione)

Con queste assunzioni, viene identificata un'area di interesse sulla base di vincoli prospettici e considerazioni pratiche; dopodiché vengono estratti i bordi verticali, eliminati per quanto possibile gli oggetti appartenenti allo sfondo (facendo per lo più considerazioni euristiche), e vengono considerate le aree che presentano un'alta simmetria verticale. A questo punto per ogni area viene cercata, in una posizione approssimativa, la testa del pedone attraverso una misura dell'affinità con un semplice modello. Infine, solo i candidati che soddisfano vincoli specifici sulla dimensione e sull'aspetto vengono classificati come pedoni.

Questo metodo ha buone prestazioni ed è abbastanza rapido, per cui si presta bene ad una

realizzazione real-time. Il beneficio maggiore è apportato dalle considerazioni fatte sulla simmetria dei contorni verticali e dai vincoli imposti ai rapporti di altezza e larghezza, operazioni che possono essere svolte in maniera molto rapida e che eliminano diversi possibili match. Al confronto, gli approcci tradizionali basati sulla semplice inter-correlazione sono più esigenti sui tempi di calcolo, anche se negli ultimi anni si è resa disponibile una soluzione in frequenza per questa operazione, che snellisce in buona misura i tempi di elaborazione.

3.4.2 Costruzione di un rilevatore combinato

Come detto a inizio capitolo, non tutte le parti di un'immagine contengono informazioni per il moto. In termini generali, la strategia che consente di superare questa difficoltà è quella di usare, in fase di tracking, solo regioni ricche di texture. D'altro canto, affidarsi solamente ai suggerimenti forniti dal movimento di cumuli di feature non permette di discernere con alta affidabilità entità diverse, sebbene fornisca un'ottima inizializzazione. Un possibile miglioramento consiste dunque nell'adozione di un approccio che tenga conto anche dell'aspetto delle entità da tracciare. Di seguito vengono proposte alcune idee basate su quanto discusso finora; tuttavia, i metodi che seguono non sono stati sperimentati e di conseguenza rimangono un utile suggerimento per realizzazioni future.

L'obiettivo del rilevatore è, come prima, riconoscere i corpi presenti in sequenze video. Il primo passo consiste nell'estrazione delle feature *e* dei contorni per il frame corrente. L'informazione sui contorni viene utilizzata per identificare percorsi *chiusi* di pixel; ad ogni area così ottenuta viene associato, temporaneamente, un identificativo per il corpo che potrebbe rappresentare. Dopodiché, per ogni area vengono identificati i cluster di punti compresi, e per essi viene effettuato il tracking utilizzando una delle tecniche di sez. 3.3. Se, durante l'inseguimento, il cluster osservato appartiene per un'alta percentuale di tempo alla stessa area, allora con alta probabilità rappresenta un'entità indipendente e può essere marcato con un codice definitivo.

Chiaramente, un procedimento di questo tipo richiede che i contorni estratti siano il più possibile lunghi e contigui. Inoltre, la posizione dei contorni deve essere piuttosto precisa e robusta a bordi vicini (vedere fig. 47), in quanto è importante che corpi molto prossimi non si sovrappongano (così facendo si può fare a meno di un parametro, quello che rappresenta la massima distanza tra due cluster di punti). In altre parole, il rilevatore in multirisoluzione discusso in sez. 2.3.2 potrebbe fornire i risultati migliori, in quanto consente di estrarre contorni solidi nelle loro posizioni “esatte”.

Fig. 79: Esempio di contorni che racchiudono cluster di punti

Pur disponendo di buoni contorni, tuttavia, rimangono aperti due problemi:

- È necessario inseguire i contorni stessi per avere poligoni affidabili nel tempo
- Sebbene possibile, è altamente improbabile che si riescano a formare percorsi chiusi completi

Il primo punto pone a sua volta diverse domande, ed è argomento di fervida ricerca nell'ambito della Computer Vision. Le tecniche tradizionali di tracking, sostanzialmente basate sul flusso ottico, non funzionano in questo caso, in quanto un bordo non è (per sua natura) un pattern ben tracciabile. Ciò che può essere fatto è tracciarne i punti terminali (in maniera simile a quanto avviene nel rilevatore di sez. 2.2.4) nel tempo. Nel fare questo, è pensabile anzi utilizzare lo stesso rilevatore di Shih e Tseng (sez. 2.2.4) per l'estrazione nel singolo frame. Si ricorderà che il difetto principale del metodo suddetto consiste nella limitata capacità di rilevare bordi vicini; questo limite, in realtà, può essere tollerato. Di fatti, sebbene sia piuttosto importante discriminare bene due corpi distinti facendo in modo che i contorni non si sovrappongano, occorre tenere conto che, per via del rumore e degli errori intrinsechi di acquisizione dell'immagine, durante la scena video i corpi perdono e acquisiscono feature in continuazione. In più, gli oggetti stessi possono mutare aspetto durante il moto; queste osservazioni suggeriscono che un contorno chiuso debba essere abbastanza flessibile da ammettere cluster di punti

“dinamici”, cioè gruppi di pixel il cui centroide si muove lungo una traiettoria minimamente oscillante, entro determinati limiti di tolleranza.

Il secondo punto è probabilmente più difficile da affrontare. Sebbene alcuni dei rilevatori visti nel cap. 2 permettano di ottenere contorni molto lunghi, sperimentalmente si osservano, il più delle volte, percorsi spezzati. Un metodo per determinare, approssimativamente, l'area che racchiude un certo cluster di punti, consiste nell'applicazione di una forma di *flood-fill*. Un algoritmo di questo tipo consente di determinare l'area connessa a un dato nodo entro un vincolo di tolleranza. Una volta effettuato l'”allagamento”, facendolo partire dal centroide, è possibile restringere la zona riempita cercando contorni solidi lungo le quattro direzioni cardinali. Oppure, è possibile procedere con approcci basati su un modello come la *trasformata di Hough* [4].

In conclusione, il sistema sopra proposto consiste nel cercare poligoni che racchiudano cluster di punti ben tracciabili nel tempo; se il metodo ha successo, allora può essere certamente utilizzato per migliorare gli algoritmi di rilevamento basati su coerenza del moto (sez. 3.3.3).

Un secondo sistema può fare uso della conoscenza sull'aspetto degli oggetti da tracciare tenendo conto della distribuzione tipica delle feature per determinate classi di entità. A partire da un modello di distribuzione dei cluster, potrebbe essere possibile infatti discriminare un uomo da un cavallo, semplicemente valutando i rapporti tra le distanze dei centroidi e sulla popolosità relativa dei gruppi di feature. Questo tipo di rilevamento ovviamente assume che gli oggetti in analisi siano semplici e “puliti”: fenomeni come il cambiamento di postura, variazioni di luminosità, e presenza di texture (un uomo a petto nudo non esibisce le stesse feature di un uomo che indossa una maglietta) ne inficiano l'efficacia.

Capitolo 4: Software

Nel contesto della presente tesi, sono state sviluppate tre distinte applicazioni che realizzano quanto esposto in alcune delle sezioni precedenti.

Il software è stato realizzato in C/C++ e in tutti e tre i casi fa uso del framework Qt (alla versione 4.3.0, [19]) per la realizzazione della GUI e l'utilizzo di thread di lavoro separati. Il codice che realizza le funzionalità di interesse è completamente indipendente dalle librerie Qt, per favorirne il riutilizzo e la modularizzazione.

Tuttavia si tenga presente che, per motivi di tempo, non sono state implementate tutte le possibili ottimizzazioni e in particolare l'*exception-safety* non è assicurata. A ogni modo è stato effettuato estensivo debugging e il software ha un'alta affidabilità.

Di seguito vengono presentate scelte progettuali e funzionamento di ognuna delle tre applicazioni, nell'ordine in cui le funzionalità da esse implementate sono state affrontate nella stesura di questa tesi.

4.1 BMDetect

Questa applicazione realizza i meccanismi per il rilevamento di contorni visti nelle sez. 2.3.1 e 2.3.2.

Il programma è dotato di una GUI realizzata con il framework Qt e fa uso di altre librerie per permettere la lettura e la scrittura di file in diversi formati. In particolare, è possibile estrarre i contorni da immagini statiche (in diversi formati, ad esempio BMP o JPG) o da sequenze video (ad esempio in formato MPG o da un container AVI).

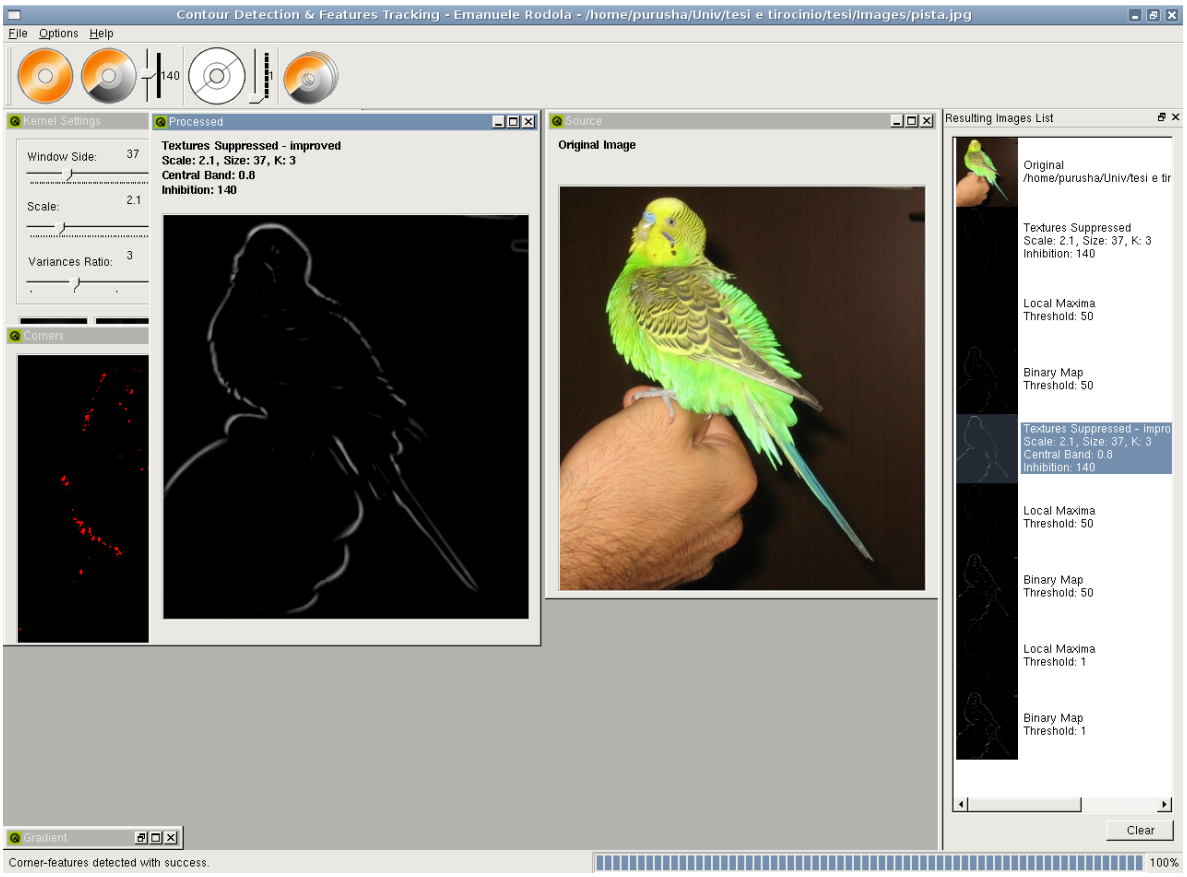


Fig. 80: BMDetect

In fig. 81 viene mostrato un semplice diagramma delle associazioni tra i componenti principali dell'applicazione.

Le librerie Qt permettono di realizzare l'interfaccia grafica con poche righe di codice (basandosi su un design pattern chiamato *Observer* [19]) e assicurano la portabilità dell'applicazione su più sistemi operativi; inoltre, forniscono un'implementazione veloce e sicura del multi-threading, di cui si è fatto uso in maniera estensiva per suddividere i compiti di elaborazione in thread di lavoro che non appesantissero l'interfaccia grafica - eventualmente congelandola. Si noti che il codice che si occupa dell'interazione tra le funzionalità di elaborazione e l'interfaccia grafica, essendo inserito in una piattaforma Qt, non rispecchia una sintassi C++ standard: di fatti, il framework utilizzato prevede una fase di traduzione del codice per il tramite di un preprocessore chiamato *moc* che si occupa di portare quanto scritto nel dialetto Qt verso il C++ standard.

