Università degli studi di Roma "Tor Vergata"

Facoltà di Ingegneria

Corso di laurea in Ingegneria Elettronica

Tesi di Laurea in Robotica Industriale

Sistemi di visione in tempo reale per la navigazione di robot mobili

Laureando: Ivano Ras
matr. II003508

Relatore: Prof.Ing. Salvatore Nicosia

Correlatore: Ing. Francesco Martinelli

Anno Accademico 2002 / 2003

Alla mia famiglia,

Indice

Introduzione

I sistemi basati sulla visione artificiale sono in *fase* d'integrazione alle tecnologie già al servizio dell'uomo.
Un classico esempio è quello dei sistemi, come il pilota automatico per le autovetture, in grado di "*vedere*" la strada, le altre automobili, i cartelli stradali, i pedoni intenti ad attraversare la carreggiata e di conseguenza *prendere decisioni* riguardo la *guida* dell'autoveicolo.
Altri *sistemi*, in continuo sviluppo, sono quelli per la *videosorveglianza di ambienti privati* (banche, uffici, parcheggi, etc.) di cui esistono varie tipologie in grado di "distinguere" le persone e di tenere sotto continuo controllo i loro movimenti.
A tal proposito, in Italia, vanno sicuramente citati il progetto ARGO (Univ. di Parma; Broggi et al.; tra i numerosi articoli si veda [4] e [15]) ed i progetti di videosorveglianza portati avanti dal CVG dell'Univ. di Bologna.
Non è poi difficile richiamare alla mente le ripetute occasioni in cui i *media* mettono in risalto le qualità del robot ASIMO (Honda), nell'adempiere alla parte di *cerimoniere* in apertura di eventi e fiere, probabilmente con l'intento di evidenziare il cambiamento dei tempi. In tali occasioni il robot si *destreggia* in un ambiente "progettato" per *evidenziare* la *sua* presenza come fosse una celebrità invitata a far mostra di se e del suo *design*. Sorprendono le sue capacità di *riconoscimento delle persone* e dei *comandi* che gli vengono impartiti in *maniera gestuale* uniti all'abilità di *imitazione della postura altrui* ed alla capacità di spostarsi autonomamente "*camminando*". (cfr. link (1))

Questi prodotti e progetti hanno in comune l'utilizzo della *Visione Artificiale* (*Computer Vision*), che si occupa dei metodi inerenti l'estrazione di informazioni dai dati acquisiti mediante una o più videocamere digitali, dall'ambiente esterno.
L'impiego sempre più diffuso della Visione Artificiale nei vari ambiti lavorativi è consentito dalla continua *crescita di potenza di calcolo* dei computer commerciali.

Nel presente lavoro vengono presentate ***due*** applicazioni circa l'*elaborazione* in *tempo reale* di informazioni acquisite dall'ambiente, per mezzo di una videocamera commerciale (*webcam*) (fig.1). Nella *prima* applicazione, la webcam dev'essere installata su un *robot mobile* (ad esempio il Nomad 150, fig.2), equipaggiato anche di *PC portatile*, necessario per l'elaborazione autonoma dei dati video e per la navigazione in un *ambiente chiuso*.
L'*ambiente chiuso* , ad esempio un sistema di corridoi, può subire *cambiamenti della viabilità nel tempo*.
Nella *seconda* applicazione, la webcam è prevista in *posizione*, prevalentemente, *fissa*.

Questo lavoro ha comportato la *realizzazione software ex novo* degli *algoritmi di base* per la Visione Artificiale *in tempo reale* in ambiente Microsoft Windows 2000/XP (filtraggi per la fase di *pre-elaborazione*, binarizzazione e rivelazione dei contorni spaziali e temporali per la fase di *segmentazione*, rete neurale di Hopfield per la fase di *riconoscimento*, etc). Di fatto, la realizzazione, costituisce un *banco di lavoro* basato su PC e sulle webcam della Logitech, *pronto* per lo sviluppo di altre applicazioni di carattere universitario e professionale.
Essendo gli algoritmi scritti in C++, le applicazioni realizzate possono essere facilmente *trasportate* in *ambiente Linux*, data la semplice reperibilità su Internet dei *driver* e delle *librerie* della Logitech, per tale ambiente.

La **prima** e *principale* applicazione si occupa del riconoscimento *in tempo reale* di informazioni rappresentate all'*interno* di appositi *target circolari*, come nel caso della segnaletica stradale dove un cartello di forma circolare reca l'iscrizione del segnale di divieto o di obbligo (informazioni *statiche*) oppure dove un tabellone (a LED), lungo l'autostrada, comunica le condizioni di viabilità stradale ai conducenti degli autoveicoli (informazioni *dinamiche*).
Nel caso di *ambiente chiuso*, tale applicazione risulta utile per *comunicare* ai robot le informazioni relative al percorso per loro previsto in un determinato lasso di tempo. Ciò avviene in maniera *visuale*, mediante tabelloni LED

inseriti in *cornici circolari* e fissati ad una parete. (cfr. fig.3).
Ovviamente tale applicazione può essere utilizzata per *altri scopi*, uno dei quali può essere l'*inseguimento*, da parte del robot, di una *segnaletica specifica* in *movimento*, tra le altre presenti nel suo campo visivo. Un altro utilizzo previsto è la *comunicazione visuale* tra robot, a scopo di *cooperazione* in lavori specifici in *ambienti speciali* in cui si deve ridurre o addirittura fare a meno della comunicazione acustica e/o via etere. In tal caso, il robot mobile dovrebbe essere anche dotato di un piccolo display (fissato allo *chassis*, fig.4) per la visualizzazione delle informazioni destinate agli altri robot (cfr.fig.5).

La **seconda** applicazione si occupa della rivelazione di oggetti in movimento in *ambiente chiuso* (*illuminazione non strutturata*), mediante la visualizzazione su schermo, *in tempo reale*, delle loro sagome.
Tale applicazione si presta, evidentemente, come *base* per un sistema di videosorveglianza.

Le due applicazioni descritte separatamente, possono *coesistere* nello stesso sistema di visione, qualora lo si richieda, con l'accortezza di utilizzare un computer di poco più veloce di quello impiegato in questa sede (Cap.3).

In particolare il contributo del presente lavoro è stato quello di realizzare, mediante noti algoritmi di visione artificiale (aggiustati per il funzionamento in *tempo reale* ed i cui riferimenti bibliografici saranno dati nel corso della presentazione), il sistema di riconoscimento di "segnaletica" ed il sistema di rivelazione del movimento. Il tutto avviene *in tempo reale*, su una videocamera commerciale (*webcam*) impostata su una risoluzione di 320x240 pixel e a 30 fotogrammi al secondo (l'elaborazione di ogni fotogramma avviene *entro* un trentesimo di secondo, prima cioè dell'acquisizione del fotogramma successivo).

Il contenuto della tesi è articolato nel seguente modo:

Nel *primo* capitolo è riportata una breve descrizione di qualcuno dei metodi matematici utilizzati nel campo della visione artificiale.

Nel *secondo* capitolo viene esaminata in dettaglio la tecnica di riconoscimento mediante rete neurale, impiegata nella prima applicazione.

Nel *terzo* si passa in rassegna l'hardware ed il software su cui è basato lo sviluppo delle due applicazioni.

Il *quarto* ed il *quinto* capitolo si occupano di spiegare ed approfondire i vari aspetti dell'applicazione di riconoscimento di "segnaletica" e dell'applicazione di rivelazione di sagome in movimento.

Nel *sesto* si analizzano rapidamente le prestazioni delle due applicazioni e nel *settimo*, infine, si tracciano le conclusioni e gli sviluppi futuri.

fig.1 webcam QuickCam PRO 4000

fig.2 Robot mobile **Nomad 150**: 16 sensori ad ultrasuoni per il rilevamento di ostacoli + 20 sensori di pressione (contatto) + 3 sensori odometrici (un encoder per ruota)

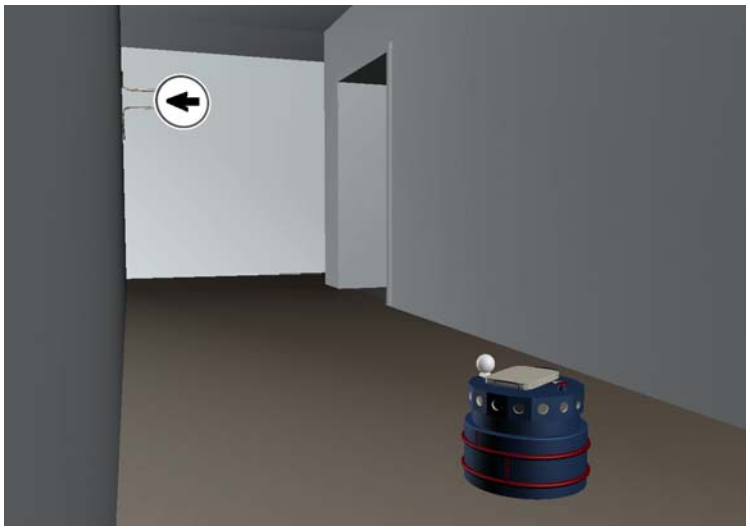


fig.3 esempio di comunicazione mediante tabellone

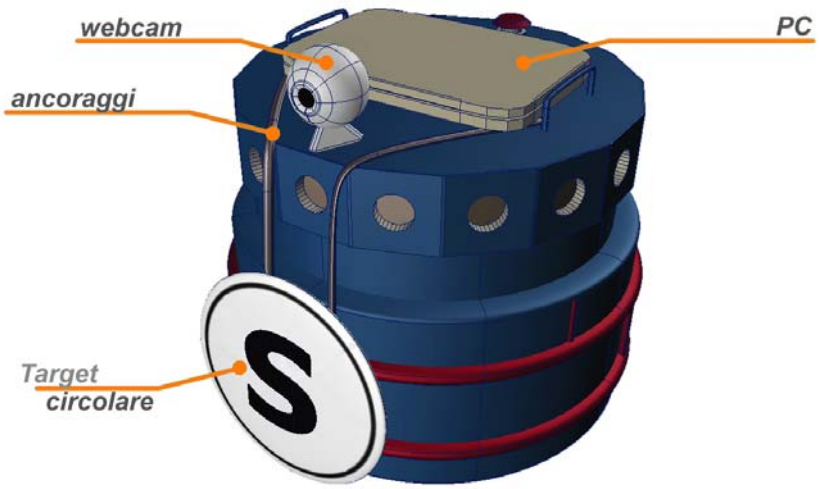


fig.4 esempio di possibile equipaggiamento per *comunicazione visuale tra robot.* Un sistema *più economico* consiste nell'utilizzare il monitor del PC portatile per *visualizzare* i diversi target.

fig.5 esempio di comunicazione tra robot.

1 Visione Artificiale (Computer Vision)

La *visione* può essere definita come il *processo* di *estrazione*, *caratterizzazione* ed *interpretazione* delle informazioni provenienti dalle immagini acquisite dal mondo esterno.Tale processo può essere suddiviso in sei aree principali (cfr.[5]):

1) ***percezione*** (processo che fornisce un'immagine visiva)
2) ***pre-elaborazione*** (tecniche di riduzione del rumore e di miglioramento dei particolari)
3) ***segmentazione*** (suddivisione dell'immagine in oggetti di interesse)
4) ***descrizione*** (calcolo delle caratteristiche utilizzabili per differenziare un tipo di oggetti da un altro)
5) ***riconoscimento*** (processo di identificazione di tali oggetti)
6) ***interpretazione*** (conferimento di significato ad un insieme di oggetti riconosciuti)

Queste aree sono anche *raggruppabili* in *3* livelli di elaborazione (*visione a basso, medio* ed *alto livello* corrispondenti a diversi livelli di descrizione dell'immagine).
Anche se *non vi sono limiti precisi* tra queste suddivisioni, esse forniscono tuttavia una struttura per la categorizzazione dei diversi processi inerenti un sistema visivo di una macchina.

Di seguito verranno considerate più in dettaglio le *singole* aree e qualche metodo matematico ad esse associato.

1.1 Percezione

L'acquisizione delle immagini viene effettuata mediante videocamera digitale (in particolare qui viene utilizzata una *webcam* ovvero una videocamera commerciale a basso costo dedicata prevalentemente alla videocomunicazione via Internet).

Le immagini sono convertite in segnali elettrici mediante sensori ottici.
Il campionamento spaziale (matrice nxm di fotoelementi) e la quantizzazione dell'ampiezza (livelli di grigio) del segnale elettrico, danno origine ad un'immagine *digitale*.

Il sensore ottico in questione è il CCD (dispositivo a trasferimento di carica). Ogni fotoelemento è costituito da un dispositivo al Silicio. I fotoni giunti al fotoelemento, attraversano una struttura di gate di Si policristallino trasparente e vengono assorbiti dallo strato interno (sempre di Si), dando origine a coppie elettrone-lacuna. La carica elettrica generata, in ogni fotoelemento, è proporzionale all'intensità luminosa focalizzata in quel punto.
Il numero di fotoelementi ovvero il numero di valori d'intensità luminosa acquisiti (nxm) definisce la *risoluzione* dell'immagine ottenuta, es. 640x480 pixel.

Tali valori (quantizzati) vengono poi inviati e memorizzati nella memoria del computer che procederà alla loro elaborazione.

1.2 Pre-elaborazione

Vedremo ora brevemente qualche tecnica che permette il miglioramento dell'immagine, predisponendola ad una successiva estrazione dell'informazione.

1.2.1 Filtraggio

Si adotta allo scopo di ridurre il rumore e/o altri effetti indesiderati che si possono produrre nell'immagine acquisita per effetto del campionamento, quantizzazione, trasmissione (anche compressione 'lossy' dei dati nel nostro caso) o per via di disturbi ambientali di varia natura.
Una possibile *suddivisione* per i filtri è quella in *lineari* e *non lineari*, a seconda che essi possano o meno essere espressi mediante la sola *convoluzione*. L'operazione di *convoluzione* (caso discreto e nel dominio spaziale) è riportata sotto:

$$g(x,y) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(x-i,y-j)\, m(i,j) \equiv f() \bullet m() ;$$

dove f(x,y) è l'intensità nel punto (x,y) ed m(x,y) è una funzione denominata *maschera*.

(1) media degli intorni

Tale semplice tecnica lineare nel dominio spaziale, consiste nel generare un'immagine g(x,y), sulla base di quella acquisita f(x,y), in cui l'intensità in ogni punto (x,y) si ottiene operando la media dei valori d'intensità dei pixel di f() appartenenti ad un *intorno* predefinito S di (x,y) (ad esempio 3x3, chiamato anche 8-intorno):

per ogni punto (x,y): $g(x,y) = 1/N \sum_{n,m \in S} \sum f(n,m)$;

dove N è il numero di punti nell'intorno S ed $m( ) = 1$ per ogni n,m di S ;

Nel caso di sistemi in tempo reale una tale operazione potrebbe essere proibitiva in termini di calcolo, pertanto dev'essere omessa, richiedendo ai moduli di segmentazione una maggior robustezza nel suddividere un'immagine affetta da rumore.
In alcuni casi è possibile ottenere un tale filtraggio mettendo l'immagine fuori fuoco, cioè agendo sull'obiettivo della videocamera. Con lo stesso espediente si possono utilizzare filtri ottici colorati o polarizzanti dinanzi all'obiettivo, allo scopo di sgravare il computer da filtraggi software per applicazioni specifiche.

(2) filtraggio di immagini binarie

Tali immagini composte solo da pixel neri (1) e bianchi (0) hanno origine dall'uso dell'illuminazione in controluce o da un'illuminazione strutturata, oppure da una precedente elaborazione di binarizzazione o sogliatura, allo scopo di ricercare contorni di oggetti.
I disturbi introdotti nell'immagine possono causare contorni irregolari, piccoli buchi e punti isolati. L'idea alla base di tale filtraggio è l'uso di una funzione booleana (funzione che fa uso di funzioni logiche AND (*),OR(+) e NOT) valutata in un intorno con centro in un pixel **p** e che poi assegna a **p** un valore 0 o 1.
Seguono alcuni esempi di vantaggi offerti da funzioni booleane specifiche:
(α) riempie piccoli buchi in aree per il resto nere, (β) riempie piccoli buchi in segmenti rettilinei, (γ) elimina gli 1 isolati, (δ) elimina piccole protuberanze lungo segmenti rettilinei, (ε) ripristina i punti mancanti agli angoli;
per i primi due esempi (α e β) la funzione è:

a b c
d **p** e B = p + b*g*(d + e) + d*e*(b + g);
f g h

Più avanti si citerà una generalizzazione di tale metodo.

