POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

PARTICLE FILTER PER L'INSEGUIMENTO DI OGGETTI IN MOVIMENTO

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Ing. Matteo Matteucci
Correlatore: Dott. Davide Migliore

Tesi di Laurea di:
Manuel Fossati, matricola 653443

Anno Accademico 2007-2008

Sommario

Le tematiche affrontate nello sviluppo di questa tesi si collocano all'interno della Computer Vision, la scienza mirata alla costruzione di sistemi intelligenti capaci di interpretare immagini allo scopo di ricavarne informazioni. In particolare, il settore di appartenenza del lavoro svolto è quello dell'object tracking, che si occupa di identificare il movimento di oggetti per mantenere traccia della loro posizione. Lo scopo della tesi è la costruzione di un sistema probabilistico in grado di stimare iterativamente, in fotogrammi successivi, la posizione di un oggetto all'interno di una scena statica. La stima è operata da un filtro ricorsivo chiamato Particle Filter, il quale necessita di un modello per la rappresentazione degli oggetti e di criteri per la loro identificazione. Lo studio dei metodi presentati in letteratura ha portato all'individuazione di due diversi modelli; si è scelto di implementarli entrambi, sia per aumentare la versatilità del sistema, sia per poterne confrontare le prestazioni.

Indice

Capitolo 1

Introduzione

We may hope that machines will eventually compete with men in all purely intellectual fields. But which are the best ones to start with? Many people think that a very abstract activity, like the playing of chess, would be best. It can also be maintained that it is best to provide the machine with the best sense organs that money can buy, and then teach it to understand and speak English.

Alan M. Turing

1.1 Inquadramento generale

La Computer Vision è la branca dell'intelligenza artificiale che si occupa di ricavare informazioni da immagini acquisite e prendere decisioni utili sulla base di quanto elaborato. Una delle principali tematiche ascrivibili alla computer vision è l'Object Tracking, che mira all'identificazione e all'inseguimento di oggetti in moto. Essa trova applicazione in molteplici campi:

- Robotica.
- Ispezione di processi industriali.
- Interpretazione di immagini provenienti da satelliti.
- Indicizzazione di database multimediali.
- Interazione uomo-macchina.
- Analisi di immagini medico-diagnostiche.

Lo scopo della tesi è realizzare un sistema di tracking capace di generare un modello statistico di un oggetto, grazie al quale è possibile individuarne la posizione in fotogrammi successivi mediante una serie di confronti. L'approccio seguito è di tipo ricorsivo: si considerano diversi nuovi stati possibili dell'oggetto a partire dalla posizione stimata nel frame precedente, valutandoli rispetto al modello e stimando la nuova posizione sulla base delle valutazioni effettuate. È quindi fondamentale costruire un modello robusto, capace di individuare l'oggetto anche in presenza di cambi non repentini di illuminazione, e disporre di una misura di similiarità sufficientemente discriminante.

Dallo studio della letteratura sono emersi in particolare due modelli dotati delle caratteristiche necessarie: si è scelto di implementarli entrambi, in modo da rendere il sistema più flessibile. L'uso di due modelli ha inoltre permesso di effettuare un'analisi comparativa, valutandone le prestazioni in presenza di diverse condizioni.

1.2 Breve descrizione del lavoro

L'inseguimento di oggetti in movimento è da anni oggetto di studio nell'ambito della computer vision. Diversi approcci, documentati in letteratura, hanno portato allo sviluppo di metodologie che affrontano il problema da differenti punti di vista. L'approccio seguito in questa tesi, noto con il nome di Particle Filter [39, 30], si basa sulla generazione di ipotesi sullo stato dell'oggetto, che vengono valutate in base a una misura di similiarità con un modello costruito in precedenza. La valutazione è di tipo probabilistico: ogni ipotesi generata rappresenta un campione, a cui corrisponde una determinata probabilità di corrispondenza con l'oggetto. Per ogni fotogramma della sequenza viene stimata la posizione dell'oggetto mediante una somma pesata dei campioni. La nuova posizione servirà come base di partenza per la generazione di nuovi campioni nel fotogramma successivo; il particle filter viene perciò definito *ricorsivo*.

La scelta di un modello appropriato per la rappresentazione degli oggetti è fondamentale ai fini dell'accuratezza del particle filter. L'approccio tradizionale consiste nel modellare gli oggetti costruendone l'istogramma colore [52, 64, 29]. Esso costituisce una rappresentazione semplice e robusta della distribuzione del colore nell'area in cui è presente l'oggetto, che in molte condizioni si rivela sufficiente a distinguerlo da altri oggetti e dallo sfondo. I limiti di questo modello si evidenziano in presenza di oggetti con colori simili tra loro o difficili da distinguere rispetto allo sfondo. In questi casi

l'istogramma colore non si rivela abbastanza discriminante e il particle filter può perdere traccia dell'oggetto.

Il modello SMOG [23] è una mistura (ovvero una somma pesata) di gaussiane che considera la caratteristica congiunta spazio-colore, in modo da avere informazioni non solo sulla distribuzione del colore, ma anche sulla posizione dell'oggetto. La maggiore ricchezza di informazione apportata da questo modello rende il tracking più accurato in presenza di condizioni che mettono in crisi il modello basato unicamente sulla distribuzione del colore.

In questa tesi sono stati implementati entrambi i modelli sopracitati in differenti spazi colore. L'applicazione realizzata permette di selezionare, mediante appositi parametri, modello e spazio colore da utilizzare. Dato che il lavoro svolto è stato focalizzato sull'object tracking, la fase di *object detection* (responsabile dell'identificazione della presenza di oggetti nella scena) non è stata sviluppata. È quindi possibile seguire lungo una sequenza di fotogrammi soltanto oggetti di cui si specifica a priori la posizione iniziale. Il vantaggio di questo approccio è che la conoscenza esatta della posizione permette la costruzione di un modello accurato; d'altro canto, però, questo costituisce un vincolo importante, rendendo l'applicazione dipendente da una conoscenza esterna. Pertanto, un possibile sviluppo futuro potrebbe riguardare l'integrazione del metodo di tracking con una precedente fase di detection, in grado di individuare la presenza di oggetti nella scena (ad esempio mediante confronti con lo sfondo).

1.3 Struttura della tesi

Di seguito è mostrata la struttura della tesi, con una breve descrizione del contenuto dei capitoli che la compongono.

- Nel Capitolo 2 viene analizzato lo stato dell'arte, attraverso una panoramica su metodi e algoritmi presenti in letteratura.
- Nel Capitolo 3 sono esposte le problematiche affrontate e i vincoli imposti per la loro risoluzione.
- Nel Capitolo 4 è descritto dettagliatamente il sistema di tracking realizzato. Dapprima è mostrato l'algoritmo di particle filtering, illustrato sia dal punto di vista teorico che da quello dell'implementazione. Viene quindi posta particolare enfasi nella descrizione dei modelli adibiti alla rappresentazione degli oggetti (istogramma colore e SMOG). In ultimo vi è una descrizione degli spazi colore utilizzati e degli algoritmi di conversione che li mettono in relazione.

- Nel Capitolo 5 è mostrata l'architettura del sistema, considerando i diversi moduli che lo compongono da un punto di vista logico.
- Nel Capitolo 6 sono discussi i risultati ottenuti dal sistema, comparando i modelli e individuando possibili sviluppi futuri.
- Nell'Appendice A sono mostrate alcune parti rilevanti del codice sorgente dell'applicativo sviluppato.
- L'Appendice B contiene il manuale utente relativo all'installazione e all'utilizzo dell'applicazione.
- Nell'Appendice C è mostrato un esempio di utilizzo dell'applicazione.

Capitolo 2

Stato dell'arte

Nulla si conosce interamente finché non vi si è girato tutt'attorno per arrivare al medesimo punto provenendo dalla parte opposta.

Arthur Schopenhauer

L'*object tracking* è una delle applicazioni più importanti nell'ambito della computer vision. Esso trova infatti applicazione in svariati ambiti, tra cui la videosorveglianza, l'indicizzazione di video in database multimediali (video indexing), l'interazione uomo-macchina (human-computer interaction), il monitoraggio del traffico.

Seppur semplice concettualmente, l'inseguimento di oggetti in movimento non è di facile realizzazione. In letteratura si trovano una moltitudine di approcci diversi, alcuni dei quali si dimostrano migliori di altri in determinate condizioni. È possibile classificare i vari metodi di object tracking suddividendoli in tre categorie principali [1]:

- Point Tracking.
- Kernel Tracking.
- Silhouette Tracking.

Queste categorie, a loro volta suddivisibili in sottocategorie, saranno analizzate singolarmente nel corso del capitolo. L'analisi sarà accompagnata da una panoramica dei metodi di object tracking presenti in letteratura, opportunamente divisi nelle rispettive categorie.

2.1 Point Tracking

In questa classe di algoritmi gli oggetti di cui tenere traccia sono rappresentati da punti. È quindi necessario un meccanismo per riconoscere gli oggetti in ogni frame, creando una corrispondenza di punti con il frame precedente (il che può creare problemi in caso di occlusioni o di oggetti che entrano ed escono dalla scena). I metodi di point tracking possono essere di due tipi: *deterministici* e *probabilistici*.

2.1.1 Metodi deterministici

I metodi deterministici fanno uso di vincoli di movimento euristici per determinare la corrispondenza dei punti tra frame successivi. Lo scopo di questi metodi è minimizzare una funzione di costo, definita come combinazione di vincoli, mediante un problema di ottimizzazione combinatoria. I vincoli utilizzati possono essere di vario tipo:

- Vincoli riguardanti la *velocità* del movimento. Essi stabiliscono che la velocità (e di conseguenza la posizione) di un oggetto da un frame al successivo non possa cambiare radicalmente. Vincoli di questo tipo sono ad esempio quelli di *prossimità*, *massima velocità*, *piccoli cambiamenti di velocità*.

- Vincoli di *rigidità*. Essi stabiliscono che gli oggetti nello spazio tridimensionale debbano essere rigidi, mantenendo di conseguenza costante la distanza tra due punti di un oggetto.

I principali metodi collocabili in questa categoria sono estensioni dell'algoritmo proposto da Sethi e Jain [26]. Tale algoritmo, di tipo *greedy* (che ottimizza iterativamente la soluzione), è basato sui vincoli di prossimità e rigidità. Esso, nella sua versione di base, non consente di gestire le occlusioni, le entrate e le uscite di oggetti dalla scena. Tuttavia, data la sua struttura deterministica, risulta semplice aggiungere ulteriori vincoli che ne migliorino le prestazioni [12, 25]. Il problema delle occlusioni è risolto dal tracker MGE proposto da Salari e Sethi [62], il cui approccio consiste nell'aggiunta di punti ipotetici nel caso in cui non si abbia alcuna corrispondenza con l'oggetto.

Altri metodi deterministici sono quelli proposti da Mehrotra [44] e da Rangarajan e Sha [34]. Quest'ultimo, insieme al metodo di Sethi e Jain, è alla base del tracker GOA proposto da Veenman et al. [10]. Esso introduce un'importante innovazione: la differenziazione tra *individual motion models*, *combined motion models*, *global motion model*. Il vincolo di common motion

è utilizzato per avere una traccia coerente dei punti appartenenti allo stesso oggetto (non ha invece effetto in caso di punti appartenenti a oggetti diversi che si muovono in direzioni diverse). Pur essendo in grado di gestire le occlusioni, anche il tracker GOA può seguire solo un numero di oggetti fisso, e non è in grado di individuare oggetti che entrano o escono dalla scena.

Strettamente collegato al tracker di Veenman et al. è il metodo proposto da Shafique e Shah [58]. Entrambi, infatti, utilizzano l'Algoritmo Ungherese per il problema dell'assegnamento. Il problema dell'assegnamento per un grafo bipartito completo $G(N, A)$ consiste nel trovare un matching perfetto di peso minimo. Il matching è definito come $M \subseteq A$ tale che $\forall n \in N$ esiste non più di un arco in M che incide su n. L'Algoritmo Ungherese, proposto da Kuhn nel 1955 e perfezionato da Munkres nel 1957, risolve questo problema in tempo polinomiale ($O(n^3)$).

La principale differenza tra il metodo di Shafique e Shah [58] e il tracker GOA [10] è che quest'ultimo considera solo una corrispondenza tra due frame, mentre Shafique e Shah introducono un approccio "Multi-Frame" (considerato come caso generale della corrispondenza tra due frame). Questo consente di preservare la coerenza temporale della velocità e della posizione. La corrispondenza multi-frame viene affrontata come un problema di teoria dei grafi, dove per ogni punto si determina il cammino unico ottimo $P_i = (x^0, \ldots, x^k)$ per i frame da 0 a k.

2.1.2 Metodi probabilistici

I metodi probabilistici tengono conto dell'incertezza di misura e dell'approssimazione dei modelli nel determinare la corrispondenza. Essi utilizzano una rappresentazione a stati per modellare caratteristiche dell'oggetto come posizione, velocità, accelerazione: le informazioni riguardanti l'oggetto sono rappresentate da una sequenza di stati $x_1, \ldots, x_n$, e il passaggio da uno stato al successivo avviene tenendo conto del rumore bianco W.

Filtro di Kalman

Uno dei metodi probabilistici più popolari nell'ambito dell'object tracking è il filtro di Kalman [37], una soluzione ricorsiva al problema di data filtering nello spazio discreto. La versione originale dell'algoritmo è di tipo lineare, mentre la versione estesa (*Extended Kalman Filter*, *EKF*) permette di linearizzare sistemi non lineari partendo da media e covarianza correnti, utilizzando le serie di Taylor [20]. Ciò che accumuna tutte le versioni del filtro di Kalman è il fatto che la distribuzione di probabilità del sistema stimato è considerata di tipo gaussiano.

Il filtro di Kalman è essenzialmente un insieme di equazioni che implementano uno stimatore articolato in due passi: *predizione* e *correzione*. Questo stimatore è ottimale nel senso che minimizza la covarianza d'errore stimata (quando si riscontrano determinate condizioni) [20]. Il passo di predizione usa il modello degli stati per predire il nuovo stato delle variabili:

$$\bar{X}^t = \mathbf{D}X^{t-1} + W \tag{2.1}$$

$$\bar{\Sigma}^t = \mathbf{D}\Sigma^{t-1}\mathbf{D}^T + Q^t \tag{2.2}$$

dove $\bar{X}^t$ e $\bar{\Sigma}^t$ sono lo stato e la covarianza previsti al tempo t, $\mathbf{D}$ è la matrice di transizione degli stati, W è il rumore e Q la sua covarianza. Il passo di correzione utilizza le osservazioni correnti Z^t per aggiornare lo stato dell'oggetto:

$$K^t = \bar{\Sigma}^t\mathbf{M}^T[\mathbf{M}\bar{\Sigma}^t\mathbf{M}^T + R^t]^{-1} \tag{2.3}$$

$$X^t = \bar{X}^t + K^t \underbrace{[Z^t - \mathbf{M}\bar{X}^t]}_{v} \tag{2.4}$$

$$\Sigma^t = \bar{\Sigma}^t - K^t\mathbf{M}\bar{\Sigma}^t \tag{2.5}$$

dove v è detta innovazione di misurazione o residuo, $\mathbf{M}$ è la matrice di misura, K è il guadagno di Kalman (una matrice $n \times m$ scelta in modo da minimizzare l'errore di covarianza a posteriori [20]).

Il filtro di Kalman (sia nella versione di base che in quella estesa) ha trovato nel corso degli anni diverse applicazioni nell'ambito dell'object tracking. Esso ha avuto ad esempio un ruolo fondamentale nei metodi proposti da Broida e Chellappa [60], Rosales e Sclaroff [55], Beymer e Konolige [14], Cuevas et al. [17].

Particle Filter

La limitazione principale del filtro di Kalman consiste nell'assunzione che le variabili di stato siano di tipo gaussiano. Questa limitazione viene meno se si utilizza invece un *particle filter* [59]. Il particle filter (metodo prescelto in questo lavoro e descritto dettagliatamente nel Capitolo 4) è un filtro ricorsivo che rappresenta la funzione di densità a posteriori con un insieme di campioni che individuano possibili stati del sistema, a ciascuno dei quali è associato un peso che ne definisce l'importanza. L'algoritmo di particle filtering alla base della maggior parte delle applicazioni viene definito SIS (Sequential Importance Sampling). Esistono delle varianti di questo algoritmo chiamate SIR (Sequential Importance Resampling), ASIR (Auxiliary Sampling Importance Resampling), RPF (Regularised Particle Filter) [59].

Dopo una prima fase in cui vengono generati i campioni iniziali, il particle filter esegue iterativamente (come il filtro di Kalman) i due passi di predizione e correzione:

predizione: I campioni usati nel passo precedente vengono propagati, generando in questo modo i nuovi campioni correnti. La propagazione avviene applicando un modello di moto del tipo $\mathbf{s}_t = A\ \mathbf{s}_{t-1} + \mathbf{w}_{t-1}$, dove $\mathbf{w}_{t-1}$ rappresenta il rumore gaussiano.

correzione: Ad ogni campione viene assegnato un peso, in base a una funzione di similarità con il modello dell'oggetto.

Una volta assegnati tutti i pesi, essi vengono normalizzati. La posizione corrente dell'oggetto è stimata considerando la somma pesata di tutti i campioni al passo corrente.

Tracking simultaneo di più oggetti

Esistono metodi orientati al tracking simultaneo di più oggetti, per renderlo più semplice ed efficiente. I due metodi principali sono detti *Joint Probability Data Association Filtering* (JPDAF) e *Multiple Hypothesis Tracking* (MHT) [11]. Il metodo JPDAF utilizza N tracce, che corrispondono a sequenze di misure generate dallo stesso oggetto. Le m misure vengono assegnate alle N tracce. Si calcola quindi la somma pesata delle probabilità a posteriori che le misure i siano generate dagli oggetti assegnati alle tracce l:

$$v^l = \sum_{i=1}^{m} \beta_i^l v_{i,l} \tag{2.6}$$

Il problema principale di JPDAF è il fatto che, essendo esso progettato per seguire un numero fisso di oggetti, il suo funzionamento è compromesso nel caso di cambiamento del numero di oggetti nella scena (causato ad esempio dall'entrata o dall'uscita di oggetti dal campo visivo). Il Multiple Hypothesis Tracking risolve questo problema. L'idea su cui si basa il MHT è che la corrispondenza potrebbe non essere determinata correttamente considerando solo due frame: si considerano quindi più ipotesi di corrispondenza lungo più frame, stimando poi il nuovo stato dell'oggetto utilizzando l'insieme di ipotesi più probabile. Ad ogni iterazione dell'algoritmo le ipotesi correnti vengono valutate mediante una misura di distanza, introducendo così nuove ipotesi per il passo successivo. Ogni misura può appartenere a uno qualsiasi degli oggetti presenti sulla scena, compresi quindi eventuali oggetti non presenti nel frame precedente.