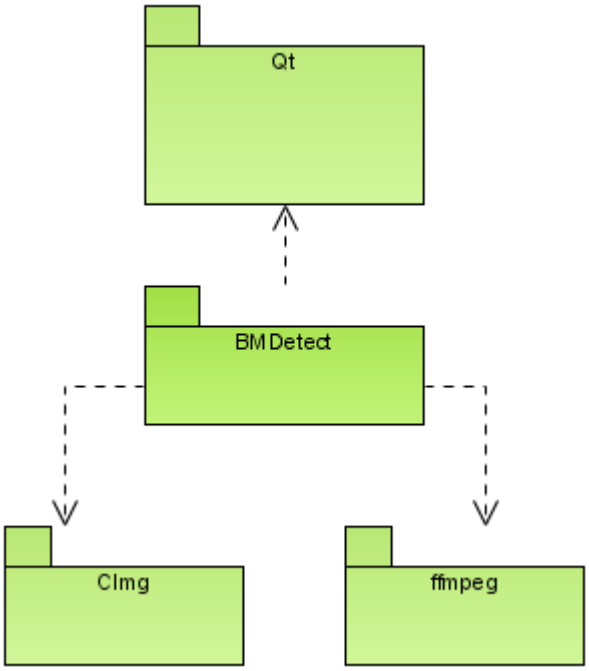


Fig. 81: Diagramma delle associazioni tra i componenti principali di BMDetect

L'applicazione è progettata in maniera tale che sia possibile fornire diverse implementazioni per la GUI, così come diverse GUI per le funzionalità di elaborazione. Vale a dire, le due componenti sono altamente disaccoppiate.

In particolare, le funzionalità di *core* sono realizzate principalmente dalla classe ScalableImg. Questa classe dipende dalla libreria CImg (scritta in C++), che consente di svolgere task tipici nell'elaborazione delle immagini, compiti la cui realizzazione non è di interesse nell'ambito di questa tesi. Operazioni come convoluzione, dilatazione morfologica e ridimensionamento sono lasciate alla libreria, che ne offre già un'implementazione abbastanza rapida, mentre meccanismi chiave come l'estrazione dei componenti connessi (sez. 2.3.1.3) o la creazione delle maschere di convoluzione o del termine di inibizione (sez. 2.3.1.2) sono implementati da zero.

Da ultimo, per il caricamento e il salvataggio di sequenze video, l'applicazione si affida alla libreria C ffmpeg, che slega il programmatore dalle problematiche inerenti la gestione dei file video su diversi sistemi operativi, selezionando coerentemente (in base alle caratteristiche dei frame, come la loro dimensione e il bitrate prescelto) i codec disponibili a run-time. Per una discussione più dettagliata delle funzionalità implementate per l'elaborazione video, si rimanda al capitolo 4.3.

Si noti che sia CImg che ffmpeg sono perfettamente portabili; questo fatto, congiunto all'utilizzo esclusivo di funzioni standard (e della STL), rende BMDetect un'applicazione multi-

piattaforma (a patto di una ricompilazione sul sistema target).

Passiamo adesso a un'analisi più dettagliata dell'applicazione, focalizzandoci sulle sue funzionalità principali e dando riferimenti ai capitoli precedenti laddove necessario. Si tenga presente che questo capitolo (così come quelli successivi, relativi alle rimanenti due applicazioni) *non* va inteso come un manuale di utilizzo, per il quale invece si rimanda a [30].

Anzitutto mostriamo il diagramma dei casi d'uso, che descrive in maniera intuitiva e dal punto di vista dell'utente le funzioni e i servizi offerti dal sistema; ci restringiamo all'elaborazione di immagini statiche, lasciando, come detto, la discussione sulla parte video al capitolo 4.3:

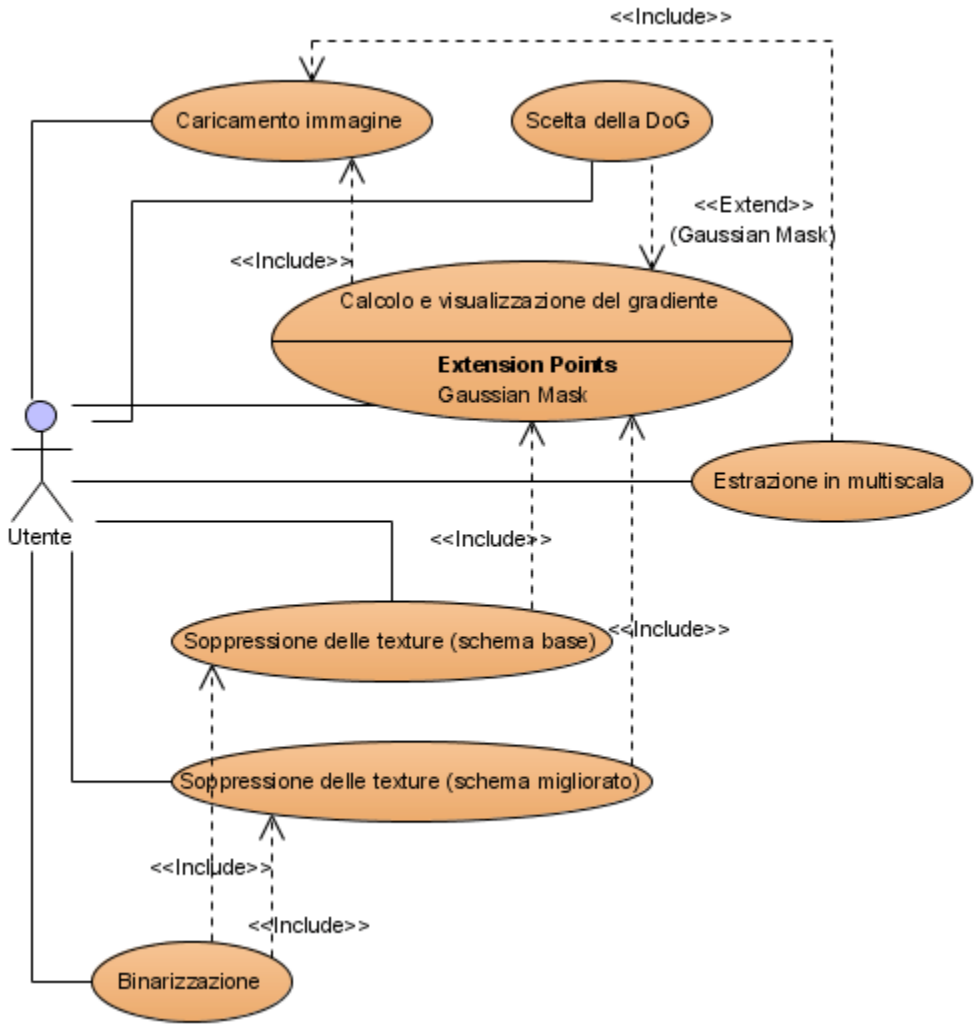


Fig. 82: Diagramma dei casi d'uso di BMDetect

Il programma consiste essenzialmente in una serie di classi di interfacciamento con l'utente, una classe che implementa il thread di lavoro e un'unica classe che implementa le funzionalità di

interesse. Le classi il cui nome comincia per “Dlg” fanno riferimento all'interfaccia grafica e in quanto tali non sono in questo momento meritevoli di attenzione.

Di seguito vengono elencate le responsabilità principali della classe *ScalableImg*, vero cuore di questo programma:

- Rappresenta una singola immagine statica e ne predica le *proprietà principali*, come la scala (risoluzione) corrente e le dimensioni (in pixel).
- Incapsula diverse immagini di tipo CImg<> per *mantenere informazioni sui vari stati assunti* dall'immagine di partenza (es. gradiente e massimi locali), nonché le maschere necessarie per la realizzazione dei meccanismi di soppressione delle texture e di binarizzazione (sez. 2.3.1).
- L'immagine rappresentata può essere fatta variare lungo una *scala diadica di risoluzioni* (metodi nextScale(), getLowerScaleImage()).
- L'immagine rappresentata può essere trasformata con l'algoritmo di *Haar* [7] (metodi getTransformedImage(), getHorizontalDetail(), getVerticalDetail(), getDiagonalDetail()).
- È possibile *ottenere il gradiente* (eventualmente ed adeguatamente sfocato) dell'immagine, e una sua versione approssimata per tempi di calcolo minori (metodi calcGradientMagnitude(), calcGradientOrientation()).
- È possibile *ottenere i massimi locali del gradiente* (metodo suppressNonMaxima()).
- È possibile *sopprimere le texture* nell'immagine adottando i due schemi di soppressione delle sezioni 2.3.1.2.1 e 2.3.1.2.2 (metodi suppressTextures() e suppressTexturesEnh()).
- È possibile *rilevare gli angoli* di Rosten-Drummond (sez. 3.2.1, metodo detectCorners()).
- È possibile rilevare i componenti connessi e *binarizzare il gradiente* calcolando il peso globale dei contorni come da sez. 2.3.1.3 (metodi getConnectedComponents() e binarize()).

Percorriamo adesso i casi d'uso visti in precedenza per dare una visione più dettagliata (al livello giusto) di come i vari oggetti collaborano tra di loro per realizzare il sistema di estrazione dei contorni, dal caricamento dell'immagine fino alla costruzione della mappa binaria dei contorni. I flussi mostrati di seguito hanno come precondizione il caricamento dell'immagine.