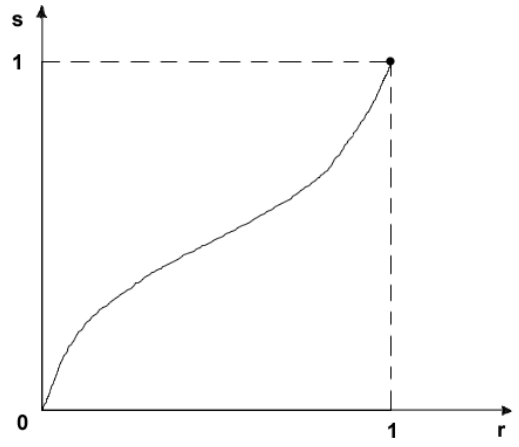
1.2.2 Equalizzazione dell'istogramma per il miglioramento delle immagini

La variazione delle condizioni operative, in particolar modo dell'illuminazione, obbliga a realizzare algoritmi di adattamento automatico a tali mutate condizioni.

Sia **r** l'intensità di un pixel in un'immagine da migliorare, con **r** normalizzata ($0 \leq \mathbf{r} \leq 1$) e continua. Porremo l'attenzione sulle trasformazioni **s=T(r)** che restituiscono un valore di intensità **s** per ogni pixel **r** dell'immagine d'ingresso.
Si suppone che:

1) T(r) sia una funzione ad un sol valore crescente monotonamente in $0 \leq \mathbf{T(r)} \leq 1$.
2) $0 \leq \mathbf{T(r)} \leq 1$ per $0 \leq \mathbf{r} \leq 1$.

La 1^a condizione preserva l'ordine da nero a bianco nella scala delle intensità e la 2^a garantisce una corrispondenza consistente con il range da 0 ad 1 di valori dei pixel.
Un esempio di trasformazione che verifica la *1*) e la *2*) è il seguente:



La funzione inversa $\mathbf{r = T^{-1}(s)}$ per ipotesi soddisfa le 2 condizioni già descritte.
Le variabili **r** ed **s** sono quantità casuali in [0,1] ed in quanto tali possono

essere caratterizzate da **funzioni densità di probabilità** (PDF) $p_r(r)$ e $p_s(s)$.

Dalla teoria della probabilità deriva che se $\mathbf{p_r(r)}$ e **T(r)** sono note e $\mathbf{T^{-1}(r)}$ soddisfa la 1ª condizione allora la PDF delle intensità trasformate è data da:

$$p_s(s) = [\ p_r(r) \cdot dr/ds]_{\ r = T^{-1}(s)}\ ;$$

Supponiamo di scegliere una funzione di trasf. **T(r)** specifica:

$$(*) \qquad s = T(r) = \int_0^r p_r(w)\, dw\ ; \qquad \text{con } 0 \le \mathbf{r} \le 1\ ;$$

Il secondo membro è la *funzione di distribuzione cumulativa* di $p_r(w)$ ed è compatibile con la *1*) e la *2*). La derivata di **s** rispetto ad **r** per questa particolare funzione di trasformazione risulta essere:

$$ds / dr = p_r(r)\ ;$$

La sostituzione della quale nella terzultima formula, ci restituisce: $p_s(s) = 1$ per $0 \le \mathbf{s} \le 1$, ovvero si ha una *densità uniforme* nell'intervallo di definizione della variabile trasformata s.
Si noti che tale risultato è indipendente dalla funzione di trasformazione inversa, fatto importante in quanto è spesso difficile trovare $T^{-1}(s)$ analiticamente.
Inoltre va messo in evidenza che usando la trasformazione in (*) si ottengono intensità trasformate che hanno sempre una PDF *piatta*, indipendentemente dalla forma di $p_r(r)$; proprietà ideale per il miglioramento automatico ovvero per bilanciare la distribuzione delle intensità anche se siamo in presenza di una variabilità nell'illuminazione.

Nel caso *discreto* (quello che ci riguarda direttamente): $p_r(r) = n_k / N$;
dove n_k è il numero di pixel aventi il livello di grigio k ed N è il numero totale di pixel nell'immagine.

La trasformazione ora assume la seguente forma:

$$s_k = T(r_k) = \sum_{i=0}^{k} n_i / N = \sum_{i=0}^{k} p_r(r_i) \; ; \qquad \text{con } 0 \le r_k, s_k \le 1 \; ;$$

e $k = 0,1,2,\ldots,255$. I valori di s_k saranno poi riscalati a 255 ed arrotondati all'intero più vicino in modo che i valori di uscita di tale trasformazione apparterranno al range [0,255].
Va notato che la discretizzazione e l'arrotondamento di s_k implicheranno che l'immagine trasformata avrà un *istogramma non perfettamente uniforme*.

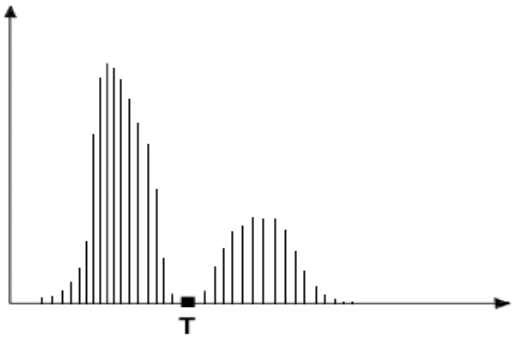
Tale elaborazione (disattivabile) è eseguita *automaticamente* dalla webcam, non appena le condizioni di luce mutano in modo significativo (ciò avviene una frazione di secondo dopo il cambiamento delle condizioni di luminosità).

1.3 Segmentazione

La segmentazione è il processo che tenta una suddivisione della scena nelle sue *parti costituenti* (ovvero in 'oggetti', in senso lato). Si darà qualche cenno circa le tecniche utilizzate per *estrarre* gli 'oggetti', in seguito sottoposti ad analisi delle loro caratteristiche.

1.3.1 Sogliatura (Thresholding)

Per la sua alta velocità di elaborazione è impiegata principalmente nei sistemi visivi industriali nella ricerca di oggetti e per il controllo qualità. Di seguito viene descritta l'idea base.
Supponiamo che l'istogramma, raffigurato di seguito, corrisponda ad un'immagine f(x,y), costituita da oggetti chiari su sfondo scuro (o viceversa), in modo che i pixel, degli oggetti e dello sfondo, abbiano intensità raggruppate in 2 gruppi dominanti.



Un modo ovvio di estrarre gli oggetti dallo sfondo è selezionare una soglia (**T**) che suddivida i gruppi di intensità dei pixel. Quindi qualsiasi pixel di coordinate (x,y), per il quale f(x,y) > T , viene riconosciuto come punto di un oggetto, altrimenti risulta appartenente allo sfondo.

Un caso più generale è caratterizzato da un grafico con 3 picchi (3 gruppi

dominanti) e 2 soglie T_1 e T_2, atto a rappresentare, ad esempio, due tipi diversi di oggetti chiari su sfondo scuro.
Applicando lo stesso metodo di prima, classifichiamo un punto (x,y) appartenente ad una classe di oggetti se $T_1 < f(x,y) \leq T_2$, all'altra classe di oggetti se $f(x,y) > T_2$, ed allo sfondo se $f(x,y) \leq T_1$. Questa *sogliatura a più livelli* è generalmente *meno affidabile* della sua controparte *a soglia singola* a causa della difficoltà nello stabilire soglie a più livelli che effettivamente isolino le regioni d'interesse, in particolar modo quando il numero dei gruppi d'intensità dell'istogramma corrispondente è alto. Tipicamente i problemi di tale natura, se trattati con una sogliatura, danno risultati migliori se si utilizza una *singola soglia variabile*.

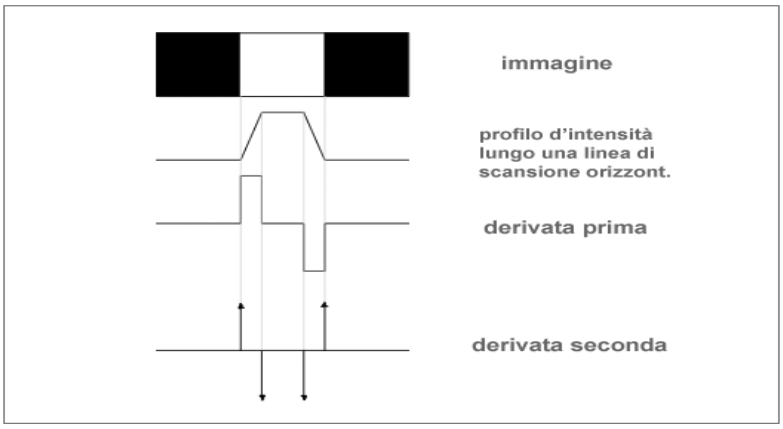
La sogliatura può essere vista come un'operazione che implica il confronto con una funzione $T=T[x,y,p(x,y),f(x,y)]$, con f(x,y) l'intensità nel punto (x,y) e $p(x,y)$ una data *proprietà locale* di tale punto, per esempio l'intensità media in un intorno di centro (x,y).
L'immagine sogliata g(x,y) si ottiene dalla seguente condizione:

$$g(x,y) = 1 \qquad \text{se } f(x,y) > T$$
$$g(x,y) = 0 \qquad \text{se } f(x,y) \leq T$$

Se T dipende solo da f(x,y), la soglia è detta *globale*. Se la dipendenza è estesa a $p(x,y)$, la soglia è detta *locale* e se dipende anche da x,y si dice *dinamica*.

1.3.2 Estrazione dei contorni

L'idea di base di molte tecniche di estrazione dei contorni (spaziali) è il calcolo di un operatore derivativo locale. La figura esemplifica il concetto, partendo da un oggetto chiaro su sfondo scuro:



Il profilo d'intensità dei margini (zone di passaggio dal nero al bianco) è rappresentato da una *rampa*, e non come ci si potrebbe aspettare da un cambiamento a *gradino* dell'intensità. Tale modellazione del margine rispetta il fatto che i contorni delle immagini digitali sono generalmente confusi come risultato del campionamento.
Risulta quindi evidente che per ricercare un bordo può essere impiegato il valore della derivata prima, mentre il segno della derivata seconda può essere impiegato per determinare se un pixel giace sul lato nero o su quello bianco di un margine.

Si ricorda che la derivata prima, per ogni punto dell'immagine, può essere ottenuta utilizzando l'*operatore gradiente* mentre la derivata seconda utilizzando l'*operatore di Laplace*.

(1) operatori a gradiente

Il gradiente di un'immagine f(x,y) nella posizione (x,y) è definito come il vettore bidimensionale:

$$G\,[f(x,y)] = [\partial f/\partial x\ ,\ \partial f/\partial y]\ ;$$

Il vettore G, come noto, è orientato nella direzione della variazione più rapida di f nella posizione (x,y). Comunque , per lo scopo presente, è d'interesse solo il *modulo* di tale vettore, chiamato ora solo *gradiente*:

$$Gr\,[f(x,y)] = (G_x^{\,2} + G_y^{\,2})^{1/2}\ ;$$

che si è soliti approssimare, per uso algoritmico, con i valori assoluti:

$$Gr\,[f(x,y)] = |G_x| + |G_y|\ ;$$

si riporta anche la direzione del gradiente: $\theta = \tan^{-1}(G_y/G_x)$;

Sono possibili *vari modi* per calcolare le derivate prime (nell'appross.discreta). Il metodo più immediato consiste nell'effettuare le differenze tra i valori di pixel adiacenti (*operatori di Roberts*):

$$G_x = f(x,y) - f(x-1,y)\ ; \qquad G_y = f(x,y) - f(x,y-1)\ ;$$

Una definizione leggermente più complessa che coinvolge i pixel di un'intorno 3x3 con centro in (x,y) è data da:

$$G_x = [\ f(x+1,y-1) + 2\,f(x+1,y) + f(x+1,y+1)\ ] - [f(x-1,y-1) + 2\,f(x-1,y) + f(x-1,y+1)\ ]\ ;$$

$$G_y = [\ f(x-1,y+1) + 2\,f(x,y+1) + f(x+1,y+1)\ ] - [f(x-1,y-1) + 2\,f(x,y-1) + f(x+1,y-1)\ ]\ ;$$

ovvero le G_x,G_y possono essere calcolate usando le seguenti *maschere* al pixel in questione:

$$\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \qquad\qquad \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix}$$

che sono chiamate *operatori* (o *maschere*) *di Sobel* (l'*operatore di Prewitt* è praticamente uguale tranne che ha, nelle maschere, degli 1 al posto dei 2). Applicandoli ad ogni pixel si ottiene il gradiente in tutti i punti.

(2) operatori di Laplace

Brevemente, facendo il prodotto scalare del gradiente con se stesso si ottiene:

$$G\,[f(x,y)] \cdot G\,[f(x,y)] = Lp\,[f(x,y)] = \partial^2 f/\partial x^2 + \partial^2 f/\partial y^2 \;;$$

anche qui ci sono vari modi per calcolare le derivate seconde, il *più immediato* è:

$$\partial^2 f/\partial x^2 = [f(x+1,y)-f(x,y)]-[f(x,y)-f(x-1,y)] = f(x+1,y)-2f(x,y)+f(x-1,y)$$

e analogamente rispetto ad y, quindi riferendoci alla rappresentazione delle maschere:

$$\begin{matrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{matrix} \quad + \quad \begin{matrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{matrix} \quad = \quad \begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

Comunque tale operatore è *suscettibile* al rumore ed in generale più l'ordine della derivata è alto e più cresce la sensibilità dell'operatore.

Quindi è preferibile utilizzare un operatore Laplaciano accoppiato ad un filtro Gaussiano (che è simile a quello visto prima di *media degli intorni* (linear noise smoothing)).
Tale soluzione (1979) è nota come *operatore di Marr-Hildreth* oppure come *operatore LoG* (Laplacian-of-Gaussian).
Consideriamo un immagine f(x,y) a cui si applica il filtro Gaussiano $G_{auss}(x,y)$ e sul cui risultato si applica il Laplaciano:

$$f'(x,y) = Lp\,[\ f(x,y) \bullet G_{auss}(x,y)\ ]; \qquad (\ ' \bullet ' \equiv \textit{convoluzione}\ )$$

ciò rivela i cambiamenti di segno (*zero-crossing*) (cfr. figura precedente) e quindi rivela i contorni, inoltre dato che la derivata e la convoluzione sono entrambe operazioni lineari, esse sono *associativ*e e *commutative*, quindi:

$$Lp\,[\ f() \bullet G_{auss}()\ ] = Lp\,[\ f()\ ] \bullet G_{auss}() = f() \bullet Lp\,[\ G_{auss}()\ ];$$

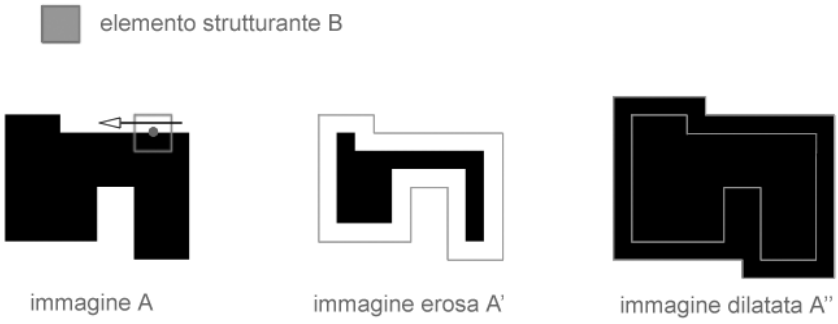
con $Lp\,[\ G_{auss}()\ ]$ denominato *operatore di Marr-Hildreth* o *operatore LoG*, qualche volta è anche chiamato *operatore a 'cappello Messicano'* per via della forma della sua maschera.
Occorre prestare attenzione nella scelta della σ della Gaussiana in quanto essa controlla il livello di dettaglio del risultato, che è in ogni caso costituito da contorni chiusi.