2.2 Kernel Tracking

Il Kernel Tracking consiste nel determinare il movimento di un oggetto da un frame al successivo, rappresentato generalmente in forma parametrica (ad esempio una traslazione). Yilmaz et al. [1] operano due ulteriori divisioni all'interno di questa categoria, in base al tipo di rappresentazione degli oggetti utilizzata: modelli basati su *template e densità* e modelli *multiview*.

2.2.1 Modelli basati su template e densità

L'approccio più comune in questa categoria è detto *template matching*. Esso è un metodo di tipo brute force che cerca in tutta l'immagine nel frame corrente un'area simile al template dell'oggetto, definito nel frame precedente. Il confronto avviene mediante un'opportuna misura di similarità. Per costruire il template vengono solitamente utilizzate informazioni sull'intensità di colore, anche se questo causa una notevole sensibilità del modello ai cambiamenti di illuminazione. Per ovviare a questo problema, Birchfield [5] propone un template basato sul gradiente dell'immagine. La principale limitazione del template matching è il suo notevole costo computazionale, dovuto alla ricerca di tipo brute force.

Oltre ai template possono essere usati altri tipi di rappresentazione, come istogrammi colore o misture in regioni rettangolari o ellittiche. I metodi principali di kernel tracking basato su template e densità sono Mean Shift [15], KLT [32], Layering [22].

Mean-shift Tracker

Il Mean Shift Tracker utilizza una rappresentazione basata su istogrammi, calcolati in regioni circolari. Invece di effettuare una ricerca brute force, viene utilizzata una procedura di tipo mean-shift. La similarità tra istogrammi viene misurata dal coefficiente di Bhattacharyya [4]:

$$\sum_{u=1}^{b} P(u)Q(u)$$

dove b è il numero di bin dell'istograma. L'algoritmo di mean-shift tracking può essere riassunto nei seguenti passi [63]:

1. Inizializzare il modello colore target $\hat{q}_u$:

$$\hat{q}_u = C \sum_{i=1}^{n} k(\|x_i^*\|^2)\delta[b(x_i^*) - u] \tag{2.7}$$

2. Calcolare il modello colore del candidato target $\hat{p}_u$:

$$\hat{p}_u(y) = C_h \sum_{i=1}^{n_h} k\left(\left\|\frac{y - x_i}{h}\right\|^2\right) \delta[b(x_i) - u] \tag{2.8}$$

3. Aggiornare i pesi per ogni pixel nella finestra target:

$$w_i = \sum_{u=1}^{m} \sqrt{\frac{\hat{q}_u}{\hat{p}_u(\hat{y}_0)}} \delta[b(x_i) - u] \tag{2.9}$$

4. Calcolare la nuova posizione, sommando al centro corrente il risultato dell'equazione:

$$\hat{y}_1 = \frac{\sum_{i=1}^{n_h} x_i w_i g\left(\left\|\frac{\hat{y}_0 - x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n_h} w_i g\left(\left\|\frac{\hat{y}_0 - x_i}{h}\right\|^2\right)} \tag{2.10}$$

5. Ripetere i passi 2-4 nel frame corrente fino alla convergenza (fino a quando $\hat{y}_1 = 0$).

6. Caricare il frame successivo, e ripartire dal passo 2.

KLT Tracker

Il tracker KLT tiene traccia della posizione degli oggetti attraverso la determinazione di *interest point*. Essi sono punti dell'immagine con particolari caratteristiche che ne determinano l'invarianza rispetto a cambiamenti di illuminazione e rotazione. L'uso degli interest point non è limitato all'object tracking, ma si estende a molte applicazioni di computer vision. In letteratura, oltre al KLT, si trovano vari metodi per la determinazione di interest point: *Harris detector* [7], *SIFT detector* [42], *Moravec interest operator* [50].

Il KLT detector, così come l'Harris detector, calcola le derivate prime (I_x, I_y) dell'immagine nelle direzioni x e y, valutando quindi per ogni pixel, in un piccolo intorno, la matrice:

$$M = \begin{pmatrix} \Sigma I_x^2 & \Sigma I_x I_y \\ \Sigma I_x, I_y & \Sigma I_y^2 \end{pmatrix} \tag{2.11}$$

La matrice M è invariante a rotazioni e traslazioni. Si calcola quindi R usando il minimo autovalore di M, $\lambda_{\min}$, e si determinano i possibili interest point applicando una soglia a R. Infine, tra i possibili interest point vengono eliminati quelli che si trovano in posizioni adiacenti.

Il tracker KLT calcola iterativamente la traslazione (du, dv) di una regione (patch) centrata in un interest point:

$$\begin{pmatrix} \Sigma I_x^2 & \Sigma I_x I_y \\ \Sigma I_x I_y & \Sigma I_y^2 \end{pmatrix} \begin{pmatrix} du \\ dv \end{pmatrix} = \begin{pmatrix} \Sigma I_x I_t \\ \Sigma I_x I_t \end{pmatrix} \tag{2.12}$$

Una volta determinata la nuova posizione dell'interest point, viene valutata la qualità della patch tracciata calcolando la trasformazione affine tra le due regioni corrispondenti in due frame consecutivi:

$$\begin{pmatrix} x\prime \\ y\prime \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{2.13}$$

Se la differenza tra le due regioni è piccola la caratteristica viene mantenuta anche nei frame successivi, altrimenti viene eliminata. In questo modo l'algoritmo continua a tenere traccia solo degli interest point la cui traslazione è seguita correttamente.

Layering

L'idea di questo metodo è suddividere l'immagine in diversi strati (layer): uno che identifica lo sfondo e uno per ogni oggetto da seguire. Ogni layer, identificato da regioni di forma ellittica, è definito da un modello di moto (traslazione e rotazione) e un modello di intensità (gaussiano). Per ogni pixel si calcola la probabilità p_l di appartenere al layer l, in base alle caratteristiche di moto e di forma dell'oggetto corrispondende. Ai pixel non identificabili con nessun layer viene assegnata una probabilità uniforme p_b (background probability), che viene successivamente combinata con la probabilità p_a (appearance probability) per ottenere la stima finale del layer. I parametri del modello (Φ_t, Θ_t, A_t) sono stimati iterativamente utilizzando un algoritmo di expectation-maximization (descritto nella Sezione 4.6.2).

2.2.2 Modelli Multiview

I modelli multiview considerano l'oggetto da diversi punti di vista. Affinché questo sia possibile, il modello dell'oggetto deve essere costruito offline, prima di effettuarne il tracking. I principali metodi di tracking che fanno uso di modelli multiview sono *EigenTracking* e *Support Vector Tracking*.

Il metodo EigenTracking [46] costruisce una rappresentazione dell'oggetto nell'autospazio (eigenspace), partendo da un insieme di immagini di training, che riprendono l'oggetto da diverse angolazioni. Questa rappresentazione permette di seguire l'oggetto anche in presenza di cambiamenti

di visuale. In particolare, Black e Japson utilizzano questo metodo per tracciare e riconoscere i gesti di una mano in movimento (Figura 2.1).

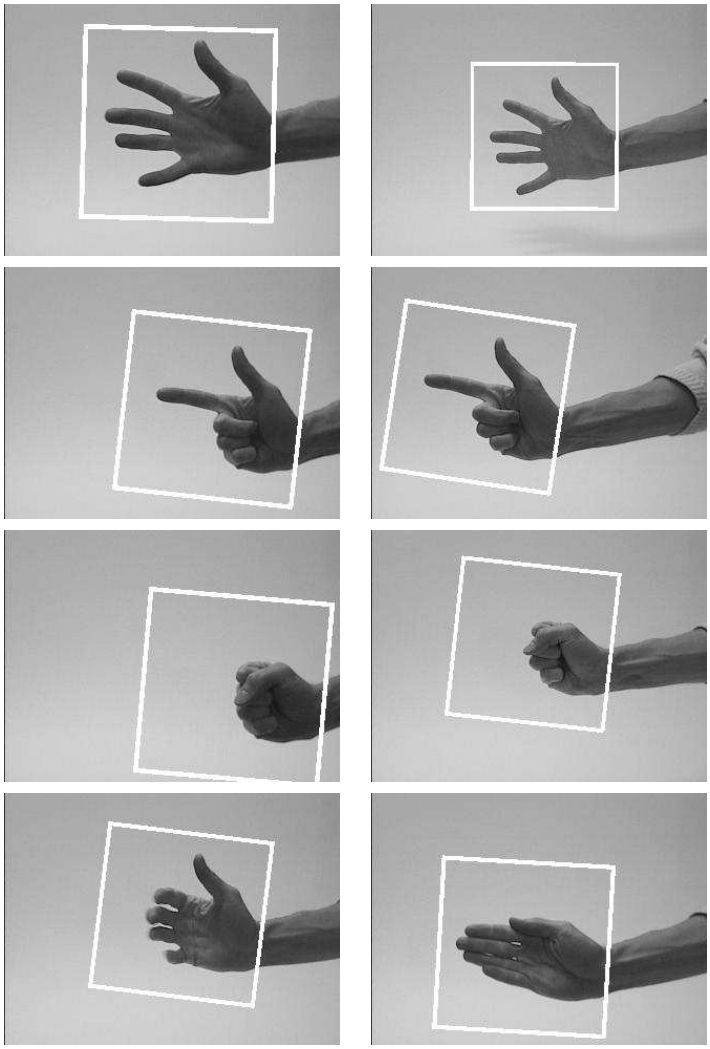


Figura 2.1: Tracking e riconoscimento di gesti di una mano con il metodo EigenTracking.

Il Support Vector Tracking (SVT) [3] è simile al metodo EigenTracking, ma invece di usare una rappresentazione nell'autospazio utilizza un classificatore SVM (Support Vector Machine) [6]. Dato un insieme di esempi positivi e negativi, il classificatore SVM trova l'iperpiano migliore che separa le due classi. Si considerano varie ipotesi sulla posizione dell'oggetto nel frame corrente: i support vector sono le ipotesi più vicine all'iperpiano separatore (come mostrato in Figura 2.2).

In Figura 2.3 è mostrato un esempio di tracking con l'algoritmo SVT.

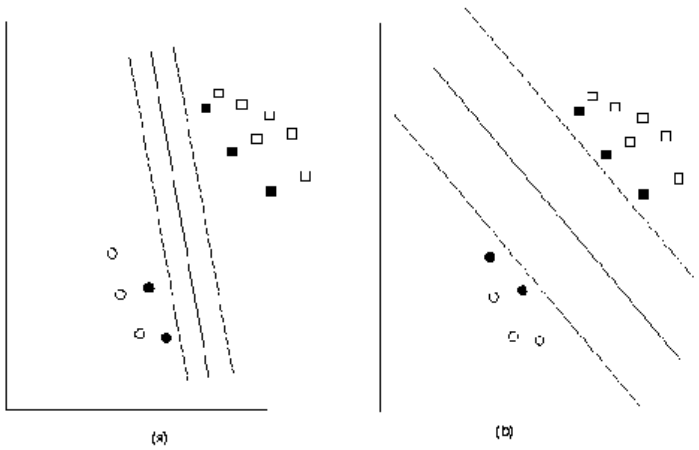
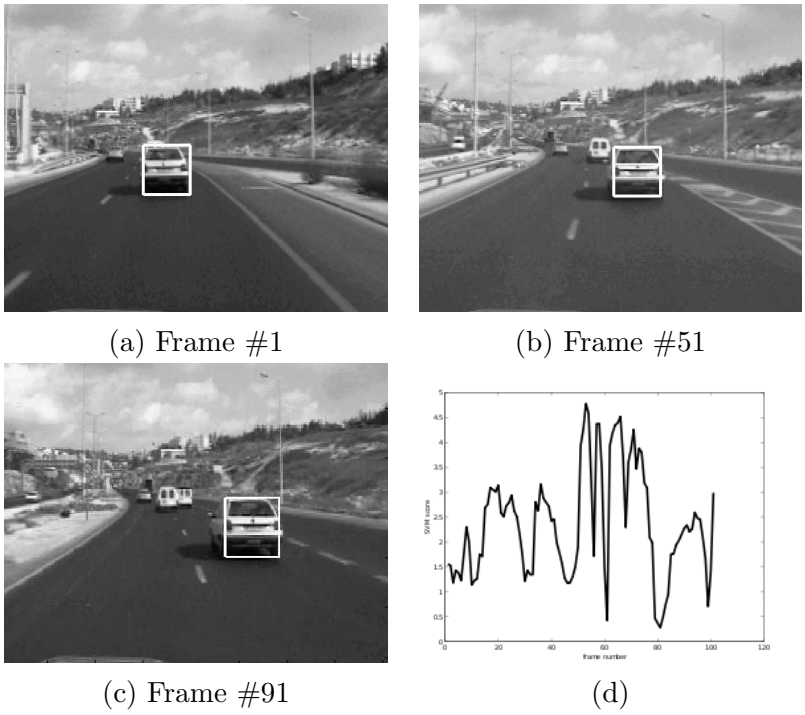


Figura 2.2: Iperpiano separatore con basso (a) e alto (b) margine. Le aree più vicine all'iperpiano, evidenziate in nero, rappresentano i support vector.



(a) Frame #1 (b) Frame #51

(c) Frame #91 (d)

Figura 2.3: Tracking con il metodo SVT. I frame (a),(b),(c) sono parte di una sequenza di 101 frame. In (d) è mostrato il punteggio SVM (sull'asse y) durante la sequenza dei frame (sull'asse x).

2.3 Silhouette Tracking

Questa categoria di algoritmi di tracking è rivolta all'inseguimento di oggetti non rappresentabili attraverso semplici forme geometriche. Yilmaz et al. [1] individuano due forme principali di silhouette tracking: *shape matching* e *contour tracking*.

2.3.1 Shape Matching

Con un approccio simile al template matching (Sezione 2.2.1), si costruisce un modello dinamico della silhouette dell'oggetto (chiamato *blob*), confrontandolo ad ogni frame con l'immagine corrente mediante una misura di similarità. Questo avviene nell'ipotesi che la forma dell'oggetto subisca traslazioni nello spazio da un frame al successivo.

Il tracker proposto da Huttenlocher et al. [13] utilizza la distanza di Hausdorff come misura di similarità con un modello basato sugli *edge* (bordi) dell'immagine. La distanza di Hausdorff tra due insieme di punti A, B,

definita come:

$$H(A,B) = \max\{h(A,B), h(b,A)\} \tag{2.14}$$

dove $h(A,B) = \sup_{a\in A}\inf_{b\in B}\|a-b\|$, individua gli edge per cui il grado di matching è inferiore. Questo permette di distinguere le parti dell'immagine meno soggette a cambiamenti di forma durante il movimento (ad esempio, nel caso di una persona che cammina, testa e torso).

Sato e Aggarwal [35] propongono un metodo che utilizza una trasformata chiamata TSV (temporal spatio-velocity). L'immagine $K_n(x)$ viene inizialmente pesata tramite una finestra esponenziale, ottenendo:

$$L_{np}(x,n) = K_n(x) \cdot F_{np}(n) \tag{2.15}$$

dove

$$F_{np}(n) = \begin{cases} (1-e^{-\lambda})e^{\lambda(n-n_p)} & \text{se } n \leq n_p \text{ (passato o corrente)} \\ 0 & \text{se } n > n_p \text{ (non definito: futuro)} \end{cases} \tag{2.16}$$

e λè la costante di tempo. Quindi, si costruisce l'immagine TSV applicando la trasformata di Hough a $L_{np}(x,n)$ (viene in realtà usata una versione semplificata della trasformata, per ridurre i tempi di computazione):

$$V_{np}(p,v) = \sum_{n=-\inf}^{n_p} \{K_n(v(n-n_p)+p)\}(1-e^{-\lambda})e^{\lambda(n-n_p)} \tag{2.17}$$

All'immagine trasformata viene quindi imposta una soglia, allo scopo di ottenere blob con posizioni e velocità istantanee simili. Il matching è basato sul moto, considerando traiettorie che corrispondono a schemi di *single interaction unit* (SIU). Le SIU sono una semplificazione di interazioni complesse, espresse come cambiamenti tra stati di quiete e di moto.

Kang et al. [33] utilizzano una rappresentazione basata su istogrammi comprendenti colore e bordi dell'immagine. In questo metodo gli istogrammi non vengono generati nel modo tradizionale, bensì a partire da cerchi concentrici centrati in un insieme di punti di controllo, situati nel più piccolo cerchio che contiene il blob dell'oggetto. Utilizzando cerchi concentrici vengono incorporate nell'istogramma anche informazioni sulla posizione, cosa non possibile con gli istogrammi tradizionali. La rappresentazione ottenuta risulta invariante a cambiamenti di scala, rotazioni e traslazioni. Per effettuare il matching è possibile utilizzare diverse misure di similarità, come ad esempio la distanza di Bhattacharyya [4].

2.3.2 Contour Tracking

Questa classe di algoritmi (chiamata anche Active Contour) effettua il tracking considerando l'evoluzione della posizione dei contorni dell'oggetto da un frame al successivo. I metodi appartenenti a questa categoria si distinguono soprattutto per le modalità con cui avviene l'evoluzione del modello del contorno.

Isard e Blake [45] propongono un modello a stati, applicando il concetto di particle filter (Sezione 2.1.2) all'aggiornamento dello stato del contorno dell'oggetto. L'algoritmo di particle filtering utilizzato è chiamato *CONDENSATION* (da "conditional density propagation"). Per inizializzare il particle filter, il modello viene sottoposto a una fase di training, in cui i contorni vengono stimati considerando frame consecutivi. Le immagini sono quindi rappresentate da un insieme di curve, rappresentate al tempo t da una curva parametrica $r(s,t)$. Le misure vengono calcolate lungo la normale rispetto ai contorni (come mostrato in Figura 2.4).

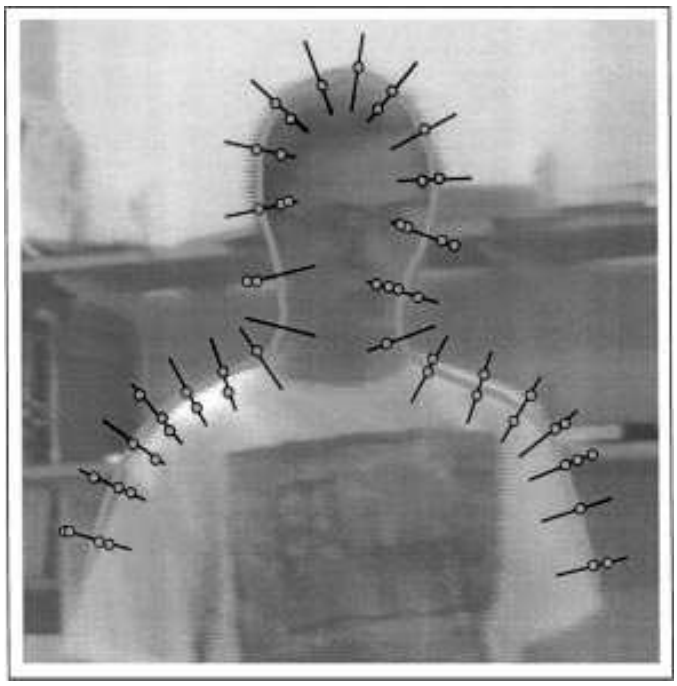


Figura 2.4: Rappresentazione dei contorni nell'algoritmo CONDENSATION.