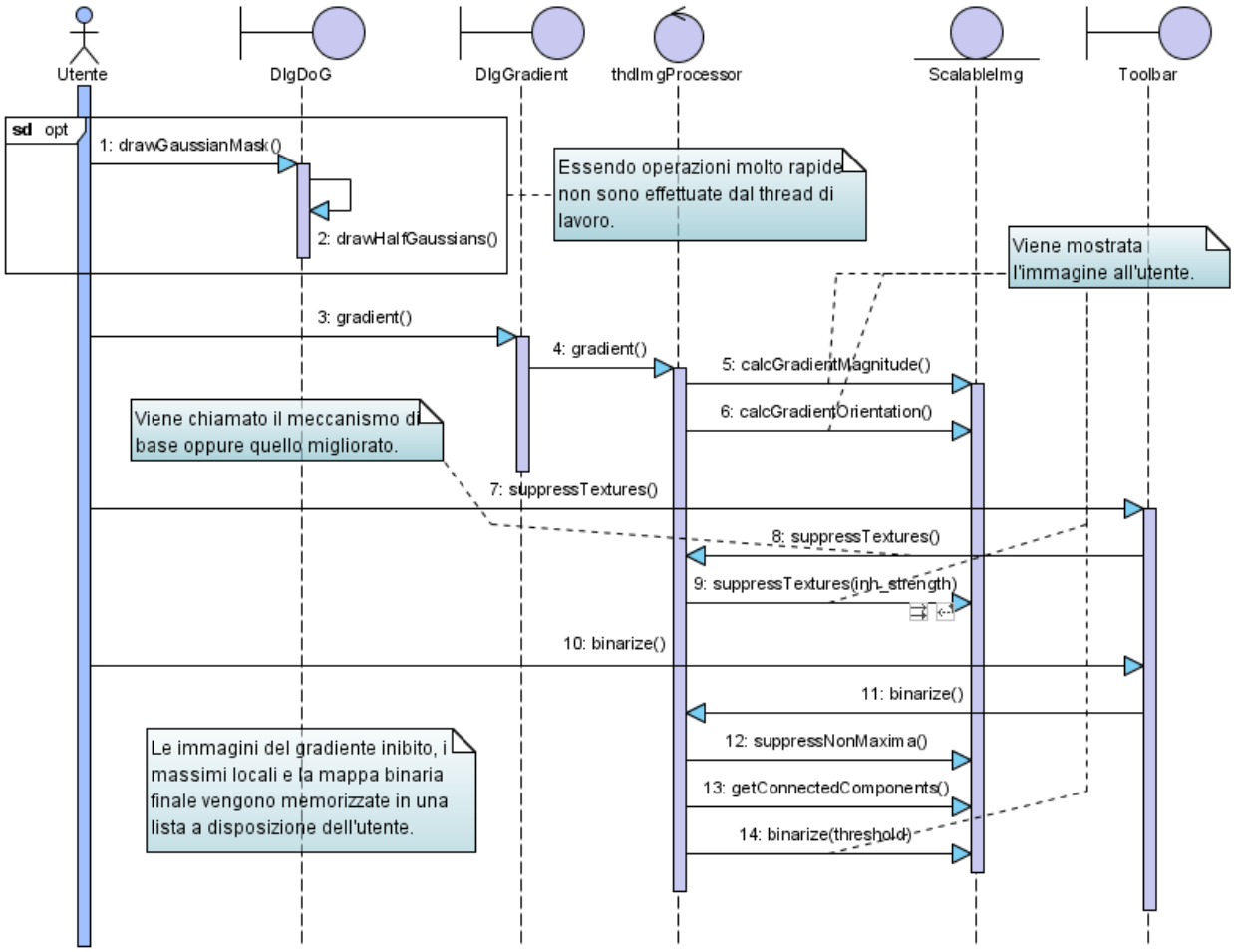


Fig. 83: Flusso base di estrazione dei contorni di BMDetect

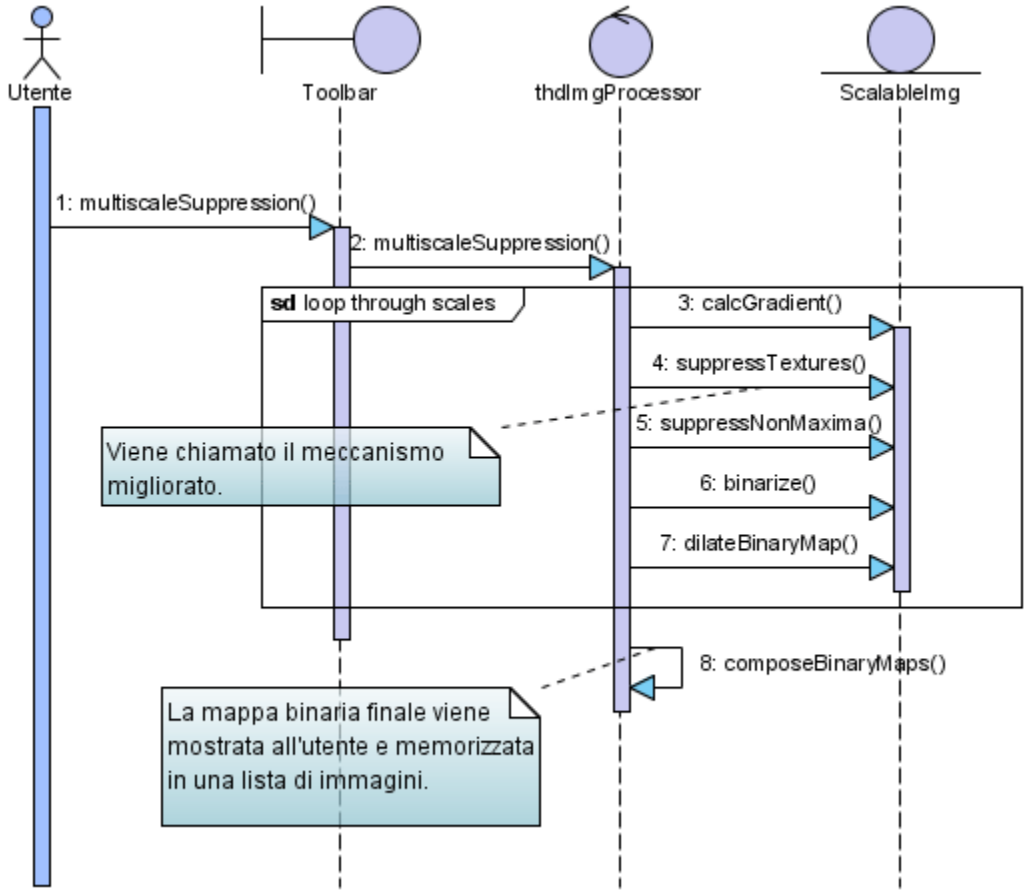


Fig. 84: Flusso relativo alla funzionalità di estrazione in multirisoluzione

4.2 FAST Trainer

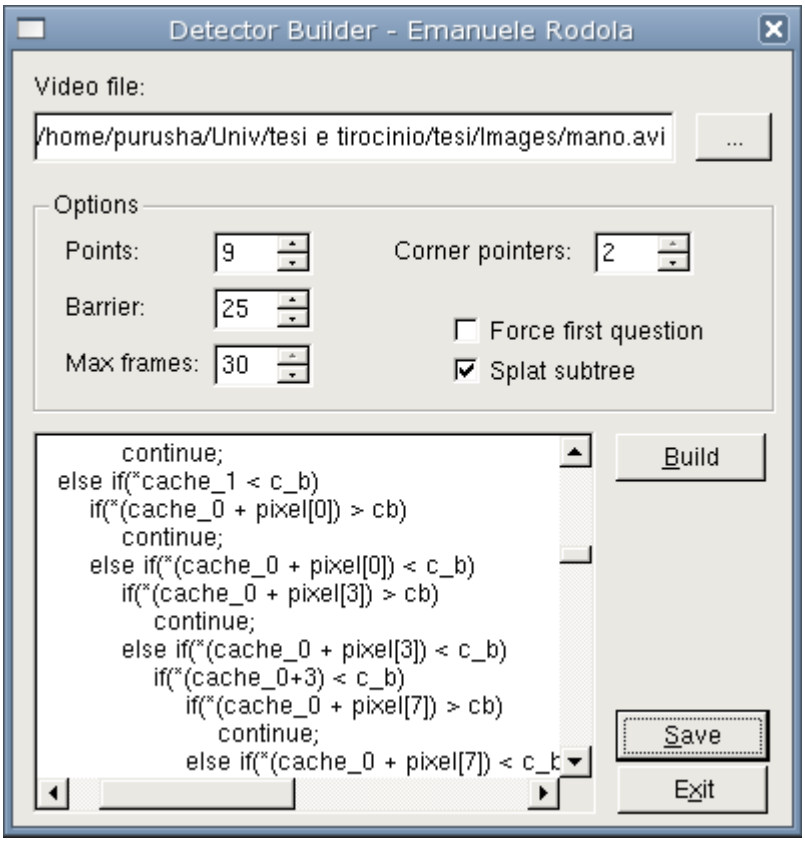


Fig. 85: FAST Trainer

FAST Trainer è l'applicativo che permette di realizzare il meccanismo di apprendimento adottato da Rosten e Drummond per il rilevamento rapido degli angoli (sez. 3.2.1). Il software permette di caricare un file video e utilizzarne i primi n fotogrammi per la fase di training.

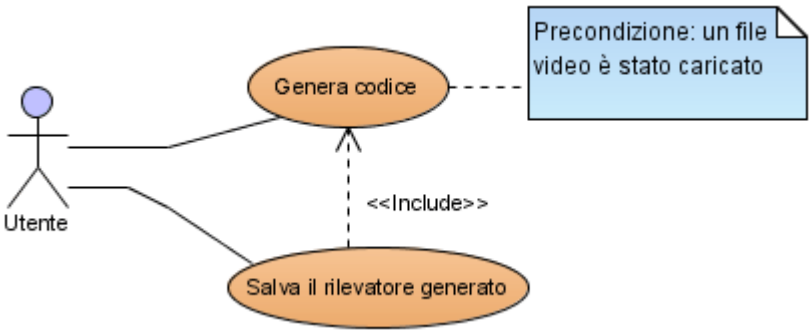


Fig. 86: Diagramma dei casi d'uso di FAST Trainer

Il codice che realizza il segment test, la sua versione veloce, e la funzione di score per la ritenzione dei massimi è scritto in C ed è fornito direttamente dagli autori. In questa sede, tale codice è stato diversamente ottimizzato a livello di sintassi per velocizzare alcune operazioni (senza tuttavia incidere sostanzialmente sulle prestazioni finali) e per adattarsi a un ambiente C++: le funzionalità sono state incapsulate in alcune classi, mostrate nel diagramma seguente:

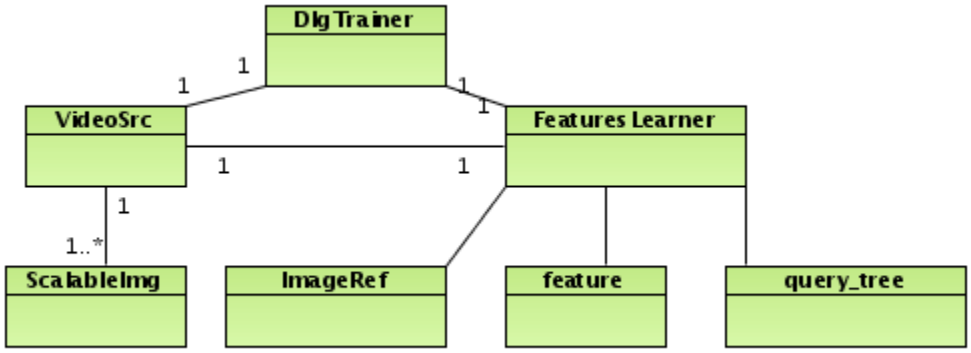


Fig. 87: Diagramma delle classi di FAST Trainer

La classe *VideoSrc* si occupa del caricamento e del salvataggio di file video e verrà discussa in dettaglio nel capitolo dedicato all'applicazione BodyTracker. La classe *ImageRef* fornisce una rappresentazione appropriata per coordinate bidimensionali all'interno di un'immagine, e la classe *print_code* realizza un funtore per la generazione del codice in seguito alla fase di apprendimento.

FeaturesLearner è la classe che implementa il segment test, la funzione di score e la logica di controllo per l'apprendimento. I metodi principali esposti dall'interfaccia sono i seguenti:

loadVideo()	Imposta un riferimento a un oggetto di tipo *VideoSrc*.
generateCode()	Genera il codice C per il rilevamento degli angoli secondo i parametri passati come argomento (numero di pixel contigui e valore di soglia per il segment test). Il codice così generato viene scritto nello stream passato per riferimento come primo argomento.

Di seguito viene mostrato il flusso di base (relativo all'utilizzo tipico delle classi) dell'applicazione.