Un altro rivelatore in cui la σ della Gaussiana controlla il livello di dettaglio è il noto *rivelatore* (ottimo) *di contorni di Canny* (1983), il quale vi è pervenuto trattando il problema della rivelazione dei bordi come un problema tipico della teoria dei segnali, mirando cioè alla progettazione di un filtro ottimo, secondo determinati criteri ed ipotesi.

1.3.3 Elaborazione morfologica

In genere viene utilizzata dopo una delle due tecniche precedenti per ultimare l'estrazione degli 'oggetti' (un caso particolare si è visto prima, riguardo al filtraggio di immagini binarie), ma altre sue proprietà permettono, senza dubbio, di collocarla nella fase di segmentazione e/o in quella di descrizione.
La tecnica (sviluppata da Matheron e Serra per l'analisi di dati geologici a metà degli anni '60) è basata sulla teoria degli insiemi.

Le *due* trasformazioni morfologiche principali sono l'*erosione* e la *dilatazione* (duali).
Tali trasformazioni comportano l'interazione tra un *immagine* A (l'oggetto d'interesse) ed un insieme B (*elemento strutturante*).
Tipicamente B è un cerchio o un quadrato (in ogni caso una matrice di punti come nel caso delle *maschere di Sobel*) il cui centro, informalmente, viene fatto *scorrere* lungo il perimetro dell'oggetto che s'intende trasformare.
Se si effettua, di volta in volta lungo il perimetro di A, un'operazione di sottrazione (cioè la parte di B sovrapposta all'immagine A viene *cancellata* dall'immagine A), al termine si otterrà l'*immagine erosa* A'.
Viceversa se viene effettuata, lungo il perimetro, un'operazione di addizione (all'immagine A viene *aggiunta* la parte di B che non risulta sovrapposta ad A), al termine si otterrà l' *immagine dilatata* A''.
Formalmente tali operazioni si definiscono mediante la *traslazione* di A, e la *riflessione* di B.



Applicando *consecutivamente* l'operazione di *erosione* e poi di *dilatazione*, si ottiene l'operazione di *apertura* (A')'', mentre invertendo l'ordine delle due operazioni si ottiene la *chiusura* (A'')'.
Tali operazioni, intuitivamente, restituiscono un'immagine "*vicina*" a quella di partenza.
Nel *primo* caso, le parti meno spesse, che fanno da "ponte" tra quelle più spesse, nell'immagine A, vengono *cancellate*, spezzando in più parti l'immagine di partenza; nel *secondo* caso, se sono presenti delle concavità strette, queste vengono *riempite* (per esempio basta applicare l'erosione ad A'' in figura, per ottenere l'immagine di partenza *senza concavità*)
Si tralascia di illustrare l'operazione di *Hit*$_{and}$*Miss* (che rappresenta la primitiva di numerose operazioni).

Una prima applicazione di tali operazioni è quella dell'*estrazione dei bordi*, infatti *se* dallo scorrimento di B sul perimetro di A, si conserva l'*intersezione* dei due insiemi, quello che si ottiene è proprio il *contorno di* A (l'operazione inversa dell'*estrazione dei bordi* è quella del *riempimento* di regioni).
Un'altra applicazione (in questo caso delle operazioni di *apertura* e *chiusura*) consiste nell'eliminazione di piccoli grumi di pixel (più piccoli di B) distribuiti fuori dall'oggetto d'interesse e l'eliminazione di piccoli buchi interni all'oggetto.

La penultima applicazione, che si cita, è quella che consente l'estrazione dello *scheletro* dell'immagine A (modalità a *grafo* di descrizione di un oggetto).
Infatti *se* si fa la sottrazione tra A e (A')'' si ottengono gli 'spigoli', *se* poi si riapplica la sottrazione a partire dall'immagine erosa, si ottengono degli 'spigoli' più interni, che si devono conservare insieme a quelli trovati nel precedente stadio. Iterando l'operazione (e conservando quindi gli 'spigoli' ogni volta) fino a che l'immagine risulta "consumata" del tutto, si giunge allo *scheletro* di A (questo risulta uno dei modi *più rapidi* per ottenerlo).

La figura seguente mostra un esempio di *scheletro* (sulla destra) ottenuto partendo dall'immagine A (a sinistra.).

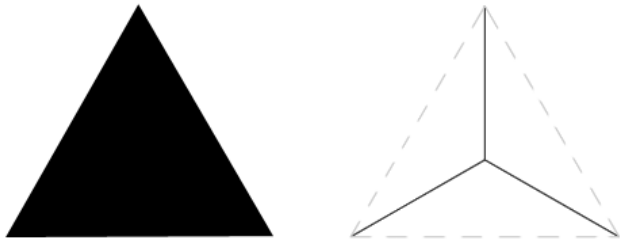


Immagine A scheletro di A

Per ultima si cita l'applicazione relativa al *thinning* di un oggetto (effettuata tramite l'operazione di *Hit*$_{and}$*Miss*), cioè l'*assottigliamento* di un oggetto sino ad ottenere una rappresentazione *curvilinea connessa* dell'oggetto di partenza (è un *altro* tipo di *scheletro* di A).
Si tralascia l'applicazione di *thickening*.

1.3.4 Flusso ottico

Tale tecnica è utilizzata in situazioni in cui si richiede l'estrazione di oggetti (che posseggono determinate caratteristiche cinetiche) da una *sequenza di immagini*.
Essa consente il calcolo del *campo di velocità* dell'immagine (un *vettore velocità* per ogni quadrato di nxn pixel). Il campo ottenuto *non* è detto rispecchi il *campo di moto* nello spazio tridimensionale (caso di movimenti illusori: asta zebrata rotante del barbiere).

Si parte da un'*ipotesi*: i punti, nell'immagine, si muovono *conservando inalterato* il loro livello di grigio.

$$I(x,y,t) = I(x-ut, y-vt, 0) \;; \qquad \text{con} \quad u=dx/dt, \; v=dy/dt \;;$$

tale ipotesi *non* è sempre verificata nella realtà, dato che l'illuminazione e le ombre cambiano muovendosi rispetto alla sorgente di illuminazione ed inoltre i punti possono scomparire per via di occlusioni; tuttavia costituisce una base di partenza.
Sviluppando con Taylor, troncando al primo ordine:

$$I(x-ut, y-vt, 0) = I(x,y,t) + \text{grad}\, I(x,y,t)^T (u,v)^T t + dI(x,y,t)/dt \; t \;;$$

da cui sostituendo l'ipotesi, si ottiene l'equazione:

$$\text{grad}\, I(x,y,t)^T (u,v)^T + dI(x,y,t)/dt = 0 \;;$$

chiamata *equazione di vincolo del gradiente*.
Si noti che vi compaiono 2 incognite u e v, ma l'equazione è solo una. Tale equazione esprime un vincolo sul primo addendo che è la proiezione del vettore velocità (dx/dt, dy/dt) sulla direzione del gradiente. Tale proiezione è l'*unica* quantità che si può ricavare dall'equazione sopra (*effetto dell'apertura*).

Il *flusso ottico* è il vettore di componenti u e v, ovvero: (dx/dt, dy/dt).

Occorre aggiungere un'altra equazione (*vincolo*) per il calcolo del flusso ottico:

(1) metodo di Horn e Schunck

Tale problema viene affrontato mediante la tecnica della *regolarizzazione* (cfr.[11]).
Occorre minimizzare un funzionale composto da 2 termini, uno che esprime il vincolo dato dal problema ed uno che introduce un vincolo di *regolarità della soluzione*, valutata mediante un operatore derivativo:

$$\min \int_W \; [\text{grad } I(x,y,t)^T (u,v)^T + dI(x,y,t)/dt] + \lambda(||\text{grad } u||^2 + ||\text{grad } v||^2)^2 \; dW \;\; ;$$

L'equazione viene discretizzata e risolta iterativamente.

(2) metodo di Lucas e Kanade

Qui si assume che il flusso ottico sia *costante* in un *intorno W* (nxn) di ciascun punto, ottenendo in tal modo più equazioni nella medesima incognita (u,v). Ciascun punto p_i appartenente a *W* dà luogo ad un'equazione lineare:

$$\text{grad } I(x,y,t)^T (u,v)^T + dI(x,y,t)/dt$$

e mettendole insieme si ha un sistema lineare: $A\,(u,v)^T = b$.
La soluzione ai *minimi quadrati* di tale sistema si calcola col metodo della pseudoinversa.
Per ulteriori dettagli cfr. link (4).

Esistono molti altri approcci per la determinazione del flusso ottico (cfr. [20]).

1.4 Descrizione

Questa fase si occupa dell'analisi delle caratteristiche degli 'oggetti' estratti, allo scopo di differenziarli e selezionarli per la successiva fase di riconoscimento.

1.4.1 Caratteristiche geometriche

Brevemente, esistono *vari* metodi per ottenere informazioni sulle caratteristiche geometriche degli oggetti.
Ad esempio *se* si ricercano *spigoli*, lungo il perimetro di un oggetto, si può procedere nel calcolo della sua *curvatura*:

$$|k(t)^2| = (d^2x/dt^2)^2 + (d^2y/dt^2)^2 ;$$

Se si applica tale calcolo ad un cerchio, ovviamente viene restituito un valore *costante* (nessuna presenza di spigoli), viceversa *se* lo si applica ad un quadrato, si ottengono quattro *picchi* (spike), che indicano la presenza di spigoli.

1.4.2 Teoria dei momenti

Tale tecnica potrebbe trovarsi tra le tecniche di riconoscimento, ma dato che consiste in un'analisi delle caratteristiche degli oggetti è ragionevole che possa *anche* trovare posto nella fase di descrizione.
La tecnica dei momenti è uno strumento per descrivere la *forma* (*area*, *centro di massa*, *orientazione*,etc.) di un oggetto.

Nel caso *continuo*, il *momento dell'immagine* è:

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x,y)\, dx\, dy ; \qquad p,q = 0,1,2,\ldots$$

e quindi il *centro di massa* corrisponde a :

$$\bar{x} = m_{10}/m_{00} ; \qquad \bar{y} = m_{01}/m_{00} ;$$

il *momento centrale* invece è:

$$\mu_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \bar{x})^p (y - \bar{y})^q f(x,y)\, dx\, dy ; \qquad p,q = 0,1,2,\ldots$$

Nel caso *discreto*, il *momento dell' immagine* è: $m_{pq} = \sum_i \sum_j i^p j^q f(i,j)$;
ed il *momento centrale* è:

$$\mu_{pq} = \sum_i \sum_j (i - \bar{x})^p (j - \bar{y})^q f(i,j) ;$$

Va detto che i momenti sono *invarianti* per *rotazione*, *traslazione* e cambiamento di *scala*.

Combinazioni opportune di momenti di diverso ordine, consentono l'estrazione di *aspetti specifici* della forma dell'oggetto. Ad esempio, se l'oggetto è di forma allungata, si può sapere l'*orientazione* θ:

$$\theta = \tfrac{1}{2} \arctan(2\mu_{11} / (\mu_{20} - \mu_{02})) ;$$

oppure il valore dei lati del *rettangolo* che contiene l'oggetto (rettangolo orientato di θ) oppure ancora l'*ellisse* che meglio di adatta (fit) all'oggetto (ellisse orientata di θ).

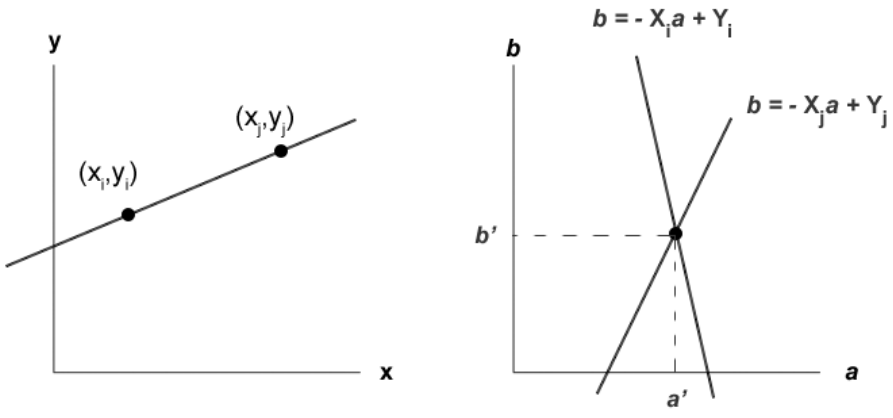
E' implicito che tali momenti possono essere applicati solo ad oggetti *singol*i, cioè separati dal resto (dopo la sogliatura o binarizzazione). Una sovrapposizione parziale, se non recuperabile con l'*erosione* (morfologica), *non* consente l'analisi corretta del singolo oggetto.

1.4.3 Trasformata di Hough

La sua utilità consiste nell'isolare determinate *forme* (dei contorni) all'interno dell'immagine e dato che richiede che tali forme siano descritte in maniera parametrica, tale tecnica è comunemente impiegata nella ricerca di curve regolari come rette, cerchi, ellissi, etc.
Esiste tuttavia una generalizzazione per curve non parametrizzabili.
Il maggior vantaggio della trasformata risiede nella sua tolleranza alla mancanza di completezza dei contorni ed alla presenza del rumore.

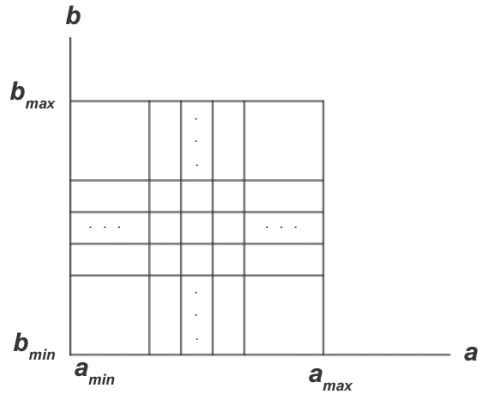
(1) ricerca di rette

Sia $y_i = ax_i + b$ l'equaz. di una retta passante per il punto (x_i,y_i). Un numero infinito di rette attraversa (x_i,y_i) ed ognuna di esse soddisfa l'equaz. della retta per diversi valori di *a* e *b*. Però scrivendo tale equaz. come $b = -ax_i + y_i$ e considerando il piano *ab* (spazio dei parametri), si ottiene l'equaz. di una singola linea per diversi valori di x_i e y_i.
Inoltre, anche un secondo punto (x_j,y_j) avrà una linea nello spazio dei parametri ad esso associata e tale linea intersecherà quella associata a (x_i,y_i) in (*a*',*b*') dove *a*' è la pendenza e *b*' l'intercetta della linea contenente sia (x_i,y_i) che (x_j,y_j) sul piano xy. I punti situati su tale linea avranno 'linee', nello spazio dei parametri, che si intersecano in (*a*',*b*').



L'attrattiva computazionale per la trasformata di Hough deriva nella

possibilità di suddivisione dello spazio dei parametri nelle cosiddette *celle accumulatrici*.



La cella accumulatrice A(i,j) corrisponde all'elemento della matrice identificata dalla coppia (*a,b*). Inizialmente tali celle sono poste a zero. Per ogni punto (x_k,y_k) sul piano dell'immagine, si pone il parametro *a* uguale ad ognuno dei valori della suddivisione consentiti sull'asse ***a*** e risolviamo per il corrispondente *b* mediante l'eq. $b = - ax_k + y_k$.

Le *b* risultanti sono poi arrotondate al valore più prossimo consentito sull'asse ***b***.

Se una scelta a dà come risultato b, allora viene incrementato di 1 il valore di A(a,b).

Al termine di tale procedimento, un valore di M nella cellula A(i,j) corrisponde ad M punti sul piano xy giacenti sulla linea $y= a_i x+ b_j$. La precisione con cui tali punti vengono assegnati ad una linea è data dal numero di suddivisioni sul piano *ab*.

Un problema, dovuto all'uso dell'equaz. $y = ax + b$ per rappresentare una retta, è che la pendenza e l'intercetta si avvicinano all'infinito quando la linea tende ad un'orientazione verticale. Un modo di aggirare ciò consiste nell'uso della rappresentazione polare di una retta $x \cos(\theta) + y \sin(\theta) = \rho$, con la differenza che ora abbiamo sinusoidi sul piano $\theta\rho$ al posto delle rette.