Il tracker proposto da Bertalmio et al. [43] traccia i contorni dell'immagine minimizzando l'energia di contorno, definita in termini di gradiente temporale. L'evoluzione del contorno da un frame al successivo è ottenuta

per mezzo del vincolo di luminosità costante, determinato in base al flusso ottico:

$$I^{t+1}(x, y) - I^t(x - u, y - v) = 0 \tag{2.18}$$

dove I rappresenta l'immagine, t il tempo, (u, v) il vettore che rappresenta il flusso nelle direzioni (x, y). L'obiettivo è determinare u e v iterativamente per tutti i contorni dell'immagine. Vengono usati due tipi di energia, una per il tracking dei contorni (E_t) e una per il morphing dell'intensità (E_m):

$$E_m(\Gamma) = \int_0^1 E_{\text{im}}(\mathbf{v})ds \tag{2.19}$$

$$E_t(\Gamma) = \int_0^1 E_{\text{ext}}(\mathbf{v})ds \tag{2.20}$$

dove E_{ext} è calcolata in base a E_m. Vengono minimizzate contemporaneamente la funzione di morphing intensity, che minimizza i cambiamenti di intensità tra frame successivi:

$$\frac{\partial F(x, y)}{\partial t} = \nabla I_t(x, y)\|\nabla F(x, y)\| \tag{2.21}$$

e l'equazione di tracking dei contorni:

$$\frac{\partial \phi(x, y)}{\partial t} = \nabla I_t(x, y)\vec{n_F}\vec{n_\phi}\|\nabla \phi(x, y)\| \tag{2.22}$$

Ad esempio, se $\nabla I_t(x, y) \gg 0$, allora il contorno si muove con massima velocità lungo la direzione normale, e $I^{t-1}(x, y)$ è trasformato in $I^t(x, y)$. Invece, se $\nabla I_t(x, y) \approx 0$, allora la velocità di evoluzione sarà pari a zero [1].

Ronfard [56] propone un modello calcolato localmente, negli intorni del contorno dell'immagine, senza considerare il flusso ottico. L'energia locale è basata su modelli stazionari formulati come distanze di Ward. La distanza di Ward, che può essere considerata una misura del contrasto dell'immagine [27], non ha una definizione analitica. Per questo l'evoluzione dei contorni avviene considerandone l'intorno, con tecniche euristiche.

Capitolo 3

Impostazione del problema

Machines have less problems. I'd like to be a machine.

Andy Warhol

Il problema di cui si occupa l'object tracking è concettualmente semplice: mantenere traccia del movimento di un oggetto in una sequenza di fotogrammi. Il metodo più intuitivo per individuare la posizione di un oggetto consiste nell'utilizzare il cosiddetto *bounding box* (detto anche *minimum bounding rectangle, MBR*), un rettangolo che racchiude l'oggetto. Esso è individuato dalle sue coordinate $\min(x)$, $\max(x)$, $\min(y)$, $\max(y)$. In Figura 3.1 è mostrato un esempio di bounding box applicato a un essere umano.

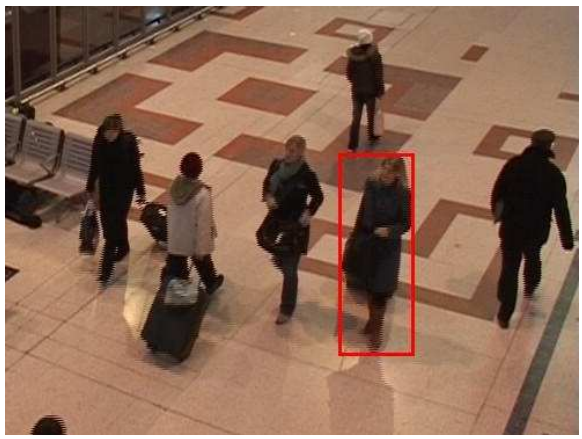


Figura 3.1: Esempio di bounding box. Le coordinate del rettangolo denotano la posizione della persona nel fotogramma corrente.

Per poter seguire la posizione dell'oggetto è dunque necessario conoscere il suo bounding box iniziale. Nell'affrontare il problema si è assunto che esso sia noto a priori. L'algoritmo di tracking consiste quindi nel determinare per ogni fotogramma f_t con $t = 1 \dots n$ le coordinate del suo bounding box, a partire dalle coordinate nel fotogramma precedente f_{t-1}.

Oltre ad assumere che la posizione iniziale sia nota, vengono considerati validi i seguenti vincoli:

- Le immagini vengono riprese da una sola telecamera, con inquadratura fissa.
- Le dimensioni del bounding box sono fisse. Si presuppone che l'oggetto non si muova eccessivamente lungo la profondità rispetto all'inquadratura.
- Non ci sono cambiamenti repentini di illuminazione (come ad esempio l'accensione o lo spegnimento di una lampada).
- L'oggetto non cambia improvvisamente posizione o velocità.

Riassumendo, il problema di object tracking affrontato consiste nel mantenere traccia delle coordinate del bounding box di uno o più oggetti in una sequenza di fotogrammi, dati i vincoli sopracitati. In assenza delle condizioni previste da tali vincoli il comportamento del sistema non è prevedibile. Per quanto riguarda il tracking simultaneo di più oggetti, non è prevista la gestione di urti e interazioni tra i diversi oggetti. Questi eventi, infatti, provocherebbero cambiamenti radicali e istantanei delle traiettorie degli oggetti entrati in contatto, violando i vincoli del problema.

Capitolo 4

Progetto logico della soluzione del problema

Tra il dire e il fare c'è di mezzo "e il".

Elio e le Storie Tese

In questo capitolo verrà descritto il sistema di tracking realizzato, valutando le diverse alternative prese in considerazione e motivando le scelte effettuate (anche con l'ausilio di diagrammi e pseudocodice).

Si partirà da una descrizione generale dell'algoritmo di particle filtering utilizzato, indicandone i passi principali. Saranno quindi analizzati nel dettaglio i singoli passi, evidenziando la funzione e la struttura logica di ciascuno di essi. Particolare attenzione sarà dedicata ai due modelli utilizzati per rappresentare gli oggetti in movimento: il modello istogramma colore e il modello SMOG. Per entrambi verranno illustrate le tecniche di inizializzazione, confronto ed eventuale aggiornamento.

4.1 Particle Filter

Il particle filter appartiene alla famiglia dei metodi Monte Carlo Sequenziali (SMC), ed è spesso utilizzato per la stima degli stati di sistemi con densità di probabilità non gaussiana [59, 38]. Per stimare la pdf, di cui non esiste in generale una rappresentazione analitica, viene implementato un filtro Bayesiano ricorsivo, che utilizza una simulazione di tipo Monte Carlo (basata sulla generazione di campioni non correlati tra loro). Il vettore X_t descrive lo stato dell'oggetto seguito, mentre il vettore Z_t rappresenta le osservazioni $\{\mathbf{z}_1, \ldots, \mathbf{z}_t\}$ fino al tempo t. Si approssima la distribuzione di probabilità con un set di campioni $S = \{(\mathbf{s}^{(n)}, w^{(n)}) | n = 1 \ldots N\}$. Ad ogni campione $\mathbf{s}$ è associato un peso w, che rappresenta la probabilità che l'oggetto si trovi nello stato rappresentato da quel campione. Questi pesi sono poi normalizzati, in modo che $\sum_{n=1}^{N} w^{(n)} = 1$.
Lo stato dell'oggetto viene stimato ad ogni passo con:

$$E[S] = \sum_{n=1}^{N} w^{(n)} \mathbf{s}^{(n)} \tag{4.1}$$

I campioni vengono chiamati *particle*. Nonostante i metodi SMC fossero già noti da diversi anni, il primo uso del termine "particle" risale al 1996, ad opera di Kitagawa [39]. L'espressione "particle filter", invece, è stata usata per la prima volta da Carpenter et al. nel 1999 [30]. Altri nomi che questi metodi hanno assunto in letteratura sono bootstrap filtering [51], interacting particle approximations [49], condensation algorithm [45, 28], survival of the fittest [57].

L'algoritmo di particle filtering implementato nel sistema di tracking realizzato (illustrato in Figura 4.1) è composto da alcuni semplici passi. Come prima cosa vengono generate e inizializzate le particle, sulla base di opportuni criteri. Quindi, per ogni fotogramma preso in esame, esse vengono propagate secondo un modello di moto prestabilito. A questo punto si calcolano i pesi dei singoli campioni, confrontando ciascuno di essi con il modello dell'oggetto costruito in precedenza (i modelli utilizzati saranno discussi nelle sezioni 4.5 e 4.6). I pesi vengono poi normalizzati, per poter effettuare la stima della posizione dell'oggetto nel fotogramma corrente. Infine, prima di procedere con il fotogramma successivo, avviene la fase di resampling (descritta nella sezione 4.4).

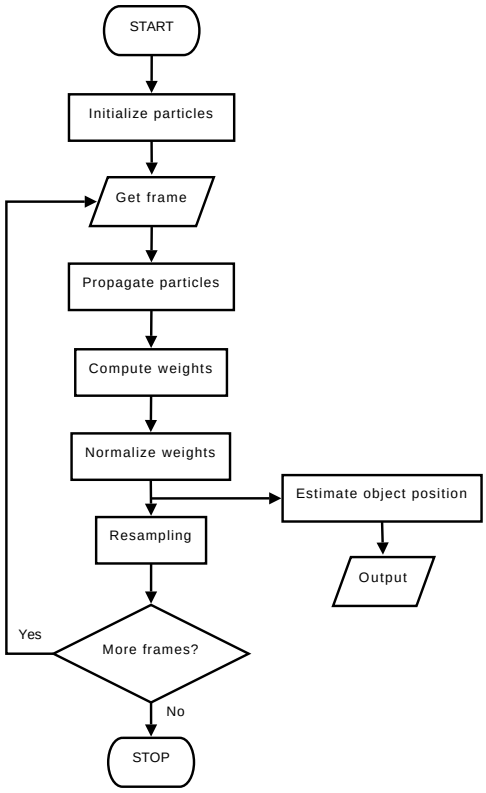


Figura 4.1: Diagramma a blocchi del particle filter.

Nelle prossime sezioni saranno analizzate singolarmente le varie fasi dell'algoritmo, concludendo con la descrizione dei modelli e degli spazi colore utilizzati. Dato che per ogni passo dell'algoritmo sono possibili diverse soluzioni implementative, laddove ne sia stata realizzata anche solo una è stata lasciata aperta la possibilità di aggiornamenti futuri.

4.2 Inizializzazione delle particle

Operando nell'assunzione che la posizione iniziale dell'oggetto sia nota, le particle possono essere generate e posizionate casualmente all'interno del bounding box che contiene l'oggetto (come mostrato in Figura 4.2). E' evidente come questo rappresenti un vantaggio rispetto al caso in cui, non conoscendo la posizione iniziale dell'oggetto, le particle debbano essere sparse per tutta l'immagine.

Figura 4.2: Inizializzazione delle particle all'interno del bounding box.

Un altro aspetto importante è la scelta del numero di particle. Se le particle generate fossero in numero molto basso, infatti, si avrebbero ampie fluttuazioni nella posizione stimata da un frame al successivo, che renderebbero poco preciso il tracking (o, nel peggiore dei casi, si perderebbe traccia della posizione dell'oggetto). Se invece venisse generato un numero molto alto di particle, cosa buona da un punto di vista teorico (maggiore è il numero di campioni, più vicina all'ottimo è la stima), si avrebbe un impatto negativo sulle prestazioni del sistema, rendendolo troppo lento per un uso funzionale. E' stato quindi determinato, sulla base di risultati sperimentali, un numero di particle da usare come valore predefinito. Dato che si tratta di un valore arbitrario, che rappresenta una sorta di media sperimentale, esso potrebbe non essere ottimale in ogni situazione. Per questo motivo è stata data all'utente la possibilità di modificarlo a suo piacimento, purché resti in un intervallo delimitato da un limite inferiore e un limite superiore prestabiliti.

Una volta generate e posizionate le particle, a ognuna di esse viene assegnata una velocità casuale (che può assumere valori positivi o negativi) lungo i due assi x, y.

Ogni particle sarà quindi definita dai seguenti valori:

- Coordinata x della posizione attuale.
- Coordinata y della posizione attuale.
- Velocità v_x lungo l'asse x.
- Velocità v_y lungo l'asse y.
- Peso w (normalizzato in modo che $\sum_{i=1}^{N} w(i) = 1$).

4.3 Propagazione delle particle

Per poter predire i possibili movimenti dell'oggetto da un frame al successivo (in tutte le direzioni possibili) è necessario che le particle vengano propagate ad ogni iterazione dell'algoritmo. La propagazione consiste nel determinare il nuovo stato di tutte le particle, ovvero la loro nuova posizione, calcolata in base a un determinato modello di moto (questo perchè la posizione e la velocità di un oggetto non possono cambiare radicalmente da un frame al successivo). Le posizioni e le velocità delle particle lungo gli assi x e y, quindi, vengono aggiornate applicando le equazioni del moto.

La scelta di un buon modello di moto è essenziale per poter tracciare correttamente la posizione dell'oggetto. Bisogna tenere conto anche del rumore di processo, modellizzato secondo [16] e [38] con valori casuali di una distribuzione gaussiana. Lo stato delle particle viene perciò aggiornato secondo un'equazione del tipo:

$$\mathbf{s}_t = A\ \mathbf{s}_{t-1} + \mathbf{w}_{t-1} \tag{4.2}$$

dove A rappresenta la componente deterministica del modello e $\mathbf{w}_{t-1}$ il rumore. Nel filtro realizzato la propagazione avviene secondo un moto uniformemente accelerato, dove l'accelerazione a è rappresentata dal rumore (quindi calcolata come valore casuale gaussiano):

$$\mathbf{s}_t = \mathbf{s}_{(t-1)} + \mathbf{v}_{s(t-1)}\ \Delta t + \frac{1}{2}\ a\ \Delta t^2 \tag{4.3}$$

Oltre alla posizione, viene aggiornata ad ogni passo anche la velocità di ogni particle, utilizzando nuovamente il rumore gaussiano come accelerazione:

$$\mathbf{v}_t = \mathbf{v}_{(t-1)} + a\ \Delta t \tag{4.4}$$

Infine, se la nuova posizione di una particle è tale da posizionarla fuori dai margini dell'immagine (quindi se una delle sue coordinate è minore di

zero o maggiore della dimensione massima), la particle in questione viene riposizionata all'interno dell'immagine. L'effetto della fase di propagazione è mostrato in Figura 4.3.

Figura 4.3: Propagazione delle particle. Si nota che esse si allontanano dal bounding box corrente dell'oggetto, in modo da poterne seguire lo spostamento nel frame successivo.

4.4 Resampling

Uno dei problemi maggiori legati al particle filtering è il fenomeno della *degenerazione*. Questo fenomeno porta il sistema, dopo poche iterazioni, a una situazione in cui quasi tutto il peso è concentrato su una sola particle, mentre tutte le altre hanno peso prossimo allo zero. Come dimostrato da Doucet et al. [2], la degenerazione è inevitabile, perché la varianza dei pesi continua a crescere nel tempo.

Il metodo più efficace per contrastare il fenomeno della degenerazione è il resampling (ricampionamento). Esso consiste nell'eliminare le particle con pesi bassi a favore di quelle con pesi alti, in modo da mantenere stabile il sistema ad ogni passo. Per questo scopo sono stati presentati e si trovano in letteratura diversi approcci: multinomial resampling [51], stratified resampling [39], residual resampling [41], systematic resampling [39, 59]. Nel particle filter realizzato è stato implementato un algoritmo di systematic resampling, che è computazionalmente efficiente ($O(n)$) e teoricamente superiore agli altri in presenza di distribuzioni uniformi [31]. L'algoritmo, descritto in Algorithm 1, è di facile implementazione. Si costruisce la funzione di distribuzione cumulativa (CDF), sommando progressivamente i pesi di tutte le particle (come mostrato in Figura 4.4). Quindi si genera un valore casuale u_1 compreso tra 0 e N^{-1} e a ogni iterazione j ci si muove da quel punto con un passo di $1/N$. Seguendo la CDF, si eliminano le particle per

cui il valore è inferiore a u_j, facendole convergere sulla prima con CDF maggiore. I pesi vengono quindi ridistribuiti in modo uniforme. In Figura 4.5 è mostrato l'effetto del resampling, confrontando la posizione delle particle prima e dopo l'esecuzione dell'algoritmo.

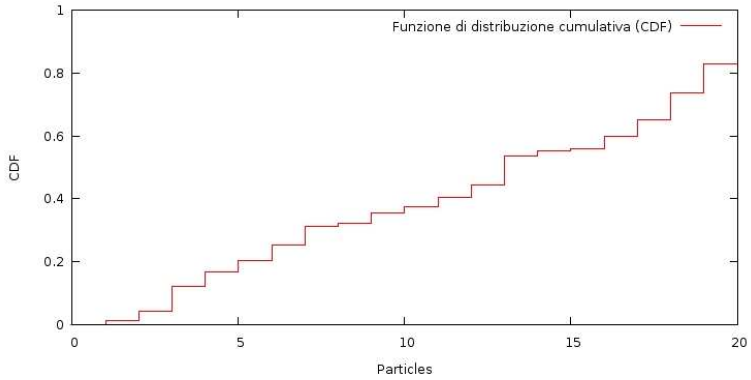
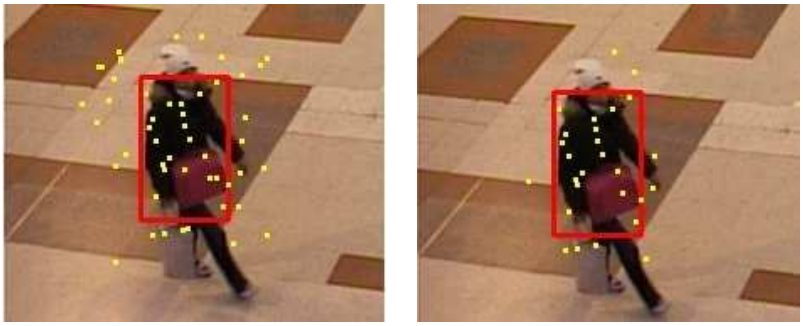


Figura 4.4: Esempio di costruzione della CDF con 20 particle.

Algorithm 1 Algoritmo di Systematic Resampling.