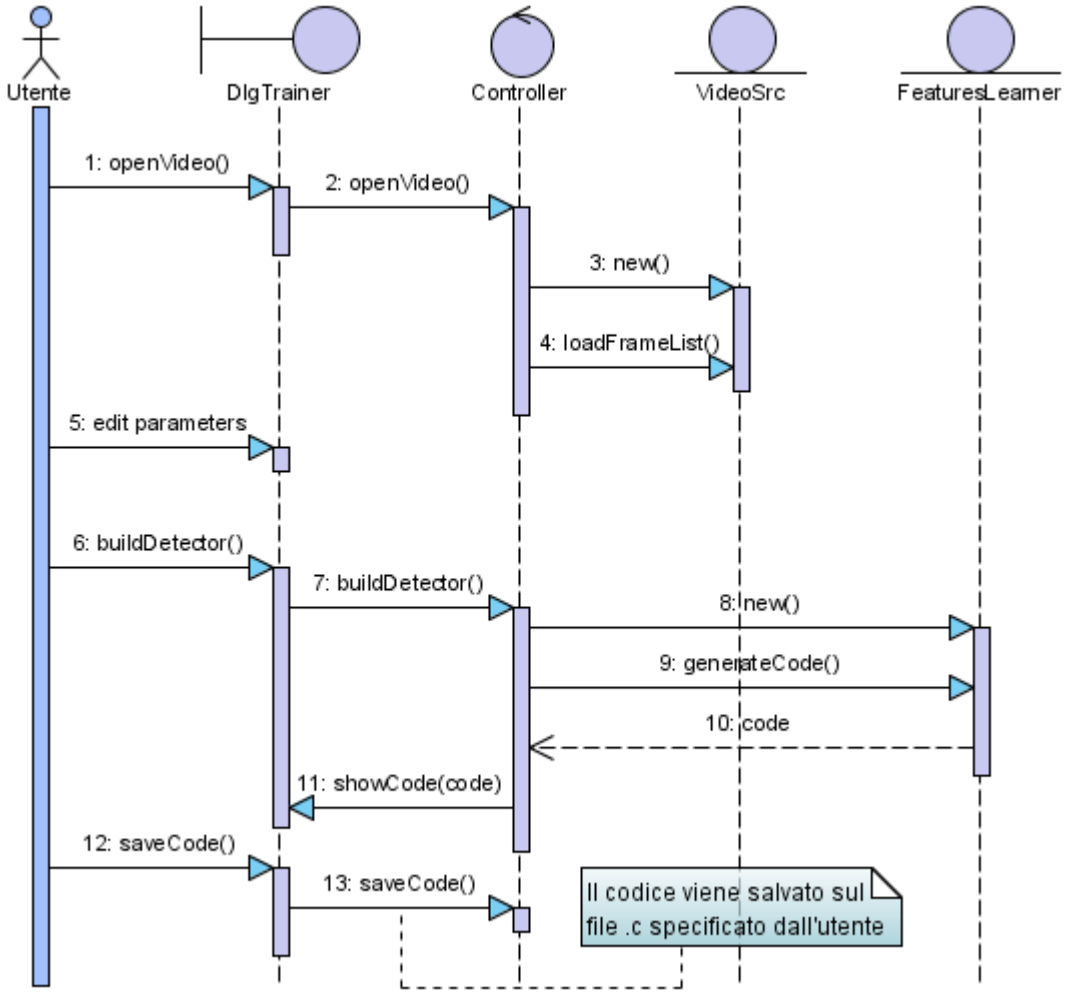


Fig. 88: Flusso base di utilizzo di FAST Trainer

4.3 BodyTracker

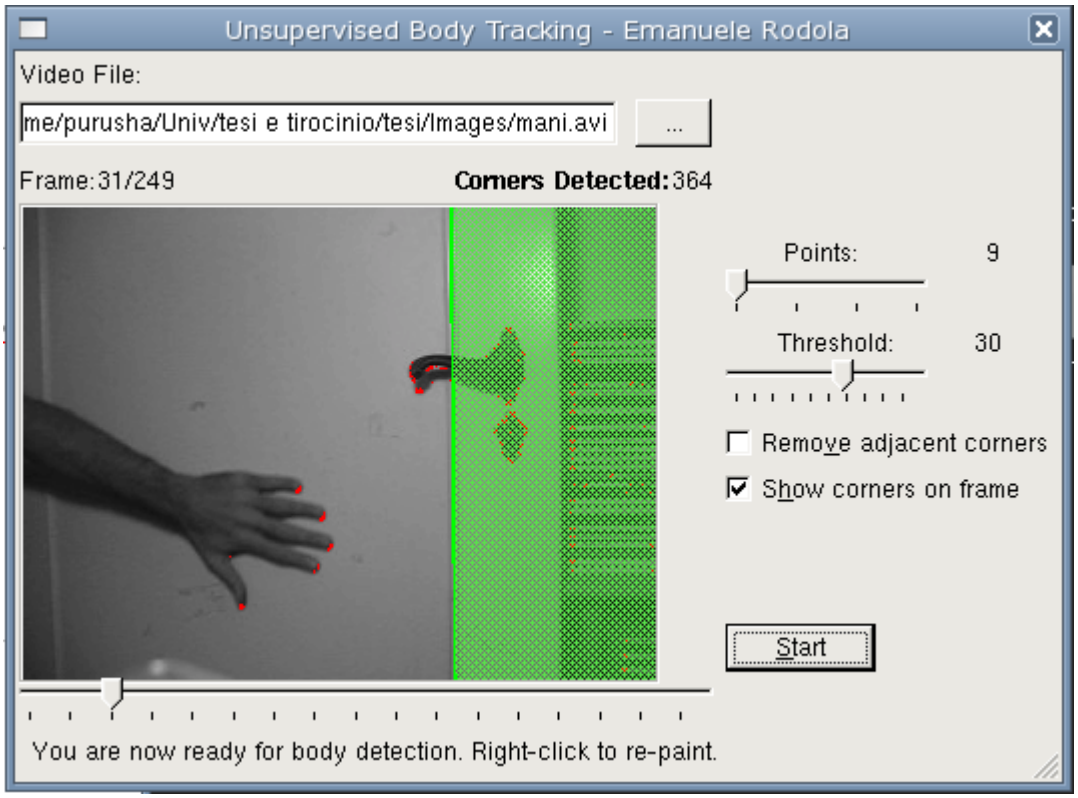


Fig. 89: Body Tracker

BodyTracker è l'applicazione che si occupa del tracking di entità di vario tipo in sequenze video, secondo il meccanismo discusso in sez. 3.3.3. Come per i casi precedenti, il programma è inserito in un framework Qt e utilizza le librerie CImg e ffmpeg per i task triviali di elaborazione di immagini e video (ffmpeg viene unicamente utilizzata per il caricamento e il salvataggio di file video).

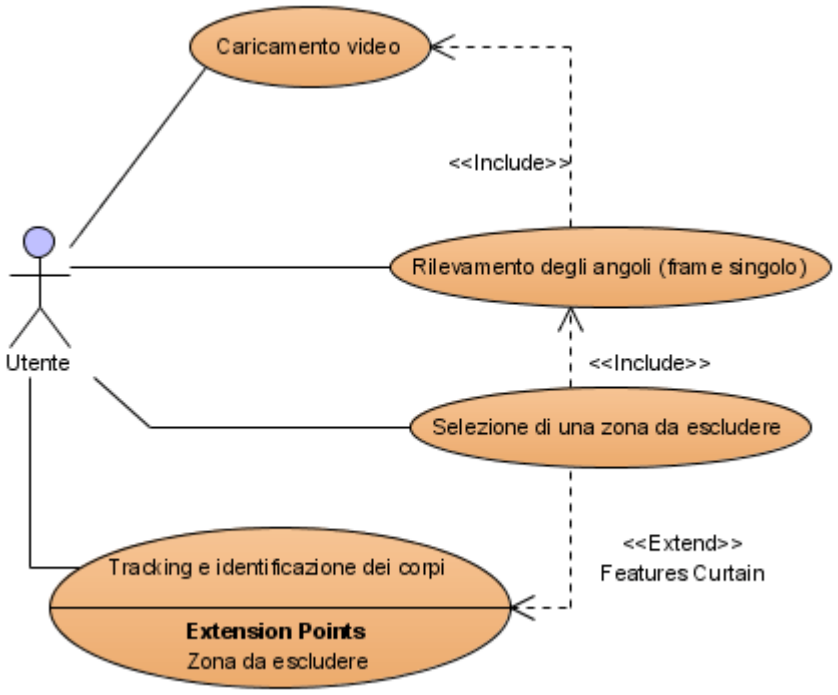


Fig. 90: Diagramma dei casi d'uso di BodyTracker

Il programma fa uso della stessa classe *ScalableImg* utilizzata in BMDetect (sez. 4.1), e della classe *VideoSrc* accennata in FAST Trainer (sez. 4.2). Vediamo in particolare le responsabilità principali delle due classi in questo caso:

ScalableImg
• È possibile rilevare gli angoli di Rosten-Drummond secondo i parametri specificati dall'utilizzatore • È possibile ottenere un'immagine degli angoli su sfondo nero oppure ottenere un'immagine degli angoli sovrimpressi al frame d'origine
VideoSrc
• Permette di caricare il file video prescelto in due liste concatenate, contenenti i frame in ordine temporale crescente: una lista in scala di grigi e una lista in formato RGB24 • Permette di salvare una qualsiasi lista di frame su un file video, a patto che i frame abbiano le medesime caratteristiche (dimensione, risoluzione, formato); permette inoltre di sovraimporre su ogni frame del testo • Permette di rilevare gli angoli di Rosten-Drummond per tutti i frame • Permette di disegnare un'immagine che contenga una traccia sfumata di tutti gli angoli

rilevati nella sequenza video

- *Realizza i meccanismi di tracking* esposti nel capitolo 3.3, e permette di visualizzare i corpi tracciati in un formato pratico alla vista e utilizzando colori diversi per ogni corpo

Oltre a queste due classi principali ne sono state realizzate di altre. Di seguito un succinto diagramma delle classi per BodyTracker, disegnato per mettere in luce le associazioni tra le classi piuttosto che la loro interfaccia:

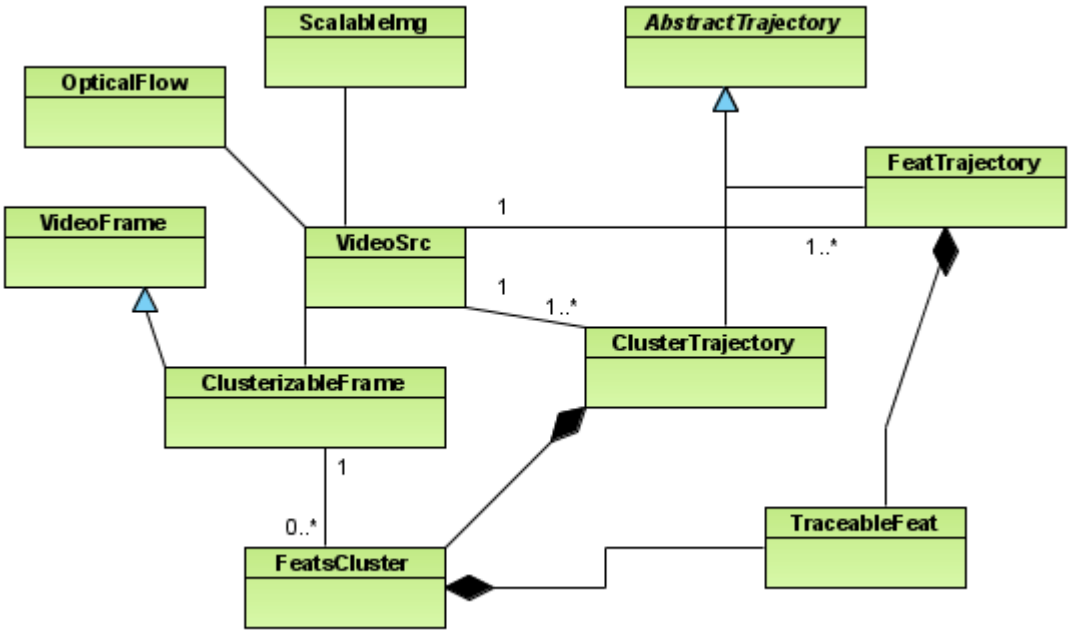


Fig. 91: Diagramma delle classi di BodyTracker

Vediamo le responsabilità delle classi principali:

OpticalFlow
• Calcola il flusso ottico tra due immagini, secondo il metodo di Lucas-Kanade (sez. 3.1)
TraceableFeat
• Rappresenta una feature che può essere tracciata lungo una sequenza di frame; tale feature corrisponde a una coppia di coordinate (x, y) ed ha associato il numero del frame cui fa riferimento e un identificatore univoco del cluster di punti cui appartiene

- Una feature "vuota" (*blank*) ha le coordinate impostate ai valori speciali BLANK_POS

FeatsCluster

- Definisce un cluster di feature, cioè un insieme (nella fattispecie, un vettore) di feature che condividono lo stesso identificatore; ha un numero identificativo univoco ed è associato a un numero di frame
- Un cluster è rappresentato dal suo centroide, un punto le cui coordinate sono calcolate come la media delle coordinate dei punti presenti nel cluster
- Un cluster ha associato un colore, che viene utilizzato per disegnare il corpo che rappresenta
- Un cluster viene chiamato *outlier* se contiene un solo punto; un cluster è vuoto (*blank*) se non contiene punti
- È possibile *fondere* un cluster con un altro, a patto che facciano riferimento allo stesso frame; il colore e l'identificativo mantenuti sono quelli di uno dei due cluster (quale non ha importanza)
- Viene rappresentato graficamente in questo modo: su sfondo nero viene disegnato, utilizzando il colore associato al cluster, il suo centroide; dopodiché viene disegnato ogni punto del cluster e per ognuno di essi viene tracciato un segmento verso il centroide; infine, viene tracciata una circonferenza centrata nel centroide e il cui raggio è la distanza media di ogni punto dal centroide

VideoFrame

- Rappresenta un frame video, cioè un'immagine cui è associato un numero di frame

ClusterizableFrame

- Rappresenta un frame video i cui pixel possono essere raggruppati in cluster, secondo un certo algoritmo di clusterizzazione
- È possibile cercare, ottenere e gestire in maniera separata ogni singolo cluster rilevato
- È possibile cercare il cluster che contenga una data feature
- È possibile cercare il cluster che contenga tutti i punti disegnati su una data immagine

FeatTrajectory

- Rappresenta una traiettoria di feature, è cioè una rappresentazione del moto di una feature nel tempo
- Una feature può essere cercata o aggiunta (ma non eliminata) alla lista che di fatto rappresenta la traiettoria
- La traiettoria può essere "estesa" su un intervallo $[0,n]$, quando questa copre un suo sotto-intervallo $[a,b]$, inserendo appositamente delle feature *blank*
- La traiettoria ha due diverse definizioni di *dimensione*:

1. il numero di frame coperti dalla traiettoria (*size*)
2. il numero *effettivo* di frame coperti dalla traiettoria, cioè senza contare i *blank* (*actualSize*)

ClusterTrajectory

- Rappresenta una traiettoria di cluster, una rappresentazione del moto di un cluster di punti nel tempo
- Come nel caso di FeatTrajectory, la traiettoria può essere estesa su un intervallo, e quindi anche in questo caso ci sono due diverse definizioni di *dimensione*
- La traiettoria può essere "fusa" con un altra che abbia la sua stessa *size*, operazione che consiste nell'accorpamento frame per frame dei cluster presenti nelle due liste; la traiettoria da incorporare può essere un'istanza di ClusterTrajectory o di FeatTrajectory
- È possibile determinare se la traiettoria rappresenti un cluster fermo, oppure un cluster che compare/scompare all'improvviso e lontano dai bordi della scena video, operazioni effettuate tenendo conto di dati limiti di tolleranza

Per una migliore comprensione del codice, di seguito vengono mostrati i sequence diagram relativi al flusso base di riconoscimento di corpi secondo la tecnica di coherent motion detection.