(2) ricerca di cerchi

Come abbiamo già detto la trasformata di Hough è applicabile a qualsiasi funzione della forma g(**x**,**c**) = 0, con **x** vettore delle coordinate e **c** vettore dei coefficenti.

Nel caso del cerchio, la cui funzione è: $(x - c_1)^2 + (y - c_2)^2 = c_3^2$, si può procedere nello stesso modo visto per le rette. La differenza principale è che ora abbiamo 3 parametri c_1, c_2 e c_3 che danno origine ad uno spazio tridimensionale (celle ubiche A(i,j,k)). Il procedimento richiede di incrementare c_1 e c_2, ricavare c_3 che soddisfa l'equaz. del cerchio e aggiornare l'accumulatore corrispondente alla cella associata alla terna (c_1,c_2,c_3).

Ovviamente la complessità computazionale cresce (molto) con il n.ro dei parametri e dei coefficenti usati.

Quando poi la forma da ricercare *non è regolare*, nel senso che non ha una rappresentazione parametrica, si fa riferimento ad una Look-Up Table (LUT).

1.5 Riconoscimento

Tra le numerose tecniche esistenti se ne citano due.

1.5.1 Intercorrelazione normalizzata

In breve, si utilizza nel caso di *misure* di somiglianza di *porzioni dell'immagine* con una *sagoma* (*template matching*) che, come nel caso delle maschere, si fa scorrere sull'immagine di partenza. La misura di somiglianza si effettua mediante somme di quadrati di differenze (tra sagoma e porzione d'immagine):

$$S(i,j) = \sum_{x,y \in T} (I(x,y) - f(i+x,j+y))^2 \ ;$$

dove S=0 *se* I( ) ed f( ) sono *localmente* uguali. Espandendo il secondo membro si ottiene:

$$\sum_{x,y \in T} (I(x,y))^2 \ - \ 2 \sum_{x,y \in T} I(x,y)\, f(i+x,j+y) + \ \sum_{x,y \in T} f(i+x,j+y)^2$$

dove il primo ed il terzo addendo si possono ritenere *costanti* ed il secondo rappresenta il doppio dell'*intercorrelazione* (notare la somiglianza con la *convoluzione*).

Può risultare più comodo adoperare però l'*intercorrelazione normalizzata* (chiamata anche *coefficiente d'intercorrelazione*):

$$s(i,j) = \left(\sum_{x,y \in T} I(x,y)\, f(i+x,j+y) \right) \Big/ \sqrt{\sum_{x,y \in T} (I(x,y))^2 \sum_{x,y \in T} f(i+x,j+y)^2} \ ;$$

dove s=1 per f( )=I( ) oppure per f( )= -I( ) mentre s=0 quando *non* ci sono somiglianze (per il calcolo *veloce* di *s(i,j)* crf. [18]).

Si aggiunge che questo non è l'unico modo di utilizzare l'intercorrelazione al fine di confrontare due oggetti.

1.5.2 Reti neurali

Tale tecnica viene trattata nel capitolo seguente.

1.6 Interpretazione

L'interpretazione, dopo la fase di riconoscimento, è direttamente connessa con gli *scopi* per cui il sistema è stato progettato.
Ad esempio, in casi *semplici*, tale fase si esplica nel seguente modo:
Se si verifica un certo riconoscimento *allora* a ciò deve corrispondere una determinata azione di controllo (retroazione, etc).

Nei casi più *complessi* si deve far ricorso a tecniche di *rappresentazione della conoscenza*, *modelli semantici*, *schemi d'inferenza* (tecniche d'intelligenza artificiale) oppure ancora a sistemi a *rete neurale* o a tecniche *fuzzy*, affinchè il sistema sia i grado di decidere su *come* intraprendere l'azione di controllo.

Per ulteriori dettagli ed esempi di applicazioni di Visione Artificiale si può fare riferimento al testo [6] e/o al sito (3).

2 Reti neurali per il riconoscimento

Prima di passare ad introdurre i modelli di *reti neurali* e ad approfondire il modello di *Hopfield*, si descriverà sinteticamente la struttura biologica in cui esse sono inserite, nel dare origine al *sistema di visione umano*. Per maggiori dettagli si rimanda a [7].

2.1 cenni di fisiologia della percezione umana

Molte delle funzioni esaminate nel primo capitolo sono effettuate anche dal *cervello umano*, con la principale differenza che esse vengono eseguite da *strati* di reti di neuroni (disposti più o meno in maniera gerarchica),
strati che *filtrano*, strati che *elaborano*, strati che *memorizzano* e *riconoscono*.

Alla luce delle conoscenze attuali, prendendo come esempio la *via parvocellulare interblob*, ai primi livelli di percezione (*retina* e *cgl (corpo genicolato laterale)*) la connettività locale tra i neuroni (disposizione retinotopica) dà origine ad operazioni di *filtraggio* (neuroni con campi recettivi *concentric*i, andamento a cappello messicano) dell'immagine percepita dai *fotorecettori*; *filtraggio* che serve principalmente a *contrastare* l'immagine ed a *rivelarne i bordi*.
(Per *disposizione retinotopica di un certo strato*, s'intende che due punti "vicini" nella retina, si mantengono "vicini" anche nello strato in questione.)

Strati successivi (*Area V1*) (neuroni con campi recettivi a *direzioni preferenziali*) operano un filtraggio di caratteristiche legate alle direzioni locali dei bordi, rivelati prima.

Strati ulteriori (*Area V2,V4, inferotemporale*, *etc.*) (disposizione, in genere, *meno* retinotopica e presumibilmente con connettività a più ampio raggio) utilizzano maggiormente l'attivazione *parallela* dei neuroni (in uno stesso

strato) e quindi sfruttano la proprietà delle reti di *memorizzare* e/o *elaborare* le informazioni. Gli strati profondi (ad esempio LOa (praticamente non retinotopica)) sono poi specializzati nell'occuparsi di forme particolari quali *volti umani* e *scene*.
Tale *via*, come si sarà già capito, si occupa prevalentemente del riconoscimento delle *forme*. Riassumendo si ha il seguente schema:

retina **->** *cgl* **->** *V1* **->** *V2* **->** *V4*/*V8* **->** *LO* **->** *Loa / FFA / PPA …*

Esiste comunque almeno un altro *percorso parallelo* di elaborazione delle immagini percepite (*via magnocellulare*). Quella accennata prima è una *diramazione* della *via parvicellulare* (l'altra *diramazione*, la quale si occupa principalmente delle componenti cromatiche percepite dalla *retina*, converge anch'essa nella zona *inferotemporale*).

Ripartendo dalla *retina*, esistono in essa anche neuroni sensibili al *movimento* (*aumentano* la frequenza della loro scarica quando il loro ingresso *cambia*) ed essi innervano negli strati successivi secondo lo schema seguente (che rappresenta *parte* della *via magnocellulare*):

retina **->** *cgl* **->** *V1* **->** *MT (Area V5)* **->** *MST (Area V5a)* **->** *PP / IPS …*

Tale percorso dell'elaborazione consente: la percezione del movimento (*retina*,*cgl*), la discriminazione della velocità e direzione locale (anche *globale*, per mezzo di una minoranza di neuroni) (*MT*) e l'integrazione dei dati per funzioni che genericamente possono essere riassunte in: percezione visiva dei movimenti globali, mantenimento dei movimenti di inseguimento degli occhi e guida dei movimenti dell'individuo nell'ambiente (*MST, e suuccessivi*).
Più precisamente *MST* è implicata direttamente nella valutazione del *flusso ottico* e del *movimento non lineare* nell'immagine percepita (utili per lo *shape from motion*).
Per un'applicazione, derivata direttamente da tale *via*, si rimanda ad [8]

(Fujitsu Laboratories); riguardo ad uno studio sulle capacità di calcolo basate sull'area *MT*, si rimanda a [9] e per ciò che riguarda un'approfondimento circa la tematica della percezione del movimento si rimanda a [13].

L'esistenza anche di neuroni selettivi alla *disparità visiva*, nella *via magnocellulare*, consente l'estrazione di informazioni di carattere *tridimensionale* dalle 2 immagini percepite dagli occhi (leggermente differenti tra di loro), cioè si sfrutta la *visione stereoscopica* (cfr. [19]).
Lo schema precedente, a tale scopo, va integrato delle aree *V2/V3*, dopo *V1*.

Si tralasciano altri aspetti elaborati da tale *via* (tipo la *profondità di campo, dinamica,etc.*) e da quella *parvicellulare* (*texture, etc.).*

Le elaborazioni, portate avanti su *vie* diverse, si possono incrociare, cioè elaborazioni di una *via* possono *cooperare* con elaborazioni di un'altra *via* (ad esempio per la *fusione sensoriale*) ed inoltre uno strato più profondo, in alcuni casi, influenza uno strato più superficiale; quindi un certo strato ha la possibilità di essere *modulato*, nelle sue funzioni, da altre 'zone' del cervello, in anelli a retroazione non ancora completamente noti.

Per ciò che riguarda la *realizzazione microelettronica* di reti neurali si possono citare:

Reti a connettività locale,

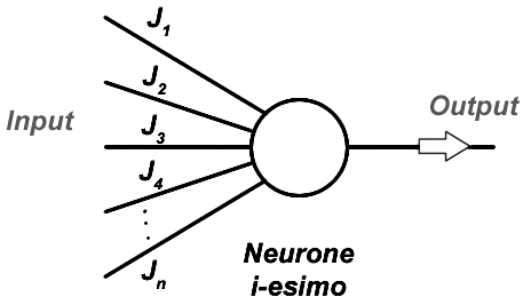
ad esempio le CNN (cellular neural network), che si prestano bene per i primi stadi dell'elaborazione dell'immagine, *retina*, etc.) (cfr.[10]),

Reti a connettività estesa,

ad esempio le ANN (attractor neural network), adatte ad imitare gli strati più profondi (cfr.[3]).

2.2 Modelli di reti neurali

Una rete neurale è un sistema costituito da un insieme di unità (neuroni) collegate reciprocamente, ciascuna con *un*'uscita e *più* ingressi.



Ad ogni interconnessione fra tali unità (eventualmente connesse anche con l'esterno) viene associato un numero J_i ('peso sinaptico' o soltanto 'peso'), che esprime quantitativamente il segnale che "viene lasciato passare" nel percorso verso l'unità successiva.
I neuroni impiegati nei vari modelli si differenziano per la legge adottata nel determinare l'uscita a partire dal valore degli ingressi. Ad esempio si citano i *neuroni binari con soglia*, dove l'uscita $X_i(t+1)$ vale:

$$X_i(t+1) = f\,(P(t) - \sigma); \quad \text{dove:} \quad f(x) = (\mathbf{1} \ \text{per } x \geq 0 \ , \ \mathbf{0} \ \text{per } x < 0) \ ;$$
$$P(t) = \sum_j J_j X_j(t) \ ; \qquad \sigma \equiv \text{soglia},$$

si potrebbero citare neuroni con uscita *sigmoidale* oppure neuroni ad *attivazione probabilistica*, etc.
La struttura delle interconnessioni e la modalità di assegnazione dei relativi pesi ha un importanza cruciale nel *comportamento* del sistema. Per dare solo un'idea, *reti a connessione completa* (ogni neurone è collegato a ciascun altro) esibiscono principalmente comportamenti di *memorizzazione*, mentre *reti a multistrato* (ogni neurone di uno strato è collegato *solo* a neuroni dello strato successivo) esibiscono principalmente comportamenti di

classificazione, anche se i due tipi di comportamento possono *sfumare* uno nell'altro.

Passiamo ora ad esaminare in dettaglio il tipo di rete di cui si è poi fatto uso nell'applicazione di riconoscimento di segnaletica.

2.2.1 Rete neurale di Hopfield

Tale modello, introdotto da J.J.Hopfield nel 1982 (cfr.[1]), sulla base anche di ricerche precedenti, ha contribuito alla rinascita dell'interesse per le reti neurali ed anche se non possiede delle ottime prestazioni per quanto concerne la memorizzazione dei pattern, si è dimostrato *sufficentemente rapido* nella programmazione e nell'esecuzione in tempo reale.
Dall' '82 ad oggi sono state proposte numerose generalizzazioni (più onerose computazionalmente) che ne hanno migliorato le caratteristiche (cfr.cap.4 di [2]).

Tale rete a connessione completa con neuroni binari a soglia, per le sue modalità di funzionamento, costituisce un sistema dinamico e in quanto tale ammette una funzione energia (più precisamente di Lyapunov). La legge di *evoluzione* è tale da dare una funzione energia monotona *non crescente*: ovvero il sistema *evolve* verso un minimo locale dell'energia (*stato localmente stabile*, che corrisponderà ad uno dei pattern che avremo precedentemente memorizzato).

Da un altro punto di vista, se si assume che la funzione energia possa essere associata ad una *funzione costo*, di un certo problema di ottimizzazione, l'evoluzione della rete tende a ricercare rapidamente lo stato (stati) che minimizza(no) tale funzione, risolvendo tale problema.

L'analogia di tale modello di rete con i *vetri di spin* (materiali che manifestano contemporaneamente comportamenti ferromagnetici ed antiferromagnetici)

ha fatto si che si pervenisse anche ad una meccanica statistica di tale tipo di rete e della sua versione analogica che si presta ad una realizzazione microelettronica (cfr.[3]).

Sia V il vettore di stato (N componenti (neuroni) di 0 o 1), si definisce *matrice dei pesi* T la matrice composta di numeri interi relativi, dove il singolo componente Tij è detto *peso sinaptico* relativo al neurone i e a quello j.
Si definisce I il vettore di ingresso (N componenti di numeri interi relativi) e *passo* un numero intero. Il vettore V e quello I sono funzioni del passo k: V = V(k) e I = I(k).

Fissato un valore V(0) (condizione iniziale), l'evoluzione del vettore V è data da:

(*)

$$V_i(k+1) = \begin{cases} 0 & \text{se } I_i(k) =< 0 \\ 1 & \text{se } I_i(k) > 0 \end{cases}$$

dove:

$$I_i(k) = \sum_{j=1}^{i-1} T_{ij}\ V_j(k+1) \ + \sum_{j=i+1}^{N} T_{ij}\ V_j(k)$$

Tale legge di evoluzione è detta *rete di Hopfield discretizzata con aggiornamento asincrono sequenziale*. L'aggiornamento può essere *sincrono*, ovvero tutti i neuroni evolvono allo stesso istante, che tradotto in formule significa che nel primo addendo, della formula sopra, invece di Vj(k+1) si ha solo Vj(k).
Però tale aggiornamento (*sincrono*) permette un rilassamento della dinamica non solo verso punti fissi bensì anche verso cicli limite, il che per questo tipo di applicazione risulta indesiderato. Invece un aggiornamento *asincrono* garantisce l'esistenza *solo* di punti fissi. Ciò significa che ogni neurone evolve indipendentemente dagli altri.
Il tipo di evoluzione può essere *sequenziale* o *casuale*. Qui si è scelto la *sequenziale*, cioè in un certo ordine prefissato, per maggior semplicità nella

realizzazione.

Tale tipo di memoria viene a volte designata come CAM (Contents Addressable Memory) cioè memoria che permette di raggiungere un vettore memorizzato se ne viene posto in ingresso una versione *incompleta* o *danneggiata* del medesimo vettore.

La *memorizzazione* dei pattern nei pesi sinaptici della rete avviene secondo la *legge di Hebb* (codificata alla maniera di Hopfield) (cfr.[17]).
Siano $V^{(1)}, V^{(2)},\ldots, V^{(n)}$ i vettori (pattern) da memorizzare.
Al generico Tij assegnamo allora il valore (con soglia *nulla*, altrimenti anche il vettore nullo diventa un punto fisso):

$$T_{ij} = \sum_{s=1}^{n} (2V_i^{(s)} - 1)(2V_j^{(s)} - 1) \qquad \text{per } i \neq j ;$$

Da tale formula si evince immediatamente la proprietà di *simmetria* della *matrice dei pesi*.