$c_1 \leftarrow 0$ {Inizializzazione della CDF}
for i=2 to N **do**
  $c_i \leftarrow c_{i-1} + w_i$ {Costruzione della CDF}
end for
$i \leftarrow 1$ {Si parte dall'inizio della CDF}
$u_1 \leftarrow \text{random}[0, N^{-1}]$ {Scelta di un valore iniziale casuale tra 0 e $1/N$}
for $j = 1$ to N **do**
  $u_j = u_1 + N^{-1}(j-1)$ {Movimento lungo la CDF}
  while $u_j > c_i$ **do**
    $i \leftarrow i + 1$
  end while
  $x_j \leftarrow x_i$ {Assegna i valori al nuovo campione}
  $w_j \leftarrow N^{-1}$ {Assegna il peso al nuovo valore}
end for

Il problema principale legato al resampling è il fatto che, eliminando alcuni campioni e ripetendone altri, se ne riduce la diversità. Questo problema, noto come *sample impoverishment*, è particolarmente sentito in caso di rumore di processo molto basso [59], caso in cui tutte le particle potrebbero collassare in un singolo punto. Sono stati recentemente proposti diversi approcci per contrastare il fenomeno di sample impoverishment (anche se nel nostro caso il problema non è sembrato così evidente da prenderne in



(a) (b)

Figura 4.5: Posizione delle particle prima (a) e dopo (b) il resampling.

considerazione l'implementazione). Tra questi figurano metodi basati su algoritmi genetici [40], Support Vector Regression [21] e l'algoritmo *resample-move* [19]. In [8], invece, è proposto un particle filter modificato, chiamato Regularised Particle Filter (RPF), che ricampiona utilizzando un'approssimazione continua della densità a posteriori (a differenza del normale particle filter, che utilizza un'approssimazione discreta) [59].

4.5 Modello Istogramma Colore

L'istogramma colore è uno dei modelli più utilizzati in questo genere di applicazioni, in quanto capace di fornire una certa robustezza in caso di oggetti non rigidi, rotazioni e occlusioni parziali [38]. Per realizzare l'istogramma, la distribuzione del colore nell'immagine viene discretizzata in un numero prestabilito di *bin*: ogni pixel viene assegnato al bin corrispondente, in base al suo colore. In questo caso sono stati utilizzati bin di dimensioni $8 \times 8 \times 8$ nello spazio RGB e di dimensioni $8 \times 8 \times 4$ nello spazio HSV (per una descrizione degli spazi colore, si veda la sezione 4.7). La rappresentazione della distribuzione di colore tramite istogramma, quindi, è data dal numero di pixel di ogni intervallo di colore definito dai singoli bin (valori che vengono poi normalizzati). In Figura 4.6 è mostrato un esempio di istogramma colore nei singoli canali R,G,B (con 8 bin ciascuno).

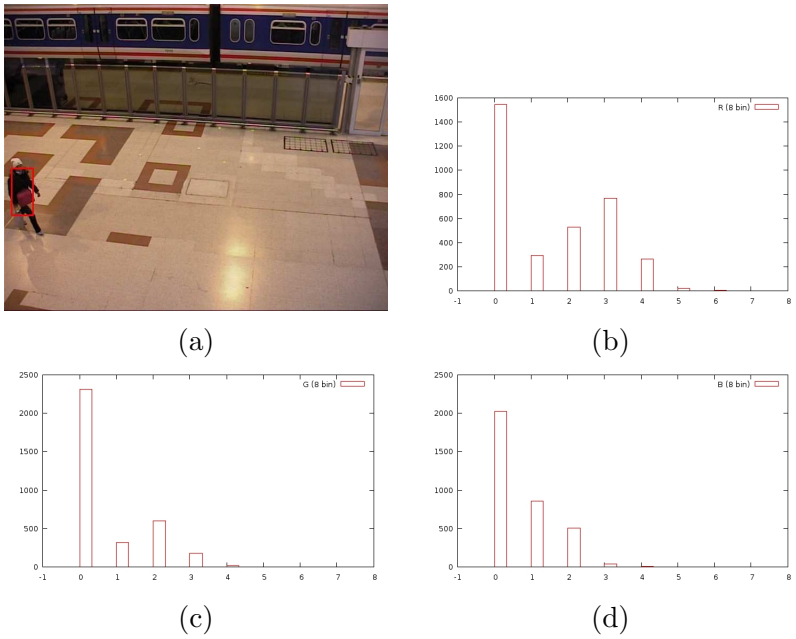


Figura 4.6: Istogramma colore dell'immagine (a) nelle componenti R (b), G (c), B (d). L'istogramma riguarda solo l'area racchiusa nel bounding box.

4.5.1 Inizializzazione del modello

L'inizializzazione avviene dopo aver ricevuto il primo fotogramma e le coordinate del bounding box in cui si trova inizialmente l'oggetto. Si utilizza infatti come modello su cui basare il tracking l'istogramma colore relativo alla sola parte dell'immagine in cui è contenuto l'oggetto da seguire. L'algoritmo con cui viene costruito l'istogramma (nel caso di spazio colore RGB con bin di 8x8x8) è mostrato in Algorithm 2.

Algorithm 2 Costruzione dell'istogramma colore nello spazio RGB (utilizzando bin di dimensioni 8x8x8).

Inizializza l'istogramma con valori nulli
for all pixels in bounding box **do**
  $r = (pixel.R)/32$ {255 valori, 8 bin}
  $g = (pixel.G)/32$ {255 valori, 8 bin}
  $b = (pixel.B)/32$ {255 valori, 8 bin}
  Incrementa il valore dell'istogramma alla posizione r, g, b
end for
Normalizza l'istogramma

4.5.2 Calcolo dei pesi

Il calcolo dei pesi avviene mediante un confronto tra istogrammi. Si considera, per ogni particle, l'intorno di dimensioni uguali a quelle del bounding box che racchiude l'oggetto. In quest'area si costruisce l'istogramma colore (sempre utilizzando l'algoritmo mostrato in Algorithm 2). Per poter confrontare i due istogrammi si deve utilizzare una misura di similarità appropriata. Una misura molto popolare per confrontare due distribuzioni di probabilità discrete $p(u)$ e $q(u)$ è la distanza di Bhattacharyya [4, 38, 18, 36]:

$$d = \sqrt{1 - \rho[p, q]} \tag{4.5}$$

dove $\rho[p, q]$ è detto coefficiente di Bhattacharyya, ed è definito come:

$$\rho[p, q] = \sum_{x \in X} \sqrt{p(x)q(x)} \tag{4.6}$$

Viene quindi calcolato il coefficiente di Bhattacharyya tra l'istogramma usato come modello dell'oggetto e quelli corrispondenti alle singole particle, assegnando in questo modo un peso a ciascuna di esse. Infatti, più i due istogrammi sono simili, più il coefficiente di Bhattacharyya sarà elevato (il coefficiente tra due istogrammi normalizzati identici è uguale a 1). L'algoritmo è mostrato in Algorithm 3.

Algorithm 3 Calcolo dei pesi tramite coefficiente di Bhattacharyya nello spazio colore RGB (istogrammi colore con 8x8x8 bin).

```
for i = 1 to N do
  H_1 ← istogramma colore che rappresenta l'oggetto
  Costruisci l'istogramma colore della particle s_i e memorizzalo in H_2
  w_i ← 0
  for r = 1 to 8 do
    for g = 1 to 8 do
      for b = 1 to 8 do
        w_i = w_i + sqrt(H_1(r,g,b) H_2(r,g,b))
      end for
    end for
  end for
end for
```

4.6 Modello SMOG

Il modello SMOG [23], crea una mistura di gaussiane in un modello congiunto spazio-colore. Secondo gli autori, questo modello garantisce una maggiore robustezza rispetto al normale istogramma colore, perché contiene anche informazioni sulla posizione dell'oggetto. Il modello SMOG viene quindi proposto come un'alternativa al più classico istogramma, i cui principali svantaggi (evidenziati dagli stessi autori) sono:

- Viene ignorata completamente la componente spaziale degli oggetti seguiti. Per questo motivo, due oggetti con colori simili possono essere facilmente confusi se si trovano l'uno vicino all'altro.
- Dato che l'aspetto dell'oggetto è ridotto a un istogramma globale, la misura di similarità non è abbastanza discriminante.
- La costruzione dell'istogramma non è computazionalmente efficiente, costituendo un collo di bottiglia.

4.6.1 Descrizione del modello

Il modello SMOG può essere considerato come un'evoluzione del modello a misture di gaussiane (MOG), il quale contiene informazioni solo sulla distribuzione di colore. La mistura di gaussiane è una combinazione convessa di K distribuzioni normali (gaussiane):

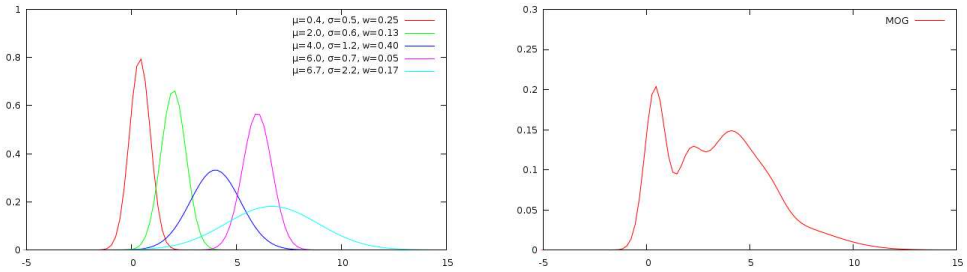
$$MOG(x) = \sum_{k=1}^{K} w_k \; g_{(\mu_k, \Sigma_k)}(x) \tag{4.7}$$

dove w_k sono numeri positivi la cui somma è uguale a 1. In Figura 4.7 è mostrata una mistura di cinque gaussiane monovariate.

Nel modello SMOG si fa uso di una mistura di gaussiane che comprende, oltre all'informazione sul colore, anche un'informazione sulla distribuzione nello spazio. Si indica rispettivamente con $S_i = (x_i, y_i)$ la componente spaziale e con $C_i = \{C_i^j\}_{j=1,\dots,d}$ la componente relativa al colore, dove d corrisponde al numero di canali (ad esempio, negli spazi RGB e HSV, $d = 3$). La caratteristica globale del pixel x_i è data dal prodotto cartesiano di C_i e S_i (assunte indipendenti tra loro):

$$x_i = (S_i, C_i) \tag{4.8}$$

La media e la covarianza della k-esima classe della mistura di gaussiane sono $\mu_{t,k} = (\mu_{t,k}^S, \mu_{t,k}^C)$ e $\Sigma_{t,k} = (\Sigma_{t,k}^S, \Sigma_{t,k}^C)$. La densità stimata nel punto x_i



(a) (b)

Figura 4.7: Mistura di gaussiane monovariate. Le cinque gaussiane in (a) vengono sommate a seconda dei loro pesi, in modo da ottenere la mistura (b).

è data dalla somma delle K classi della mistura moltiplicate per il rispettivo peso $w_{(t,k)}$, dove ciascuna delle classi è il prodotto delle due gaussiane nello spazio e nel colore:

$$p_0(x_i) = \sum_{k=1}^{K} w_{t,k} \frac{e^{-\frac{1}{2}(S_i-\mu^S_{t,k})^T(\Sigma^S_{t,k})^{-1}(S_i-\mu^S_{t,k})}}{2\pi|\Sigma^S_{t,k}|^{1/2}} \frac{e^{-\frac{1}{2}(C_i-\mu^C_{t,k})^T(\Sigma^C_{t,k})^{-1}(C_i-\mu^C_{t,k})}}{(2\pi)^{d/2}|\Sigma^S_{t,k}|^{1/2}} \tag{4.9}$$

Vengono quindi moltiplicate tra loro una gaussiana bivariata e una d-variata. Nel nostro caso si ha sempre $d = 3$, in quanto gli spazi colore utilizzati (RGB, HSV, rgI) sono composti da tre canali. Il valore di K è un intero compreso fra 3 e 8, la cui scelta è lasciata disponibile all'utente.

4.6.2 Inizializzazione del modello

Da un punto di vista più generale, l'uso delle misture di gaussiane nell'ambito della computer vision può essere considerato un caso particolare di *data clustering*. L'apprendimento non supervisionato del modello avviene solitamente per mezzo di un algoritmo chiamato Expectation-Maximization (EM) [61, 47, 53]. Affinchè questo algoritmo sia efficace, è necessario che esso sia inizializzato con valori opportuni. Nel nostro caso, i valori iniziali sono determinati eseguendo l'algoritmo K-means [48] prima dell'algoritmo EM.

K-means

L'algoritmo K-means ha come idea di base l'individuazione dei K cluster attraverso i loro punti centrali, chiamati centroidi. Si parte posizionando i centroidi in modo casuale, procedendo poi iterativamente nell'assegnare

ciascun pixel dell'immagine al centroide più vicino. Le posizioni dei centroidi vengono poi ricalcolate come valore medio dei pixel assegnati a ciascun cluster. Il tutto viene ripetuto fino a quando i centroidi non si muovono più, quando cioè si è raggiunta la convergenza. Come misura di distanza tra pixel e centroidi è stata utilizzata la distanza euclidea:

$$D(P,Q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{4.10}$$

dove $P = (p_1, p_2, \ldots, p_n)$ e $Q = (q_1, q_2, \ldots, q_n)$. Dato che non ci interessano i valori effettivi delle distanze, ma soltanto un confronto tra esse al fine di stabilire qual è la minore, è possibile togliere la radice quadrata dalla formula della distanza euclidea, ottenendo così la distanza euclidea quadrata:

$$D(P,Q) = \sum_{i=1}^{n}(p_i - q_i)^2 \tag{4.11}$$

Questo crea un vantaggio dal punto di vista computazionale, semplificando notevolmente il calcolo. In Algorithm 4 è illustrato l'algoritmo utilizzato.

Algorithm 4 K-means clustering

```
Inizializza K centroidi C1_k ← 0
Posiziona K centroidi C2_k in modo casuale all'interno del bounding box
while C1 ≠ C2 (i centroidi si sono spostati) do
  C1 ← C2
  for all pixels p in bounding box do
    for k = 1 to K do
      d_k = (p_R − C1_kR) + (p_G − C1_kG) + (p_B − C1_kB)
      if d_k < d_min then
        d_min ← d_k
        Assegna il pixel al cluster k
      end if
    end for
  end for
  for k = 1 to K do
    Assegna al cluster C2_k il valore medio dei pixel nel cluster k
  end for
end while
```

L'algoritmo k-means ha il pregio di convergere in poche iterazioni, ma non è detto che trovi la soluzione ottima. Per questo non è sufficiente a costruire il modello SMOG, ma è un'ottima base di partenza per l'algoritmo di Expectation-Maximization.

Expectation-Maximization

L'algoritmo EM dà ottimi risultati nella stima di densità, aspetto fondamentale del clustering. Il problema di stima di densità può essere definito [61] come: dato un insieme di N punti in D dimensioni, $x_1, \ldots, x_N \in \mathbb{R}^D$ e una famiglia F di funzioni di densità di probabilità in $\mathbb{R}^D$, trovare la densità di probabilità $f(x) \in F$ che con maggiore probabilità ha generato i punti dati. Nel nostro caso, la famiglia di funzioni F è una mistura di gaussiane. Seguendo un approccio di stima a massima verosimiglianza, si definisce una funzione di likelihood:

$$\Lambda(X;\theta) = \prod_{n=1}^{N} \sum_{k=1}^{K} w_k \; g(x_n, \mu_k, \Sigma_k) \tag{4.12}$$

La stima di densità parametrica può quindi essere definita come:

$$\hat{\theta} = \arg\max_{\theta} \Lambda(X;\theta) \tag{4.13}$$

Per determinare $\hat{\theta}$ si utilizza l'algoritmo EM. Dopo aver inizializzato

$$\lambda_t = \{\mu_1(t), \mu_2(t), \ldots, \mu_K(t), \Sigma_1(t), \Sigma 2(t), \ldots, \Sigma_K(t), p_1(t), p_2(t), \ldots, p_K(t)\}$$

usando i valori determinati con l'algoritmo K-means, si eseguono iterativamente i due passi seguenti, fino alla convergenza:

Expectation (E)
Calcola i valori attesi di probabilità per ogni classe:

$$\begin{aligned} P(w_i|x_n, \lambda_t) &= \frac{p(x_n|w_i, \lambda_t) \; P(w_i|\lambda_t)}{p(x_n|\lambda_t)} \\ &= \frac{p(x_n|w_i, \mu_i(t), \Sigma_i(t)) \; p_i(t)}{\sum_{k=1}^{K} p(x_n|w_k, \mu_k(t), \Sigma_k(t)) \; p_k(t)} \end{aligned} \tag{4.14}$$

Maximization (M)
Calcola la nuova stima a massima verosimiglianza di μ_k basata sui valori attesi correnti (calcolati nel passo precedente):

$$\begin{aligned} \mu_i(t+1) &= \frac{\sum_{n=1}^{N} P(w_i|x_n, \lambda_t) \; x_k}{\sum_{n=1}^{N} P(w_i|x_n, \lambda_t)} \\ \Sigma_i(t+1) &= \frac{\sum_{n=1}^{N} P(w_i|x_n, \lambda_t)[x_n - \mu_i(t+1)][x_n - \mu_i(t+1)]^T}{\sum_{n=1}^{N} P(w_i|x_n, \lambda_t)} \\ p_i(t+1) &= \frac{1}{N} \sum_{n=1}^{N} P(w_i|x_n, \lambda_t) \end{aligned} \tag{4.15}$$

Questo algoritmo migliora ad ogni passo la parametrizzazione $\hat{\theta}$ stimata, fino a raggiungere un massimo locale [53]. Una volta raggiunta la convergenza (quando $\hat{\theta}$ non cambia sensibilmente rispetto al passo precedente), l'algoritmo termina.

Per quanto riguarda il calcolo di $p(x_n|w_i, \mu_i(t), \Sigma_i(t))$ nel passo E, esso consiste nel calcolo del valore di una distribuzione normale multivariata con parametri x_n, $\mu_i(t)$, $\Sigma_i(t)$:

$$f_x(x_1, \ldots, x_N) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{4.16}$$

In Algorithm 5 è mostrato lo pseudocodice dell'algoritmo EM utilizzato.

Algorithm 5 Expectation-Maximization

```
repeat
  for all K clusters do
    for all pixels in bounding box do
      Calcola P(w_i|x_n, λ_t) {E-step}
    end for
  end for
  for all K clusters do
    Calcola le nuove μ, Σ, p {M-Step}
  end for
until convergence
```

Una volta terminato, l'algoritmo EM dà come risultato i parametri del modello SMOG:

$$\{w_{t=1,k}^{O_t}, \mu_{t=1,k}^{S,O_t}, \mu_{t=1,k}^{C,O_t}, \Sigma_{t=1,k}^{S,O_t}, \Sigma_{t=1,k}^{C,O_t}\} \tag{4.17}$$

dove O_t indica l'oggetto target, ovvero l'oggetto da seguire, di cui è stato costruito il modello.