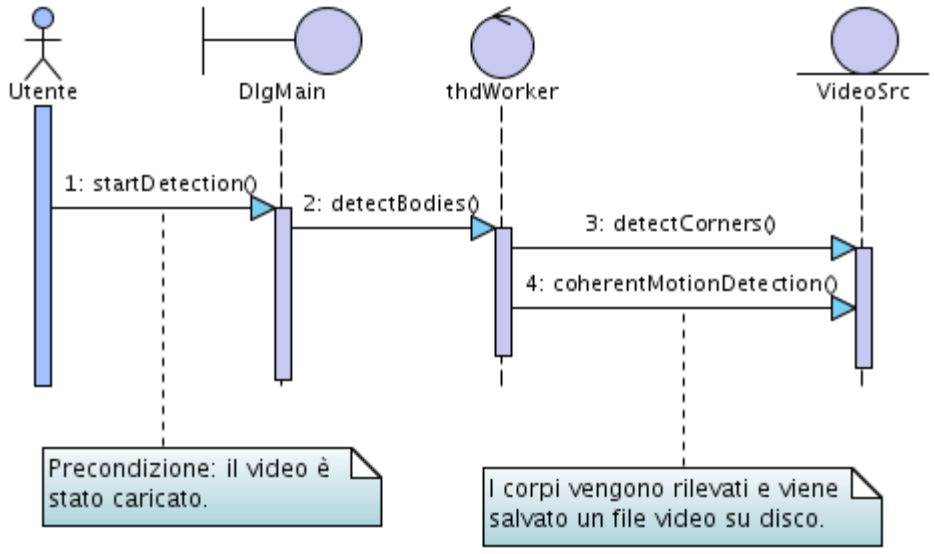


Fig. 92: Flusso base di utilizzo per il rilevatore di coerenza del moto

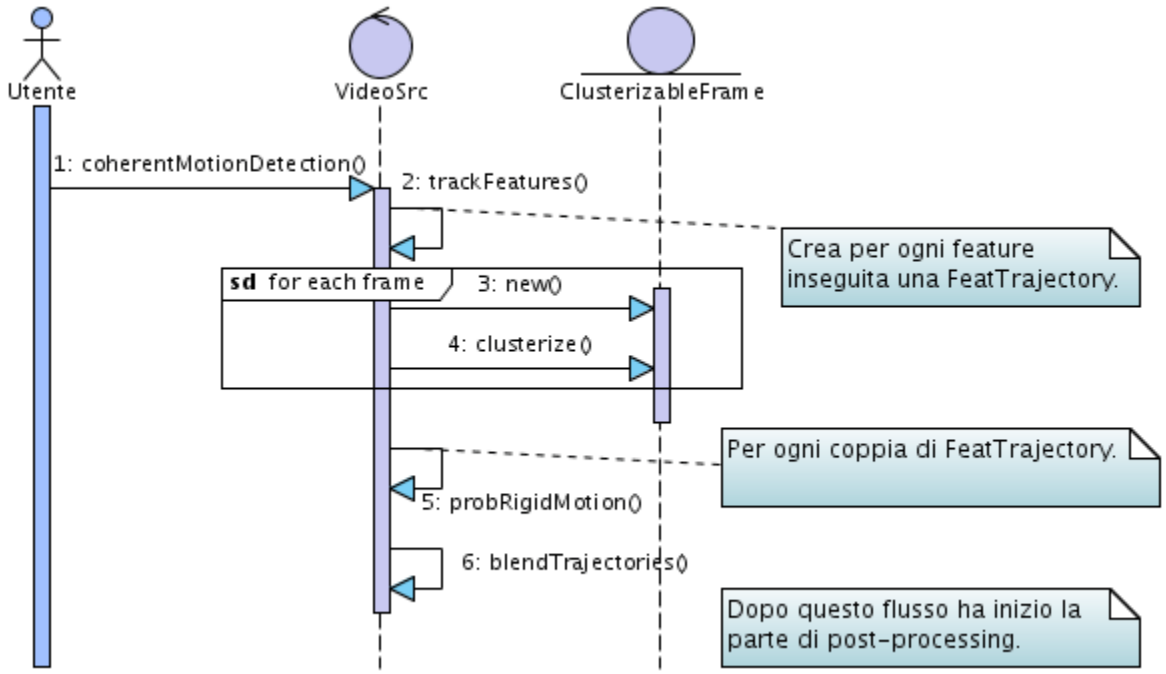


Fig. 93: Rilevamento di coerenza del moto

Capitolo 5: Risultati sperimentali

In questo capitolo vengono presentati e discussi alcuni risultati sperimentali. Vengono valutate le prestazioni delle applicazioni realizzate, in merito sia alla loro efficacia che alla loro efficienza, e vengono fatti paragoni con altri rilevatori e sistemi di riconoscimento.

Il software è stato sviluppato e sperimentato in ambiente operativo Linux Ubuntu 7.04, Kernel 2.6.20, GCC 4.1.2, ld 2.17.50, CImg 1.2.4, ffmpeg CVS-2007-12-12, Qt 4.3.0, su macchina Intel(R) Pentium(R) 4 CPU 2.80GHz , 1GB RAM DDR-400.

5.1 BMDetect

BMDetect è l'applicazione dedicata all'estrazione di contorni da immagini. I risultati del rilevamento dipendono da alcuni parametri impostati dall'utente, e il rilevamento stesso può essere effettuato seguendo tre schemi distinti (di cui si è discusso ampiamente nel capitolo 2.3). Le tabelle di seguito riassumono detti parametri e schemi di rilevamento:

Parametro	***Descrizione***
σ	Scala (dispersione) della maschera di convoluzione (schemi 1 e 2)
k	Rapporto tra varianze della DoG (schemi 1 e 2)
ws	Dimensione della maschera in pixel per lato (schemi 1 e 2)
α	Intensità d'inibizione (tutti)
B	Ampiezza della banda centrale nella DoG (schemi 2 e 3)
th	Soglia sul peso globale del contorno (tutti)
m	Numero di livelli (schema 3)

Schema	***Descrizione***
1	Rilevamento con schema base (sez. 2.3.1.2.1)
2	Rilevamento con schema migliorato (sez. 2.3.1.2.2)
3	Rilevamento in multirisoluzione (sez. 2.3.2)

Dipendentemente dai parametri selezionati dall'utente, si hanno ovviamente risultati diversi. Tuttavia, mentre alcuni di questi parametri influenzano in maniera sostanziale la qualità dei contorni estratti, altri sono stati osservati non incidere in maniera altrettanto decisiva.

I parametri σ , k e ws misurano la dispersione e il supporto del kernel gaussiano, nonché la sua dimensione in pixel (vedere a titolo di esempio la fig. 52). A scale alte si ottiene un modulo del gradiente piuttosto sfocato, e dunque una migliore dispersione del rumore. Inoltre, più la maschera è grande, più onerose diventano le convoluzioni con l'immagine di partenza, ragion per cui questi parametri influenzano pesantemente non solo l'efficacia degli algoritmi di estrazione, bensì anche la loro efficienza. Di seguito alcuni risultati al variare della maschera di convoluzione (il parametro *k* è stato osservato non incidere sostanzialmente sui risultati, quindi di qui in poi è fissato al valore $k=3$):

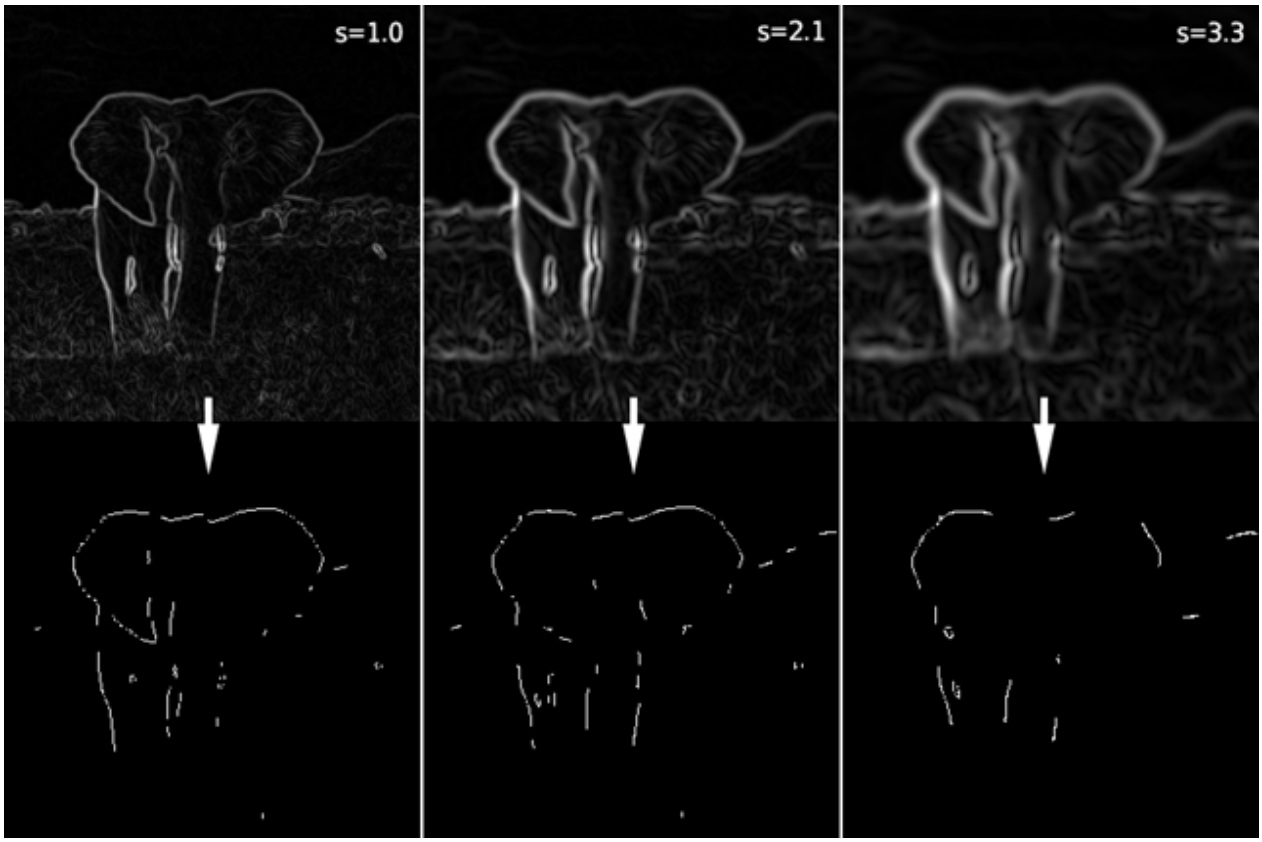


Fig. 94: Diverse mappe binarie ottenute con lo schema di base al variare della maschera di convoluzione. Per ogni mappa è indicato il valore del parametro σ . Gli altri parametri sono, per tutti e tre i casi, $\alpha=140$ e $th=50$.

Nello schema di soppressione migliorato, per ovviare al problema noto come auto-inibizione, viene esclusa una banda orientata dalla maschera DoG (sez. 2.3.1.2.2). L'ampiezza di tale banda corrisponde alla distanza tra i due semi-anelli e per semplicità viene fatta variare nell'intervallo [0.1,1.0].