Ora se sostituiamo l'ultima espressione nella penultima, si può ottenere la proprietà che il generico elemento risulta *stabile* purchè il numero di 0 e 1, nei vettori memorizzati, sia circa lo stesso. Infatti supponendo che $V(0)=V^{(s)}$:

$$I_i = \sum_{\substack{j=1 \\ j\neq i}}^{N} T_{ij}\, V_j(0) = \sum_{\substack{j=1 \\ j\neq i}}^{N} T_{ij}\, V_j^{(s)} = \sum_{\substack{j=1 \\ j\neq i}}^{N} \sum_{s'=1}^{n} (2V_i^{(s')} - 1)(2V_j^{(s')} - 1)V_j^{(s)} =$$

$$= \sum_{\substack{s'=1 \\ s'\neq s}}^{n} (2V_i^{(s')} - 1) \sum_{\substack{j=1 \\ j\neq i}}^{N} (2V_j^{(s')} - 1)V_j^{(s)} + (2V_i^{(s)} - 1) \sum_{\substack{j=1 \\ j\neq i}}^{N} (2V_j^{(s)} - 1)V_j^{(s)}$$

Si nota che il secondo addendo dell'ultimo termine ha in media lo stesso segno di $V_i^{(s)}$ e che il primo addendo ha un valore medio che può considerarsi nullo purchè il numero di 0 e di 1 nei vettori memorizzati sia uguale. Allora

dalla (*) si deduce che gli elementi memorizzati sono stabili se è soddisfatta la condizione indicata.

Per vedere come la regola di Hebb implichi un evoluzione del sistema verso punti asintoticamente stabili, basta costruire una funzione energia del tipo:

$$E = -1/2 \sum_{i,j} T_{ij} V_i V_j$$

Si nota che essendo le Tij limitate, per definizione la E è limitata, inoltre per ogni variazione del vettore V(k) la E diminuisce sempre, fino a raggiungere un minimo locale. Infatti, dopo ogni variazione di V(k) è:

$$\Delta E = -\Delta V_i \sum_{j \neq i} T_{ij} V_j$$

e si osserva che, *se* $\sum$ TijVj > 0 è ΔVi > 0 per cui ΔE < 0 mentre *se* $\sum$ TijVj < 0 è ΔVi < 0 e ancora ΔE< 0.
Quindi V(k) converge sempre verso un vettore asintoticamente stabile, il problema è che non sempre si tratta di quello giusto.

Ogni vettore memorizzato ha un suo *dominio* (o *bacino*) *di attrazione*, pertanto se la condizione iniziale, è vicina (informalmente) ad un vettore memorizzato allora l'evoluzione del sistema convergerà verso quest'ultimo. Quando una condizione iniziale condivide in egual modo parti che si avvicinano ad un vettore memorizzato e parti che si avvicinano ad un altro vettore memorizzato, il fatto che il sistema evolva verso l'uno o l'altro vettore dipende dall'aggiornamento; cioè se il neurone successivamente aggiornato permetterà un avvicinamento ad un vettore (ed un corrispondente allontanamento dall'altro), l'evoluzione convergerà verso il vettore più vicino.

Va detto poi che oltre ai vettori memorizzati (o pattern) sono presenti anche gli *stati spurii* e gli *stati invertiti* rispetto a quelli memorizzati (gli zero al posto degli uno e viceversa). Per esempio se insegnamo 2 pattern alla rete ci ritroviamo, oltre a questi ultimi, anche gli stati invertiti come attrattori e anche

gli stati risultanti dall'AND (logico) dei primi quattro (se hanno degli '1' in comune) (cfr.cap.4 di [2]).

La rete non fa confusione se il numero dei vettori memorizzati non supera circa 0.1 N, con N il numero dei neuroni. Va detto che questo è un limite *teorico*, in realtà la capacità di memoria è ancora inferiore.

3 Hardware e Software

E' stato impiegato il seguente hardware e software:

- PC Pentium4 2,4 GHz, (256Kb cache L2) 1Gb di Ram
- webcam Logitech QuickCam Pro 4000 (CCD)

- Microsoft Visual C++ 6.0 (ambiente Windows 2000 Pro / Windows XP)
- librerie MFC SDK Logitech (webcam).
  (su Internet tali librerie si trovano anche per l'ambiente *Linux*)

MFC è l'acronimo di *Microsoft Foundation Classes* mentre SDK sta per *Software Development Kit*.
Le MFC sono le *classi* che permettono, tra le altre cose, l'utilizzo delle *finestre* tipiche dell'ambiente Windows, mentre le SDK sono le librerie (per il *Visual* C++) relative alla *webcam*, che contengono i *metodi*, le *proprietà* e le *notifiche* necessarie per lo scambio dati tra il PC e la webcam.

La webcam è stata impostata sulla modalità 320x240 pixel a *toni di grigio* a *30 fps* (*frame per second*). Questa acquisisce immagini nel *formato YUV420P* che poi vengono depositate in memoria (3 byte per pixel = 16 milioni di colori).
Tale *formato* ha un canale per la luminosità (*Y*) e due canali per i colori (*U*,*V*), con i quali si rappresentano tutti i colori possibili; la sigla *420P* si legge *4*:*2*:*0 planare*, ed indica che nella codifica digitale dell'immagine, inizialmente sono scritti tutti i valori *Y*, (320x240 byte nel caso in esame), seguiti dai valori delle *U*, in numero pari ad 1/4 dei pixel *Y*, (cioé 320x240/4), ed infine dalla stessa quantità (320x240/4) di valori di *V*. Quindi, ogni 4 valori di *Y* vi è un valore per la *U* ed uno per la *V*, ovvero ogni valore di *U* e *V* si applica ad un blocco di 2x2 valori di *Y*.

Le applicazioni presentate, per funzionare in *tempo reale*, si avvalgono del *metodo* di videoHooking, relativo alle librerie SDK della webcam:

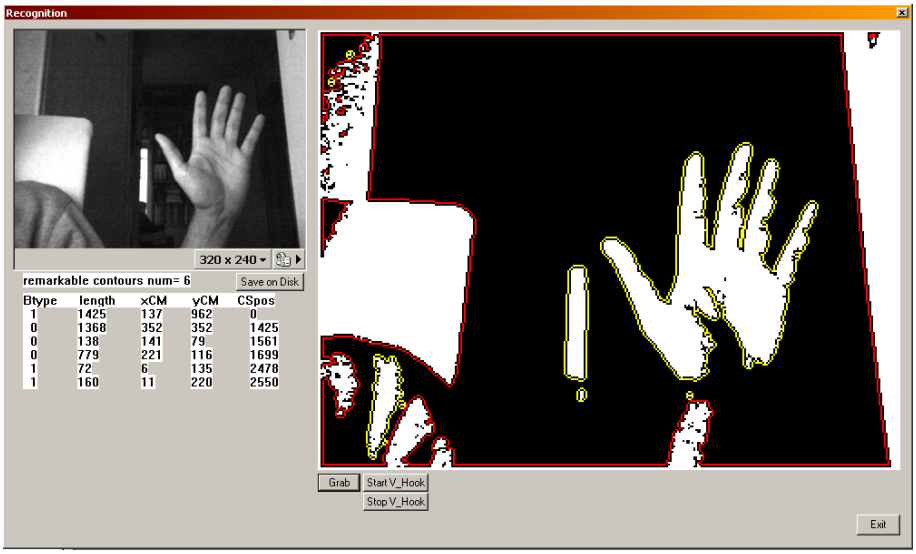
Ogni *trentesimo di secondo* (alla risoluzione di 320x240 pixel), viene chiamata dal sistema operativo, una *funzione*, predisposta all'elaborazione dei *dati video,* resi disponibili dalla webcam in tale periodo temporale; la *funzione* di elaborazione, dev'essere ultimata, ovviamente, prima della chiamata successiva, pena un sovraccarico per la CPU ed un *rallentamento* di tutto il sistema.

Si menziona inoltre *un problema* (aggirato) con le SDK di questa famiglia di webcam: Viene fornita, dalla casa, una libreria che risulta, sotto il VC++, con il nome di VideoPortal Control (tra i controlli Active-X registrati), quest'ultima non sembra funzionare bene (può anche risultare assente!), pertanto si può utilizzare alternativamente il controllo Hydra VideoPortal (che sembra poi essere proprio quello utilizzato dalle Applicazioni eseguibili fornite con l'installazione), che difetta solo delle *proprietà* relative ai testi sovrapponibili alle acquisizioni (ovviamente per i nostri scopi è ininfluente).

Riguardo alla struttura dati delle *dib* (bitmap) acquisite dalle webcam della Logitech, si fa riferimento al Tutorial T.R.I.P.O.D., liberamente fruibile da Internet, presso il link (2),
(si ringrazia il Prof. Paul Y. Oh per le informazioni riportate sul suo Tutorial).

Per entrambe le applicazioni la *finestra di esecuzione* è la medesima ed è qui denominata *finestra Recognition*. Come si può vedere nella figura seguente, tale finestra è suddivisa in *due* zone, sulla sinistra si ha la visualizzazione di ciò che la webcam rileva all'esterno (*riquadro di input*) e sulla destra si ha la visualizzazione dell'*elaborazione finale* (*riquadro di elaborazione*) di ciò che è stato ovviamente rilevato sulla sinistra. Sotto al *riquadro di input,* si può notare un tasto, la cui funzione è quella di *salvare su disco* le immagini che si presentano sopra, ed una tabella di dati relativa ai *blob* trovati nel *riquadro di elaborazione*. Tali dati (*tipo*, *perimetro* in pixel, *coordinate* del centro di massa del contorno e *posizione* nell'array) sono relativi soltanto alla *prima* applicazione.

Sotto al *riquadro di elaborazione*, sono riportati *tre* tasti di cui *due* incolonnati.
Il tasto sulla sinistra (*Grab*) esegue il funzionamento, di una delle due applicazioni, *soltanto quando viene premuto*, ovvero consente l'uso dell'applicazione soltanto nell'istante desiderato dall'utente.
Gli altri due tasti *abilitano* e *disabilitano* il funzionamento *in tempo reale* dell'applicazione, cioè il funzionamento continuativo a 30 *fps*.
Il tasto in basso a destra (*Exit*) consente l'uscita dal programma.



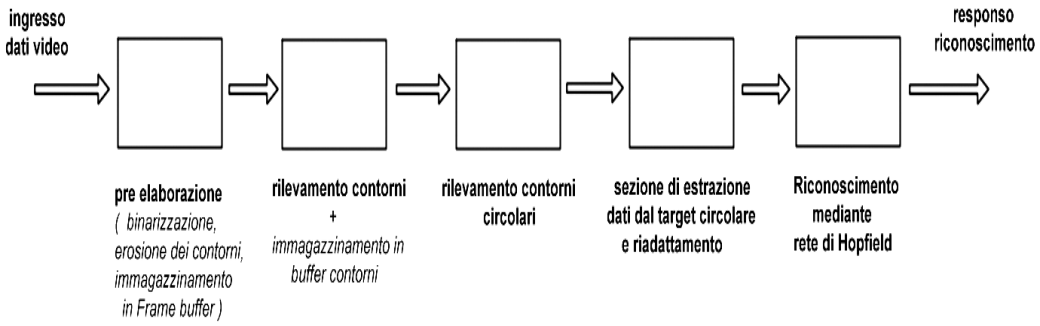
Si vuole sottolineare che nella figura sopra, che mostra una schermata della prima applicazione (Cap.4), nel momento in cui sono presenti target circolari (non ve ne sono nella figura sopra) il programma tenta di *leggere* al loro interno (paragr. 4.3), pertanto nel *riquadro di elaborazione* vengono sovrapposti due quadrati con bordo verde, in basso e a destra, che rappresentano lo stato *iniziale* e *finale* della rete neurale.

Si sarà inoltre notato che nel *riquadro di elaborazione* sono presenti contorni gialli e rossi; ciò serve per differenziare i *blob neri* da quelli *bianchi*.

4 Acquisizione di informazioni da target circolare

Come già accennato nell'introduzione tale sistema si occupa del riconoscimento di informazioni rappresentate all'interno di target (riferimenti o bersagli) circolari.

Il contorno circolare del target ha la *duplice* funzione di "richiamare l'attenzione" del sistema su tale oggetto e contemporaneamente di restituire un riferimento geometrico per la "lettura" dell'informazione ivi contenuta, mediante il successivo sistema di riconoscimento a rete neurale.
Lo schema a blocchi del sistema è il seguente:



L'intero sistema è riferito al modulo Recognition() nel listato: *Image.cpp*.
Nel prossimo paragrafo si presentano le prime *due* fasi del sistema.

4.1 Ricerca e memorizzazione di contorno per immagini binarizzate

Lo stadio di pre-elaborazione, si occupa della *binarizzazione* (in base ad una certa soglia) e del *filtraggio ad erosione semplice* (basato sul conteggio dei pixel adiacenti) eseguita per ogni pixel, con lo scopo di eliminare i punti singoli (bianchi e neri) e di 'limare' il bordo delle zone più frastagliate. Le immagini si presenteranno ovviamente come zone bianche e nere (*blob*) che ritrarranno in modo "semplificato" ciò che è stato acquisito dalla webcam in un certo istante.

esempio di *binarizzazione* (senza *erosione*)

L'obiettivo, in questa fase, è *ricercare* e *memorizzare* i *contorni* dei *blob* (ovvero delle macchie nere di forma indefinita).

Ogni *blob* risulta *separato* da un altro *blob*, *se* (informalmente) esiste almeno un pixel (bianco) che li divide.

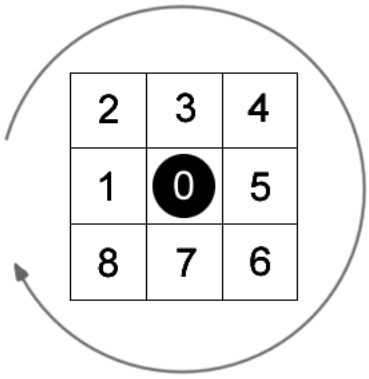
L'algoritmo che effettua la ricerca e la memorizzazione dei contorni è una variante dell'*etichettamento delle componenti* (component labeling, ovvero il

metodo che consente la segmentazione di immagini binarie in insiemi connessi).

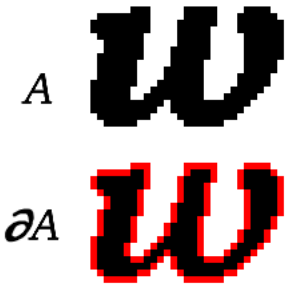
S'inizia la ricerca dei contorni effettuando una scansione *pixel per pixel* dell'intero quadro video (dal basso verso l'alto e da sx a dx).

Quando si trova un pixel nero (non marcato in rosso) si procede ad una *scansione (locale)* dei suoi pixel adiacenti (in senso *antiorario*) alla ricerca di un altro pixel nero; appena trovato si *marca in rosso* e si memorizzano le coordinate di quello precedente, quindi si ricomincia daccapo l'operazione (della *scansione locale*) sul nuovo pixel di bordo trovato e si procede in tal modo sin quando non si ritorna al pixel di partenza.

Le immagini seguenti sintetizzano la *scansione locale* e l'effetto su di un *blob* (lettera w):



schema dell'**intorno** del punto d'interesse.



contorno ottenuto dall'algoritmo.

C'è da precisare che l'algoritmo di *scansione locale* si basa sull'*ipotesi* di conoscere la posizione (nell'intorno) del precedente pixel di bordo trovato (quello marcato in rosso); ovviamente però all'inizio tale informazione non è disponibile pertanto si fa un controllo di *provenienza* (nella scansione globale), cioè se si era in una zona bianca e si passa ad esaminare il bordo di una zona nera si assume che il pixel rosso sia il n.ro 8 altrimenti se si era in una zona nera e si passa in una bianca (per esempio un buco bianco in un blob) si assume che il pixel rosso sia il n.ro 4.