4.6.3 Calcolo dei pesi

Per calcolare la funzione di likelihood è necessario determinare i parametri $\{w_{t=1,k}^{O_v}, \mu_{t=1,k}^{S,O_v}, \mu_{t=1,k}^{C,O_v}, \Sigma_{t=1,k}^{S,O_v}, \Sigma_{t=1,k}^{C,O_v}\}$ per ogni oggetto candidato O_v (ovvero per ogni particle), in modo da poterli confrontare con quelli dell'oggetto effettivo (mostrati in Eq. 4.17). Per calcolare questi parametri si procede nel modo seguente:

1. Calcolare la distanza di Mahalanobis tra i pixel $\{x_i\}$ dell'oggetto candidato $O_v = \{x_i\}_{i=1,\dots,n}$ e ogni classe della mistura SMOG di O_t, nello spazio colore:

$$D_k^2(C_i, \mu_{t,k}^{C,O_t}, \Sigma_{t,k}^{C,O_t}) = (C_i - \mu_{t,k}^{C,O_t})^T \ (\Sigma_{t,k}^{C,O_t})^{-1} \ (C_i - \mu_{t,k}^{C,O_t}) \quad (4.18)$$

2. Etichettare i pixel per cui esiste almeno una $|D_k|_{k=1,\dots,K} \leq 2.5$ con il numero della classe SMOG per cui la distanza di Mahalanobis è minima. Gli altri pixel avranno etichetta zero.

$$\mathrm{LB}(x_i) = \arg\min_k |D_k| \quad (4.19)$$

3. Calcolare i parametri $\{w_{t=1,k}^{O_v}, \mu_{t=1,k}^{S,O_v}, \mu_{t=1,k}^{C,O_v}, \Sigma_{t=1,k}^{S,O_v}, \Sigma_{t=1,k}^{C,O_v}\}$ di O_v:

$$w_{t,k}^{O_v} = \frac{\sum_{i=1}^{N} \delta(\mathrm{LB}(x_i) - k)}{\sum_{k=1}^{K} \sum_{i=1}^{N} \delta(\mathrm{LB}(x_i) - k)} \quad (4.20)$$

$$\mu_{t,k}^{O_v} = (\mu_{t,k}^{S,O_v}, \mu_{t,k}^{C,O_v}) = \frac{\sum_{i=1}^{N} x_i \ \delta(\mathrm{LB}(x_i) - k)}{\sum_{i=1}^{N} \delta(\mathrm{LB}(x_i) - k)} \quad (4.21)$$

$$\Sigma_{t,k}^{O_v} = \frac{\sum_{i=1}^{N} (x_i - \mu_{t,k}^{O_v})^T (x_i - \mu_{t,k}^{O_v}) \ \delta(\mathrm{LB}(x_i) - k)}{\sum_{i=1}^{N} \delta(\mathrm{LB}(x_i) - k)} \quad (4.22)$$

La matrice di covarianza viene considerata diagonale, per semplicità. Con δ si intende la funzione delta di Kronecker, definita come:

$$\delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Per quanto riguarda le etichette $\mathrm{LB}(x_i)$, esse andrebbero calcolate per tutti i pixel di ogni particle. Questo però comporterebbe uno spreco, perché molti pixel, trovandosi associati a diverse particle, sarebbero etichettati più volte. Per evitare che ciò accada, le etichette sono calcolate in precedenza, considerando l'area coperta da tutte le particle. Per ogni singola particle, poi, vengono utilizzate le etichette precalcolate relative ai pixel corrispondenti.

Una volta determinati i parametri di O_v, si calcola la misura di similarità tra O_v e O_c:

$$\Lambda(O_t, O_v) = \sum_{k=1}^{K} \Lambda_{t,k}^{S} \Lambda_{t,k}^{C} \tag{4.23}$$

dove $\Lambda_{t,k}^{S}$ e $\Lambda_{t,k}^{C}$ sono rispettivamente la misura di similarità nello spazio e nel colore, calcolate come:

$$\Lambda_{t,k}^{S} = \exp\left\{-\frac{1}{2}(\mu_{t,k}^{S,O_v} - \mu_{t,k}^{S,O_t})^T(\hat{\Sigma}_{t,k}^{S})^{-1}(\mu_{t,k}^{S,O_v} - \mu_{t,k}^{S,O_t})\right\}$$

$$\Lambda_{t,k}^{C} = \min(w_{t,k}^{O_v}, w_{t,k}^{O_t})$$

dove $(\hat{\Sigma}_{t,k}^{S})^{-1} = (\Sigma_{t,k}^{S,O_v})^{-1} + (\Sigma_{t,k}^{S,O_t})^{-1}$.

La funzione di likelihood è data da:

$$L(Y_t|X_t) \propto \exp\left\{-\frac{1}{2\sigma_b^2}(1 - \Lambda(O_t, O_v))\right\} \tag{4.24}$$

In Algorithm 6 è mostrato l'algoritmo utilizzato per l'assegnamento dei pesi alle N particle.

Algorithm 6 Calcolo dei pesi

```
for all pixels in predicted samples area do
  for k=1 to K do
    d = distanza di Mahalanobis tra il pixel x_i e la classe SMOG k
    if d < 2.5 AND d < d_min then
      d_min = d
      LB(x_i) = k {Assegna l'etichetta k al pixel x_i}
    end if
  end for
end for
for all particles do
  Calcola il numero di pixel x_k assegnati a ogni etichetta
  Per ogni etichetta, calcola la somma delle componenti spaziali x e y
  for k=1 to K do
    Calcola μ_x, μ_y, Σ_x, Σ_y
    Calcola la misura di similarità Λ(O_t, O_v)
  end for
end for
```

4.6.4 Aggiornamento del modello

Ipotizzando che l'aspetto dell'oggetto (nelle sue distribuzioni di spazio e colore) non cambi radicalmente da un frame al successivo, il modello SMOG viene aggiornato dinamicamente, utilizzando un learning rate α. Si aggiorna quindi in modo *exponentially forgetting*, aggiungendo gradualmente nuove informazioni al modello. L'equazione utilizzata per l'aggiornamento dei pesi è la seguente [9]:

$$w_{k,t} = (1 - \alpha)w_{k,t-1} + \alpha(M_{k,t}) \tag{4.25}$$

dove $M_{k,t}$ è uguale a 1 per il modello che dà una corrispondenza, 0 per gli altri. Medie e varianze si aggiornano con:

$$\mu_t = (1 - \rho)\mu_{t-1} + \rho X_t \tag{4.26}$$

$$\sigma_t^2 = (1 - \rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^T(X_t - \mu_t) \tag{4.27}$$

dove

$$\rho = \alpha\eta(X_t|\mu_k, \sigma_k) \tag{4.28}$$

(η è una distribuzione di probabilità gaussiana, nel nostro caso di tipo SMOG).

4.7 Descrizione degli spazi colore utilizzati

L'algoritmo di tracking realizzato è stato implementato in tre diversi spazi colore: RGB, HSV, rgI. Essi sono descritti di seguito.

4.7.1 RGB

Lo spazio colore **RGB** (**R**ed, **G**reen, **B**lue) è uno spazio colore di tipo additivo, in cui i diversi colori vengono composti aggiungendo diverse quantità di rosso, verde, blu. I valori RGB sono rappresentati con un cubo nello spazio tridimensionale, come mostrato in Figura 4.8. In Figura 4.9, invece, sono mostrati i colori corrispondenti ai vertici del cubo.

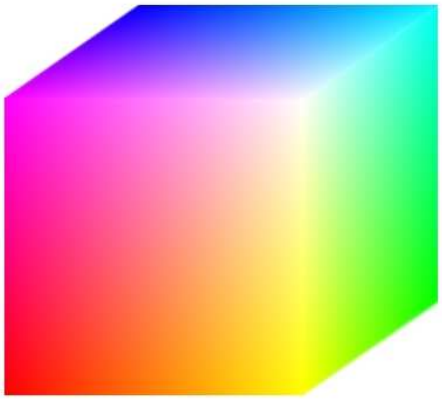


Figura 4.8: Cubo dei valori RGB. I tre colori (rosso, verde, blu) sono rappresentati rispettivamente lungo gli assi (x, y, z).

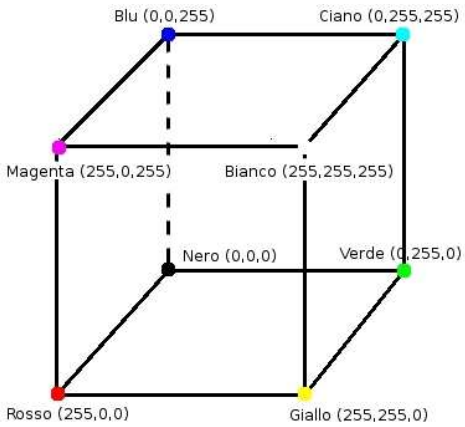


Figura 4.9: Colori corrispondenti ai vertici del cubo RGB.

Lo spazio RGB utilizzato ha una profondità di colore di 24 bit (8 bit per canale). I colori vengono quindi rappresentati con una terna di interi compresi tra 0 e 255, dove (0,0,0) rappresenta il nero (assenza di colore) e (255,255,255) rappresenta il bianco (somma di tutti i colori). In questo modo è possibile rappresentare 16, 7 milioni di colori, corrispondenti a quasi tutto lo spettro visibile.

L'istogramma colore nello spazio RGB è tipicamente calcolato utilizzando 8 bin per canale [38]. Dato che ogni canale ha 255 valori, ogni bin ne contiene 32.

4.7.2 HSV

Il modello colore **HSV** (**H**ue, **S**aturation, **V**alue) è una descrizione del colore nello spazio RGB più orientata alla percezione umana. Infatti, il colore viene descritto attraverso tre parametri che si avvicinano alla nostra prospettiva:

- Hue (H): indica la tonalità del colore (rosso, verde, blu, ...).
- Saturation (S): indica l'intensità del colore.
- Value (V): indicato a volte anche come Brightness (B), indica la luminosità del colore.

Il sistema di riferimento HSV è di tipo cilindrico, e i valori sono rappresentati con un cono all'interno del sistema di riferimento. Il valore di H è misurato dall'ampiezza dell'angolo intorno all'asse verticale, come mostrato in Tabella 4.1.

Tabella 4.1: Tonalità di colore nello spazio HSV

Angolo	**Colore**
0-60	Rosso
60-120	Giallo
120-180	Verde
180-240	Ciano
240-300	Blu
300-360	Magenta

Il raggio e l'altezza del cono corrispondono rispettivamente ai valori di S e V, e variano entrambi da 0 a 1. In Figura 4.10 è rappresentato il cono HSV.

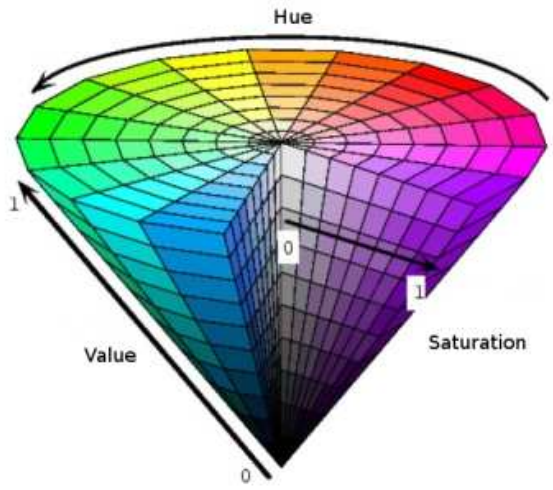


Figura 4.10: Cono HSV. I valori di H sono rappresentati da angoli rispetto all'asse verticale, mentre i valori di S e V si trovano rispettivamente sul raggio e sull'altezza del cono.

L'uso del modello HSV, rispetto al modello RGB, può rendere l'algoritmo meno sensibile ai cambiamenti di illuminazione [38]. In questo caso, nel realizzare l'istogramma, si utilizza un numero minore di bin per il canale V. Si è quindi deciso di utilizzare 8 bin per i canali H e S, e solo 4 bin per il canale V. Questo dovrebbe far sì che cambiamenti di illuminazione anche piuttosto consistenti non influiscano sulla composizione dell'istogramma, rendendo quindi il modello più robusto.

Per poter agire in questo modello colore è necessario poter convertire i valori RGB ricavati dall'immagine nei corrispondenti valori HSV. Per ogni pixel si considerano quindi i valori HSV ricavati applicando uno specifico algoritmo di conversione ai valori RGB. Tale algoritmo è mostrato in Algorithm 7.

Algorithm 7 Algoritmo di conversione da RGB a HSV

```
Determinare min (R, G, B) e max (R, G, B)
V ← max (R, G, B)
if V = 0 then
  H ← 0
  S ← 0
else
  S = {255 × (max (R, G, B) − min (R, G, B))} / V
  if S = 0 then
    H ← 0
  else
    if max (R, G, B) = R then
      H ← 43 × (G − B)/(max (R, G, B) − min (R, G, B))
    else if max (R, G, B) = G then
      H ← 85 + 43 × (B − R)/(max (R, G, B) − min (R, G, B))
    else if max (R, G, B) = B then
      H ← 171 + 43 × (R − G)/(max (R, G, B) − min (R, G, B))
    end if
  end if
end if
```

4.7.3 rgI

Lo spazio colore rgI è stato scelto [23] a causa della sua maggiore robustezza ai cambiamenti di illuminazione rispetto a RGB. La conversione da RGB a rgI è molto semplice [54]:

$$r = \frac{R}{R+G+B} \tag{4.29}$$

$$g = \frac{G}{R+G+B} \tag{4.30}$$

$$I = \frac{R+G+B}{3} \tag{4.31}$$

Le prime due componenti (r, g) sono quindi i valori di (R, G) normalizzati, mentre la componente I rappresenta l'intensità del colore (in realtà ne rappresenta solo un'approssimazione, in quanto i tre colori non sono percepiti esattamente con la stessa intensità dall'occhio umano).

Per lavorare in questo spazio colore è sufficiente convertire i valori RGB in rgI (utilizzando le formule mostrate in Eq. 4.29). L'unico problema in questa conversione è riscontrato in presenza di colore RGB nero (0,0,0). In questo caso, infatti, il calcolo di r e g non è possibile, in quanto le equazioni producono un risultato indeterminato (0/0). Questo caso deve quindi essere trattato separatamente: è stato deciso di assegnare valori rgI (0,0,0) quando si hanno valori RGB (0,0,0). L'algoritmo di conversione è mostrato in Algorithm 8.

Algorithm 8 Conversione da RGB a rgI

if $R = 0$ AND $G = 0$ AND $B = 0$ **then**
  $r \leftarrow 0$
  $g \leftarrow 0$
  $I \leftarrow 0$
else
  $r \leftarrow R/(R + G + B)$
  $g \leftarrow G/(R + G + B)$
  $I \leftarrow (R + G + B)/3$
end if

Capitolo 5

Architettura del sistema

Non posso giudicare il mio lavoro mentre lo compio. Devo fare come i pittori, arretrare e guardarlo da una certa distanza, che non sia però troppa. Quanta distanza? Chissà.

Blaise Pascal

Il sistema di tracking descritto nel Capitolo 4 è stato implementato utilizzando una struttura modulare. Questo ha permesso innanzitutto di semplificarne la realizzazione, scomponendo il problema in più semplici sottoproblemi secondo un classico approccio *top-down*. Un altro vantaggio della modularità del sistema è la possibilità di aggiungere nuove funzionalità (o modificare quelle esistenti) senza dover intervenire sulla sua struttura di base.

In questo capitolo è mostrata l'architettura globale del sistema, attraverso i moduli principali che ne definiscono la struttura logica. Vengono poi analizzati i singoli moduli, evidenziando le caratteristiche di ognuno (con eventuali dettagli implementativi) e le possibilità di espansioni future. La descrizione del sistema è coadiuvata da schemi che espongono le interazioni tra i diversi moduli.

5.1 Architettura generale

Considerando il sistema dal livello logico più alto, possiamo individuare quattro operazioni principali che esso deve compiere:

- Acquisizione dei fotogrammi da analizzare.
- Identificazione degli oggetti da seguire.
- Inseguimento dell'oggetto.
- Visualizzazione dei risultati.

In Figura 5.1 sono mostrati i moduli che svolgono le funzioni sopracitate, con le rispettive relazioni che li mettono in comunicazione.

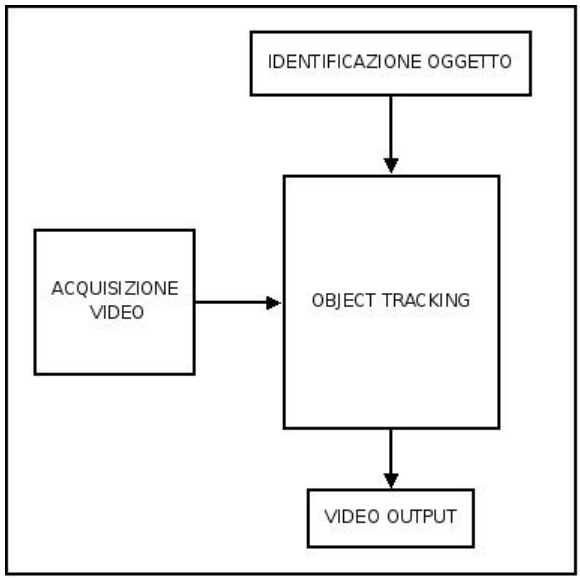


Figura 5.1: Struttura generale del sistema di tracking.

Il sistema è stato implementato utilizzando il linguaggio C, con l'ausilio di alcune liberie esterne. Esse sono indicate nelle sezioni seguenti, che descrivono i singoli moduli del sistema. Per ogni modulo sono specificate le librerie utilizzate, chiarendone le funzioni e l'utilità implementativa.

5.2 Acquisizione video

Per la realizzazione del modulo è stata utilizzata la libreria *OpenCV*, reperibile all'indirizzo `http://www.intel.com/technology/computing/opencv/`. Questa libreria, sviluppata da Intel, è open source e dispone di svariate funzioni associabili a molti ambiti della computer vision. Il suo uso è stato tuttavia limitato a operazioni di input/output.

Il modulo di acquisizione video implementato consiste nel semplice caricamento di immagini acquisite in precedenza, e ordinate in una specifica directory. Grazie alla versatilità della libreria OpenCV sarebbe possibile estendere le capacità di acquisizione a sequenze video o interfacciarsi direttamente con una videocamera.

5.3 Identificazione dell'oggetto

Il primo passo da compiere per identificare l'oggetto consiste nel determinare la sua posizione nel primo fotogramma della sequenza. Per semplicità si è assunto che essa sia nota a priori: la posizione dell'oggetto, sotto forma di coordinate del bounding box che lo racchiude, è ricevuta come input dall'utente.

Una volta individuata la posizione dell'oggetto è necessario costruirne un modello. Essendo disponbili diversi tipi di modelli, il modulo deve poter stabilire quale utilizzare. Tale compito è assolto da un'apposita variabile di controllo, che può assumere i seguenti valori:

- MODEL_HIST: modello istogramma colore.
- MODEL_SMOG: modello smog.

Se si volesse aggiungere un ulteriore modello sarebbe sufficiente assegnare un nuovo valore alla variabile di controllo, associandolo alla funzione di inizializzazione del nuovo modello.

5.4 Object tracking

Il modulo che svolge la funzione di object tracking è il vero e proprio motore dell'applicazione. Esso riceve in ingresso la sequenza di fotogrammi, elabora la stima della nuova posizione dell'oggetto e invia i risultati al modulo di output.