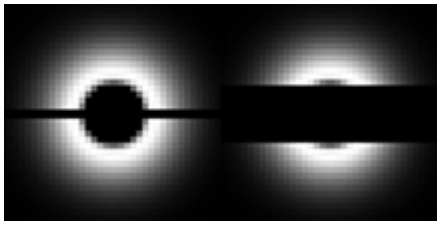


Fig. 95: Banda ampia 0.1 (sulla sinistra) e 1.0 (sulla destra)

L'ampiezza della banda si desidera grande per bordi grossolani, e piccola per bordi sottili (per i diversi tipi di bordo, vedere l'introduzione al capitolo 2); in questo modo possiamo essere sicuri di annullare quasi del tutto l'auto-inibizione. Tuttavia, sorgono problemi nelle regioni spigolose dell'immagine e, a ogni modo, capire lo spessore di un bordo in questa fase del processo di estrazione non è un compito banale. Per questo motivo, una volta scelta la distanza tra i semi-anelli, questa resta fissa per l'intero processo. Quanto detto ci suggerisce che una scelta adeguata per questo parametro dipende dalla natura degli oggetti da riconoscere e dalla qualità dell'immagine analizzata. Di seguito vengono mostrati degli esempi di estrazione dei contorni secondo lo schema migliorato, al variare di *B* (figg. 96 e 97).

Dagli esempi si osserva che al di sotto di un certo valore di *B* si hanno pressoché gli stessi risultati , e al di sopra dello stesso accade la medesima cosa. Ciò significa che per bande centrali "strette" si ottiene una versione solo leggermente migliorata del rilevatore di base (che soffre in buona misura dell'auto-inibizione), mentre oltre una certa soglia, che dipende dalla natura dei contorni degli oggetti in analisi così come dalla misura in cui il modulo del gradiente è stato preventivamente sfumato, si ottengono i risultati sperati. Inoltre, oltre questa soglia il valore esatto di *B* non ha importanza, portando in effetti a variazioni delle prestazioni davvero minime [1]. Dunque, detto parametro può essere fatto variare tra soli due valori, e messo in relazione con il parametro di scala σ . In conclusione, sebbene il rilevatore migliorato introduca un nuovo parametro, questo può essere lasciato sostanzialmente invariato per il dominio applicativo corrente, e anzi si presta a una qualche forma di auto-configurazione (tuttavia non adottata nel software realizzato).

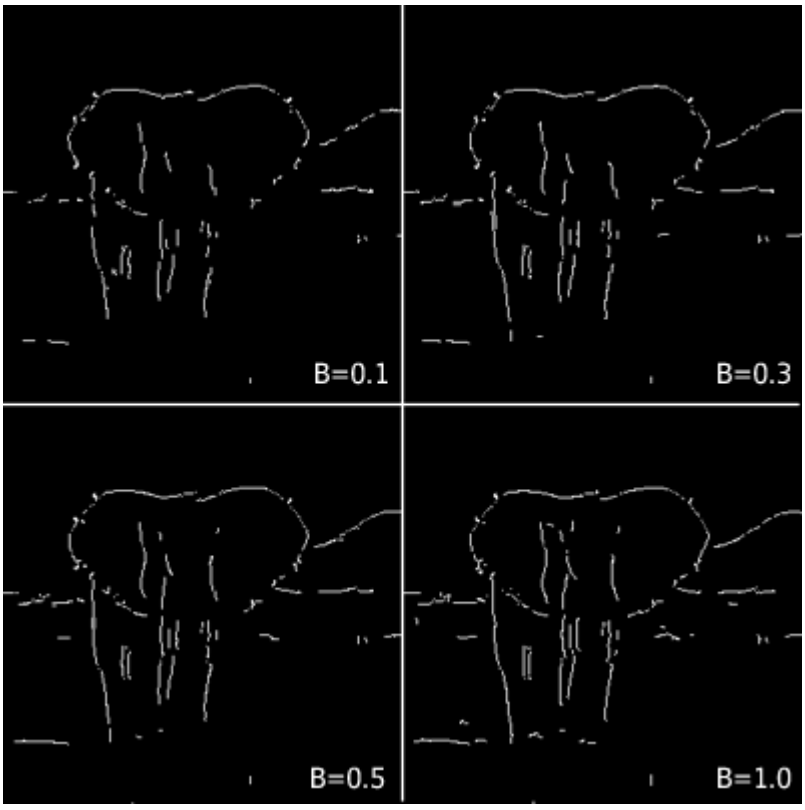


Fig. 96: Diverse mappe binarie ottenute con lo schema migliorato, al variare della banda centrale

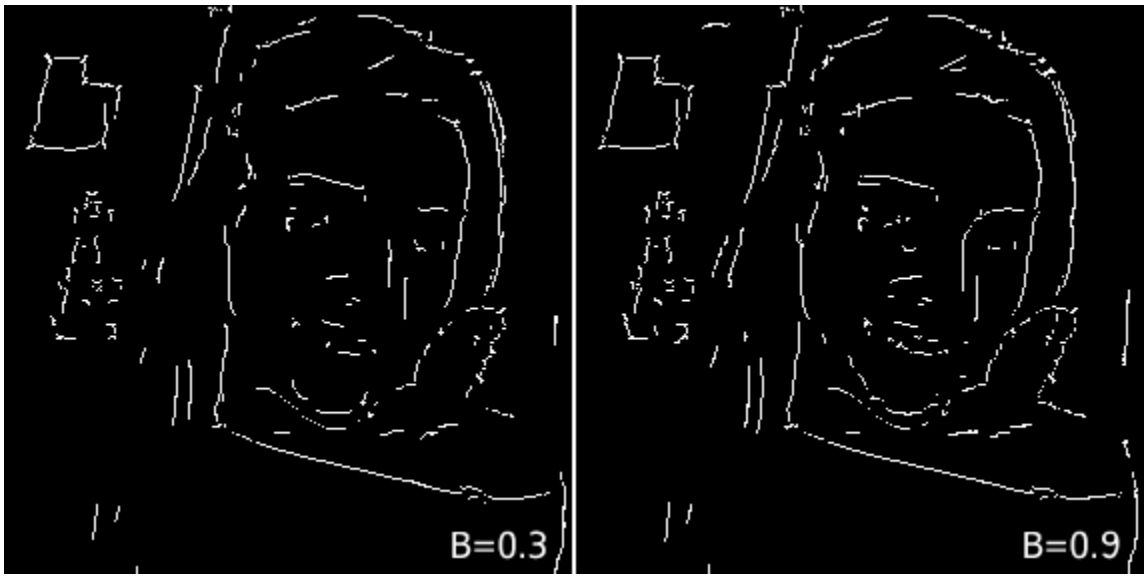


Fig. 97: Mappe binarie per l'immagine di fig. 49 al variare della banda centrale

L'intensità di inibizione e il valore di soglia per la fase di binarizzazione sono i due parametri che, indubbiamente, influiscono in misura maggiore sull'intero sistema di rilevamento. Il primo controlla la misura in cui il termine di inibizione ha effetto sul modulo del gradiente: per valori alti di α il termine di inibizione ha un peso maggiore, il che comporta un'alta soppressione dei dettagli ma anche eventuali casi di inibizione dei contorni effettivi. Per questo motivo, specie quando si adotta lo schema di base, è preferibile mantenere valori contenuti. Con lo schema migliorato, invece, è possibile alzare il valore del parametro, dal momento che l'auto-inibizione non ha luogo: aumentando l'intensità di inibizione si ottiene una migliore soppressione delle texture mantenendo contorni forti.

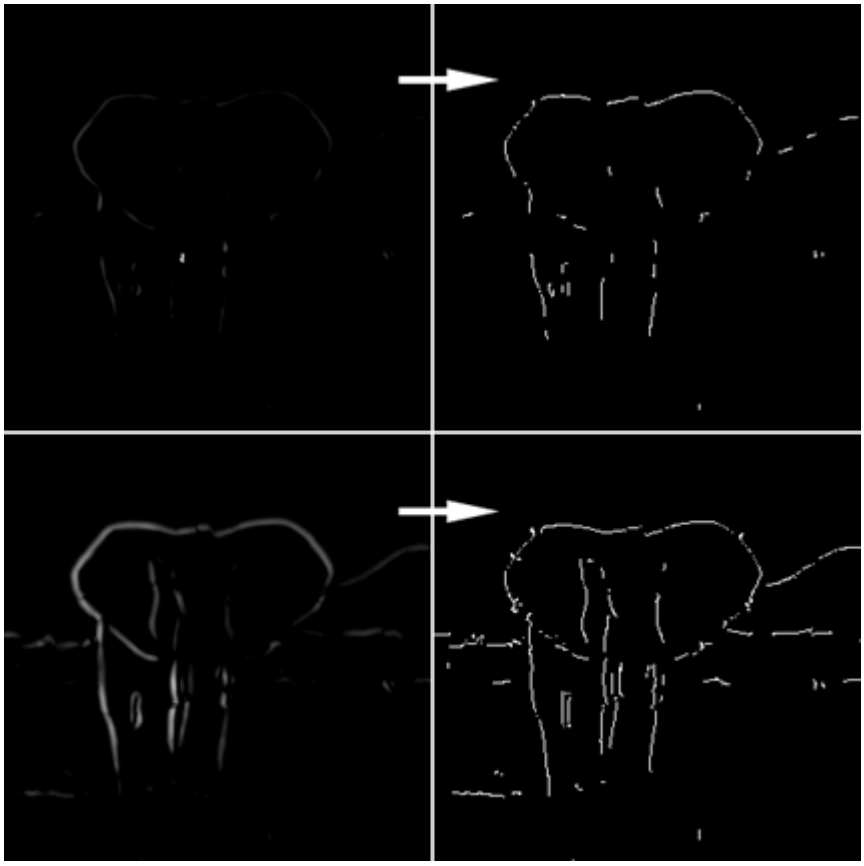


Fig. 98: Schema base (riga superiore) confrontato con quello migliorato (riga inferiore). I parametri utilizzati sono i medesimi in entrambi i casi; nel caso migliorato si è utilizzato B=0.8.

In fig. 98 si ha un esempio di applicazione dei due schemi di soppressione utilizzando i medesimi parametri. Come si vede, molti dei contorni vengono sostanzialmente inibiti nel primo caso. Il risultato del secondo schema può essere ulteriormente raffinato aumentando la forza

d'inibizione α .

Il parametro th controlla il valore di soglia da utilizzare in fase di binarizzazione. Per valori alti di th vengono eliminati molti dettagli, ma anche parte dei contorni effettivi. Per valori bassi, d'altra parte, i contorni vengono mantenuti ma è possibile che molti dettagli non vengano eliminati; inoltre, c'è il rischio che nella mappa binaria finale compaiano componenti dovute al rumore. Quindi, dipendentemente dal successo della fase di inibizione, e cioè dipendentemente dal valore di α , il parametro th deve essere scelto in maniera opportuna, ed è anzi pensabile automatizzare la scelta optando per una soluzione adattativa. In particolare, i due parametri sono in relazione inversamente proporzionale tra di loro. A titolo di esempio si consideri fig. 59.

Vediamo infine i risultati ottenuti con il rilevamento a più scale. La tecnica trae beneficio dal fatto che i contorni degli oggetti possono essere discriminati dai bordi dovuti ai dettagli perché, oltre una certa scala, i primi sono presenti a tutte le risoluzioni, mentre gli ultimi lo sono solamente alle scale più fini. Il vantaggio principale dell'approccio risiede nel fatto che è possibile ottenere contorni solidi e contigui mantenendo intatte le giunzioni, ed eliminando gran parte dei dettagli. Inoltre, la dilatazione morfologica permette di ovviare al problema della traslazione dei contorni (vedere le figure di sez. 2.3.2).

Un problema piuttosto evidente in questo metodo è la sua sensibilità al rumore in corrispondenza di contorni disconnessi (fig. 99). Il problema si presenta in quanto il poco rumore pur presente ad alta risoluzione viene mantenuto per via della congiunzione tra le versioni dilatate delle diverse mappe. Cioè, il rilevatore agisce “tentando” di utilizzare il rumore per ricostruire le giunzioni.

Fig. 99: La mappa ottenuta dal rilevatore in multirisoluzione è sensibile al rumore in prossimità di bordi spezzati

La tabella che segue presenta i tempi di calcolo richiesti in media per gli esperimenti effettuati su un'immagine di ~85000 pixel.