Terminata la scansione del quadro video, si è in posseso delle coordinate del contorno (e del *centro di massa del contorno*) di *ogni blob*, memorizzate in

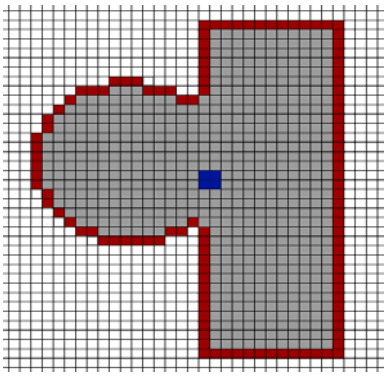
array ed *ordinate* nella direzione da cui è iniziata la *scansione locale*, disponibili per le successive operazioni di descrizione.

I contorni memorizzati sono solo quelli che hanno un numero di pixel superiore ad una certa soglia, allo scopo di memorizzare solo le sagome di grandezza rilevante.

4.2 Rivelazione di contorni circolari

In fase di progetto si è valutata inizialmente la possibilità d'impiego della *trasformata di Hough* (paragr. 1.4.3) per la rivelazione dei contorni circolari; possibilità che è stata scartata per l'elevato onere di calcolo richiesto da tale metodo, nella realizzazione di un sistema di visione *in tempo reale*. Ci si è orientati, pertanto, nell'analisi delle proprietà geometriche dei contorni.

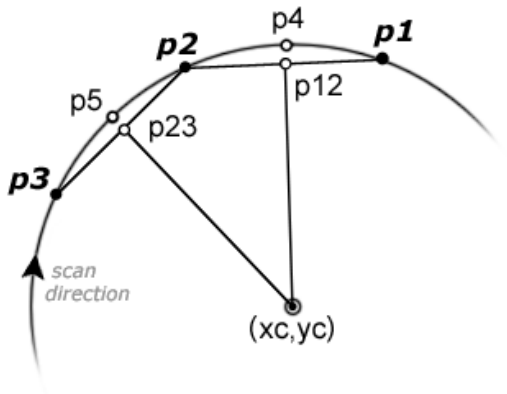
Per il rilevamento di un *target circolare* si è partiti inizialmente facendo uso semplicemente della conoscenza del *centro di massa del contorno*, resa disponibile, per ogni *blob*, dall'algoritmo precedente.
Infatti, noto il centro di massa del contorno, si calcola, per ogni punto di ogni *contorno*, la *distanza euclidea* (al *quadrato* per *evitare* l'operazione di radice quadrata, onerosa computazionalmente) *tra* tale punto e il centro di massa del contorno, verificando che tale distanza resti *costante*, entro un margine di tolleranza, per tutti i punti del contorno considerato.
Nonostante la semplicità e la rapidità, tale metodo soffre del fatto che le circonferenze devono essere *isolate* dal contesto e dai bordi del quadro video, affinchè il centro di massa del contorno risulti *anche* il centro della circonferenza. Una sovrapposizione *parziale* di un altro oggetto (occlusione) fa si che si abbia un contorno (rosso) solo in parte circolare ed un centro di massa non corrispondente col centro della circonferenza.



Il codice di tale metodo è comunque menzionato nel sorgente, ma *disattivato* in forma di *commento*.

Si è proceduto quindi a considerare *un altro* metodo intuitivo, cioè quello di effettuare un *recupero* del *centro di massa di un contorno circolare* mediante *considerazioni geometriche* applicate ad ogni contorno.

Infatti dato che per 3 punti sul piano passa una sola circonferenza, si procede alla scansione di un contorno prendendo 3 punti *equidistanziati* p1, p2 e p3 (in termini di distanza curvilinea in pixel) e si calcola il centro di massa come se ci si trovasse su una circonferenza.



p12 e **p23** sono punti utilizzati dalla sezione di calcolo di (xc,yc).

Si effettua una *prima verifica di massima* su 2 punti intermedi (p4 e p5, oltre che sugli altri 3) calcolando la distanza di ognuno con il centro calcolato. *Se* si ha esito positivo si procede ad effettuare una *verifica precisa* su ogni punto del contorno contenuto tra p1 e p3.
Se anche questa ha esito positivo allora si prosegue *oltre* il punto p1, ad effettuare la distanza di ogni pixel del contorno e a verificare *se* la distanza dal centro si mantiene costante (sempre entro un margine di tolleranza).

Quando la distanza dal centro non viene più verificata si passa ad esaminare i punti *prima* del punto p3 e quando si esce si verifica *se* il numero di pixel, che hanno superato i test, risultano maggiori di una costante (1,1 ad esempio) *moltiplicata* per il semiperimetro della circonferenza con tale raggio (cioè in totale poco più di mezza circonferenza).

Se quest'ultimo test è verificato *allora* la circonferenza viene *riconosciuta* (e nella progettazione si tracciava un segno "+" nel punto (xc,yc)).
Ogni volta che uno qualsiasi dei test *fallisce* i punti p1, p2 e p3 vengono spostati avanti (lungo il contorno) di un certo numero di pixel (5 in questo caso) e si ricomincia daccapo.

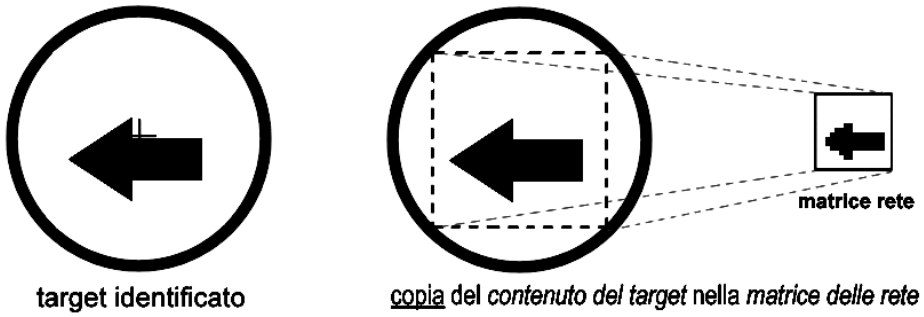
Questo metodo non possiede gli svantaggi del precedente e mantiene un *ottima* prestazione in termini di velocità (dal Task Manager di Windows *non si rileva un appesantimento percepibile della CPU* rispetto all'assenza di questa sezione di programma).
Inoltre agendo sui valori di tolleranza si riesce a riconoscere circonferenze affette da *lievi* distorsioni, dovute a vari motivi d'illuminazione ambientale o ad effetti prospettici.

Per quanto riguarda i semplici calcoli effettuati per la determinazione di (xc,yc) si rimanda all'Appendice A.

4.3 Riconoscimento mediante rete neurale di Hopfield

Disponendo del *centro* e del *raggio* del target, si procede ad un'operazione di *spostamento e riadattamento* della porzione di memoria video, contenuta nel target, nella matrice (16x16) della rete neurale. Tale configurazione della rete, costituisce lo *stato iniziale* per la legge di evoluzione e viene visualizzata, nel *riquadro di elaborazione*, in un quadrato (di bordo verde) in basso a destra, come si vedrà dopo.

Nel sorgente, il modulo che esegue l'operazione di *spostamento e riadattamento*, è denominato 'rescaler'.



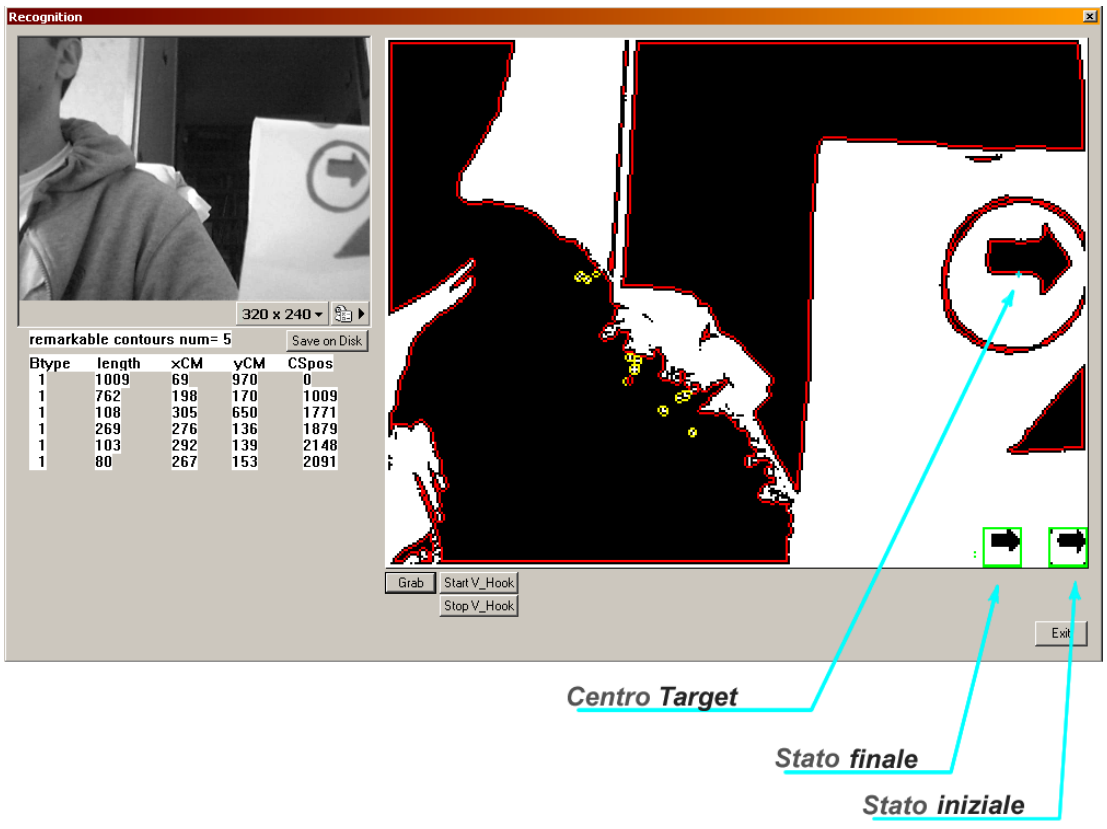
L'ultimo passo consiste nel lasciare *evolvere* la rete.

I *pattern* memorizzati nella rete sono 3 (incluso quello rappresentato sopra) per via dello scopo solo *dimostrativo* di tale applicazione e sono citati all'inizio del listato *RecognitionDlg.cpp*, dove sono posti in forma matriciale e sono quindi di *immediata* modifica da parte dell'utente.

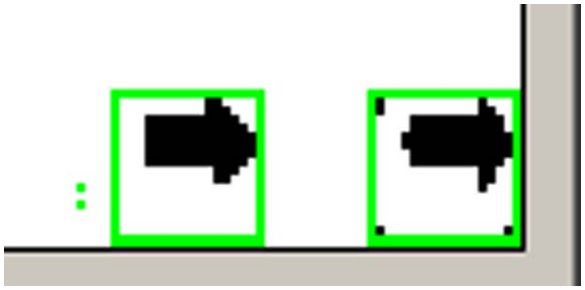
La *memorizzazione* dei pattern nella rete avviene all'*inizio* dell'esecuzione del programma.
L'*aggiornamento* della rete avviene per *cicli*, nel senso che in un *ciclo* si procede all'aggiornamento *asincrono sequenziale* di tutti i 256 neuroni, secondo le modalità viste nel paragrafo 2.2.

Nel *riquadro di elaborazione*, alla sinistra del quadrato (con bordo verde) dello *stato iniziale*, vi è il quadrato (sempre con bordo verde) relativo allo *stato finale* raggiunto dalla rete ed alla sua sinistra sono visualizzati, in verticale, dei punti verdi che rappresentano il numero dei *cicli* che sono stati necessari per giungere alla convergenza della rete.

Lo stato finale della rete è *confrontato* con tutti i pattern memorizzati e *se* tale confronto dà esito *positivo* allora nel quadrato dello *stato finale* viene visualizzato il *pattern riconosciuto*, *altrimenti* viene visualizzato un punto interrogativo, a significare che la rete è *caduta* in uno stato *spurio* o in uno stato *invertito*.



Un target è stato *disegnato* su un foglio di carta (con un pennarello) e *sottoposto* quindi alla webcam; nel *riquadro di elaborazione* si possono notare i due quadrati, visualizzati dall'applicazione, che contengono i due stati delle rete neurale e sotto è riportato un ingrandimento.



Si conclude mettendo in evidenza che la *legge di evoluzione della rete* (listato *Image.cpp*) è stata realizzata *senza eseguire le moltiplicazioni* di cui formalmente si fa uso per definirla (cfr. paragrafo 2.2), infatti la struttura dell'algoritmo ha consentito di far uso *solo* di *addizioni* ed *espressioni condizionali*. Ciò ha permesso un risparmio in termini di *tempi di esecuzione*, dal momento che, com'è noto, l'operazione di *moltiplicazione* a basso livello (in questo caso *assembly x86*) è molto più onerosa di quella di *addizione* e di *confronto*.
Tale condizione, unita alla semplicità della modalità di memorizzazione dei pattern, è stata determinante per la scelta del tipo di rete da utilizzare per quest'applicazione *in tempo reale*, dove tutte le funzioni suddette devono essere *eseguite 30 volte al secondo*.

Nel caso in cui l'applicazione necessiti di un consistente numero di pattern, è consigliabile utilizzare due (o più) matrici sinaptiche (quindi due o più *reti*), ognuna delle quali abbia un *basso* numero di pattern memorizzati, al fine di *evitare* il sovraccarico di una singola rete e quindi di *non* giungere ad un peggioramento della sua capacità di convergenza (maggiore numero di pattern memorizzati, minore grandezza del dominio di attrazione per ciascun pattern).

5 Rivelazione del movimento

La seconda applicazione realizzata, si occupa della *rivelazione* di soggetti (oggetti) *in movimento*. Come già anticipato, tale rivelazione è, in genere, alla base di sistemi per videosorveglianza.
L'applicazione *visualizza* le *sagome* dei soggetti in movimento e restituisce quindi la loro *posizione spaziale*, all'interno del quadro video.
Tali sagome sono volutamente *grossolan*e (bassa quantizzazione spaziale, ad esempio 64x48 pixel, cfr. paragr. 5.3) per tentare di ovviare al *problema* che si presenta nel caso in cui sia lo *sfondo* (dinanzi al quale si ha il movimento) che alcune porzioni dell'immagine del soggetto (in movimento) *abbiano il medesimo* dettaglio superficiale (texture), ovvero abbiano le medesime intensità di grigio (cfr. proprietà **(ii)** paragr. 5.1).
Tale problema *ostacola* la corretta rivelazione del contorno del soggetto in movimento.

L'intero sistema è riferito al modulo MotionDetect() nel listato: *Image.cpp*.

Va prestata attenzione nell'utilizzo di quest'applicazione, nel caso di *robot mobili autonomi*.
Infatti si ricorda che la webcam risulta solidale al robot, un movimento del quale si riflette in un identico movimento della webcam, sul piano orizzontale. Ciò implica che qualora la webcam venga posta in movimento (ego-motion), per l'osservatore, solidale con la webcam, sarà lo sfondo (più i soggetti in movimento e non) a muoversi e pertanto avremo un movimento di *tutti* i contorni del quadro video, *senza distinzione alcuna* tra *soggetti* e *sfondo*. Quindi, ovviamente, la rivelazione di soggetti in movimento dev'essere eseguita a robot immobile.
Ad esempio ciò può essere eseguito dal robot, durante periodi di latenza o di valutazione sul cammino da intraprendere (robot immobile) oppure fermando il robot ad intervalli temporali prefissati per permettergli la rivelazione del movimento, oppure ancora tale lavoro può essere svolto da webcam fissate in posti opportuni lungo il corridoio (pareti o soffitto), che poi comunicano al

robot, le condizioni di *movimento in corso* nel corridoio (la comunicazione potrebbe avvenire acusticamente o mediante un sistema visivo basato sulla applicazione di riconoscimento di segnaletica presentato nel precedente capitolo).
Un'alternativa molto onerosa per il computer, soprattutto *in tempo reale*, ma che consentirebbe il *rivelamento differenziato* dello sfondo e dei soggetti, *durante* il movimento del robot, consiste nella realizzazione di un algoritmo per il calcolo del *flusso ottico*, il quale consente la rivelazione del *campo di velocità* (vettore velocità *per ogni* quadrato di nxn pixel) dell'intera immagine (cfr. paragr. 1.3.4)

Di seguito saranno evidenziate alcune proprietà del rivelatore di movimento e a seguire sarà citato anche un sistema di rivelamento di bordi (o contorni o margini), mediante approssimazione del gradiente spaziale, implementato insieme al rivelatore di movimento, che permette una distinzione dei contorni anche quando non è presente movimento nella scena, disponibile per *eventuali* usi successivi a quelli presentati in questo lavoro.
Concluderà il capitolo la breve spiegazione dell'espediente pratico intuitivo, che nonostante i suoi limiti, consente il rapido ottenimento di sagome 'piene' (filled) a partire dai *contorni*, ottenuti dal rivelatore di movimento, *quando* sono abbastanza *consistenti*.