Il modulo di object tracking può essere suddiviso in due parti fondamentali: il particle filter, che si occupa materialmente dell'elaborazione statistica

e permette di tracciare la posizione dell'oggetto, e le funzioni di inizializzazione e calcolo di similarità relative ai diversi modelli, che permettono al particle filter di assegnare i pesi ai diversi campioni. Le due parti sono poste in comunicazione mediante una variabile di controllo che indica al particle filter quale modello utilizzare (Figura 5.2).

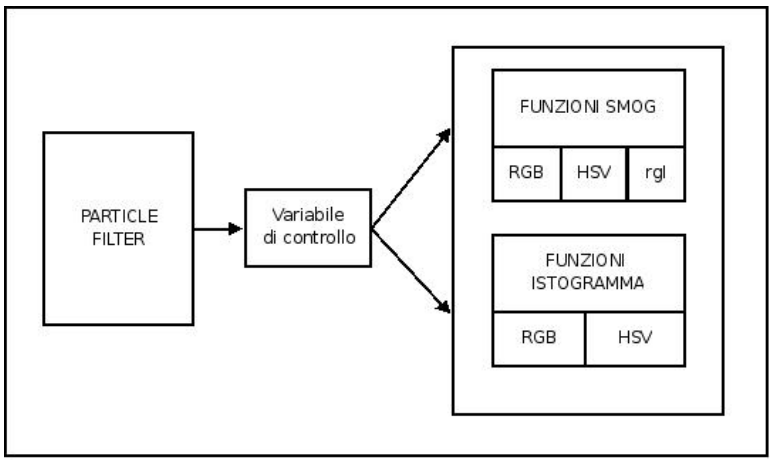


Figura 5.2: Struttura del modulo di object tracking. Si noti che le funzioni di inizializzazione e similarità dei vari modelli sono implementate nei diversi spazi colore.

La struttura interna del particle filter, descritta nel Capitolo 4, è a sua volta modulare: è quindi possibile modificare o aggiungere procedure per la propagazione e il ricampionamento delle particle.

Dal punto di vista implementativo, il modulo di object tracking (in particolare il particle filter) fa uso della libreria *GSL* (GNU Scientific Library), reperibile all'indirizzo `http://www.gnu.org/software/gsl/`.

5.5 Output

Il modulo di output utilizza la libreria OpenCV per visualizzare a schermo il fotogramma corrente, disegnando il bounding box nella posizione stimata. In questo modo è possibile vedere la sequenza video e comparare l'effettiva posizione dell'oggetto con quella stimata dal particle filter.

Capitolo 6

Realizzazioni sperimentali e valutazione

L'essere confutabile non è certo la minore attrattiva di una teoria; proprio per questo attira i cervelli più sottili.

Friedrich W. Nietzsche

Il sistema di tracking descritto nei capitoli precedenti è stato implementato nel linguaggio C (parte del codice sorgente è consultabile nell'Appendice A). Per valutare sperimentalmente il funzionamento dell'applicazione sono state utilizzate sequenze video appartenenti al PETS Data Set (Performance Evaluation of Tracking and Surveillance), reperibili presso l'indirizzo web `http://www.cvg.rdg.ac.uk/slides/pets.html`.

In questo capitolo saranno mostrati e discussi i risultati ottenuti, suddividendone la trattazione in base ai due modelli utilizzati (istogramma colore e SMOG). Per ognuno di essi saranno proposti esempi pratici, la cui analisi porterà all'identificazione di vantaggi e svantaggi derivanti dal loro utilizzo.

La parte conclusiva sarà dedicata all'analisi complessiva del sistema e dei suoi possibili sviluppi futuri.

6.1 Modello istogramma colore

Il primo test eseguito mostra l'istogramma colore applicato in condizioni favorevoli, considerando un'area dell'oggetto la cui distribuzione di colore differisce considerevolmente da quella dello sfondo. In questo caso il tracking avviene senza problemi, permettendo di seguire l'oggetto dall'inizio alla fine della sequenza. La staticità del modello, che potrebbe essere considerata un limite nell'affrontare situazioni generiche, in questo caso specifico si rivela utile nel consentire di superare senza difficoltà una situazione di lieve occlusione. In Figura 6.1 sono mostrati i risultati ottenuti.

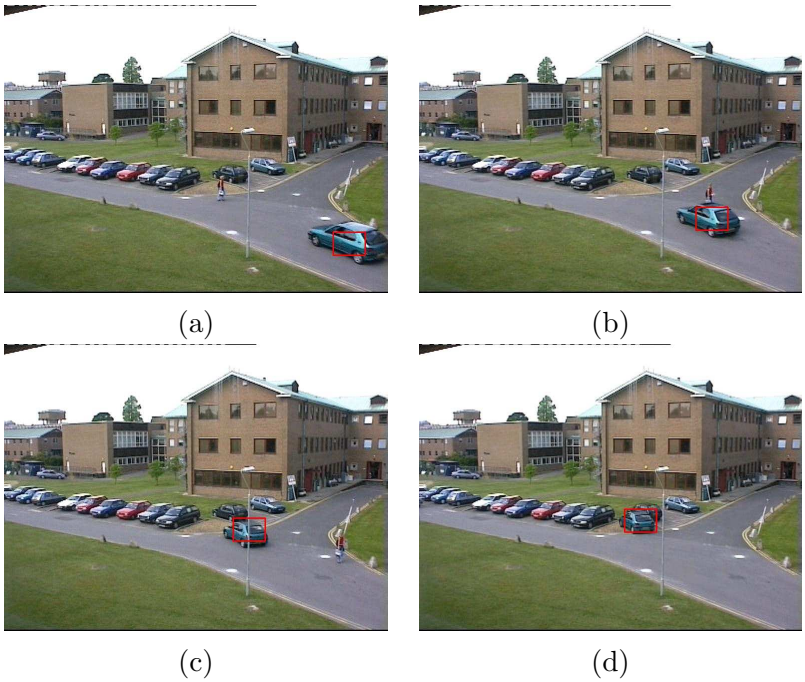
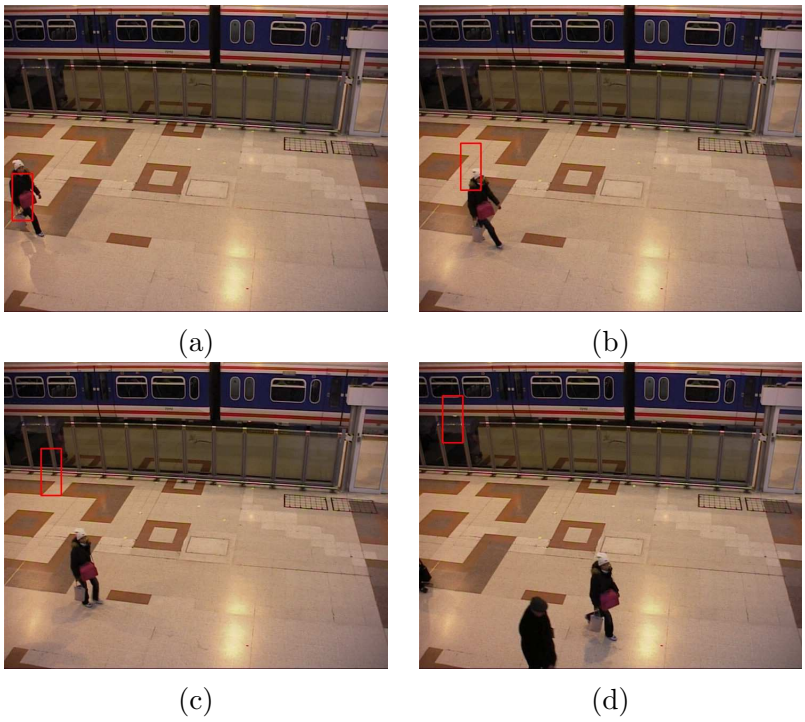


Figura 6.1: Inseguimento di un oggetto con modello istogramma colore. Si nota (c) che viene mantenuta traccia dell'oggetto anche in presenza di una parziale occlusione.

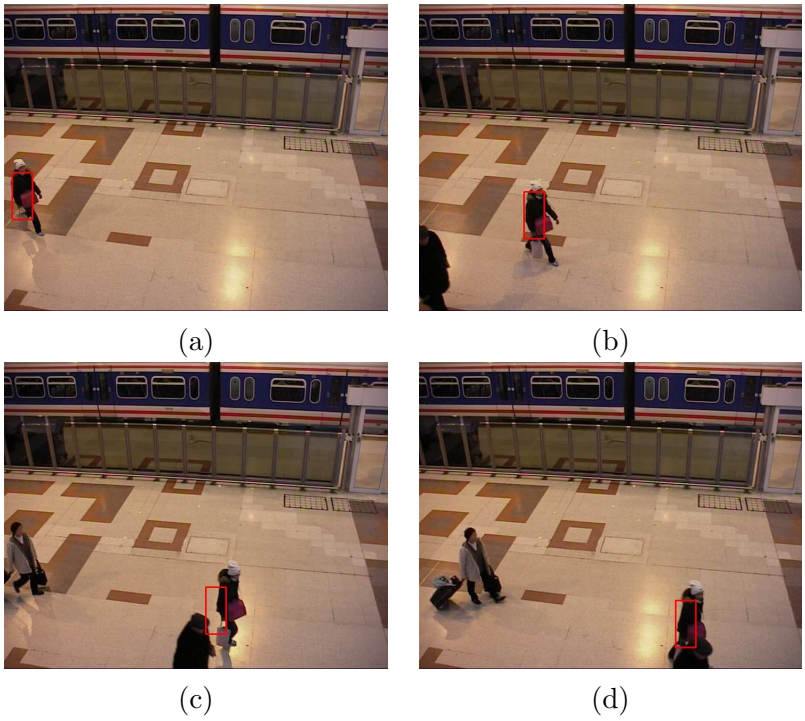
Il secondo test evidenzia la non adeguatezza del modello a gestire ogni tipo di situazione. Nel caso in questione la prova è ripetuta due volte, con risultati differenti. Nel primo caso (Figura 6.2) la posizione viene persa e il sistema diverge, fino a raggiungere un'altra area dai colori molto simili al modello. Nel secondo caso (Figura 6.3) utilizzando gli stessi parametri l'oggetto viene seguito per tutta la sequenza (pur con qualche difficoltà in presenza di un altro oggetto simile nelle vicinanze).



(a) (b)

(c) (d)

Figura 6.2: Malfunzionamento del modello istogramma colore. L'oggetto in (a) viene seguito fino a perderne traccia (b). A questo punto il sistema non ha più modo di recuperare l'oggetto, e diverge (c) fino a raggiungere un'area dai colori simili al modello iniziale (d).

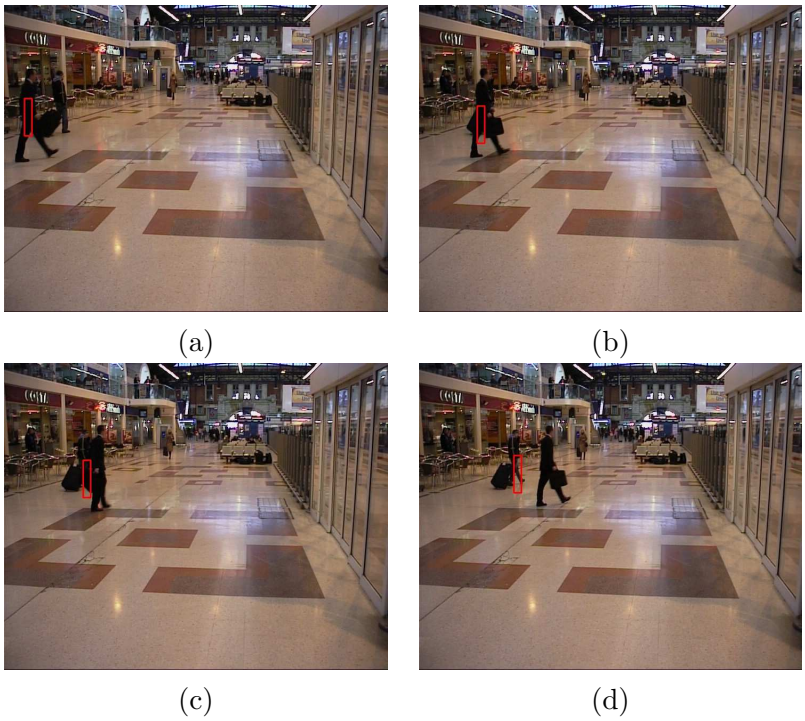


(a) (b)

(c) (d)

Figura 6.3: In questo caso l'oggetto viene seguito correttamente per tutta la sequenza, ma il sistema si trova in difficoltà in presenza di un oggetto dai colori simili nelle vicinanze (c).

Nel terzo test (Figura 6.4) viene mostrata una situazione in cui la sovrapposizione di due oggetti molto simili manda in crisi il sistema, che viene ingannato e continua a seguire il nuovo oggetto invece dell'originale. Questo è forse il limite principale dei modelli basati sul colore: l'impossibilità di distinguere due oggetti simili che si trovano in posizioni adiacenti.



(a) (b)

(c) (d)

Figura 6.4: In questa situazione le traiettorie dei due oggetti si sovrappongono (b). Il sistema non riesce più a riconoscere l'oggetto da seguire (c) e prende la decisione di seguire l'oggetto sbagliato (d).

L'istogramma colore mostra chiaramente i suoi limiti se messo di fronte ad una situazione complessa, come quella in Figura 6.5. Lo scopo è seguire il movimento di due diverse persone (che indossano abiti di colore simile) in una scena affollata. Il modello non si rivela sufficientemente robusto, perdendo traccia della posizione delle due persone dopo pochi fotogrammi.

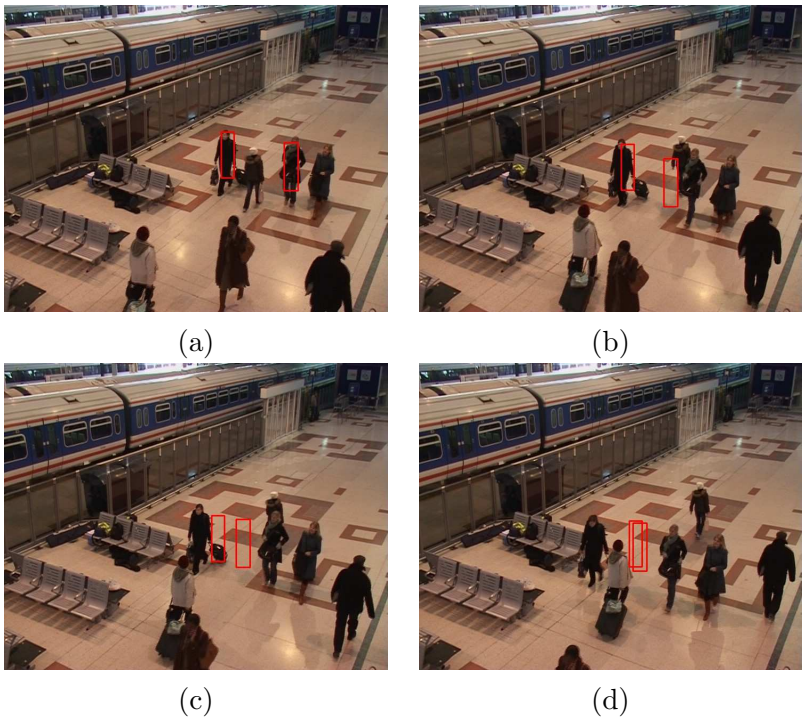


Figura 6.5: Scena complessa per cui l'istogramma colore si rivela inadeguato. Si parte dal fotogramma iniziale della sequenza (a). Dopo 20 fotogrammi si è perso traccia dell'oggetto di destra (b). Dopo altri 10 fotogrammi anche l'oggetto di sinistra viene perso (c). Infine, dopo altri 15 fotogrammi le due traiettorie stimate convergono in un unico punto intermedio (d).

6.2 Modello SMOG

Nel primo scenario proposto (Figura 6.6) il modello SMOG si dimostra in grado di seguire l'oggetto per tutta la durata della sequenza. Tuttavia, dato che la parziale occlusione incontrata non è sufficiente ad ottenere una misura SMOG inferiore alla soglia stabilita, il modello viene aggiornato comprendendo anche parte dell'oggetto estraneo. Questo rende l'individuazione della posizione dell'oggetto meno precisa nei fotogrammi in questione.

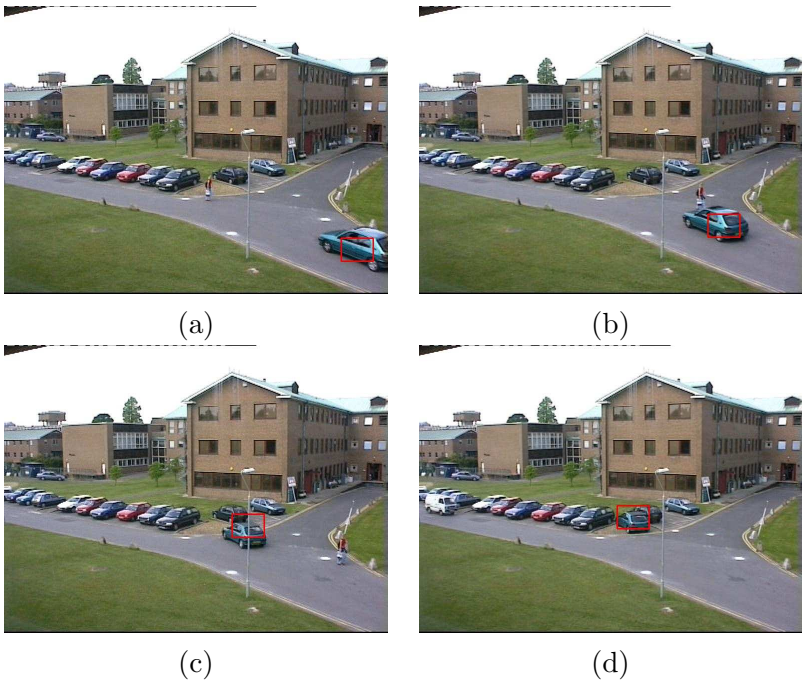


Figura 6.6: Inseguimento di un oggetto con modello SMOG. Nonostante un lieve spostamento dovuto ad una parziale occlusione (c) l'oggetto viene seguito fino alla fine della sequenza (d).

Nel secondo test (Figura 6.7) il modello SMOG si dimostra superiore all'istogramma colore, mostrandosi in grado di seguire l'oggetto senza mai perderne traccia.