	Operazione	***ws=37***	***ws=65***	***ws=105***
Dim. Immagine 289x292	Calcolo modulo del gradiente	4.7 sec	18.7 sec	1 min 4.7 sec
	Calcolo e applicazione termine di inibizione (base)	2.4 sec	9.4 sec	32.7 sec
	Calcolo e applicazione termine di inibizione (migliorato)	18.4 sec	1 min 15.29 sec	4 min 20.6 sec

Operazione	***Tempo***
Binarizzazione	0.1 sec
Multiscala, m=3	2 min 3.80 sec

Dalle misure riportate nelle tabelle si osservano tempi di elaborazione piuttosto alti. In particolare, i passi che richiedono la percentuale maggiore dei tempi di calcolo risultano essere, senza sorpresa, quelli relativi alle convoluzioni. Nel caso del rilevamento secondo il paradigma in multirisoluzione, inoltre, all'avanzare dell'elaborazione i tempi si dilatano, in quanto la maschera di convoluzione cresce con la scala. Le convoluzioni hanno complessità computazionale $O(NlogN)$, dove N è la dimensione lineare dell'immagine, mentre le altre operazioni (operazioni morfologiche, ridimensionamenti, ecc.) hanno tutte complessità lineare. Quindi, la complessità dell'algoritmo è $O(NlogN)$.

Si presti anche attenzione al fatto che la fase di binarizzazione richiede tanto più tempo quante più sono le componenti connesse nell'immagine dei massimi locali, informazione non nota a priori. Tuttavia, si sono osservati tempi accettabili anche nelle condizioni meno favorevoli.

Occorre dunque, chiaramente, agire sul calcolo delle convoluzioni tra immagini per velocizzare il sistema di rilevamento. A tale scopo, è pensabile effettuare le convoluzioni svolgendo semplici prodotti nel dominio delle frequenze; tuttavia, dal momento che lo stesso calcolo della trasformata (e dell'anti-trasformata) di Fourier richiede tempi che possono anche essere alti, sarebbe opportuno individuare una soglia sulla dimensione della maschera di convoluzione oltre la quale passare al calcolo nel dominio delle frequenze, e al di sotto della quale effettuare le convoluzioni direttamente nello spazio. Da un punto di vista prettamente

tecnico, tra l'altro, la libreria CImg (cap. 4) si può interfacciare all'ottima FFTW, altra libreria (sviluppata al MIT) piuttosto nota per la rapidità con cui permette di calcolare la DFT di segnali a due dimensioni.

Con queste modifiche ci si aspetta un netto miglioramento nei tempi di elaborazione, rimanendo tuttavia abbastanza lontani da una possibile realizzazione pensata per agire in tempo reale.

5.2 BodyTracker

BodyTracker è l'applicazione che realizza i sistemi di rilevamento discussi nella sez. 3.3. In questo capitolo concentriamo la nostra attenzione sulla tecnica che si basa sul *rilevamento del moto coerente*, e ne diamo una valutazione sulle prestazioni dal punto di vista computazionale (in quanto una valutazione qualitativa è stata già presentata in precedenza).

Come nel caso del capitolo precedente, l'applicazione presente si basa su alcuni parametri, che vengono riassunti in tabella:

Parametro	***Descrizione***
n	Numero di punti contigui per il *segment test*
t	Soglia da sommare/sottrarre agli angoli candidati per il *segment test*
nm	Indica se applicare o no la soppressione degli angoli non-massimi
d	Distanza massima (in pixel) tra cluster
q	Soglia sulla probabilità di moto rigido; è un valore compreso tra 0 e 1
dt	Distanza massima (in pixel) tra due traiettorie per poterle fondere insieme
pt	Percentuale minima di tempo di coesistenza di due traiettorie, per poterle fondere insieme (tra 0 e 1)
cd	Distanza minima (in pixel) tra un corpo e i confini dell'immagine
st	Numero di frame per le traiettorie da considerare “brevi”
sb	Distanza massima che un cluster può percorrere per essere considerato “fermo”

Il numero di parametri da configurare sicuramente non gioca a favore dell'algoritmo; tuttavia, molti di questi parametri dipendono dal tipo di scena video (ad esempio un'autostrada vista da una certa angolazione) e vanno configurati *una tantum*, altri invece possono essere valorizzati allo stesso modo per diverse applicazioni in quanto rappresentano requisiti di efficienza (parametri n, q, sb). Inoltre, sebbene il software implementato attualmente non realizzi questa funzionalità, alcuni parametri possono essere messi in relazione tra loro, o possono auto-configurarsi sulla base di alcune "scoperte" fatte in fase di inizializzazione.

Gli esperimenti seguenti sono stati effettuati su brevi sequenze video con frame di 320x400 pixel e FPS pari a 15. Le riprese sono state effettuate a colori ma convertite in toni di grigio per il processo di elaborazione; tuttavia, i video finali vengono mostrati a colori per una migliore rappresentazione.

Il software permette attualmente all'utente di impostare i soli parametri n, t, nm. Inoltre, permette di selezionare a mano una zona della scena video da escludere dal processing per diminuire i tempi di calcolo e l'eventuale rilevamento di falsi positivi. Dal momento che le riprese su cui si sono effettuati gli esperimenti contengono corpi di dimensioni simili, gli altri parametri sono stati impostati (via codice) come segue:

Param.	***Valore***
d	70
q	75.0
dt	25
pt	0.1
cd	50
st	3
sb	7

Con questi valori, il programma offre i risultati mostrati in sez. 3.3.3. C'è da dire che, a parte per quel che riguarda il rilevamento degli angoli, operazione che può essere svolta tranquillamente in tempo reale, i tempi di calcolo sono piuttosto alti.

Fig. 100: Frame tratti da due video di esempio; in rosso sono indicati gli angoli di Rosten-Drummond. Il primo frame in particolare è molto rumoroso, mentre il secondo presenta oggetti statici.

La tabella riportata in basso fa riferimento a due sequenze video: nella prima, a qualità volutamente bassa per studiare l'influenza del rumore, una mano si muove dall'alto verso il basso; nella seconda, a qualità più elevata, la scena è molto simile ma sono presenti due mani che, spostandosi, avvicinano le loro traiettorie che tra l'altro escono ed entrano dalla scena. Inoltre, nel secondo video sono presenti oggetti statici.

Anzitutto, per aggirare il problema della bassa qualità del primo video, la soglia t è stata impostata a 30 e il numero di punti $n=9$ perché questi valori danno, sperimentalmente, buoni risultati. Si osservi la fig. 101.

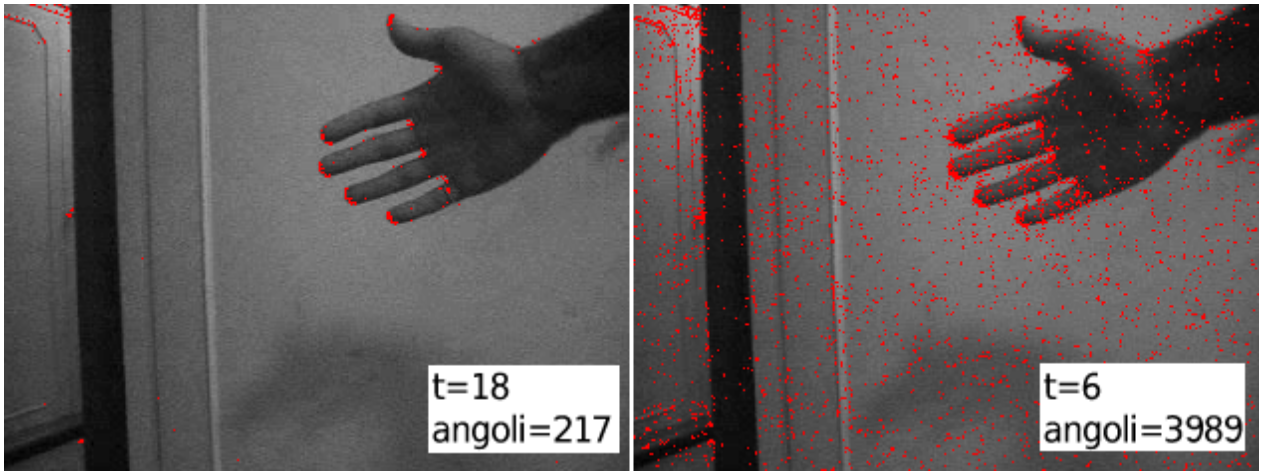


Fig. 101: A valori alti di t vengono rilevati pochi angoli in ogni frame e quindi il tracking può non avere successo. A valori troppo piccoli, d'altro canto, vengono rilevate feature dovute al rumore e alla quantizzazione.

Di fatti, anche per il secondo video di esempio, con questi valori per *n* e *t* si ottengono dei piccoli cumuli di feature sulle punte delle dita e sulle giunzioni tra di esse; questi cumuli formano dei cluster ben tracciabili con il flusso ottico, e seguono traiettorie altamente coerenti, il che rende una mano un corpo facilmente rilevabile dal sistema. Per questo motivo dunque, gli angoli adiacenti non devono essere rimossi perché utili, e il parametro *nm* è lasciato impostato a 0.

sequenza video	*zona esclusa*	*post-processing*	*tempo*
mano.avi	no	sì	7.6 sec
Dim. frame: 320x400	sì	sì	4.9 sec
Num. frame: 30	no	no	7.5 sec
	sì	no	4.9 sec
mani.avi	no	sì	~14 min
Dim. frame: 320x400	sì	sì	~1min 20 sec
Num frame: 249	no	no	~11 min
	sì	no	~1 min

Dalle misure si osserva che per video brevi e con pochi corpi (nell'esempio, una singola mano), il programma è abbastanza efficiente. Ovviamente, basta aumentare il numero di frame e dei corpi per far levitare i tempi (dal momento che l'applicazione agisce esclusivamente in modalità offline); si noti a questo proposito che la definizione di una regione di interesse all'interno della scena, per il tramite della definizione di una zona esclusa, aiuta ad abbassare di molto i tempi.

In particolare, la fase di clustering (sez. 3.3.3) dell'algoritmo richiede un'alta percentuale di tempo, rispetto a quello necessario a completare il rilevamento. D'altro canto, l'algoritmo utilizzato a questo proposito è un algoritmo semplice e immediato, e non trae vantaggio da alcuna tecnica particolare o libreria specializzata. Dal momento che esistono molti algoritmi diversamente ottimizzati per operazioni di questo tipo, l'adozione di uno di essi gioverebbe certamente in buona misura ai tempi di calcolo.

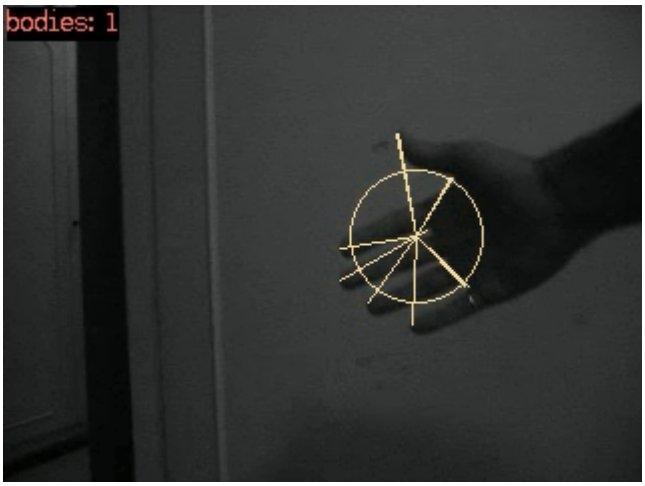


Fig. 102: Una mano che si sposta in ambiente fisso è facilmente riconoscibile dal sistema di rilevamento realizzato.

La fase di post-elaborazione richiede altresì molte risorse; tuttavia, senza questa fase i risultati del sistema di rilevamento non sono soddisfacenti, in quanto si risente troppo della presenza di oggetti statici e dell'effetto del rumore (sez. 3.3.3). Nel secondo video, il post-processing da solo elimina molti corpi candidati, portando il numero di corpi rilevati (nel corso dell'intera sequenza) da 630 a 25.

A ogni modo, come detto, allo stato attuale l'applicazione non è ottimizzata né il codice è stato scritto al meglio delle possibilità, essendo in questa sede di quasi esclusivo interesse l'efficacia degli algoritmi. In merito a ciò, si rimanda al capitolo successivo.

Capitolo 6: Conclusioni e sviluppi futuri

Il presente lavoro di tesi ha toccato diversi argomenti nell'ambito della Machine Vision, argomenti che possono essere racchiusi in due *grandi* obiettivi: l'estrazione di contorni e l'inseguimento di corpi.