5.1 Rivelatore di movimento (motion detector) ed alcune proprietà

Il rivelatore è basato sulla differenza pixel a pixel tra fotogrammi (frame) successivi (image differencing). In altre parole, considerato di aver immagazzinato in un'area di memoria (frame buffer), il valore di ogni pixel dell'immagine *precedente* f'(x,y) e di avere a disposizione, in un'altra area di memoria, il valore di ogni pixel dell'immagine *attuale* f(x,y), si può effettuare la semplice operazione di valore assoluto della differenza tra 2 pixel con le stesse (x,y):

(#) $$diff(x,y) = | f(x,y) - f'(x,y) | \ ;$$

$$marking = (\ \mathbf{0}\ \text{per}\ \ diff(x,y) \leq Tt\ \ ,\mathbf{1}\ \text{per}\ \ diff(x,y) > Tt\ )\ ;$$

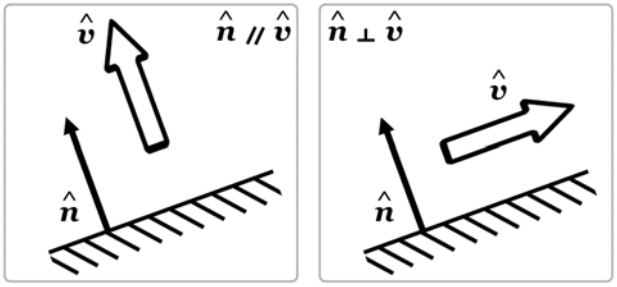
dove Tt è la soglia di tolleranza.
Se ora si provvede a *marcare* (ad esempio in *rosso*) i pixel il cui valore di *m* risulta pari all'unità e a lasciare immutati gli altri (anche se l'effetto finale è più evidente ponendo il valore di questi ultimi in modo da farli diventare *bianchi*), si potrà osservare che le zone marcate, corrispondono ai contorni degli oggetti che da un frame all'altro hanno effettuato uno spostamento (cfr. figura seguente di sfera in movimento).

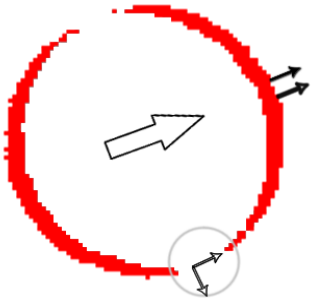
Dato che si utilizzerà nel seguito, definiamo subito il concetto di *motion blur* (sfocatura da movimento), come l'effetto che si produce su porzioni di un'immagine, corrispondenti ad oggetti in movimento all'esterno, durante il tempo di acquisizione (*apertura di camera*) di una webcam.
Questo è il medesimo effetto di "sbavatura" cui si presta attenzione ad evitare quando si scatta una fotografia.
La webcam consente una *riduzione* del tempo di *apertura di camera*.
Si aggiunge che non sempre il *motion blur* risulta essere un effetto indesiderato.

Enunciamo ora alcune *proprietà* della rivelazione del movimento che si notano immediatamente quando si fa uso della procedura (#) prima citata:

(o) Per effetto del *motion blur*, lo spessore dei bordi rivelati (in rosso) *aumenta proporzionalmente* alla velocità dell'oggetto (…da quanto detto si sarebbe tentati di pensare di poter valutare la velocità (dei bordi) misurando tale inspessimento; il problema è che può risultare *non semplice* dedurre la *direzione* del movimento, sempre che non sia nota *a priori*).
L'alta velocità può causare una sovrapposizione di bordi rivelati (rossi) tale da rendere ciò che viene rivelato *molto diverso* dall'oggetto originario in movimento (specie se l'oggetto in questione non è un corpo rigido).

(i) Se la direzione della *normale* del bordo dell'oggetto in movimento *è parallela* alla direzione del movimento, il rivelamento ha *effetto massimo* (uno o più punti rossi attestano il rivelamento locale), viceversa nel caso di ortogonalità il rivelamento *non ha effetto* e nessun pixel viene marcato, nonostante tale bordo appartenga all'oggetto in movimento.





esempio di rivelazione del movimento di una sfera.

(ii) Se il valore di un pixel, appartenente al bordo dell'oggetto in movimento, è *simile* a quello del pixel (adiacente) appartenente allo sfondo, allora il rivelamento *fallisce* (nessun bordo rivelato).
Il termine *simile* va inteso nel senso che '*rientra nella soglia di tolleranza*' (cfr.(#)).

Tale seconda proprietà, ancor prima che per il rivelamento del movimento, vale per il rivelamento dei bordi (gradiente spaziale), dove spazialmente *non è possibile* determinare il bordo in una situazione del genere (oggetto *fisso* con toni di grigio *vicini* a quelli dello sfondo *fisso*).
Tale problema si presenta anche per la percezione visiva umana; infatti qualora l'oggetto, il cui bordo si confonde con lo sfondo, non sia riconducibile ad una forma nota (in base all'esperienza), *non* si sarà in grado di distinguere i bordi dell'oggetto.
In alcuni casi diminuendo la soglia di tolleranza si può migliorare un po' la situazione, tenendo però presente la crescita del rumore.

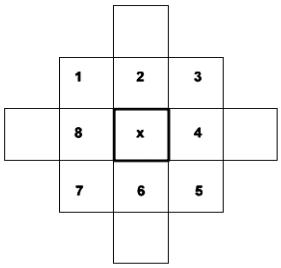
(iii) Presenza intrinseca di *rumore* sotto forma di pixel isolati che appaiono e scompaiono nell'arco di 2 frame successivi. Le cause sono *interne* ed *esterne* ed in ordine dipendono dal CCD, dal circuito elettrico a monte della quantizzazione, dal quantizzatore e poi a valle (numericamente) dalla compressione lossy YUV420P (già introdotta) residente nella webcam ed *esternamente* dall'illuminazione ambientale.
Considerando il contributo di tutte le sorgenti di rumore si ha l'esigenza di un filtraggio software per i punti isolati (anche in condizioni di ottima illuminazione).

(iv) Il rivelatore di movimento, nel rivelare le variazioni di luminosità dei singoli pixel, ci restituisce oltre al profilo (non completo per la (ii)) dell'oggetto in movimento su sfondo fisso, anche i profili dei cambi di luminosità delle *zone interne* all'oggetto, per via della sua tridimensionalità (luce diffusa e/o riflessa, ombre proprie e/o proiettate, texture non omogenei, etc).

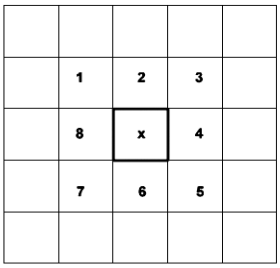
5.2 Rivelatore di bordo (edge detector)

Questo metodo (per rivelare il bordo spaziale) è basato su approssimazione del gradiente ed è simile a quello degli *operatori di Roberts* (cfr. paragr.1.3.2).
Il funzionamento è semplice, per ogni pixel si esaminano i pixel adiacenti (*intorno*) e *se* quello centrale presenta un valore maggiore di uno di quelli adiacenti, al di là di una certa soglia, *allora* il pixel viene *marcato*, ovvero riconosciuto come *bordo* di una zona più scura.

Il tipo d'*intorno*, cui si è fatto riferimento nella messa a punto, è quello formato da 12 pixel (8+4). Se dovesse occorrere un contorno *più spesso* si può scegliere un *intorno* di 24 pixel (8+16).

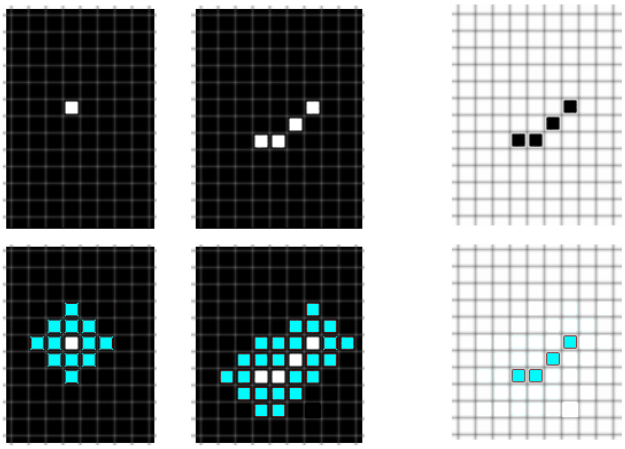


Intorno di 12 pixel (8+4)



Intorno di 24 pixel (8+16)

Per come è stato impostato, il rivelatore di bordo *circonda* le zone più chiare ponendo i pixel marcatori sulle zone più scure (cfr. figura seguente), quindi se si rivela un pixel bianco su sfondo nero, quest'ultimo verrà circondato da pixel marcatori (che quindi verranno scritti al posto di quelli neri che lo circondavano) creando una sorta di anello, che può variare la sua forma a seconda che si usino intorni di 12 (8+4) o 24 (8+16) pixel. Allo stesso modo una corta curva bianca, di spessore 1 pixel, su sfondo nero verrà circondata da pixel marcatori.
Viceversa un punto o una curva nera su sfondo bianco sarà l'unica ad essere marcata.



risultato Intorno di 12 Pixel

risultato Intorno qualsiasi

Si ricorda che ciò non vale per il rivelatore di movimento, per il quale ogni variazione viene marcata.
Tale sistema è in grado di rivelare solo bordi abbastanza *netti* (cioè variazioni che si articolano su spessori di massimo 2 pixel), il che può risultare utile nel caso si desideri 'filtrare' le variazioni più lente.

Come si può notare, alcune proprietà del rivelatore di movimento ((ii), (iii) e (iv)) si ripropongono per il rivelatore di bordo (il quale rivela variazioni nette nel *gradiente spaziale* mentre l'altro rivela variazioni nette nella *derivata temporale*).

5.3 Generatore di sagome

Tale semplice espediente pratico, consente il rapido ottenimento di sagome 'piene' (filled) a partire dai contorni del rivelatore di movimento, quando tali contorni risultano *abbastanza consistenti*.

La griglia, su cui agisce il generatore di sagome, è *ridotta* (ad esempio *64x48* pixel) rispetto all'immagine acquisita (*320x240* pixel). Questo è stato fatto per limitare la frammentazione dei contorni per effetto della proprietà **(ii)**, ovvero in presenza di sfondi che possano avere pixel di valore simile a quello dei bordi del soggetto in movimento.
La quantizzazione, che dà origine alla griglia *ridotta* (roughscan[ ][ ] nel sorgente), si ottiene semplicemente *contando* (per ogni pixel della griglia *ridotta*) il numero dei pixel *rivelat*i nel corrispondente quadrato (*5x5* pixel, per continuità rispetto ai valori dati sopra) dell'immagine di partenza. In tal modo se alcune zone *non* vengono rivelate per la proprietà (ii), si confida che in alcune altre, presenti sempre nel medesimo quadrato, ciò non sia accaduto.

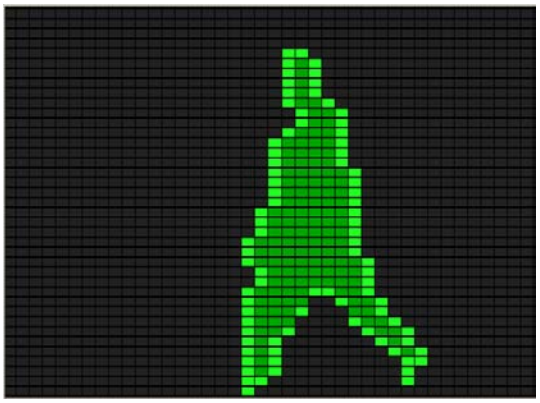
Come già detto, l'espediente per ottenere sagome 'piene', *agisce* sulla griglia *ridotta*.
Brevemente, si utilizzano 2 vettori della stessa lunghezza (pari a quella dell'ascissa della griglia *ridotta*). Supponiamo di avere nella griglia un contorno rivelato, come in fig.a:

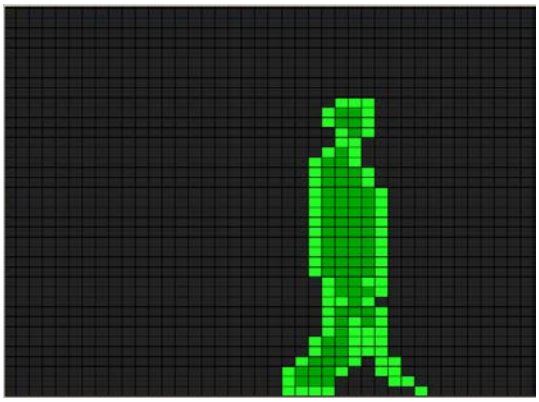
fig. a

Si procede nella scansione dell'immagine *memorizzando* nel primo vettore, per ogni casella dell'ascissa, il valore massimo dell'ordinata, raggiunto dal contorno, e nel secondo vettore i valori delle ordinate minime. Pertanto in fase di *visualizzazione* non si farà altro che tracciare tutti i punti *compresi* tra il

massimo ed il minimo, di ogni ascissa.

Tale sistema risulta adeguato se gli oggetti da visualizzare sono contorni di *insiemi convessi*, risulta invece *approssimato se* gli oggetti presentano concavità (caso generale).
Si è proceduto quindi ad aggiungere altri due vettori (destro e sinistro) di lunghezza pari all'ordinata affinchè contengano le ascisse massime e minime. Tale miglioramento però è solo *parziale*, nonostante consenta ora di ottenere una sagoma 'piena' riconoscibile.





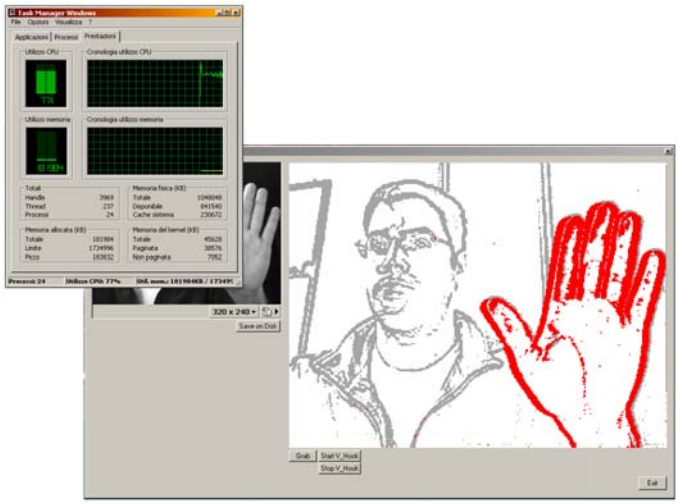
Nel momento in cui sono presenti, sulla griglia *ridotta*, 2 oggetti (con concavità), i loro bordi (compresi nello spazio che li divide) perdono il dettaglio apportato dall'aggiunta dei due vettori (per le ascisse massime e minime), per ovvi motivi.

Si ripete che tutto ciò vale nel caso di una certa *consistenza* dei contorni rivelati (di un oggetto in movimento). Infatti se l'oggetto si muove *continuamente* (ad esempio una persona che cammina) i contorni conservano una certa consistenza e la rivelazione del suo contorno ha luogo. Nel momento in cui il movimento cessa, ovviamente, la rivelazione non ha più luogo e la sagoma, precedentemente visualizzata, perde rapidamente forma sino a scomparire.

Se si desidera mantenere la sagoma di una persona che si ferma, mentre magari *agita* un braccio (quest'ultimo viene rivelato) si deve procedere nella realizzazione di un sistema che sia in grado di *differenziare* i soggetti dallo sfondo e di memorizzare quest'ultimo.