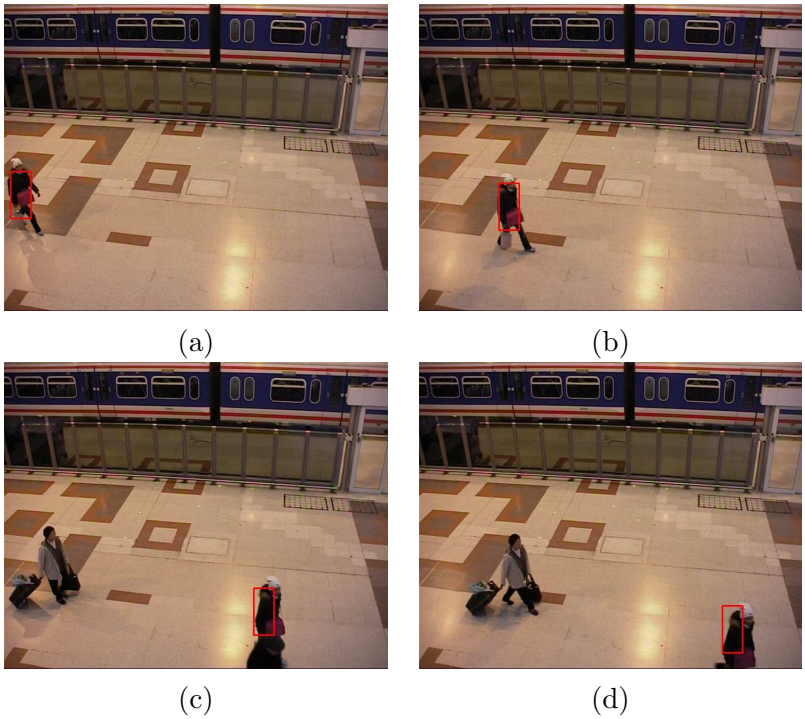


Figura 6.7: Il modello SMOG si dimostra in grado di gestire questo scenario, stimando correttamente la posizione dell'oggetto dal fotogramma iniziale (a) a quello finale (d) anche in presenza di una parziale occlusione con un oggetto di colore simile (c).

Il terzo caso ha richiesto numerosi tentativi per determinare i parametri corretti con cui inizializzare il modello SMOG. Procedendo empiricamente si è verificato che con α molto basso, quindi aggiornando lentamente il modello (è stato usato un valore pari a $0,002$), la posizione dell'oggetto viene tracciata correttamente anche in presenza di sovrapposizione con un altro oggetto (Figura 6.8).

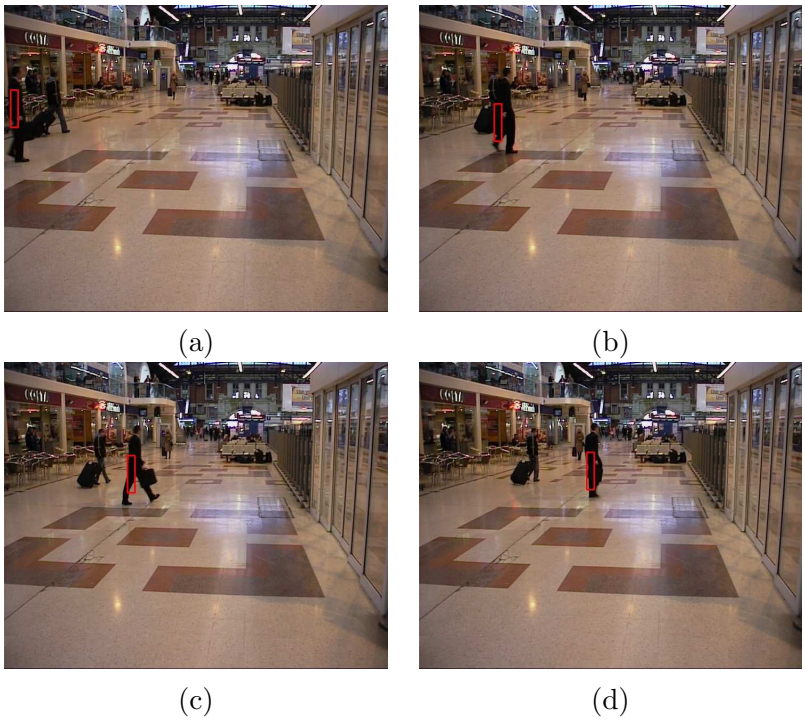


Figura 6.8: Tracking di un oggetto complicato dalla sovrapposizione di un altro oggetto simile. Il sistema non riesce a distinguere i due oggetti nel momento in cui sono sovrapposti (b). Tuttavia, grazie alle informazioni sulla posizione contenute nel modello SMOG, una volta che le traiettorie dei due oggetti si separano il tracker continua a seguire l'oggetto iniziale (c) fino alla fine della sequenza presa in considerazione (d).

L'analisi della scena complessa mostrata in Figura 6.9 ha richiesto di determinare procedendo per tentativi non solo il learning rate α del modello, ma anche la costante di tempo dT di propagazione delle particle. Utilizzando i valori $\alpha = 0,75$ e $dT = 0,04$ il tracker è stato in grado di seguire entrambi gli oggetti per tutta la durata della sequenza.

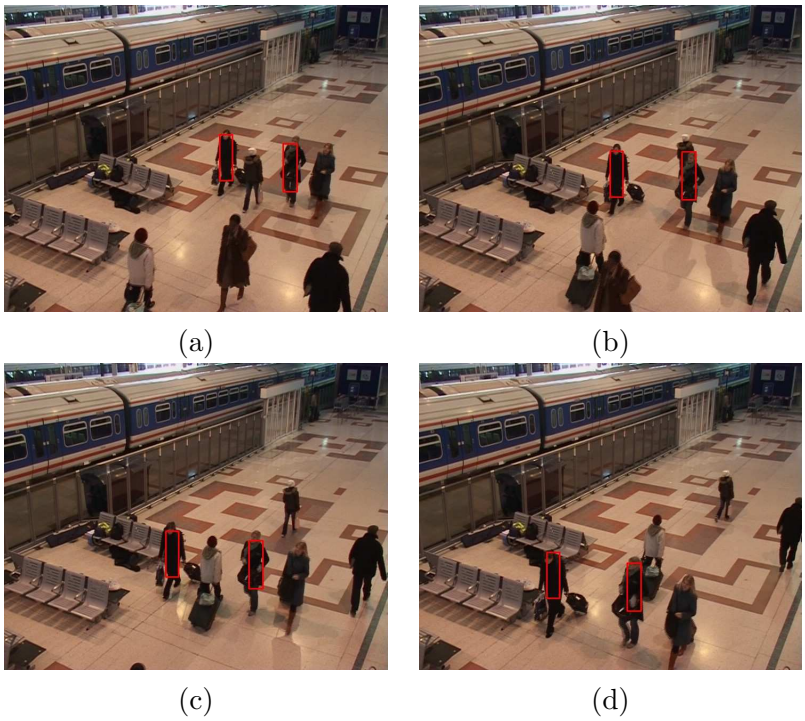


Figura 6.9: Multitracking con modello SMOG. Configurando opportunamente il modello, entrambi gli oggetti vengono seguiti correttamente, anche in presenza di sovrapposizioni (b).

6.3 Conclusioni e possibili sviluppi futuri

In questo lavoro è stato illustrato un metodo probabilistico per l'inseguimento di oggetti in movimento chiamato particle filter. Sono stati presentati due diversi modelli di immagine applicabili all'algoritmo di tracking, comparandone l'efficacia in differenti scenari. L'istogramma colore, approccio tradizionale di facile implementazione, risulta efficace in condizioni semplici ma si rivela inadeguato nel gestire scene complesse, in cui la semplice informazione sulla distribuzione del colore non è sufficiente. Il modello SMOG [23] contiene informazioni sia sul colore che sulla posizione dell'oggetto; questo permette una maggiore robustezza del sistema, come è evidente dall'analisi sperimentale mostrata in questo capitolo. Tuttavia, la sua applicazione in un particle filter generico rende necessaria la determinazione di diversi parametri di inizializzazione a seconda della scena da analizzare. Questo renderebbe l'algoritmo di tracking poco applicabile nella pratica, seppur funzionante in presenza dei parametri corretti. La soluzione a questo problema potrebbe essere lo sviluppo di un sistema di tracking apposito, basato esclusivamente sul modello SMOG. Wang et al. [24] propongono un algoritmo di tracking completo utilizzando il modello SMOG2, un'evoluzione di SMOG che contiene informazioni aggiuntive relative ai contorni dell'immagine. È inoltre proposto un nuovo metodo più efficiente per determinare i parametri del modello.

In conclusione, il sistema di tracking sviluppato si è mostrato funzionale ma non esente da problemi. Il modello SMOG, pur dimostrandosi capace di gestire situazioni in cui il semplice istogramma colore fallisce, ha evidenziato difficoltà di configurazione, forse dovute alla non perfetta integrazione con il tipo di particle filter realizzato e il relativo modello di moto. Ulteriori ricerche in questo campo potrebbero portare alla costruzione di un algoritmo di tracking robusto e affidabile.

Bibliografia

[1] Mubarak Shah Alper Yilmaz, Omar Javed. Object tracking: A survey. *ACM Computing Surveys (CSUR)*, 38 (4), 2006.

[2] Simon Godsill Arnaud Doucet and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. 1998.

[3] S. Avidan. Support vector tracking. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 184–191, 2001.

[4] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by probability distributions. *Bull. Calcutta Math. Soc.*, 35:99–109, 1943.

[5] S. Birchfield. Elliptical head tracking using intensity gradients and color histograms. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 232–237, 1998.

[6] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2 (2):121–167, 1998.

[7] M. Stephens C. Harris. A combined corner and edge detector. *4th Alvey Vision Conference*, pages 147–151, 1988.

[8] N Oudjane C Musso and F LeGland. Improving regularized particle filters. *Sequential Monte Carlo Methods in Practice*, 2001.

[9] W.E.L. Grimson Chris Stauffer. Adaptive background mixture models for real-time tracking. *Computer Vision and Pattern Recognition*, pages 246–252, 1999.

[10] E. Backer CJ. Veenman, M.J.T. Reinders. Resolving motion correspondence for densely moving points. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23 (1):54–72, 2001.

[11] I.J. Cox. A review of statistical data association techniques for motion correpsondence. *Int. J. Comp. Vision*, 10 (1):53–66, 1993.

[12] Judit Verestói D. Chetverikov. Feature point tracking for incomplete trajectories. *Computing, Devoted Issue on Digital Image Processing*, 62:321–338, 1999.

[13] W. Rucklidge D. Huttenlocher, J. Noh. Tracking nonrigid objects in complex scenes. *IEEE International Conference on Computer Vision*, pages 93–101, 1993.

[14] Kurt Konolige David Beymer. Real-time tracking of multiple people using continuous detection. 1999.

[15] Peter Meer Dorin Comanciu, Visvanathan Ramesh. Kernel-based object tracking. *IEEE Trans. Patt. Analy. Mach. Intell.*, 18 (2):138–150, 2003.

[16] Daniel Zaldivar Erik Cuevas and Raul Rojas. Particle filter in vision tracking. 2005.

[17] Raul Rojas Erik Cuevas, Daniel Zaldivar. Kalman filter for vision tracking. 2005.

[18] N. Thacker F. Aherne and P. Rockett. The bhattacharyya metric as an absolute similarity measure for frequency coded data. *Kybernetika*, 32(4):1–7, 1997.

[19] W R Gilks and C Berzuini. Following a moving target - monte carlo inference for dynamic bayesian models. *Journal of the Royal Statistical Society*, 63:127–146, 2001.

[20] Gary Bishop Greg Welch. An introduction to the kalman filter. *http://www.cs.unc.edu/~{welch,gb}*.

[21] Yang Liu Qingming Huang Guangyu Zhu, Dawei Liang and Wen Gao. Improving particle filter with support vector regression for efficient visual tracking. *IEEE International Conference on Image Processing*, 2, 2005.

[22] Rakesh Kumar Hai Tao, Harpreet S. Sawhney. Object tracking with bayesian estimation of dynamic layer representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24 (1):75–89, 2002.

[23] David Suter Hanzi Wang and Konrad Schindler. Effective appearance model and similarity measure for particle filtering and visual tracking. 2006.

[24] Konrad Schindler Chunhua Shen Hanzi Wang, David Suter. Adaptive object tracking based on an effective appearance filter. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29 (9):1661–1667, 2007.

[25] K. I. Hodges. Adaptive constraints for feature tracking. *Monthly Weather Review*, 127:1362–1373, 1998.

[26] R. Jain I.K. Sethi. Finding trajectories of feature points in a monocular image sequence. *IEEE Transactions on Pattern Analysis and Machine Intelligence archive*, 9 (1):56 – 73, 1987.

[27] M. Goldberg J. M. Beaulieu. Hierarchy in picture image segmentation-a step-wise optimization approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11 (2):150–163, 1989.

[28] A Blake J MacCormick. A probabilistic exclusion principle for tracking multiple objects. *Proc Int. Conf. Computer Vision*, pages 572–578, 1999.

[29] H. Kihl J. Urban, J. Buessler. Color histogram footprint technique for visual object tracking. *Proceedings of 2005 IEEE Conference on Control Applications*, 28 (31):761–766, 2005.

[30] Peter Cliffordy James Carpenter and Paul Fearnhead. Improved particle filter for non-linear problems. *IEE Proceedings on Radar and Sonar Navigation*, 1999.

[31] Thomas B. Schön Jeroen D. Hol and Fredrik Gustafsson. On resampling algorithms for particle filters.

[32] Carlo Tomasi Jianbo Shi. Good features to track. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[33] G. Medioni Jinman Kang, I. Cohen. Object reacquisition using invariant appearance model. *Proceedings of the 17th International Conference on Pattern Recognition*, 4:759 – 762, 2004.

[34] M. Sha K. Rangarajan. Establishing motion correspondence. *CVGIP: Image Understanding*, 24 (6):56–73, 1991.

[35] J. Aggarwal K. Sato. Temporal spatio-velocity transform and its application to tracking and interaction. *Computer Vision Image Understand*, 96:100–128, 2004.

[36] T. Kailath. The divergence and bhattacharyya distance measures in signal selection. *IEEE Transactions on Communication Technology*, 1967.

[37] Rudolph E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of Basic Engineering*, 82:34–45, 1960.

[38] Esther Koller-Meier Katja Nummiaro and Luc Van Gool. An adaptive color-based particle filter. 2002.

[39] Genshiro Kitagawa. Monte carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, Volume 5(1):1–25, 1996.

[40] N. M. Kwok and G. Dissanayake. Bearing-only slam in indoor environments using a modified particle filter. 2007.

[41] J.S. Liu and R. Chen. Sequential monte carlo methods for dynamic systems. *journal of the American Statistical Association*, 93, no. 443:1032–1044, 1998.

[42] D. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comp. Vision*, 60 (2):91–110, 2004.

[43] Gregory Randall Marcelo Bertalmio, Guillermo Sapiro. Morphing active contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22 (7):733–737, 2000.

[44] R. Mehrotra. Establishing motion-based feature point correspondence. *Pattern Recognition*, 31 (2):23–30, 1998.

[45] Andrew Blake Michael Isard. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29 (1):5–28, 1998.

[46] Allan D. Jepson Michael J. Black. Eigentracking: Robust matching and tracking of articulated objects using a view-based representation. 1998.

[47] Andrew W. Moore. Clustering with gaussian mixtures. *http://www.cs.cmu.edu/~awm/tutorials/gmm.html*.

[48] Andrew W. Moore. K-means and hierarchical clustering. *http://www.cs.cmu.edu/~awm/tutorials/kmeans.html*.

[49] P Del Moral. Nonlinear filtering: Interacting particle solution. *Markov Processes and Related Fields*, Volume 2(4):555–580.

[50] H. Moravec. Visual mapping by a robot rover. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 698–600, 1979.

[51] D J Salmond N J Gordon and A F M Smith. A novel approach to nonlinear and non-gaussian bayesian state estimation. *IEE Proceedings-F*, Volume 140:107–113, 1993.

[52] Haijing Wang Peihua Li. Object tracking with particle filter using color information. *Computer Vision/Computer Graphics Collaboration Techniques*, pages 534–541, 2007.

[53] Justus H. Piater. Mixture models and expectation-maximization.

[54] Dmitriy E. Skopin Rashad J. Rasras, Ibrahiem M. M. El Emary. Developing a new color model for image analysis and processing.

[55] Stan Sclaroff Rómer Rosales. 3d trajectory recovery for tracking multiple objects and trajectory guided recognition of actions. 1999.

[56] Remi Ronfard. Region-based strategies for active contour models. *International Journal of Computer Vision*, 13 (2):229–251, 1994.

[57] K Kanazawa D Koller S J Russell. Stochastic simulation algorithms for dynamic probabilistic networks. *Proceedings of the Eleventh Annual Conference on Uncertainty in AI (UAI '95)*, pages 346–351, 1995.

[58] Khurram Shafique and Mubarak Shah. A non-iterative greedy algorithm for multi-frame point correspondence. *IEEE International Conference on Computer Vision (ICCV)*, pages 110–115, 2003.

[59] Neil Gordon Simon Maskell. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 2001.

[60] R Chellappa T J Broida. Estimation of object motion parameters from noisy images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8 (1):90–99, 1986.

[61] Carlo Tomasi. Estimating gaussian mixture densities with em - a tutorial.

[62] I.K. Sethi V. Salari. Feature point correspondence in the presence of occlusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:87–91, 1990.

[63] Ian Wallace. A mean-shift tracker: implementations in c++ and hume. 2005.

[64] Nanning Zheng Xi Li. Adaptive target color model updating for visual tracking using particle filter. *IEEE international conference on systems, man & cybernetics*, 2004.