Nel primo capitolo sono stati presentati alcuni concetti matematici di base, necessari per una comprensione adeguata degli argomenti successivi; particolare enfasi è dedicata alla teoria delle wavelet e dell'analisi multirisoluzione (MRA), basandosi in buona parte su uno studio svolto in precedenza [7], allo scopo di rendere chiari i sistemi di rilevamento a più scale discussi più avanti. Il capitolo 2 ha dunque introdotto il concetto di "bordo" da un punto di vista matematico, e ha presentato le tecniche classiche per la costruzione di un rilevatore di contorni. Tra queste tecniche, sicuramente, la più nota è quella proposta da Canny, che consiste sostanzialmente nell'applicazione di una maschera gaussiana all'immagine di partenza: in questo modo è possibile ridurre l'influenza del rumore e allo stesso tempo ottenere l'intensità dei bordi per gli oggetti presenti nell'immagine. A seguito di questa fase, l'immagine d'intensità dei bordi viene sottoposta a una forma di assottigliamento dei contorni e dunque a una forma di sogliatura degli stessi per ottenere la mappa binaria finale. Dunque sono stati studiati, e discussi nel capitolo 2.2, alcuni rilevatori di contorni basati su wavelet. Questi metodi adottano approcci piuttosto diversi e hanno alte prestazioni; è interessante notare che, sebbene si basino tutti inerentemente sull'analisi multirisoluzione, non tutti i metodi visti propongono un modo con cui utilizzare le informazioni presenti a più scale. Una delle idee originarie che sottostanno al presente lavoro di tesi, era quella di creare una wavelet adattata (*ad-hoc*) per il riconoscimento di bordi in ambienti fissi. I ricercatori, negli anni, hanno compiuto diversi sforzi in questa direzione, e vi sono molti risultati incoraggianti. Tuttavia, la complessità matematica dell'argomento e la multi-disciplinarietà a cui si presta, lo rendono un lavoro lungo e – per quanto affascinante – non ha potuto trovare spazio su queste pagine. Il contributo principale del capitolo 2 consiste nella realizzazione di un rilevatore di contorni basato su risultati noti nell'ambito della fisiologia umana [1]; il rilevatore agisce sostanzialmente calcolando, per l'immagine in analisi, un termine di inibizione che assume una certa intensità, e che viene poi sottratto dal modulo del gradiente per l'immagine di partenza. Dunque, un nuovo sistema di binarizzazione prende atto, sistema che trae beneficio dall'effetto di soppressione delle texture derivante dall'applicazione del termine di inibizione. Il rilevatore offre risultati migliori di gran parte degli estrattori di contorni attuali, in termini di texture soppresse e di contorni rilevati. Il capitolo 3 ha presentato il problema della registrazione traslazionale di immagini, al fine di costruire un rilevatore di corpi in sequenze video basato sul moto. A questo scopo, è stato realizzato un sistema per il rilevamento di angoli in real-time che prevede una fase di apprendimento (chiamato FAST [27]), ed è stato introdotto il concetto di flusso ottico; dunque,

sono state proposte alcune tecniche originali, e un rilevatore combinato che faccia uso di informazioni sull'aspetto delle entità d'interesse. Il capitolo 4 presenta il software realizzato e ne dà una descrizione dell'architettura al livello "giusto" di dettaglio. Sono state implementate tre applicazioni: *BMDetect*, un'applicazione per il rilevamento dei contorni basata sugli schemi di estrazione del cap. 2.3; *FAST Trainer*, un piccolo programma che realizza la fase di apprendimento del rilevatore di angoli FAST; *BodyTracker*, il sistema di riconoscimento dei corpi che si basa sulle tecniche del cap. 3.3. Tuttavia, il capitolo non è pensato come un manuale per le applicazioni, per il quale si rimanda ai riferimenti bibliografici. Nel capitolo 5 vengono discussi i risultati ottenuti dagli esperimenti svolti con le applicazioni sviluppate. Gli esperimenti mostrano che i sistemi di rilevamento realizzati sono piuttosto efficaci, al costo però dell'utilizzo delle risorse e dei tempi di calcolo; sono infatti possibili diversi miglioramenti. Per quanto riguarda i rilevatori di contorni, anzitutto, è necessario svolgere le convoluzioni in frequenza, nelle situazioni in cui la dimensione della maschera di convoluzione supera una certa soglia. Una modifica di questo tipo diminuirebbe certamente di molto i tempi di calcolo, ed è di rapida implementazione. Un secondo miglioramento, adottato tra l'altro dagli autori di [1], consiste nell'applicazione di una fase di riduzione del rumore prima ancora del calcolo del gradiente; tale fase consentirebbe di migliorare di gran lunga i risultati del processo di estrazione, già ottimi di per sé. Per quel che riguarda l'applicazione di riconoscimento di corpi in sequenze video, invece, bisogna tener conto che il codice allo stato attuale non è affatto ottimizzato, quindi sono attesi miglioramenti anche sostanziali a seguito di una revisione dettagliata. Oltre a quanto detto nel capitolo dedicato ai risultati sperimentali, è importante osservare che il sistema di tracking si presta bene a una realizzazione parallela. Ad esempio, la fase di clustering può essere facilmente distribuita tra calcolatori, dal momento che è un'operazione che va svolta in fase di inizializzazione delle traiettorie per ogni singolo frame. Un calcolatore "master" svolge le funzioni di controllo e coordina le attività dei calcolatori secondari, raccogliendo i risultati e parallelizzando i compiti secondo determinate strategie. Anche le feature possono essere inseguite in maniera indipendente, ma in questo caso il lavoro di coordinazione deve essere piuttosto accurato, per evitare di effettuare il tracking di feature per le quali esiste già una traiettoria. In conclusione, parallelizzare i calcoli, in concomitanza alle ottimizzazioni già proposte (capitolo 5), può far pensare a una migrazione del software verso una implementazione real-time – o quasi. Di fatti, una funzionalità interessante e sicuramente piuttosto immediata da un punto di vista realizzativo, consiste nel portare il sistema di tracking ad operare in maniera ibrida, a metà tra real-time ed elaborazione offline. L'idea è la seguente: l'algoritmo può essere applicato in tempo "quasi" reale su un buffer lungo un certo numero di secondi (o frame), consentendo in questo modo di vincere alcune limitazioni presenti nel sistema corrente. Per cominciare, elaborare -anche in maniera parallela- un buffer limitato di frame per volta consente non solo di diminuire (di molto) i tempi di calcolo, ma anche di revisionare parti dell'algoritmo che attualmente prevedono traiettorie complete. Inoltre, il sistema stesso può essere esteso per

apprendere dei pattern di spostamento delle entità: ad esempio osservando, posta la telecamera di fronte a un semaforo, che la maggioranza delle traiettorie va da una parte all'altra della strada. Una fase di learning di questo tipo doterebbe il rilevatore di capacità di apprendimento e lo metterebbe in grado, ad esempio, di imparare la posizione dei punti di ingresso e di uscita dalla scena, o di “riempire” i salti nelle traiettorie spezzate di cui si è parlato estensivamente in sez. 3.3.3. Salti che possono essere dovuti a occlusioni temporanee o camuffamenti da rumore, o ancora da variazioni improvvise di luminosità, o trasformazioni morfologiche degli stessi oggetti in analisi. Tuttavia, in questo approccio ibrido è necessario agire con la dovuta attenzione, in quanto, rimanendo di base un sistema di rilevamento basato sul moto, se un corpo smette di muoversi del tutto all'interno di un buffer, esso non viene più rilevato.

Bibliografia

[1] G.Papari, P.Campisi, N.Petkov and A.Neri, “A Biologically Motivated Multiresolution Approach to Contour Detection”, EURASIP Journal on Advances in Signal Processing, Vol. 2007

[2] R.Rifaat and W.Kinsner, “Experiments with Wavelet and other Edge Detection Techniques”, WESCANEX '97 Proceedings, May 22-23 1997

[3] J.Li, “A Wavelet Approach to Edge Detection”, Master Thesis in Mathematics, Sam Houston University, 2003

[4] W.K.Pratt, “Digital Image Processing - 3rd Edition”, Wiley-Interscience

[5] S.Eddins, “Connected component labeling”, MATLAB Central, disponibile online all'indirizzo: http://blogs.mathworks.com/steve/2007/06/13/connected-component-labeling-wrapping-up/

[6] L.Zhang and P.Bao, “Edge detection by scale multiplication in wavelet domain”, Pattern Recognition Letters 23 (2002) 1771-1784

[7] E.Rodolà, “Introduzione alle Wavelet e all'Analisi Multirisoluzione”, Report del Tirocinio 3 in Ingegneria Informatica, Università di Roma Tor Vergata, 2007

[8] M.Y.Shih and D.C.Tseng, “A wavelet-based multiresolution edge detection and tracking”, Image and Vision Computing 23 (2005) 441-451

[9] D.M.Gavrila, “The Visual Analysis of Human Movement: A Survey”, Computer Vision and Image Understanding, vol. 73 no. 1 January 1999

[10] S.Hanov, “Wavelets and Edge Detection”, CS698 Final Project, University of Waterloo

[11] M.Frigo and S.G.Johnson, "FFTW", disponibile online all'indirizzo http://www.fftw.org

[12] D.Glotsos, “Development of a wavelet-assisted edge-detection algorithm for boundary detection of Fine Needle Aspiration images of thyroid nodules”, ERASMUS visiting PhD student Report, Research Reports of CMP, Czech Technical University in Prague, No. 3, 2005

[13] J.R.Parker, “Advanced Edge Detection Techniques”, Algorithms for Image Processing and Computer Vision, John Wiley & Sons Ltd., 1996

[14] H.S.Neoh and A.Hazanchuk, “Adaptive Edge Detection for Real-Time Video Processing using FPGAs”, Altera Corporation, 2005

[15] D.Menascé and V.Almeida, "Capacity Planning for Web Services", Prentice Hall PTR, 2001

[16] C.Grigorescu, N.Petkov and M.A.Westenberg, “Contour and boundary detection improved by surround suppression of texture edges”, Image and Vision Computing 22 (2004) 609-622

[17] C.Harris and M.Stephens, “A combined corner and edge detector”, Proceedings of The Fourth Alvey Vision Conference, Manchester, pp 147-151. 1988

[18] B.D.Lucas and T.Kanade, “An Iterative Image Registration Technique with an Application to Stereo Vision”, Proc 7th Intl Joint Conf on Artificial Intelligence 1981, August 24-28, pp.674-679

[19] Trolltech, “Qt Reference Documentation (Open Source Edition)”, consultabile online all'indirizzo http://doc.trolltech.com/4.3/index.html

[20] D.Tschumperlé, “The CImg Library Documentation”, consultabile online all'indirizzo http://cimg.sourceforge.net/reference/index.html

[21] “FFMpeg”, http://ffmpeg.mplayerhq.hu

[22] T.B.Moeslund and E.Granum, "A Survey of Computer Vision-Based Human Motion Capture", Computer Vision and Image Understanding 81, 231-268 (2001)

[23] H.J.A.M.Heijmans, "Connected Morphological Operators and Filters for Binary Images", International Conference on Image Processing, 1997, Volume 2, Issue , 26-29 Oct 1997 Page(s): 211 - 214 vol.2

[24] C.Tomasi and T.Kanade, "Detection and Tracking of Point Features", Carnegie Mellon University Technical Report CMU-CS-91-132, April 1991

[25] Y.Ivanov, A.Bobick and J.Liu, "Fast Lighting Independent Background Subtraction", MIT Media Laboratory Perceptual Computing Section Technical Report No.437, 1998

[26] E.Rosten and T.Drummond, "Fusing points and lines for high performance tracking", October 2005, IEEE International Conference on Computer Vision

[27] E.Rosten and T.Drummond, "Machine learning for high-speed corner detection", May 2006, European Conference on Computer Vision

[28] A.Broggi, M.Bertozzi, A.Fascioli and M.Sechi, "Shape-based Pedestrian Detection", Proceedings of the IEEE Intelligent Vehicles Symposium, 2000

[29] G.J.Brostow and R.Cipolla, "Unsupervised Bayesian Detection of Independent Motion in Crowds", IEEE Computer Vision and Pattern Recognition, I: 594-601, 2006

[30] E.Rodolà, “BMDetect: Manuale d'uso”, disponibile online

[31] Wikipedia, "Snow (codec)", http://en.wikipedia.org/wiki/Snow_(codec)