6 Considerazioni sulle prestazioni

Ciascuna delle due applicazioni descritte (in *tempo reale* cioè a 30 fps), *separatamente* fa uso, in media, di circa l'80% di *utilizzo di CPU*, come si può facilmente verificare 'aprendo' la finestra del *Task Manager* di Windows, durante l'esecuzione di una delle suddette applicazioni.



Vale a dire che la *CPU* (central processing unit) rimane *libera* dall'eseguire operazioni per il 20% circa dell'*unità di tempo*. Rimane disponibile il tempo necessario per far funzionare, *contemporaneamente* al sistema di *visione*, un sistema di *navigazione* per il robot mobile, che si avvalga di algoritmi per la *fusione sensoriale* dei dati visuali acquisiti dalla webcam e di quelli disponibili mediante i sensori del robot.

Si è parlato di un utilizzo, *in media*, dell'80% tempo di CPU, questo perché l'*andamento* del tempo di lavoro della CPU può subire delle oscillazioni (sempre entro il 7 - 8 %) che tendono lentamente a smorzarsi o a stabilizzarsi (intorno a dei valori percentuali *bassi* di ampiezza di oscillazione).
Ciò è dovuto a diverse cause, una delle quali è conseguenza diretta delle applicazioni, infatti l'elaborazione delle immagini arriva a richiedere maggiore

capacità di calcolo, in particolar modo, quando si è in presenza di target da riconoscere.
Altre cause vanno ricercate nella gestione della memoria e dello scambio dati con le periferiche da parte del sistema operativo.

La *spazio di memoria* occupato dal programma, durante l'esecuzione, è generalmente inferiore ai 6 Mb.

Le applicazioni presentate in questo lavoro, possono essere *rapidamente* modificate per funzionare nella modalità (640x480) a *15 fps*, disponendo di un computer leggermente più potente di quello utilizzato in questa sede.

7 Conclusioni

Le applicazioni presentate in questo lavoro sono risultate adeguate dal punto di vista della velocità di calcolo ed il prossimo passo consisterà nell'impiego della *prima* applicazione direttamente nella navigazione del robot NOMAD.
Come in tutti i sistemi di Visione Artificiale, le cattive condizioni di illuminazione possono inficiare le prestazioni delle applicazioni presentate (nonostante la compensazione automatica eseguita dalla webcam), ciò rientra tra i miglioramenti da apportare in futuro.

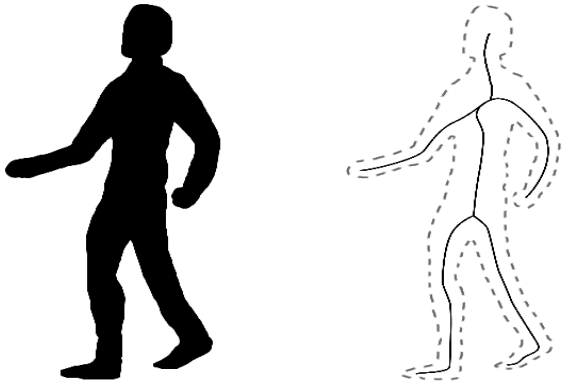
Si vogliono ora esaminare alcuni *sviluppi possibili* per le applicazioni descritte.
C'è da premettere che da qui a breve, con l'introduzione delle porte USB 2.0 e con la completa diffusione dei collegamenti Internet ad alta velocità (dell'ordine di 10 Mbit/sec), saranno disponibili sul mercato webcam dalle caratteristiche *più spinte* che consentiranno quindi un'*acquisizione migliore* delle immagini (risoluzioni più elevate, possibilità di acquisizione in formato *non compresso* e qualità superiore del CCD che si traduce in una *diminuzione del rumore*).

Per ciò che riguarda la *prima* applicazione, avendo a disposizione una maggiore potenza di calcolo, si può *sostituire* la rete, di cui si fa uso, con una rete neurale con più neuroni (24x24 per esempio) e che utilizzi una legge di evoluzione che presenti caratteristiche *migliori*, in termini di riconoscimento, rispetto a quella base di Hopfield.
Si rammenta che il *rapido* sistema di rivelazione di un target circolare consente al robot, abbinato ad un adeguato sistema di navigazione, di potersi destreggiare nell'*inseguimento* di un… pallone da calcio (…RoboCup).

Riguardo alla *seconda* applicazione, si cita una possibile *linea* di sviluppo.
Come *primo* passo occorre *migliorare* il sistema di estrazione di sagome 'piene'; occorre un maggior *dettaglio* ed una *persistenza* della sagoma del

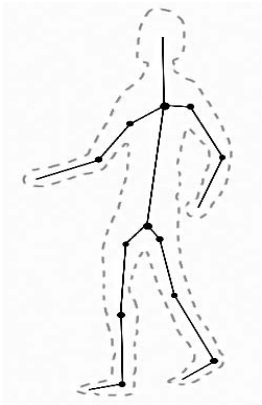
soggetto nel caso questi si fermi (e pertanto anche la possibilità di definire ed immagazzinare lo sfondo).

Come <u>*secondo*</u> passo, si può ricorrere all'elaborazione morfologica per estrarre lo *scheletro* della sagoma, mediante il *thinning* morfologico (paragr. 1.3.3):



sagoma estratta *thinning* della sagoma estratta

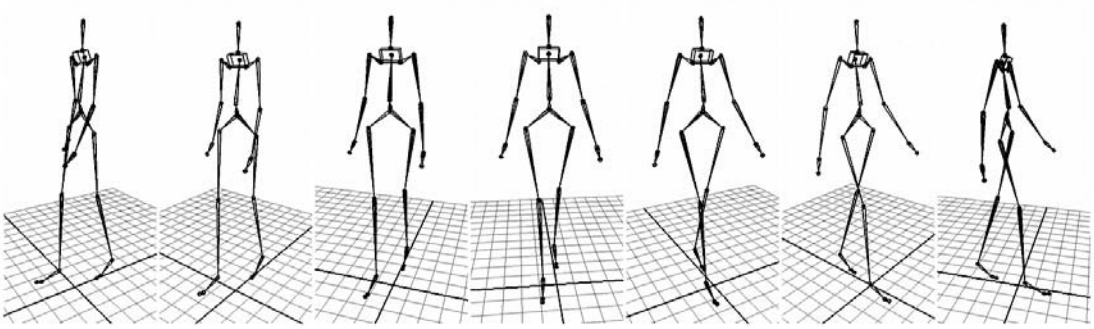
Come *terzo* passo si può cercare di ottenere una *rappresentazione a rette* mediante la *trasformata di Hough* (paragr. 1.4.3), del risultato curvilineo del *thinning*:



Trasformata di Hough del *thinning* della sagoma estratta

E come *ultimo* passo si può tentare di *associare* le direzioni di tali rette a quelle di un *modello interno* di persona, al fine di ottenere la *postura* del soggetto osservato.

Nel caso poi che la scena risulti acquisita da *due* diverse angolazioni, la *postura recuperata* risulterebbe essere tridimensionale:



Lo scopo del *recupero di postura* è quello dell'*acquisizione di gestualità*, che consentirebbe ad un sistema di videosorveglianza di poter *discriminare* le azioni in corso.
Un tale sistema permetterebbe inoltre una grossolana *cattura dei movimenti* (*markerless motion capture*) del soggetto osservato.
Un altro approccio a tale *cattura* è riportato in [12], in [16] è riportata una guida sui metodi per la cattura del movimento esistenti e per ciò che riguarda l'*inseguimento* della locomozione umana cfr. [14].

E' ovvio che un'applicazione *in tempo reale* di tal genere può risultare *troppo pesante* (computazionalmente parlando) per un solo computer, quindi si può ricorrere ad un'elaborazione in *cascata* di più computer.
Ad esempio il 1°computer si occuperebbe dell'acquisizione dati e dei filtraggi base (elaborazione preliminare, segmentazione) in modo da mantenere un'acquisizione ad esempio di 30 fps, il 2°computer riceverebbe i dati (sempre tramite porta USB 2.0 o "Firewire" (protocollo IEEE1394)), già preelaborati dal 1°computer e quindi potrebbe elaborare dati a partire da un livello più alto. Tale procedura può, in caso di necessità, essere *iterata* mediante l'*accodamento* di altri computer ai primi due.

Si aggiungono ora *due* brevi considerazioni su altre *modalità* di acquisizione dati mediante webcam, che possono risultare utili per applicazioni future.

La *prima* considerazione è che la webcam, potendo acquisire a *colori*, permette di mettere a punto un sistema di segmentazione più *raffinato*, di quanto si possa fare con i soli toni di grigio.

La *seconda* considerazione riguarda il fatto che disponendo di *due* webcam si può sfruttare la *stereopsi*. Un modo immediato per ottenere la sua *rivelazione* è mediante 2 webcam distanti tra loro 6 cm (come la distanza tra gli occhi, per esempio), con gli obiettivi centrati verso un punto nello spazio di fronte (al limite all'infinito - webcam *parallele*) e fare la *differenza pixel a pixel* tra le due immagini acquisite, marcando in rosso i pixel che risultano differenti, nell'immagine in uscita.
Com'è noto tale sistema ha la proprietà di permettere la distinzione degli oggetti in base alla distanza che li separa dall'osservatore. Gli oggetti più vicini saranno quelli che risulteranno con i bordi più marcati in quanto, nelle 2 immagini acquisite, saranno quelli che presenteranno maggiori discordanze.
Oltre una certa distanza, dall'osservatore, le variazioni di un oggetto, ritratto nelle 2 immagini, diventano trascurabili e quindi la visione diventa essenzialmente monoculare.
Le SDK della webcam della Logitech *già contemplano* il caso di più webcam collegate ad uno stesso computer (cioè le librerie sono predisposte a gestire un flusso dati (multiplo) da 2 o più webcam collegate simultaneamente).

Bibliografia:

[1] Hopfield, J.J., "Neurons and physical systems with emergent collective computational abilities", Proceedings of the National Academy of Sciences, 79, pp. 2554-2558, 1982.

[2] R.Serra,G.Zanarini , Sistemi complessi e processi cognitivi, , ed.Calderini, 1994.

[3] Amit D.J., Modellizzare le funzioni del cervello, CEDAM, Padova, 1995.

[4] A. Broggi, M. Bertozzi, A. Fascioli, Guarino Lo Bianco, and Aurelio Piazzi, “Visual Perception of Obstacles and Vehicles for Platooning”, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, Sept. 2000.

[5] King-Sun Fu, Rafael C.Gonzalez, C.S. George Lee, Robotica, McGraw-Hill, 1989.

[6] Rafael C.Gonzalez, Richard E.Wood, Digital Image Processing, Addison-Wesley Publishing Company, 1992.

[7] E.R.Kandel, J.H.Schwartz, T.M. Jessell, Principi di Neuroscienze, C.E.A., 1994.

[8] Kawakami, S., Matsuoka, M., Okamoto, H. & Hosogi S. (2000). A neural network model for detecting planar surface spatially from optical flow in area MST of the visual cortex. IEICE J83-D-II, pp. 2786-2797.

[9] G.T. Buracas, T. D. Albright, “Contribution of Area MT to Perception of Three-Dimensional Shape: A Computational Study”, (reperibile su Internet, mediante motore di ricerca)

[10] Leon O. Chua, Tamas Roska, “The CNN Paradigm”, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, March 1993.

[11] T. Poggio, V. Torre, and C. Koch. Computational vision and regularization theory. Nature, 317:314–319, September 1985.

[12] ..., “Markerless Kinematic Model and Motion Capture from VolumeSequences”, submitted to Computer Vision and Pattern Recognition, 2003, (reperibile su Internet mediante motore di ricerca)

[13] M. A. Giese, T. Poggio, “neural mechanisms for the recognition of biological movements”, Nature Reviews, Neuroscience, march 2003.

[14] T. Zhao, R. Nevatia, “3D Tracking of Human Locomotion: A Tracking as Recognition Approach”, (reperibile su Internet mediante motore di ricerca)

[15] A. Broggi, M. Bertozzi, A. Fascioli, M. Sechi, “Shape-based Pedestrian Detection”, IEEE Intelligent Vehicles Symposium 2000.

[16] T. B. Moeslund, E. Granum “A Survey of Computer Vision-Based Human Motion Capture”, Computer Vision and Image Understanding, 2001.

[17] Hebb D. “The organization of behaviour”, John Wiley, 1949.

[18] J.P. Lewis, “Fast Normalized Cross-Correlation” , (reperibile su Internet mediante motore di ricerca)

[19] G.C. DeAngelis, B.G. Cumming, W.T. Newsome,“Cortical area MT and the perception of stereoscopic depth”, Nature, August 1998.

[20] H. Liu, T.Hong, M. Herman, R.Chellappa, “Accuracy vs. Efficiency Trade-offs in Optical Flow Algorithms”, (reperibile su Internet mediante motore di ricerca)

Riferimenti Internet:

(1) www.honda.com
(2) http://www.boondog.com/%5Ctutorials%5CtripodBinarize%5CtripodBinarize.html
(3) http://www.dai.ed.ac.uk/CVonline/
(4) http://www.sci.univr.it/~fusiello

Appendice A:

Calcoli per la determinazione del centro della circonferenza sul piano noti 3 punti.

Eq. 1ª retta tra x_1 e x_2 : $(y_2 - y_1) = m_{12} (x_2 - x_1)$;

m_{12} = pendenza retta passante per x_1,y_1 e x_2,y_2
$q_{12} = -1/m_{12}$ = pendenza retta passante per x_{12},y_{12} e x_c,y_c

retta passante tra p_c e p_{12} : $(y_{12} - y_c) = q_{12} (x_{12} - x_c)$;
con : $1/m_{12} = (x_2 - x_1) / (y_2 - y_1)$; $x_{12} = (x_1 + x_2)/2$; $y_{12} = (y_1 + y_2)/2$;

Eq. 2ª retta tra x_2 e x_3 : $(y_3 - y_2) = m_{23} (x_3 - x_2)$;

m_{23} = pendenza retta passante per x_2,y_2 e x_3,y_3
$q_{23} = -1/m_{23}$ = pendenza retta passante per x_{23},y_{23} e x_c,y_c

retta passante tra p_c e p_{23} : $(y_{23} - y_c) = q_{23} (x_{23} - x_c)$;
con : $1/m_{23} = (x_3 - x_2) / (y_3 - y_2)$; $x_{23} = (x_2 + x_3)/2$; $y_{23} = (y_2 + y_3)/2$;

pertanto risulta un sistema di **2 equazioni di primo grado** nelle incognite x_c e y_c :

$$(y_{12} - y_c) = q_{12} (x_{12} - x_c)$$
$$(y_{23} - y_c) = q_{23} (x_{23} - x_c)$$

la cui soluzione in generale risulta essere :

$$x_c = (y_{23} - y_{12} + q_{12} x_{12} - q_{23} x_{23}) / (q_{12} - q_{23}) ;$$
$$y_c = q_{12} (x_c - x_{12}) + y_{12} = q_{23} (x_c - x_{23}) + y_{23} ;$$

in particolare se :
1) $y_2 = y_1$ allora $q_{12} = +/- \infty$; *corda*$_{12}$ orizz.
In tal caso: $x_c = x_{12}$; $y_c = q_{23} (x_c - x_{23}) + y_{23}$;
2) $y_3 = y_2$ allora $q_{23} = +/- \infty$; *corda*$_{23}$ orizz.
In tal caso: $x_c = x_{23}$; $y_c = q_{12} (x_c - x_{12}) + y_{12}$;

Se però capitano entrambe i casi, si è su una retta orizzontale e quindi non si prosegue oltre.

3) $q_{12} = q_{23}$ allora *rette parallele*, centro all'infinito
si è su una retta e non si prosegue oltre.
Questa eventualità non viene contemplata, viene delegato ad un test successivo di verificare *se* le coord. (x_c,y_c) rientrino o meno in un range accettabile.

Appendice B: sorgente C++

I listati C++ che seguono (ed i relativi file *header*) sono i listati *essenziali* del *project* in Visual C++ *Recognition*, che contiene entrambe le applicazioni.

Essi sono in ordine:

RecognitionDlg.h,
RecognitionDlg.cpp,
Image.h,
Image.cpp.

La *classe* principale è codificata in *Image.cpp*.