Appendice A

Listato

Vengono mostrate alcune parti rilevanti del codice sorgente dell'applicazione realizzata.

particle_filter.c

```
//-------------------------------------------------------------//
//                   user-defined types                        //
//-------------------------------------------------------------//
typedef struct  //particle struct
{
    int x;     //x coordinate
    int y;     //y coordinate
    float vx;  //velocity along x axis
    float vy;  //velocity along y axis
} particle;
//-------------------------------------------------------------//

//Particle filter with SMOG model (in RGB color space)
void filter_SMOG_RGB()
{
   int xdim=(xf-xi)/2;
   int ydim=(yf-yi)/2;
   int label[WIDTH][HEIGHT];
   scgaussian_RGB SMOG[K]; //SMOG mixture
   particle P[N],P1[N];    //particles
   double W[N];            //weights

   int distance = DIST_EUCLIDEAN;
```

```
   BW = yf-yi+1;  //bounding box dimensions
   BH = xf-xi+1;
   //initialize SMOG model
   em_RGB(SMOG,curr_frame,xi,xf,yi,yf,distance,K);
   //initialize samples
   init(P,N,INIT_RANDOM,xi,xf,yi,yf);

   while(count<nframes-1)
   {
     count++;
     //release the last frame and get a new one
     cvReleaseImage(&curr_frame);
     get_frame();
     //propagate particles
     propagate(P,P1,xdim,ydim);

     //evaluate likelihood
     evaluate_SMOG_RGB(P1,W,SMOG,xdim,ydim,label);

     //estimate object position
     estimate_position(P1,W,xdim,ydim,count);

     //resample particles
     resample(P1,P,W,RES_SYSTEMATIC);

     //show current frame with tracking rectangle
     cvNamedWindow( "Particle Filter", 1 );
     cvRectangle(curr_frame, cvPoint(xi,yi), cvPoint(xf,yf),
                 cvScalar(0x0,0x0,0xff,0), 2,8,0);
     cvShowImage("Particle Filter", curr_frame);
     while((cvWaitKey(10)>=0));

     //update SMOG model
     updateSMOG_RGB(SMOG,label);
   }
}
```

```
/* Particle propagation */
/* Takes particles in array P and propagates them.
   Results are stored in array P1 */
void propagate(particle *P, particle *P1, int xdim, int ydim)
{
  int i;
  double a;

  for(i=0; i<N; i++)
  {
    a = gsl_ran_gaussian(r,0.3); //random gaussian number

    //update position
    P1[i].x = P[i].x + (P[i].vx * dT) + (1/2 * a * dT*dT);
    if(P1[i].x < xdim) P1[i].x = xdim; //check if out of bounds
    if(P1[i].x+xdim >= WIDTH) P1[i].x = WIDTH-xdim;
    P1[i].y = P[i].y + (P[i].vy * dT) + (1/2 * a * dT*dT);
    if(P1[i].y < ydim) P1[i].y = ydim; //check if out of bounds
    if(P1[i].y+ydim >= HEIGHT) P1[i].y = HEIGHT-ydim;

    //update velocity
    P1[i].vx = P[i].vx + a*dT;
    P1[i].vy = P[i].vy + a*dT;
  }
}

/* Evaluate SMOG similarity */
void evaluate_SMOG_RGB(particle *P, double *W,
scgaussian_RGB *SMOG, int xdim, int ydim, int label[WIDTH][HEIGHT])
{
  int l,p;
  int ix,iy;
  float w,mean_x,mean_y; //parameters of current sample
  float sigma_x,sigma_y;
  float lambda;
  int startx,starty,endx,endy;
  int sumtot; //total number of pixels assigned to any cluster
  int sum[K+1];  //number of pixels of cluster l
  int sum_x[K+1], sum_y[K+1];  //sum of x,y values for cluster l
```

```
float sumq_x[K+1], sumq_y[K+1];  //(x-xm), (y-ym)
int r,g,b;
float d2,tmpdist; //squared Mahalanobis distance

//Label the pixels in all the predicted samples area
startx=endx=P[0].x;
starty=endy=P[0].y;
for(p=1; p<N; p++)
{
  if(P[p].x<startx) startx=P[p].x;
  else if(P[p].x>endx) endx=P[p].x;
  if(P[p].y<starty) starty=P[p].y;
  else if(P[p].y>endy) endy=P[p].y;
}

//compute labels
for(iy=starty-ydim; iy<endy+ydim; iy++)
{
   for(ix=startx-xdim; ix<endx+xdim; ix++)
   {
     label[iy][ix] = 0; //no mode has distance <= 2.5
     tmpdist = 999999; //big value
     //get r,g,b values
     r = ((uchar*)(curr_frame->imageData +
          curr_frame->widthStep*iy))[ix*3+2];
     g = ((uchar*)(curr_frame->imageData +
          curr_frame->widthStep*iy))[ix*3+1];
     b = ((uchar*)(curr_frame->imageData +
          curr_frame->widthStep*iy))[ix*3];

     for(l=0; l<K; l++) //for each mode of SMOG
     {
       //compute squared Mahalanobis distance
       d2 = (((r-SMOG[l].mean_r)*(r-SMOG[l].mean_r))/
              (SMOG[l].sigma_r*SMOG[l].sigma_r)) +
            (((g-SMOG[l].mean_g)*(g-SMOG[l].mean_g))/
              (SMOG[l].sigma_g*SMOG[l].sigma_g)) +
            (((b-SMOG[l].mean_b)*(b-SMOG[l].mean_b))/
              (SMOG[l].sigma_b*SMOG[l].sigma_b));
```

```
        //if |d|<=2.5
        if(d2 <= 6.25 && d2 < tmpdist)
        {
          tmpdist = d2;
          label[iy][ix] = l+1;
        }
      }
    }
}
//for each particle
for(p=0; p<N; p++)
{
  //initialization
  for(l=1; l<=K; l++)
  {
    sum[l] = 0;
    sum_x[l] = sum_y[l] = 0;
    sumq_x[l] = sumq_y[l] = 0;
  }
  sumtot = 0;

  for(iy=P[p].y-ydim; iy<P[p].y+ydim; iy++)
  {
    //for each pixel in current particle
    for(ix=P[p].x-xdim; ix<P[p].x+xdim; ix++)
    {
      if(label[iy][ix]!=0)
      {
        sum[(label[iy][ix])]++;
        sum_x[(label[iy][ix])] += ix;
        sum_y[(label[iy][ix])] += iy;
        sumq_x[(label[iy][ix])] += (ix*ix);
        sumq_y[(label[iy][ix])] += (iy*iy);
        sumtot++;
      }
    }
  }
  lambda = 0;
  for(l=1; l<=K; l++)  //for each mode of SMOG
  {
```

```
      if(sum[l]!=0)
      {
        w = (float)sum[l] / (float)sumtot;
        mean_x = (float)(sum_x[l]) / (float)sum[l];
        mean_y = (float)(sum_y[l]) / (float)sum[l];
        sigma_x = (float)(sumq_x[l]) / (float)(sum[l]) -
                  (mean_x*mean_x);
        sigma_y = (float)(sumq_y[l]) / (float)(sum[l]) -
                  (mean_y*mean_y);
        lambda += exp(-0.5 *((
                  (mean_x/WIDTH-SMOG[(l-1)].mean_x/WIDTH)*
                  (mean_x/WIDTH-SMOG[(l-1)].mean_x/WIDTH)*
                  (1/(sigma_x/WIDTH) + 1/
                  (SMOG[(l-1)].sigma_x/WIDTH)))+
                  ((mean_y/HEIGHT-SMOG[(l-1)].mean_y/HEIGHT)*
                  (mean_y/HEIGHT-SMOG[(l-1)].mean_y/HEIGHT)*
                  (1/(sigma_y/HEIGHT) + 1/
                  (SMOG[(l-1)].sigma_y/HEIGHT)))))*
                  min(w,SMOG[(l-1)].p);
     }
   }
     W[p] = lambda; //assign weight
  }
}

/* Estimate object position (weighted mean of samples) */
void estimate_position(particle *P, double *W,
                       int dx,int dy, int count)
{
  double x=0, y=0;
  double tot=0;
  int tmp_xi, tmp_xf, tmp_yi, tmp_yf;
  int i;

  //calculate total weight
  for(i=0; i<N; i++)
  {
    tot += W[i];
  }
```

```
  //weighted mean
  for(i=0; i<N; i++)
  {
    W[i]/=tot;
    x += P[i].x * W[i];
    y += P[i].y * W[i];
  }
  //update current object position
  tmp_xi = x-dx;
  tmp_xf = x+dx;
  tmp_yi = y-dy;
  tmp_yf = y+dy;

  //bound check
  if(tmp_xi>0) {xi=tmp_xi; xf=tmp_xf;}
  if(tmp_yi>0) {yi=tmp_yi; yf=tmp_yf;}

  //show current frame with tracking rectangle
  cvNamedWindow( "Particle Filter", 1 );
  cvRectangle(curr_frame, cvPoint(xi,yi),
              cvPoint(xf,yf),
              cvScalar(0x0,0x0,0xff,0),
              2,8,0);
  cvShowImage("Particle Filter", curr_frame);
  while((cvWaitKey(10)>=0));
}

/* Resampling algorithm*/
void resample(particle *P1, particle *P,
              double *W, int res_algorithm)
{
  double C[N];  //CDF
  double u[N];
  int i,j;
  particle P2[N];

  //copy P1 to P2
  for(i=0; i<N; i++)
  {
    P2[i].x = P1[i].x;
```

```
  P2[i].y = P1[i].y;
  P2[i].vx = P1[i].vx;
  P2[i].vy = P1[i].vy;
}

switch(res_algorithm)
{
  case RES_SYSTEMATIC: //systematic resampling
    C[0]=W[0]; //initialize the CDF
    for(i=1; i<N; i++)
    {
      C[i] = C[i-1] + W[i];  //construct CDF
    }
    i=0; //start at the bottom of the CDF
    u[0]=1;
    //draw a starting point (a random point between 0 and 1/N)
    while((u[0]=(double)(rand()%1000000)/1000000) > (1.00/N));
    for(j=0; j<N; j++)
    {
      u[j] = u[0] + (1.00/N)*j;  //move along the CDF
      while(u[j] > C[i])
      {
        ++i;
      }
      //assign sample
      P2[j].x = P1[i].x; // printf("%d %d ---",i,P1[i].x);
      P2[j].y = P1[i].y; // printf("%d\n",P1[i].y);
      //assign weight
      W[j] = 1.00/N;
    }
  break;

  default:
  break;
}
//copy P2 to P
for(i=0; i<N; i++)
{
  P[i].x = P2[i].x;
  P[i].y = P2[i].y;
```

```
    P[i].vx = P2[i].vx;
    P[i].vy = P2[i].vy;
  }
}
```

histogram.c

```
/* Color histogram */
/* computes the color histogram of an object
   (in the rectangle xi,xf,yi,yf) */
void color_histogram_RGB(float Hist[RBINS][GBINS][BBINS],
           IplImage *img, int xi, int xf, int yi, int yf)
{
  int i,j,k;
  int x,y;
  int r,g,b;

  int npixels = (xf-xi)*(yf-yi);

  //initialize histogram with zero values
  for(i=0;i<RBINS;i++)
    for(j=0;j<GBINS;j++)
      for(k=0;k<BBINS;k++)
        Hist[i][j][k]=0;

  //compute histogram values
  for(y=yi; y<yf; y++)
  {
    for(x=xi; x<xf; x++)
    {
      //get r,g,b values
      r = ((uchar*)(img->imageData + img->widthStep*y))[x*3+2];
      g = ((uchar*)(img->imageData + img->widthStep*y))[x*3+1];
      b = ((uchar*)(img->imageData + img->widthStep*y))[x*3];

      i = r >> 5; //division by 32 (255 total colors / 8 bins)
      j = g >> 5;
      k = b >> 5;
```

```
      Hist[i][j][k]++; //increment histogram
    }
  }
  // normalize histogram
  for(i=0;i<RBINS;i++)
    for(j=0;j<GBINS;j++)
      for(k=0;k<BBINS;k++)
        Hist[i][j][k] /= npixels;
}

/* Compute Bhattacharyya coefficient
   between histograms H1 and H2 */
double bhat_HSV(float H1[HBINS][SBINS][VBINS],
                float H2[HBINS][SBINS][VBINS])
{
  int i,j,k;
  double distance=0.0;

  for(i=0; i<HBINS; i++)
  {
    for(j=0; j<SBINS; j++)
    {
      for(k=0; k<VBINS; k++)
      {
        distance += sqrt(H1[i][j][k]*H2[i][j][k]);
      }
    }
  }
  return distance;
}
```

Appendice B

Il manuale utente

L'applicazione realizzata permette di effettuare il tracking di uno o più oggetti, applicando un particle filter a una sequenza di immagini ricevuta in ingresso e mostrandone il risultato a video.

B.1 Requisiti software

Il programma opera su piattaforma GNU/Linux, e richiede che siano presenti nel sistema, oltre al compilatore *gcc*, le seguenti librerie:

- GTK+ 2.x o superiori
  `http://www.gtk.org/`.
- Open Computer Vision Library (OpenCV) 1.0
  `http://www.intel.com/technology/computing/opencv/index.htm`.
- GNU Scientific Library (GSL)
  `http://www.gnu.org/software/gsl/`.

B.2 Installazione

1. Decomprimere il file contenente il codice sorgente:

   ```
   $ tar xzvf ParticleFilter.tar.gz
   ```

2. Spostarsi nella directory creata:

   ```
   $ cd ParticleFilter
   ```

3. Compilare il codice sorgente:

   ```
   $ make
   ```

B.3 Esecuzione del programma

Per poter utilizzare il programma è necessario innanzitutto disporre di una sequenza di immagini ordinate, corrispondenti a fotogrammi successivi di una sequenza video. I formati supportati sono: *bmp*, *dib*, *jpeg*, *ppm*, *sr*, *ras*, *tiff*. Se non si dispone di una sequenza di fotogrammi è possibile crearla a partire da un video, utilizzando ad esempio *mplayer* (reperibile al sito `http://www.mplayerhq.hu/`) con l'opzione *-vo jpeg*, *-vo png* o *-vo gif89a*:

```
mplayer -vo jpeg <nome file video>
```

Per eseguire il programma, digitare:

```
$ ./particle_filter [OPZIONI] <dir> <xi> <xf> <yi> <yf>
```

dove *dir* è il percorso della directory contenente i fotogrammi su cui eseguire il tracking, xi, xf, yi, yf rappresentano le coordinate del bounding box che racchiude l'oggetto da seguire (come mostrato in Figura B.1). In caso di tracking di più oggetti, è necessario inserire le coordinate di tutti i bounding box:

```
<x1i> <x1f> <y1i> <y1f> <x2i> <x2f> <y2i> <y2f> ...
```

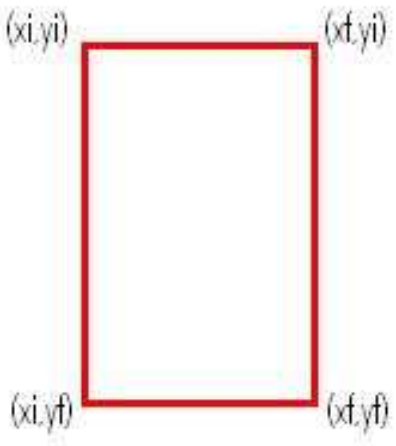


Figura B.1: Coordinate corrispondenti ai vertici del bounding box.

B.4 Opzioni

Il programma accetta le seguenti opzioni.

-h: visualizza una schermata di aiuto ed esce dal programma.

-m <hist | smog>: indica se utilizzare il modello istogramma colore o il modello SMOG. Se l'opzione non è specificata, il valore di default è *smog*.

-c <rgb | hsv | rgi>: seleziona lo spazio colore (RGB, HSV, rgI). Lo spazio colore rgI può essere usato solo in combinazione con "-m smog". Se questa opzione non è specificata, viene assunto il valore di default *rgb*.

-T <numero>: valore (compreso tra 0 e 1) della costante di tempo dT, usata per la propagazione delle particle. Il valore di default è 0.08.

-N <numero>: numero di particle, compreso tra 10 e 70000. Il valore di default è 1000.

-K <numero>: numero di classi per il modello SMOG, compreso tra 3 e 8. Questa opzione puùo essere usata solo in combinazione con "-m smog'. Il valore di default è 3.

-a <numero>: learning rate α per l'aggiornamento del modello SMOG, compreso tra 0 e 1. Questa opzione puùo essere usata solo in combinazione con "-m smog'. Il valore di default è 0.8.

-o <numero>: numero di oggetti da seguire. Il valore di default è 1 (Se il valore di questa opzione è maggiore di 1, dovranno essere specificate le coordinate dei bounding box di tutti gli oggetti).

Appendice C

Esempio di impiego

Viene mostrato un esempio di impiego del software realizzato. La sequenza video usata nel test, ubicata nella directory *v1/*, è parte del *PETS data set* (`http://www.cvg.rdg.ac.uk/slides/pets.html`). Il particle filter è configurato in modo da avere 5000 particle e operare nello spazio colore normalizzato rgI. Il modello utilizzato per rappresentare l'oggetto è di tipo SMOG con sei classi e $\alpha = 0.3$.
Il bounding box è inizializzato nella posizione identificata dalle coordinate $xi = 9,\ xf = 48,\ yi = 318,\ yf = 406$.

Il programma viene lanciato con il comando:

```
$ ./particle_filter -N 5000 -a 0.3 -c rgi v1/ 9 48 318 406
```

La sequenza considerata è composta da 158 frame. Il programma cerca di tenere traccia della posizione dell'oggetto lungo tutti i frame. In Figura C.1 è mostrato il comportamento del particle filter. La persona che cammina, individuata nel frame iniziale, viene seguita fino alla sua scomparsa dalla scena.

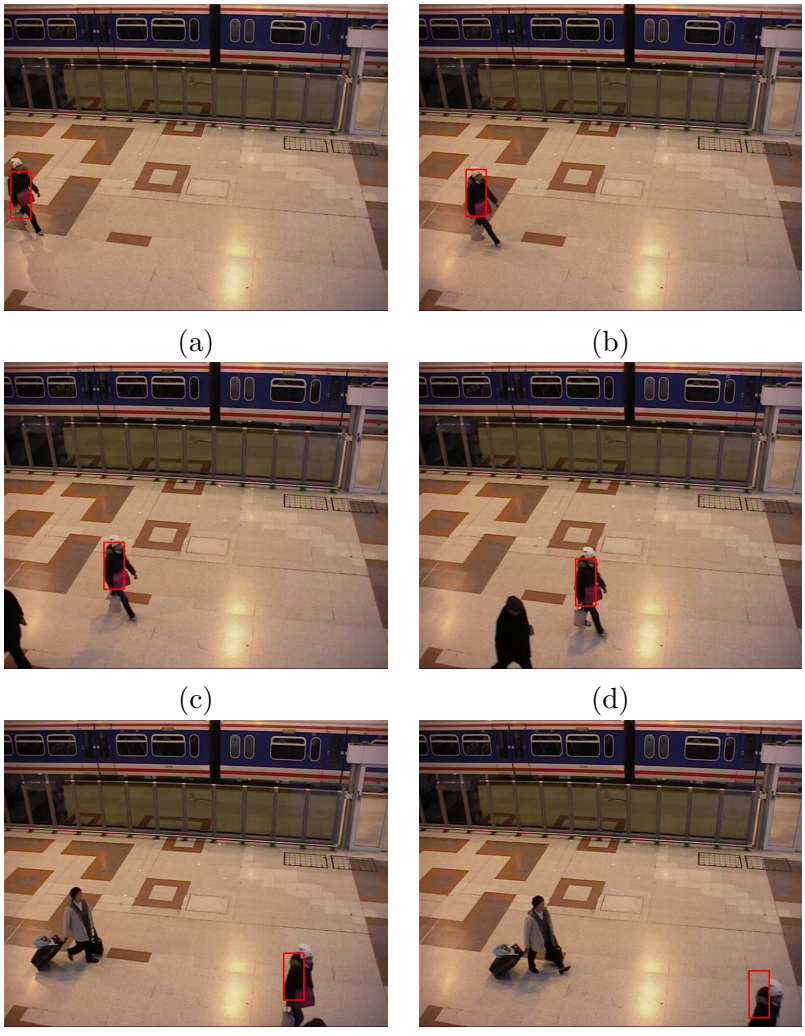


Figura C.1: Output del programma nei frame 1 (a), 25 (b), 50 (c), 75 (d), 125 (e), 145 (f).